Algoritmos de control en un prototipo de brazo robótico acuático modular con interfaz de realidad aumentada

Trabajo de grado para optar al título
de Ingeniero en Mecatrónica

Autor:
Maria Camila Moreno Vergara

Directores:
PhD. José Antonio Baca García
PhD. César Augusto Peña Cortés

Diciembre 2021

Algoritmos de control en un prototipo de brazo robótico acuático modular con interfaz de realidad aumentada

Autor
Maria Camila Moreno Vergara

Directores
PhD. José Antonio Baca García
Universidad de Texas A&M - Corpus Christi, USA

PhD. César Augusto Peña Cortés
Universidad de Pamplona - Norte de Santander, COL

Trabajo de grado para optar al título de Ingeniero en Mecatrónica

Universidad de Pamplona
Facultad de Ingenierías y Arquitectura
Pamplona, Colombia
Diciembre 2021

A mis padres. Gracias por todo

Agradecimientos

En el aspecto profesional, quiero dar las gracias a José Antonio Baca García y César Augusto Peña Cortés. Han contribuido a que este proyecto saliera adelante y han ofrecido un inestimable apoyo a lo largo del mismo. Los esfuerzos destinados para la dirección de este trabajo de grado fueron excepcionales.

Doy las gracias también a los jurados Yara Angeline Oviedo Durango y Cristhian Ivan Riaño Jaimes, siendo extraordinariamente diligentes y atentos durante la preparación de este documento. Sus comentarios fueron honestos, inteligentes y concisos.

Por supuesto, mi mayor deuda de gratitud la tengo con la Universidad de Pamplona. A lo largo de mi carrera universitaria, he encontrado preguntas e incógnitas que no he podido resolver. En cada caso hubo un docente dispuesto a proporcionar sus directrices para encontrar la respuesta. Muchas gracias.

Debo dar un agradecimiento especial a mis amigos, colegas y demás allegados. Son demasiados para enumerarlos aquí, pero es maravilloso terminar un proceso como este y llegar al otro lado con espíritus tan nobles.

John Carlos Moreno Castro, mi padre, y Elizabeth Vergara Fragozo, mi madre, fueron decisivos para llevar a cabo este proyecto. Hace mucho tiempo, en un pueblo lejano, mis padres pensaron que valía la pena destinar tiempo y esfuerzo en tener una educación superior. Por lo que, desde un punto de vista práctico, este libro no habría existido sin su apoyo.

Por último, quiero reconocer el esfuerzo de todos aquellos que realizaron aportes e investigaciones sobre los cuales se fundamenta este estudio, y sobre los cuales se inspirarán futuros avances aún no concebidos.

I think it is possible for ordinary people to choose to be extraordinary
"Creo que es posible para la gente normal elegir ser extraordinaria"

Elon Musk

Resumen

Este documento presenta los resultados de un proyecto de investigación sobre el control de un prototipo de brazo robótico acuático modular con interfaz de realidad aumentada. Conforme a lo anterior, se estructuran tres fases principales: En la Fase I, se expone el análisis cinemático, modelo dinámico y control cinemático del prototipo. En la Fase II, se presentan simulaciones que permitan corroborar el análisis realizado en la Fase I. Este apartado incluye interfaces de realidad aumentada para estimular la interacción humano-robot a través de teléfonos inteligentes, cámaras web y Microsoft HoloLens. Finalmente, en la Fase III, se establece comunicación entre las interfaces de realidad aumentada y las simulaciones mediante protocolos de transmisión de datos para generar referencias de posición o velocidad que son validadas de forma remota en el sistema físico localizado en la Universidad de Texas A&M-Corpus Christi. Este contempla un robot antropomórfico de 4 grados de libertad con módulos intercambiables para adaptar diversas herramientas en el elemento terminal o ser integrado en otros sistemas que compartan estándares de compatibilidad. Los movimientos generados por el prototipo en el sistema físico presentan un alto grado de correspondencia al expuesto en las simulaciones, evidenciando la relación entre los análisis y las interfaces de visualización.

Palabras clave: control, robótica modular, robótica acuática, simulación, realidad aumentada, protocolos de comunicación.

Abstract

This document presents the results of a research project on the control of a prototype of a modular aquatic robotic arm with an augmented reality interface. According to the above, three main phases are structured: In Phase I, the kinematic analysis, dynamic model and kinematic control of the prototype are exposed. In Phase II, simulations are presented to corroborate the analysis carried out in Phase I. This section includes augmented reality interfaces to stimulate human-robot interaction through smartphones, web cameras and Microsoft HoloLens. Finally, in Phase III, communication is established between the augmented reality interfaces and the simulations through data transmission protocols to generate position or speed references that are remotely validated in the physical system located at the University of Texas A&M-Corpus Christi. This contemplates an anthropomorphic robot with 4 degrees of freedom with interchangeable modules to adapt various tools in the terminal element or to be integrated into other systems that share compatibility standards. The movements generated by the prototype in the physical system present a high degree of correspondence to that shown in the simulations, evidencing the relationship between analysis and visualization interfaces.

Keywords: control, modular robotics, aquatic robotics, simulation, augmented reality, communication protocols.

Índice general

Índice de figuras

Índice de tablas

Índice de Códigos

1 Introducción

Existen dos áreas principales de investigación en la robótica: la manipulación y la locomoción. Los robots manipuladores son utilizados en operaciones de procesamiento, ensamblaje o empaque en conjunto con otras máquinas-herramientas para la formación de células de trabajo, su interés científico está ligado a una tarea en concreto. Por el contrario, los robots móviles presentan procesos de investigación mucho más activos debido a su capacidad de desplazarse por tierra, aire, bajo el agua o incluso en el espacio exterior. Dos claros ejemplos de aplicación son los robots *Spirit* y *Opportunity* lanzados por la National Aeronautics and Space Administration (NASA) con el objetivo de realizar análisis geológicos y atmosféricos en diferentes posiciones de Marte. La Figura. 1.1. y Figura. 1.2. presentan a un manipulador industrial y el robot *Spirit* del Programa de Exploración de Marte [1].

Figura 1.1: Robot industrial
Fuente: de Garibay Pascual

Otro ejemplo del interés dirigido al uso de robots móviles en entornos hostiles son los océanos. Investigaciones médicas en seres humanos plantean la posibilidad de descender alrededor de 100 metros, pero los altos niveles de presión o la falta de luminosidad representan serias limitaciones al momento de acceder a grandes profundidades. Como resultado, un gran número de oceanógrafos se encuentran desarrollando tecnología de avanzada para la monitorización de los océanos [1].

Figura 1.2: Robot Spirit del Programa de Exploración de Marte de la NASA
Fuente: de Garibay Pascual

En vista de las consideraciones mencionadas anteriormente, resulta indispensable diseñar sistemas dinámicos que generen soluciones a tareas repetitivas y hostiles en escenarios inciertos durante los próximos años. Algunas tendencias actuales y desarrollos conceptuales permiten anticipar la proyección a futuro de la robótica, destacando investigaciones en aplicaciones espaciales, submarinas, subterráneas, entre muchos otros [2], [3], [4], [5]. Sin embargo, es necesario considerar aspectos que permitan aumentar la movilidad, destreza y autonomía. En la actualidad, gran parte de robots son de base estática, implementados en distintas áreas productivas e industriales. No obstante, otro tipo de aplicaciones con fines particulares han causado una evolución en la concepción y morfología de los robots [6]. Tal es el caso de la robótica modular, un concepto que integra sistemas capaces de brindar solución a una gran diversidad de tareas combinando áreas relacionadas con la manipulación y la locomoción, se centra en problemas de escalabilidad, comunicación, simplificación de hardware y claridad de lógica. Un diseño modular es altamente personalizable, autosuficiente, reemplazable y reparable en contraste a las configuraciones fijas o diseños especializados.

Este documento está orientado a la implementación de algoritmos de control para aplicaciones de manipulación y desplazamiento en un prototipo de robot antropomórfico fundamentado en el concepto de modularidad. Una sección interesante es el apartado dirigido al uso de sistemas de realidad aumentada que aminoran inconvenientes relacionados con la falta de práctica, interés y demanda de las necesidades individuales de los estudiantes. Las principales razones que justifican sus altos estándares de efectividad en el ámbito formativo se basan en el estímulo de los procesos de comprensión, percepción, retención y atención prestada (inspirado en la metodología del *Learning by doing*) al promover la interacción en escenarios donde se combinan objetos reales y virtuales que permiten al usuario llevar a cabo la resolución de problemas mediante la indagación, el descubrimiento, el estímulo de sus habilidades motrices y habilidades de razonamiento espacial que los humanos desarrollan instintivamente [7].

1.1 Justificación

Aplicaciones con foco en la exploración espacial, misiones de búsqueda y rescate, exploración submarina o transporte de objetos requieren de sistemas robóticos que combinen características de robustez en entornos inestables para solventar posibles fallas de hardware, versatilidad para funcionar correctamente en diferentes circunstancias y navegación autónoma todo terreno en ambientes complejos no estructurados [8]. En adición a esto, las limitaciones de rendimiento ocasionadas por el tamaño y el entorno operativo de la morfología fija de los robots convencionales han generado el estudio de una gran variedad de sistemas bajo el término robótica modular [9].

Conceptualmente, una configuración modular comprende una serie de módulos independientes, como células animales que le permiten formar una amplia variedad de configuraciones [10]. Este enfoque encuentra sus raíces en estudios relacionados con la autoorganización y, autoensamble observables en especies sociales de insectos. Una entidad robótica de este tipo tiene la capacidad de realizar misiones tanto de desplazamiento como de manipulación de objetos.

Cabe resaltar que las principales razones que atienden al planteamiento del problema del presente estudio, han sido descritas de una forma generalizada en el apartado anterior. Sin embargo, es debido mencionar que la robótica modular a menudo es relegada a un segundo plano por limitaciones relacionadas con el diseño de hardware, aplicación, planificación y control [11]. Esto representa un reto y punto de partida para las futuras aplicaciones de distintas técnicas de control en sistemas robóticos modulares y el desarrollo de sistemas innovadores que difieran en características a los convencionales. Luego de realizar una revisión bibliográfica de la literatura relacionada, no se registraron implementaciones de algoritmos de control en brazos robóticos acuáticos con características modulares empleando interfaces de visualización de realidad aumentada. Por lo tanto, este trabajo pretende elaborar un método para generar referencias para los actuadores de un prototipo de brazo robótico acuático modular a través de interfaces de visualización de realidad aumentada y simulaciones en Matlab incorporando el uso de protocolos de comunicación. Además, se han desarrollado unos módulos de aprendizaje mediante videotutoriales para generar un gusto por las prácticas y la adquisición de los conocimientos descritos durante el desarrollo del proyecto.

1.2 Objetivos

1.2.1 Objetivo general

- Controlar un prototipo de brazo robótico acuático modular con interfaz de realidad aumentada.

1.2.2 Objetivos específicos

- Realizar el análisis cinemático, modelo dinámico y control cinemático de un robot antropomórfico de cuatro grados de libertad.
- Diseñar una aplicación de realidad aumentada que permita simular, visualizar e interactuar con un modelo 3D del prototipo en teléfonos inteligentes y cámaras web.
- Diseñar una aplicación de realidad aumentada que permita visualizar modelos 3D e información relevante del prototipo en dispositivos Microsoft HoloLens.
- Establecer comunicación entre la interfaz de realidad aumentada y las simulaciones a través de protocolos de transmisión de datos.
- Validar un grupo de referencias especificas generadas por los algoritmos de control en el sistema físico.
- Elaborar módulos de aprendizaje que permitan la asimilación y puesta en marcha de las técnicas presentadas por medio de videotutoriales.

2 Marco teórico

La robótica es una ciencia de carácter interdisciplinario que describe un conjunto de disciplinas básicas y tecnologías relacionadas con la teoría de control, la mecánica, la electrónica, el álgebra, la informática, entre otras [6]. Sus campos de aplicación comprenden desde la asistencia a personas con discapacidades físicas hasta sofisticados desarrollos tecnológicos diseñados para la exploración espacial [1].

Antes de considerar los temas relacionados con dos de los subgrupos específicos de robots clasificados a partir de su área de aplicación, es necesario presentar algunas generalidades sobre su morfología y algunas herramientas matemáticas para la localización espacial. En la próxima sección, serán examinados diferentes tipos de articulaciones entre dos eslabones consecutivos, sistemas de transmisión y reductores, actuadores y sistemas sensoriales. Concluye considerando una breve revisión de los elementos terminales situados en el extremo del robot para su interacción con el mundo exterior.

2.1 Morfología de un robot

El significado del término "robot industrial" se ha ampliado en los últimos años, varias de sus definiciones coinciden en dispositivos con capacidad de manipulación dotados de uno o más brazos mecánicos controlados a través de un ordenador. Estos poseen tecnologías transdisciplinarias que integran ciencias afines a las matemáticas, física, electrónica, mecánica, computación y automatización [12]. En un sentido más amplio, un robot está formado por su estructura mecánica, sistemas de transmisión, sistemas de accionamiento, sistemas sensoriales, sistemas de control y elementos terminales [6].

2.1.1 Estructura mecánica

Mecánicamente, un brazo robot se compone por una secuencia de eslabones unidos a través de articulaciones, permitiendo el movimiento relativo entre cada dos eslabones inmediatos. Dicho movimiento puede ser de desplazamiento, giro o de una combinación de ambos. Un parámetro importante en cada articulación son los Grados De Libertad (GDL), este indica los movimientos independientes que puede hacer una articulación respecto a la anterior [6]. En esa medida, son posibles seis tipos de articulaciones que se presentan en la Figura.2.1.

Con frecuencia se han sugerido términos como cuerpo, brazo, codo y muñeca para describir las partes que conforman a un robot. Esto debido a que la constitución física de una gran parte de los robots industriales se asemeja en cierta medida a la anatomía del brazo humano.

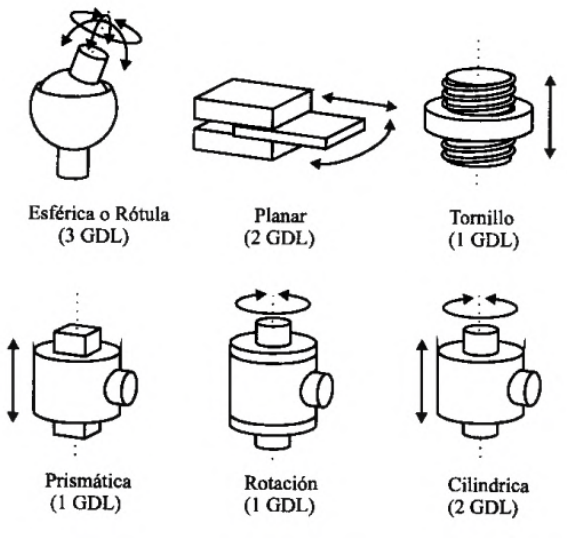

Figura 2.1: Tipos de articulaciones
Fuente: Barrientos et al.

Configuraciones de uso más frecuentes en la industria

Los robots manipuladores están disponibles en distintos tamaños, formas y configuraciones físicas. Dependiendo de la disposición o combinación de sus articulaciones, es posible obtener ciertas propiedades en cuanto a espacio de trabajo y accesibilidad. En este apartado, se examinan algunas estructuras básicas clasificadas a partir de las primeras tres articulaciones de un robot.

- **Robot cartesiano.** Esta topología es una de las más extendidas por su facilidad y bajos costes de instalación. Su esquema de movimiento se basa en un sistema de tres ejes de control lineales ortogonales entre sí. Son ampliamente utilizados en procesos de inserción, extracción, fresado y embalaje.
- **Robot cilíndrico.** Presentan una estructura mecánica compleja formada por una articulación rotacional situada en torno a la base y dos articulaciones prismáticas para un movimiento angular. Algunas aplicaciones de los robots cilíndricos en la industria se destinan al ensamblaje, soldadura por puntos y gestión de máquinas-herramientas.
- **Robot esférico o polar.** Es una configuración de robot industrial caracterizado por tener las dos primeras articulaciones de tipo rotacional y una tercera articulación tipo prismática, obedece a coordenadas esféricas o polares. Al igual que el robot cilíndrico, es empleado en tareas de soldadura por puntos, gestión de herramientas y vaciado de metales.
- **Robot SCARA.** Conocidos por sus altos niveles de velocidad, repetitividad y capacidad de carga, los robots SCARA, siglas en ingles de Selective Compliant Assembly Robot Arm, son máquinas programables de 4 grados de libertad con posicionamiento horizontal. Entre sus aplicaciones habituales se encuentra el montaje de componentes, soldadura, guiado y Pick & Place.

- **Robot angular o antropomórfico.** Recibe su nombre dadas las similitudes entre su estructura mecánica y el brazo humano. Disponen de una gran destreza en su espacio de trabajo al contar con tres articulaciones rotacionales. Funcionan como elementos centrales en las celdas de paletización [6].

2.1.2 Transmisiones y reductores

Los sistemas de transmisión son mecanismos encargados de transmitir el movimiento entre dos o más elementos de una máquina. Por otra parte, los reductores adaptan factores de par y velocidad de la salida del actuador según las órdenes dadas por la unidad de control.

- **Transmisiones.** La principal causa que justifica el uso de transmisiones de movimiento en las articulaciones reside en las elevadas aceleraciones en el extremo del robot y la distancia de las masas del actuador. En consecuencia, se procura ubicar los actuadores pesados lo más cerca posible a la base. Las transmisiones también permiten convertir movimiento circular en lineal o viceversa. En la Tabla.2.1. se presentan los sistemas de transmisión para robots.

Entrada-Salida	**Denominación**	**Ventajas**	**Inconvenientes**
Circular-Circular	Engranaje	Pares altos	Holguras
	Correa dentada	Distancia grande	-
	Cadena	Distancia grande	Ruido
	Paralelogramo	-	Giro limitado
	Cable	-	Deformabilidad
Circular-Lineal	Tornillo sin fin	Poca holgura	Rozamiento
	Cremallera	Holgura media	Rozamiento
Lineal-Circular	Paralelogramo articulado	-	Control difícil
	Cremallera	Holgura media	Rozamiento

Tabla 2.1: Sistemas de transmisión para robots

Fuente: Autor.

- **Reductores.** Las altas prestaciones en cuanto a precisión y velocidad de posicionamiento de los robots industriales requieren de diseños especializados con las siguientes prestaciones:

  1. Compacto y ligero.
  2. Alto rendimiento.
  3. Alta velocidad de entrada.
  4. Alta rigidez de torsión.
  5. Bajo momento de inercia.
  6. Bajo juego angular.
  7. Bajo rozamiento [6].

2.1.3 Actuadores

Generan el movimiento de los elementos que componen a un robot según las instrucciones proporcionadas por la unidad de control. Los principales actuadores se clasifican conforme a la energía que utilizan en neumáticos, hidráulicos y eléctricos. Cada uno con características de potencia, controlabilidad, peso, volumen, precisión, velocidad, mantenimiento y coste.

- **Actuadores neumáticos.** La fuente de energía para estos dispositivos procede de un compresor de aire a presión entre 5 a 10 bar. Abarcan desde los cilindros lineales hasta motores neumáticos y suelen emplearse en robots pequeños destinados a tareas de recolección o posicionamiento de objetos. Estos sistemas se caracterizan por su facilidad y bajo precio de instalación.
- **Actuadores hidráulicos.** Este tipo de actuadores emplean aceites minerales a presión entre un rango habitual de 50 y 100 bar, esta cifra en ocasiones puede superar el valor de 300 bar dependiendo de la aplicación. Las elevadas presiones de trabajo permiten desarrollar elevadas fuerzas y pares, destacando características de auto lubricación y robustez. A su vez, presentan estabilidad ante cargas estáticas y una mayor precisión en comparación a los actuadores neumáticos.
- **Actuadores eléctricos.** Permiten la generación de movimiento a partir de la aplicación de energía eléctrica. Las características de control, sencillez y precisión que ofrecen los posicionan como uno de los actuadores más usados en la industria, siendo ideales para tareas de alta precisión. Dentro de esta categoría pueden distinguirse los motores de corriente continua, motores de corriente alterna y motores paso a paso [6].

2.1.4 Sistemas sensoriales

Los sensores son dispositivos con propiedades sensibles a una magnitud de un parámetro físico. Proporcionan al robot conocimiento tanto de su estado como del estado de su entorno, se dividen en sensores internos y externos. Los sensores internos suministran información relacionada al estado del robot. Por otro lado, los sensores externos captan estímulos relacionados al estado de su entorno. En ambas categorías se presentan sensores de presencia, de posición o de velocidad [6].

2.1.5 Elementos terminales

Los elementos terminales, también conocidos como efectores finales son dispositivos ubicados en el extremo del robot para la interacción directa entre un sistema y su entorno. Estos son específicamente diseñados para cada tipo de trabajo y se clasifican atendiendo a un elemento de sujeción o una herramienta. Los elementos del primer grupo se utilizan para agarrar o sostener objetos, mientras que los elementos del segundo grupo permiten dotar al robot de múltiples herramientas durante la realización de su tarea [6].

2.2 Herramientas matemáticas

El movimiento espacial del extremo del robot implica conocer su posición y orientación con respecto a la base. Se evidencia en tales casos la necesidad de prescindir de un conjunto

de herramientas matemáticas que posibiliten determinar relaciones espaciales entre distintos objetos y el manipulador. Es necesario resaltar desde el principio, que las herramientas abordas en esta sección son de tratamiento general y no exclusivas del campo de la robótica [6].

2.2.1 Representación de la posición

Existen varios métodos generales para especificar la ubicación de un punto, algunos corresponden a coordenadas polares para sistemas de dos dimensiones, y las cilíndricas y esféricas para sistemas de tres dimensiones. Aun así, una de las formas más intuitivas para determinar estos parámetros son las coordenadas cartesianas.

Coordenadas cartesianas

El sistema de coordenadas cartesianas comprende un tipo de coordenadas ortogonales usadas en espacios euclídeos. Es decir, si se trabaja en un plano con un sistema coordenado OXY, un punto a expresado por la componente en X y la componente en Y, tendrá asociado un vector p(x,y). Este vector va desde el origen O del sistema hasta el punto a. En caso de trabajar en el espacio, el vector estará definido con respecto a sistema de referencia OXYZ [6].

2.2.2 Representación de la orientación

Aunque en el apartado anterior se presentó un método para determinar los datos sobre la posición de un punto, es de aclararse que para un robot no es suficiente tener a disposición estos parámetros, si no que en general, se debe indicar su orientación. Por consiguiente, se hará un breve repaso por dos de los métodos más utilizados para la representación de la orientación de un punto en el espacio.

Matrices de rotación

La comodidad que proporciona el uso del álgebra matricial para la descripción de orientaciones la convierte en uno de los métodos más extendidos en el campo de la robótica. Suponga que se tiene en el espacio dos sistemas de referencia OXYZ y OUVW coincidentes en el origen. El primer sistema es fijo y el segundo sistema es solidario al objeto el cual se pretende definir su orientación [6]. Es posible obtener una equivalencia de un vector P en el espacio referido a cualquiera de los dos sistemas de la siguiente manera:

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \boldsymbol{R} \begin{bmatrix} p_u \\ p_v \\ p_w \end{bmatrix} \tag{2.1}$$

Donde:

$$R = \begin{bmatrix} i_x i_u & i_x j_v & i_x k_w \\ j_y i_u & j_y j_v & j_y k_w \\ k_z i_u & k_z j_v & k_z k_w \end{bmatrix} \tag{2.2}$$

A modo de resumen, se han establecido las orientaciones del sistema OUVW.

- El eje OU coincidente con el eje OX:

$$R(x,\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & -sen\alpha \\ 0 & sen\alpha & cos\alpha \end{bmatrix} \tag{2.3}$$

- El eje OV coincidente con el eje OY:

$$R(y,\phi) = \begin{bmatrix} cos\phi & 0 & sen\phi \\ 0 & 1 & 0 \\ -sen\phi & 0 & cos\phi \end{bmatrix} \tag{2.4}$$

- El eje OW coincidente con el eje OZ:

$$R(z,\theta) = \begin{bmatrix} cos\theta & -sen\theta & 0 \\ sen\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

Ángulos de Euler

Los llamados ángulos de Euler representan otro método para la definición de la orientación. A diferencia de las matrices de rotación, únicamente se necesitan tres componentes para su descripción. Este término propone que todo sistema OUVW solidario al objeto cuya orientación se desee determinar, puede ser definido respecto al sistema OXYZ mediante tres ángulos: ϕ, θ y ψ. Algunas de las diversas posibilidades más usadas de los ángulos de Euler, son las siguientes: Ángulos de Euler ZXZ y Ángulos de Euler ZYZ. Nótese que el orden de los giros sucesivos alrededor de un sistema no es conmutativo y, por ende, es necesario conocer los valores de los ángulos y los ejes sobre los cuales se realizan los giros [6].

2.2.3 Matrices de transformación homogénea

Este concepto describe a una matriz de dimensión 4x4 que expresa la posición y orientación de un sistema de coordenadas con respecto a otro. Su estructura se haya compuesta por cuatro submatrices correspondientes a la rotación, traslación, perspectiva y escalado.

$$T = \begin{bmatrix} Rotación & Translación \\ Perspectiva & Escalado \end{bmatrix} = \begin{bmatrix} R_{3\times3} & p_{3\times1} \\ f_{1\times3} & w_{1\times1} \end{bmatrix} \tag{2.6}$$

Donde:

- $R_{3\times3}$: Matriz de rotación.
- $p_{3\times1}$: Vector de traslación.
- $f_{1\times3}$: Vector de perspectiva

- $w_{1\times1}$: Factor de escala.

Para la mayor parte de las aplicaciones de la robótica solo se interesará conocer los componentes de rotación y traslación, el vector correspondiente a la perspectiva se considerará nulo y el factor de escala será igual a la unidad [6].

2.3 Robótica modular

El concepto de modularidad adquiere presencia en el campo cognitivo de las ciencias a principios de la década de 1980. El tema abordado por Fodor en la obra "*The Modularity of Mind*" plantea una serie de propiedades representativas de los sistemas modulares. Por otra parte, Coltheart propone la posibilidad de haber cierta diversidad en los tipos de módulos cognitivos. Finalmente, el autor Ned Block argumenta que los módulos planteados por Fodor podrían descomponerse en módulos más finos.

A grandes rasgos, un sistema robótico modular se caracteriza en la medida en que cada una de sus unidades opera de acuerdo a sus principios, los cuales han sido determinados de manera intrínseca. Sin embargo, hasta hace unos años, la robótica modular ha avanzado desde el desarrollo de sistemas de pruebas de concepto, hasta sofisticados sistemas para elaborar implementaciones físicas y simulaciones. Factores como la versatilidad, robustez y el bajo costo en el desarrollo de estos sistemas abre un amplio abanico de posibilidades en un gran número de aplicaciones [13].

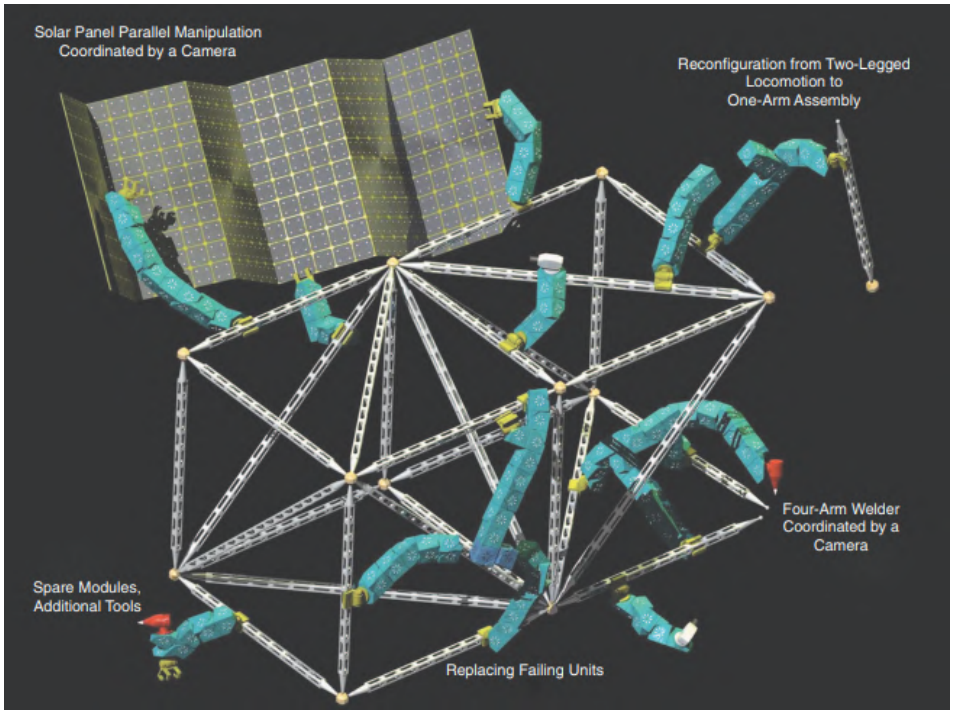

Figura 2.2: Interpretación artística de una aplicación espacial basada en robótica modular
Fuente: Yim et al.

Los sistemas robóticos modulares representan máquinas cinemáticas autónomas altamente versátiles y extensibles. Tienen la capacidad de adaptarse a nuevas circunstancias, realizar nuevas tareas o recuperarse de daños cambiando y reorganizando la conectividad de sus partes [11]. Pueden ser clasificados en dependencia de la disposición geométrica de sus componentes, como se observa en la Tabla.2.2.

Arquitectura	**Características**
Celosía	- Las unidades están dispuestas y conectadas en patrones tridimensionales regulares, como cuadrículas cúbicas o hexagonales. - El control y movimiento se pueden ejecutar en paralelo.
Cadena/Árbol	- Las unidades conectadas presentan una topología de cuerda o árbol. - Tienen la capacidad de alcanzar cualquier punto u orientación en el espacio. - A nivel computacional, son más difíciles de representar, analizar y controlar.
Móvil	- Disponen de unidades que utilizan el entorno para maniobrar y reorganizarse en forma de cadenas, retículos complejos o en series de robots pequeños. - Ejecutan movimientos coordinados.

Tabla 2.2: Tipos de arquitecturas entre robots modulares

Fuente: Yim et al.

La esencia principal detrás del enfoque de la robótica modular radica en dividir un sistema complejo en diversas unidades funcionales con alta portabilidad y simplicidad de mantenimiento [13]. De hecho, es posible clasificar los sistemas robóticos modulares según la forma en como sus unidades se reconfiguran [11]. La Tabla.2.3 presenta los tipos de reconfiguraciones entre los sistemas robóticos modulares.

Reconfiguración	**Características**
Determinística	- Se conoce en todo momento la ubicación exacta de cada unidad. - Es posible descubrir y calcular el tiempo de ejecución. - Permiten garantizar los tiempos de reconfiguración del sistema. - El control por realimentación es necesario para la manipulación precisa.
Estocástica	- La ubicación exacta de cada unidad solo es conocida cuando está conectada a la estructura principal. - Solo se pueden garantizar los tiempos de reconfiguración mediante estadísticas. - El medio ambiente proporciona gran parte de la energía al sistema.

Tabla 2.3: Tipos de reconfiguraciones entre sistemas robóticos modulares

Fuente: Yim et al.

Generalmente, su estructura se compone por bloques de construcción que hacen parte de un repertorio considerablemente pequeño con múltiples unidades de acoplamiento de interfaces; usualmente formado por una unidad principal y algunas unidades adicionales como pinzas, ruedas, carga útil, unidad de almacenamiento y generación de energía. También poseen interfaces de acoplamiento para la transferencia uniforme de fuerzas, momentos mecánicos, energía eléctrica y comunicación en todo el robot [11].

2.3.1 Áreas de aplicación

A continuación, se exponen dos de los campos de aplicación de la robótica modular más demandados en la actualidad. Estos incluyen tendencias, proyecciones a futuro y áreas de investigación.

- **Exploración espacial.** El desarrollo tecnológico espacial a largo plazo comprende misiones que precisan de una ecología robótica autosuficiente con la capacidad de manejar situaciones imprevistas y la habilidad de proporcionar una reparación propia. En términos de magnitudes, las misiones espaciales demandan una gran cantidad de volumen, ocasionando como resultado diversas limitaciones de masa. El envío de sistemas robóticos modulares contempla un escenario en donde se posibilita llevar a cabo una amplia variedad de tareas, permitiendo reducir las limitaciones relacionadas con la masa y el volumen comúnmente presentes en las configuraciones fijas.

- **Contendor de cosas.** Se denomina "*Bucket of Stuff*" a una visión a largo plazo en el cual los consumidores disponen de un contenedor de módulos auto configurables con el objetivo de realizar tareas cotidianas. Algunas aplicaciones provistas sirven como fuente de inspiración para el desarrollo de estos sistemas. Así como también el propósito de originar sistemas biológicos adaptativos con módulos de nivel inferior [11].

2.3.2 Ejemplos y casos de aplicación

Las indiscutibles ventajas que ofrecen los diseños modulares representan un gran panorama de oportunidades para el desarrollo de nuevos sistemas en escenarios que requieran de prototipados rápidos para la demostración y validación de nuevas ideas. Es por esto que prestigiosas instituciones han generado diversos aportes en el campo de la robótica modular.

- **Miche.** Es un sistema robótico modular desarrollado en el Instituto Tecnológico de Massachusetts (MIT). Comprende un conjunto de 28 módulos implementados como robots autónomos que parten de una estructura inicial a transformarse en una estructura de objetivos. Emplean un sistema de auto desmontaje que elimina módulos innecesarios haciendo uso de la gravedad, esto genera que el proceso de desarmar el sistema solo requiera sacudir los módulos no utilizados. Paralelamente, sirve como banco de pruebas para el desarrollo de algoritmos no distribuidos [14]. Cada módulo del sistema es completamente autónomo e incluso pueden funcionar durante varias horas por sus propios medios; un cubo mide 1,77 pulgadas de cada lado y pesa 4.5 oz, como se observa en la Figura.2.3.

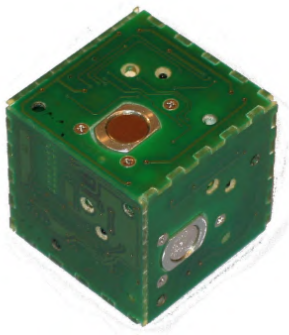

Figura 2.3: Módulo individual del sistema Miche
Fuente: Gilpin et al.

- **PolyBot.** Es considerado uno de los robots reconfigurables más versátiles de todos. Uno de los experimentos más interesantes fue presentado como portada de la revista IEEE Spectrum, se trata de un sistema compuesto por 20 módulos generación G1v4 organizados para representar dos piernas y una cintura con el objetivo de apoyarse en los pedales de un triciclo. El movimiento coordinado permitía el avance del triciclo con el robot encima. El resultado de este experimento se muestra en la Figura.2.4

Figura 2.4: Experimento del triciclo realizado por módulos generación G1v4
Fuente: Gonzalez

PolyBot fue desarrollado por el Centro de Investigación de Palo Alto (PARC), anteriormente conocido como Xerox PARC. Utiliza un control centralizado de bucle abierto

mediante tablas de control que le permiten replicar movimientos tipo serpiente, rueda y araña. Del mismo modo, cuenta con sensores de fuerza para detectar la presión ejercida sobre la superficie en la que avanza, obteniendo excelentes resultados en aspectos de adaptación de terreno y capacidad de superar diversos obstáculos. Actualmente existen tres generaciones de módulos: G1, G2 y G3 [15].

2.3.3 Desafíos y oportunidades a futuro

Si bien las ventajas de los sistemas robóticos modulares son incontables, los múltiples desafíos a los que se enfrentan los investigadores la convierten en una disciplina con un interesante nivel conceptual a menudo relegada a un segundo plano por limitaciones relacionadas con el diseño de hardware, aplicación, planificación y control. Estudios han demostrado que hasta hace unos años solo se han registrado sistemas modulares con un máximo de 50 unidades, valor estancado durante casi una década. A pesar de esto, numerosas instituciones y centros de investigación destinan esfuerzos conjuntos para generar soluciones a factores de tamaño, robustez y rendimiento con el objetivo de controlar millones de ejemplares en un mismo sistema. De igual manera, es necesario incidir en los grandes desafíos que darían lugar una generación de módulos superiores:

- Grandes sistemas.
- Sistemas de auto-reparación.
- Sistemas auto-sostenibles.
- Auto replicación y auto-extensión.
- Conciliación con la termodinámica [11].

2.4 Robótica acuática

La robótica acuática engloba un conjunto de robots móviles que usan el agua como medio de transporte [1]. Los robots acuáticos han sido diseñados para realizar misiones de inspección, recolección de muestras y servir como equipos de apoyo en misiones de rescate a personas en ambientes de difícil acceso [16]. Estos dispositivos han revolucionado la forma en cómo se llevan a cabo los métodos de exploración en el lecho marino, ya que en la mayoría de casos permiten obtener una mejor información a un costo reducido [17]. Los robots acuáticos o también denominados robots submarinos, pueden ser clasificados dependiendo el grado de autonomía, el tipo de misión a ejecutar y su sistema de propulsión.

2.4.1 Sistemas de propulsión acuáticos

Existe una gran variedad de sistemas de desplazamiento. En la Tabla.2.4 son presentados los principales mecanismos implementados en robots acuáticos.

Tipo de sistema	Características
Turbinas	Permiten la propulsión en conjunto con la maniobrabilidad al usar paletas de dirección.
Aletas	Replican el movimiento de algunos animales para desplazarse en medios acuosos.
Movimiento ondulante	Imitan el movimiento ondulante de algunas orugas o serpientes. Generalmente el desplazamiento se realiza sobre la superficie del entorno.
Caminante	Emplean patas, ruedas y demás recursos con el objetivo de desplazarse por el fondo o sobre la superficie del agua.

Tabla 2.4: Sistemas de desplazamiento en robots acuáticos

Fuente: de Garibay Pascual

2.4.2 Estudios relacionados a especies marinas

El Biologically Inspired Robotics Group (BIRG) en la Escuela Politécnica Federal de Lausana (EPFL), se ha interesado en comprender las habilidades de control y aprendizaje observables en los animales, desarrollando sistemas dinámicos para replicar los comportamientos más distintivos. Su foco principal se centra en sistemas capaces de evolucionar, adaptarse, auto-reorganizarse y repararse [18]. En breve, serán abordados los proyectos *Salamandra robotica II* y *AgnathaX* como una muestra de un amplio portafolio que este grupo de investigación tiene a su disposición.

- **Salamandra robotica II.** El proyecto Salamandra robotica II presenta a un dispositivo equipado con extremidades plegables y microcontroladores que permiten el cálculo distribuido de modelos de redes neuronales de la medula espinal, como también simulaciones de propiedades musculares. Un aspecto a resaltar es su modularidad, debido a que aparte de cambiar su morfología, es posible dividir el robot en diferentes partes y, aun así, funcionaría [19].
- **AgnathaX.** Es un robot de natación ondulatoria alargado que emula la morfología de una lamprea, esta equipado con sensores de fuerza distribuida diseñados para el estudio de distintos mecanismos neuronales que controlan la locomoción en la medula espinal. El propósito de este estudio es la investigación de la interacción de mecanismos centrales y periféricos [20].

2.5 Realidad aumentada

El término realidad aumentada integra un conjunto de tecnologías que permiten la superposición capas de información generada por computadoras sobre imágenes del mundo real, enriqueciendo la percepción de la realidad física con información digital. Actualmente, existen dos definiciones ampliamente aceptadas sobre la descripción de esta tecnología. La definición aportada por Ronald Azuma en 1997 propone tres requisitos que debe cumplir todo sistema de realidad aumentada: Combinar objetos virtuales con el mundo real, los objetos virtuales deben presentarse tridimensionalmente y debe permitir la interacción en tiempo real [21].

En cuanto a la definición creada por Paul Milgram y Fumio Kishino en 1994, sostienen que entre un entorno real y uno virtual se encuentra la realidad mixta, esta se subdivide en dos: La realidad aumentada (es más cercana a la realidad) y la virtualidad aumentada (es más cercana a la virtualidad pura) [22].

2.5.1 Niveles de realidad aumentada

Los niveles de realidad aumentada indican el grado de complejidad de una aplicación en términos de la tecnología que incorpora.

- **Nivel 0: Hiperenlaces en el mundo físico.** En este nivel, los activadores son códigos QR que enlazan con un sitio web. Un código QR (Quick Response) consiste en un módulo para almacenar información en una matriz de puntos o en un código de barras en dos dimensiones.
- **Nivel 1: Realidad aumentada basada en marcadores.** Los activadores para este nivel son los marcadores, un marcador es un objeto utilizado para la visualización de imágenes, sirven como punto de referencia o de medida. Es preciso tener presente el uso de patrones únicos que permitan a la cámara reconocer y determinar el objeto o conjunto de objetos a presentar.
- **Nivel 2: Realidad aumentada sin marcadores.** Existe la posibilidad de incorporar sistemas de realidad aumentada sin marcadores. En su defecto, los activadores son imágenes, objetos o localizaciones GPS. Durante los últimos años se han desarrollado navegadores de realidad aumentada que emplean recursos de teléfonos inteligentes para localizar y superponer capas de información en puntos de interés de nuestro entorno.
- **Nivel 3: Visión aumentada.** La Visión aumentada concibe a la realidad aumentada incorporada en gafas con el propósito de mostrar información al usuario sin que este requiera el uso de sus manos. Adicionalmente, la mayoría de estos dispositivos disponen de comandos de voz que permiten el acceso a internet [22].

2.5.2 Gafas inteligentes de realidad aumentada

La realidad aumentada ha ganado popularidad como medio para brindar asistencia técnica, el uso de dispositivos portátiles en diferentes actividades permite el acceso a contenido exclusivo en cualquier momento, mejora la participación de los trabajadores y aumenta la eficiencia de los entrenamientos. Por esta razón, las gafas inteligentes se han vuelto una herramienta indispensable que puede mantener e incluso aumentar el valor de una compañía.

Microsoft HoloLens

Las HoloLens son un par de gafas inteligentes de realidad mixta fabricado y desarrollado por Microsoft en 2016. En 2019 se lanzó al mercado una nueva versión bajo el nombre de HoloLens 2, estas últimas contemplan mejoras de confiabilidad, seguridad y escalabilidad. El sistema operativo de las HoloLens es una versión modificada de Windows 10. La selección de objetos, aplicaciones o botones se lleva a cabo mediante los gestos con las manos del usuario, siendo interpretados por las cámaras exteriores de las gafas inteligentes. Dispone

de un cursor que rastrea los movimientos de cabeza realizados por el usuario, el cual es compatible en funcionalidad con los movimientos de un mouse en la computadora. Además, cuenta con comandos de voz para determinadas acciones [23]. Es necesario incidir en que la imagen es proyectada sobre el cristal, adaptándose al ambiente y brindando la posibilidad de interacción con el individuo. Tiene como objetivo facilitar la labor de educadores, empresarios y diseñadores [24]. En la Figura.2.5 se puede apreciar un dispositivo Microsoft HoloLens.

Figura 2.5: Microsoft HoloLens
Fuente: Merino

Todas las operaciones realizadas en el dispositivo Microsoft HoloLens se basan en la manipulación de hologramas a través del visor; los hologramas son imágenes de luz bidimensionales o tridimensionales que representan un objeto virtual. El uso de luz para la proyección de imágenes a través de un visor ha sido trabajado desde la década de los años 2010, anunciado con el término de *holo-algo* e inspirado en una de las escenas de la famosa película de Star Wars donde R2D2 proyecta un holograma de la Princesa Leia [25]. Tal y como se muestra en la Figura.2.6

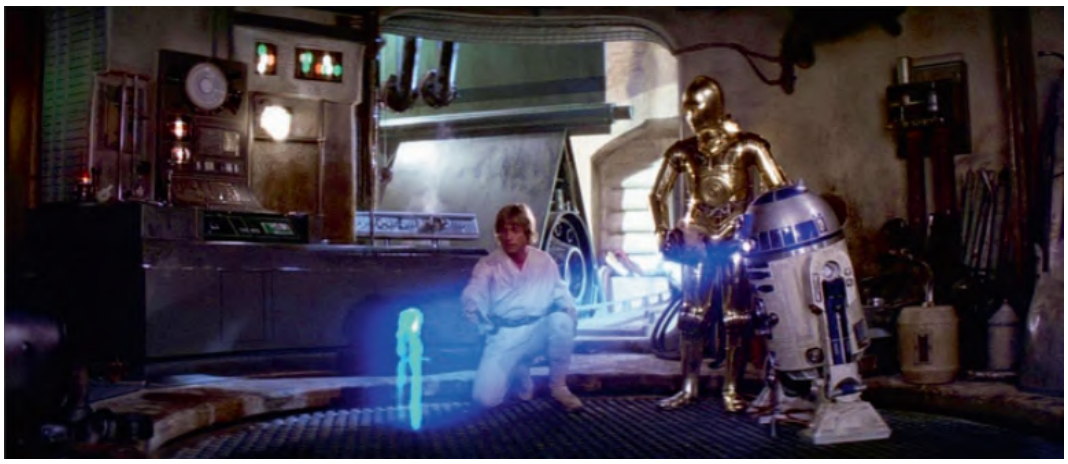

Figura 2.6: Holograma de la Princesa Leia proyectado en un espacio abierto
Fuente: Peddie

El creciente desarrollo tecnológico en los visores de Microsoft ha despertado gran interés en múltiples compañías para renovar el enfoque tecnológico en la capacitación de sus empleados. Mercedes-Benz, una de las principales marcas automovilísticas se encuentra equipando a todos sus concesionarios estadounidenses con dispositivos Microsoft HoloLens 2 para recortar los tiempos de servicio y mantenimiento de del personal técnico [26].

La Lockheed Martin, contratista principal que construye Orion en apoyo al programa Artemis de la NASA, ha incorporado el uso de HoloLens 2 en varias tareas de ensamblaje de la nave espacial diseñada para transportar astronautas a la luna y allanar el camino a Marte presentada en la Figura.2.7. Shelley Peterson, investigadora principal del desarrollo de sistemas basados en realidad aumentada y realidad mixta de la Lockheed Martin menciona que los técnicos tienen a disposición todo el contenido que necesitan saber para hacer esa tarea superpuesta en la estructura [27].

Figura 2.7: Los dispositivos Microsoft HoloLens 2 asistieron a los técnicos en la fabricación del escudo térmico de Artemis II

Fuente: Langston

2.5.3 Campos de aplicación

El desarrollo conceptual y técnico de la realidad aumentada ha desencadenado un gran número de contribuciones, dando lugar a numerosas innovaciones tecnológicas que representan mejoras incrementales a la civilización actual. Las diversas áreas de acción de la realidad aumentada comprenden desde la industria del entretenimiento hasta diversificarse en disciplinas de educación superior, áreas de la salud, procesos de capacitación y mantenimiento, turismo e inclusive entrenamiento para sofisticadas misiones espaciales.

2.5.3.1 Educación

Tras unos primeros pasos académicos, prestigiosas instituciones como el Instituto Tecnológico de Massachusetts y la Universidad de Harvard se encuentran desarrollando aplicaciones

de realidad aumentada en formatos de juegos con la intención de involucrar a estudiantes de educación secundaria en situaciones que integren experiencias del mundo real con información adicional generada por ordenadores y presentada a través de dispositivos móviles. Asimismo, han desarrollado aplicaciones en formato de juegos para el proceso enseñanza-aprendizaje de ciencias en forma colaborativa entre los estudiantes [28]. El resultado de los sistemas de enseñanzas mencionados anteriormente se observan en la Figura.2.8

Figura 2.8: Sistema de enseñanza de conceptos de geometría basado en la plataforma Studierstube
Fuente: Basogain et al.

2.5.3.2 Medicina

En los últimos años la realidad aumentada ha presentado un gran desarrollo el campo de la medicina. La atención clínica, por ejemplo, ha extendido sus intereses en el uso de la realidad aumentada para proporcionar al personal médico la capacidad de obtener una vista interna del paciente a través modelos tridimensionales en tiempo real y desde ubicaciones remotas, sin la necesidad de llevar a cabo procesos invasivos.

La clara necesidad en la educación sanitaria de involucrar experiencias prácticas situacionales en estudiantes y profesionales de la salud aumentó el interés en el estudio de la realidad aumentada durante los últimos años, destacando las siguientes creencias:

- Proporciona un rico aprendizaje contextual en competencias básicas.
- Ofrece oportunidades para un aprendizaje más auténtico, personalizado y exploratorio.
- Preserva la integridad del paciente si se comenten errores durante el entrenamiento [25].

La Figura.2.9 proporciona un ejemplo de aplicación de los estudiantes de la Universidad Case de la Reserva Occidental en el estudio de la anatomía de ser humano con dispositivos Microsoft HoloLens.

Figura 2.9: Estudiantes de la Universidad Case de la Reserva Occidental en Cleveland, aprendiendo anatomía con dispositivos Microsoft HoloLens
Fuente: Peddie

2.5.3.3 Aeroespacial

Uno de los campos más interesantes de aplicación de la realidad aumentada es sin lugar a duda la industria aeroespacial. En los últimos 3 años, la NASA a través de su programa AReProc, y la Agencia Espacial Europea a través de su programa MARSOP, han demostrado el enorme potencial que ofrece la realidad aumentada para mejorar los entrenamientos tipo práctico y justo a tiempo de los astronautas. La aplicación en tiempo real de esta tecnología durante el entrenamiento estimula capacidades de comunicación espontáneas y flexibles que no suelen encontrarse en los sistemas tradicionales.

Análogamente, el uso de sensores para la detección de gestos y visores de realidad aumentada permiten ampliar los alcances en el desarrollo de la actividad extravehicular (abreviado con el acrónimo EVA, Extravehicular activity) u otras aplicaciones que requieran el uso de trajes espaciales. Estas investigaciones asumen como objetivo principal la efectividad del control de gestos ejecutados con sistemas de manos libres, de forma remota o de proximidad [25].

En la Figura.2.10 se presenta al astronauta de la NASA e ingeniero de vuelo de la Expedición 65 Shane Kimbrough mientras da servicio al hardware de la Estación Espacial Internacional a través de unas gafas inteligentes de realidad aumentada.

Impulsado por nuestras ideas y el deseo de conocer otros mundos habitables más allá de la Tierra, la Jet Propulsion Laboratory (JPL) ha desarrollado dos experimentos basados en la implementación del dispositivo Microsoft HoloLens: *OnSight* y *Destination: Mars*.

Figura 2.10: Astronauta Shane Kimbrough usando unas gafas inteligentes de realidad aumentada
Fuente: Garcia

- **OnSight.** El software OnSight desarrollado por la NASA en asociación con Microsoft, es una nueva tecnología que permite a los científicos trabajar virtualmente en Marte mediante unas Microsoft HoloLens. OnSight utiliza datos reales obtenidos del rover para la creación de una simulación 3D del entorno marciano donde científicos de todo el mundo pueden reunirse a planificar las futuras operaciones del rover. El proyecto OnSight surgió como resultado de una asociación continua con la compañía Microsoft para investigar los avances en la interacción humano-robot [30].

- **Destination: Mars.** El proyecto "Destination: Mars" es una adaptación de OnSight. La NASA y Microsoft se han asociado nuevamente para ofrecer una exhibición interactiva desde unas Microsoft HoloLens. Los usuarios podrán "visitar" varios sitios en Marte, reconstruidos a partir de imágenes reales del Rover Curiosity Mars de la NASA. El astronauta Buzz Aldrin, quien participó en la misión Apollo 11, servirá como "guía turístico holográfico" en el viaje [31].

2.5.4 Tecnologías de software para desarrolladores

Algunas de las más importantes empresas a nivel global ofrecen kits de desarrollo para la creación de aplicaciones basadas en tecnologías de realidad virtual, realidad aumentada y realidad mixta. En este apartado se tratan algunas herramientas populares en la industria para dar cavidad a nuevas ideas y enfoques en entornos de realidad aumentada.

Khronos Group

El Grupo Khronos crea estándares abiertos libres de regalías para abordar todas las partes relacionadas con la fusión de cámaras, visión y sensores. La organización se estableció en el año 2000 para estabilizar y armonizar las API (Application Programming Interfaces) en procesadores gráficos tanto para la realidad virtual como para la realidad aumentada.

ARToolkit

El ARToolkit es una biblioteca de software para la creación de aplicaciones de realidad aumentada lanzado en 1999 y desarrollado por el investigador Hirokazu Kato del Instituto de Ciencia y Tecnología de Nara. Sus bibliotecas de seguimiento de video permiten calcular la posición y orientación real de la cámara en torno a los marcadores físicos.

- **Vuforia.** Con un poco más de 300.000 desarrolladores y 300 millones de instalaciones de aplicaciones, Vuforia se posiciona como uno de los kits de herramientas más accesibles, reconocidos y mejor valorados en todo el mundo. Vuforia permite a los desarrolladores insertar contenido 3D en entornos físicos, uno de sus principales objetivos es reconocer y rastrear imágenes impresas en una superficie plana.

- **HoloLens.** Microsoft no ofrece un kit de desarrollo de software de HoloLens específico debido a que lo integra como parte de su cartera de desarrolladores en Visual Studio, formando parte de un continuo de capacidades de Windows. Al ser Windows una de las principales plataformas informáticas del mercado, cuenta con una posición estratégica en aplicaciones comerciales e industriales.

- **ViewAR.** Es un proveedor de aplicaciones de visualización 3D para soluciones empresariales móviles fundado en Vienna, Austria, en el año 2010. El SDK (Software Development Kit) de ViewAR cuenta con interfaces HTML, Javascript y CSS personalizables que proporcionan la captura de objetos para la manipulación y personalización de modelos. También admite sistemas de seguimiento como Vuforia, Metaio, Pointcloud, entre otros [25].

2.5.5 Herramientas de modelado

La industria de la animación 2D y 3D se está expandiendo continuamente buscando aumentar la demanda de simulaciones de entornos y productos para compañías especializadas. Por ejemplo, la creación de animaciones interactivas para empresas constructoras que le permitan al potencial residente caminar y visualizar su futura propiedad.

- **Blender.** Es un programa informático multiplataforma de producción 3D enfocado a la creación de animaciones, gráficos, modelado y renderizado. Blender se ha popularizado por su versatilidad y compatibilidad con otros programas como el motor Unity. Cuenta con su propio formato de archivos, pero admite un amplio número de formatos 2D, 3D y de video entre los cuales se encuentran DirectX, MilkShape 3D, OpenInventor o Stl [32].

- **Autodesk 3ds Max.** Desarrollado por Autodesk y reconocido como uno de los programas de modelado de 3D masivos más utilizados en la industria, se centra en la creación de gráficos y animaciones que permitan a los diseñadores crear una visión más realista de sus productos. Es ampliamente usado en proyectos de arquitectura, películas, anuncios de televisión y efectos especiales.

- **SketchUp.** Es un software de diseño 3D multipropósito especialmente dirigido a entornos relacionados con la arquitectura, ingeniería civil o diseño industrial. SketchUp permite exportar proyectos de construcciones a distintos formatos gráficos para posteriormente realizar procesos de renderización de objetos y escenas con diversas herramientas de personalización en etapas de diseño esquemático o construcción.

2.6 Revisión de trabajos previos

El presente apartado tiene como objetivo describir una revisión bibliográfica que recoge aportaciones e investigaciones de diferentes disciplinas relacionadas con la robótica modular y la realidad aumentada a fin de brindar un mejor entendimiento sobre el alcance que esta solución supone. De esta manera, se establecen los antecedentes que sirven como base para el desarrollo del presente trabajo de grado.

2.6.1 Antecedentes

Las misiones espaciales modernas presentan una combinación que integra el trabajo colaborativo entre seres humanos, robots y sistemas autónomos a bordo de naves espaciales. Los beneficios que ofrece la implementación de nuevas tecnologías permitirán ampliar el alcance de la exploración más allá de las limitaciones relacionadas con los vuelos tripulados, reducción de riesgos y costos. Representando potenciales mejoras en el rendimiento de misiones científicas, de exploración y operativas. En [33], se presenta el diseño de un sistema robótico modular homogéneo basado en 4 GDL por módulo, incluye una articulación prismática para aumentar la versatilidad de su reconfiguración y capacidad de locomoción. Los módulos ModRED (Modular Robot for Exploration and Discovery) son desarrollados con mecanismos de acoplamiento de un solo lado que permiten configuraciones de tipo cadena y conducen a configuraciones de tipo híbrido. Este trabajo también aborda inconvenientes relacionados con la reconfiguración dinámica de un robot modular auto configurable y simula varios movimientos de locomoción a través del simulador Webots los cuales son implementados posteriormente en el sistema ModRED real. En la Figura.2.11 se presentan diferentes representaciones y comparaciones de los espacios de trabajo para los módulos del sistema.

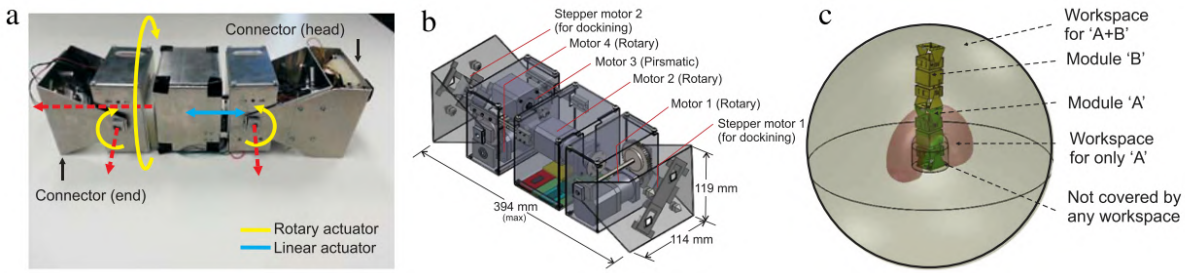

Figura 2.11: Cada módulo tiene cuatro GDL (b) Representación del módulo 3D (c) Comparación de los espacios de trabajo aproximados para un módulo único y uno doble
Fuente: Baca et al.

Por otra parte, en [13], se describe el diseño de un sistema robótico modular heterogéneo destinado a dar una solución rápida a múltiples tareas. La arquitectura del sistema se divide en módulos, M-Robots y colonias, partiendo como referencia de un inventario compuesto por tres módulos: Módulo de potencia y control, módulo de conjunto y módulo especializado. Cada módulo tiene como objetivo equilibrar la versatilidad y funcionalidad del sistema. El diseño presentado permite una fabricación rápida y rentable que puede ser implementado en diferentes aplicaciones durante tareas de teleoperación. Dependiendo de la tarea a realizar,

el operador tendrá la posibilidad de decidir qué tipo de robot puede proporcionar el mejor rendimiento dentro de la misión.

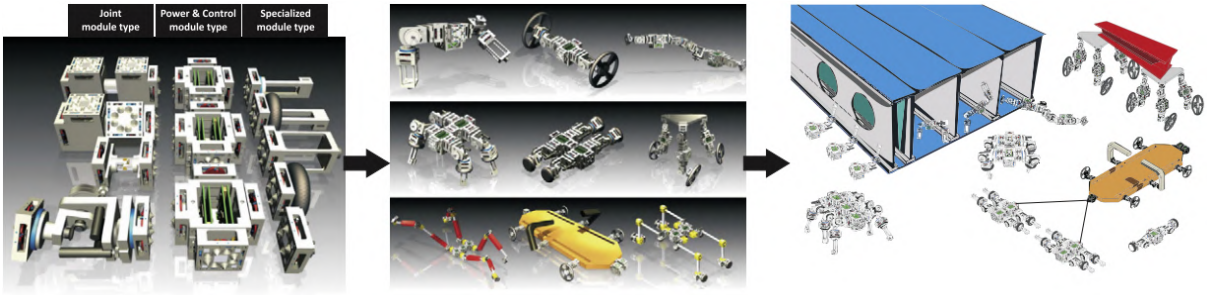

Figura 2.12: Diversas configuraciones de un sistema modular heterogéneo
Fuente: Baca et al.

Además, en [18], se pone a consideración el problema de descubrir y representar la topología de un sistema robótico modular auto configurable (MSR) en el cual los módulos no tienen información a priori sobre la ubicación y las conexiones con otros módulos que participan en el proceso de reorganización. Por tanto, se propone combinar dos características principales relacionando la forma geométrica del módulo y a la representación gráfica de la conectividad entre estos. La reorganización de los módulos que hacen parte de la configuración actual da lugar a la creación de una nueva estructura robótica, los módulos pueden ser agregados o eliminados sobre la marcha. Proporcionando al MSR la capacidad de superar el obstáculo y posteriormente continuar con el equipo hacia su destino. Adicionalmente, este trabajo combina protocolos de comunicación infrarrojos, XBee y recursos computacionales limitados disponibles en un módulo. De la Figura.2.13 es posible visualizar las posibles configuraciones del robot modular con ModRED II y algunas representaciones relacionadas a la unión de sus módulos.

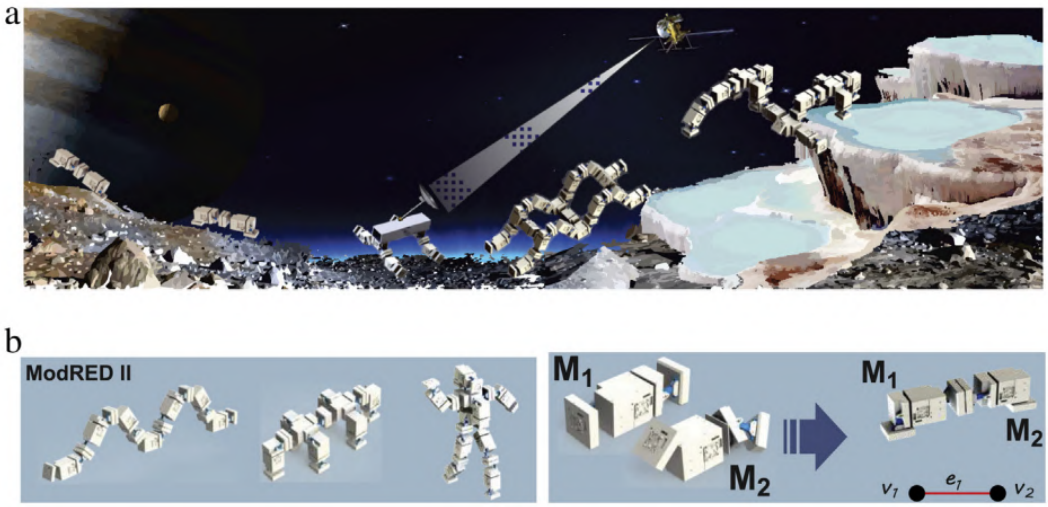

Figura 2.13: (a) Posibles configuraciones con ModRED II. (b) Unión de módulos
Fuente: Baca et al.

Para concluir, se recopila un conjunto de aportes relacionados con sistemas de realidad aumentada como fuente de distribución de recursos didácticos para el estudio de diversas disciplinas. Por ejemplo, en [35], un sistema basado en realidad virtual y realidad aumentada es desarrollado para el aprendizaje de un microscopio biológico. El sistema describe un modelo tridimensional detallado de la estructura del microscopio en donde cada componente estará representado por sus interrelaciones topológicas y asociaciones. Un subsistema de realidad aumentada permitió a los participantes conocer la estructura y funciones del instrumento a través de un dispositivo móvil que capturaba una fotografía de un microscopio de un libro de texto. El resultado de este trabajo de investigación se muestra en la Figura.2.14.

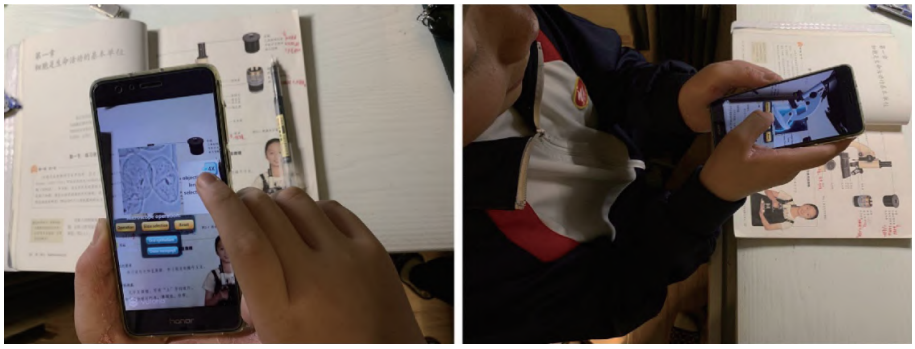

Figura 2.14: Participación de estudiantes en el sistema experimental
Fuente: Zhou et al.

Otro ejemplo [36], refiere a un sistema basado en realidad aumentada compuesto por imágenes que actúan como marcadores para detectar el objeto que el usuario está observando e indicarle el contenido multimedia que este dispone, como se observa en la Figura.2.15. Para el desarrollo de la primera versión se realizaron tutoriales y guías de fuentes como Mixed Reality Academy, Unity3D, Vuforia Engine, entre otros. Para él desacoplo de los contenidos multimedia se utilizó la librería Express para la implementación de un servidor de archivos estáticos. La segunda fase del desarrollo del proyecto trajo consigo la inclusión de un sistema de funcionalidades de texto a voz, los archivos de audio son solicitados una vez y almacenados en el servidor de contenidos.

Figura 2.15: Ejemplo de la interfaz de usuario para un objeto capturada desde Unity
Fuente: Sabogal Rojas et al.

3 Metodología

La metodología seleccionada para el desarrollo del presente estudio se ilustra en la Figura.3.1. Esta describe los pasos claves que conforman cada una de las etapas desde el planteamiento del problema, hasta la validación de la propuesta de manera experimental. Cabe mencionar que las etapas iniciales ya han sido abordadas en apartados anteriores. Por lo tanto, los siguientes apartados presentarán la descripción, análisis, simulaciones e interfaces de visualización, uso de protocolos de comunicación y la validación de referencias en el sistema físico. Finalmente, cada una de las fases serán documentadas de tal manera que permitan poner fácilmente en práctica los conceptos matemáticos, simulaciones, interfaces de visualización y el control del robot de estudio.

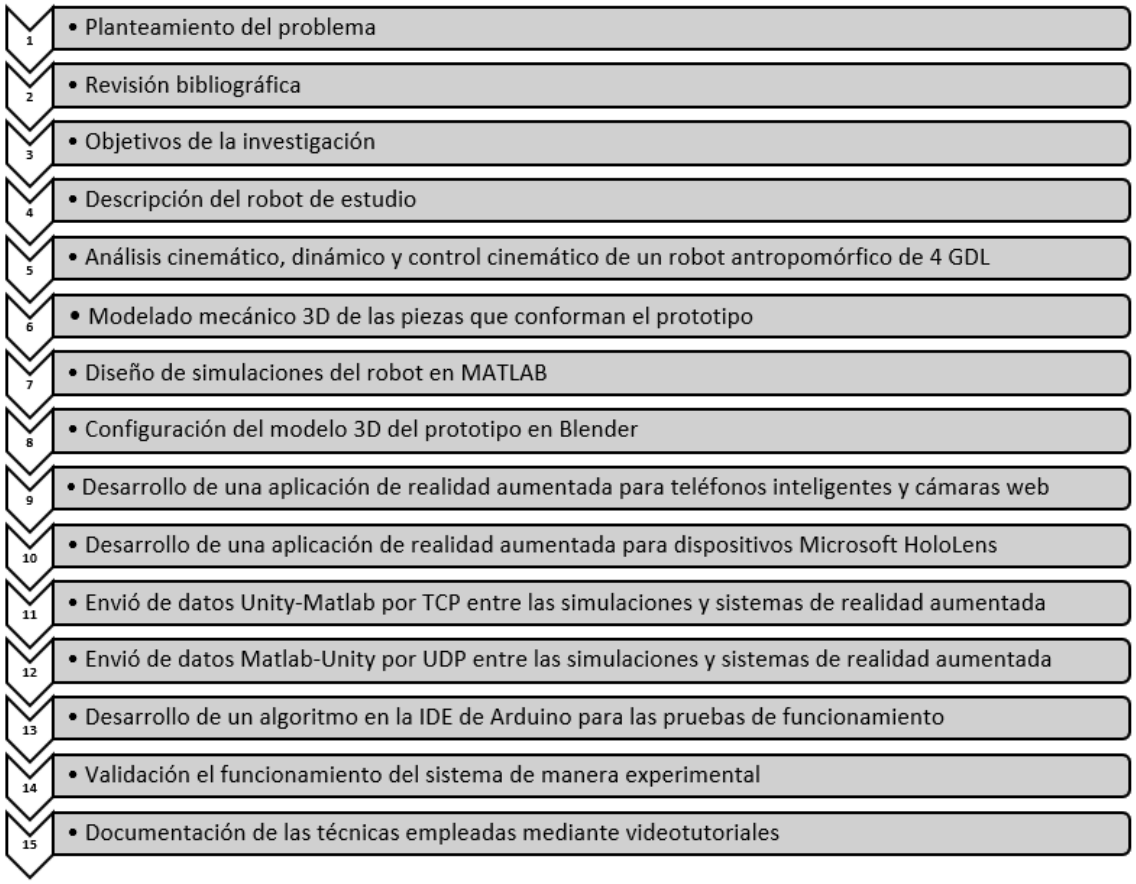

Figura 3.1: Diagrama descriptivo de la metodología secuencial
Fuente: Autor

3.1 Descripción del robot de estudio

En cuanto al propósito y las funcionalidades del robot, ha sido diseñado para desarrollar tareas de manipulación o desplazamiento en medios acuáticos. Se pretende que el prototipo pueda acoplarse en otros dispositivos para ampliar la diversidad de aplicaciones que este puede ejercer, ya sea para desplazarse o manipular objetos con mayor facilidad. La Figura.3.2 presenta de izquierda a derecha, el prototipo de brazo robótico integrado con otro dispositivo para realizar distintas actividades. Por otro lado, se puede evidenciar como es posible disponer de diferentes herramientas como elemento terminal dependiendo del escenario de aplicación.

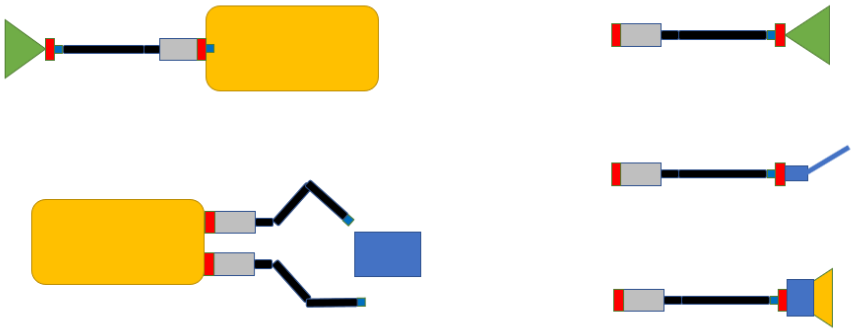

Figura 3.2: Aplicaciones del brazo robótico acuático modular
Fuente: José Baca, Department of Engineering | Texas A&M University-Corpus Christi

Además, el mecanismo conector localizado en el elemento terminal del sistema le permite al robot ser integrado con otros sistemas robóticos modulares y acoplar hasta un máximo de cuatro ejemplares con el propósito de extender su alcance. En la Figura.3.3 se aprecia el resultado de las configuraciones mencionadas anteriormente.

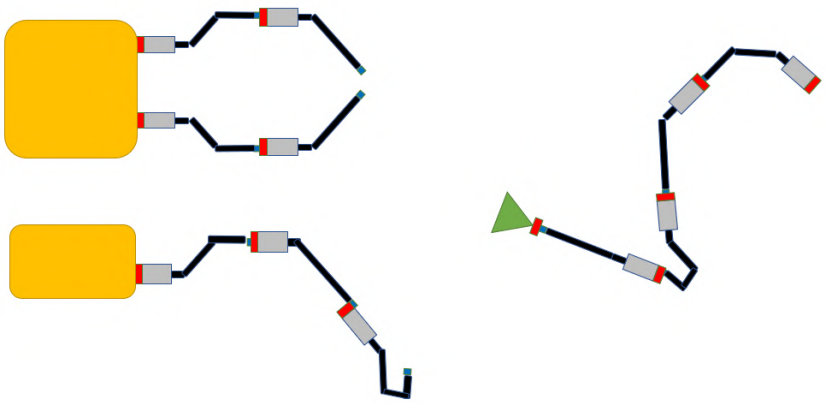

Figura 3.3: Extensiones del sistema
Fuente: José Baca, Department of Engineering | Texas A&M University-Corpus Christi

3.1.1 Estructura mecánica

El sistema físico está compuesto por un robot antropomórfico de 4 grados de libertad, con todas sus articulaciones rotacionales. El accionamiento es llevado a cabo por servomotores DYNAMIXEL XL430-W250-T; un servomotor por grado de libertad. Las piezas que conforman los eslabones corresponden a juegos FR12-H101K y FR12-S102K unidos mediante tornillos de perno FHS M2.5x14 y tornillos de llave de perno M2×3. En la Figura.3.4 se aprecia el ensamble tridimensional del prototipo.

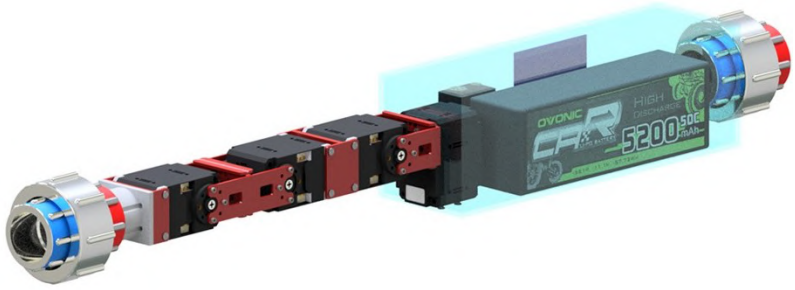

Figura 3.4: Ensamble tridimensional del prototipo
Fuente: José Baca, Department of Engineering | Texas A&M University-Corpus Christi

Con el objetivo de brindar un mejor entendimiento sobre el montaje de las piezas que componen el sistema físico se ha proporcionado un esquemático que relaciona desde los elementos centrales a los más extremos, considerando de forma minuciosa todos los componentes que hacen parte del prototipo. En la Figura.3.5 se presenta una vista explosionada del robot de estudio. En la Tabla.3.1 se describen las partes que hacen parte del prototipo.

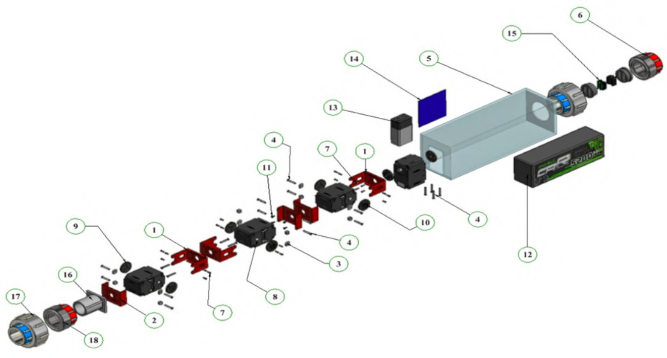

Figura 3.5: Vista explosionada del prototipo
Fuente: José Baca, Department of Engineering | Texas A&M University-Corpus Christi

Lista de partes (Part List)		
Artículo (Item)	**Cantidad (QTY)**	**Número de pieza (Part Number)**
1	3	Marco guía (Frame Idler)
2	3	Marco (Frame)
3	12	Espaciador del motor (Motor Spacer)
4	20	Perno FHS M2.5 14 (Bolt FHS M2.5 14)
5	1	Encapsulado de vidrio (Glass Encasement)
6	1	Unión F con conectores (Union F With Connectors)
7	22	Perno de llave M2x3 (Wrench Bolt M2x3)
8	4	Motor DYNAMIXEL XL430 W250-T (DYNAMIXEL XL430 W250-T Motor)
9	4	Lado de la etiqueta adhesiva de la pieza giratoria del motor (Motor Rotating Part Sticker Side)
10	4	Parte giratoria del motor, lado opuesto (Motor Rotating Part, Opposite side)
11	4	ISO 7046 - 1 - M2x5 - 4.8 - Z
12	1	Estuche batería Lipo Ovonic 11.1V (Ovonic 11.1V Hardcase Lipo Battery)
13	1	Batería Energizer 9V (Energizer 9V Battery)
14	1	Placa de circuito impreso DYNAMIXEL (DYNAMIXEL Shield)
15	1	Conectores Unión M (Union M Connectors)
16	1	Adaptador de brazo (Arm Adapter)
17	1	Unión M (Union M)
18	1	Unión F (Union F)

Tabla 3.1: Índice de componentes
Fuente: José Baca, Department of Engineering | Texas A&M University-Corpus Christi

3.1.2 Elemento terminal

Una de las partes fundamentales de este sistema es su mecanismo de acoplamiento situado en el efector final del robot. Este componente es necesario para que el prototipo pueda interactuar con otros módulos, ser reconfigurado para otras aplicaciones y transmitir información, pares mecánicos y potencia eléctrica a través de todo el sistema. El objetivo de este desarrollo pretendía dotar al robot de un elemento terminal simple, independiente de la orientación y de carácter universal. En la Figura.3.6 se ilustra la solución propuesta por el equipo Aqua-Bots de la Universidad de Texas A&M-Corpus Christi inspirada en un diseño con conectores "macho" y "hembra" para el acoplamiento de diversos módulos.

Figura 3.6: Diseño del mecanismo de acoplamiento
Fuente: José Baca, Department of Engineering | Texas A&M University-Corpus Christi

Al examinar los componentes presentados en la Figura.3.7 observe que el conector se ha elaborado en material ¾ de PVC, este incorpora la unión de múltiples elementos entre los cuales es posible destacar conectores magnéticos y juntas de tipo toroidal (O-Rings) para mantener la arquitectura de comunicación y prevenir el ingreso de fluidos en el sistema.

Figura 3.7: Vista explosionada del mecanismo de acopamiento
Fuente: José Baca, Department of Engineering | Texas A&M University-Corpus Christi

3.1.3 Aplicaciones y configuraciones

Entre las diferentes aplicaciones que puede desarrollar este dispositivo se hace hincapié en la posibilidad de ser integrado con otros sistemas modulares que compartan estándares de compatibilidad para proveer nuevas funciones, como se observa en la Figura.3.8

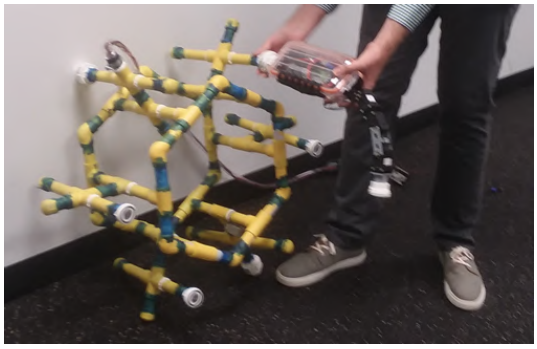

Figura 3.8: Acoplamiento del prototipo a un sistema modular con arquitectura de celosía
Fuente: José Baca, Department of Engineering | Texas A&M University-Corpus Christi

3.2 Fase I: Análisis matemático

3.2.1 Modelo cinemático

En la primera sección se requiere encontrar la matriz de transformación homogénea que relacione la posición y orientación del efector final con respecto al sistema de referencia de la base del robot. La segunda sección consiste en encontrar los valores que deben adoptar las coordenadas articulares para adoptar una localización espacial. En la Figura.3.9 se ilustran las dimensiones del prototipo.

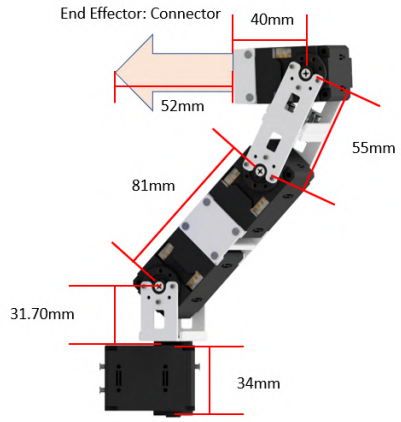

Figura 3.9: Datos iniciales del brazo robot
Fuente: José Baca, Department of Engineering | Texas A&M University-Corpus Christi

Cinemática directa

Para la resolución del problema cinemático directo se ha implementado el algoritmo de Denavit-Hartenberg para la localización de los sistemas de referencia de cada una de las articulaciones. Dicho algoritmo se enuncia en el libro *Fundamentos de Robótica* (2a. ed.), (Páginas 97-98), por A. Barrientos, L. Peñin, C. Balaguer, R. Aracil, 2007, Madrid: McGraw-Hill.

A continuación, se describe el algoritmo de Denavit-Hartenberg:

1. Numerar los eslabones comenzando con 1 (correspondiente al primer eslabón móvil de la cadena) y acabando con n (último eslabón móvil). Se numerará como eslabón 0 a la base fija del robot.
2. Numerar cada articulación comenzando por 1 (la correspondiente al primer grado de libertad) y acabando en n.
3. Localizar el eje de cada articulación. Si ésta es rotativa, el eje será su propio eje de giro. Si es prismática, será el eje a lo largo del cual se produce el desplazamiento.
4. Para i de 0 a n-1 situar el eje Z_i sobre el eje de la articulación i+1.
5. Situar el origen del sistema de la base S_0 en cualquier punto del eje z_0. Los ejes x_0 e y_0 se situarán de modo que formen un sistema dextrógiro con z_0.
6. Para i de 1 a n-1, situar el sistema S_i (solidario al eslabón i) en la intersección del eje z_i con la línea normal común a $z_{i\text{-}1}$ y z_i. Si ambos ejes se cortasen se situaría S_i en el punto de corte. Si fuesen paralelos S_i se situaría en la articulación i+1.
7. Situar x_i en la línea normal común a $z_{i\text{-}1}$ y z_i.
8. Para i de 1 a n-1, situar y_i de modo que forme un sistema dextrógiro con x_i y z_i.
9. Situar el sistema S_n en el extremo del robot de modo que zn coincida con la dirección de $z_{n\text{-}1}$ y x_n sea normal a $z_{n\text{-}1}$ y z_n.
10. Obtener θi como el ángulo que hay que girar en torno a $z_{i\text{-}1}$ para que $x_{i\text{-}1}$ y x_i queden paralelos.
11. Obtener d_i como la distancia, medida a lo largo de $z_{i\text{-}1}$, que habría que desplazar $S_{i\text{-}1}$ para que x_i y $x_{i\text{-}1}$ quedasen alineados.
12. Obtener ai como la distancia medida a lo largo de x_i, que ahora coincidiría con $x_{i\text{-}1}$, que habría que desplazar el nuevo $S_{i\text{-}1}$ para que su origen coincidiese con S_i.
13. Obtener αi como el ángulo que habría que girar en torno a x_i, que ahora coincidiría con $x_{i\text{-}1}$, para que el nuevo $S_{i\text{-}1}$ coincidiese totalmente con S_i.
14. Obtener las matrices de transformación A_i^{i-1} .
15. Obtener la matriz de transformación que relaciona el sistema de la base con el del extremo del robot $T = A_{01}, A_{12} \dots A_i^{i-1}$

16. La matriz T define la orientación (submatriz de rotación) y posición (submatriz de traslación) del extremo referido a la base en función de las n coordenadas articulares.

Para hacer más amena la explicación, se ha desarrollado un esquema simple del brazo robot con el objetivo de indicar los sistemas de coordenadas de cada una de las articulaciones, como se ilustra en la Figura.3.10. Esto con base a las siguientes consideraciones: El primer eslabón tiene una longitud de 65.70 mm, el segundo eslabón tiene una longitud de 81 mm, el tercer eslabón tiene una longitud de 55 mm y el cuarto eslabón tiene una longitud de 92 mm.

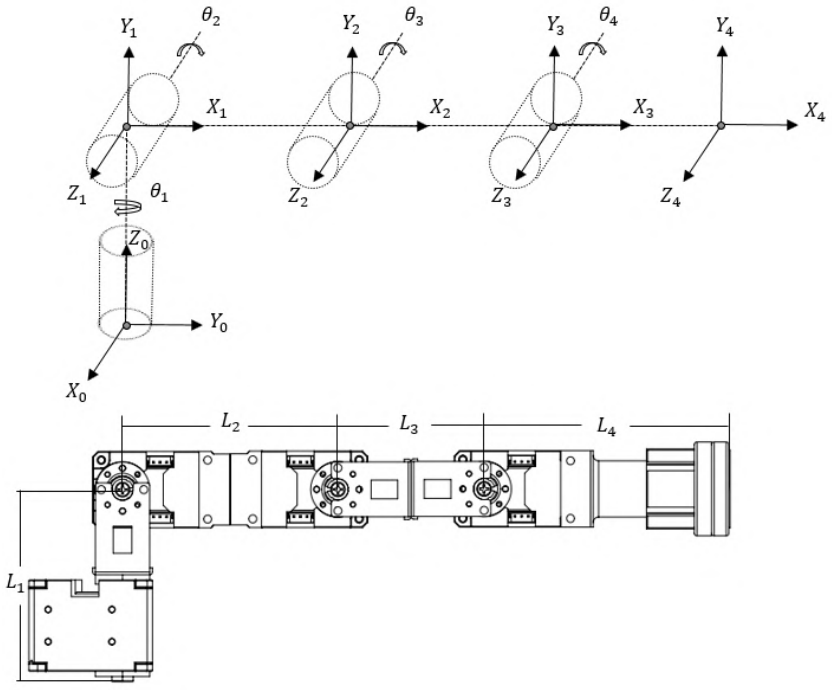

Figura 3.10: Sistemas de coordenadas del robot
Fuente: Autor

Seguidamente, utilizando las medidas de las articulaciones y la ubicación de los sistemas de coordenados, se determinan los parámetros de Denavit-Hartenberg (Tabla 3.2).

Articulaciones	θ(Grados)	d(mm)	a(mm)	α(Grados)
1	q_1+90	L_1	0	90
2	q_2	0	L_2	0
3	q_3	0	L_3	0
4	q_4	0	L_4	0

Tabla 3.2: Parámetros según Denavit-Hartenberg
Fuente: Autor.

Una vez calculados los parámetros de cada eslabón se presenta un algoritmo computacional

en Matlab para el cálculo de las matrices de transformación homogénea para cada uno de los sistemas coordenados. Se ha introducido como ejemplo la posición original del robot para demostrar el funcionamiento del algoritmo, dicha posición obedece a un ángulo igual a cero en cada una de las articulaciones.

$$A_{01} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

$$A_{12} = \begin{bmatrix} 1 & 0 & 0 & L_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

$$A_{23} = \begin{bmatrix} 1 & 0 & 0 & L_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

$$A_{34} = \begin{bmatrix} 1 & 0 & 0 & L_4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

Finalmente, se obtiene la matriz de transformación homogénea T. Esta matriz indica la localización del sistema final con respecto al sistema de referencia de la base del robot.

$$T = A_{01} * A_{12} * A_{23} * A_{34} = A_{04} \tag{3.5}$$

$$A04 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & L_2 + L_3 + L_4 \\ 0 & 1 & 0 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

Código 3.1: Algoritmo para el cálculo de la cinemática directa

```
clear all
clc
close all

% ------------------------------------------------------------
% Declaración de variables
% ------------------------------------------------------------
LA=34;
LB=31.70;
L1=LA+LB;
```

```
    L2=81;
    L3=55;
    LC=40;
    LD=52;
    L4=LC+LD;

    L = [L1,L2,L3,L4];

    A00 = eye(4);

% ------------------------------------------------------------------------
% Tabla de parámetros Denavit-Hartenberg
% ------------------------------------------------------------------------
    theta_DH = [ q(1)+pi/2, q(2), q(3), q(4) ];
    d_DH = [ L(1), 0, 0, 0 ];
    a_DH = [ 0, L(2), L(3), L(4) ];
    alpha_DH = [ pi/2, 0, 0, 0 ];

% ------------------------------------------------------------------------
% Cálculo de las matrices A01, A02, A03 y A04
% ------------------------------------------------------------------------
    A01 = DH(theta_DH(1), d_DH(1), a_DH(1), alpha_DH(1));
    A12 = DH(theta_DH(2), d_DH(2), a_DH(2), alpha_DH(2));
    A23 = DH(theta_DH(3), d_DH(3), a_DH(3), alpha_DH(3));
    A34 = DH(theta_DH(4), d_DH(4), a_DH(4), alpha_DH(4));

% ------------------------------------------------------------------------
% Matrices con respecto al sistema cero
% ------------------------------------------------------------------------
    A02 = A01*A12;
    A03 = A02*A23;
    A04 = A03*A34;
```

Cinemática inversa

El problema de la cinemática inversa de este trabajo ha sido desarrollado a través del método de desacoplo cinemático. En este se llevan a cabo métodos geométricos que permiten obtener los valores de las tres primeras articulaciones por medio de relaciones trigonométricas. De manera similar, se emplean ecuaciones relacionadas con el modelo cinemático anteriormente expuesto para despejar las variables restantes haciendo uso de las matrices de rotación. Inicialmente, es debe expresar la posición de la muñeca P_m en función del sistema de coordenadas correspondiente a la base del robot como se observa en la Figura.3.11

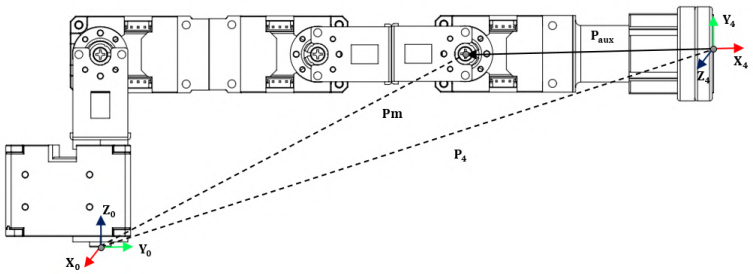

Figura 3.11: Posición de la muñeca en relación con los vectores P_{aux} y P_4
Fuente: Autor

Donde:

$$P_m = P_4 + P_{aux} \tag{3.7}$$

$$P_{aux} = -X_4 * L_4 \tag{3.8}$$

Reemplazando la ecuación (3.8) en la ecuación (3.7) se obtiene:

$$P_m = P_4 - X_4 * L_4 \tag{3.9}$$

La primera coordenada articular se calcula de forma sencilla a partir de una vista superior del prototipo presentada en la Figura.3.12. Obteniendo como resultado las ecuaciones (3.10) y (3.11).

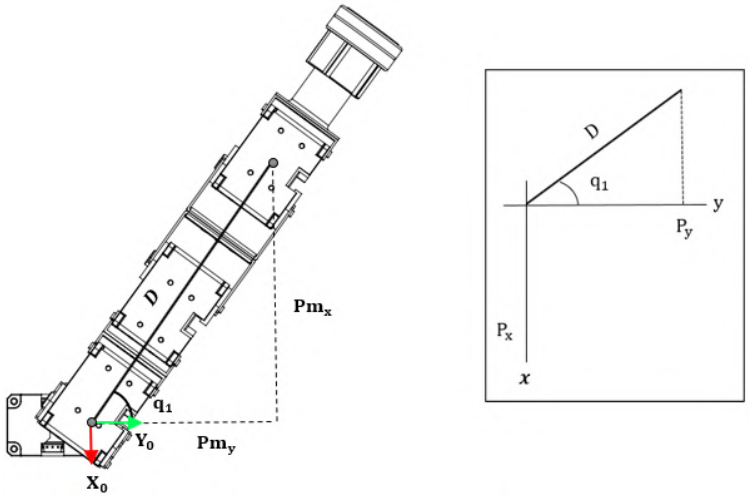

Figura 3.12: Vista superior del robot para el cálculo de la primera coordenada articular
Fuente: Autor

$$D = \sqrt{Pm_y^2 + Pm_x^2} \tag{3.10}$$

$$q_1 = atan2\left(-Pm_x, Pm_y\right) \tag{3.11}$$

Las coordenadas articulares q_2 y q_3 se obtienen calculando los valores geométricos desde una vista lateral del robot de estudio, esta configuración se ilustra en la Figura.3.13. Haciendo uso de identidades trigonométricas se obtiene la ecuación (3.12)

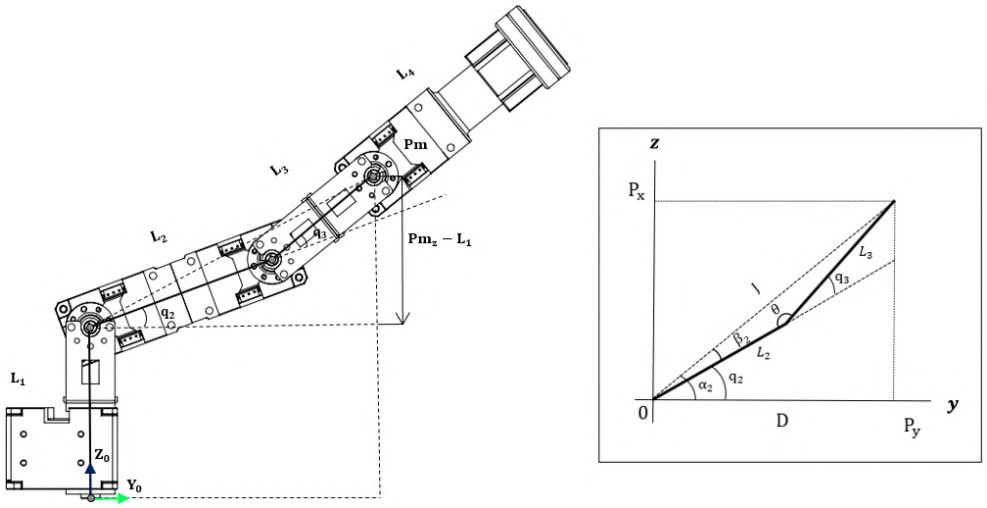

Figura 3.13: Vista lateral del robot para el cálculo de la segunda y tercera coordenada articular
Fuente: Autor

$$J^2 = D^2 + (Pm_z + L_1)^2 \tag{3.12}$$

Aplicando la ley de cosenos se obtiene la ecuación (3.13)

$$J^2 = L_2^2 + L_3^2 - 2\,(L_2)\,(L_3)\,cos\theta \tag{3.13}$$

Despejando θ en la ecuación (3.14) y utilizando identidades trigonométricas para la resta de dos ángulos en la ecuación (3.15) se determina la ecuación (3.16)

$$\theta = 180^\circ - q_3 \tag{3.14}$$

$$cos\,(180^\circ - q_3) = -cosq_3 \tag{3.15}$$

$$J^2 = L_2^2 + L_3^2 - 2\,(L_2)\,(L_3)\,cosq_3 \tag{3.16}$$

Seguidamente, se despeja $cosq_3$ la ecuación (3.16) dando lugar a la ecuación (3.17)

$$cosq_3 = \frac{J^2 - \left(L_2^2 + L_3^2\right)}{2\,(L_2)\,(L_3)} \tag{3.17}$$

Debido a la dos posibles configuraciones que puede tomar el sistema para una misma ubicación final se ha incluido la variable *CODO* en la resolución de la ecuación (3.18). Pos-

teriormente, se procede a calcular q_3 por medio de las variables $cosq_3$ y $senq_3$ en la ecuación (3.19).

$$senq_3 = CODO\sqrt{1 - cos^2 q_3} \tag{3.18}$$

$$q_3 = atan2\left(senq_3, cosq_3\right) \tag{3.19}$$

Una vez determinado q_3 es posible establecer la coordenada articular q_2 teniendo en cuenta las siguientes consideraciones de la Figura.3.13

$$\beta_2 = atan2\left(L_3 sen\left(q_3\right), L_2 + L_3 cos\left(q_3\right)\right) \tag{3.20}$$

$$\alpha_2 = atan2\left(Pm_z + L_1, D\right) \tag{3.21}$$

Expresando q_2 como la resta de las variables α_2 y β_2

$$q_2 = \alpha_2 - \beta_2 \tag{3.22}$$

El cálculo de la coordenada articular q_4 requiere considerar a un observador cuya ubicación será opuesta al eje de giro de la articulación 3. Suponga que ambos sistemas presentan la misma posición, como se ilustra en la Figura.3.14. De esta manera, se obtienen las ecuaciones (3.23) y (3.24) mediante el producto punto de las variables indicadas.

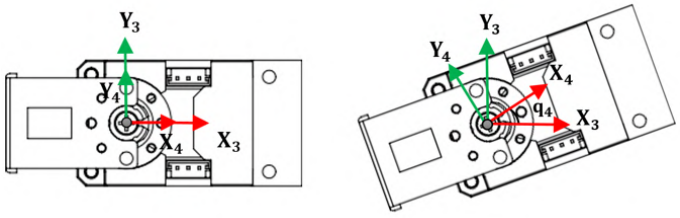

Figura 3.14: Vista lateral del robot para el cálculo de la cuarta coordenada articular
Fuente: Autor

$$y_3 \cdot x_4 = cos\left(90 - q_4\right) = cos\left(90\right) cos\left(q_4\right) + sen\left(90\right) sen\left(q_4\right) = senq_4 \tag{3.23}$$

$$x_3 \cdot x_4 = cosq_4 \tag{3.24}$$

Por último, se despeja la variable q_4 descrita en la ecuación (3.25)

$$q_4 = atan2\left(y_3 \centerdot x_4, x_3 \centerdot x_4\right) = atan2\left(senq_4, cosq_4\right) \tag{3.25}$$

3.2.2 Modelo dinámico

El modelo dinámico de un robot ocupa la relación entre el movimiento del robot y las fuerzas aplicadas en el mismo. A través de este, es posible determinar que movimiento surge al aplicar unas fuerzas o que fuerzas hay que aplicar para obtener un movimiento determinado. Asimismo, juega un papel preponderante para conseguir los siguientes objetivos:

1. Simulación del movimiento del robot.
2. Diseño y evaluación de la estructura mecánica.
3. Dimensionamiento de los actuadores.
4. Diseño y evaluación del modelo dinámico del robot.

Por otra parte, el planteamiento del equilibrio de fuerzas y pares de un robot origina los denominados modelo dinámico directo e inverso.

- **Modelo dinámico directo.** Expresa la evolución temporal correspondiente a las coordenadas articulares en relación a las fuerzas y pares que se generan en el robot.
- **Modelo dinámico inverso.** Expresa las fuerzas y pares que se generan en el robot en relación con la evolución de las coordenadas articulares y sus respectivas derivadas [6].

Como planteamiento alternativo, se propone usar la formulación Lagrange-Euler basada en consideraciones energéticas. El esquema general del modelo dinámico dada la formulación Lagrangiana establece las ecuaciones 3.26 y 3.27.

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_i} - \frac{\partial \mathcal{L}}{\partial q_i} = \tau \tag{3.26}$$

$$\mathcal{L} = K - U \tag{3.27}$$

Donde:

q_i : Coordenadas generalizadas (articulares).

τ : Vector de fuerzas y pares aplicados en las q_i

$\mathcal{L}$: Función Lagrangiana.

K : Energía cinética.

U : Energía potencial.

La implementación del algoritmo de Lagrange-Euler contempla parámetros propios de la estructura mecánica del robot. Por ejemplo, inercias (Tabla.3.3), centros de masas (Tabla.3.4) y masas (Tabla.3.5) atendiendo a los materiales y formas características del sistema. Estos parámetros han sido calculados mediante programas de diseño asistido por computadora.

Inercia eslabón 1 [kg-m^2]		
I_{xx}=3.348e-05	I_{xy}=0	I_{xz}=0
I_{yx}=0	I_{yy}=1.392e-05	I_{yz}=0
I_{zx}=0	I_{zy}=0	I_{zz}=2.441e-05
Inercia eslabón 2 [kg-m^2]		
I_{xx}=4.189e-05	I_{xy}=5e-08	I_{xz}=6.5e-07
I_{yx}=5e-08	I_{yy}=0.00048099	I_{yz}=0
I_{zx}=6.5e-07	I_{zy}=0	I_{zz}=0.00046587
Inercia eslabón 3 [kg-m^2]		
I_{xx}=2.785e-05	I_{xy}=0	I_{xz}=-3e-08
I_{yx}=0	I_{yy}=0.00010471	I_{yz}=0
I_{xz}=-3e-08	I_{zy}=0	I_{zz}=8.658e-05
Inercia eslabón 4 [kg-m^2]		
I_{xx}=2.093e-05	I_{xy}=3e-08	I_{xz}=2.6e-07
I_{yx}=3e-08	I_{yy}=7.301e-05	I_{yz}=0
I_{zx}=2.6e-07	I_{zy}=0	I_{zz}=6.547e-05

Tabla 3.3: Inercia de los elementos con respecto a sus ejes de rotación generados en el software CAD

Fuente: Autor.

Centro de masa [m]			
Eslabón 1	x=0	y=-0.02025412	z=0
Eslabón 2	x=-0.0405	y=0	z=-0.00021792
Eslabón 3	x=-0.0278	y=0	z=1.106e-05
Eslabón 4	x=-0.01741176	y=-6.2e-06	z=-6.606e-05

Tabla 3.4: Centro de masas de los eslabones

Fuente: Autor.

Masa [Kg]	
Eslabón 1	0.045
Eslabón 2	0.2044
Eslabón 3	0.09
Eslabón 4	0.1022

Tabla 3.5: Masas de los eslabones

Fuente: Autor.

Una vez registrados los parámetros mencionados anteriormente, es posible aplicar el algoritmo recursivo de Lagrange-Euler. La ecuación del modelo dinámico se expresa en la ecuación 3.28.

$$\tau = D(q)\ddot{q} + H(q, \dot{q}) + C(q) + F_v\dot{q} \tag{3.28}$$

Con:

τ : Matriz de fuerzas y pares.

D : Matriz de inercias.

H : Vector columna de fuerzas de coriolis y centrifugas.

C: Vector columna de fuerzas de gravedad.

F_v: Matriz del coeficiente de fricción del robot.

q: Vector de coordenadas articulares.

Para la obtención de las matrices y vectores requeridos para el análisis del modelo dinámico del robot, se ha desarrollado un algoritmo computacional a partir de las formulaciones de Lagrange-Euler presentadas en [6]. Esto facilito considerablemente el desarrollo del modelo dinámico directo e inverso del robot de estudio. Ver anexos, sección 6.2.

3.2.3 Control cinemático

La premisa del control de un robot atiende al desarrollo de estrategias que redunden en una mejor calidad sus movimientos. En ese sentido, el control cinemático es el responsable de establecer las trayectorias para cada articulación como funciones a lo largo del tiempo. Con frecuencia es necesario seleccionar trayectorias que contemplen restricciones físicas y criterios de calidad para ser muestreadas con un periodo de tiempo a definir por el usuario, esto generara como resultado un vector de referencias con las coordenadas articulares para el control dinámico del robot.

Para simplificar esta exposición, el presente estudio se limita a abordar desde la conversión de la especificación de la trayectoria dada por el usuario, hasta la interpolación de los puntos articulares obtenidos. Dada estas restricciones, se examinarán algunas trayectorias espaciales que le permitirán al robot moverse desde un punto inicial a un punto final.

- **Trayectorias punto a punto.** Cada una de las articulaciones del robot evoluciona desde su posición inicial a la final sin considerar la evolución de las otras articulaciones. Se distinguen dos casos: movimiento eje a eje y movimiento simultáneo de ejes.
- **Trayectorias coordinadas o isócronas.** Todas las articulaciones son coordinadas para empezar y acabar su movimiento al mismo tiempo, ajustándose a la articulación más lenta del sistema.

- **Trayectorias continuas.** La trayectoria que sigue el extremo del robot es conocida por el usuario, precisando conocer de forma constante las trayectorias articulares [6].

En este libro se emplea el uso de interpoladores cúbicos con una trayectoria continua para el control cinemático del robot de estudio. Esto se debe a la capacidad de evitar discontinuidades de velocidad durante el paso por varios puntos al usar un polinomio de grado 3 que tiene como función unir cada pareja de puntos próximos entre sí. Esta expresión está dada por:

$$q(t) = a + b(t - t^i) + c(t - t^i)^2 + d(t - t^i)^3 \tag{3.29}$$

$$t^i < t < t^{i+1} \tag{3.30}$$

Para $t = t^i$

$$q(t^i) = a + b(t^i - t^i) + c(t^i - t^i)^2 + d(t^i - t^i)^3 \tag{3.31}$$

$$a = q^i \tag{3.32}$$

Derivando la ecuación (3.29)

$$\dot{q}(t) = b + 2c(t - t^i) + 3d(t - t^i)^2 \tag{3.33}$$

Para $t = t^i$

$$\dot{q}(t) = b + 2c(t^i - t^i) + 3d(t^i - t^i)^2 \tag{3.34}$$

$$b = \dot{q}^i \tag{3.35}$$

Suponga $T = t^{i+1} - t^i$ y $t = t^{i+1}$para la ecuación (3.29)

$$q(t^{i+1}) = a + b(T) + c(T)^2 + d(T)^3 \tag{3.36}$$

Reemplazando a y b se obtiene

$$q(t^{i+1}) = q^i + \dot{q}^i(T) + c(T)^2 + d(T)^3 \tag{3.37}$$

Derivando la ecuación (3.36)

$$\dot{q}(t^{i+1}) = \dot{q}^i + 2c(T) + 3d(T)^2 \tag{3.38}$$

Despejando c de la ecuación (3.36)

$$c(T)^2 = q^{i+1} - q^i - \dot{q}^i(T) - d(T)^3 \tag{3.39}$$

$$c(T)^2 = q^{i+1} - q^i - \dot{q}^i(T) - (T)^3 \left(\frac{\dot{q}^{i+1} - \dot{q}^i - 2c(T)}{3(T)^2} \right) \tag{3.40}$$

$$-\frac{2}{3}c(T)^2 + c(T)^2 = q^{i+1} - q^i - \dot{q}^i(T) - \left(\frac{\dot{q}^{i+1}}{3} - \frac{\dot{q}^i}{3} \right)(T) \tag{3.41}$$

$$\frac{1}{3}c(T)^2 = q^{i+1} - q^i - \dot{q}^i(T) - \frac{\dot{q}^{i+1}}{3}(T) + \frac{\dot{q}^i}{3}(T) \tag{3.42}$$

$$c = \frac{3q^{i+1}}{T^2} - \frac{3q^i}{T^2} - \frac{3\dot{q}^i}{T} - \frac{\dot{q}^{i+1}}{T} + \frac{\dot{q}^i}{T} \tag{3.43}$$

$$c = \frac{3}{T^2}\left(q^{i+1} - q^i\right) - \frac{1}{T}\left(\dot{q}^{i+1} + 2\dot{q}^i\right) \tag{3.44}$$

Reemplanzando a, b y c en (3.36) para despejar d

$$d = \frac{q^{i+1}}{T^3} - \frac{q^i}{T^3} - \frac{\dot{q}^i(T)}{T^3} - \left[\frac{3}{T^2}\left(q^{i+1} - q^i\right) - \frac{1}{T}\left(\dot{q}^{i+1} + 2\dot{q}^i\right) \right](T)^2 \tag{3.45}$$

$$d = \frac{1}{T^3}\left(q^{i+1} - q^i\right) - \frac{\dot{q}^i}{T^2} - \frac{3}{T^3}\left(q^{i+1} - q^i\right) + \frac{1}{T^2}\left(\dot{q}^{i+1} + 2\dot{q}^i\right) \tag{3.46}$$

$$d = -\frac{2}{T^3}\left(q^{i+1} - q^i\right) + \frac{1}{T^2}\left(\dot{q}^{i+1} + \dot{q}^i\right) \tag{3.47}$$

Finalmente, estos cuatro coeficientes permiten imponer cuatro condiciones de contorno; dos de posición y dos de velocidad. Para la visualización de ejemplos de la propuesta de control cinemático desarrollada en el estudio, referirse a la sección 4.3.4 "Simulación empleando interpoladores cúbicos."

En la Figura.3.15 se presenta un diagrama descriptivo con las etapas principales que dan lugar a la propuesta de control cinemático. Ver anexos, sección 6.3.

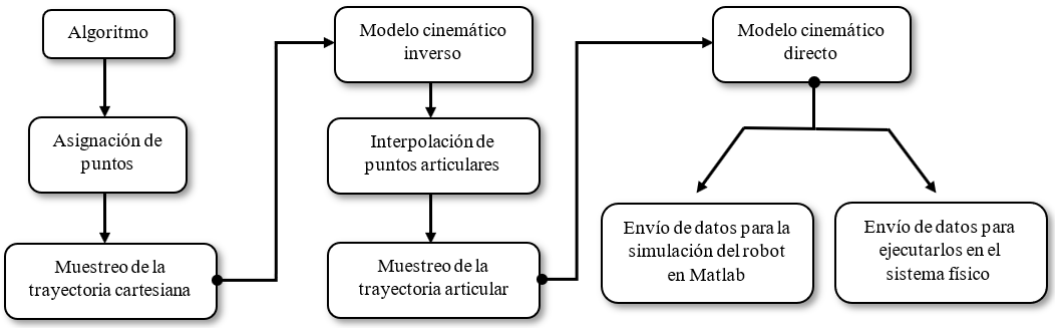

Figura 3.15: Funcionamiento del control cinemático
Fuente: Autor

3.3 Fase II: Simulaciones e interfaces de visualización

Se ha creado, como parte de este trabajo, una serie de simulaciones en Matlab de gran utilidad para la validación de los modelos matemáticos del robot desarrollados en la Fase I. Las simulaciones fueron elaboradas con base en archivos de estereolitografía correspondientes al modelo CAD del brazo robot. En la Figura.3.16 se aprecian las distintas configuraciones del prototipo para distintas aplicaciones. Ver anexos, sección 6.1.

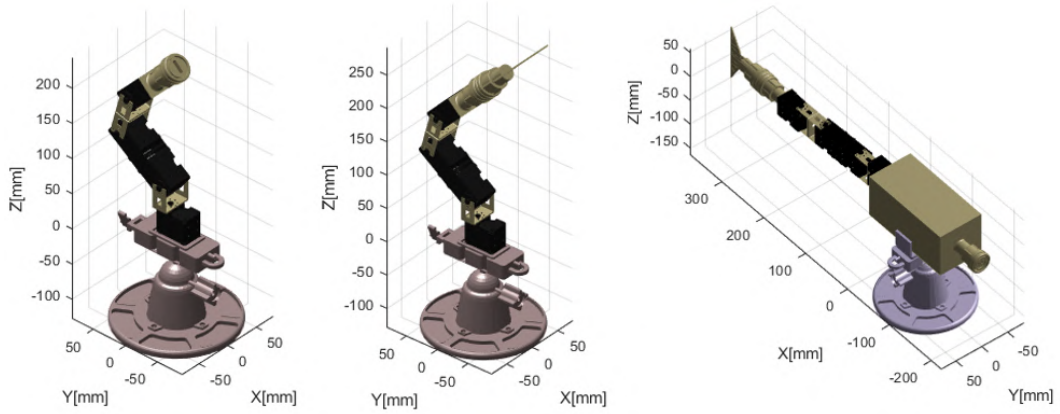

Figura 3.16: Simulaciones de diferentes configuraciones del robot de estudio
Fuente: Autor

3.3.1 Simulación de la cinemática directa

Implementando los códigos de programación correspondientes a la cinemática directa del robot, se desarrolló una animación que permite enviar valores a las articulaciones para simular un aleteo. La Figura.3.17 ilustra el movimiento de subida partiendo de una posición de reposo a una posición final dada por las coordenadas articulares expuestas. Por otro lado, en la Figura.3.18, se presenta el movimiento de bajada de la herramienta experimental partiendo de la última posición registrada en el movimiento de subida.

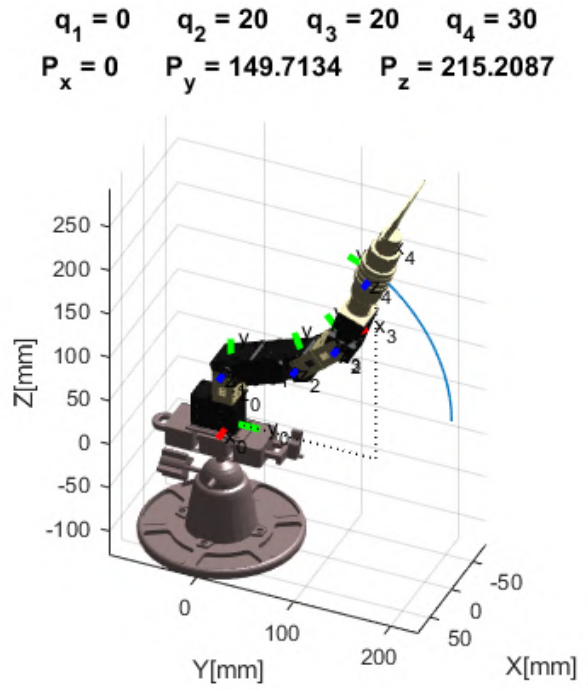

Figura 3.17: Movimiento de subida empleando una herramienta experimental
Fuente: Autor

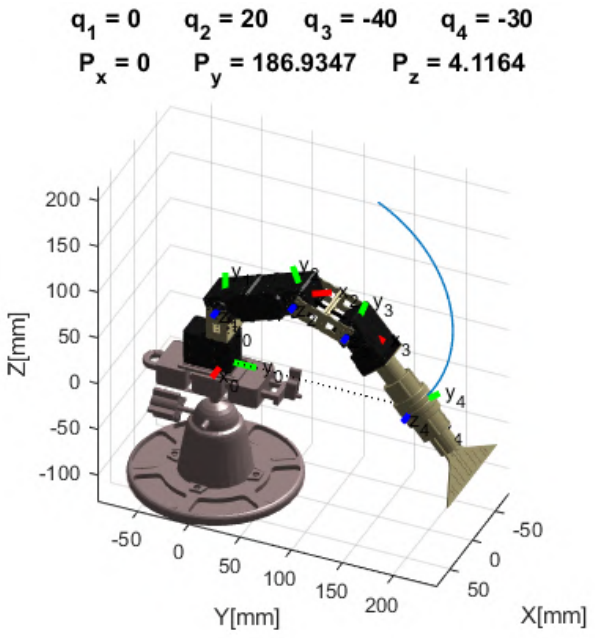

Figura 3.18: Movimiento de bajada empleando una herramienta experimental
Fuente: Autor

Código 3.2: Algoritmo para la animación del robot empleando una herramienta experimental

```
clear, clc, close all
f1 = figure(1); set(f1,'Color',[1, 1, 1]);
% ----------------------------------------------------------------------
% Primera configuración(ángulos)
% ----------------------------------------------------------------------
a=20; b=20; c=30;
% ----------------------------------------------------------------------
% Cantidad de pulsos
% ----------------------------------------------------------------------
Pulsos = 100;
% ----------------------------------------------------------------------
% Ángulos iniciales y finales
% ----------------------------------------------------------------------
q1_ini1 = 0; q1_fin1 = 0;
q2_ini1 = 0; q2_fin1 = b*pi/180;
q3_ini1 = 0; q3_fin1 = a*pi/180;
q4_ini1 = 0; q4_fin1 = c*pi/180;

M(:,1) = linspace(q1_ini1, q1_fin1,Pulsos)';
M(:,2) = linspace(q2_ini1, q2_fin1,Pulsos)';
M(:,3) = linspace(q3_ini1, q3_fin1,Pulsos)';
M(:,4) = linspace(q4_ini1, q4_fin1,Pulsos)';

% ----------------------------------------------------------------------
% Segunda configuración(ángulos)
% ----------------------------------------------------------------------
d=60; e=20; f=30;
% ----------------------------------------------------------------------
% Ángulos iniciales y finales
% ----------------------------------------------------------------------
q1_ini2 = 0; q1_fin2 = 0;
q2_ini2 = b*pi/180; q2_fin2 = b*pi/180;
q3_ini2 = a*pi/180; q3_fin2 = (a*pi/180)-d*pi/180;
q4_ini2 = c*pi/180; q4_fin2 = (c*pi/180)-d*pi/180;

M1(:,1) = linspace(q1_ini2, q1_fin2,Pulsos)';
M1(:,2) = linspace(q2_ini2, q2_fin2,Pulsos)';
M1(:,3) = linspace(q3_ini2, q3_fin2,Pulsos)';
M1(:,4) = linspace(q4_ini2, q4_fin2,Pulsos)';

% ----------------------------------------------------------------------
% Envío de parámetros a la cinemática directa
% ----------------------------------------------------------------------
for i=1:Pulsos
    cla
    q = M(i,:);
    A04=CD_Robot(q);
% ----------------------------------------------------------------------
% Mostrar coordenadas Px, Py y Pz
% ----------------------------------------------------------------------
    line([0, A04(1,4)], [0, 0], [0, 0],'Color', 'k' ,'LineWidth',1, 'LineStyle',':');
    line([A04(1,4), A04(1,4)], [0, A04(2,4)], [0, 0],'Color', 'k' ,'LineWidth',1, 'LineStyle',':');
    line([A04(1,4), A04(1,4)], [A04(2,4), A04(2,4)], [0, A04(3,4)],'Color', 'k' ,'LineWidth',1, 'LineStyle',':')
        ↪ ;
    A04(1,4)=round(A04(1,4),3);
% ----------------------------------------------------------------------
% Dibujar trayectoria
% ----------------------------------------------------------------------
    tray(i,:) = A04(1:3,4)';
    line(tray(:,1),tray(:,2),tray(:,3),'LineWidth',1);
    pause(0.0001);
end

```

```
% ------------------------------------------------------------
% Envío de parámetros a la cinemática directa
% ------------------------------------------------------------
for i=1:Pulsos
    cla
    q = M1(i,:);
    A04=CD_Robot(q);
% ------------------------------------------------------------
% Mostrar coordenadas Px, Py y Pz
% ------------------------------------------------------------
    line([0, A04(1,4)], [0, 0], [0, 0],'Color', 'k' ,'LineWidth',1, 'LineStyle',':');
    line([A04(1,4), A04(1,4)], [0, A04(2,4)], [0, 0],'Color', 'k' ,'LineWidth',1, 'LineStyle',':');
    line([A04(1,4), A04(1,4)], [A04(2,4), A04(2,4)], [0, A04(3,4)],'Color', 'k' ,'LineWidth',1, '↩
        ↪ LineStyle',':');
    A04(1,4)=round(A04(1,4),3);
% ------------------------------------------------------------
% Dibujar trayectoria
% ------------------------------------------------------------
    tray1(i,:) = A04(1:3,4)';
    line(tray1(:,1),tray1(:,2),tray1(:,3),'LineWidth',1);
    pause(0.0001);
end
```

3.3.2 Simulación de la cinemática inversa

El algoritmo realizado para la animación de la cinemática inversa del robot se llevó a cabo de la misma manera que en la cinemática directa. Sin embargo, al momento de ingresar los parámetros de entrada, estos obedecen a un vector con las coordenadas de las componentes en X, Y y Z de un punto en el espacio, tal y como se muestra en la Figura. 3.19. Ver anexos, sección 6.1.

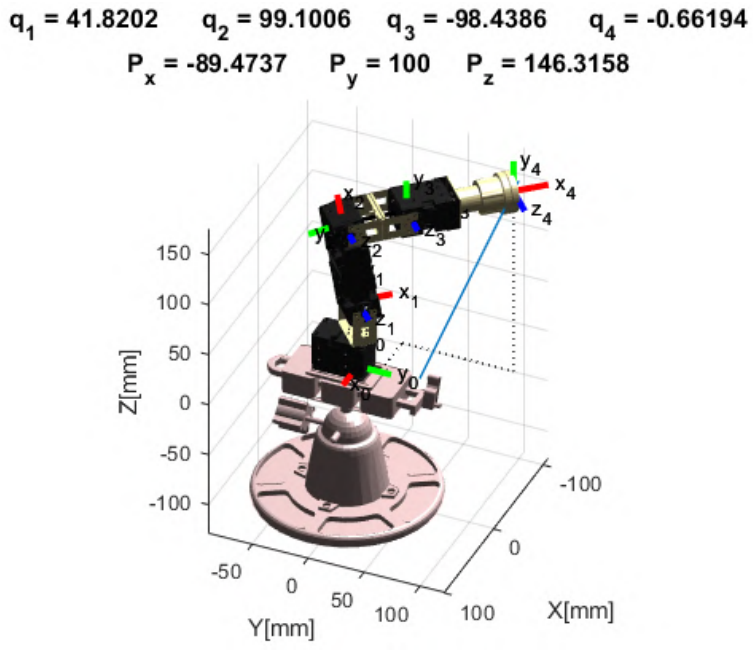

Figura 3.19: Animación de la cinemática inversa dado un punto inicial y un punto final
Fuente: Autor

Código 3.3: Algoritmo para la animación de la cinemática inversa dado un punto inicial y un punto final

```
clear all
clc
close all
f1 = figure(1); set(f1,'Color',[1, 1, 1]);
% ----------------------------------------------------------------------
% Asignación del punto inicial,final y pulsos
% ----------------------------------------------------------------------
Pf = [-100;100; 150];
Pi = [100; 100; 80];
Pulsos=20;

Pxf = linspace(Pi(1),Pf(1), Pulsos);
Pyf = linspace(Pi(2),Pf(2), Pulsos);
Pzf = linspace(Pi(3),Pf(3), Pulsos);
% ----------------------------------------------------------------------
% Envío de parámetros a la cinemática inversa
% ----------------------------------------------------------------------
for i=1:Pulsos
 cla;
 Px4 = Pxf(i); Py4 = Pyf(i); Pz4 = Pzf(i);
 Angle=0; CODO = -1;
 q=CI_Robot(Px4, Py4, Pz4,Angle,CODO);
% ----------------------------------------------------------------------
% Mostrar coordenadas Px, Py y Pz
% ----------------------------------------------------------------------
    line([0, Px4], [0, 0 ], [l1, l1],'Color', 'k' ,'LineWidth',1, 'LineStyle',':');
    line([Px4, Px4], [0, Py4], [l1, l1],'Color', 'k' ,'LineWidth',1, 'LineStyle',':');
    line([Px4, Px4], [Py4, Py4], [l1, Pz4],'Color', 'k' ,'LineWidth',1, 'LineStyle',':');
% ----------------------------------------------------------------------
% Dibujar trayectoria
% ----------------------------------------------------------------------
    line(Pxf(:),Pyf(:),Pzf(:),'LineWidth',1);
    pause(0.0001);

end
```

3.3.3 Simulación del modelo dinámico

Quizá el compromiso más grande en el diseño de un robot manipulador sea la determinación de los motores que le permitan cumplir los objetivos fijados por el usuario. Por consiguiente, se ha generado un algoritmo para simular al robot de estudio por medio del modelo dinámico planteado en la Fase I. De esta manera, es posible verificar si la selección de los motores cumple con los criterios para soportar los pares permisibles a los que estará expuesta la estructura. La validación se llevará a cabo partiendo del cálculo de los máximos pares. La Figura.3.20 ilustra el diagrama de simulación en la herramienta Simulink de Matlab [37].

En primer lugar, se debe destacar que la peor posición en la que se puede encontrar la primera articulación del robot se produce cuando todas las coordenadas articulares adoptan un valor de cero y la gravedad estará en el mismo sentido que la componente X_0. En la Figura.3.21 se presenta la configuración del robot de estudio con la posición más exigente para la primera articulación.

Por otro lado, en la Figura.3.23, se presentan los pares resultantes de las articulaciones que conforman el robot para la configuración ilustrada en la Figura.3.21. Es necesario aplicar un perfil de velocidad trapezoidal a cada una de las articulaciones para obtener el par pico y el

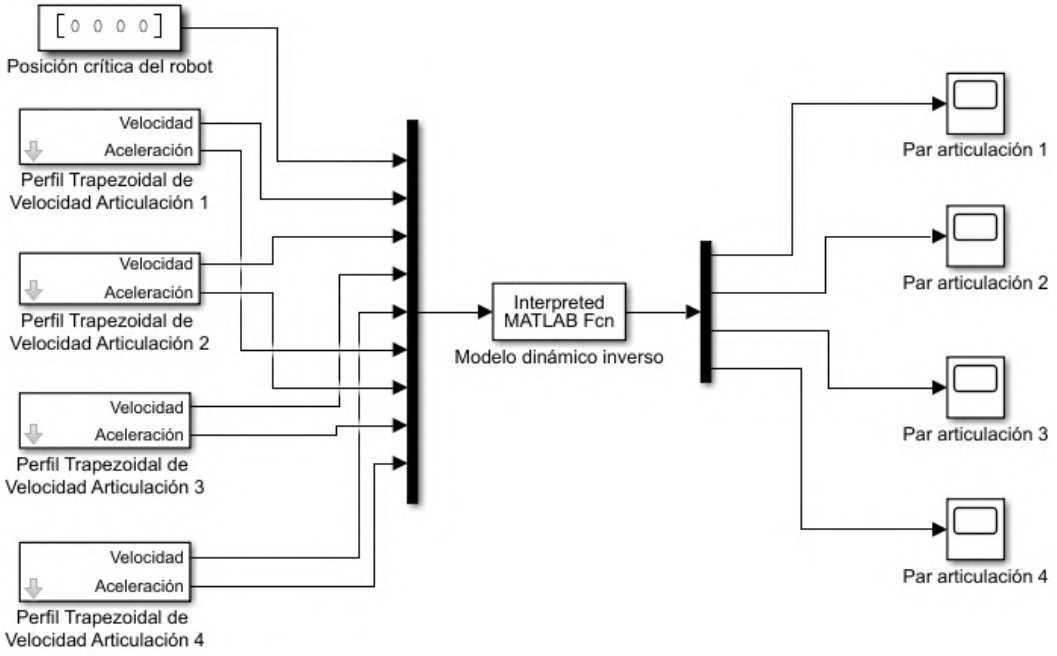

Figura 3.20: Esquema en Simulink para el cálculo de los máximos pares en la primera posición crítica
Fuente: Autor

par nominal de cada motor a través del modelo dinámico inverso. Este perfil se caracteriza por un intervalo de aceleración, mantenimiento y deceleración. Para el presente estudio, se utilizaron los tiempos de 0.1 Seg, 0.4 Seg y 0.1 Seg para cada intervalo respectivamente. Además, se va a considerar que la velocidad máxima para las articulaciones es de $\frac{\pi}{3}$ rad/s.

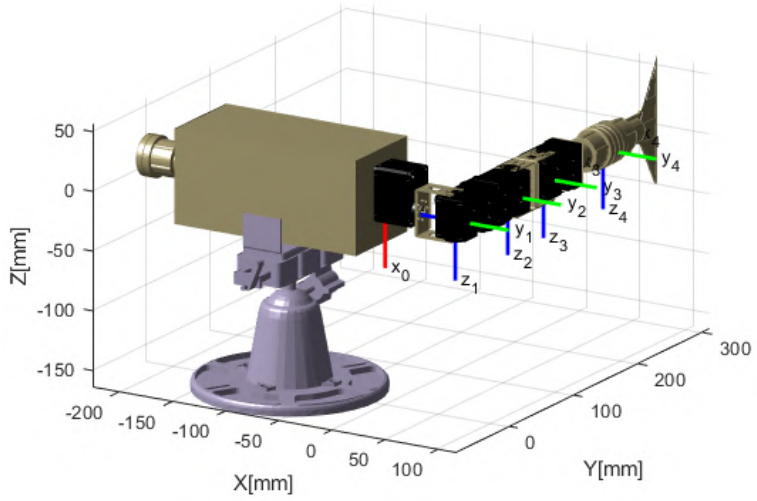

Figura 3.21: Configuración seleccionada para el estudio de la primera articulación
Fuente: Autor

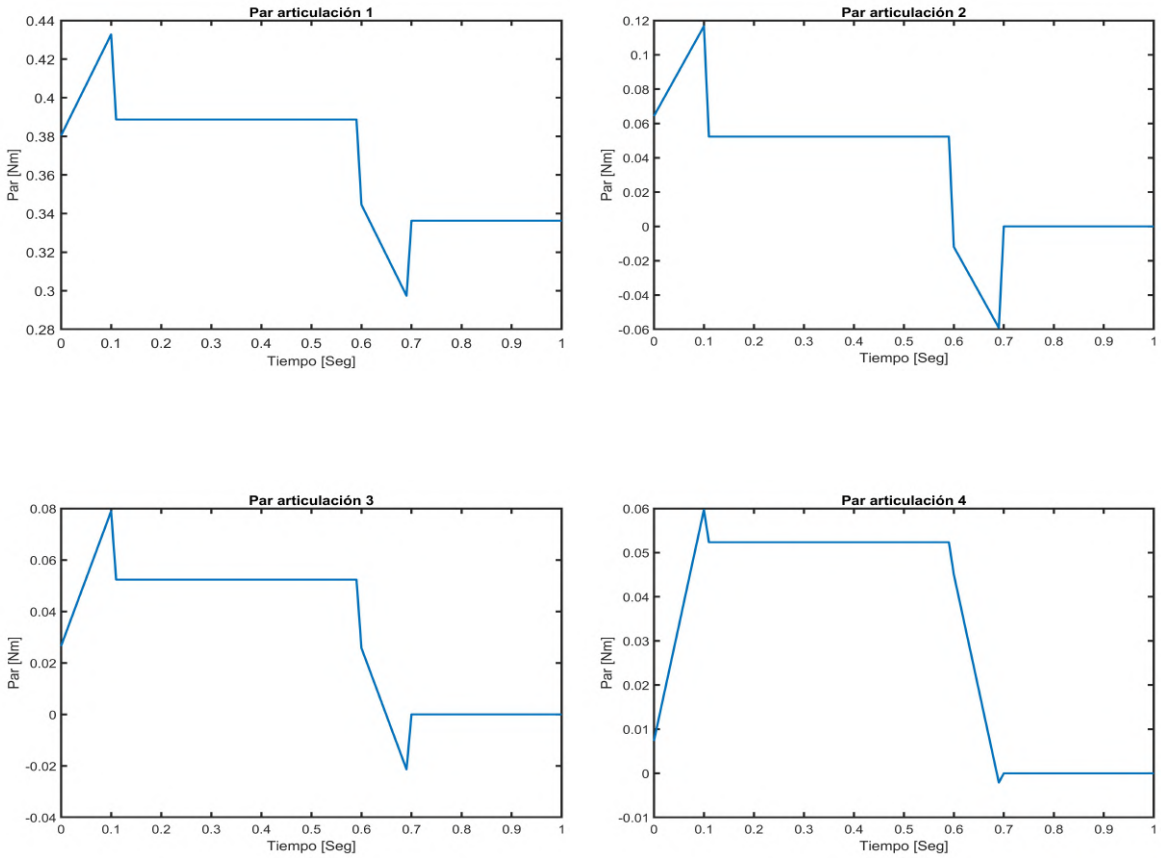

Figura 3.23: Gráficas correspondientes al par de las articulaciones 1, 2, 3 y 4

Es debido mencionar que el par pico corresponde al par máximo en valor absoluto que se ilustra en cada una de las gráficas de la Figura.3.23. De manera consecuente, se procede a calcular el vector de fuerzas y pares ejercidos en el extremo del robot expresado en el sistema de coordenadas de la base.

$$\tau_m = \tau_a + \tau_b \tag{3.48}$$

Donde:

τ_m : Par de selección del motor.

τ_a : Par requerido por la articulación para mover el robot.

τ_b : Fuerzas y pares de propulsión

De la Figura.3.23 se extraen los datos (3.49), (3.50), (3.51) y (3.52)

$$\tau_{a1} = 0.435[N \cdot m] \tag{3.49}$$

$$\tau_{a2} = 0.118[N \cdot m] \tag{3.50}$$

$$\tau_{a3} = 0.08[N \cdot m] \tag{3.51}$$

$$\tau_{a4} = 0.06[N \cdot m] \tag{3.52}$$

Seguidamente, se asigna un factor de seguridad igual a 2.0 para considerar las distintas fuerzas de rozamiento implicadas en el movimiento y las condiciones de diseño del prototipo. El par de selección del motor fue extraído de la hoja de características de los actuadores con los que fue elaborado el robot.

$$\tau_{a1} = 0.435[N \cdot m] * (2.0) = 0.87[N \cdot m] \tag{3.53}$$

$$1.4[N \cdot m] = 0.87[N \cdot m] + \tau_{b1} \tag{3.54}$$

$$\tau_{b1} = 0.53[N \cdot m] \tag{3.55}$$

$$\tau_{a2} = 0.118[N \cdot m] * (2.0) = 0.236[N \cdot m] \tag{3.56}$$

$$1.4[N \cdot m] = 0.236[N \cdot m] + \tau_{b2} \tag{3.57}$$

$$\tau_{b2} = 1.164[N \cdot m] \tag{3.58}$$

$$\tau_{a3} = 0.08[N \cdot m] * (2.0) = 0.16[N \cdot m] \tag{3.59}$$

$$1.4[N \cdot m] = 0.16[N \cdot m] + \tau_{b3} \tag{3.60}$$

$$\tau_{b3} = 1.24[N \cdot m] \tag{3.61}$$

$$\tau_{a4} = 0.06[N \cdot m] * (2.0) = 0.12[N \cdot m] \tag{3.62}$$

$$1.4[N \cdot m] = 0.12[N \cdot m] + \tau_{b4} \tag{3.63}$$

$$\tau_{b4} = 1.28[N \cdot m] \tag{3.64}$$

Posteriormente, se despeja y calcula τ_b para las ecuaciones (3.54), (3.57), (3.60) y (3.63) para generar el vector τ. Nótese que la ecuación (3.67) se origina debido a que el par efectivo será igual a la diferencia entre el par que ejerce el motor y los pares que consume el robot para el ejecutar un movimiento.

$$\tau = [\tau_{m1}; \tau_{m2}; \tau_{m3}; \tau_{m4}] \tag{3.65}$$

$$\tau = [\tau_{a1} + \tau_{b1}; \tau_{a2} + \tau_{b2}; \tau_{a3} + \tau_{b3}; \tau_{a4} + \tau_{b4}] \tag{3.66}$$

$$\tau = [0.53; 1.164; 1.24; 1.28] \tag{3.67}$$

En la ecuación (3.68) se presentan los componentes del vector de fuerzas y pares ejercidos en el extremo del robot. Para el análisis matemático, considere la ecuación (3.69). Esta puede ser reescrita como se presenta en la ecuación (3.70). Donde J representa la matriz Jacobiana. Ver anexos, sección 6.2.

$$T = [F_{ex}; F_{ey}; F_{ez}; M_{ex}; M_{ey}; M_{ez}] \tag{3.68}$$

$$Potencia = Par \cdot Velocidad \tag{3.69}$$

$$T' * J = \tau' \tag{3.70}$$

Multiplicando a ambos lados de la ecuación (3.70) por el Jacobiano transpuesto inverso, se obtiene la ecuación (3.72).

$$T' * J * J_{d^-} = \tau' * J_{d^-} \tag{3.71}$$

$$T' = \tau' * J_{d^-} \tag{3.72}$$

La matriz Jacobiana generada en el estudio tiene una dimensión de 6x4. Por tanto, se propone implementar la matriz pseudoinversa por izquierda puesto que se presenta un mayor número de filas que de columnas. La ecuación (3.73) describe dicho procedimiento.

$$J_{d^-} = inv(J' * J) * J' \tag{3.73}$$

Una vez calculadas las variables τ' y J_{d^-}, se procede a realizar el cálculo de como se describe a continuación:

$$T' = [0.53; 1.164; 1.24; 1.28]' * \begin{bmatrix} -0.1707 & 0 & 0 & 0 & 0 & 0.9700 \\ 0 & 0 & 10.1250 & 2.0000 & 0 & 0 \\ 0 & 0 & -6.5000 & -8.0000 & 0 & 0 \\ 0 & 0 & -2.8750 & 4.0000 & 0 & 0 \end{bmatrix} \tag{3.74}$$

$$T' = [-0.0905, 0.0000, 0.0455, -2.4720, -0.0000, 0.5141] \tag{3.75}$$

Por último, se lleva a cabo el mismo procedimiento para el análisis de las articulaciones 2, 3 y 4. En este caso, la posición crítica del robot es la descrita en la Figura.3.24 e ilustrada en la Figura.3.27.

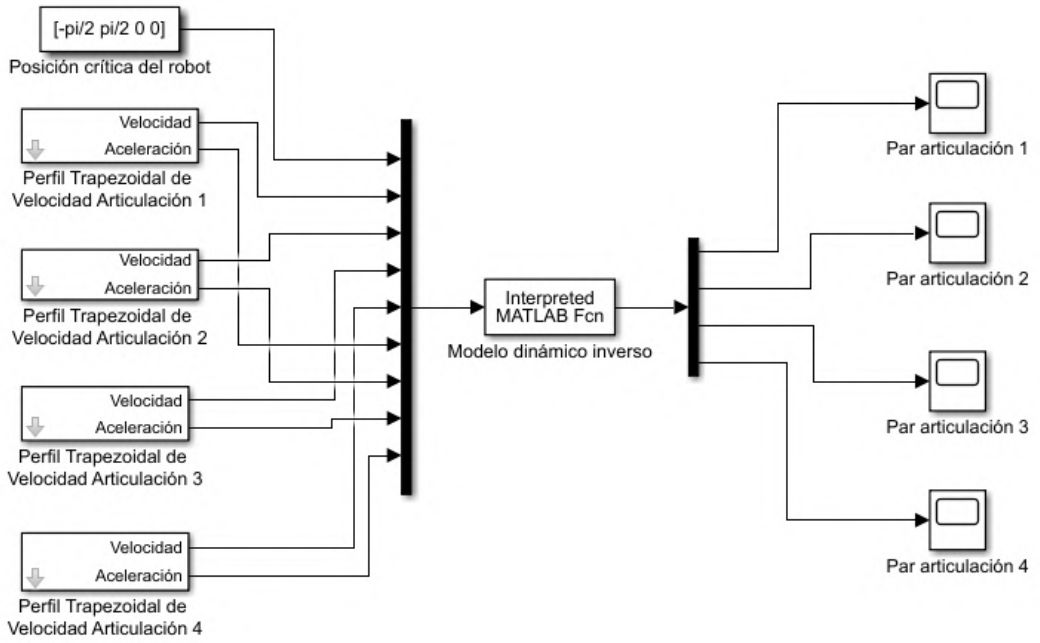

Figura 3.24: Esquema en Simulink para el cálculo de los máximos pares en la segunda posición crítica

Fuente: Autor

En la Figura.3.26 se presentan los pares de las articulaciones 1, 2, 3 y 4 atendiendo a un vector de coordenadas articulares [-pi/2, pi/2, 0, 0]. La gravedad se asignó de la misma manera que en el análisis de la posición más exigente para la primera articulación.

A continuación, se presenta el análisis matemático para la segunda configuración crítica del prototipo:

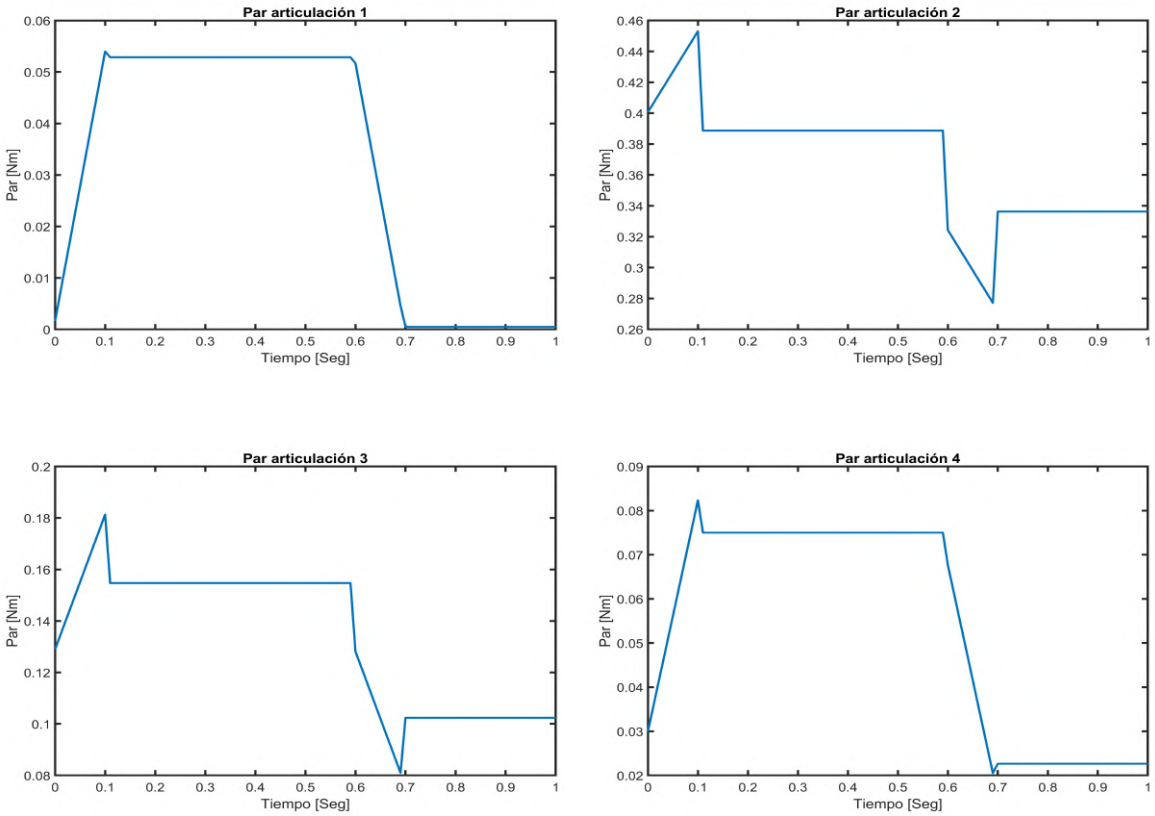

Figura 3.26: Gráficas correspondientes al par de las articulaciones 1, 2, 3 y 4 para la segunda configuración propuesta

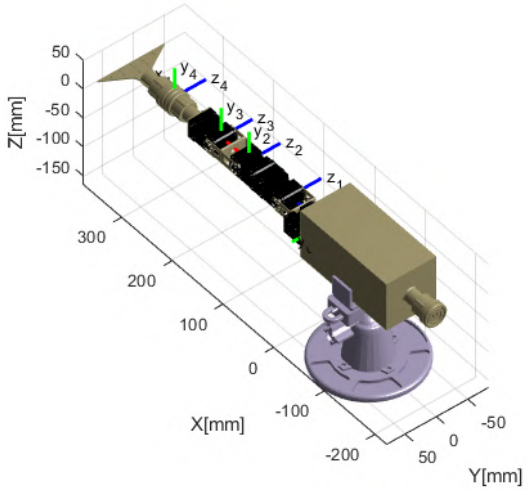

Figura 3.27: Configuración seleccionada para el estudio de la articulación 2, 3 y 4
Fuente: Autor

Variables τ_{a1}, τ_{a2}, τ_{a3} y τ_{a4}

$$\tau_{a1} = 0.054[N \cdot m] \tag{3.76}$$

$$\tau_{a2} = 0.455[N \cdot m] \tag{3.77}$$

$$\tau_{a3} = 0.182[N \cdot m] \tag{3.78}$$

$$\tau_{a4} = 0.083[N \cdot m] \tag{3.79}$$

Despeje de las variables τ_{b1}, τ_{b2}, τ_{b3} y τ_{b4}

$$\tau_{a1} = 0.054[N \cdot m] * (2.0) = 0,108[N \cdot m] \tag{3.80}$$

$$1.4[N \cdot m] = 0,108[N \cdot m] + \tau_{b1} \tag{3.81}$$

$$\tau_{b1} = 1,292[N \cdot m] \tag{3.82}$$

$$\tau_{a2} = 0.455[N \cdot m] * (2.0) = 0.91[N \cdot m] \tag{3.83}$$

$$1.4[N \cdot m] = 0.91[N \cdot m] + \tau_{b2} \tag{3.84}$$

$$\tau_{b2} = 0.49[N \cdot m] \tag{3.85}$$

$$\tau_{a3} = 0.182[N \cdot m] * (2.0) = 0.364[N \cdot m] \tag{3.86}$$

$$1.4[N \cdot m] = 0.364[N \cdot m] + \tau_{b3} \tag{3.87}$$

$$\tau_{b3} = 1.036[N \cdot m] \tag{3.88}$$

$$\tau_{a4} = 0.083[N \cdot m] * (2.0) = 0.166[N \cdot m] \tag{3.89}$$

$$1.4[N \cdot m] = 0.166[N \cdot m] + \tau_{b4} \tag{3.90}$$

$$\tau_{b4} = 1.234[N \cdot m] \tag{3.91}$$

Vector τ

$$\tau = [1.292; 0.49; 1.036; 1.234] \tag{3.92}$$

Vector de fuerzas y pares resultantes para la segunda configuración.

$$T' = [-0.2206, 0.0000, -5.3205, -2.3720, -0.0000, 1.2532] \tag{3.93}$$

Con base en los pares obtenidos al aplicar el esquema de simulación para la selección de los motores, se procede a buscar en catálogos comerciales los parámetros de los motores que permitan satisfacer dichos requerimientos. Una vez concluido el procedimiento anterior, se ha considerado que los actuadores con los que cuenta el prototipo cumplen con las especificaciones dadas por el usuario.

3.3.4 Simulación empleando interpoladores cúbicos

Al analizar el desarrollo de la simulación del robot empleando interpoladores cúbicos, es preciso generar una función que permita obtener los coeficientes de los splines cúbicos que describen una trayectoria a tramos partiendo del número de puntos en instantes de tiempo específicos. De usarse un bajo número de puntos en los muestreos, el sistema tendrá como resultado una trayectoria que difiere en gran medida a la trayectoria fijada por el usuario. Por ejemplo, al examinar la Figura.3.29 es posible identificar que el proceso no garantiza la suavidad, velocidad y aceleración permisibles para obtener un resultado satisfactorio. Pese a esto, se puede apreciar como las trayectorias resultantes pasan por todos los puntos indicados con velocidad continua. Los ejes verticales de cada una de las gráficas corresponden a los valores de las coordenadas articulares en los instantes de tiempo asignados por el usuario, este último corresponde al eje horizontal.

No obstante, si aumenta el número de puntos para las etapas de muestreo, la diferencia entre la trayectoria objetivo y la trayectoria realizada por el robot disminuye de forma significativa. Al igual que en el ejemplo anterior, se ha realizado un nuevo procedimiento empleando un número de puntos igual a 20 para el primer muestreo. La Figura.3.31 ilustra los resultados de dicha configuración.

La Figura.3.32 presenta la validación gráfica de algunas posiciones de la tarea cinemática asignada. En este caso, el número de puntos es igual a 20 para el primer muestro y 10 para el segundo muestreo. La trayectoria seleccionada para corroborar el funcionamiento del interpolador está compuesta por tres puntos en el espacio.

Conviene especificar que no es necesario definir desde un principio las velocidades de paso para el cálculo de los coeficientes. Según la regla heurística, es posible establecer estas velocidades como la media de las pendientes de los tramos anterior y posterior (si presentan el

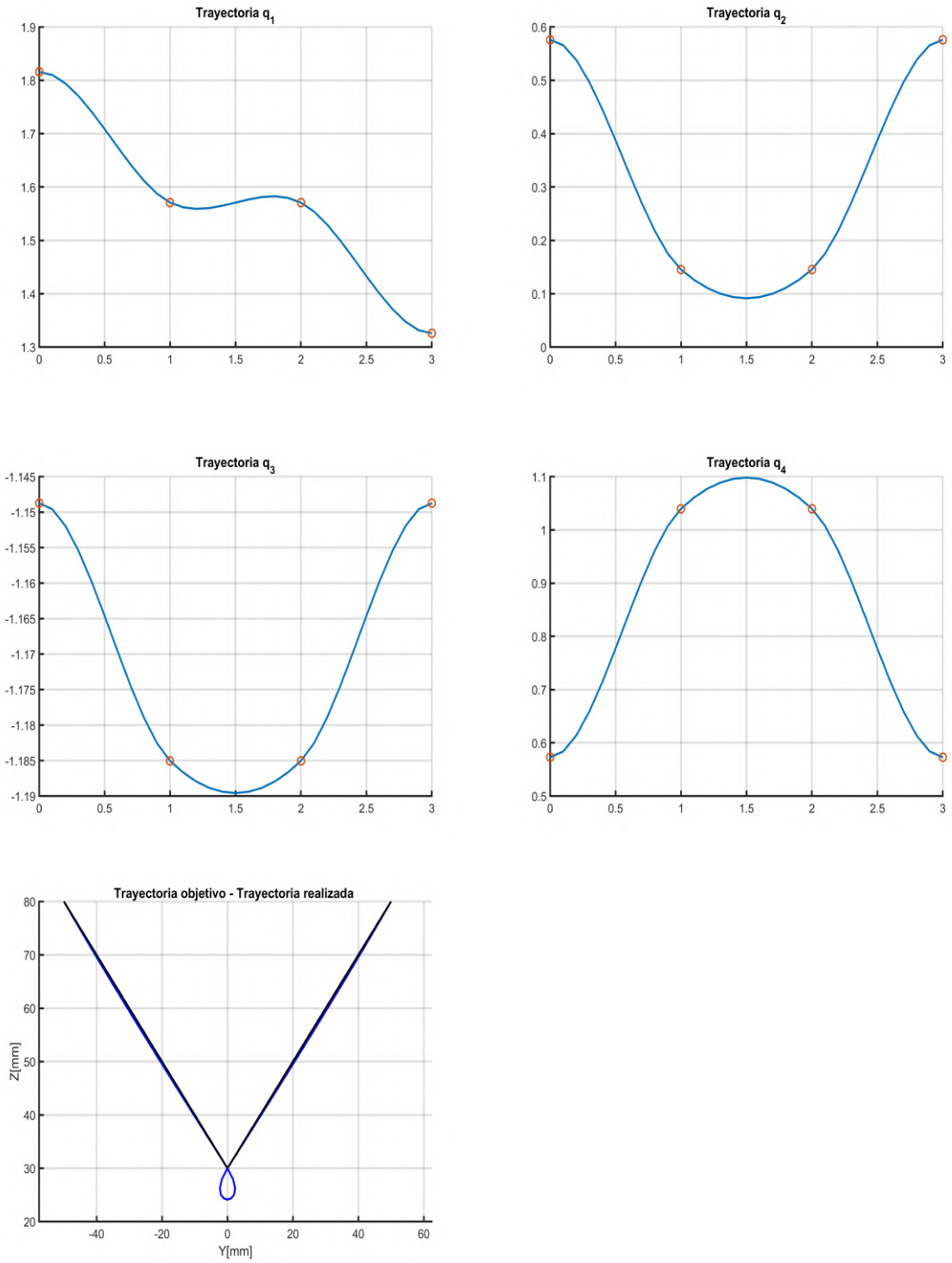

Figura 3.29: Trayectorias de las articulaciones del robot con pocos puntos de muestreo

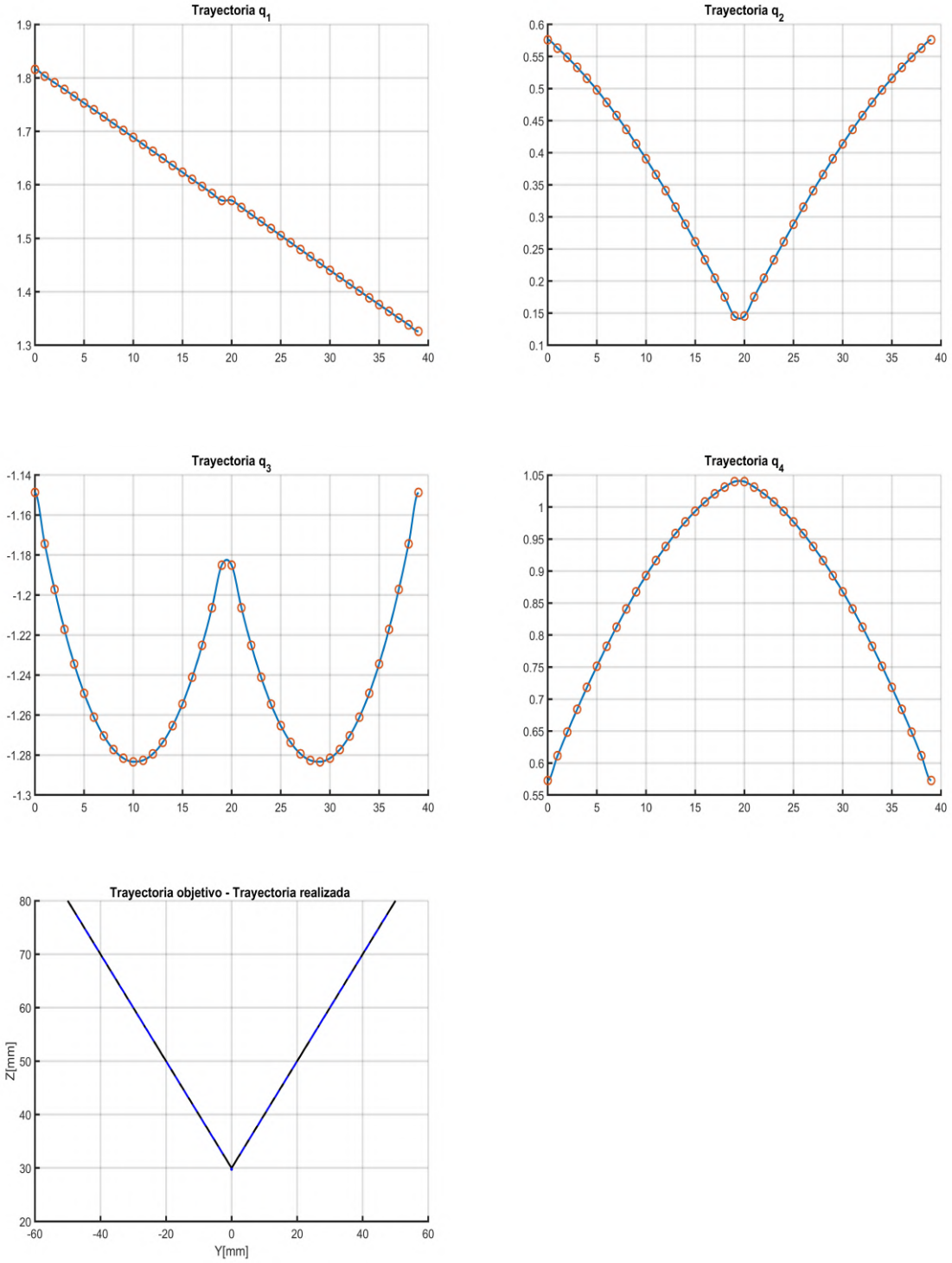

Figura 3.31: Trayectorias de las articulaciones del robot con más puntos de muestreo

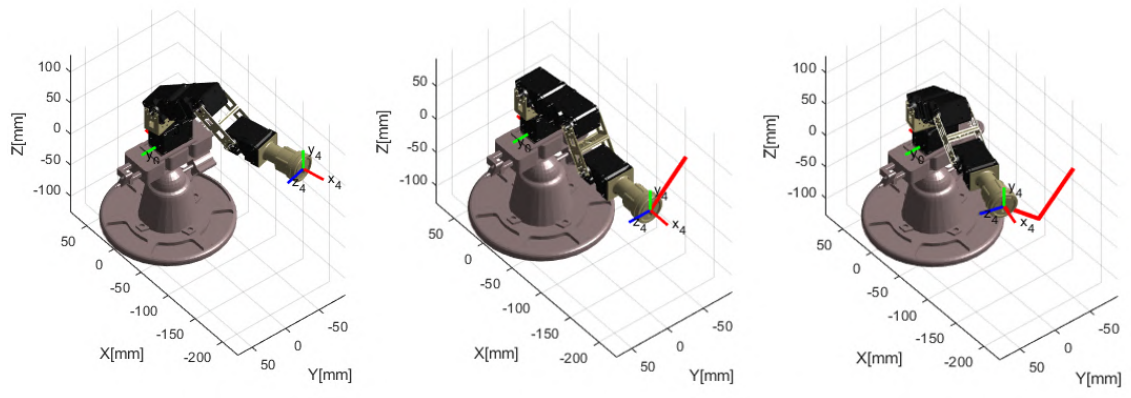

Figura 3.32: Validación gráfica del interpolador cúbico propuesto con una asignación de 20 puntos para el primer muestreo y 10 puntos para el segundo muestreo
Fuente: Autor

mismo signo), o en su defecto, asignarle un valor nulo igual a cero (en caso de producirse un cambio de signo).

Para concluir este apartado, se han generado distintas pruebas que permitan corroborar la relación entre el número de muestreos y los costos computacionales al momento de efectuar las simulaciones. En la Tabla.3.6 presenta los resultados de este procedimiento.

Asignación de puntos para el primer muestreo	**Asignación de puntos para el segundo muestreo**	**Tiempo [ms]**
2	10	3000
5	10	9000
10	10	20000
20	10	42000
30	10	64000
2	2	1000
5	2	2000
10	2	5000
20	2	11000
30	2	17000

Tabla 3.6: Pruebas de funcionamiento para el algoritmo del interpolador cúbico
Fuente: Autor

3.3.5 Algunas restricciones mecánicas

Después de realizar el estudio matemático del robot, resulta indispensable especificar ciertas restricciones mecánicas que impiden al prototipo moverse en una gran cantidad de ubicaciones en el espacio de trabajo. Según los estudios realizados por [38], es posible generar una

serie de ayudas dirigidas al usuario que sirvan de base para la ilustración de los movimientos permitidos por el brazo robot.

Aunque el espacio de trabajo corresponde a los puntos en los que puede acceder su efector final, no significa que este pueda ocupar cualquier tipo de orientación. Conforme a lo anterior, es necesario definir el acceso a la muñeca del robot.

En la Figura.3.33 se aprecia que el punto de la muñeca corresponde al tercer sistema de coordenadas. El centro de la esfera se ubica en el primer sistema de coordenadas y tiene como radio la distancia entre los sistemas 1 y 3 cuando la articulación 2 toma el valor de 90 grados, indicando el límite máximo (sin tener colisiones en cuenta) que puede adoptar dicha articulación dadas las geometrías de los eslabones 2 y 3 del robot.

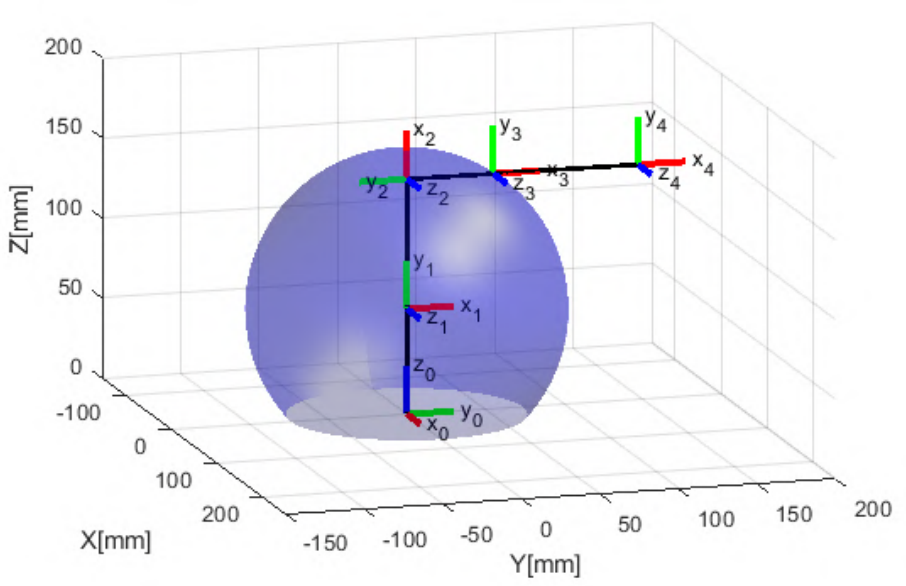

Figura 3.33: Límite interior del espacio de trabajo de la muñeca
Fuente: Autor

Código 3.4: Algoritmo para el límite interior del espacio de trabajo de la muñeca

```
clear all
clc
close all
f1 = figure(1), set(f1, 'Color', [1,1,1]), clf

% Algunos ángulos
a=90*pi/180;
b=-90*pi/180;
c=125*pi/180;

% Vector para los parámetros de las articulaciones
q=[0,a,b,0];

% Función de cinemática directa
```

```
A04=CD_Robot(q);
X04=A04(1:3,1);
VL=X04*(92);

% Esfera 1
r=97.90;
x0=0;
y0=0
z0=65.70;
[x,y,z]=sphere(50);
pe_0=[x0,y0,z0]';
surf((x+x0)+r*x,(y+y0)+r*y,(z+z0)+r*z,'FaceColor','b','FaceAlpha',0.3,'EdgeColor','none');
axis equal

hold on

% Ajuste del espacio de trabajo
view(56,12),camlight(56,40);, lighting phong;,grid on;
axis([ -150 250 -150 200 0 200]);
xlabel('X[mm]'); ylabel('Y[mm]');zlabel('Z[mm]');
```

Seguidamente, se ilustra el límite exterior del espacio de trabajo de la muñeca en donde se percibe que es coincidente cuando los eslabones 2 y 3 se encuentran alineados. Esta configuración es presentada en la Figura.3.34

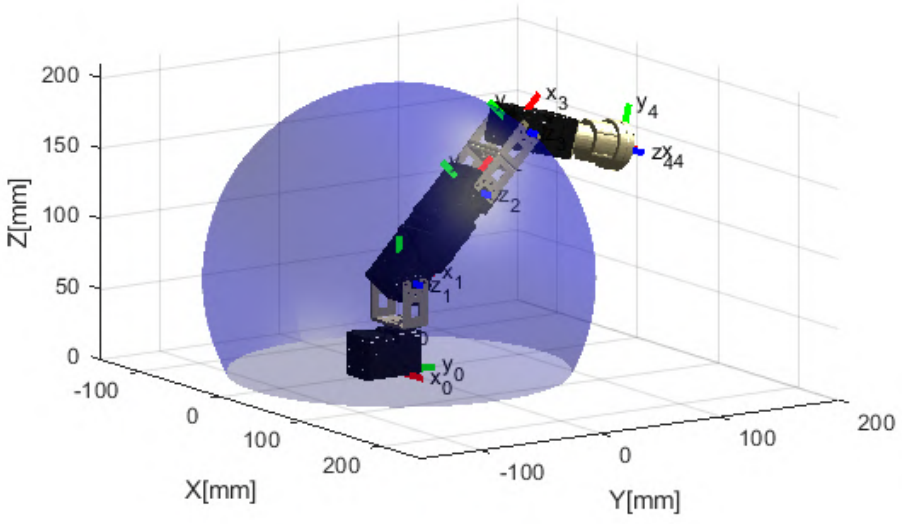

Figura 3.34: Límite exterior del espacio de trabajo de la muñeca
Fuente: Autor

Código 3.5: Algoritmo para el límite exterior del espacio de trabajo de la muñeca

```
clear all
clc
close all
f1 = figure(1), set(f1, 'Color', [1,1,1]), clf

```

```
% Algunos ángulos
a=45*pi/180;
b=60*pi/180;

% Vector para los parámetros de las articulaciones
q=[0,a,0,-b];

% Función de cinemática directa
A04=CD_Robot(q);
X04=A04(1:3,1);
VL=X04*(92);

% Esfera 1
r=136.0190;
x0=0;
y0=0
z0=65.70;
[x,y,z]=sphere(50);
pe_0=[x0,y0,z0]';
surf((x+x0)+r*x,(y+y0)+r*y,(z+z0)+r*z,'FaceColor','b','FaceAlpha',0.3,'EdgeColor','none');
axis equal

hold on

% Ajuste del espacio de trabajo
view(56,12),camlight(56,40);, lighting phong;,grid on;
axis([ -150 200 -150 200 0 210]);
xlabel('X[mm]'); ylabel('Y[mm]');zlabel('Z[mm]');
```

Posteriormente, se calcula la sección del espacio de trabajo del efector final. El análisis de esta sección guarda similitud al desarrollado para la muñeca, pero con un desplazamiento igual a la diferencia entre las posiciones de los sistemas de coordenadas 3 y 4, como se muestra en la Figura.3.35.

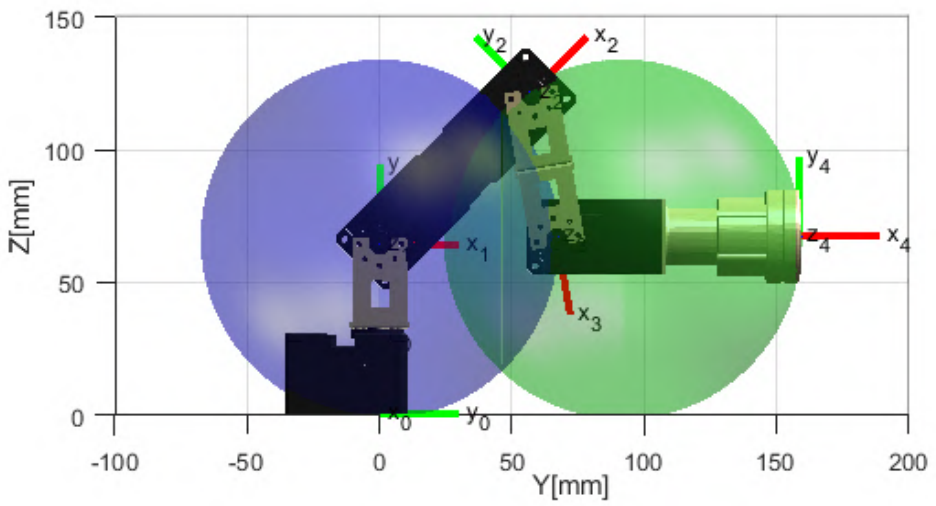

Figura 3.35: Límite interior del espacio de trabajo del efector final
Fuente: Autor

De acuerdo a lo anterior, se presenta el siguiente análisis para el cálculo del radio de la esfera; α será igual a la resta del ángulo llano menos el valor de la coordenada articular como se ilustra en la ecuación (3.94).

$$\alpha = 180° - 125° = 55° \tag{3.94}$$

$$a = 81[mm] \tag{3.95}$$

$$b = 55[mm] \tag{3.96}$$

Conociendo los valores de a, b y α, se procede a calcular el lado restante del triángulo a través de la ley de cosenos en la ecuación (3.97). Despejando el valor de c se obtiene:

$$c^2 = (81[mm])^2 + (55[mm])^2 - 2\,(81[mm])\,(55[mm])\,cos\,(55°) \tag{3.97}$$

$$c^2 = 4475.433[mm] \tag{3.98}$$

$$c = 66.8986[mm] \tag{3.99}$$

Código 3.6: Algoritmo para el límite interior del espacio de trabajo del efector final

```
clear all
clc
close all
f1 = figure(1), set(f1, 'Color', [1,1,1]), clf

% Algunos ángulos
a=45*pi/180;
b=80*pi/180;
c=125*pi/180;

% Vector para los parámetros de las articulaciones
q=[0,a,-c,b];

% Función de cinemática directa
A04=CD_Robot(q);
X04=A04(1:3,1);
VL=X04*(92);

% Esfera 1
r=66.8986;
x0=0;
y0=0
z0=65.70;
[x,y,z]=sphere(50);
pe_0=[x0,y0,z0]';
surf((x+x0)+r*x,(y+y0)+r*y,(z+z0)+r*z,'FaceColor','b','FaceAlpha',0.3,'EdgeColor','none');
axis equal

hold on

% Esfera 2
```

```
r1=66.8986;
pe_1=pe_0+VL;
x1=pe_1(1);
y1=pe_1(2);
z1=pe_1(3);
[xx,yy,zz]=sphere(50);
surf((xx+x1)+r1*xx,(yy+y1)+r1*yy,(zz+z1)+r1*zz,'FaceColor','g','FaceAlpha',0.3,'EdgeColor','none')
    ↪ ;


% Ajuste del espacio de trabajo
view(56,12),camlight(56,40);, lighting phong;,grid on;
axis([ -100 100 -100 200 0 150]);
xlabel('X[mm]'); ylabel('Y[mm]');zlabel('Z[mm]');
```

Finalmente, se presenta la relación entre el límite exterior del espacio de trabajo de la muñeca y el extremo del efector final del robot en la Figura.3.36. Conviene indicar que los espacios de trabajo expuestos en este apartado giran en torno al eje Z_0 de la primera articulación.

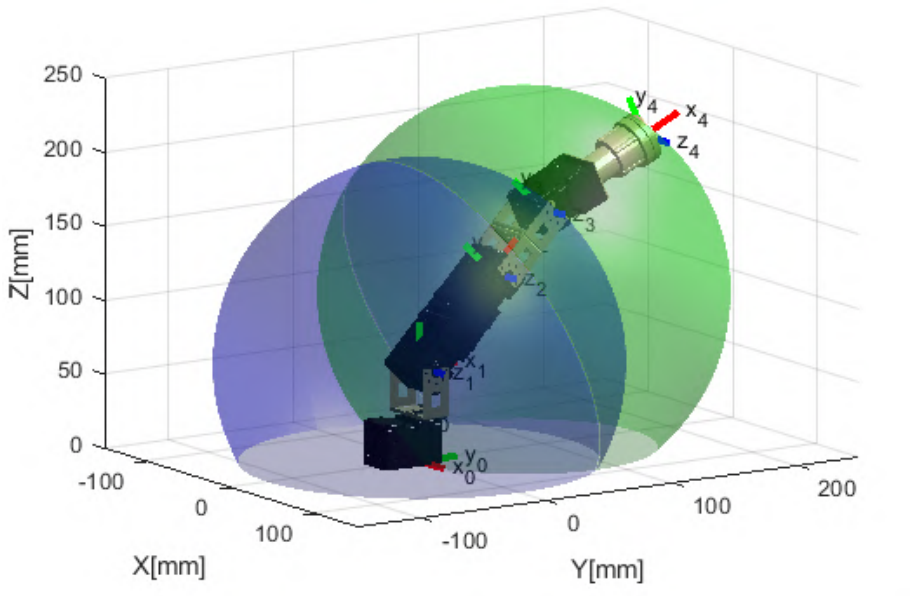

Figura 3.36: Límite exterior del espacio de trabajo del efector final
Fuente: Autor

Código 3.7: Algoritmo para el límite exterior del espacio de trabajo del efector final

```
clear all
clc
close all
f1 = figure(1), set(f1, 'Color', [1,1,1]), clf
```

```
5
6 % Algunos ángulos
7 a=45*pi/180;
8 b=20*pi/180;
9
10 % Vector para los parámetros de las articulaciones
11 q=[0,a,0,-b];
12
13 % Función de cinemática directa
14 A04=CD_Robot(q);
15 X04=A04(1:3,1);
16 VL=X04*(92);
17
18 % Esfera 1
19 r=136.0190;
20 x0=0;
21 y0=0
22 z0=65.70;
23 [x,y,z]=sphere(50);
24 pe_0=[x0,y0,z0]';
25 surf((x+x0)+r*x,(y+y0)+r*y,(z+z0)+r*z,'FaceColor','b','FaceAlpha',0.3,'EdgeColor','none');
26
27 hold on
28
29 % Esfera 2
30 r1=136.0190;
31 pe_1=pe_0+VL;
32 x1=pe_1(1);
33 y1=pe_1(2);
34 z1=pe_1(3);
35 [xx,yy,zz]=sphere(50);
36 surf((xx+x1)+r1*xx,(yy+y1)+r1*yy,(zz+z1)+r1*zz,'FaceColor','g','FaceAlpha',0.3,'EdgeColor','none')↩
      ↪ ;
37 axis equal
38
39 % Ajuste del espacio de trabajo
40 view(56,12),camlight(56,40);, lighting phong;,grid on;
41 axis([ -150 150 -150 250 0 250]);
42 xlabel('X[mm]'); ylabel('Y[mm]');zlabel('Z[mm]');
```

3.3.6 Interfaces de visualización de realidad aumentada

Para el desarrollo de los modelos 3D de las interfaces de realidad aumentada se utilizó el programa informático multiplataforma Blender. Este permite gestionar grupos de objetos y realizar jerarquías para definir objetos "padres" y objetos "hijos". La importancia del uso de jerarquías en este trabajo se justifica al momento de mover el objeto padre, pues el objeto hijo mantendrá su posición y orientación con respecto este último. De esta forma, es posible replicar el comportamiento del prototipo a través de una adecuada jerarquía de las piezas que lo conforman.

Por otro lado, Blender es compatible con Unity, por lo tanto, permite importar archivos sin procesos complicados, instalaciones adicionales o convertidores. Sin embargo, aspectos relacionados con los sistemas de coordenadas y el factor de escala serán distintos. Estos problemas han sido solucionados desde Blender mediante la rotación y el escalado del modelo 3D del prototipo. La Figura.3.37 presenta una visualización del prototipo configurado en el entorno de Blender.

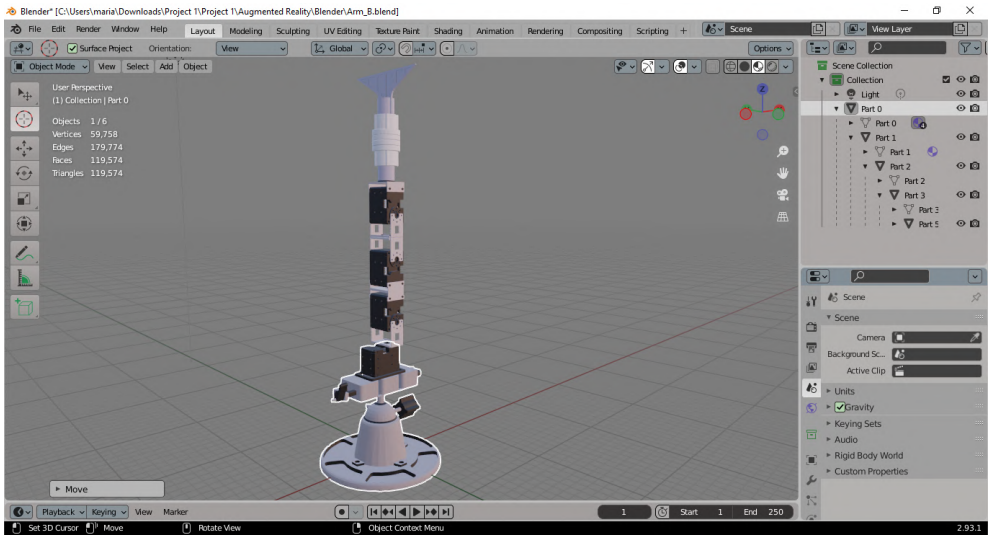

Figura 3.37: Prototipo configurado en Blender
Fuente: Autor

3.3.6.1 Aplicación para teléfonos inteligentes y cámaras web

Inicialmente, se generó un nuevo proyecto 3D en Unity con la configuración de *Vuforia Augmented Reality*. La versión de software utilizada en este estudio fue Unity 2018.4.36f1, esto debido a que posibilita integrar desde la instalación de Unity Vuforia Engine, evitando posibles inconvenientes en instalaciones externas de este componente en versiones más avanzadas. Seguidamente, se ha generado una licencia en el *License Manager* del *Vuforia Engine developer portal* para configurar el proyecto y posteriormente generar una *Image Target* que servirá como marcador. Al añadir los modelos realizados en Blender, estos conservarán su jerarquía de componentes en Unity, por lo que solo bastara con arrastrar el modelo a la *Image Target* para que este aparezca cuando la imagen enfoque el patrón establecido como marcador. Concluido esto, se desarrollan scripts para rotar cada una de las articulaciones del modelo, así como también se elaboran scripts para cambiar el factor de escala y mantener en rotación el modelo empleando funciones propias del programa. Por ejemplo: *Object.transform.localScale* y *objectRotate.transform.Rotate*. Se ha configurado un canvas para facilitar la interacción entre el usuario y la aplicación. Además, se han precisado varias escenas para distribuir de una mejor forma la información y el contenido virtual.

En este proyecto se contempla una aplicación compuesta por cuatro escenas en Unity. La primera escena cuenta con distintos paneles, textos, deslizadores y botones para la interacción con el modelo 3D del prototipo. El usuario podrá trabajar sobre la cinemática directa, cambiar el factor de escala, subir, bajar e incluso rotar el modelo. Por otro lado, la segunda escena dispone de la misma configuración que la primera escena, pero además es posible visibilizar el prototipo con la herramienta experimental y a su vez, permite reproducir dos videos correspondientes a la cinemática directa e inversa realizada en Matlab. Además, en la tercera escena, se presentan imágenes relacionadas con el mecanismo conector, el ensamble del robot y algunas configuraciones de sus articulaciones. Finalmente, en la escena número

cuatro, se presenta una fotografía del prototipo real con una breve descripción. La información de las escenas tres y cuatro han sido suministrada en español e inglés para mantener la universalidad de la solución. La Figura.3.38 ilustra los modelos utilizados en las escenas uno y dos.

Figura 3.38: Modelos empleados en las escenas número 1 y 2 respectivamente
Fuente: Autor

Se han situado tres marcadores para alojar el modelo tridimensional del robot y los videos para la validación de la cinemática directa e inversa generados previamente en la Fase I. Los marcadores pueden ser impresos en una hoja de papel o presentados desde un dispositivo que permita su visualización. La Figura.3.39 presenta una visualización de los elementos que conforman la escena dos.

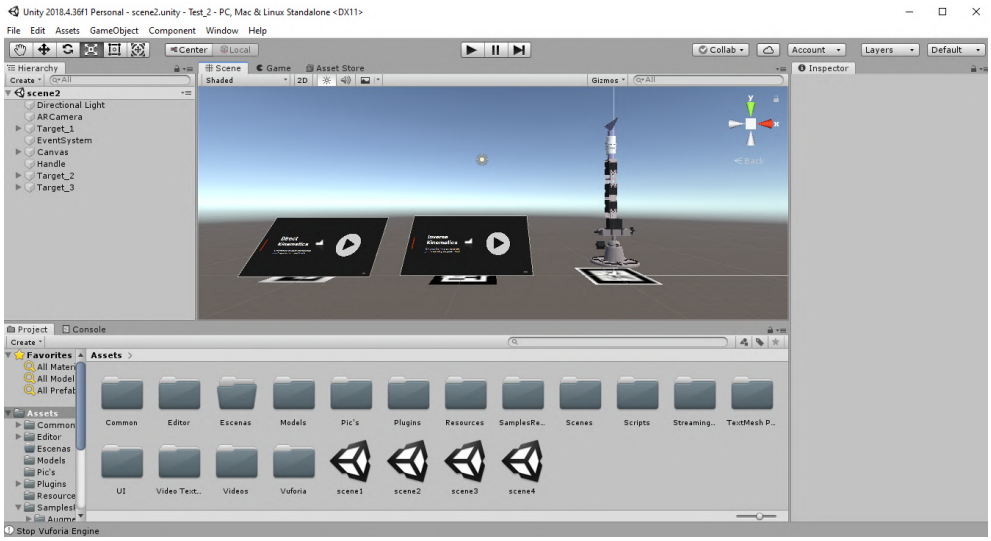

Figura 3.39: Diseño de la escena número 2 en el entorno de Unity
Fuente: Autor

3.3.6.2 Aplicación para dispositivos Microsoft HoloLens

En cuanto al desarrollo de la interfaz para Microsoft HoloLens, se implementó el HoloLens Emulator por medio de Visual Studio con el SDK de Windows 10.HoloLens Emulator es un software que permite compilar aplicaciones para Microsoft HoloLens y cascos envolventes de Windows Mixed Reality sin necesidad de acceder físicamente a estos dispositivos.

La aplicación de realidad aumentada para dispositivos Microsoft HoloLens cuenta con modelos 3D y documentación relacionada con el robot de estudio. El proyecto ha sido diseñado a través del complemento Virtual Reality Supported que a su vez permite utilizar el Windows Mixed Reality. La plataforma seleccionada para el desarrollo de la aplicación fue el Universal Windows Platform. Además, se han modificado algunas propiedades relacionadas con la cámara; se actualizó su sistema de coordenadas para que este coincida con el origen del sistema general. Por otro lado, se configuró el color de las bandas claras (Clear Flags) para que sea de un color sólido, el cual se ha seleccionado de color negro. Esto debido a que desde las HoloLens, cualquier elemento que sea representado con el color negro será transparente. De esta manera, será posible observar el mundo real a través del dispositivo, además de cualquier holograma o contenido virtual. En la Figura.3.40 se aprecian los componentes que hacen parte de la escena de la aplicación de realidad aumentada para dispositivos Microsoft HoloLens.

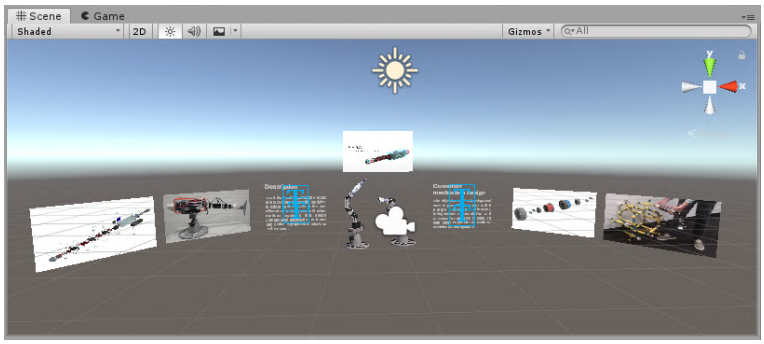

Figura 3.40: Escena con modelos y contenido para las Microsoft HoloLens en Unity
Fuente: Autor

3.4 Fase III: Implementación en el sistema físico

Esta fase está dirigida a la validación de referencias específicas generadas en las Fases I y II en el sistema físico. Para lo anterior, se han desarrollado dos métodos que comprenden el envío de datos Unity-Matlab y Matlab-Unity a través de protocolos de transporte en TCP/IP. La primera opción es abordada empleando el protocolo de control de transmisión (TCP). En cambio, la segunda opción, plantea la creación de un servidor en Matlab con un cliente en Unity mediante el protocolo de datagramas de usuario (UDP). Una vez establecida la comunicación entre las interfaces de visualización de realidad aumentada en Unity y las simulaciones en Matlab, se selecciona un grupo referencias para posteriormente asignarlas en un algoritmo desarrollado en la IDE de Arduino para ejecutar dichas acciones

en los actuadores del prototipo. Para la arquitectura electrónica se utiliza una DYNAMIXEL Shield sobre una tarjeta Adafruit METRO 328, como se ilustra en la Figura.3.41. Además, el dispositivo cuenta con una OpenCM9.04-C y un U2D2 para ser programado dependiendo de la aplicación.

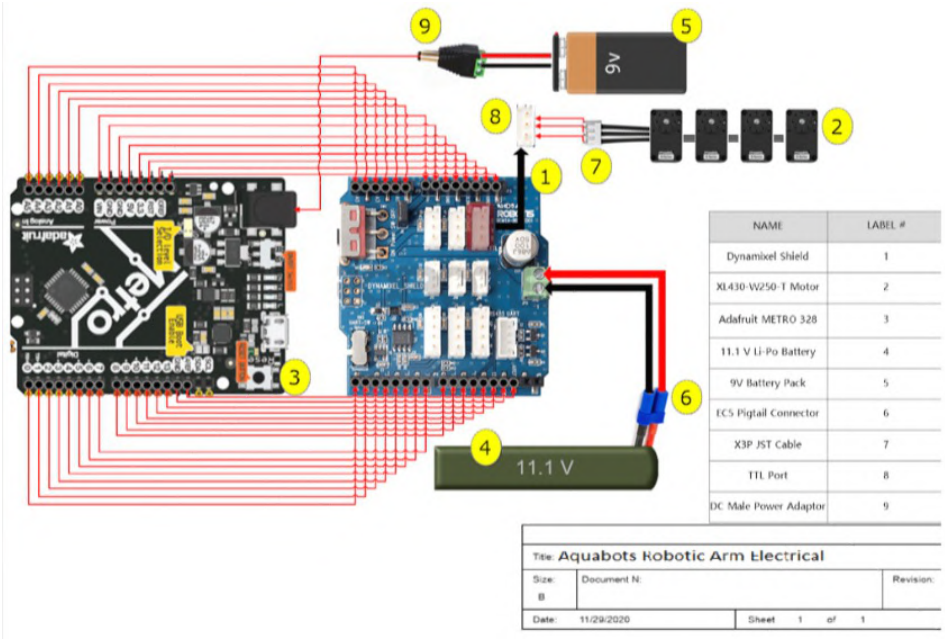

Figura 3.41: Esquema eléctrico del prototipo
Fuente: José Baca, Department of Engineering | Texas A&M University-Corpus Christi

3.4.0.1 Envío de parámetros Unity-Matlab mediante protocolo de control de transmisión (TCP)

Inicialmente, se decidió realizar un primer acercamiento mediante el uso del protocolo de control de transmisión (TCP) que permitiera adentrarnos en el tema y posteriormente, generar un modelo mejor equipado con otras aplicaciones. El protocolo TCP está orientado a la conexión, ofreciendo un servicio considerablemente fiable con un significativo trafico adicional en la red. Esto garantiza que los datos sean entregados a su destino sin errores, conservando el mismo orden en el que han sido transmitidos [39].

La herramienta Instrument Control Toolbox de Matlab proporciona un soporte para establecer una comunicación TCP/IP que incluye la capacidad de crear clientes y servidores. Su compatibilidad le permite realizar la comunicación de socket de red para conectarse a hosts remotos desde MATLAB con la finalidad de poder leer y escribir datos binarios y ASCII [40].

El robot es simulado a través de la librería Robotics Toolbox (Peter Corke) [41], cuyos movimientos corresponderán a los generados en el modelo de realidad aumentada creado en Unity. Es importante resaltar que la entrada que recibe el programa es una matriz de transformación homogénea, cuya salida resulta en un cuaternio. Se han utilizado funciones para

permitir la comunicación entre Unity-Matlab, con el fin de distinguir entre los mensajes enviados se emplean 4 cadenas de caracteres con la información de cada una de las articulaciones. Así, la trama de datos es recibida e interpretada de forma correcta para cada articulación del robot en Matlab. La Figura.3.42 ilustra el envió de parámetros entre el modelo en realidad aumentada en Unity y la simulación del robot a través de la librería Robotics Toolbox en Matlab.

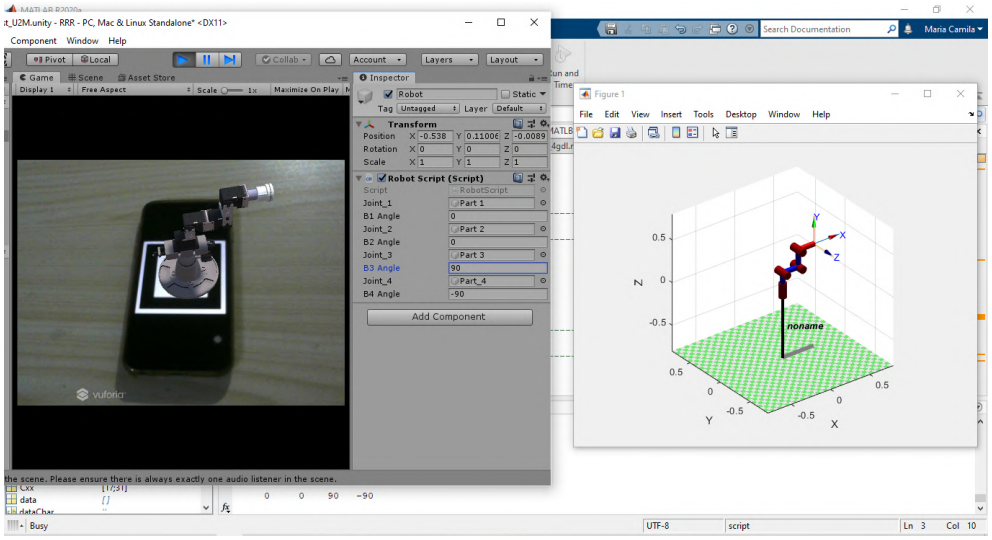

Figura 3.42: Envío de parámetros Unity-Matlab usando TCP
Fuente: Autor

3.4.0.2 Envío de parámetros Matlab-Unity mediante protocolo de datagramas de usuario (UDP)

La comunicación Matlab-Unity por UDP se llevó a cabo de la misma manera que en la comunicación Unity-Matlab por TCP. Sin embargo, para la simulación del prototipo se han implementado los análisis realizados en apartados anteriores. El protocolo UDP a diferencia del protocolo TCP, no emplea ninguna sincronización entre origen y destino. Este intercambia información en forma de bloques de bytes que provocan poca carga adicional a la red. Al no haber una conexión, es posible utilizar como dirección IP de destino la dirección broadcast o multicast. Esto permite el envío de un mismo paquete de datos a múltiples destinos de forma simultánea [39].

Por otro lado, resulta conveniente aclarar que el prototipo al estar fundamentado en el concepto de la modularidad puede adoptar múltiples configuraciones para llevar a cabo diversas aplicaciones. En este caso, se contempla la configuración expuesta en la Figura.3.43 con una herramienta experimental como efector final. Para comprender de una mejor forma el objetivo de este apartado, se han desarrollado una serie de simulaciones en Matlab del robot de estudio las cuales se ilustran en la Figura.3.44.

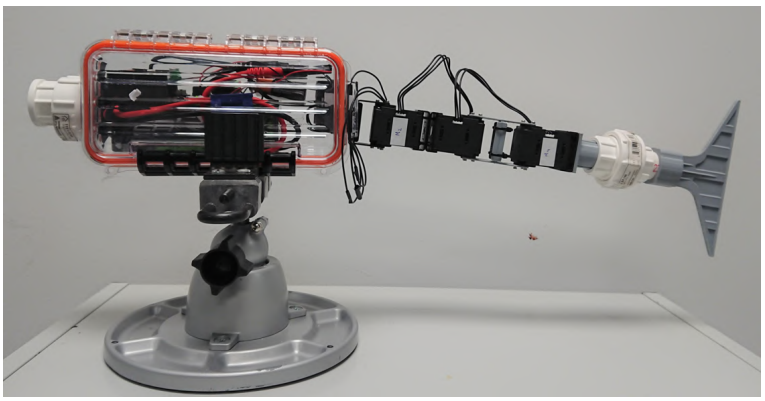

Figura 3.43: Configuración utilizada para la fase de implementación de algoritmos
Fuente: José Baca, Department of Engineering | Texas A&M University-Corpus Christi

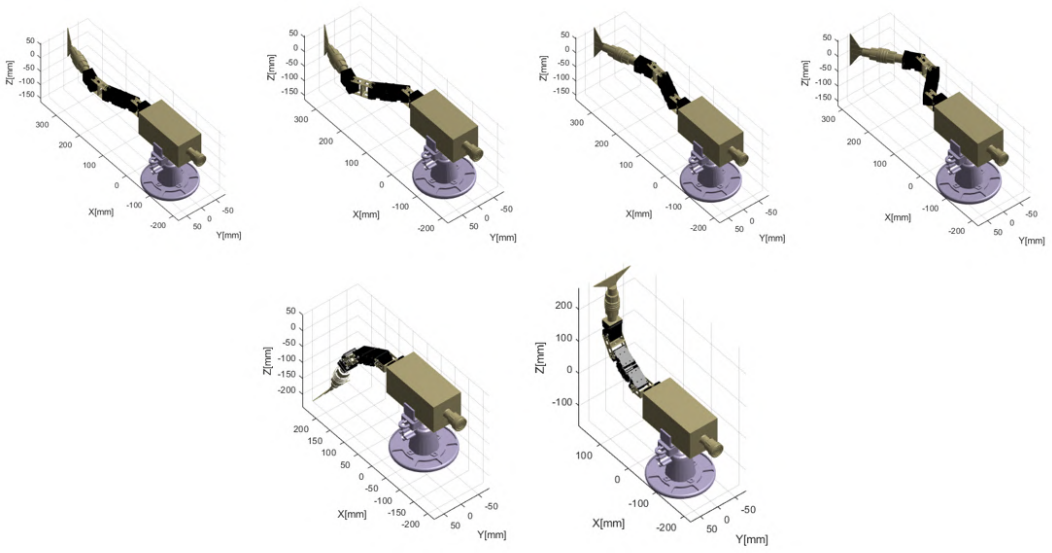

Figura 3.44: Simulaciones de distintas configuraciones del prototipo en Matlab
Fuente: Autor

Para facilitar el estudio y la transmisión de datos del prototipo entre las simulaciones e interfaces de visualización de realidad aumentada, se ha generado una GUI (Graphical User Interface) en Matlab con funciones destinadas a calcular la cinemática directa e inversa del robot que a su vez, permiten obtener un grupo de referencias que serán válidas experimentalmente. En la Figura.3.45 se aprecia la GUI elaborada en Matlab.

La Figura.3.46 presenta los resultados de la configuración establecida para la transmisión de datos Matlab-Unity.

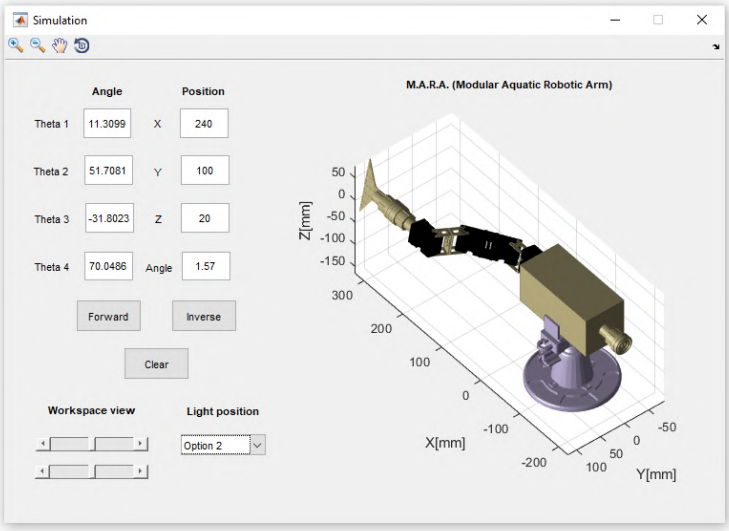

Figura 3.45: Interfaz gráfica desarrollada en Matlab
Fuente: Autor

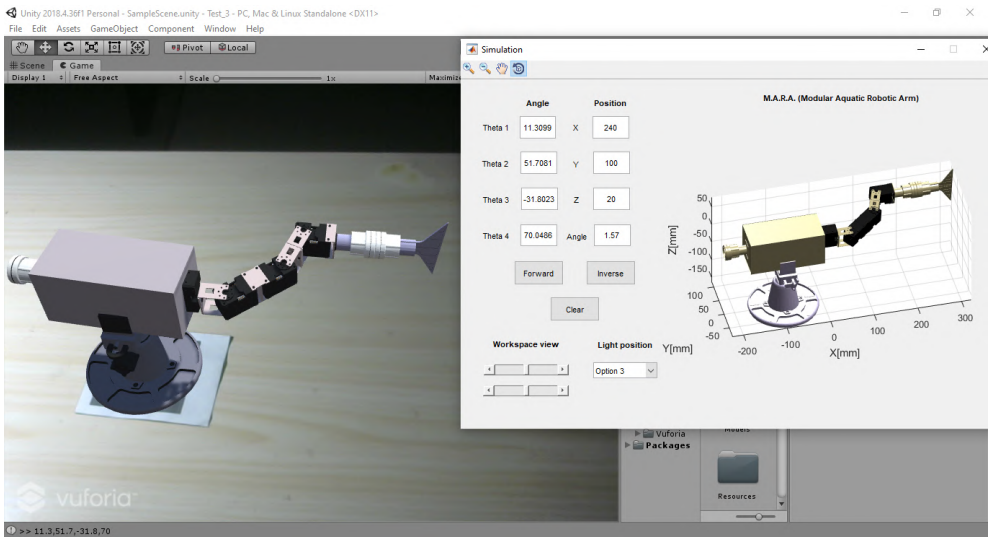

Figura 3.46: Envío de parámetros Matlab-Unity usando UDP
Fuente: Autor

3.5 Módulos de aprendizaje

Con el fin de proporcionar una adecuada asimilación de cada uno de los componentes que estructuran el presente estudio, se ha destinado un apartado que reúne los contenidos más relevantes de cada una de las fases por medio de un hipervínculo. Este permite enlazar con una lista de videostutoriales alojada en la plataforma YouTube. Se hace hincapié en que no se abordará un estudio exhaustivo de las fases, pues no es el objetivo propio de esta sección, sino servir como información complementaria para la puesta en marcha de las técnicas

desarrolladas. La Figura.3.47 ilustra una visualización de la playlist destinada a recopilar los videotutoriales.

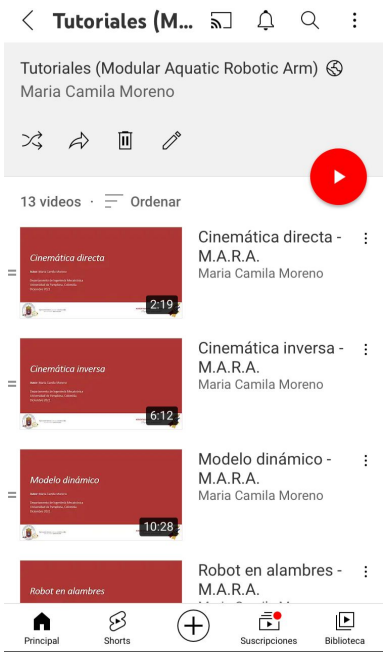

Figura 3.47: Playlist en la plataforma YouTube
Fuente: Autor

En la Figura.3.48 se presenta un código QR generado específicamente para que el lector pueda escanear empleando un teléfono inteligente y acceder a la lista de reproducción de contenido.

Figura 3.48: Hipervínculo asignado para el enlace con la documentación
Fuente: Autor

4 Resultados

4.1 Aplicaciones de realidad aumentada

En esta sección se exponen los resultados obtenidos de las pruebas de funcionamiento de las interfaces de visualización de realidad aumentada con Unity y Vuforia Engine empleando diferentes dispositivos.

4.1.1 Funcionamiento para teléfonos inteligentes y cámaras web

Inicialmente, se realizaron pruebas de funcionamiento empleando la cámara web del sistema de cómputo. Posteriormente, se llevó a cabo su implementación en teléfonos móviles que cuenten con Android como sistema operativo. Por otro lado, es preciso tener presente al momento de implementar la aplicación en teléfonos inteligentes contar con los componentes SDK, JDK y NDK para su correcto funcionamiento.Todo lo anterior gracias a la versatilidad de Unity para implementar proyectos en dispositivos móviles, computadoras de escritorio, VR/AR, consolas o la Web, entre otros. En la Figura.4.1 se puede apreciar una visualización del entorno de la interfaz de realidad aumentada en la escena número 2.

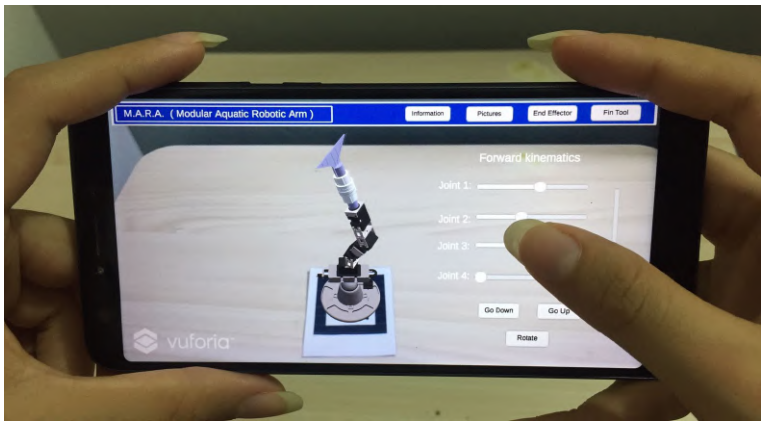

Figura 4.1: Aplicación de realidad aumentada para teléfonos inteligentes
Fuente: Autor

La Figura.4.2 ilustra las escenas número 1 y 2 de la aplicación de realidad aumentada en donde se expone en un principio al robot sin una herramienta experimental en su efector final. En cambio, en la segunda escena (Figura.4.3), se observa como el prototipo cuenta con un módulo inspirado en la aleta de algunos animales acuáticos para dotarlo con capacidades de locomoción.

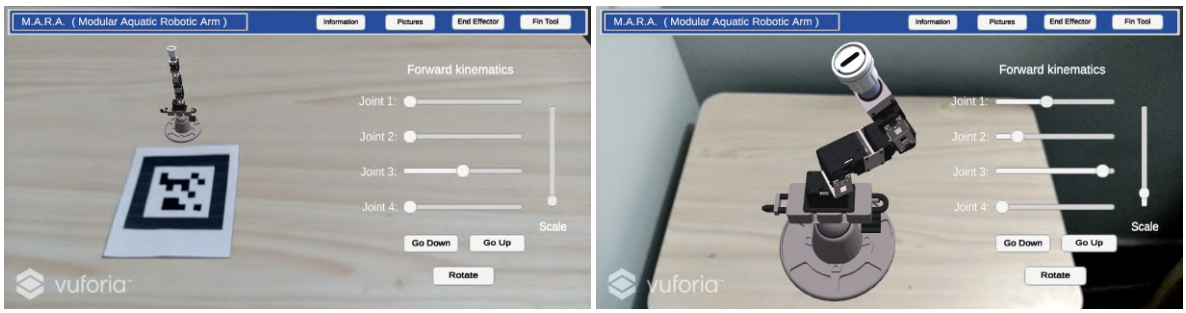

Figura 4.2: Modelo 3D del prototipo en diferentes escalas
Fuente: Autor

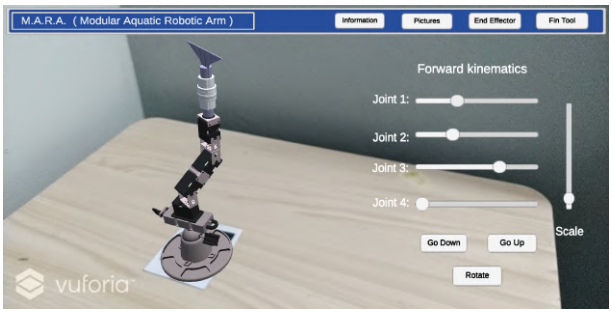

Figura 4.3: Modelo del robot de estudio con la herramienta experimental
Fuente: Autor

En la Figura.4.4 y Figura.4.5 se presentan los videos de las simulaciones realizadas para los apartados de la cinemática inversa y directa del prototipo empleando Matlab como software computacional. El proceso de reproducción del video se efectúa mediante un toque en el botón "Play". Además, es posible pausar el video en un momento determinado o incluso volver a reproducirlo. Por otra parte, en la Figura.4.6 se presentan todos los elementos que hacen parte de la escena número 2 de la aplicación.

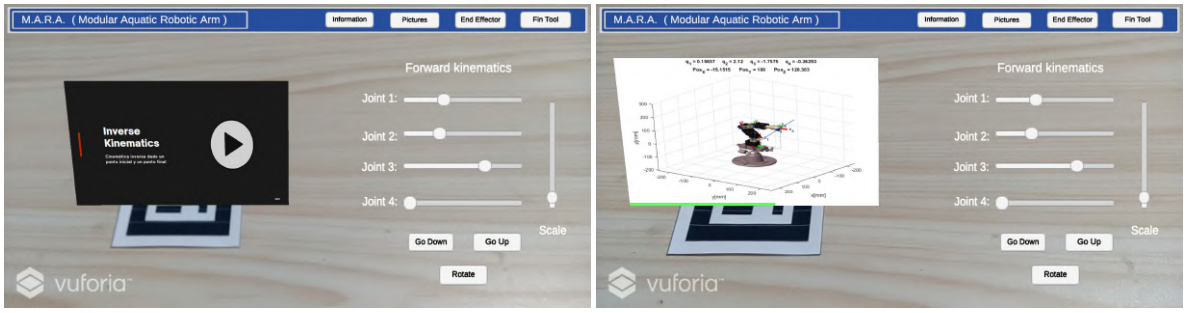

Figura 4.4: Visualización de los videos de la cinemática inversa en la escena número 2
Fuente: Autor

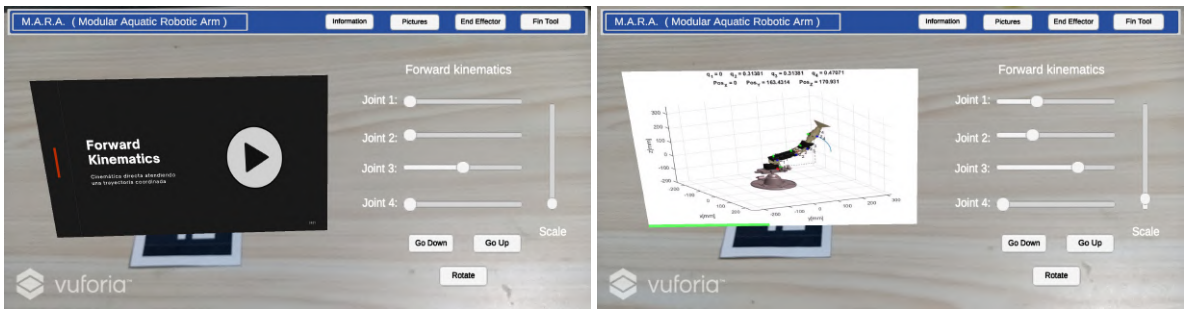

Figura 4.5: Visualización de los videos de la cinemática directa en la escena número 2
Fuente: Autor

Figura 4.6: Contenido virtual disponible en la escena número 2
Fuente: Autor

Para evitar confusiones entre el modelo tridimensional y el robot real de estudio, se ha precisado de documentación, imágenes y fotografías del prototipo anexadas en las escenas número 3 y 4 de la aplicación. La Figura.4.7 presenta los aspectos más importantes del brazo robótico acuático modular.

Figura 4.7: Visualización de la escenas número 3 y 4
Fuente: Autor

4.1.2 Funcionamiento para dispositivos Microsoft HoloLens

En la Figura.4.8 se presentan algunas propiedades técnicas relacionadas con la información del sistema operativo HoloLens (HoloLens Operating System Information) y el estado del dispositivo (Device Status).

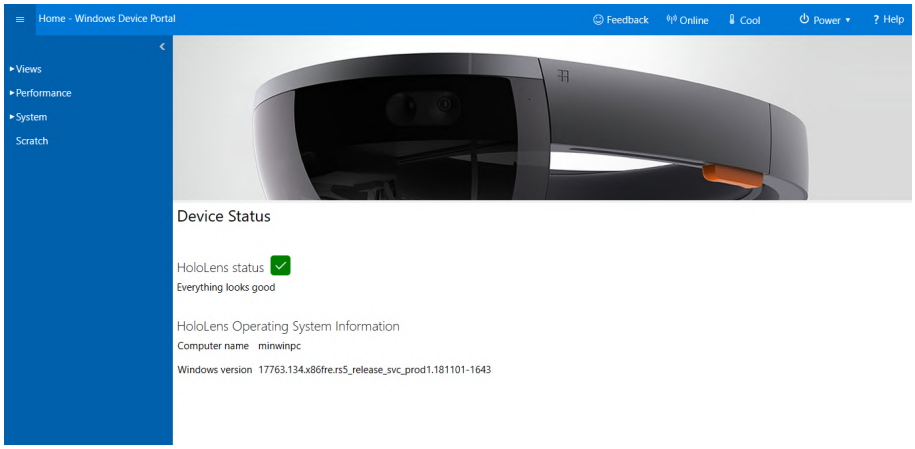

Figura 4.8: Portal de dispositivos de Windows
Fuente: Autor

La Figura.4.9 ilustra la etapa de inicialización de la aplicación en el HoloLens Emulator.

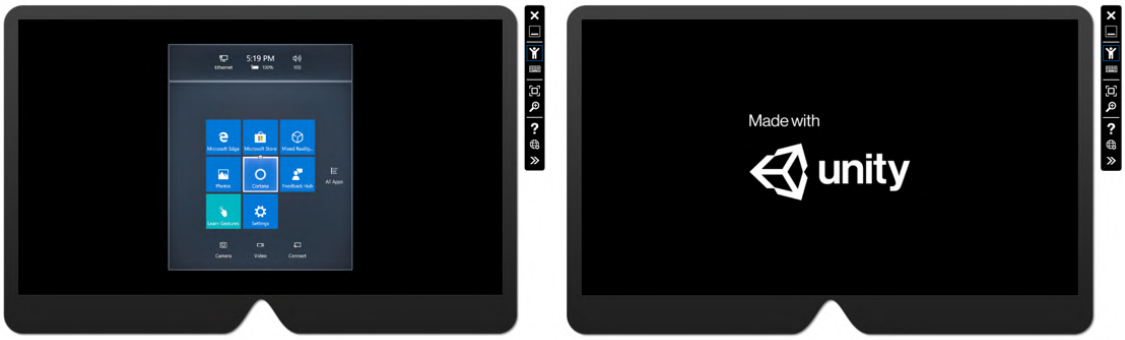

Figura 4.9: Inicialización de la aplicación desde HoloLens Emulator
Fuente: Autor

Inicialmente, se presenta una breve descripción del mecanismo conector en conjunto con imágenes del diseño y funcionamiento propuestos para su implementación, como se observa en la Figura.4.10. La Figura.4.11 expone información general sobre el prototipo.

La Figura.4.12 presenta una visualización general de todos los elementos que hacen parte de la escena. Por último, se ilustra el campo de visión del usuario si estuviera usando unas Microsoft HoloLens en el mundo real.

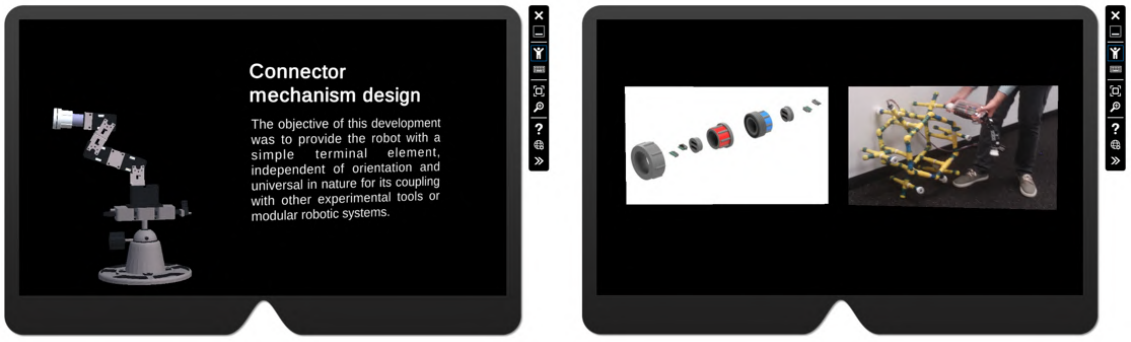

Figura 4.10: Documentación relacionada con el mecanismo conector
Fuente: Autor

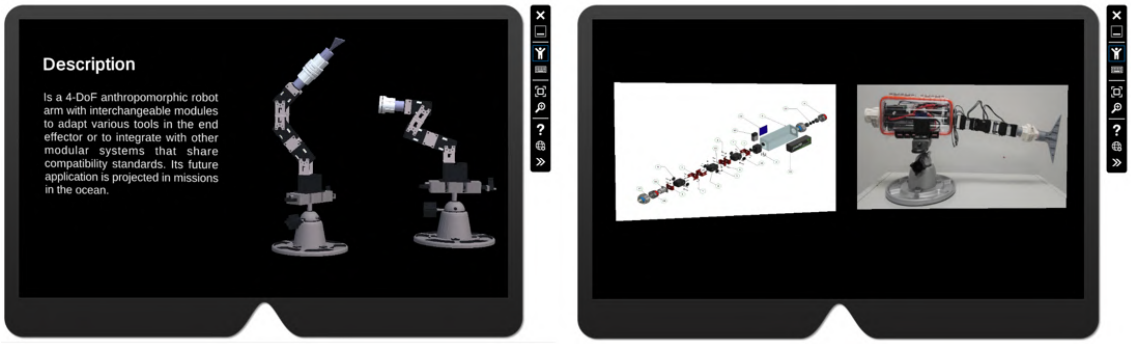

Figura 4.11: Documentación relacionada con algunas generalidades del prototipo
Fuente: Autor

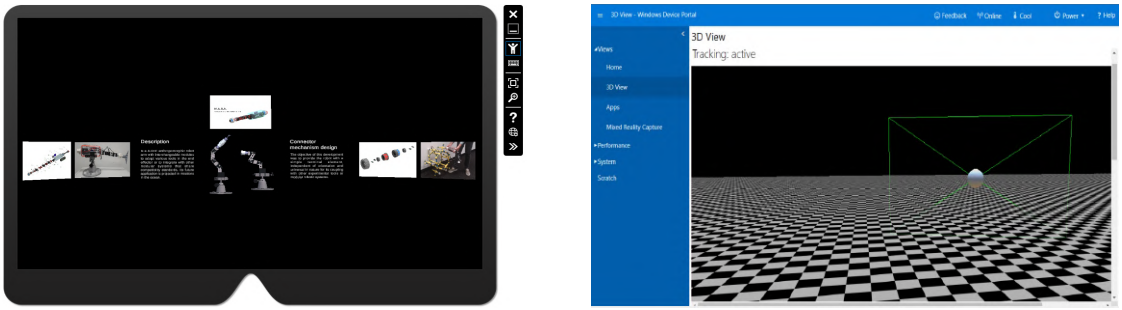

Figura 4.12: Contenido virtual de la aplicación de realidad aumentada para Microsoft HoloLens
Fuente: Autor

4.2 Validación de los algoritmos de control

Para la validación de las referencias generadas por los algoritmos previamente expuestos, se llevó a cabo la corroboración de los análisis cinemático directo e inverso. Es preciso tener presente que estas referencias deben ajustarse de acuerdo a la hoja de características técnicas del fabricante de los actuadores incorporados en el prototipo (XL430-W250). La Figura.4.13 y Figura.4.14 ilustran el proceso de corroboración del análisis cinemático.

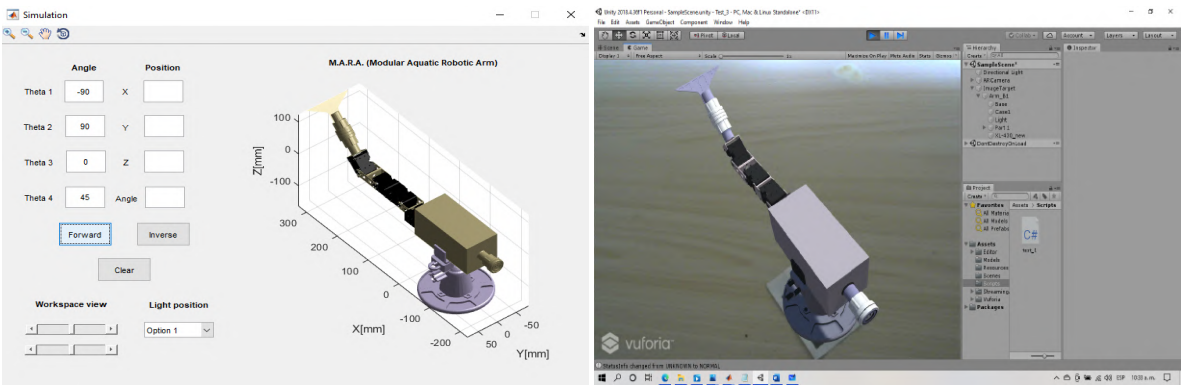

Figura 4.13: Funcionamiento de las interfaces de visualización en realidad aumentada y las simulaciones

Fuente: Autor

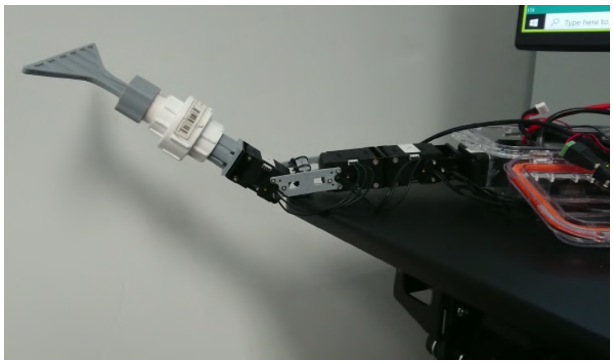

Figura 4.14: Validación en el sistema físico
Fuente: José Baca, Department of Engineering | Texas A&M University-Corpus Christi

Asimismo, se lleva a cabo la validación para las referencias de posición generadas en el método *Envío de parámetros Unity-Matlab mediante protocolo de control de transmisión (TCP)*. En la Figura.4.15 se aprecia la comunicación entre la interfaz de realidad aumentada y la simulación del robot empleando la librería Robotics Toolbox de Matlab. La Figura.4.16 ilustra el resultado de la configuración en el sistema físico.

Además, se propone la validación de un algoritmo empleando referencias de velocidad para demostrar algunas aplicaciones de desplazamiento que puede ejecutar el robot de estudio. Dicho código está diseñado para realizar tres movimientos de desplazamiento, estos posteriormente son cargados y puestos en funcionamiento. La Figura.4.17 describe el resultado de esta configuración.

Finalmente, el prototipo es preparado para ser sumergido en una piscina con el algoritmo expuesto anteriormente. Para esta prueba de funcionamiento, un equipo de estudiantes del laboratorio del departamento de ingeniería de la universidad de Texas A&M - Corpus Christi desarrollo una manta protectora que pueda cubrir y evitar el ingreso de agua al sistema empleando múltiples capas de material antifluido y sellándolos con una pistola térmica. Esto con

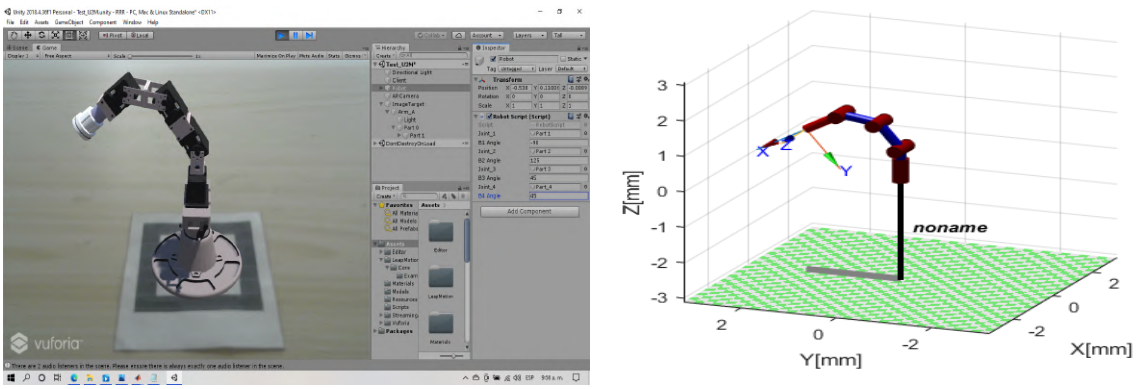

Figura 4.15: Validación del método Envío de parámetros Unity-Matlab mediante protocolo de control de transmisión (TCP)
Fuente: Autor

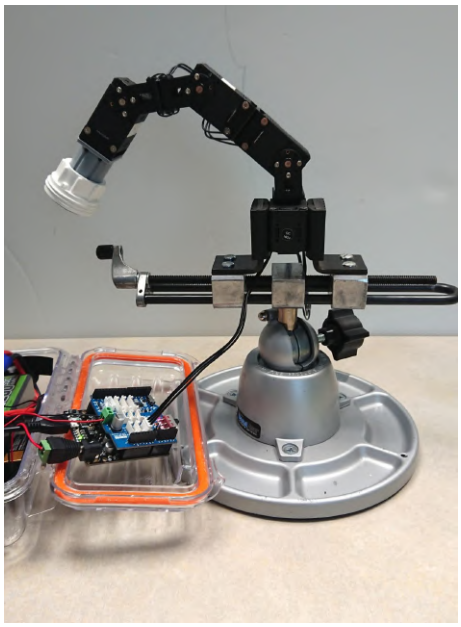

Figura 4.16: Validación de las referencias de posición generadas en el sistema físico
Fuente: José Baca, Department of Engineering | Texas A&M University-Corpus Christi

el fin de evitar comprometer el funcionamiento y presentación del robot durante su estudio. La Figura.4.18 presenta como se llevó a cabo la prueba de funcionamiento del prototipo en un medio acuático.

La Figura.4.19 y Figura.4.20 presentan algunas de las pruebas de funcionamiento llevadas a cabo bajo la supervisión del Ph.D. José Baca, Assistant Professor, Dept. of Engineering, Texas A&M University-Corpus Christi. En estas pruebas, se pretendía demostrar diversos movimientos que puede ejecutar el prototipo adoptando una configuración de manipulador.

Figura 4.17: Validación de las referencias de velocidad
Fuente: José Baca, Department of Engineering | Texas A&M University-Corpus Christi

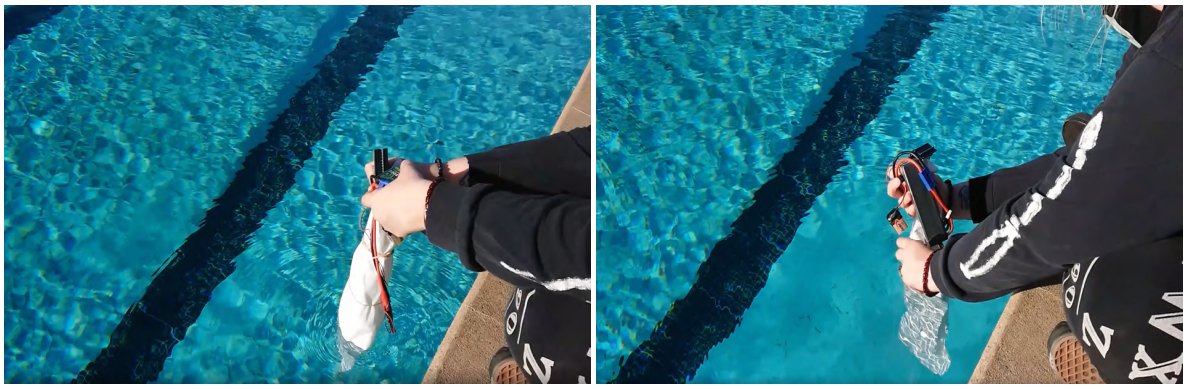

Figura 4.18: Funcionamiento del prototipo en un medio acuático
Fuente: José Baca, Department of Engineering | Texas A&M University-Corpus Christi

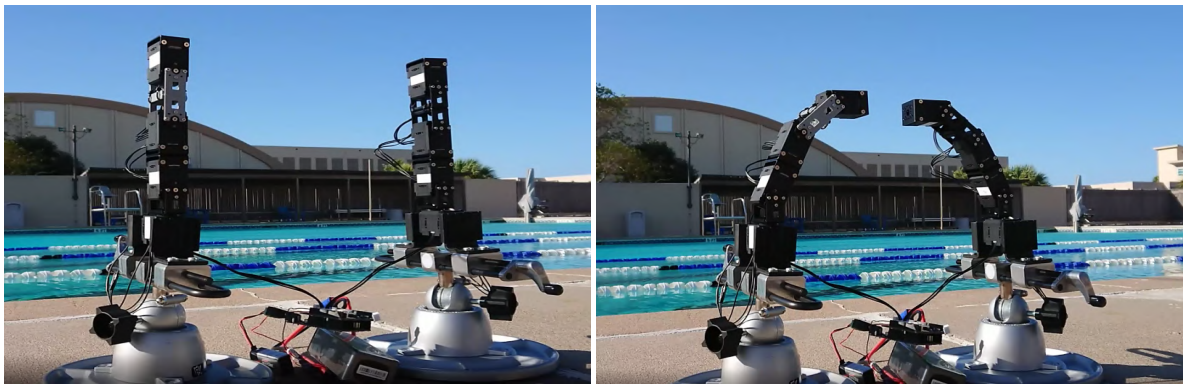

Figura 4.19: Primeras fotografías sobre los movimientos del robot como manipulador
Fuente: José Baca, Department of Engineering | Texas A&M University-Corpus Christi

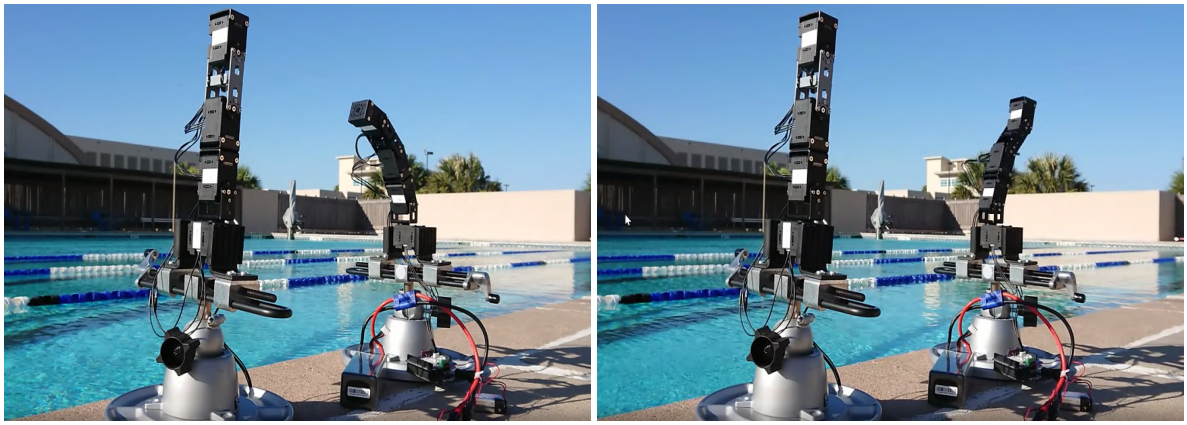

Figura 4.20: Segundas fotografías sobre los movimientos del robot como manipulador
Fuente: José Baca, Department of Engineering | Texas A&M University-Corpus Christi

5 Conclusiones y trabajo futuro

El presente capítulo resume las conclusiones del estudio abordado en el trabajo de grado. Conjuntamente, se presentan las principales líneas de investigación que merecen un estudio o profundización en el futuro dados sus interesantes aspectos.

5.1 Conclusiones

Se ha presentado un método que permite analizar, diseñar e implementar diferentes técnicas de control en un robot antropomórfico de 4 grados de libertad con todas sus articulaciones rotacionales. A partir de esto, se desarrollan algoritmos de control para misiones de manipulación y desplazamiento dadas las características del prototipo y sus posibles escenarios de aplicación.

Las simulaciones permitieron comprobar de manera asertiva la resolución del problema cinemático, dinámico y posibilitar la observación y análisis del control cinemático aplicado al prototipo. Partiendo de lo expuesto anteriormente, fue posible visualizar los efectos que implica la ejecución de un conjunto de movimientos establecidos para posicionar el efector final del robot de acuerdo a una determinada aplicación.

Desde una perspectiva computacional, resulta indispensable definir el límite de puntos intermedios para abordar las estrategias de control de un robot empleando interpoladores cúbicos, esto se debe a que un mayor número de puntos requeriría mayores prestaciones respecto a la capacidad de procesamiento y almacenamiento del sistema de cómputo. Por tanto, la estrategia propuesta en este documento puede presentar implicaciones de precisión, reducción de velocidad o disminución en el tiempo de respuesta.

Las interfaces de visualización de realidad aumentada para teléfonos inteligentes, cámaras web y Microsoft HoloLens estimulan la interacción humano-robot y favorecen los procesos de exploración, comprensión y retención de la información más relevante sobre el prototipo empleando elementos de fácil acceso para su implementación. Además, le otorga al usuario la libertad de indagar y descubrir cada uno de los contenidos anexados a su propio ritmo, evitando altos niveles de presión y estrés durante el proceso.

En los experimentos realizados para la validación de los algoritmos de control en la Fase III, se han obtenido unos buenos resultados para aplicaciones de desplazamiento. Los movimientos generados por el prototipo en el sistema físico, tienen un grado de correspondencia al expuesto en las simulaciones. Evidenciando de una forma más realista la relación entre los análisis y las interfaces de visualización.

5.2 Trabajo futuro

Se han abierto varias líneas de trabajo futuro como resultado de esta investigación. Una de ellas es el estudio de un control teleoperado del robot empleando dispositivos Microsoft HoloLens, interfaces de realidad aumentada y protocolos de internet para facilitar las acciones de control proporcionadas por el usuario. Se pretende además involucrar otras aplicaciones que brindan las HoloLens como la telepresencia o videollamadas holográficas para poder reunir a investigadores, docentes y estudiantes con el objetivo de discutir las próximas operaciones o acciones de control que deberá ejecutar el prototipo en escenarios futuros. También se considera de interés analizar la implementación de un sistema sensorial para adquirir registros sobre las variables con las que interactuará el robot.

Por último, se sugiere la implementación de una propuesta de control dinámico en el prototipo para disminuir algunas perturbaciones presentadas durante las pruebas de funcionamiento en el sistema físico. Esto mejorará la precisión y efectividad de las tareas que pueda ejecutar el robot.

6 Anexos

6.1 Cálculo del modelo cinemático

Código 6.1: Función para la cinemática directa

```
1
2 function A04=CD_Robot(q)
3
4 % ------------------------------------------------------------------------
5 % Declaración de variables
6 % ------------------------------------------------------------------------
7     LA=34;
8     LB=31.70;
9     L1=LA+LB;
10    L2=81;
11    L3=55;
12    LC=40;
13    LD=52;
14    L4=LC+LD;
15
16    L = [L1,L2,L3,L4];
17
18    A00 = eye(4);
19
20 % ------------------------------------------------------------------------
21 % Tabla de parámetros Denavit-Hartenberg
22 % ------------------------------------------------------------------------
23    theta_DH = [ q(1)+pi/2, q(2), q(3), q(4) ];
24    d_DH = [ L(1), 0, 0, 0 ];
25    a_DH = [ 0, L(2), L(3), L(4) ];
26    alpha_DH = [ pi/2, 0, 0, 0 ];
27
28 % ------------------------------------------------------------------------
29 % Cálculo de las matrices A01, A12, A23 y A34
30 % ------------------------------------------------------------------------
31    A01 = DH(theta_DH(1), d_DH(1), a_DH(1), alpha_DH(1));
32    A12 = DH(theta_DH(2), d_DH(2), a_DH(2), alpha_DH(2));
33    A23 = DH(theta_DH(3), d_DH(3), a_DH(3), alpha_DH(3));
34    A34 = DH(theta_DH(4), d_DH(4), a_DH(4), alpha_DH(4));
35
36 % ------------------------------------------------------------------------
37 % Matrices con respecto al sistema 0
38 % ------------------------------------------------------------------------
39    A02 = A01*A12;
40    A03 = A02*A23;
41    A04 = A03*A34;
42
43    Dibujar_Robot_2(A00,A01,A02,A03,A04)
44
45 end
```

Código 6.2: Función para la cinemática inversa

```
1
```

```
function q=CI_Robot(Px4, Py4, Pz4,b,CODO)

% Longitudes
LA=34;
LB=31.70;
L1=LA+LB;
L2=81;
L3=55;
LC=40;
LD=52;
L4=LC+LD;

% Vector de longitudes del robot
L = [L1,L2,L3,L4];

% Primera coordenada articular
Px4=Px4*-1;
q1 = atan2(Px4,Py4);

% Vector P4
P4 = [Px4; Py4; Pz4; 1];

% Vector X4
X4 = [cos(b)*sin(q1); cos(b)*cos(q1); sin(b); 0];

% Punto muñeca
Pm = P4 - X4*L(4);

Px4 = Pm(1);
Py4 = Pm(2);
Pz4 = Pm(3);

D = sqrt(Px4^2+Py4^2);
J12 = D^2 + (Pz4-L(1))^2;
cosq3= (J12 - L(2)^2 - L(3)^2) / (2*L(2)*L(3));

% Error de redondeo
if cosq3> 1 & cosq3< 1.001
    cosq3= 1;
    warning('Error de redondeo en Cosq3')
end

if cosq3< -1 & cosq3> -1.001
    cosq3= -1;
    warning('Error de redondeo en Cosq3')
end

sinq3 = CODO*sqrt(1-cosq3^2);

% Error al exceder el espacio de trabajo
Y = imag(sinq3)
if (Y~=0)
    qq=[0,0,0,0];
    CD_Robot(qq);
    error('Error: Ingrese un punto dentro del espacio de trabajo');
end

% Tercera coordenada articular
q3= atan2(sinq3,cosq3)
beta = atan2(L(3)*sin(q3),L(2)+L(3)*cos(q3));
alpha = atan2(Pz4-L(1),D);

% Segunda coordenada articular
q2 = alpha - beta;
```

```
66
67 % Cuarta coordenada articular con respecto a la horizontal
68 q4 = b − q2− q3;
69
70 % Vector con coordenadas articulares del robot
71 q = [ q1, q2, q3, q4]
```

Código 6.3: Función para dibujar los alambres e importar las piezas del robot

```
1
2 function Dibujar_Robot_2(A00,A01,A02,A03,A04)
3
4     % Carga de STL's
5      load('Preloaded_Robot_Geometries_B.mat');
6
7
8     % Matriz correspondiente a la base del robot
9      Base=eye(4)*TrasZ(34);
10
11    % Matriz correspondiente a la pieza 0 del robot
12     Ac=Base;
13
14    % Función dibujar objetos
15     dibujar_objeto_matlab_from_stl(robot.stl.piece0,Base);
16     dibujar_objeto_matlab_from_stl(robot.stl.piece1,A01);
17     dibujar_objeto_matlab_from_stl(robot.stl.piece2,A02);
18     dibujar_objeto_matlab_from_stl(robot.stl.piece3,A03);
19     dibujar_objeto_matlab_from_stl(robot.stl.piece4,A04);
20     dibujar_objeto_matlab_from_stl(robot.stl.piece5,A04);
21     dibujar_objeto_matlab_from_stl(robot.stl.piece6,Ac);
22
23    % Color y grosor del alambre
24    Color = [0,0,0];
25    Gro_1 = 2;
26
27    % Longitud y grosor de los ejes coordenados
28    Gro_2 = 3;
29    Lon=30;
30
31    hold on
32
33     % Dibujo de los sistemas de referencia
34    dibujar_sistema_referencia_MTH(A00,Lon,Gro_2,'0');
35    dibujar_sistema_referencia_MTH(A01,Lon,Gro_2,'1');
36    dibujar_sistema_referencia_MTH(A02,Lon,Gro_2,'2');
37    dibujar_sistema_referencia_MTH(A03,Lon,Gro_2,'3');
38    dibujar_sistema_referencia_MTH(A04,Lon,Gro_2,'4');
39
40    % Cálculo de los puntos para el alambre del robot
41    P00= A00(1:4,4);
42    P1 = A01(1:4,4);
43    P2 = A02(1:4,4);
44    P3 = A03(1:4,4);
45    P4 = A04(1:4,4);
46
47    % Dibujo de los puntos anteriormente calculados
48    dibujar_linea(P00, P1, Color,Gro_1);
49    dibujar_linea(P1, P2, Color,Gro_1);
50    dibujar_linea(P2, P3, Color,Gro_1);
51    dibujar_linea(P3, P4, Color,Gro_1);
52
53    % Ajuste del espacio de trabajo
54    view(114,35),camlight(56,57);grid on, lighting phong;
55    axis equal
```

```
56    xlabel('X[mm]'); ylabel('Y[mm]');zlabel('Z[mm]');
57
58    end
```

6.2 Cálculo del modelo dinámico

Código 6.4: Algoritmo para el cálculo del modelo dinámico

```
1
2 clear all
3 clc
4 close all
5
6 % ------------------------------------------------------------------------
7 % Declaración de variables simbólicas
8 % ------------------------------------------------------------------------
9 syms q1 q2 q3 q4 % Posiciones articulares
10 syms qp1 qp2 qp3 qp4 % Velocidades articulares
11 syms m1 m2 m3 m4 % Masa
12 syms g % Gravedad
13 pi=sym('pi'); % pi como un objeto simbólico
14
15 % ------------------------------------------------------------------------
16 % Declaración de los vectores q, qp, masa y vec_q
17 % ------------------------------------------------------------------------
18 q = [q1 ; q2; q3; q4]; % Vector de posición
19 qp = [qp1 ; qp2; qp3; qp4]; % Vector de velocidades
20 masa = [m1 ; m2; m3; m4]; % Vector de masas del robot
21 vec_g = [ g, 0, 0, 0]; % Vector de gravedad
22
23 % Longitudes
24 LA=0.034;
25 LB=0.03170;
26 L1=LA+LB;
27 L2=0.081;
28 L3=0.055;
29 LC=0.040;
30 LD=0.052;
31 L4=LC;
32
33 % ------------------------------------------------------------------------
34 % Tabla de parámetros D-H.
35 % ------------------------------------------------------------------------
36 theta_dh = [ q(1)+pi/2, q(2), q(3), q(4) ];
37 d_dh = [ L1, 0, 0, 0 ];
38 a_dh = [ 0, L2, L3, L4 ];
39 alfa_dh = [ pi/2, 0, 0, 0 ];
40
41 % Matrices A01, A12, A23 y A34
42 A01=denavit(theta_dh(1),d_dh(1),a_dh(1),alfa_dh(1));
43 A12=denavit(theta_dh(2),d_dh(2),a_dh(2),alfa_dh(2));
44 A23=denavit(theta_dh(3),d_dh(3),a_dh(3),alfa_dh(3));
45 A34=denavit(theta_dh(4),d_dh(4),a_dh(4),alfa_dh(4));
46
47 % ------------------------------------------------------------------------
48 % Matrices de transformación
49 % ------------------------------------------------------------------------
50 A01 = simplify(A01);
51 A02 = simplify(A01*A12);
52 A03 = simplify(A02*A23);
53 A04 = simplify(A03*A34);
```

```
54
55% ------------------------------------------------------------------
56% Matrices Uij
57% ------------------------------------------------------------------
58U11=diff(A01,q(1));
59U12=diff(A01,q(2));
60U13=diff(A01,q(3));
61U14=diff(A01,q(4));
62
63U21=diff(A02,q(1));
64U22=diff(A02,q(2));
65U23=diff(A02,q(3));
66U24=diff(A02,q(4));
67
68U31=diff(A03,q(1));
69U32=diff(A03,q(2));
70U33=diff(A03,q(3));
71U34=diff(A03,q(4));
72
73U41=diff(A04,q(1));
74U42=diff(A04,q(2));
75U43=diff(A04,q(3));
76U44=diff(A04,q(4));
77
78% ------------------------------------------------------------------
79% Matrices Uijk
80% ------------------------------------------------------------------
81U111=diff(U11,q(1));
82U112=diff(U11,q(2));
83U113=diff(U11,q(3));
84U114=diff(U11,q(4));
85
86U121=diff(U12,q(1));
87U122=diff(U12,q(2));
88U123=diff(U12,q(3));
89U124=diff(U12,q(4));
90
91U131=diff(U13,q(1));
92U132=diff(U13,q(2));
93U133=diff(U13,q(3));
94U134=diff(U13,q(4));
95
96U141=diff(U14,q(1));
97U142=diff(U14,q(2));
98U143=diff(U14,q(3));
99U144=diff(U14,q(4));
100
101U211=diff(U21,q(1));
102U212=diff(U21,q(2));
103U213=diff(U21,q(3));
104U214=diff(U21,q(4));
105
106U221=diff(U22,q(1));
107U222=diff(U22,q(2));
108U223=diff(U22,q(3));
109U224=diff(U22,q(4));
110
111U231=diff(U23,q(1));
112U232=diff(U23,q(2));
113U233=diff(U23,q(3));
114U234=diff(U23,q(4));
115
116U241=diff(U24,q(1));
117U242=diff(U24,q(2));
```

```
U243=diff(U24,q(3));
U244=diff(U24,q(4));

U311=diff(U31,q(1));
U312=diff(U31,q(2));
U313=diff(U31,q(3));
U314=diff(U31,q(4));

U321=diff(U32,q(1));
U322=diff(U32,q(2));
U323=diff(U32,q(3));
U324=diff(U32,q(4));

U331=diff(U33,q(1));
U332=diff(U33,q(2));
U333=diff(U33,q(3));
U334=diff(U33,q(4));

U341=diff(U34,q(1));
U342=diff(U34,q(2));
U343=diff(U34,q(3));
U344=diff(U34,q(4));

U411=diff(U41,q(1));
U412=diff(U41,q(2));
U413=diff(U41,q(3));
U414=diff(U41,q(4));

U421=diff(U42,q(1));
U422=diff(U42,q(2));
U423=diff(U42,q(3));
U424=diff(U42,q(4));

U431=diff(U43,q(1));
U432=diff(U43,q(2));
U433=diff(U43,q(3));
U434=diff(U43,q(4));

U441=diff(U44,q(1));
U442=diff(U44,q(2));
U443=diff(U44,q(3));
U444=diff(U44,q(4));

% ------------------------------------------------------------------
% Matrices de PseudoInercias
% ------------------------------------------------------------------

Lon_1=[0,-0.02025,0];
J1=[0.000002425, 0, 0, m1*Lon_1(1);
                0, 0.000021985, 0, m1*Lon_1(2);
                0, 0, 0.000011495, m1*Lon_1(3);
    m1*Lon_1(1), m1*Lon_1(2), m1*Lon_1(3), m1];

Lon_2=[-0.0405,0,-0.00021792];
J2=[0.000452485, 5e-08, 6.5e-07, m2*Lon_2(1);
          5e-08, 0.000013385, 0, m2*Lon_2(2);
        6.5e-07, 0, 0.000028505, m2*Lon_2(3);
    m2*Lon_2(1), m2*Lon_2(2), m2*Lon_2(3), m2];

Lon_3=[-0.02715,0,1.106e-05];
J3=[ 0.00008172, 0, -3e-08, m3*Lon_3(1);
                0, 0.00000486, 0, m3*Lon_3(2);
          -3e-08, 0, 0.00002299, m3*Lon_3(3);
    m3*Lon_3(1), m3*Lon_3(2), m3*Lon_3(3), m3];
```

```
182
183 Lon_4=[-0.0174117,-6.2e-06,-6.606e-05];
184 J4=[0.000058775, 3e-08, 2.6e-07, m4*Lon_4(1);
185            3e-08, 0.000006695, 0, m4*Lon_4(2);
186          2.6e-07, 0, 0.000014235, m4*Lon_4(3);
187     m4*Lon_4(1), m4*Lon_4(2), m4*Lon_4(3), m4];
188
189 % ------------------------------------------------------------------------
190 % Matriz de inercia
191 % ------------------------------------------------------------------------
192 d11=trace(U11*J1*(U11.'))+...
193     trace(U21*J2*(U21.'))+...
194     trace(U31*J3*(U31.'))+...
195     trace(U41*J4*(U41.'));
196 d12=trace(U22*J2*(U21.'))+...
197     trace(U32*J3*(U31.'))+...
198     trace(U42*J4*(U41.'));
199 d13=trace(U33*J3*(U31.'))+...
200     trace(U43*J4*(U41.'));
201 d14=trace(U44*J4*(U41.'));
202
203 % Simplificación
204 d11=simplify(d11);
205 d12=simplify(d12);
206 d13=simplify(d13);
207 d14=simplify(d14);
208
209 d21=trace(U21*J2*(U22.'))+...
210     trace(U31*J3*(U32.'))+...
211     trace(U41*J4*(U42.'));
212 d22=trace(U22*J2*(U22.'))+...
213     trace(U32*J3*(U32.'))+...
214     trace(U42*J4*(U42.'));
215 d23=trace(U33*J3*(U32.'))+...
216     trace(U43*J4*(U42.'));
217 d24=trace(U44*J4*(U42.'));
218
219 % Simplificación
220 d21=simplify(d21);
221 d22=simplify(d22);
222 d23=simplify(d23);
223 d24=simplify(d24);
224
225 d31=trace(U31*J3*(U33.'))+...
226     trace(U41*J4*(U43.'));
227 d32=trace(U32*J3*(U33.'))+...
228     trace(U42*J4*(U43.'));
229 d33=trace(U33*J3*(U33.'))+...
230     trace(U43*J4*(U43.'));
231 d34=trace(U44*J4*(U43.'));
232
233 % Simplificación
234 d31=simplify(d31);
235 d32=simplify(d32);
236 d33=simplify(d33);
237 d34=simplify(d34);
238
239 d41=trace(U41*J4*(U44.'));
240 d42=trace(U42*J4*(U44.'));
241 d43=trace(U43*J4*(U44.'));
242 d44=trace(U44*J4*(U44.'));
243
244 % Simplificación
245 d41=simplify(d41);
```

```
d42=simplify(d42);
d43=simplify(d43);
d44=simplify(d44);

D=[d11 d12 d13 d14;...
   d21 d22 d23 d24;...
   d31 d32 d33 d34;...
   d41 d42 d43 d44];

disp('Matriz de inercias D = ');disp(vpa(D,5));

% ----------------------------------------------------------------
% Terminos hikm
% ----------------------------------------------------------------

% Primera sección
h111=trace(U111*J1*(U11.'))+...
     trace(U211*J2*(U21.'))+...
     trace(U311*J3*(U31.'))+...
     trace(U411*J4*(U41.'));
h112=trace(U212*J2*(U21.'))+...
     trace(U312*J3*(U31.'))+...
     trace(U412*J4*(U41.'));
h113=trace(U313*J3*(U31.'))+...
     trace(U413*J4*(U41.'));
h114=trace(U414*J4*(U41.'));

h121=trace(U221*J2*(U21.'))+...
     trace(U321*J3*(U31.'))+...
     trace(U421*J4*(U41.'));
h122=trace(U222*J2*(U21.'))+...
     trace(U322*J3*(U31.'))+...
     trace(U422*J4*(U41.'));
h123=trace(U323*J3*(U31.'))+...
     trace(U423*J4*(U41.'));
h124=trace(U424*J4*(U41.'));

h131=trace(U331*J3*(U31.'))+...
     trace(U431*J4*(U41.'));
h132=trace(U332*J3*(U31.'))+...
     trace(U432*J4*(U41.'));
h133=trace(U333*J3*(U31.'))+...
     trace(U433*J4*(U41.'));
h134=trace(U434*J4*(U41.'));

h141=trace(U441*J4*(U41.'));
h142=trace(U442*J4*(U41.'));
h143=trace(U443*J4*(U41.'));
h144=trace(U444*J4*(U41.'));

% Simplificación
h111=simplify(trace(U111*J1*(U11.'))+...
     trace(U211*J2*(U21.'))+...
     trace(U311*J3*(U31.'))+...
     trace(U411*J4*(U41.')));
h112=simplify(trace(U212*J2*(U21.'))+...
     trace(U312*J3*(U31.'))+...
     trace(U412*J4*(U41.')));
h113=simplify(trace(U313*J3*(U31.'))+...
     trace(U413*J4*(U41.')));
h114=simplify(trace(U414*J4*(U41.')));

h121=simplify(trace(U221*J2*(U21.'))+...
     trace(U321*J3*(U31.'))+...
```

```
310     trace(U421*J4*(U41.')));
311 h122=simplify(trace(U222*J2*(U21.'))+...
312     trace(U322*J3*(U31.'))+...
313     trace(U422*J4*(U41.')));
314 h123=simplify(trace(U323*J3*(U31.'))+...
315     trace(U423*J4*(U41.')));
316 h124=simplify(trace(U424*J4*(U41.')));
317
318 h131=simplify(trace(U331*J3*(U31.'))+...
319     trace(U431*J4*(U41.')));
320 h132=simplify(trace(U332*J3*(U31.'))+...
321     trace(U432*J4*(U41.')));
322 h133=simplify(trace(U333*J3*(U31.'))+...
323     trace(U433*J4*(U41.')));
324 h134=simplify(trace(U434*J4*(U41.')));
325
326 h141=simplify(trace(U441*J4*(U41.')));
327 h142=simplify(trace(U442*J4*(U41.')));
328 h143=simplify(trace(U443*J4*(U41.')));
329 h144=simplify(trace(U444*J4*(U41.')));
330
331 % Segunda sección
332 h211=trace(U211*J2*(U22.'))+...
333     trace(U311*J3*(U32.'))+...
334     trace(U411*J4*(U42.'));
335 h212=trace(U212*J2*(U22.'))+...
336     trace(U312*J3*(U32.'))+...
337     trace(U412*J4*(U42.'));
338 h213=trace(U313*J3*(U32.'))+...
339     trace(U413*J4*(U42.'));
340 h214=trace(U414*J4*(U42.'));
341
342 h221=trace(U221*J2*(U22.'))+...
343     trace(U321*J3*(U32.'))+...
344     trace(U421*J4*(U42.'));
345 h222=trace(U222*J2*(U22.'))+...
346     trace(U322*J3*(U32.'))+...
347     trace(U422*J4*(U42.'));
348 h223=trace(U323*J3*(U32.'))+...
349     trace(U423*J4*(U42.'));
350 h224=trace(U424*J4*(U42.'));
351
352 h231=trace(U331*J3*(U32.'))+...
353     trace(U431*J4*(U42.'));
354 h232=trace(U332*J3*(U32.'))+...
355     trace(U432*J4*(U42.'));
356 h233=trace(U333*J3*(U32.'))+...
357     trace(U433*J4*(U42.'));
358 h234=trace(U434*J4*(U42.'));
359
360 h241=trace(U441*J4*(U42.'));
361 h242=trace(U442*J4*(U42.'));
362 h243=trace(U443*J4*(U42.'));
363 h244=trace(U444*J4*(U42.'));
364
365 % Simplificación
366 h211=simplify(trace(U211*J2*(U22.'))+...
367     trace(U311*J3*(U32.'))+...
368     trace(U411*J4*(U42.')));
369 h212=simplify(trace(U212*J2*(U22.'))+...
370     trace(U312*J3*(U32.'))+...
371     trace(U412*J4*(U42.')));
372 h213=simplify(trace(U313*J3*(U32.'))+...
373     trace(U413*J4*(U42.')));
```

```
h214=simplify(trace(U414*J4*(U42.')));

h221=simplify(trace(U221*J2*(U22.'))+...
    trace(U321*J3*(U32.'))+...
    trace(U421*J4*(U42.')));
h222=simplify(trace(U222*J2*(U22.'))+...
    trace(U322*J3*(U32.'))+...
    trace(U422*J4*(U42.')));
h223=simplify(trace(U323*J3*(U32.'))+...
    trace(U423*J4*(U42.')));
h224=simplify(trace(U424*J4*(U42.')));

h231=simplify(trace(U331*J3*(U32.'))+...
    trace(U431*J4*(U42.')));
h232=simplify(trace(U332*J3*(U32.'))+...
    trace(U432*J4*(U42.')));
h233=simplify(trace(U333*J3*(U32.'))+...
    trace(U433*J4*(U42.')));
h234=simplify(trace(U434*J4*(U42.')));

h241=simplify(trace(U441*J4*(U42.')));
h242=simplify(trace(U442*J4*(U42.')));
h243=simplify(trace(U443*J4*(U42.')));
h244=simplify(trace(U444*J4*(U42.')));

% Tercera sección
h311=trace(U311*J3*(U33.'))+...
    trace(U411*J4*(U43.'));
h312=trace(U312*J3*(U33.'))+...
    trace(U412*J4*(U43.'));
h313=trace(U313*J3*(U33.'))+...
    trace(U413*J4*(U43.'));
h314=trace(U414*J4*(U43.'));

h321=trace(U321*J3*(U33.'))+...
    trace(U421*J4*(U43.'));
h322=trace(U322*J3*(U33.'))+...
    trace(U422*J4*(U43.'));
h323=trace(U323*J3*(U33.'))+...
    trace(U423*J4*(U43.'));
h324=trace(U424*J4*(U43.'));

h331=trace(U331*J3*(U33.'))+...
    trace(U431*J4*(U43.'));
h332=trace(U332*J3*(U33.'))+...
    trace(U432*J4*(U43.'));
h333=trace(U333*J3*(U33.'))+...
    trace(U433*J4*(U43.'));
h334=trace(U434*J4*(U43.'));

h341=trace(U441*J4*(U43.'));
h342=trace(U442*J4*(U43.'));
h343=trace(U443*J4*(U43.'));
h344=trace(U444*J4*(U43.'));

% Simplificación
h311=simplify(trace(U311*J3*(U33.'))+...
    trace(U411*J4*(U43.')));
h312=simplify(trace(U312*J3*(U33.'))+...
    trace(U412*J4*(U43.')));
h313=simplify(trace(U313*J3*(U33.'))+...
    trace(U413*J4*(U43.')));
h314=simplify(trace(U414*J4*(U43.')));

```

```
h321=simplify(trace(U321*J3*(U33.'))+...
    trace(U421*J4*(U43.')));
h322=simplify(trace(U322*J3*(U33.'))+...
    trace(U422*J4*(U43.')));
h323=simplify(trace(U323*J3*(U33.'))+...
    trace(U423*J4*(U43.')));
h324=simplify(trace(U424*J4*(U43.')));

h331=simplify(trace(U331*J3*(U33.'))+...
    trace(U431*J4*(U43.')));
h332=simplify(trace(U332*J3*(U33.'))+...
    trace(U432*J4*(U43.')));
h333=simplify(trace(U333*J3*(U33.'))+...
    trace(U433*J4*(U43.')));
h334=simplify(trace(U434*J4*(U43.')));

h341=simplify(trace(U441*J4*(U43.')));
h342=simplify(trace(U442*J4*(U43.')));
h343=simplify(trace(U443*J4*(U43.')));
h344=simplify(trace(U444*J4*(U43.')));

% Cuarta sección
h411=trace(U411*J4*(U44.'));
h412=trace(U412*J4*(U44.'));
h413=trace(U413*J4*(U44.'));
h414=trace(U414*J4*(U44.'));

h421=trace(U421*J4*(U44.'));
h422=trace(U422*J4*(U44.'));
h423=trace(U423*J4*(U44.'));
h424=trace(U424*J4*(U44.'));

h431=trace(U431*J4*(U44.'));
h432=trace(U432*J4*(U44.'));
h433=trace(U433*J4*(U44.'));
h434=trace(U434*J4*(U44.'));

h441=trace(U441*J4*(U44.'));
h442=trace(U442*J4*(U44.'));
h443=trace(U443*J4*(U44.'));
h444=trace(U444*J4*(U44.'));

% Simplificación
h411=simplify(trace(U411*J4*(U44.')));
h412=simplify(trace(U412*J4*(U44.')));
h413=simplify(trace(U413*J4*(U44.')));
h414=simplify(trace(U414*J4*(U44.')));

h421=simplify(trace(U421*J4*(U44.')));
h422=simplify(trace(U422*J4*(U44.')));
h423=simplify(trace(U423*J4*(U44.')));
h424=simplify(trace(U424*J4*(U44.')));

h431=simplify(trace(U431*J4*(U44.')));
h432=simplify(trace(U432*J4*(U44.')));
h433=simplify(trace(U433*J4*(U44.')));
h434=simplify(trace(U434*J4*(U44.')));

h441=simplify(trace(U441*J4*(U44.')));
h442=simplify(trace(U442*J4*(U44.')));
h443=simplify(trace(U443*J4*(U44.')));
h444=simplify(trace(U444*J4*(U44.')));

% ----------------------------------------------------------------
```

```
% Matriz de fuerzas de coriolisis y centripeta H
% ------------------------------------------------------------------------

h1=h111*qp(1)*qp(1)+ h112*qp(1)*qp(2)+ h113*qp(1)*qp(3)+ h114*qp(1)*qp(4)+ ...
   h121*qp(2)*qp(1)+ h122*qp(2)*qp(2)+ h123*qp(2)*qp(3)+ h124*qp(2)*qp(4)+ ...
   h131*qp(3)*qp(1)+ h132*qp(3)*qp(2)+ h133*qp(3)*qp(3)+ h134*qp(3)*qp(4)+ ...
   h141*qp(4)*qp(1)+ h142*qp(4)*qp(2)+ h143*qp(4)*qp(3)+ h144*qp(4)*qp(4);

h2=h211*qp(1)*qp(1)+ h212*qp(1)*qp(2)+ h213*qp(1)*qp(3)+ h214*qp(1)*qp(4)+ ...
   h221*qp(2)*qp(1)+ h222*qp(2)*qp(2)+ h223*qp(2)*qp(3)+ h224*qp(2)*qp(4)+ ...
   h231*qp(3)*qp(1)+ h232*qp(3)*qp(2)+ h233*qp(3)*qp(3)+ h234*qp(3)*qp(4)+ ...
   h241*qp(4)*qp(1)+ h242*qp(4)*qp(2)+ h243*qp(4)*qp(3)+ h244*qp(4)*qp(4);

h3=h311*qp(1)*qp(1)+ h312*qp(1)*qp(2)+ h313*qp(1)*qp(3)+ h314*qp(1)*qp(4)+ ...
   h321*qp(2)*qp(1)+ h322*qp(2)*qp(2)+ h323*qp(2)*qp(3)+ h324*qp(2)*qp(4)+ ...
   h331*qp(3)*qp(1)+ h332*qp(3)*qp(2)+ h333*qp(3)*qp(3)+ h334*qp(3)*qp(4)+ ...
   h341*qp(4)*qp(1)+ h342*qp(4)*qp(2)+ h343*qp(4)*qp(3)+ h344*qp(4)*qp(4);

h4=h411*qp(1)*qp(1)+ h412*qp(1)*qp(2)+ h413*qp(1)*qp(3)+ h414*qp(1)*qp(4)+ ...
   h421*qp(2)*qp(1)+ h422*qp(2)*qp(2)+ h423*qp(2)*qp(3)+ h424*qp(2)*qp(4)+ ...
   h431*qp(3)*qp(1)+ h432*qp(3)*qp(2)+ h433*qp(3)*qp(3)+ h434*qp(3)*qp(4)+ ...
   h441*qp(4)*qp(1)+ h442*qp(4)*qp(2)+ h443*qp(4)*qp(3)+ h444*qp(4)*qp(4);

H=[h1; h2; h3; h4];
H=simplify(H);
disp('Matriz de fuerzas coriolis y centripeta H=');disp(vpa(H,5));

% ------------------------------------------------------------------------
% Matriz columna de fuerzas de gravedad C
% ------------------------------------------------------------------------
r11=[Lon_1(1);Lon_1(2);Lon_1(3);1];
r22=[Lon_2(1);Lon_2(2);Lon_2(3);1];
r33=[Lon_3(1);Lon_3(2);Lon_3(3);1];
r44=[Lon_4(1);Lon_4(2);Lon_4(3);1];

C1=-masa(1)*vec_g*U11*r11-masa(2)*vec_g*U21*r22-masa(3)*vec_g*U31*r33-masa(4)*vec_g*U41*r44;
C2=-masa(1)*vec_g*U12*r11-masa(2)*vec_g*U22*r22-masa(3)*vec_g*U32*r33-masa(4)*vec_g*U42*r44;
C3=-masa(1)*vec_g*U13*r11-masa(2)*vec_g*U23*r22-masa(3)*vec_g*U33*r33-masa(4)*vec_g*U43*r44;
C4=-masa(1)*vec_g*U12*r11-masa(2)*vec_g*U24*r22-masa(3)*vec_g*U34*r33-masa(4)*vec_g*U44*r44;

C=[C1;C2;C3;C4];
C=simplify(C);
disp('Matriz columna de fruerzas de gravedad C=');disp(vpa(C,5));
```

Código 6.5: Algoritmo para el modelo dinámico inverso

```

function tao = Din_inv_MARA_4gdl(q_qp_qpp)

% ------------------------------------------------------------------------
% Declaración de vectores q, qp y qpp
% ------------------------------------------------------------------------
q1 = q_qp_qpp(1);
q2 = q_qp_qpp(2);
q3 = q_qp_qpp(3);
q4 = q_qp_qpp(4);

qp1 = q_qp_qpp(5);
qp2 = q_qp_qpp(6);
qp3 = q_qp_qpp(7);
qp4 = q_qp_qpp(8);

qpp1 = q_qp_qpp(9);
qpp2 = q_qp_qpp(10);
```

```
qpp3 = q_qp_qpp(11);
qpp4 = q_qp_qpp(12);

% ---------------------------------------------------------------------
% Vectores de velocidades y aceleraciones articulares
% ---------------------------------------------------------------------
qp=q_qp_qpp(5:8);
qpp=q_qp_qpp(9:12);

% ---------------------------------------------------------------------
% Parámetros y longitudes
% ---------------------------------------------------------------------
m1=0.045;
m2=0.2044;
m3=0.09;
m4=0.1022;
g=9.81;
% ---------------------------------------------------------------------
% Matrices del modelo dinámico
% ---------------------------------------------------------------------

D = [ Incluir en esta sección las matrices resultantes del código anterior];

H = [ Incluir en esta sección las matrices resultantes del código anterior ];

C = [ Incluir en esta sección las matrices resultantes del código anterior ];

Fv = [.50000e-1, 0., 0., 0.;...
              0., .50000e-1, 0., 0.;...
              0., 0., .50000e-1, 0.;...
              0., 0., 0., .50000e-1];

% ---------------------------------------------------------------------
% Fuerzas y pares de accionamientos
% ---------------------------------------------------------------------

tao = D*qpp + H + C + Fv*qp
```

Código 6.6: Algoritmo para el cálculo de la matriz Jacobiana

```


clear; close all; clc;

% Longitudes
LA=0.034;
LB=0.03170;
L1=LA+LB;
L2=0.081;
L3=0.055;
LC=0.040;
LD=0.052;
L4=LC;

L = [L1,L2,L3,L4];

% Coordenadas articulares
q = [0; 0; 0; 0];

% ---------------------------------------------------------------------
% Tabla de parámetros Denavit-Hartenberg
% ---------------------------------------------------------------------
    theta_DH = [ q(1)+pi/2, q(2), q(3), q(4) ];
    d_DH = [ L(1), 0, 0, 0 ];
```

```
    a_DH = [ 0, L(2), L(3), L(4) ];
    alpha_DH = [ pi/2, 0, 0, 0 ];

%Cálculo de las MTH
A01 = DH(theta_DH(1), d_DH(1), a_DH(1), alpha_DH(1));
A12 = DH(theta_DH(2), d_DH(2), a_DH(2), alpha_DH(2));
A23 = DH(theta_DH(3), d_DH(3), a_DH(3), alpha_DH(3));
A34 = DH(theta_DH(4), d_DH(4), a_DH(4), alpha_DH(4));

% MTH respecto al origen
A02 = A01*A12;
A03 = A02*A23;
A04 = A03*A34;

% Cálculo primera columna jacobiano
Z00 = [0; 0; 1];
P04 = A04(1:3,4);
J1 = [cross(Z00,P04); Z00];
% Cálculo segunda columna jacobiano
Z01 = A01(1:3,3);
P14 = A04(1:3,4) - A01(1:3,4);
J2 = [cross(Z01,P14); Z01];
% Cálculo tercera columna jacobiano
Z02 = A02(1:3,3);
P24 = A04(1:3,4) - A02(1:3,4);
J3 = [cross(Z02,P24); Z02];
% Cálculo cuarta columna jacobiano
Z03 = A03(1:3,3);
P34 = A04(1:3,4)-A03(1:3,4);
J4 = [cross(Z03,P34); Z03];

% Matriz Jacobiana
J = [J1, J2, J3, J4]

% Pseudoinversa por izquierda
M=J'*J;
L = inv(M)*J'

% Asignación de la variable Tao
Tao = [0.53;1.164;1.24;1.28]';

%Cálculo de la variable T
T = Tao*L
```

6.3 Propuesta de control cinemático

Código 6.7: Algoritmo para la simulacipon empleando interpoladores cúbicos

```

clear all
clc
close all

% Asignación de puntos para el primer muestreo
nn=20;

% Vector espaciado linealmente para cada coordenada
px = [linspace(-200,-200,nn),linspace(-200,-200,nn)];
py = [linspace(-50,0,nn),linspace(0,50,nn)];
pz = [linspace(80,30,nn),linspace(30,80,nn)];
```

```
13
14% Asignación de puntos para el segundo muestreo
15npuntos=10;
16
17% Asignación de variables
18tam = length(px);
19aux=tam-1;
20t = 0:aux;
21
22% Uso de la cinemática inversa para hallar los valores articulares
23for i=1:tam
24    CODO=-1;
25    angulo=0;
26    Px4=px(i);
27    Py4=py(i);
28    Pz4=pz(i);
29    q(i,:) = CI_Robot(Px4, Py4, Pz4,angulo,CODO)
30end
31
32% Vectores q_1, q_2, q_3 y q_4 a partir de q(i,:)
33q_1 =q(:, 1);
34q_2 =q(:, 2);
35q_3 =q(:, 3);
36q_4 =q(:, 4);
37
38% Uso de interpoladores cubicos para las trayectorias de las articulaciones
39tray1 = interpolador_cubico_dibujo(q_1,t,npuntos);title('Trayectoria q_1');
40f2 = figure(2);
41tray2 = interpolador_cubico_dibujo(q_2,t,npuntos);title('Trayectoria q_2');
42f3 = figure(3);
43tray3 = interpolador_cubico_dibujo(q_3,t,npuntos);title('Trayectoria q_3');
44f4 = figure(4);
45tray4 = interpolador_cubico_dibujo(q_4,t,npuntos);title('Trayectoria q_4');
46f5 = figure(5);
47
48% Actualización de los vectores q_1, q_2, q_3 y q_4
49% a partir de la trayectorias anteriores
50q_1 = tray1(:, 2);
51q_2 = tray2(:, 2);
52q_3 = tray3(:, 2);
53q_4 = tray4(:, 2);
54
55% Generacion de vectores tp con longitud de la dimensión de la matriz más grande
56% en q_1,q_2,q_3 y q_4
57tp1 = length(q_1);
58tp2 = length(q_2);
59tp3 = length(q_3);
60tp4 = length(q_4);
61
62% Matriz resultante conformada por q_1,q_2,q_3 y q_4
63M(:, 1) = q_1;
64M(:, 2) = q_2;
65M(:, 3) = q_3;
66M(:, 4) = q_4;
67
68% Figura que ilustra la ejecución del proceso del dibujo de letras
69f1 = figure(5); set(f1,'Color',[1, 1, 1]);
70
71% Bucle para generar el dibujo de tray1,tray2,tray3 y tray4
72for i=1:tp1
73    clc;
74    clf;
75
76    % Implementacion de la cinemática directa
```

```
    Q = M(i, :);
    A00 = eye(4);
    A04 = CD_Robot(Q)
    hold on;

    % Dibuja la trayectoria
    tray(i,:) = A04(1:3,4)';
    line(tray(:,1),tray(:,2),tray(:,3),'Color','r','LineWidth', 3);

    pause(0.00005);

    % Puntos utilizados para la trayectorias
    for j = 1:length(px);
    plot3(px(j), py(j),pz(j),'*','Color','y','LineWidth',1);
    hold on
    end
end

% Figura que ilustra la trayectoria objetivo y realizada del robot
f6 = figure(6);set(f6,'Color',[1, 1, 1]);grid on;
tray(i,:) = A04(1:3,4)';
line(tray(:,1),tray(:,2),tray(:,3),'Color','b','LineWidth',3);
line(px,py,pz,'Color','k','LineWidth',3);hold on;
```

6.4 Aplicación de realidad aumentada para teléfonos inteligentes y cámaras web

Código 6.8: Script para el eslabón 1

```
1
2 using System.Collections;
3 using System.Collections.Generic;
4 using UnityEngine;
5 using UnityEngine.UI;
6
7 public class S_Q1 : MonoBehaviour
8 {
9
10     private Slider Q1;
11     public float rotMinValue;
12     public float rotMaxValue;
13
14     void Start()
15     {
16
17         Q1 = GameObject.Find("Q1").GetComponent<Slider>();
18         Q1.minValue = 0;
19         Q1.maxValue = 360;
20
21         Q1.onValueChanged.AddListener(RotateSliderUpdate);
22
23     }
24
25     void RotateSliderUpdate(float value)
26     {
27         transform.localEulerAngles = new Vector3(transform.rotation.x, value, transform.rotation.z);
28     }
29 }
```

Código 6.9: Script para el eslabón 2

```
1
2 using System.Collections;
3 using System.Collections.Generic;
4 using UnityEngine;
5 using UnityEngine.UI;
6
7 public class S_Q2 : MonoBehaviour
8 {
9
10     private Slider Q2;
11     public float rotMinValue1;
12     public float rotMaxValue1;
13
14     void Start()
15     {
16
17         Q2 = GameObject.Find("Q2").GetComponent<Slider>();
18         Q2.minValue = 60;
19         Q2.maxValue = 285;
20
21         Q2.onValueChanged.AddListener(RotateSliderUpdate1);
22
23
24     }
25
26     void RotateSliderUpdate1(float value)
27     {
28         transform.localEulerAngles = new Vector3(transform.rotation.x, transform.rotation.y, value);
29     }
30 }
```

Código 6.10: Script para el eslabón 3

```
1
2 using System.Collections;
3 using System.Collections.Generic;
4 using UnityEngine;
5 using UnityEngine.UI;
6
7 public class S_Q3 : MonoBehaviour
8 {
9
10     private Slider Q3;
11
12     void Start()
13     {
14
15         Q3 = GameObject.Find("Q3").GetComponent<Slider>();
16         Q3.minValue = −120;
17         Q3.maxValue = 120;
18
19         Q3.onValueChanged.AddListener(RotateSliderUpdate2);
20
21     }
22
23     void RotateSliderUpdate2(float value)
24     {
25         transform.localEulerAngles = new Vector3(transform.rotation.x, transform.rotation.y, value);
26     }
27 }
```

Código 6.11: Script para el eslabón 4

```
1
2 using System.Collections;
3 using System.Collections.Generic;
4 using UnityEngine;
5 using UnityEngine.UI;
6
7 public class S_Q4 : MonoBehaviour
8 {
9
10     private Slider Q4;
11
12     void Start()
13     {
14
15         Q4 = GameObject.Find("Q4").GetComponent<Slider>();
16         Q4.minValue = 60;
17         Q4.maxValue = 300;
18
19         Q4.onValueChanged.AddListener(RotateSliderUpdate3);
20
21     }
22
23     void RotateSliderUpdate3(float value)
24     {
25         transform.localEulerAngles = new Vector3(transform.rotation.x, transform.rotation.y, value);
26     }
27 }
```

Código 6.12: Script para rotar el modelo 3D

```
1
2 using System.Collections;
3 using System.Collections.Generic;
4 using UnityEngine;
5
6 public class Button_Rot : MonoBehaviour
7 {
8     public GameObject objectRotate;
9
10     public float rotateSpeed = 50f;
11     bool rotateStatus = false;
12
13     public void RotateObject()
14     {
15
16         if (rotateStatus == false)
17         {
18             rotateStatus = true;
19         }
20         else
21         {
22             rotateStatus = false;
23         }
24     }
25
26     void Update()
27     {
28         if (rotateStatus == true)
29         {
30             //rotate object with speed
31             objectRotate.transform.Rotate(Vector3.up, rotateSpeed * Time.deltaTime);
32
```

```
        }
    }
}
```

Código 6.13: Script para cambiar el factor de escala

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;


public class Scale : MonoBehaviour
{

    private Slider scaleSlider;
    public float scaleMinValue;
    public float scaleMaxValue;


    void Start()
    {


        scaleSlider = GameObject.Find("Q5").GetComponent<Slider>();
        scaleSlider.minValue = scaleMinValue;
        scaleSlider.maxValue = scaleMaxValue;

        scaleSlider.onValueChanged.AddListener(ScaleSliderUpdate);

    }

    void ScaleSliderUpdate(float value)
    {
        transform.localScale = new Vector3(value, value, value);
    }
}
```

Código 6.14: Script para cambiar de escena

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class Escenas : MonoBehaviour
{
    public int escena;

    public void CambiarEscena()
    {
        SceneManager.LoadScene(escena);
    }
}
```

6.5 Transmisión de datos Unity-Matlab por TCP

Código 6.15: Algoritmo para la simulación del robot en Matlab

```
1
2 clear all
3 clc
4 close all
5
6 % ------------------------------------------------------------------------
7 % Declaración parámetros DH
8 % ------------------------------------------------------------------------
9 L1 = Revolute('d',0.6,'a',0,'alpha',pi/2);
10 L2 = Revolute('d',0,'a',0.8,'alpha',0);
11 L3 = Revolute('d',0,'a',0.8,'alpha',0);
12 L4 = Revolute('d',0,'a',0.9,'alpha',0);
13 robot = SerialLink([L1 L2 L3 L4]);
14
15 % ------------------------------------------------------------------------
16 % Configuración inicial para el robot
17 % ------------------------------------------------------------------------
18 joints = [0,0,0,0];
19 robot.plot(joints)
20 %robot.teach
21
22 % ------------------------------------------------------------------------
23 % Sistemas coordenados expresados en torno al eslabón 1
24 % ------------------------------------------------------------------------
25 F2 = robot.A(1:1,joints);
26 F3 = robot.A(1:2,joints);
27 F4 = robot.A(1:3,joints);
28 Fe = robot.A(1:4,joints);
29
30 % ------------------------------------------------------------------------
31 % Sistemas coordenados expresados en torno sistema al anterior
32 % ------------------------------------------------------------------------
33 L12 = robot.A(1,joints);
34 D12 = [L12.R, L12.t; 0, 0, 0, 1];
35 L23 = robot.A(2,joints);
36 D23 = [L23.R, L23.t; 0, 0, 0, 1];
37 L34 = robot.A(3,joints);
38 D34 = [L34.R, L34.t; 0, 0, 0, 1];
39 L4 = robot.A(4,joints);
40 D4 = [L4.R, L4.t; 0, 0, 0, 1];
41
42 % ------------------------------------------------------------------------
43 % Sistemas coordenados para diestros a zurdos
44 % ------------------------------------------------------------------------
45 D12_Unity = Con_M_to_U_Q(D12);
46 D23_Unity = Con_M_to_U_Q(D23);
47 T34_Unity = Con_M_to_U_Q(D34);
48 D4_Unity = Con_M_to_U_Q(D4);
49
50 % ------------------------------------------------------------------------
51 % Uso de cuaternios a través de la función Con_M_to_U_Q()
52 % ------------------------------------------------------------------------
53 [b12, v12, theta12] = Con_M_to_U_Q(D12);
54 [b23, v23, theta23] = Con_M_to_U_Q(D23);
55 [b34, v34, theta34] = Con_M_to_U_Q(D34);
56 [b4e, v4e, theta4e] = Con_M_to_U_Q(D4);
57
58 % ------------------------------------------------------------------------
59 % Configuración de MATLAB como servidor para Unity
60 % ------------------------------------------------------------------------
61 clear all
62 clc
63 close all
```

```
format compact

L1 = Revolute('d',0.6,'a',0,'alpha',pi/2);
L2 = Revolute('d',0,'a',0.8,'alpha',0);
L3 = Revolute('d',0,'a',0.8,'alpha',0);
L4 = Revolute('d',0,'a',0.9,'alpha',0);
robot = SerialLink([L1 L2 L3 L4]);

% ----------------------------------------------------------------------
% Configuración inicial para el robot
% ----------------------------------------------------------------------
joints = [0,0,0,0];
robot.plot(joints)
%robot.teach

% ----------------------------------------------------------------------
% Declaración y configuración para el servidor
% ----------------------------------------------------------------------
tcpipServer = tcpip('0.0.0.0', 55000,'NetworkRole','Server', 'InputBufferSize', 100);
i=0; J1=0; J2=0; J3=0; J4=0;
fopen(tcpipServer);

while(1)
    i = i+1;
    data = fread(tcpipServer, 45, 'char');
    dataChar = char(data);
    Ac = dataChar == 'A'; indxA = find(Ac); Ax = indxA(1);
    Bc = dataChar == 'B'; indxB = find(Bc); Bxx = indxB(indxB > Ax); Bx = Bxx(1);
    Cc = dataChar == 'C'; indxC = find(Cc); Cxx = indxC(indxC > Bx); Cx = Cxx(1);
    Dc = dataChar == 'D'; indxD = find(Dc); Dxx = indxD(indxD > Cx); Dx = Dxx(1);
    Ec = dataChar == 'E'; indxE = find(Ec); Exx = indxE(indxE > Dx); Ex = Exx(1);

    Jv1(i) = str2double(dataChar(Ax+1:Bx-1));
    Jv2(i) = str2double(dataChar(Bx+1:Cx-1));
    Jv3(i) = str2double(dataChar(Cx+1:Dx-1));
    Jv4(i) = str2double(dataChar(Dx+1:Ex-1));

    if (i>1 && (Jv1(i) ~= Jv1(i-1) || Jv2(i) ~= Jv2(i-1) || Jv3(i) ~= Jv3(i-1)|| Jv4(i) ~= Jv4(i-1)))
        if (Jv1(i) ~= Jv1(i-1))
            J1 = Jv1(i);
        end

        if (Jv2(i) ~= Jv2(i-1))
            J2 = Jv2(i);
        end

        if (Jv3(i) ~= Jv3(i-1))
            J3 = Jv3(i);
        end

        if (Jv4(i) ~= Jv4(i-1))
            J4 = Jv4(i);
        end

        robot.plot(joints+[J1*(pi/180), J2*(pi/180), J3*(pi/180), J4*(pi/180)]);

    end

    [J1, J2, J3, J4]

end

fclose(tcpipServer);
```

Código 6.16: Algoritmo para configurar y establecer la comunicación de las coordenadas articulares en Unity

```
1
2 using System.Collections;
3 using System.Collections.Generic;
4 using UnityEngine;
5
6 public class RobotScript : MonoBehaviour {
7
8     public GameObject B;
9     public GameObject B1;
10    public float B1Angle;
11    public GameObject B2;
12    public float B2Angle;
13    public GameObject B3;
14    public float B3Angle;
15
16    [HideInInspector] public Vector3 eePosition;
17
18
19    void Start () {
20
21        B1.transform.GetChild(1).transform.localPosition = new Vector3(0f, 0.2f, 0f);
22        B1.transform.GetChild(1).transform.localRotation = Quaternion.AngleAxis(−90, new Vector3(1,0,0));
23        B2.transform.GetChild(1).transform.localPosition = new Vector3(0f, 0.2f, 0f);
24        B2.transform.GetChild(1).transform.localRotation = Quaternion.AngleAxis(−120, new Vector3(0.5774f,↩
              ↪ 0.5774f, 0.5774f));
25        B3.transform.GetChild(1).transform.localPosition = new Vector3(0f, 0.2f, 0f);
26        B3.transform.GetChild(1).transform.localRotation = Quaternion.AngleAxis(−90, new Vector3(1, 0, 0));
27    }
28
29
30    void Update () {
31
32        B1.transform.position = B.transform.position;
33        B1.transform.localRotation = Quaternion.AngleAxis(−B1Angle, new Vector3(0, 1, 0));
34
35        B2.transform.position = B1.transform.GetChild(1).transform.position;
36        B2.transform.rotation = B1.transform.GetChild(1).transform.rotation;
37        var B2Q = B2.transform.localRotation;
38        B2.transform.localRotation = B2Q ∗ Quaternion.AngleAxis(−B2Angle, new Vector3(0, 1, 0));
39
40        B3.transform.position = B2.transform.GetChild(1).transform.position;
41        B3.transform.rotation = B2.transform.GetChild(1).transform.rotation;
42        var B3Q = B3.transform.localRotation;
43        B3.transform.localRotation = B3Q ∗ Quaternion.AngleAxis(−B3Angle, new Vector3(0, 1, 0));
44
45    }
46 }
```

Código 6.17: Algoritmo par configurar el cliente en Unity

```
1
2 using System.Collections;
3 using System.Collections.Generic;
4 using UnityEngine;
5 using System.Net;
6 using System.Net.Sockets;
7 using System;
8 using System.IO;
9 using System.Text;
10
11 public class Client : MonoBehaviour
12 {
```

```
13
14      TcpClient mySocket;
15      String Host = "localhost";
16      Int32 Port = 55000;
17
18      void Start()
19      {
20          mySocket = new TcpClient(Host, Port);
21
22          Debug.Log("socket is set up");
23      }
24
25      void Update()
26      {
27          GameObject robot = GameObject.Find("Robot");
28          RobotScript robotIns = robot.GetComponent<RobotScript>();
29
30          try
31          {
32              Debug.Log(robotIns.B1Angle);
33              Byte[] sendBytes = Encoding.UTF8.GetBytes("A" + (robotIns.B1Angle).ToString() + "B"
34                  + (robotIns.B2Angle).ToString() + "C" + (robotIns.B3Angle).ToString() + "D");
35
36              mySocket.GetStream().Write(sendBytes, 0, sendBytes.Length);
37          }
38          catch (Exception e)
39          {
40              Debug.Log("Socket error: " + e);
41          }
42      }
43  }
```

6.6 Transmisión de datos Matlab-Unity por UDP

Código 6.18: Algoritmo para generar la GUI en Matlab

```
1
2 function varargout = Simulation(varargin)
3 % SIMULATION MATLAB code for Simulation.fig
4 % SIMULATION, by itself, creates a new SIMULATION or raises the existing
5 % singleton*.
6 %
7 % H = SIMULATION returns the handle to a new SIMULATION or the handle to
8 % the existing singleton*.
9 %
10 % SIMULATION('CALLBACK',hObject,eventData,handles,...) calls the local
11 % function named CALLBACK in SIMULATION.M with the given input arguments.
12 %
13 % SIMULATION('Property','Value',...) creates a new SIMULATION or raises the
14 % existing singleton*. Starting from the left, property value pairs are
15 % applied to the GUI before Simulation_OpeningFcn gets called. An
16 % unrecognized property name or invalid value makes property application
17 % stop. All inputs are passed to Simulation_OpeningFcn via varargin.
18 %
19 % *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
20 % instance to run (singleton)".
21 %
22 % See also: GUIDE, GUIDATA, GUIHANDLES
23
24 % Edit the above text to modify the response to help Simulation
25
```

```
% Last Modified by GUIDE v2.5 30-Nov-2021 13:50:13

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name', mfilename, ...
                   'gui_Singleton', gui_Singleton, ...
                   'gui_OpeningFcn', @Simulation_OpeningFcn, ...
                   'gui_OutputFcn', @Simulation_OutputFcn, ...
                   'gui_LayoutFcn', [] , ...
                   'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Simulation is made visible.
function Simulation_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% varargin command line arguments to Simulation (see VARARGIN)

% Choose default command line output for Simulation
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Simulation wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Simulation_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in Forward.
function Forward_Callback(hObject, eventdata, handles)
% hObject handle to Forward (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
global V1 V2 qa
V1=-129;
V2=44;
cla;
view(V1,V2)
% camlight(50,40)
lighting phong
grid on;
```

```
axis equal
cla
q1=str2double(get(handles.theta1,'string'));
q2=str2double(get(handles.theta2,'string'));
q3=str2double(get(handles.theta3,'string'));
q4=str2double(get(handles.theta4,'string'));
q=[q1*pi/180,q2*pi/180,q3*pi/180,q4*pi/180];
qa=[q1;q2;q3;q4];
CinD_Robot(q);
unity_send(qa)


% --- Executes on button press in Inverse.
function Inverse_Callback(hObject, eventdata, handles)
% hObject handle to Inverse (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
global V1 V2
V1=-129;
V2=44;
cla;
view(V1,V2)
% camlight(50,40)
lighting phong
grid on;
axis equal

Px_1=str2double(get(handles.Px,'string'));
Py_1=str2double(get(handles.Py,'string'));
Pz_1=str2double(get(handles.Pz,'string'));
angle=str2double(get(handles.b,'string'));
CODO=-1;
Z00=RotY(pi/2)*eye(4);
P00 = [Px_1,Py_1,Pz_1,1]';
Pi = inv(Z00)*P00;
Px4=Pi(1);
Py4=Pi(2);
Pz4=Pi(3);
q=CinI_Robot(Px4, Py4, Pz4,angle,CODO);
qb=[q(1)*(180/pi);q(2)*(180/pi);q(3)*(180/pi);q(4)*(180/pi)];
unity_send(qb)

set(handles.theta1,'string',num2str(qb(1)));
set(handles.theta2,'string',num2str(qb(2)));
set(handles.theta3,'string',num2str(qb(3)));
set(handles.theta4,'string',num2str(qb(4)));



function theta1_Callback(hObject, eventdata, handles)
% hObject handle to theta1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of theta1 as text
% str2double(get(hObject,'String')) returns contents of theta1 as a double


% --- Executes during object creation, after setting all properties.
function theta1_CreateFcn(hObject, eventdata, handles)
% hObject handle to theta1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called
```

```
154
155 % Hint: edit controls usually have a white background on Windows.
156 % See ISPC and COMPUTER.
157 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
158     set(hObject,'BackgroundColor','white');
159 end
160
161
162
163 function theta2_Callback(hObject, eventdata, handles)
164 % hObject handle to theta2 (see GCBO)
165 % eventdata reserved - to be defined in a future version of MATLAB
166 % handles structure with handles and user data (see GUIDATA)
167
168 % Hints: get(hObject,'String') returns contents of theta2 as text
169 % str2double(get(hObject,'String')) returns contents of theta2 as a double
170
171
172 % --- Executes during object creation, after setting all properties.
173 function theta2_CreateFcn(hObject, eventdata, handles)
174 % hObject handle to theta2 (see GCBO)
175 % eventdata reserved - to be defined in a future version of MATLAB
176 % handles empty - handles not created until after all CreateFcns called
177
178 % Hint: edit controls usually have a white background on Windows.
179 % See ISPC and COMPUTER.
180 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
181     set(hObject,'BackgroundColor','white');
182 end
183
184
185
186 function theta3_Callback(hObject, eventdata, handles)
187 % hObject handle to theta3 (see GCBO)
188 % eventdata reserved - to be defined in a future version of MATLAB
189 % handles structure with handles and user data (see GUIDATA)
190
191 % Hints: get(hObject,'String') returns contents of theta3 as text
192 % str2double(get(hObject,'String')) returns contents of theta3 as a double
193
194
195 % --- Executes during object creation, after setting all properties.
196 function theta3_CreateFcn(hObject, eventdata, handles)
197 % hObject handle to theta3 (see GCBO)
198 % eventdata reserved - to be defined in a future version of MATLAB
199 % handles empty - handles not created until after all CreateFcns called
200
201 % Hint: edit controls usually have a white background on Windows.
202 % See ISPC and COMPUTER.
203 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
204     set(hObject,'BackgroundColor','white');
205 end
206
207
208
209 function theta4_Callback(hObject, eventdata, handles)
210 % hObject handle to theta4 (see GCBO)
211 % eventdata reserved - to be defined in a future version of MATLAB
212 % handles structure with handles and user data (see GUIDATA)
213
214 % Hints: get(hObject,'String') returns contents of theta4 as text
215 % str2double(get(hObject,'String')) returns contents of theta4 as a double
216
217
```

```
% --- Executes during object creation, after setting all properties.
function theta4_CreateFcn(hObject, eventdata, handles)
% hObject handle to theta4 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function Px_Callback(hObject, eventdata, handles)
% hObject handle to Px (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Px as text
% str2double(get(hObject,'String')) returns contents of Px as a double


% --- Executes during object creation, after setting all properties.
function Px_CreateFcn(hObject, eventdata, handles)
% hObject handle to Px (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function Py_Callback(hObject, eventdata, handles)
% hObject handle to Py (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Py as text
% str2double(get(hObject,'String')) returns contents of Py as a double


% --- Executes during object creation, after setting all properties.
function Py_CreateFcn(hObject, eventdata, handles)
% hObject handle to Py (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function Pz_Callback(hObject, eventdata, handles)
% hObject handle to Pz (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
```

```
282
283 % Hints: get(hObject,'String') returns contents of Pz as text
284 % str2double(get(hObject,'String')) returns contents of Pz as a double
285
286
287 % --- Executes during object creation, after setting all properties.
288 function Pz_CreateFcn(hObject, eventdata, handles)
289 % hObject handle to Pz (see GCBO)
290 % eventdata reserved - to be defined in a future version of MATLAB
291 % handles empty - handles not created until after all CreateFcns called
292
293 % Hint: edit controls usually have a white background on Windows.
294 % See ISPC and COMPUTER.
295 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
296     set(hObject,'BackgroundColor','white');
297 end
298
299
300 % --- Executes on slider movement.
301 function slider1_Callback(hObject, eventdata, handles)
302 % hObject handle to slider1 (see GCBO)
303 % eventdata reserved - to be defined in a future version of MATLAB
304 % handles structure with handles and user data (see GUIDATA)
305 global V1 V2
306 V2=get(handles.slider1,'value');
307 view(V1,V2);
308
309 % Hints: get(hObject,'Value') returns position of slider
310 % get(hObject,'Min') and get(hObject,'Max') to determine range of slider
311
312
313 % --- Executes during object creation, after setting all properties.
314 function slider1_CreateFcn(hObject, ~, handles)
315 % hObject handle to slider1 (see GCBO)
316 % eventdata reserved - to be defined in a future version of MATLAB
317 % handles empty - handles not created until after all CreateFcns called
318
319 % Hint: slider controls usually have a light gray background.
320 if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
321     set(hObject,'BackgroundColor',[.9 .9 .9]);
322 end
323
324
325 % --- Executes on slider movement.
326 function slider2_Callback(hObject, eventdata, handles)
327 % hObject handle to slider2 (see GCBO)
328 % eventdata reserved - to be defined in a future version of MATLAB
329 % handles structure with handles and user data (see GUIDATA)
330
331 % Hints: get(hObject,'Value') returns position of slider
332 % get(hObject,'Min') and get(hObject,'Max') to determine range of slider
333
334 global V1 V2
335 V1=get(handles.slider2,'value');
336 view(V1,V2);
337
338 % --- Executes during object creation, after setting all properties.
339 function slider2_CreateFcn(hObject, eventdata, handles)
340 % hObject handle to slider2 (see GCBO)
341 % eventdata reserved - to be defined in a future version of MATLAB
342 % handles empty - handles not created until after all CreateFcns called
343
344 % Hint: slider controls usually have a light gray background.
345 if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
```

```
346     set(hObject,'BackgroundColor',[.9 .9 .9]);
347 end
348
349
350
351 function b_Callback(hObject, eventdata, handles)
352 % hObject handle to b (see GCBO)
353 % eventdata reserved - to be defined in a future version of MATLAB
354 % handles structure with handles and user data (see GUIDATA)
355
356 % Hints: get(hObject,'String') returns contents of b as text
357 % str2double(get(hObject,'String')) returns contents of b as a double
358
359
360 % --- Executes during object creation, after setting all properties.
361 function b_CreateFcn(hObject, ~, handles)
362 % hObject handle to b (see GCBO)
363 % eventdata reserved - to be defined in a future version of MATLAB
364 % handles empty - handles not created until after all CreateFcns called
365
366 % Hint: edit controls usually have a white background on Windows.
367 % See ISPC and COMPUTER.
368 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
369     set(hObject,'BackgroundColor','white');
370 end
371
372
373
374 function edit10_Callback(hObject, eventdata, handles)
375 % hObject handle to b (see GCBO)
376 % eventdata reserved - to be defined in a future version of MATLAB
377 % handles structure with handles and user data (see GUIDATA)
378
379 % Hints: get(hObject,'String') returns contents of b as text
380 % str2double(get(hObject,'String')) returns contents of b as a double
381
382
383 % --- Executes during object creation, after setting all properties.
384 function edit10_CreateFcn(hObject, eventdata, handles)
385 % hObject handle to b (see GCBO)
386 % eventdata reserved - to be defined in a future version of MATLAB
387 % handles empty - handles not created until after all CreateFcns called
388
389 % Hint: edit controls usually have a white background on Windows.
390 % See ISPC and COMPUTER.
391 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
392     set(hObject,'BackgroundColor','white');
393 end
394
395
396 % --- Executes on button press in Clear.
397 function Clear_Callback(hObject, eventdata, handles)
398 % hObject handle to Clear (see GCBO)
399 % eventdata reserved - to be defined in a future version of MATLAB
400 % handles structure with handles and user data (see GUIDATA)
401 cla
402 set(handles.Px,'string','');
403 set(handles.Py,'string','');
404 set(handles.Pz,'string','');
405 set(handles.b,'string','');
406 set(handles.theta1,'string','');
407 set(handles.theta2,'string','');
408 set(handles.theta3,'string','');
409 set(handles.theta4,'string','');
```

```
410
411
412 % --- Executes on selection change in popupmenu2.
413 function popupmenu2_Callback(hObject, eventdata, handles)
414 % hObject handle to popupmenu2 (see GCBO)
415 % eventdata reserved - to be defined in a future version of MATLAB
416 % handles structure with handles and user data (see GUIDATA)
417
418 % Hints: contents = cellstr(get(hObject,'String')) returns popupmenu2 contents as cell array
419 % contents{get(hObject,'Value')} returns selected item from popupmenu2
420 str = get(handles.popupmenu2, 'string');
421 val = get(handles.popupmenu2, 'value');
422
423 switch str{val}
424     case 'Option 1'
425     camlight(70,10)
426     case 'Option 2'
427     camlight(-50,-10)
428     case 'Option 3'
429     camlight(30,40)
430 end
431
432
433 % --- Executes during object creation, after setting all properties.
434 function popupmenu2_CreateFcn(hObject, eventdata, handles)
435 % hObject handle to popupmenu2 (see GCBO)
436 % eventdata reserved - to be defined in a future version of MATLAB
437 % handles empty - handles not created until after all CreateFcns called
438
439 % Hint: popupmenu controls usually have a white background on Windows.
440 % See ISPC and COMPUTER.
441 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
442     set(hObject,'BackgroundColor','white');
443 end
```

Código 6.19: Función para el envío de datos a Unity

```
1
2 function trama=unity_send(q)
3 q1=q(1);
4 q2=q(2);
5 q3=q(3);
6 q4=q(4);
7 a1=num2str(q1',3);
8 a2=num2str(q2',3);
9 a3=num2str(q3',3);
10 a4=num2str(q4',3);
11 trama = strcat(a1,',', a2,',', a3,',', a4)
12 judp('SEND',8050,'127.0.0.1',int8(trama));
```

Código 6.20: Algoritmo para la recepción de las coordenadas articulares en Unity

```
1
2 using System.Collections;
3 using System.Collections.Generic;
4 using UnityEngine;
5 using System;
6 using System.Text;
7 using System.Net;
8 using System.Net.Sockets;
9 using System.Threading;
10 using UnityEngine.Networking;
11 using System.Globalization;
```

```
12
13 public class test_1 : MonoBehaviour
14 {
15     //========== Variables ==========//
16
17     public GameObject Joint_1;
18     public float B1Angle = 0.0f;
19     private float q1_m = 0.0f;
20     private float q1_r = 0.0f;
21
22     public GameObject Joint_2;
23     public float B2Angle = 0.0f;
24     private float q2_m = 0.0f;
25     private float q2_r = 0.0f;
26
27     public GameObject Joint_3;
28     public float B3Angle = 0.0f;
29     private float q3_m = 0.0f;
30     private float q3_r = 0.0f;
31
32     public GameObject Joint_4;
33     public float B4Angle = 0.0f;
34     private float q4_m = 0.0f;
35     private float q4_r = 0.0f;
36
37
38     //========== Configuración UDP ==========//
39     Thread receiveThread;
40
41
42     UdpClient client;
43
44
45     public string IP;
46     public int port;
47     IPEndPoint remoteEndPoint;
48
49
50     private static void Main()
51     {
52         test_1 receiveObj = new test_1();
53         receiveObj.init();
54
55         string text = "";
56         do
57         {
58             text = Console.ReadLine();
59         }
60         while (!text.Equals("exit"));
61     }
62     void Start()
63     {
64         init();
65     }
66
67     void OnGUI()
68     {
69         MoveRobot();
70     }
71
72
73     private void init()
74     {
75         print("UDPSend.init()");
```

```
76
77          port = 8050;
78          remoteEndPoint = new IPEndPoint(IPAddress.Parse(IP), port);
79
80          print("Sending to 127.0.0.1 : " + port);
81          print("Test-Sending to this Port: nc -u 127.0.0.1 " + port + "");
82
83          receiveThread = new Thread(
84              new ThreadStart(ReceiveData));
85          receiveThread.IsBackground = true;
86          receiveThread.Start();
87
88      }
89
90      private void MoveRobot()
91      {
92          //========= Eslabones ==========//
93          q1_r = B1Angle - q1_m;
94          Joint_1.transform.Rotate(0.0f, -q1_r, 0.0f);
95          q1_m = B1Angle;
96
97
98          q2_r = B2Angle - q2_m;
99          Joint_2.transform.Rotate(0.0f, 0.0f, -q2_r);
100         q2_m = B2Angle;
101
102
103         q3_r = B3Angle - q3_m;
104         Joint_3.transform.Rotate(0.0f, 0.0f, -q3_r);
105         q3_m = B3Angle;
106
107
108         q4_r = B4Angle - q4_m;
109         Joint_4.transform.Rotate(0.0f, -q4_r, 0.0f);
110         q4_m = B4Angle;
111
112     }
113     private void ReceiveData()
114     {
115
116         client = new UdpClient(port);
117         while (true)
118         {
119
120             try
121             {
122                 byte[] data = client.Receive(ref remoteEndPoint);
123                 string text = Encoding.UTF8.GetString(data);
124                 char[] ident = { ',', ',', ',', ',' };
125                 Int16 cot = 4;
126                 string[] matriz = text.Split(ident, cot);
127                 B1Angle = float.Parse(matriz[0], CultureInfo.InvariantCulture);
128                 B2Angle = float.Parse(matriz[1], CultureInfo.InvariantCulture);
129                 B3Angle = float.Parse(matriz[2], CultureInfo.InvariantCulture);
130                 B4Angle = float.Parse(matriz[3], CultureInfo.InvariantCulture);
131
132                 print(">> " + text);
133
134             }
135             catch (Exception err)
136             {
137                 print(err.ToString());
138             }
139         }
```

```
140     }
141
142 }
```

6.7 Pruebas de funcionamiento en Arduino IDE

Código 6.21: Algoritmo para la validación de referencias de posición

```
1
2 // Fuente: José Baca, Department of Engineering | Texas A$\&$M University−Corpus Christi
3
4 #include <DynamixelShield.h>
5
6 #if defined(ARDUINO_AVR_UNO) || defined(ARDUINO_AVR_MEGA2560)
7   #include <SoftwareSerial.h>
8   SoftwareSerial soft_serial(7, 8); // DYNAMIXELShield UART RX/TX
9   #define DEBUG_SERIAL soft_serial
10 #elif defined(ARDUINO_SAM_DUE) || defined(ARDUINO_SAM_ZERO)
11   #define DEBUG_SERIAL SerialUSB
12 #else
13   #define DEBUG_SERIAL Serial
14 #endif
15
16 const uint8_t baseMotor = 5;
17 const uint8_t bottomMotor = 6;
18 const uint8_t middleMotor = 7;
19 const uint8_t topMotor = 8;
20 const float DXL_PROTOCOL_VERSION = 2.0;
21
22 DynamixelShield dxl;
23
24 //This namespace is required to use Control table item names
25 using namespace ControlTableItem;
26
27 void setup() {
28   // put your setup code here, to run once:
29
30   // For Uno, Nano, Mini, and Mega, use UART port of DYNAMIXEL Shield to debug.
31   DEBUG_SERIAL.begin(115200);
32
33   // Set Port baudrate to 57600bps. This has to match with DYNAMIXEL baudrate.
34   dxl.begin(57600);
35   // Set Port Protocol Version. This has to match with DYNAMIXEL protocol version.
36   dxl.setPortProtocolVersion(DXL_PROTOCOL_VERSION);
37   // Get DYNAMIXEL information
38   dxl.ping(baseMotor);
39   dxl.ping(bottomMotor);
40   dxl.ping(middleMotor);
41   dxl.ping(topMotor);
42
43
44   positionCon();
45
46 }
47
48 void velocityCon(){
49   dxl.torqueOff(baseMotor);
50   dxl.setOperatingMode(baseMotor, OP_VELOCITY);
51   dxl.torqueOn(baseMotor);
52
53   dxl.torqueOff(bottomMotor);
```

```
  dxl.setOperatingMode(bottomMotor, OP_VELOCITY);
  dxl.torqueOn(bottomMotor);

  dxl.torqueOff(middleMotor);
  dxl.setOperatingMode(middleMotor, OP_VELOCITY);
  dxl.torqueOn(middleMotor);

  dxl.torqueOff(topMotor);
  dxl.setOperatingMode(topMotor, OP_VELOCITY);
  dxl.torqueOn(topMotor);
  }

void positionCon(){
  dxl.torqueOff(baseMotor);
  dxl.setOperatingMode(baseMotor, OP_POSITION);
  dxl.torqueOn(baseMotor);

  dxl.torqueOff(bottomMotor);
  dxl.setOperatingMode(bottomMotor, OP_POSITION);
  dxl.torqueOn(bottomMotor);

  dxl.torqueOff(middleMotor);
  dxl.setOperatingMode(middleMotor, OP_POSITION);
  dxl.torqueOn(middleMotor);

  dxl.torqueOff(topMotor);
  dxl.setOperatingMode(topMotor, OP_POSITION);
  dxl.torqueOn(topMotor);

  }

void loop() {
  dxl.setGoalPosition(baseMotor, 2048); //Q1 M1 2048 180 (90)
  dxl.setGoalPosition(bottomMotor, 1536); //Q2 M2 1526 (180−(44.15)) CW or CCW
  dxl.setGoalPosition(middleMotor, 2560); //Q3 M3 1543 (180+(−44.37)) CW or CCW
  dxl.setGoalPosition(topMotor, 1536); //Q4 M4 1529 (180+(−45.61) CW or CCW
  delay(5000);

}
```

Código 6.22: Algoritmo para la validación de referencias de velocidad

```

// Fuente: José Baca, Department of Engineering | Texas A$\&$M University−Corpus Christi

#include <DynamixelShield.h>

#if defined(ARDUINO_AVR_UNO) || defined(ARDUINO_AVR_MEGA2560)
  #include <SoftwareSerial.h>
  SoftwareSerial soft_serial(7, 8); // DYNAMIXELShield UART RX/TX
  #define DEBUG_SERIAL soft_serial
#elif defined(ARDUINO_SAM_DUE) || defined(ARDUINO_SAM_ZERO)
  #define DEBUG_SERIAL SerialUSB
#else
  #define DEBUG_SERIAL Serial
#endif

const uint8_t baseMotor = 1;
const uint8_t bottomMotor = 2;
const uint8_t middleMotor = 3;
const uint8_t topMotor = 4;
const float DXL_PROTOCOL_VERSION = 2.0;

DynamixelShield dxl;
```

```
23
24 //This namespace is required to use Control table item names
25 using namespace ControlTableItem;
26
27 void setup() {
28   // put your setup code here, to run once:
29
30   // For Uno, Nano, Mini, and Mega, use UART port of DYNAMIXEL Shield to debug.
31   DEBUG_SERIAL.begin(115200);
32
33   // Set Port baudrate to 57600bps. This has to match with DYNAMIXEL baudrate.
34   dxl.begin(57600);
35   // Set Port Protocol Version. This has to match with DYNAMIXEL protocol version.
36   dxl.setPortProtocolVersion(DXL_PROTOCOL_VERSION);
37   // Get DYNAMIXEL information
38   dxl.ping(baseMotor);
39   dxl.ping(bottomMotor);
40   dxl.ping(middleMotor);
41   dxl.ping(topMotor);
42
43   // Turn off torque when configuring items in EEPROM area
44   positionCon();
45   homePos();
46   velocityCon();
47
48 }
49
50 void velocityCon(){
51   dxl.torqueOff(baseMotor);
52   dxl.setOperatingMode(baseMotor, OP_VELOCITY);
53   dxl.torqueOn(baseMotor);
54
55   dxl.torqueOff(bottomMotor);
56   dxl.setOperatingMode(bottomMotor, OP_VELOCITY);
57   dxl.torqueOn(bottomMotor);
58
59   dxl.torqueOff(middleMotor);
60   dxl.setOperatingMode(middleMotor, OP_VELOCITY);
61   dxl.torqueOn(middleMotor);
62
63   dxl.torqueOff(topMotor);
64   dxl.setOperatingMode(topMotor, OP_VELOCITY);
65   dxl.torqueOn(topMotor);
66   }
67
68 void positionCon(){
69   dxl.torqueOff(baseMotor);
70   dxl.setOperatingMode(baseMotor, OP_POSITION);
71   dxl.torqueOn(baseMotor);
72
73   dxl.torqueOff(bottomMotor);
74   dxl.setOperatingMode(bottomMotor, OP_POSITION);
75   dxl.torqueOn(bottomMotor);
76
77   dxl.torqueOff(middleMotor);
78   dxl.setOperatingMode(middleMotor, OP_POSITION);
79   dxl.torqueOn(middleMotor);
80
81   dxl.torqueOff(topMotor);
82   dxl.setOperatingMode(topMotor, OP_POSITION);
83   dxl.torqueOn(topMotor);
84   }
85
86 void homePos(){
```

```
87   dxl.setGoalPosition(baseMotor, 2048);
88   dxl.setGoalPosition(bottomMotor, 2048);
89   dxl.setGoalPosition(middleMotor, 2048);
90   dxl.setGoalPosition(topMotor, 2048);
91   delay(1000);
92   }
93
94 void swim(){
95      dxl.setGoalVelocity(bottomMotor,-30);
96      dxl.setGoalVelocity(middleMotor,30);
97      dxl.setGoalVelocity(topMotor,-30);
98      delay(750);
99
100     dxl.setGoalVelocity(bottomMotor,30);
101     dxl.setGoalVelocity(middleMotor,-30);
102     dxl.setGoalVelocity(topMotor,30);
103     delay(750);
104 }
105
106 void swim2(){
107     dxl.setGoalVelocity(bottomMotor,-45);
108     dxl.setGoalVelocity(middleMotor,45);
109     dxl.setGoalVelocity(topMotor,-45);
110     delay(500);
111
112     dxl.setGoalVelocity(bottomMotor,45);
113     dxl.setGoalVelocity(middleMotor,-45);
114     dxl.setGoalVelocity(topMotor,45);
115     delay(500);
116 }
117
118 void swim3(){
119     dxl.setGoalVelocity(middleMotor,70);
120     dxl.setGoalVelocity(topMotor,-90);
121     delay(440);
122     dxl.setGoalVelocity(middleMotor,-70);
123     dxl.setGoalVelocity(topMotor,90);
124     delay(440);
125 }
126
127 void loop() {
128   // put your main code here, to run repeatedly:
129   dxl.setGoalVelocity(bottomMotor,30);
130   dxl.setGoalVelocity(middleMotor,-30);
131   dxl.setGoalVelocity(topMotor,30);
132   delay(375);
133   for (int i = 0; i < 7; i++){
134   swim();
135   }
136   dxl.setGoalVelocity(bottomMotor,-30);
137   dxl.setGoalVelocity(middleMotor,30);
138   dxl.setGoalVelocity(topMotor,-30);
139   delay(375);
140   dxl.setGoalVelocity(bottomMotor,0);
141   dxl.setGoalVelocity(middleMotor,0);
142   dxl.setGoalVelocity(topMotor,0);
143   delay(750);
144   dxl.setGoalVelocity(bottomMotor,45);
145   dxl.setGoalVelocity(middleMotor,-45);
146   dxl.setGoalVelocity(topMotor,45);
147   delay(250);
148   for (int i = 0; i < 4; i++){
149   swim2();
150   }
```

```
151   dxl.setGoalVelocity(bottomMotor,−45);
152   dxl.setGoalVelocity(middleMotor,45);
153   dxl.setGoalVelocity(topMotor,−45);
154   delay(250);
155   dxl.setGoalVelocity(bottomMotor,0);
156   dxl.setGoalVelocity(middleMotor,0);
157   dxl.setGoalVelocity(topMotor,0);
158   delay(750);
159   dxl.setGoalVelocity(middleMotor,−70);
160   dxl.setGoalVelocity(topMotor,90);
161   delay(220);
162   for (int i = 0; i < 12; i++){
163   swim3();
164   }
165   dxl.setGoalVelocity(middleMotor,70);
166   dxl.setGoalVelocity(topMotor,−90);
167   delay(220);
168   dxl.setGoalVelocity(topMotor,0);
169   dxl.setGoalVelocity(middleMotor,0);
170   delay(750);
171 }
```

Bibliografía

[1] Jonathan Ruiz de Garibay Pascual. Robótica: Estado del arte. *Universidad de Deuston. Número. Fecha*, page 54, 2006.

[2] Kyle Gilpin and Daniela Rus. Modular robot systems. *IEEE robotics & automation magazine*, 17(3):38–55, 2010.

[3] Robert Grabowski, Luis E Navarro-Serment, Christiaan JJ Paredis, and Pradeep K Khosla. Heterogeneous teams of modular robots for mapping and exploration. *Autonomous Robots*, 8(3):293–308, 2000.

[4] Alexandre Campeau-Lecours, Hugo Lamontagne, Simon Latour, Philippe Fauteux, Véronique Maheu, François Boucher, Charles Deguire, and Louis-Joseph Caron L'Ecuyer. Kinova modular robot arms for service robotics applications. In *Rapid Automation: Concepts, Methodologies, Tools, and Applications*, pages 693–719. IGI global, 2019.

[5] Jordi Pages, Luca Marchionni, and Francesco Ferro. Tiago: the modular robot that adapts to different research needs. In *International workshop on robot modularity, IROS*, 2016.

[6] Antonio Barrientos, Luis Felipe Peñin, Carlos Balaguer, and Rafael Aracil. *Fundamentos de robótica*, volume 2. McGraw-Hill Madrid, 2007.

[7] María del Rosario Neira Piñeiro, María Esther del Moral Pérez, and Inés Fombella Coto. Aprendizaje inmersivo y desarrollo de las inteligencias múltiples en educación infantil a partir de un entorno interactivo con realidad aumentada. *Magister: Revista miscelánea de investigación*, 31(2):1–8, 2019.

[8] Francesco Mondada, Giovanni C Pettinaro, Andre Guignard, Ivo W Kwee, Dario Floreano, Jean-Louis Deneubourg, Stefano Nolfi, Luca Maria Gambardella, and Marco Dorigo. Swarm-bot: A new distributed robotic concept. *Autonomous robots*, 17(2):193–221, 2004.

[9] Runxiao Ding, Paul Eastwood, Francesco Mondada, and Roderich Groß. A stochastic self-reconfigurable modular robot with mobility control. In *Conference Towards Autonomous Robotic Systems*, pages 416–417. Springer, 2012.

[10] Lei Zhang, Zhenhua Li, Hao Zhang, and Huaming Zhong. A simulation study of modular robot self-replication. In *International Symposium on Intelligence Computation and Applications*, pages 479–489. Springer, 2012.

[11] Mark Yim, Paul White, Michael Park, and Jimmy Sastra. *Modular Self-Reconfigurable Robots*, pages 5618–5631. Springer New York, New York, NY, 2009. ISBN 978-0-387-30440-3. doi: 10.1007/978-0-387-30440-3_334. URL https://doi.org/10.1007/978-0-387-30440-3_334.

[12] Marco Aurelio Troncos Riofrío. Diseño y ensamble de un brazo robot como módulo de laboratorio para el escaneo de curvas en 3d. 2017.

[13] José Baca, Manuel Ferre, and Rafael Aracil. A heterogeneous modular robotic design for fast response to a diversity of tasks. *Robotics and Autonomous Systems*, 60(4): 522–531, 2012. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2011.11.013. URL https://www.sciencedirect.com/science/article/pii/S0921889011002168.

[14] Kyle Gilpin, Keith Kotay, Daniela Rus, and Iuliu Vasilescu. Miche: Modular shape formation by self-disassembly. *The International Journal of Robotics Research*, 27(3-4): 345–372, 2008.

[15] Juan Gonzalez. DiseÑo de robots Ápodos, Dec 2003. URL http://www.iearobotics.com/personal/juan/doctorado/tea/html/node41.html.

[16] Deisy Yisneth Forero Quintero, Marco Andres Meza Calderon, et al. Diseño y construcción de un robot acuático. B.S. thesis, Universidad Piloto de Colombia, 2015.

[17] Héctor A Moreno, Roque Saltarén, Lisandro Puglisi, Isela Carrera, Pedro Cárdenas, and César Álvarez. Robótica submarina: Conceptos, elementos, modelado y control. *Revista Iberoamericana de Automática e Informática industrial*, 11(1):3–19, 2014.

[18] Auke J Ijspeert, Jonas Buchli, Alessandro Crespi, Ludovic Righetti, and Yvan Bourquin. Institute presentation: Biologically inspired robotics group at epfl. *International Journal of Advanced Robotics Systems*, 2(ARTICLE):175–199, 2005.

[19] Alessandro Crespi, Konstantinos Karakasiliotis, Andre Guignard, and Auke Jan Ijspeert. Salamandra robotica ii: an amphibious robot to study salamander-like swimming and walking gaits. *IEEE Transactions on Robotics*, 29(2):308–320, 2013.

[20] AgnathaX BioRob - EPFL. URL https://www.epfl.ch/labs/biorob/research/amphibious/agnathax/.

[21] Martín Madueño Ortega et al. *Control Teleoperado del robot RV-M1 mediante dispositivo móvil y Realidad Aumentada*. PhD thesis, Universitat Politècnica de Catalunya. Escola Politècnica Superior d …, 2013.

[22] Iván Mauricio Melo Bohórquez. Realidad aumentada y aplicaciones. *Tecnología Investigación y Academia*, 6(1):28–35, 2018.

[23] Oskari Sihvonen. Prosessinohjausratkaisun tuottaminen vuzix m4000-älylaseille. 2021.

[24] Arturo Merino. Realidad mixta.

[25] Jon Peddie. *Augmented reality: Where we will all live*. Springer, 2017.

[26] Jennifer Langston. To the moon and beyond: How hololens 2 is helping build nasa's orion spacecraft, Sep 2020. URL `https://news.microsoft.com/innovation-stories/hololens-2-nasa-orion-artemis/`.

[27] Bill Briggs. Vroom with a view: Hololens 2 powers faster fixes for mercedes-benz usa, Sep 2020. URL `https://news.microsoft.com/transform/vroom-with-a-view-hololens-2-powers-faster-fixes-mercedes-benz-usa/`.

[28] Xavier Basogain, Miguel Olabe, Koldobika Espinosa, C Rouèche, and JC Olabe. Realidad aumentada en la educación: una tecnología emergente. *Escuela Superior de Ingeniería de Bilbao, EHU. Recuperado de http://bit. ly/2hpZokY*, 2007.

[29] Mark Garcia. Astronaut shane kimbrough wears an augmented reality headset, Sep 2021. URL `https://www.nasa.gov/image-feature/astronaut-shane-kimbrough-wears-an-augmented-reality-headset`.

[30] Veronica McGregor Guy Webster and Dwayne Brown. Nasa, microsoft collaboration will allow scientists to 'work on mars' – nasa's mars exploration program, Jan 2015. URL `https://mars.nasa.gov/news/1773/nasa-microsoft-collaboration-will-allow-scientists-to-work-on-mars/`.

[31] Tony Greicius. 'mixed reality' technology brings mars to earth, Mar 2016. URL `https://www.nasa.gov/feature/jpl/mixed-reality-technology-brings-mars-to-earth`.

[32] Allan Brito. *Blender 3D*. Novatec, 2007.

[33] José Baca, S.G.M. Hossain, Prithviraj Dasgupta, Carl A. Nelson, and Ayan Dutta. Modred: Hardware design and reconfiguration planning for a high dexterity modular self-reconfigurable robot for extra-terrestrial exploration. *Robotics and Autonomous Systems*, 62(7):1002–1015, 2014. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2013.08.008. URL `https://www.sciencedirect.com/science/article/pii/S0921889013001516`. Reconfigurable Modular Robotics.

[34] José Baca, Bradley Woosley, Prithviraj Dasgupta, and Carl A. Nelson. Configuration discovery of modular self-reconfigurable robots: Real-time, distributed, ir+xbee communication method. *Robotics and Autonomous Systems*, 91:284–298, 2017. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2017.01.012. URL `https://www.sciencedirect.com/science/article/pii/S092188901630029X`.

[35] Xiang Zhou, Liyu Tang, Ding Lin, and Wei Han. Virtual augmented reality for biological microscope in experiment education. *Virtual Reality Intelligent Hardware*, 2(4):316–329, 2020. ISSN 2096-5796. doi: https://doi.org/10.1016/j.vrih.2020.07.004. URL `https://www.sciencedirect.com/science/article/pii/S2096579620300577`.

[36] Orlando Sabogal Rojas et al. Holomuseo: aplicación de realidad mixta con contenido multimedia desacoplado. B.S. thesis, Uniandes, 2019.

[37] J.M. Azorín J.M. Sabater. R. Saltarén, M. Almonacid. *Prácticas de Robótica utilizando Matlab*. Escuela Politécnica Superior de Elche, 2000.

[38] César Peña, Cristhian Riaño, and Gonzalo Moreno. Robotgreen. a teleoperated agricultural robot for structured environments. *Journal of Engineering Science & Technology Review*, 11(6), 2018.

[39] Francisco A Candelas-Herías and Jorge Pomares. Práctica 3. protocolos de transporte tcp y udp. *Redes*, 2009.

[40] Tcp/ip interface. URL `https://it.mathworks.com/help/instrument/tcp-ip-interface.html`.

[41] Robotics Toolbox - Peter Corke. URL `https://petercorke.com/toolboxes/robotics-toolbox/`.

Acrónimos

API	Interfaz de Programación de Aplicaciones *(Application Programming Interfaces)*.
CAD	Diseño Asistido por Computadora *(Computer-Aided Design)*.
EPFL	Escuela Politécnica Federal de Lausana *(École Polytechnique Fédérale de Lausanne)*.
ESA	Agencia Espacial Europea *(European Space Agency)*.
EVA	Actividad Extravehicular *(Extravehicular activity)*.
GDL	Grados De Libertad *(Degrees Of Freedom)*.
GUI	Interfaz Gráfica de Usuario *(Graphical User Interface)*.
JDK	Kit de desarrollo de Java *(Java Development Kit)*.
JPL	Laboratorio de Propulsión a Reacción *(Jet Propulsion Laboratory)*.
MIT	Instituto Tecnológico de Massachusetts *(Massachusetts Institute of Technology)*.
MRS	Sistemas Robóticos Modulares *(Modular Robotic Systems)*.
MTH	Matrices de Transformación Homogénea *(Homogeneous Transformation Matrix)*.
NASA	Administración Nacional de Aeronáutica y el Espacio *(National Aeronautics and Space Administration)*.
NDK	Kit de desarrollo nativo *(Native Development Kit)*.
PARC	Centro de Investigación de Palo Alto *(Palo Alto Research Center)*.
QR	Respuesta Rápida *(Quick Response)*.
SDK	Kit de Desarrollo de Software *(Software Development Kit)*.
STL	Estereolitografía *(Stereolithography)*.
TCP	Protocolo de control de transmisión *(Transmission Control Protocol)*.
UDP	Protocolo de datagramas de usuario *(User Datagram Protocol)*.