INSTITUTO TECNOLÓGICO DE CHIHUAHUA
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

"DESARROLLO DE UNA INTERFACE PARA UN BRAZO ROBÓTICO ARTICULADO."

TESIS

QUE PARA OBTENER EL GRADO DE

MAESTRO EN INGENIERÍA MECATRÓNICA

PRESENTA:

JESÚS HUMBERTO ÁVILA MARTÍNEZ

DIRECTOR(A) DE LA TESIS:
cDR. ROGELIO BARAY ARANA

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO®

CHIHUAHUA, CHIH., JUNIO 2019

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Chihuahua

"2019, Año del Caudillo del Sur, Emiliano Zapata"

Chihuahua, Chih., 13 de junio de 2019

M.F. LUIS CARDONA CHACÓN
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN
PRESENTE

Por medio de la presente notificamos a usted que en cumplimiento de los requerimientos para la obtención de grado de Maestro, el documento de tesis del **C. JESÚS HUMBERTO ÁVILA MARTÍNEZ,** ha sido aprobado y aceptado para su impresión. El título de la tesis es:

"DESARROLLO DE UNA INTERFACE PARA UN BRAZO ROBÓTICO ARTICULADO"

Por lo que proponemos, le sea concedida la autorización de impresión correspondiente.

Agradeciendo la atención a la presente, quedamos de usted:

ATENTAMENTE
Excelencia en Educación Tecnológica®

M.C. Rogelio Enrique Baray Arana
MIEMBRO DEL JURADO DE EXAMEN

Dra. Carmen Leticia García Mata.
MIEMBRO DEL JURADO DE EXAMEN

Dr. José Eduardo Acosta Cano de los Ríos
MIEMBRO DEL JURADO DE EXAMEN

M.C. Pedro Rafael Márquez Gutiérrez
MIEMBRO DEL JURADO DE EXAMEN

LCC/reba.*

SECRETARÍA DE EDUCACIÓN PÚBLICA
INSTITUTO TECNOLÓGICO DE CHIHUAHUA

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Chihuahua

"2019, Año del Caudillo del Sur, Emiliano Zapata"

Chihuahua, Chih., 13 de junio de 2019

C. JESÚS HUMBERTO ÁVILA MARTÍNEZ
PRESENTE

Por este conducto le comunico que, a propuesta del Jurado de Examen, la División de Estudios de Posgrado e Investigación le ha concedido la autorización para la impresión de tesis para obtener el grado de Maestro, cuyo título es:

"DESARROLLO DE UNA INTERFACE PARA UN BRAZO ROBÓTICO ARTICULADO"

Con el siguiente contenido de capítulos:

I. Conceptos generales.
II. Marco Teórico.
III. Modelado 3D.
IV. Desarrollo.
V. Sistemas de control.
VI. Conclusiones.

ATENTAMENTE
Excelencia en Educación Tecnológica.

M.F. LUIS CARDONA CHACÓN
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

LCC/reba*

SECRETARÍA DE EDUCACIÓN PÚBLICA
INSTITUTO TECNOLÓGICO DE CHIHUAHUA

Chihuahua, Chih., a 17 de junio de 2019.

Dra. María Elena Álvarez Buylla
Directora de CONACYT.

P r e s e n t e.

Por este conducto aprovecho la ocasión para saludarlo e informarle que a la fecha he obtenido el Grado de Maestría en Ingeniería Mecatrónica en la División de Estudios de Posgrado e Investigación del Instituto Tecnológico de Chihuahua. Motivo por el cual agradezco todo el apoyo brindado por esta Institución que Usted representa, el otorgamiento de esta beca-credito permitió dedicarme de tiempo completo a la realización de mis estudios de Posgrado y de esta manera lograr el cumplimiento del objetivo principal del convenio establecido.

Sin otro particular por el momento, me es grato quedar de Usted como su seguro servidor, no sin antes reiterar mi agradecimiento. Muchas Gracias!

A t e n t a m e n t e

Jesús Humberto Ávila
Ex becario CONACYT

Carta de Agradecimientos

Primeramente quiero agradecerle a Dios por darme el don de vida, por permitirme haber terminado mis estudios profesionales, darme salud y sobre todo por haberme brindado una gran familia que me ha apoyado incondicionalmente.

A mis padres Humberto avila y a Constantina Martinez quisiera agradecerles por las grandes oportunidades que me han ofrecido a lo largo de mi vida y a su vez por guiarme durante toda este trayecto, creyendo en mí y ahora por acompañarme en este gran paso y nueva etapa de mi vida en la que me integro a la sociedad como una persona con un título profesional nivel maestría.

A mí querida pareja Mabel Camuñez que desde un principio me apoyo y motivo durante toda esta trayectoria brindándome todo su amor y confianza.

Al Instituto tecnológico de Chihuahua por aceparme e inculcarme valores como la lealtad, responsabilidad y honestidad. Por haberme equipado con las herramientas necesarias para sobresalir como profesionista.

Le agradezco a todos mis profesores a lo largo de mi travesía como estudiante, les doy las gracias por compartir sus conocimientos y sus experiencias, por preocuparse por mí y mi desempeño académico y algunos por convertirse en amigos en los cuales siempre podre confiar y contar con ellos.

RESUMEN
DE TESIS

DESARROLLO DE UNA INTERFACE PARA UN BRAZO ROBÓTICO ARTICULADO

Ing. Jesús Humberto Ávila Martínez
Candidato a Maestro en Ingeniería Mecatrónica
División de Estudios de Posgrado e Investigación del
Instituto Tecnológico de Chihuahua
Chihuahua, Chih. 2017
Asesor: cDr. ROGELIO BARAY ARANA

En el presente documento se presenta la investigación y desarrollo de una interface de usuario desarrollada en *Unity* para un brazo articulado PUMA TQ2000 de 6 grados de libertar al igual del diseño del control PID cada una de las articulaciones del brazo robótico utilizando el método basado en la curva de reacción de Ziegle y Nichols para estimar el modelo matemático del brazo robótico.

Dentro de la interface de usuario existen varios elementos de los cuales son el módulo de simulación de movimientos, caculos de la cinemática directa y configuración del puerto de comunicación además de tener una simulación de movimientos de los elementos del brazo robótico en 3D todo esto gracias al ambiente de desarrollo proporcionado por *Unity* dándole al usuario la posibilidad de realizar simulación y emulación del control de movimientos del dispositivo en tiempo real.

LISTA DE FIGURAS
iii

LISTA DE TABLAS v

CAPÍTULO

1. CAPÍTULO 1: Conceptos generales. 1
1.1 Introducción. 1
1.2 Descripción del tema de tesis. 1
1.3 Justificación y viabilidad (Infraestructura). 2
1.4 Antecedentes. 3

2. CAPÍTULO 2: Estado del arte
2.1 Herramientas de visualización. 18
2.1.1 Visualización. 18
2.1.2 Técnicas de visualización. 19
2.1.3 Animación. 19
2.1.4 Realidad Virtual. 19
2.1.5 Gráficos dinámicos. 20
2.1.6 Interfaz de usuario (*User interface*). 20
2.1.7 Motor de física. 21
2.1.8 Tecnología PhysX. 21
2.2 Robótica. 21
2.3 Robot. 21

3. CAPÍTULO 3: Modelado 3D. 22
3.1 *GameObjects u object.* 28
3.2 *Sprites*. 28
3.3 Escenas. 30
3.4 Ventana de Jerarquía (Hierarchy). 30
3.5 Parentesco (*Patenting*). 30
3.6 Transform. 31

4. CAPÍTULO 4: Desarrollo. 33
4.1 Interfaz de usuario. 34
4.1 *Canvas*. 34
4.2 Componentes de Interacción. 34
4.2.1. *Button* (Botón). 34
4.2.2 *Slider* (Deslizador). 35
4.2.3 Input Field (Campo de Input) 35

4.3 Evolución de la interface. 36
4.4. Estudio Cinemático. 42
4.4.1 Estudio Cinemático directo. 43
4.4.2 Estudio Cinemático Inverso. 43
4.5 Método de análisis de la cinemática directa. 43

5. CAPÍTULO 5: Sistemas de control 54
5.1 Elementos del sistema de control. 55
5.1.1 Motor de corriente directa Maxon. 55
5.1.2 Driver del motor L298N. 56
5.1.3 Unidad de control del motor. 56
5.1.4 Sensor de posición resistivo. 57
5.1.5 Trasmisión del Motor. 58
5.2 Control PID. 60
5.3 Control de posición. 63
5.4 Comunicación. 67

6. CAPÍTULO 6: Conclusiones 76
6.1 Trabajos Futuros. 78

BIBLIOGRAFIA 80

LISTA DE FIGURAS

3. CAPÍTULO 3: Modelado 3D.

Figura 3.1 Brazo Robótico puma 2000 (Solidworks) 23
Figura 3.2 Brazo Robótico puma 2000 (Solidworks) 24
Figura 3.3 Brazo Robótico puma 2000 (3DS Max) 25
Figura 3.4 Resultados del escaneo utilizando el ZenFone AR en 3DS MAX 26
Figura 3.5 Virtualización del laboratorio LabCEM (3DS MAX) 26
Figura 3.6 Resultados de la exportación de los modelos de 3DS Max a Unity (a). 27
Figura 3.7 Resultados de la exportación de los modelos de 3DS Max a Unity (b). 28
Figura 3.8 Diferentes tipos de GameObjects en Unity. 29
Figura 3.9 Ventana de selección de sprites. 30
Figura 3.10 Ventana de Jerarquía y ventana del Transform. 31
Figura 3.11 Ventana de Jerarquía y ventana del Transform. 32

4. CAPÍTULO 4: Desarrollo.

Figura 4.1 Canvas y aplicación de este. 34
Figura 4.2 Botón (*Button*). 35
Figura 4.3 *Slider* (Deslizador). 35
Figura 4.4 *Botón Input Field* (Campo de *Input*). 36
Figura 4.5 Prototipo de interfaz Versión 1.0. 36
Figura 4.6 Prototipo de interfaz Versión 1.0. 37
Figura 4.7 Ventana de configuración de eventos *On Click*. 37
Figura 4.8 Ventana del menú principal. 38
Figura 4.9 Módulo de configuración de puerto de comunicación. 38
Figura 4.10 Módulo de simulación de cinemática directa. 39
Figura 4.11 Configuración del evento *On Value Changed*. 40
Figura 4.12 Implementación de rotaciones utilizando la función "Quaternion.Euler". 42
Figura 4.13 Relación entre cinemática directa e Inversa. 44
Figura 4.14 Representación de los parámetros necesarios para aplicar el método de D-H en los últimos 3GDL del efector final. 45
Figura 4.15 Robot Puma TQMA 2000 vista lateral. 47
Figura 4.16 Robot Puma TQMA 2000 vista Frontal. 47
Figura 4.17 Robot Puma TQMA 2000 vista isométrica. 48
Figura 4.18 Representación de ejes coordenados del Robot Puma TQMA 2000. 48
Figura 4.19 Simulación y comparación del robot Brazo robótico posición home. 51

Figura 4.20 Simulación y comparación del Brazo robótico (Rotación en: q2 = 90°). 52
Figura 4.21 Simulación y comparación del Brazo robótico (Rotación en: q1 = -90 °, q3 = 45 ° y q4 = 45). 53

5. CAPÍTULO 5: Sistemas de control

Figura 5.1 Diagrama a bloques del funcionamiento de la U.C. del brazo robótico. 54
Figura 5.2 Diagrama de control por articulación del brazo robótico. 55
Figura 5.3 Parámetros del motor Maxon según especificaciones de fabricante. 55
Figura 5.4 Tablilla armada con chip L298N y su diagrama interno. 56
Figura 5.5 Sensor de posición resistivo, su diagrama de conexión y su relación voltaje/posición. 57
Figura 5.6 Motor de corriente directa Maxon con reductor, B) Reductor C) Trasmisión de banda dentada de la segunda articulación del brazo robótico 58
Figura 5.7 Modelo eléctrico del motor MAXON y modelo matemático. 59
Figura 5.8 Diagrama de bloques de un sistema PID. 61
Figura 5.9 Respuesta escalón a la planta y aplicación del método Z-N basado en la curva de reacción 62
Figura 5.10 Código de la función dirección para determinar el giro del motor. 64
Figura 5.11 Función de cálculos del PID para cada articulación de brazo robótico. 65
Figura 5.12 Función PIDLoop necesaria para el correcto funcionamiento de la función PID. 65
Figura 5.13 Función Bucle encargada donde se estable los nuevos parámetros de la señal de referencia y se ejecutan las funciones PIDLoop y Dirección. 66
Figura 5.14 Diagrama de conexiones de la unidad de control maestro, Uc esclavo, Driver puerto H y Articulación 1. 66
Figura 5.15 Funciones encargadas de la comunicación serial entre el HUB y la HMI. 68
Figura 5.16 Funciones encargadas de la comunicación serial entre la HMI y el HUB. 69
Figura 5.17 Conexión y relación de los elementos de control de maestro/esclavo 70
Figura 5.18 Sistema de control Maestro/Esclavo del brazo robótico. 71
Figura 5.19 Resultados de la interconexión de la HMI, sistema de control y del brazo robótico 72
Figura 5.20 Prueba de monitoreo 1 del brazo robótico (Rotación en: q2 = 90°). 73

Figura 5.21 Prueba de monitoreo 2 del brazo robótico (Rotación en: q2 = 90°, q4 = 90°). 74
Figura 5.22 Prueba de monitoreo 3 del brazo robótico (Rotación en: q1 = 45°, q2 = 135°, q3 = -135°). 75

LISTA DE TABLAS

1. CAPÍTULO 1: Conceptos generales.

Tabla 1.1 Uso y conocimiento de los simuladores en el presente o pasado. 5
Tabla 1.2 Herramientas difusas para una selección de áreas de investigación. 6

2. CAPÍTULO 2: Estado del arte

Tabla 2.1 Evaluación de software etapa 1 14
Tabla 2.2 Evaluación de software etapa 2(a). 16
Tabla 2.3 Tabla 2.3 Evaluación de software etapa 2(b). 17

5. CAPÍTULO 5: Sistemas de control

Tabla 5.1 Características de los circuitos integrados ATmega2560 y ATmega328. 56
Tabla 5.2 Parámetros de ajuste para el método de curva de reacción. 63
Tabla 5.3 Resultados de los parámetros de ajuste. 63

CAPÍTULO 1.

Conceptos generales.

1.1 Introducción

El presente trabajo de investigación y desarrollo es parte de un proyecto integrador que se desarrolla en el programa de la Maestría en Ingeniería Mecatrónica de la División de Estudios de posgrado e Investigación del Instituto Tecnológico de Chihuahua, el cual, está orientado al desarrollo de tecnologías que sean pertinentes en el entorno productivo e industrial en el estado de Chihuahua, específicamente en aplicaciones de robots articulados.

1.2 Descripción del tema de tesis

Las interfaces de usuario son los medios por el cual los usuarios pueden comunicarse con una maquina o equipo y tienen como finalidad permitir al usuario interactuar y controlar de forma más eficiente las máquinas.

Este tipo de interacción que se obtiene a través de las interfaces de usuario, ha llevado a los investigadores y desarrolladores a buscar interfaces más intuitivas y poderosas con el objetivo de que estas sean más fáciles de entender y sean más amigables al momento de usarlas.

El objetivo de esta investigación y desarrollo se centra en los problemas del control de movimiento, el posicionamiento y la comunicación inalámbrica de un brazo robótico que se encuentra montado a una plataforma móvil y la interface de usuario que debe de realizar la simulación y emulación de dicho brazo articulado, por lo tanto, se deben de resolver los siguientes aspectos involucrados: es necesario implementar una sistema de control de movimiento basado en la dinámica del robot para cada una de las articulaciones, un control de posicionamiento del efector final basado en la cinemática del brazo articulado, además es necesario desarrollar una interface de entorno grafico que permita la visualización y

emulación del robot y que muestre la dinámica y la cinemática de cada uno de los eslabones y articulaciones del brazo. Finalmente implementar protocolos de comunicación inalámbrica para reducir el cableado del robot hacia la interfase de usurario.

Actualmente existen varias herramientas que nos permiten el diseño y desarrollo de diferentes tipos de usuario compatibles con diferentes protocolos de comunicación facilitando el control de diferentes tipos de dispositivos. Pero se debe de encontrar la que más se adapte y satisfaga las necesidades del proyecto a desarrollar.

1.3 Justificación y viabilidad (Infraestructura)

La presente investigación tiene como propósito principal el desarrollo de una Interface Humano-Maquina (*Human-machine interface*), como una continuación a proyectos anteriores sobre el control de movimiento en plataformas móviles y como a su vez funciona como complemento de uno de los proyectos actuales que se están realizado sobre el diseño de un brazo robótico, los cuales se han desarrollado en el laboratorio de control de Electro mecanismos (LabCEM), perteneciente al programa de la Maestría en Ingeniería Mecatrónica en el Instituto Tecnológico de Chihuahua

En esta investigación se pretende desarrollar una interface de usuario por medio del estudio y análisis de algunas de las múltiples plataformas de desarrollo de entornos visuales y así seleccionar la que reúna las características que satisfagan las necesidades del proyecto. La finalidad de esta interface de usuario es tener un entorno de programación amigable que permita al usuario evaluar el desempeño cinemático y dinámico del brazo articulado por medio de la visualización del brazo en forma emulada tanto en forma asíncrona como en forma síncrona con el brazo articulado, es decir, el usuario puede primero probar sus algoritmos de control con o sin conexión con el brazo real. En caso de ser asíncrono, la interface no se comunica con el brazo y en la pantalla se visualiza por medio de la emulación el comportamiento que tendrá el modelo del brazo articulado. En el modo síncrono

seleccionado el usuario emula el brazo y a su vez se comunica con el brazo real y observa el desempeño del mismo, en tiempo real. Se establece como primicia que la comunicación entre el sistema donde se encuentra el usuario y el brazo articulado montado en la plataforma móvil deberá ser inalámbrica.

1.4 Antecedentes

La simulación dinámica es la estratégica que hoy en día se utiliza para todo tipo de robots, especialmente para el análisis y optimización del movimiento, lo cual permite realizar el diseño de prototipos y controladores de forma más rápida , así como su validación en entornos simulados, antes de ser ejecutados en forma real.

Hay que considerar que los simuladores han sido desarrollados para resolver un problema en particular, por lo tanto, estos tienen características diferentes y requerimientos diferentes y su desempeño no siempre puede ser evaluado en la misma máquina. Son además de código y algoritmos difíciles de acceder, por lo que muchos de ellos no son de propósito general. Por otro lado es posible encontrar algunos simuladores que son de software "*open source*" los cuales están bien documentados y disponibles al público.

Teniendo en cuenta que en la actualidad el desarrollo de *software* de simulación de robots ha aumentado en comparación con los últimos años, se pueden encontrar una gran variedad de estos para la simulación orientada a la robótica más extensa. De estos algunos tienen compatibilidad con "*middleware*" para robots, por ejemplo *ROS, YARP OROCOS, MIRO, Urbi, Orca, OpenRDK*, etc., dónde su función es ocultar la complejidad de bajo nivel para mejorar el software y reutilizar la infraestructura del robot para poder resolver problemas más específicos.

En cuanto a *software* de simulación, algunos de ellos se sabe que descienden de la comunidad de gráficos por computadora; por ejemplo OPD [1] y BULLET [2] los cuales fueron

inicialmente desarrollados en "*Underlying Physics Engines*" y son usados para la creación videojuegos. Esto resulta contraproducente debido a que no satisfacen las necesidades en el campo de la robótica, las cuales son más exigentes debido a la precisión computacional, precisión física y en tiempo.

En la encuesta "*Tools for simulating humanoid robot dynamics: a survey based on user feedback"* realizada por Serena Ivaldi y otros [3] encuestaron a 119 personas para verificar qué software es el que tiene un mayor desarrollo.

En dicha encuesta, el 92% fueron hombres y 8% mujeres de 32 a 57 años de edad, de los cuales el 62% posee doctorado y el 35% posee maestría o licenciatura. Sus lugares de residencia son Estados Unidos 19%, Francia 17%, China 2%, Colombia 2%, Corea 3%, Holanda3%, Escocia 3%, España 4%. UK4% Bélgica 7%, Italia 10% y Otros 15%. Las principales áreas de aplicación fueron 26% en Robots Humanoides, 20% en Robot Móviles, 11% en Robots con extremidades múltiples, 8% en Robots de Servicio, 7% en robots Industriales, 7% en Simulaciones Numéricas de Sistemas Físicos, 5% en Robot de voladores y el otro 16% la utilizan en la competencia *"The Darpa Robotics Challenge" (DRC*). En la Tabla 1.1 podemos observar un listado de los simuladores principalmente utilizados en la actualidad.

Como se muestra en la Tabla 1.2 existen diversas áreas de esta investigacion que nosotros pretendemos desarrollar dentro de la rama de Robots móviles, asi que a continuacion se muestran las caracteristicas principales de los softwares más comunmente usados.

Tabla 1.1: Uso y conocimiento de los simuladores en el presente o pasado.
(Knowledge and past/present use of simulators, Serena Ivaldi y otros, 2014)

Software	Utilizado como Herramienta principal	Utilizado pero no como Herramienta principal	Uso frecuente solo para pruebas	Utilizado una vez para probarlo	Usado pero abandonado	Conocido pero nunca usado	Sin conocimiento de su existencia
Gazebo	13%	7%	3%	18%	10%	34%	15%
ODE	11%	12%	5%	18%	22%	22%	10%
Bullet	5%	13%	7%	12%	10%	29%	24%
V-Rep	5%	3%	3%	18%	3%	29%	39%
Webots	4%	7%	1%	16%	13%	32%	27%
OpenRave	5%	3%	2%	7%	5%	29%	49%
Robotran	4%	0%	1%	4%	2%	29%	76%
XDE	5%	3%	0%	3%	1%	14%	74%
Blender	5%	17%	7%	22%	6%	28%	15%
MujoCo	2%	0%	0%	4%	2%	21%	71%
Icub_Sim	4%	4%	2%	3%	3%	29%	55%
Nvidia PhysX	1%	1%	4%	12%	7%	43%	32%
OpenSIM	3%	4%	3%	8%	1%	41%	40%
HumanS	0%	0%	0%	1%	1%	10%	88%
Moby	2%	1%	0%	0%	2%	14%	81%
Vortex	3%	2%	0%	5%	5%	17%	68%
RoboRobo	3%	1%	0%	0%	1%	4%	91%

Gazebo [4] Es un simulador de robots múltiples para entornos al aire libre, Desarrollado por "*Open-Source Robotics Foundation*". Es el software oficial de la Competencia *"Darpa Robotics Challenge"*. Es compatible con múltiples motores de física (ODE, Bullet, Simbody, Dart).

- SO compartido: GNU / Linux
- API principal: C ++
- Motivo principal de la adopción:

  Middleware principal utilizado con: ROS

Tabla 1.2: Herramientas difusas para una selección de áreas de investigación.
(Most diffused tools for a selection of the research areas, Serena Ivaldi y otros, 2014)

Campo de investigación	usuarios	Software comúnmente usados	Otros Software Utilizados
Robots humanoides	32	ODE, Gazebo, Robotran, OpenRave, Arboris-Python, XDE y Icub_SIM	Drake, MapleSim, MujoCo, OpenSIM, Robotic-sLab, SL, Vortex, V-Rep, Webots, codigo propio
Robots móviles	25	Gazebo, ARGos, Webot, V-Rep y Vortex	ADAMS, Autodesk Inventor, Bullet, ODE, Morse, roborobo, Sim, Codigo propio
Robots con extremidades múltiples	12	Webot, ODE	Gazebo, ADAMS, Autolev, Bullet, Moby, RoboticLab, SIMPACK, VoxCad
Robots de servicios	12	Gazebo, OpenRave	OpenSIM, V-Rep, Morse, RCIS, SL
Simulaciones Numéricas de Sistemas Físicos	8	Bullet	MujoCO, ODE, oPENsim, Simulink, trep, XDE
Robot de Voladores	6	ARGos	Robotran,crrcsim, Gazebo, Simulink/Matlab
Robots Nadadores	5	ARGos	roborobo
Manipuladores Industriales	5		Bullet, Dymola, Matlab, V-Rep, XDE
diseño mecánico	4		Moby,MojoCO, V-Rep, Codigo propio
Análisis del movimiento humano	3		Robotran,Bullet, XDE
Snake Robots	3	ODE	Matlab

ARGoS [5] es un simulador multímotor para robótica de enjambres, inicialmente desarrollado dentro del "Swarmanoid project4"

- SO compatible : GNU / Linux, MAC OSX
- API principal: C ++

Webots [6] es un entorno de desarrollo utilizado para modelar, programar y simular robots móviles desarrollados por Cyberbotics Ltd.

- SO: GNU / Linux, Windows, MAC OSX
- API principal: C ++

V-Rep [7] es un software de simulador de robot con un entorno de desarrollo integrado, producido por Coppelia Robotics. Es compatible con múltiples motores de física (ODE, Bullet, Vortex).

- SO: GNU / Linux, Windows, Mac OS
- API principal: C ++, LUA
- Middleware principal utilizado con: ROS

Vortex Dynamics [8] es un software desarrollado por CM Labs, especializado en la simulación de dinámica de contactos en diferentes entornos operativos (por ejemplo, terreno, agua). Le permite transformar modelos CAD estáticos en mecanismos virtuales interactivos, lo que le permite probar sistemas inteligentes y diseño de equipos robóticos en entornos 3D envolventes.

- SO: GNU / Linux, Windows
- Licencia comercial
- API principal: C ++
- Middleware principal utilizado con: ROS

Con esta investigación se pretende obtener resultados que contribuyan al desarrollo de una interfaz gráfica de usuario o GUI (*Graphical User Interface*) que permita el control y monitoreo de un brazo robótico, usando como punto de partida los avances tecnológicos que existen en la actualidad.

Este estudio inicia en la investigación que ChuXin Chen, Mohan M. Trivedi y Clint R. Bidlack realizaron en 1992 [9], donde ellos presentan grandes aportaciones a su época al

desarrollar una GUI que permite al usuario controlar un brazo robótico, así como simular objetos alrededor de su entorno por medio de diferentes tipos de sensores que detectan las siguientes variables:

- Proximidad (*Proximity*).
- Rango De Punto Laser (*Point Laser Range*).
- Rango Ultrasónico (*Ultrasonic Range*).
- Imágenes De La Profundidad De Alcance Láser (*Laser Range Depth Imagery*).
- Imágenes Basadas En La Intensidad De La Intensidad (*Edge Based Intensity Imagery*).

Chen explica claramente que, "*Este trabajo presenta un nuevo diseño de un entorno de visualización para simulación y animación de robots con sensores. El simulador está integrado en un sistema robótico real, y se utiliza para la programación automática de robots, incluyendo la capacidad de programación fuera de línea de los robots* [9]. El software cumple con las necesidades de tener un entorno de simulación que hace más eficiente y confiable el desarrollo de programas de control para el robot, gracias a la posibilidad de simular objetos a su alrededor. Se usarán los resultados presentados en este artículo para visualizar el entorno del robot con el fin de adaptar la interfaz de usuario y que esta pueda simular los movimientos del dispositivo.

En otra publicación del 2005 se presentan diferentes tipos de Interfaces de usuario en desarrollo. Raúl Marín y Otros[10], por ejemplo, presentan una interfaz de usuario que, en conjunto a una tele laboratorio, se conecta a través de Internet y permite a los usuarios controlar y programar a distancia dos robots educativos en línea.

"Dicha interfaz de usuario fue diseñada utilizando técnicas basadas en la realidad aumentada y la realidad virtual no inversiva, que mejoran la forma en que los operadores obtienen / ponen información desde / hacia el escenario robótico. Esta destaca debido a 3 características básicas que esta ópera que son el control del brazo robótico, comunicación

vía internet y una interface que permite la operación de múltiples usuarios. Otros de los aspectos más destacables de esta es que cuenta con múltiples modalidades de control y programación. Entre ellas son la programación de movimiento directo, programación por lenguaje de alto nivel, programación vía remota y programación mediante comandos de voz. Para que todo esto fuera posible se implementaron a la estructura del brazo robótico diversos tipos de sensores tales son como sensores de presencia y sensores ultra sónicos para medir su distancia además de implementación de cámaras para poder visualizar y simular los entornos de este."[10].

Sin embargo los autores expresan que uno de los principales problemas se presentaron al momento de la transferencia de datos fue que las velocidades intermedias de internet dificultan mucho la transferencia de estos debido a que es complicado operar con un ancho de banda constante; esto limita el flujo de información del usuario al servidor y viceversa, y dicha comunicación se debe dar en un periodo de tiempo específico. Este inconveniente se solucionó desarrollando más autonomía por medio del algoritmo de predicción de movimientos, con la finalidad de deducir las trayectorias en caso de una mala conexión a internet. Esta característica vuelve más eficiente el modo *off-line* del elemento controlado y así reduce considerablemente el ancho de banda necesario para su funcionamiento óptimo.

El control y la transferencia de información vía internet es una de las principales funciones con la que la mayoría de dispositivos cuenta. En 2015, Ben Kehoe y Otros [10] desarrollaron interfaces gráficas de usuario con soporte en *"The Cloud"*, una plataforma que facilita el flujo de información entre usuarios de un servidor, y que cuenta con un extenso conjunto de recursos accesibles por Internet, teniendo el potencial de proporcionar importantes beneficios a los robots y sistemas de automatización.

Los autores definen su sistema "*Cloud Robot and Automation"* de la siguiente manera: "Cualquier robot o sistema de automatización que se base en datos o código de una red para

soportar su operación, es decir, donde no todo el sensor, la computación y la memoria está integrado en un solo sistema autónomo"[10].

Todas estas características permiten el desarrollo de interfaces graficas de usuario de manera sofisticada, apoyadas en esta plataforma donde se puede obtener información de distintos tipos de sistemas y/o dispositivos. Estas deben ser capaces de encargarse de diversas actividades, como el monitoreo de distancia recorrida, ubicación y control de un automóvil (como el vehículo autónomo desarrollado por Google), el control de una red de brazos robóticos donde estos pueden acceder a una base de datos para obtener programas previamente elaborados o el monitoreo de líneas de producción o almacenes.

Por otro lado, otras investigaciones se enfocan en el rediseño o adaptación de interfaces graficas de usuario para el control de sistemas ya existentes, con la finalidad de volver más amigables el manejo o control de estos. Un ejemplo de este tipo de avances fue desarrollado por Maj Stenmark y otros[11], el cual va enfocado al desarrollo de una interface que sea capaz de controlar o mejorar las cualidades del robot YuMi de ABB. Este robot fue diseñado con un doble brazo y pinzas flexibles para el ensamble de piezas pequeñas con el fin de facilitar los procesos dentro de la industria. En el desarrollo de la interface de usuario se agregó un panel que permite guardar posiciones con una breve descripción de lo que hace, esto para optimizar la creación de programa volviendo este proceso más rápido, el manejo directo de instrucciones más sencillo, o simplemente para hacer un programa que se usará posteriormente.

Esta interface se sometió a un estudio donde a 24 estudiantes se les dio la tarea de crear un programa sencillo de ensamblaje de unas piezas de *Lego* sin ninguna capacitación de cómo utilizar la interface: *19 de ellos lo lograron con éxito*. Entre otros aspectos destacables de dicho proyecto está la cualidad de crear secuencias de momentos por medio de

comandos de voz, esto compensando la carencia de un generador de texto para poder crear código nativo de este.

Ahora, enfocando en un estudio más recientes donde investigadores como Miguel Campusano y Johan Fabry dan a conocer a través de una revista científica una nueva técnica para facilitar el control y programación de robots. Ellos la exponen como “programación en vivo”. En su estudio del 2015 [12], Campusano hacen una comparación entre sistemas de programación tradicionales y el **LRP** (Live Robot Programming). Nos mencionan que el lenguaje que es típicamente usado en la programación de robots, así como las corridas del código que se genera con este, se ve sujeto a la introducción de su versión de prueba dentro de un simulador para encontrar errores o posibles mejoras antes de obtener una versión final. Los autores dejan claro que dicho proceso de programación y métodos de error/prueba distancian enormemente el código del comportamiento en el que este resulta ya que meten justo en medio el proceso de "rastrear hacia atrás" el error en el código basándose en el resultado, lo cual crea una brecha cognitiva por el simple hecho de tener que "volver sobre sus propios pasos". Este problema se ve resuelto, según los autores, al implementar la Programación En Vivo del Robot (LRP por sus siglas en inglés), la cual le permite al programador realizar el código al mismo tiempo que se va corriendo, proporcionando un mayor control sobre los cambios, los errores y la retroalimentación en tiempo real, sin tener que perder tiempo volviendo a parte por parte del código para corregirlo.

En estudio enfocados al desarrollo de simuladores tenemos el presentado por de Miguel A. y otros [13], donde ellos nos presentan por medio de V-Rep [7] y ROS [14], un control basado en visión para el aterrizaje autónomo de los vehículos aéreos no tripulados. En la investigación realizada por el autor evaluó diferentes tipos de herramientas para encargarse de la simulación de este, donde se consideró la implementación de Webots [6] y Gazebo [4] pero fueron descartados debido a que Webots presentaba problemas porque no había

manera directa de implementar lo simulado al robot real, siendo necesaria la implementación de un "*middleware*" como ROS que se encargue de solucionar el problema. Por otro lado tenemos a Gazebo, cuya versatilidad para realizar simulaciones 3D de mundos virtuales resulta muy útil y al implementarlo en un robot real requiere modificaciones mínimas, además de contar con compatibilidad con ROS pero este presenta la desventaja de requerir un equipo computacional poderoso para llevar acabo cálculos pesados. Para solucionar este problema el autor propone como software alternativo de simulación a V-Rep, donde este lo compara con Gazebo y describen no requiere tantos recursos computacionales, además de generar entornos virtuales este también cuenta con compatibilidad con ROS.

Por otro lado existen investigaciones enfocadas al análisis del movimiento humano. Un ejemplo es el caso que nos expone Minhui Qi y otros [15], donde ellos nos muestran un sistema de que analiza la forma de caminar de las persona censando por medio de acelerómetros y un giroscopios para transferir los datos obtenidos por WI-FI para adquirirlos en tiempo real, donde son procesador y enviados a un "*software"* de simulación para reconstruir la forma de caminar. El software implementado para realizar dicha tarea fue V-Rep. El autor en su investigación reporta resultados favorables e informa que en un futuro se pretende implementar una herramienta de posición peatonal.

En otra investigación con el mismo enfoque realizada por Joao O. Barros y otros [16]pretenden controlar una plataforma humanoide con tele operación bimanual, apoyándose de V-Rep y ROS para realizar las simulaciones.

En el área de cálculos de trayectorias podemos encontrar investigaciones como la que desarrollo Ta-Ming Shih y otros [17] la cual se basa en la cinemática inversa con control difuso para controlar un brazo robótico multi-DOF, generando una trayectoria directa para reducir el tiempo de respuesta del robot. Los resultados fueron validados previamente en un

entorno de modelos 3D, implementado LabView, para así poderlos ejecutar en un brazo robótico real. Algunos de los problemas que menciona el autor es que al tratarse de un sistema de comportamiento no lineal se vuelve complicado implementar una solución matemática. Sin embargo, a su vez reporta resultados positivos de su investigación en los que asegura que el control que desarrollo cuenta con buena estabilidad y que es robusto.

A continuación en el presente documento se presentara se presentara la siguiente arquitectura empezando con el Capítulo 2, abordando el estado del arte donde se incluye las herramientas de visualización para interactuar con los entornos 3D, en el capítulo 3 se presentan el procedimiento a seguir para exportar elementos 3D desde *Solidworks* hasta *Unity* además de las herramientas que ofrece el espacio de trabajo de *Unity* para realizar el diseño y el manejo de estos, en el capítulo 4 se presenta la metodología que se siguió para el desarrollo de la tesis paso a paso así como la documentación de los resultados obtenidos dentro del ambiente de simulación, en el capítulo 5 se presenta el desarrollo del sistema de control al igual que la documentación de los resultados obtenidos después de haber realizado la integración de del este con la HMI mostrando como interactuar entre si de manera correcta y obteniendo los resultados positivos de la investigación, en el capítulo 6 se establecen las conclusiones de la investigación así como la justificación para trabajos futuros señalando áreas de oportunidad.

CAPÍTULO 2.

Estado del arte.

En el presente capítulo se describirá el estado del arte acerca de las herramientas existentes en el desarrollo de simuladores de mundos virtuales e Interfaces de usuarios que pueden van orientadas en a las necesidades del proyecto.

Como se mencionó en el capítulo anterior existen varias herramientas de simulación como lo son Gazebo [4], ARGoS [5], Webots [6], V-Rep [7] y Vortex Dynamics [8] que son comúnmente utilizados en el campo de investigación de Robots móviles [3]. Puesto que algunos de los simuladores descienden de plataformas para el desarrollo de video juegos resulta conveniente la evaluación de alguno de ellos para la solución planteada de esta investigación, por ende incluiremos a Unity Engine [18] como una alternativa.

Uno de los puntos importantes de esta investigación es la selección de la herramienta que más se adapte al proyecto, es por eso que se realizó la primera etapa de evaluación, como se muestra en la siguiente tabla.

Tabla 2.1: Evaluación de software etapa 1.

Plataforma	Gráficos	Robotic middleware	Simulación multi usuario	Comunidad de desarrolladores	S.O	Tipo de licencia
Gazebo	Muy buenos	ROS	n/a	Amplia	GNU / Linux	Open source
ARGoS	Buenos	n/a	n/a	Baja	GNU / Linux, MAC OS	Open source
Webots	Excelentes	ROS	Soportada	Media	GNU / Linux, MAC OS, Windows	comercial
V-Rep	Muy buenos	ROS	Soportada	Media	GNU / Linux, MAC OS, Windows	Open source
Vortex Dynamics	Excelentes	ROS	Soportada	Baja	GNU / Linux, Windows	comercial
Unity Engine	Excelentes	n/a	n/a	Extensa	GNU / Linux, MAC OS, Windows	Open source

Uno de los principales para la selección es que debe ser un software *Open source* y debe ser compatible con algún middleware robótico para trabajos de integración futuros, esto nos lleva a descartar automáticamente a ARGoS, Webot y Vortex Dynamics como una opción viable, por otro lado Unity Engine no queda descartado ya que a pesar de que no es directamente compatible con algún middleware hay investigaciones donde se logró comunicarlo con ROS y YARP obteniendo resultados favorables. Teniendo esto en cuenta se opta por evaluar a Gazebo, V-Rep y a Unity para la segunda etapa que consiste en utilizar a estos durante un periodo de 30 horas efectivas, donde se pretende obtener un manejo de las herramientas a un nivel intermedio, comprender los puntos fuertes de cada uno y evaluación de avance obtenido durante su periodo de prueba.

Durante la segunda etapa de evaluación se realizaron actividades para la familiarización con los software, conocer las herramientas de estos, módulos especiales y tener la noción de lo complejo que puede llegar a ser su operación o implementación de proyectos dentro de ellos.

Como se puede observar en la tabla 2.2 se da a conocer un conjunto de características específicas de ellos, donde nos podemos percatar de las ventajas que tienen uno sobre los otros, por ejemplo Gazebo y V-Rep resultan ser compatible con motores de física como ODE, Bullet entre otros mientras que Unity solo cuenta con PhysX. Por otro lado tenemos a V-Rep con sus módulos especializados para implementar detección de colisiones y realizar cálculos de cinemática directa e inversa mientras que estas acciones se pueden realizar en Gazebo o Unity estos no cuenta con un módulo especializado realizarlo. También se debe tener en consideración la compatibilidad con algún *middleware* robótico como ROS donde V-Rep y Gazebo con compatibles además de contar con una amplia documentación de cómo implementarlo dentro de su entorno pero como se mencionó anteriormente a pesar de que Unity no es compatible tal cual se puede establecer una comunicación con estos.

Tabla 2.2 Evaluación de software etapa 2(a).

CARACTERISTICAS	Software		
	Gazebo	V-Rep	Unity
Motor de física	✓✓	✓✓	✓
Soporte en la nube	✓	✓	✓
Com. TCP/IP	✓	✓	✓
compatibilidad de API Remota	✓	✓	**n/a**
Exportación de modelos CAD	✓	✓	✓
Detección de colisiones	✓	✓✓	✓
Cinemática Inversa y directa	✓	✓✓	✓
Adquisición de datos externo	✓	✓	✓
Middleware (Robótica)	✓✓✓	✓✓	✓
API externa de control	✓	✓	**n/a**
Compatible con Matlab	✓	✓	✓
HMI (interna en producto final)	**X**	**X**	✓
Instalación Multi Plataforma	**X**	✓	✓
Exportación Multi Plataforma	**X**	**X**	✓
Comunidad de desarrolladores	✓✓	✓	✓✓✓

Otro punto a tener en consideración es el hecho de que Unity a diferencia de Gazebo y V-Rep nos da la ventaja de implementar una HMI de control dentro de el sin la necesidad de tener que desarrollar una API externa en otro software como por Microsoft VisualStudio, otra facilidad que nos brinda es al momento de la creación de proyectos ya que este nos permite exportarlo a directamente como una aplicación para Windows, MAC OS o Linux inclusive como un archivo Web, todo esto sin la necesidad de realizar modificaciones dentro del código. También a pesar de no ser un software de desarrollo en el área de la robótica cuenta con una comunidad que lo emplea con ese propósito, además de la extensa

comunidad con la que ya cuenta ya que la finalidad de Unity es proporcionar a los usuarios una herramienta de desarrollo interactivo de 2D y 3D de la manera más accesible.

Dentro de las actividades realizadas en este periodo que abarca la operación y control del software obtenemos los resultados obtenidos en la tabla 2.3 donde se puede observar los resultados obtenidos en un periodo de tiempo anteriormente establecido y los resultados obtenidos en este.

Tabla 2.3 Evaluación de software etapa 2(b).

Actividades	Software					
	Gazebo		V-Rep		Unity	
	Status	**tiempo**	**Status**	tiempo	**Status**	tiempo
Instalación	✓	**3**	✓	**1**	✓	**2**
Compresión de la GUI	✓	**3**	✓	**3**	✓	**3**
Exportación de modelos SVG	✓	**1**	✓	**2**	✓	**1**
Exportación de modelos 3D CAD	✓	**2**	✓	**3**	✓	**1**
Construcción de escenas	✓	**5**	✓	**5**	✓	**2**
Escritura del API remota	**X**	**5**	**X**	**2**	**N/A**	**0**
Comunicación serial HMI	**X**	**3**	**X**	**2**	**N/A**	**0**
Importación de modelos de brazos robótico	**X**	**3**	**X**	**2**	✓	**1**
simulaciones	✓	**5**	✓	**5**	✓	**5**
simulaciones de brazos robóticos	**X**	**0**	✓	**5**	✓	**5**
comunicación con tarjeta adquisitoria	**X**	**0**	**X**	**0**	✓	**3**
Tiempo total (horas)		**30**		**30**		**23**

Como se puede notar en los resultados de la evaluación, se trató de realizar una serie de actividades que todos los software tienen en común, con algunas excepciones como son la “Comunicación serial a la UI” o “Escritura del API remota”, las cuales no son necesarios

dentro de Unity ya que este cuenta con herramientas que suplen estas funciones, esto permitió concentrarse en otras actividades y obtener avances significativos en comparación a Gazebo o V-Rep.

Puesto que la intención de realizar esta evaluación fue el conocer la eficiencia, potencia y practicidad de estas herramientas, se puede concluir que todas tienen sus puntos a favor y en contra donde de acuerdo al campo en el que se encuentra esta investigación podríamos definir como las mejores opciones a Gazebo o V-rep, ya que estos son software especializados en ambiente de robótica, pero estos demandan una cantidad de tiempo mayor para desarrollar habilidades que permitan el uso de estos a un nivel que se pueda implementar la solución al problema planteado. Cabe mencionar que los tiempos anteriormente mostrados fueron los que se invirtieron en dichas actividades más no el tiempo invertido en investigación y lectura de manuales de las herramientas para obtener conocimiento acerca del correcto manejo de estas, ya que resulta algo complicado el monitorearlo.

Por otro lado tenemos a Unity, un motor gráfico de video juegos que no cuenta con módulos especializados en el campo de la robótica, sin embargo cuenta con componentes muy similares a Gazebo o V-Rep y otras herramientas que lo vuelven competitivo. Teniendo esto en cuenta los resultados al evaluarlo fueron mejores con respectos a los demás, logrando un mejor rendimiento y en menor tiempo. Esto no significa que Unity sea mejor pero sí que mostro mejores resultados a corto plazo, por ende lo vuelve en la opción más viable para realizar el proyecto.

2.1 Herramientas de visualización

2.1.1 Visualización

La visualización es la acción de visualizar. Este verbo, hace referencia a desarrollar mentalmente una imagen de un objeto o proceso a un pensamiento abstracto, donde se le

otorgan características a aquello que se desea representar a través de imágenes u otros objetos.

2.1.2 Técnicas de visualización

La visualización científica es la transformación de datos a imágenes que permiten realizar el análisis de alguna situación en específico. La necesidad de desarrollar herramientas que permitan extraer información o que la muestren de manera correcta, en la actualidad es una de las principales necesidades, por lo cual en la existen diversos sistemas que se encargan de hacer esta función en diferentes campos aplicados.

A continuación se mostrarán algunos de los campos en los cuales se emplean:

- Interfaces de usuario.
- Software asistido por computadora CAD.
- Simulación y animación de objetos 3D.
- Sistemas SCADA.

2.1.3 Animación

Uno de los factores más importantes dentro de esta investigación es la animación ya que esta se encarga de dar la sensación de movimientos de los objetos, lo cual permite simular casos específicos, esto con la finalidad de comunicar información de manera visual ya sea por medio de manipulación de imágenes, de objetos, de escenas o todo aquello que permita aplicar operaciones a objetos sobre el tiempo, utilizando un modelo de tiempo discreto.

2.1.4 Realidad Virtual

La realidad virtual es un entorno de escenas, objetos o una aplicación que permite a los usuarios interactuar con un ambiente 3D generado en una computadora tomando características como la inmersión e interacción apoyados de gráficos 3D para brindar una experiencia completa al usuario.

2.1.5 Gráficos dinámicos

Los gráficos dinámicos son aquellos que permiten la interacción con aplicaciones, UI, animación en tiempo real, gráficos 3D y programas de realidad virtual. Una de las principales aplicaciones de estos en la actualidad es el desarrollo de software CAD los cuales permiten la creación de objetos y la animación con ciertas limitaciones que se reflejan dependiendo del evento de entrada que defina el usuario.

2.1.6 Interfaz de usuario (*User interface*)

La interfaz de usuario o UI es el medio por el cual los usuarios pueden establecer una comunicación con un ordenador, dispositivos portátiles o máquinas de uso general que cuenten con esta, cuyo objetico es la interacción entre humano y máquina para que este pueda manipularla de manera efectiva. Las UI se pueden clasificar en 3 tipos, tales son:

- Interfaz de línea de comandos o CLI (*Command line interface).*
- Interfaz gráfica de usuario o GUI (*Graphical User interface*).
- Interfaz natural de usuario o NUI (*Natural User interface*).

CLI: Es el medio por el cual los usuarios pueden interactuar con un programa o aplicación por medio de líneas de comando en forma de texto para ejecutar una serie de instrucciones.

GUI: Es el medio por el cual los usuarios pueden interactuar con un al ordenador o máquina, utilizando una serie de objetos como lo son imágenes o elementos 3D entre otros, esto con la finalidad de proporcionar un entorno visual de comunicación con el usuario. Un ejemplo claro es el sistema operativo de las computadoras.

NUI: Es el medio por el cual los usuarios pueden interactuar con un sistema o aplicación sin la necesidad de utilizar dispositivos de mando como lo son teclados o mouse entre otros, en vez de esto, emplean gestos de movimiento realizados con el cuerpo, manos u ojos.

2.1.7 Motor de física

El motor de física (*Physis engine*) [19] es un software que implementa algoritmos matemáticos para simular ciertos sistemas físico, como lo son los cuerpos rígidos, fluidos, dinámica, estática, etc…

Otra definición de un motor de física proporcionada por la empresa NVIDIA [20]; *es una unidad de procesamiento de física diseñada para calcular algoritmos de física de video juegos. Esto requiere unas características diferentes a las de una CPU de propósito general.*

2.1.8 Tecnología PhysX

PhysX [20] es un motor de física desarrollado por NVIDIA que atreves de su arquitectura de hardware permite ejecutar cálculos de física que da a lugar a entornos de video juegos increíbles y realistas, esto debido a sus procesadores de alta capacidad dotados de cientos de núcleos, agilizando todos los cálculos.

2.2 Robótica

La robótica es la técnica utilizada en el diseño y construcción de robots o dispositivos que realicen trabajos de caso particular, como lo son dentro de la industria viéndolo como la sustitución de la mano de obra humana.

2.3 Robot

Un robot según la definición del instituto de robots de américa [21]:“un robot es un manipulador reprogramable, multifuncional desenado para mover material, partes, herramientas, o dispositivos especializados atreves de la programación de movimientos para el desempeño de una caridad de tareas”

Teniendo esto en cuenta podemos notar que la particularidad de un robot es el ser reprogramable, esta es una gran ventaja ya que permite adecuarlo a realizar una serie de actividades de manera eficiente tomando en cuenta su espacio de trabajo y sus limitaciones.

CAPÍTULO 3

Modelado 3D

El modelado 3D es el proceso de representación matemática de cualquier objeto tridimensional a través de alguna herramienta que permita desarrollar la acción, el producto de esto se le conoce como modelo 3D.

Para el desarrollo de la investigación, el modelado 3D juega un papel de suma importancia y como se menciona en los capítulos anteriores, se pretende diseñar una interfase de usuario que permita la simulación 3D tanto del brazo robótico como su entorno, esto con la finalidad de facilitar al usuario el control y la programación de este.

Un aspecto importante es la selección de las herramientas que actualmente se utilizan para desarrollar la solución al problema de esta investigación, tales como la selección del motor gráfico, el lenguaje de programación y la plataforma de desarrollo. Después de realizar una evaluación se opta por usar las siguientes herramientas:

- *Unity Engine.*
- Lenguaje de programación en C#.
- *Software* CAD: *Solidworks*.
- *Software* CAD: *AutoDesk 3DS Max*.

En el caso de Unity Engine este paquete de programación cumple la función del motor gráfico. Este nos da la opción de escoger entre dos lenguajes de programación los cuales son *Java* y C#. Se decide optar por escoger la segunda opción. Para realizar los modelos en 3D se utilizan dos paquetes de programación muy conocidos, los cuales son *Solidworks* y *AutoDesk 3D Max*, esto debido a que el motor gráfico de *Unity* no procesa directamente los archivos generados por *Solidworks*. Por lo anterior, se ve la opción de exportar un archivo guardado como *STEP* hacia *AutoDesk 3DS Max*. En este último, es en

el cual se editan algunos eslabones del diseño 3D, como lo son los puntos de unión, uniones entre piezas y puntos de pivoteo para optimizar las articulaciones donde existen los giros.

El brazo robótico a modelar en 3D es el brazo articulado PUMA 2000 que actualmente se encuentra en el laboratorio LabCEM del instituto tecnológico de chihuahua, donde primeramente se utilizó el *Solidworks* para esta tarea, ya que el conocimiento acerca del manejo de esta herramienta es más elevado en comparación a *AutoDesk 3DS Max*, en el cual se obtuvieron los siguientes resultados:

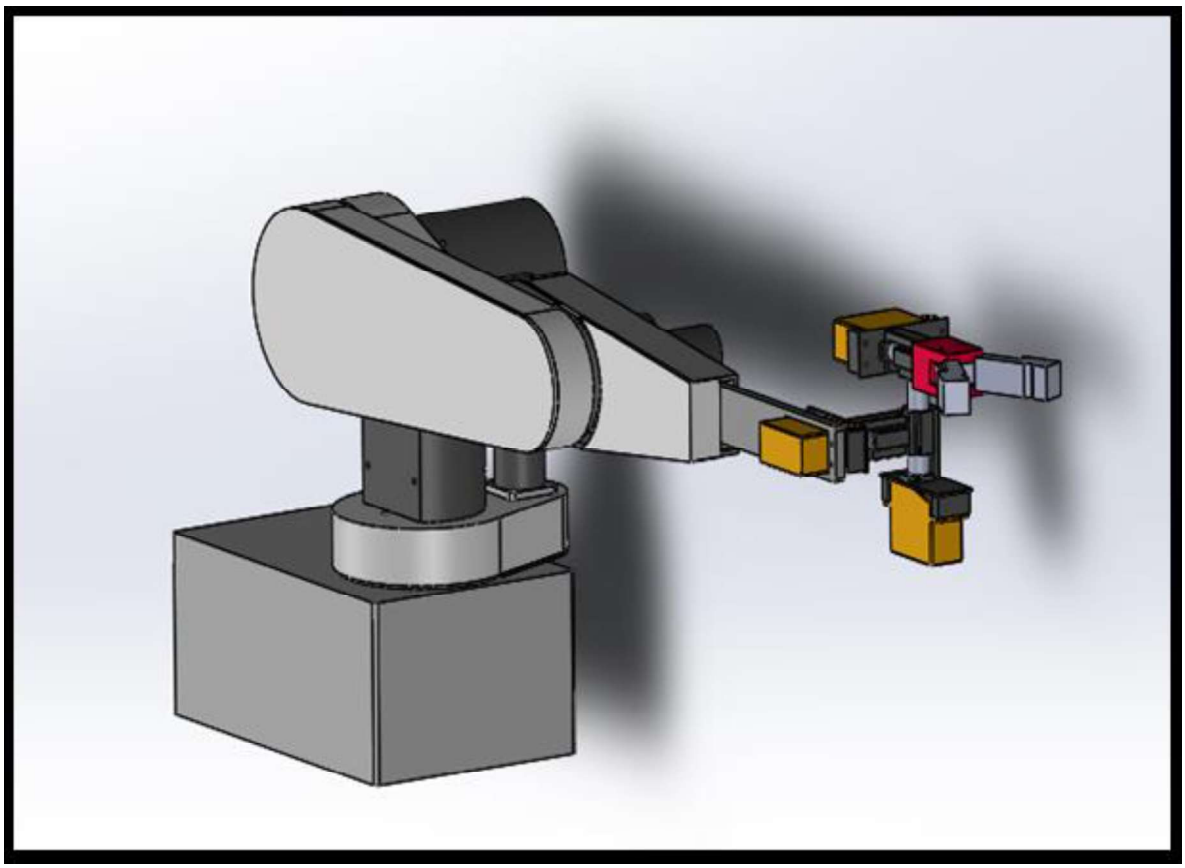


FIGURA 3.1: Brazo Robótico puma 2000 (Solidworks).

Como se puede observar tanto en la figura 3.1, como en la figura 3.2, es de apreciar que el modelado 3D del brazo robótico PUMA 2000 se realiza con las dimensiones de la estructura medidas a mano, ya que no se encontró documentación como manuales o planos de este, y el resultado fue lo bastante aproximado para los propósitos del presente trabajo.

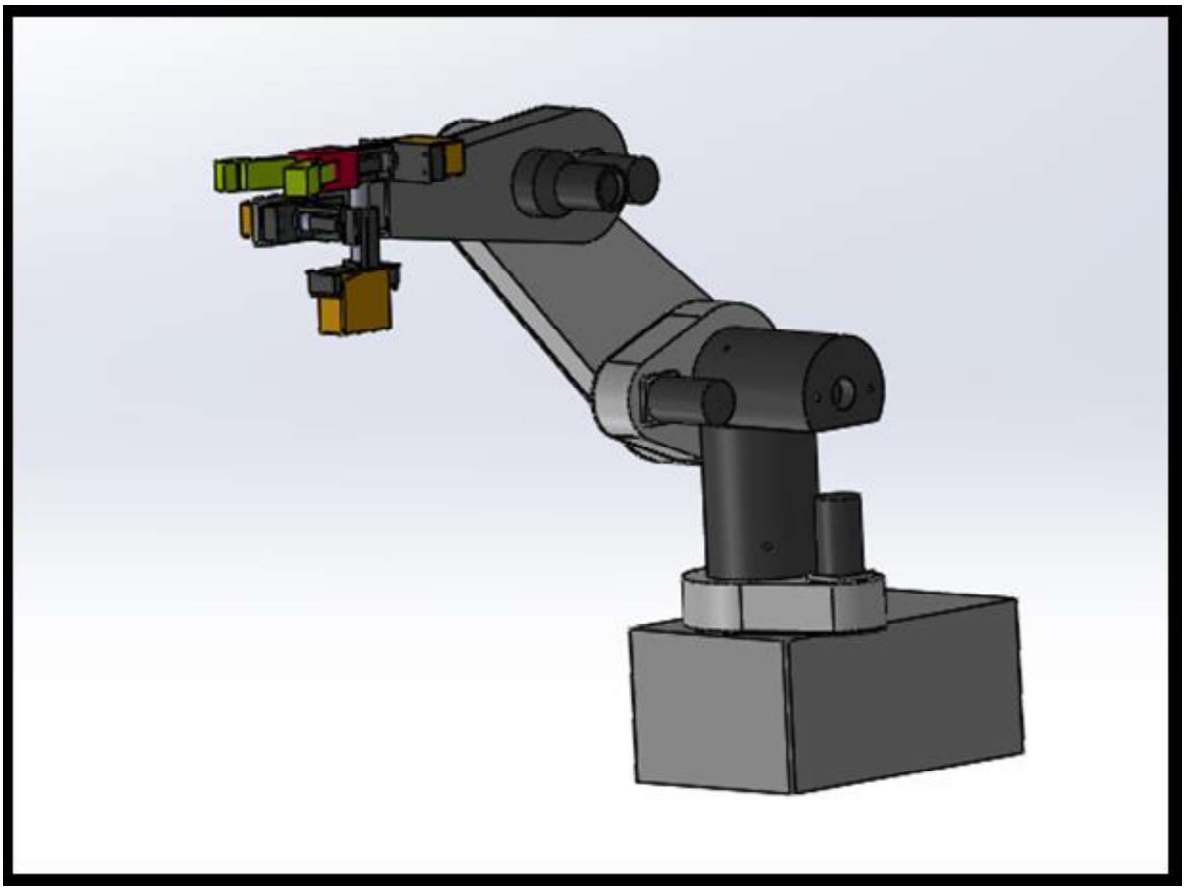


FIGURA 3.2: Brazo Robótico puma 2000 (*Solidworks*).

El siguiente paso después de realizar el modelado 3D, es exportarlo a *Unity*, pero como se menciona anteriormente, los archivos creados en *Solidworks* no son compatible con este, por lo cual, se investiga que paquetes de software son capaces de crear

modelos 3D en formatos compatibles entre ellos. Los paquetes de *Software* que se encuentran *son MAYA 3D, Blender* y *AutoDesk 3DS Max*. Donde por el tipo de acción que se desea realizar cualquiera hubiera cumplido con el propósito, pero dentro de los foros de la comunidad de Unity se recomienda que las mejores herramientas para el propósito establecido es mejor utilizar *MAYA 3D* y *3DS Max*, con la diferencia que *3DS Max* es más utilizado entre los desarrolladores así que por esto se optó por escogerlo.

Al principio solo se tenía planeado utilizar a *3DS Max* para exportar el modelo 3D del brazo robótico a un formato compatible con Unity, pero después de realizar ciertas pruebas en *3D Max* resulto ser una herramienta con un gran potencial, así que para la tarea de virtualización del laboratorio LabCEM, se toma la decisión de ser desarrollada dentro del ambiente de desarrollo de *3D Max*.

Como se puede observar en la figura 3.3, se puede apreciar cómo se modifican los modelos después de realizar la exportación de *Solidworks* a *3DS Max*, donde todos los sólidos se convirtieron en mallas formadas a base de triángulos, así como se puede apreciar en la figura.

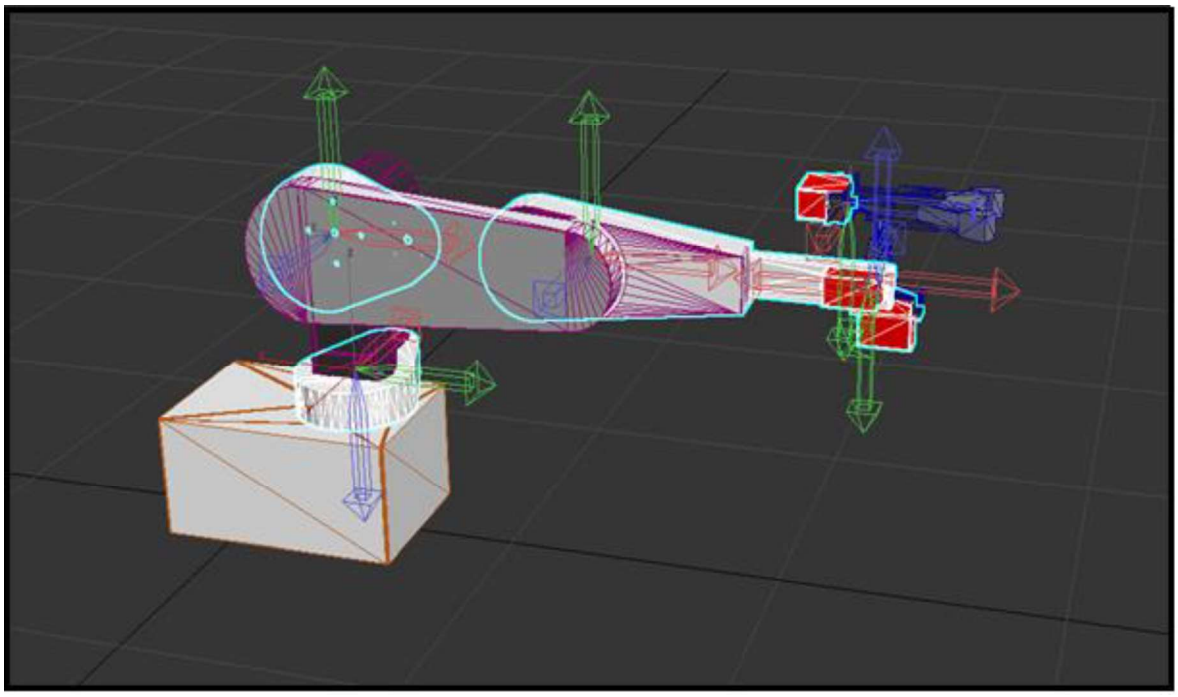


FIGURA 3.3: Brazo Robótico puma 2000 (*3DS Max*).

En la figura 3.4 y figura 3.5 se aprecia el proceso de virtualización del laboratorio con ayuda de un celular *ASUS ZenFone AR* y la aplicación *Scandy Pro*.

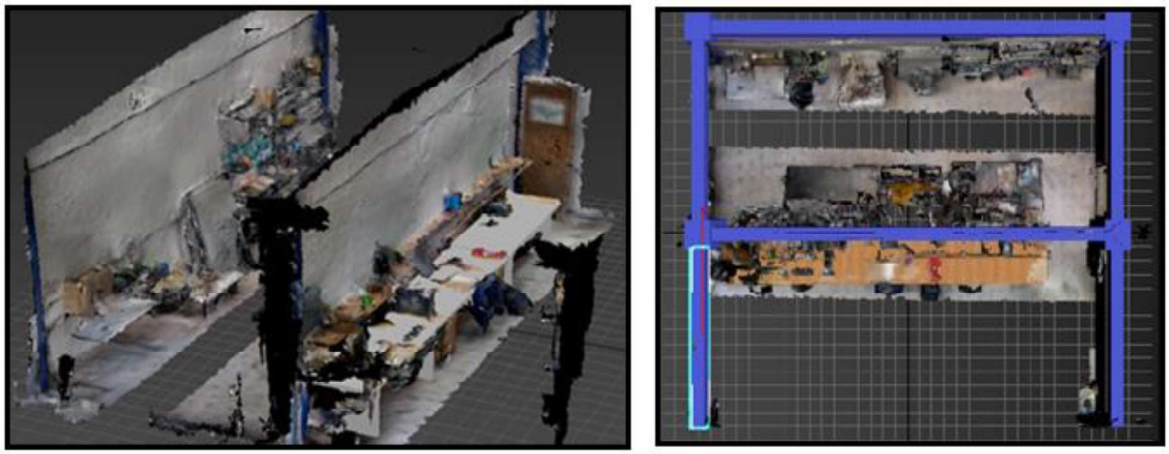
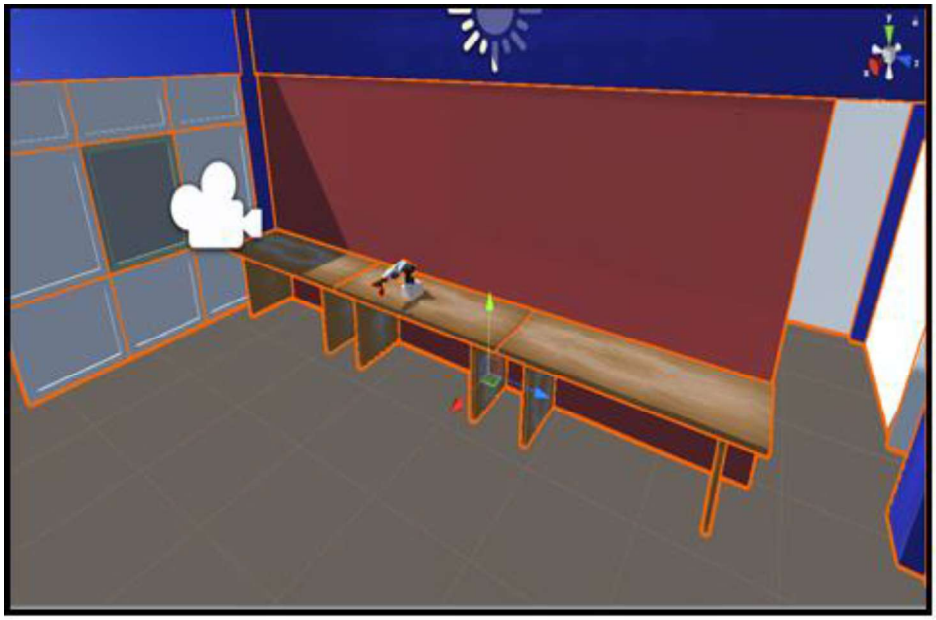


FIGURA 3.4: Resultados del escaneo utilizando el *ZenFone AR* en *3DS MAX*.

FIGURA 3.5: Virtualización del laboratorio LabCEM (*3DS MAX*)

Como se observa en la figura 3.4 los resultados del escaneo son relativamente buenos, pero para facilitar el manejo de los objetos se opta por tomarlo como una base para modelar el laboratorio LabCEM, así como se muestra en la figura 3.5.

Una vez que se realiza todo el procedimiento para terminar la virtualización del laboratorio LabCEM y la edición del brazo robótico dentro de *3DS Max* se prosigue con la exportación a un formato FBX que es compatible con *Unity* para obtener los resultados que se observan en la figura 3.6.



.

FIGURA 3.6: Resultados de la exportación de los modelos de *3DS Max a Unity (a)*.

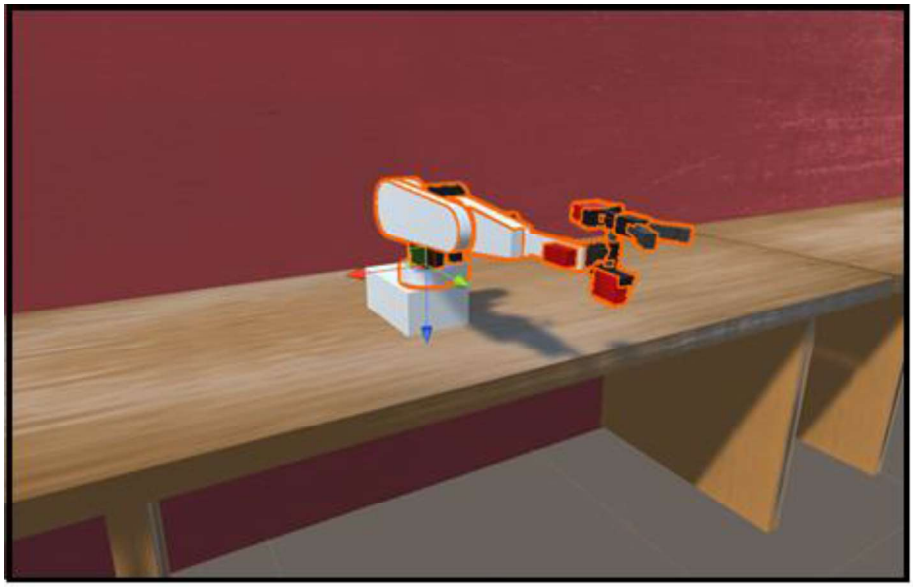


FIGURA 3.7: Resultados de la exportación de los modelos de *3DS Max* a *Unity (b).*

Tanto en la figura 3.6 como en la figura 3.7 se pueden apreciar detalles del modelo del ambiente de trabajo en el cual se encuentra ubicado el brazo articulado, reduciendo a su mínimo la cantidad de elementos en el emulador para evitar realizar proceso innecesarios al momento de realizar la simulación-emulación de los movimientos del brazo articulado y que solo se encuentren aquellos objetos con los que el robot tiene que interactuar.

Por otro lado, es importante resaltar que en el caso del modelo del brazo articulado, si se utiliza a más detalle la estructura de su morfología, con el fin de observar el comportamiento cinemático de toda la cadena de eslabones y articulaciones al realizar los diversos movimientos.

Antes de continuar con los avances dentro de *Unity* al momento de exportar los *3D* es importante conocer algunos conceptos que Unity maneja como se mostraran a continuación.

3.1 *GameObjects u object*

Un *GameObject* [18] es el concepto más importante en el Editor de *Unity*, Cada objeto en su proyecto es un *GameObject*, desde personajes, objetos hasta luces y cámaras. Sin embargo, un *GameObject* no hace nada por sí mismo, es necesario darle propiedades para que este realice la función para el cual fue programado.

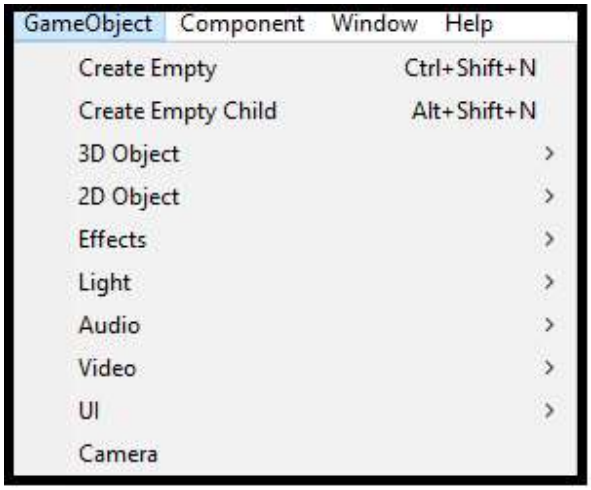


FIGURA 3.8: Diferentes tipos de *GameObjects* en *Unity*.

3.2 *Sprites*

Los Sprites son un *GameObjec*t en formato 2D. A diferencia de un objeto 3D, los Sprites son esencialmente texturas o imágenes, pero hay técnicas para combinar y manejar texturas de sprites por lo eficiente que resulta Unity al momento de realizar una renderización mejorando el aspecto visual, en la figura 3.9 se puede observar algunos elementos de sprites que se utilizaron en el proyecto .

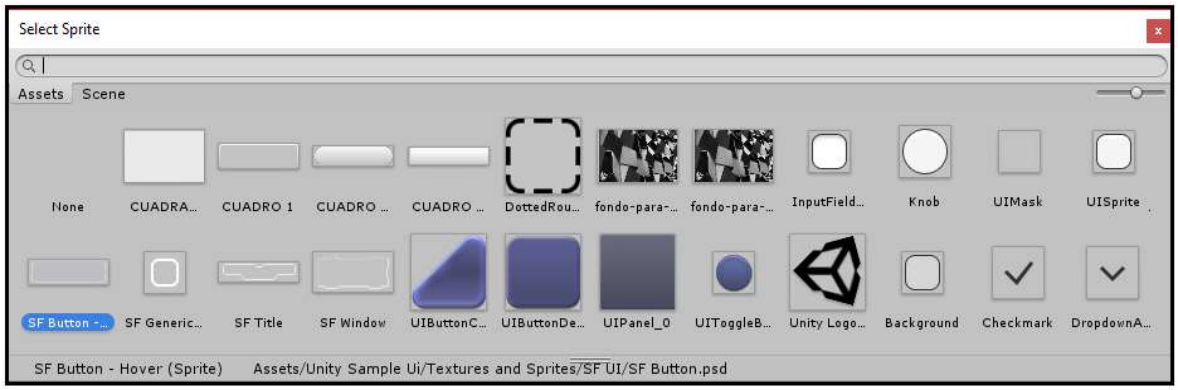


FIGURA 3.9: Ventana de selección de sprites.

3.3 Escenas

Las escenas es lo que contiene los entornos, objetos, luces o menús. Cada escena debe ser considera como un área de trabajo independiente y única. En cada una se coloca sus entornos y o *GameObject.*

3.4 Ventana de Jerarquía (*Hierarchy*)

La ventana de Jerarquía (asi como se muestra en la figura 3.9) contiene una lista de cada *object* en la escena. A medida que estos son agregados o borrados en la escena irán apareciendo y desapareciendo de la ventana de Jerarquía, Estos por defecto son listados en la ventana de en el orden en el que fueron creados.

3.5 Parentesco (*Parenting*)

Unity utiliza un concepto llamado *Parenting*. Al crear un grupo de objetos, esto se logra creando un *object* dentro de otro *object,* Al *object* principal se le conocer como "objeto padre", y todos los objetos agrupados dentro de este son llamados "objetos hijo", esta estructura de puede observar en la figura 3.10.

3.6 Transform

El componente *Transform* determina la position, rotación, y escala de cada objeto en la escena. Cada GameObject cuenta con su elemento de *Transform,* Estos valores son medidos del padre del objeto.

- **Posición:** Posición del *Transform* en coordenadas X, Y, Z.
- **Rotación:** Rotación del *Transform* alrededor de los ejes X, Y, Z, medido en grados.
- **Escala:** Escala de la *Transform* a lo largo de los ejes X, Y y Z el valor original es unitario.

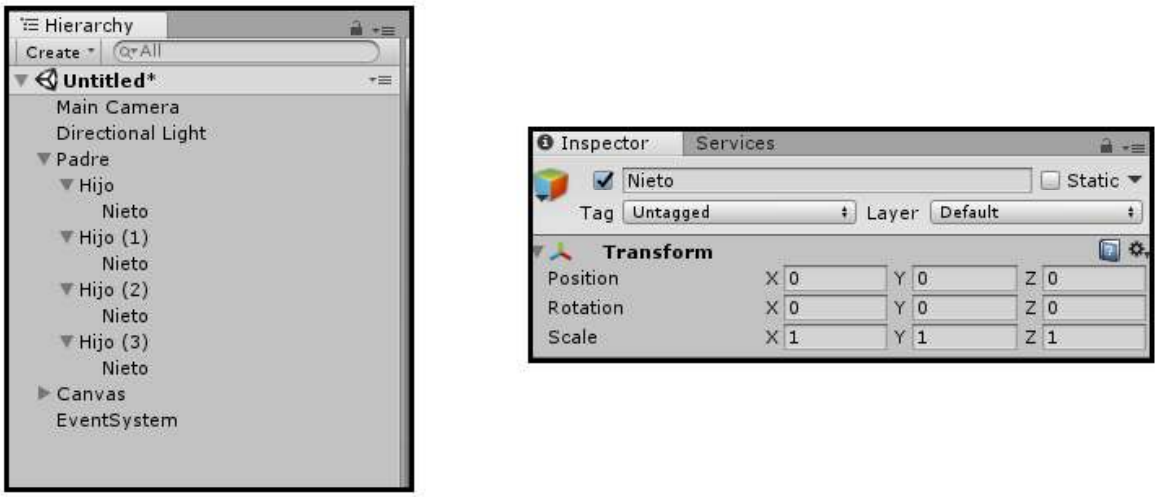
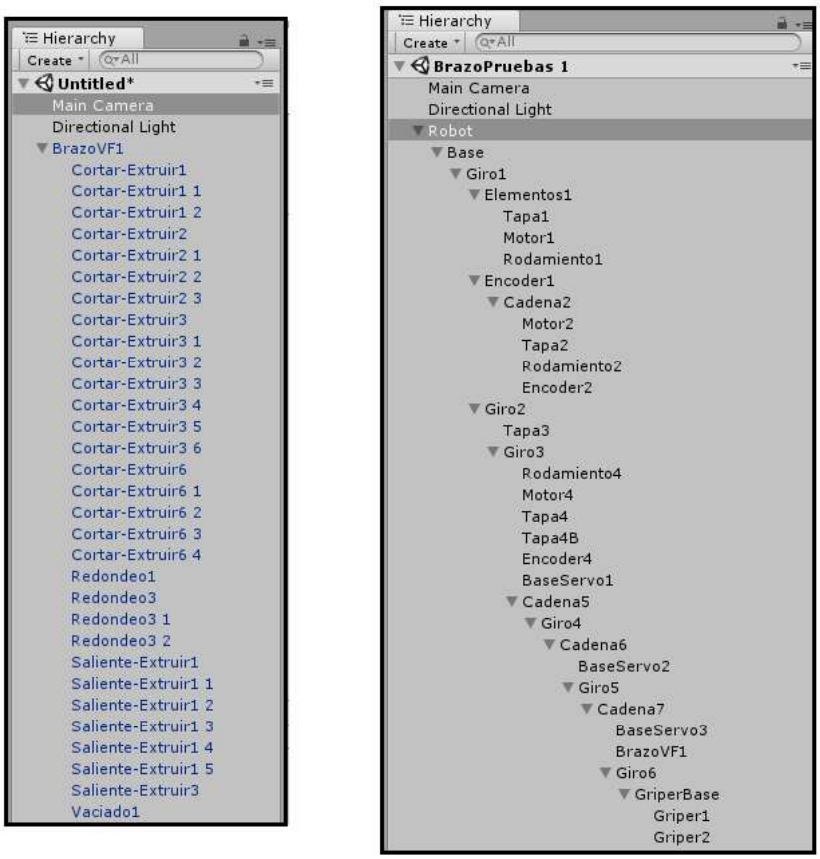


FIGURA 3.10: Ventana de Jerarquía y ventana del *Transform.*

Como se puede observar en la figura 3.11 al importar el brazo robótico e ingresarlo a la escena como un *GameObject*, este importo una jerarquía por *default* que no favorecía las necesidades del proyecto, por lo cual se tuvo que realizar un acomodo logrando una estructura de padres e hijos que logran cumplir con la finalidad del proyecto.

Una vez realizada la estructuración de la jerarquía se concluye exitosamente la importación del brazo robótico de *Solidworks* a *3dsMax* hasta *Unity* obteniendo los



resultados esperados.

FIGURA 3.11: Ventana de Jerarquía y ventana del *Transform.*

Como se pudo observar atreves del capítulo se dio una explicación acerca del entorno de *Unity* además del proceso de importación dentro diferentes software como lo fueron 3dsMax y Solidworks donde a pesar de existir una herramienta dentro de Unity que ya realiza la exportación desde Solidworks a Unity durante este periodo de investigación se optó por realizar este procedimiento debido a que la herramienta que realiza la acción de exportación de Solidworks a Unity es una extensión de paga del software donde para

fines prácticos del proyecto no fue necesario adquirirlo y desarrollarlo de la manera como se explica a lo largo del capítulo fue la mejor opción.

CAPÍTULO 4

Desarrollo

4.1 Interfaz de usuario

Uno de los objetivos de este proyecto es desarrollar una interfaz de usuario que sea ergonómica, intuitiva y fácil de usar para el usuario final. Por siguiente a continuación se mostrara el proceso de desarrollo de esta.

Dentro de las estrategias de diseño las funciones dependen principalmente de las tareas que se pretenden realizar para este proyecto en particular son las siguientes:

- Ventana principal de opciones.
- Configuración del puerto de comunicación.
- Módulo de simulación.
- Cálculos de cinemática directa.
- Módulo de detección de colisiones.

Teniendo en cuenta que en este caso de estudio se está trabajando con un brazo rotico puma 2000 el aspecto visual de este es de suma importancia, además de los elementos del entorno donde este se encuentra, estos deben ser considerados dependiendo del tipo de usuario para el cual está enfocado, que en este caso son usuarios que están capacitados por lo menos con un curso básico de robótica para que ellos estén familiarizado con las funciones que esta interface contendrá.

Dentro de *Unity* existen herramientas especiales para la creación de interfaces de usuario a continuación se mencionaran y explicaran algunas de ellas:

4.1. *Canvas*

El *Canvas* es el área donde todos los elementos UI *(User Interface*) están. El *Canvas* es un *Game Object* con un componente *Canvas* en él, y todos los elementos UI deben ser hijos de dicho *Canvas*.

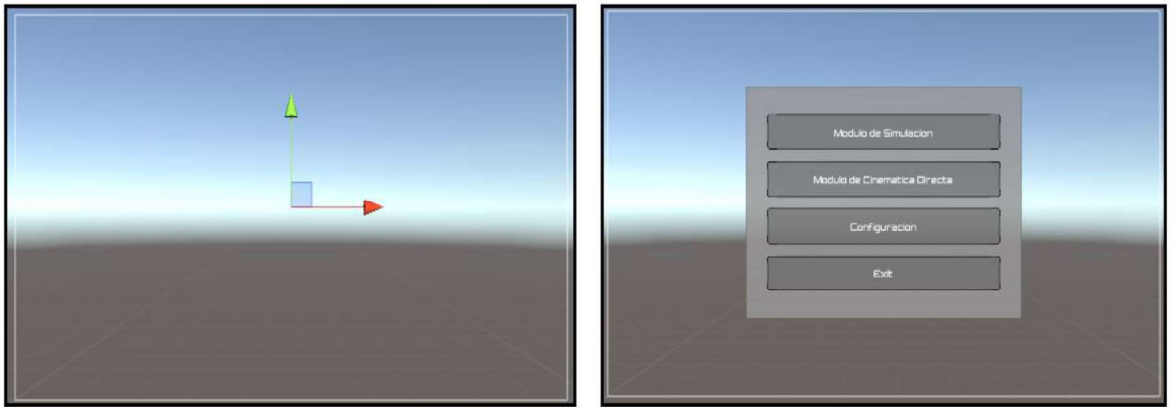


FIGURA 4.1: *Canvas* y aplicación de este.

4.2 Componentes de Interacción

Los componentes en el sistema UI pueden manejar las interacciones, tal como los eventos del *mouse* o táctiles y la interacción utilizando un teclado o controlador. Estos no son visibles por ellos mismo, y deben ser combinadas con uno o más elementos visuales con el fin de que funcionen correctamente.

4.2.1 *Button* (Botón).

El botón está diseñado para iniciar una acción cuando el usuario hace *click* y lo suelta. Si el *mouse* se mueve del control del botón antes de que el *click* se haya soltado, la acción no toma lugar. El botón tiene un solo evento llamado *On Click* que responde cuando el usuario completa un *click*. Un caso de un uso típico incluye:

- Confirmando una decisión.
- Moviéndose a un sub-menú en un GUI.
- Cancelando una acción en progreso.

FIGURA 4.2: Botón (*Button*).

4.2.2 *Slider* (Deslizador)

Un *Slider* (Deslizador) tiene un número (*Value)* decimal el cual el usuario puede arrastrar entre un valor mínimo y máximo. El valor es determinado por la posición de la manija a lo largo de su longitud. El valor aumenta de un valor máximo hasta un mínimo en proporción a la distancia que la manija que es arrastrada. El comportamiento del deslizador aumente de izquierda a derecha. Este solo tiene un evento llamado *On Value Changed* que responde a medida que el usuario arrastre la manija. El valor numérico actual del deslizador es pasado a la función de un parámetro *float*. Casos típicos de uso incluye:

- Escogiendo un nivel de dificultad en el juego, brillo de una luz, etc.
- Configurando una distancia, tamaño, tiempo o ángulo.

FIGURA 4.3: Slider (Deslizador).

4.2.3 *Input Field* (Campo de Input)

Un *Input Field* es una herramienta que permite hacer que el texto de un contenido sea editable. Para obtener el texto del *Input Field*, se debe utilizar la propiedad de texto en el componente, la propiedad de texto del componente *Text* mostrara el texto.

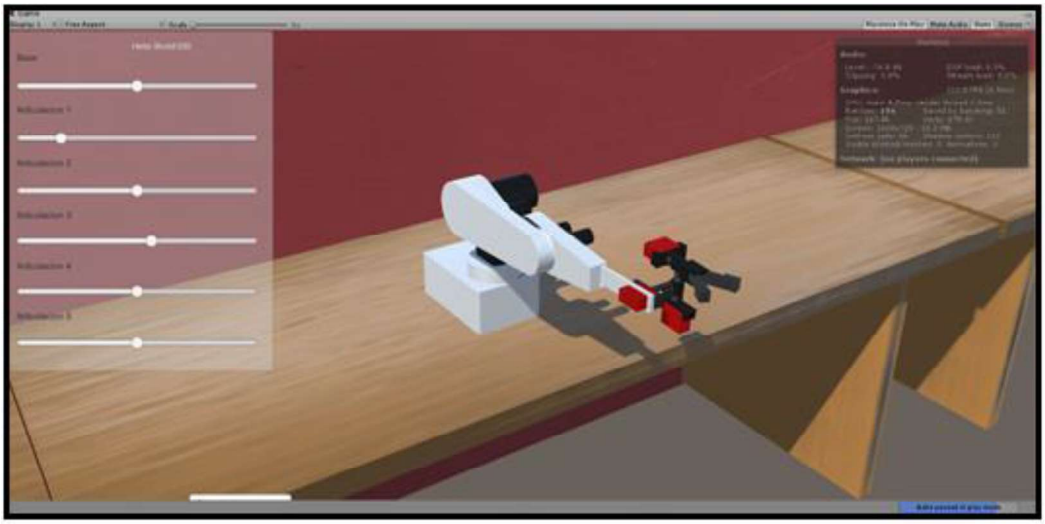
FIGURA 4.4: Botón Input Field (Campo de *Input*).

4.3 Evolución de la interface.

Como se puede observar en la Figura 4.5 y 4.6 se presenta la versión 1.0 de la interfaz de usuario, esta contiene la mayoría de sus componentes anteriormente mencionados tales como son el uso del *canvas, buttons* y *slider* además de la distribución de las herramientas a las cuales el usuario puede acceder tales como el módulo de simulación y el módulo de cinemática directa.

FIGURA 4.5: Prototipo de interfaz Versión 1.0

Debido a la evolución en el proceso de desarrollo del proyecto, se mejoró algunas habilidades que ayudaron a obtener un mejor aspecto visual en la UI, progresando en el uso de herramientas que *Unity* ofrece, tal cual es el uso de *sprites,* que gracias a



software externos a *Unity* como son *photoshop* e *illustrator*, Se logró personalizar los elementos de la UI, tales como son la imagen de fondo, logos de la aplicación, forma de botones, forma del *slider* y con esos cambios obtener una UI más atractiva para los usuarios, como se puede observar en las figura 4.8, figura 4.9 y figura 4.10.

FIGURA 4.6: Prototipo de interfaz Versión 1.0

Para realizar el cambio de transición dentro de la UI hacia algún modulo seleccionado, a los botones se les proporciono de 2 eventos, estos son eventos *On Click (),* el cual permite a un *Game Object* realizar una rutina previamente programada al hacer *click* sobre él. En nuestro caso el primer evento se encarga de ocultar la ventana la ventana actual y el segundo se encarga a acceder alguno de los módulos previamente seleccionado o volver a cargar el menú principal.

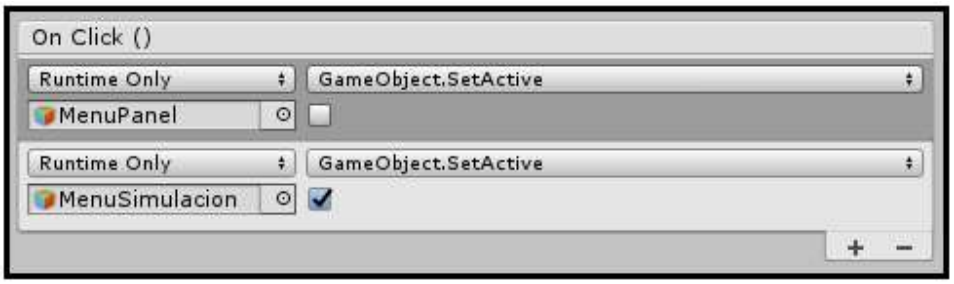


FIGURA 4.7: Ventana de configuración de eventos *On Click*.

FIGURA 4.8: Ventana del menú principal.

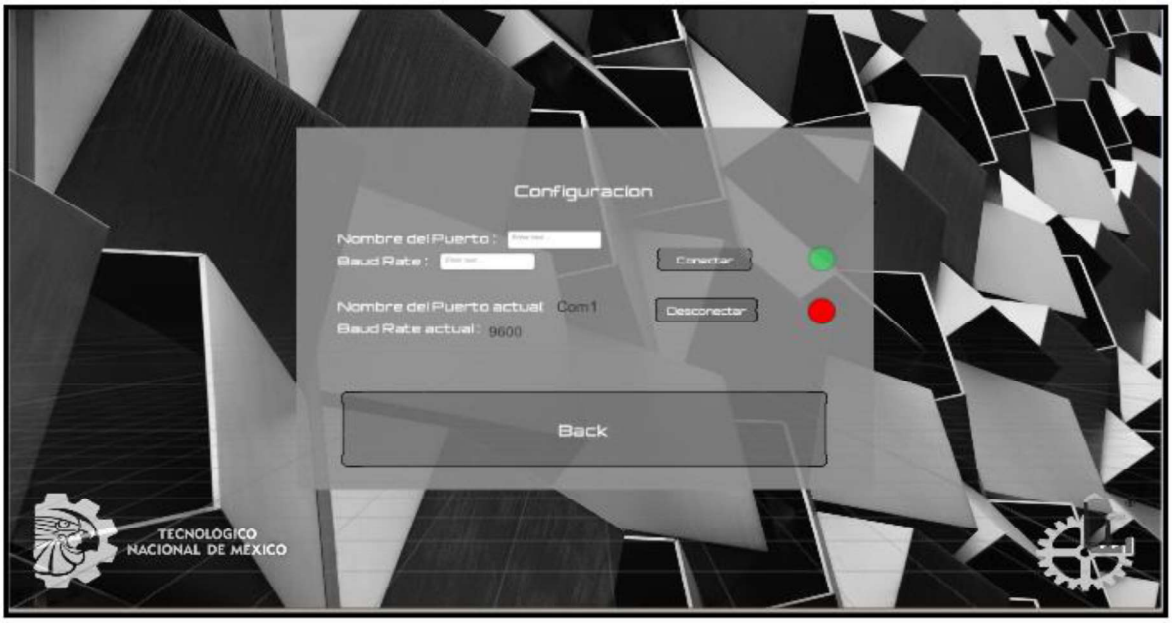


FIGURA 4.9: Módulo de configuración de puerto de comunicación.

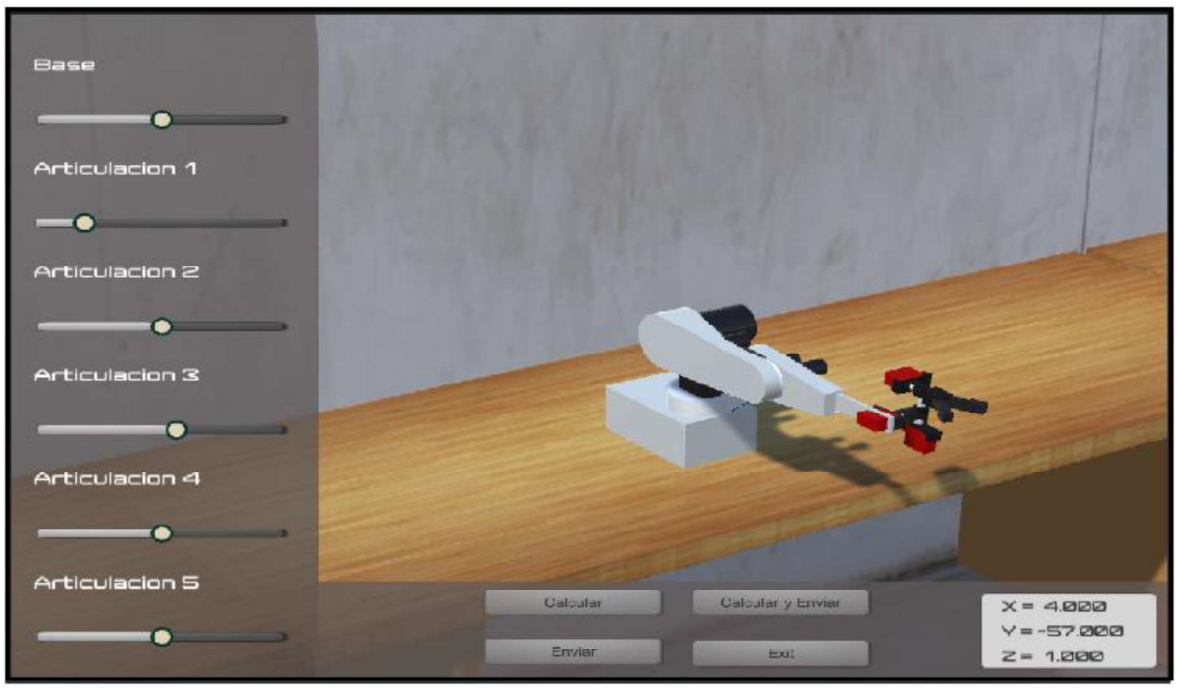


FIGURA 4.10: Módulo de simulación de cinemática directa.

En la figura 4.9 se puede observar el módulo de configuración de puerto que por medio de *text box* permiten la configurar el puerto de comunicación y la *baud rate* al igual que se añadieron 2 *sprites* indicadores de *status*, los cuales muestran si está establecida la comunicación o si esta desactivada.

Como se puede observar en la figura 4.10, se apreciar los cambios estéticos a la versión de prueba que se ve en la figura 4.6, mejorando el aspecto de las texturas del ambiente de simulación del robot además de personalizar los *Game Objects* tales como *slider*, botones y ventanas de texto, esto con el fin de brindar al usuario una UI intuitiva y fácil de usar. Además de los cambios estéticos se agregaron 4 botones que acceden a 4 *script*s diferentes por medio de un evento *On Click*, realizando una función en específico como el cual es el acceder al *script* que se encarga de calcular la cinemática directa y

enviar los datos de posición a la tarjeta de control del brazo robótico (esto se explicara claramente en el capítulo 5).

Los *slider* son los encargados de mover cada una de las articulaciones del brazo robótico por medio de un evento llamado *On Value Changed*, que básicamente se encarga de acceder al script de posición al detectar un cambio de valor del slider, sus valores representan el ángulo en el cual se encuentra la articulación del robot que esta expresado en grados y sus valores van de 0 a 359 grados.

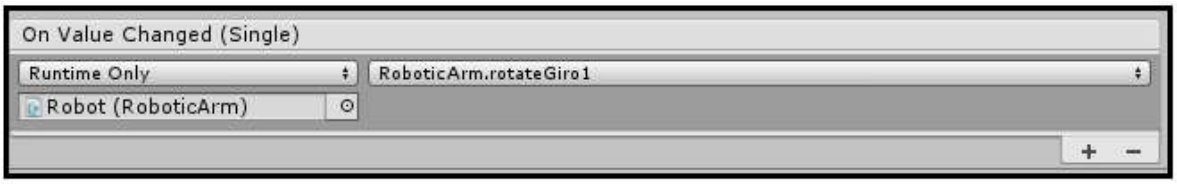


FIGURA 4.11: Configuración del evento *On Value Changed*.

Para realizar rotaciones de objetos dentro de *Unity* existen diversas herramientas que facilitan dicho acción entre las funciones podemos encontrar que satisfagan las necesidades del proyecto se encuentran:

- *Transform.Rotate*
- *GridBrush.Rotate*
- *Vector3.RotateTowards*
- *RotationDirection*
- *Quaternion*

Todas estas funciones pertenecen a las librerías de "*Unity Engine"* y "*System.Collections"* y después de una serie de pruebas se opta por utilizar la propiedad de "Euler" derivadas de la función "*Quaternion*".

Los *Quaternion* son una extensión de los números reales, similar a la de los números complejos. Donde los números complejos son una extensión de los reales por la adición de la unidad imaginaria ***i***, tal que $i^2 = -1$, los *quaternion* son una extensión generada de manera análoga añadiendo las unidades imaginarias: *i, j* y *k* a los números reales y tal que $i^2 = j^2 = k^2 = -1$. Los *Quaternion* dentro de *Unity* rara vez se vez modifica componentes individuales(x, y, z, w) en *Unity*, Comúnmente se emplea para realizar rotaciones ya existentes como por ejemplo desde el componente de la *Transform* construyendo nuevas rotaciones como por ejemplo interpolando entre 2 componentes de rotación. Las propiedades disponibles en Unity dentro de la función *quaternion* son las siguientes:

- *Quaternion.LookRotation*
- *Quaternion.Angle*
- *Quaternion.Euler*
- *Quaternion.Slerp*
- *Quaternion.FromToRotation*

Donde la más apropiada para satisfacer las necesidades del proyecto es la propiedad de Euler. Donde esta devuelve una rotación que gira alrededor del eje "Z", alrededor del eje "X", "Y" y "Z". Y esta se declara de la siguiente manera:

"Public static Quaternion.Euler (float x, float y, float z);"

La implementación de esta función en el proyecto se puede apreciar en la figura 4.12 donde esta función es activada cada vez que hay un cambio de la posición en grados dentro de los sliders cambiando el valor de la posición de cada articulación.

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

public class RoboticArm : MonoBehaviour {

// se vuelven publicas el componene transform de cada articulacion
    public Transform Giro1;
    public Transform Giro2;
    public Transform Giro3;
    public Transform Giro4;
    public Transform Giro5;
    public Transform Giro6;

// funciones que se activa al modificar el valor de slider 1//
    public void rotateGiro1(float val)
    {
        Giro1.localRotation=Quaternion.Euler(180,0,val);// ORIGINAL
        x1= val: // se guarda el valor de la nueva posicion del brazo robotico
        Debug.Log ("El angulo es  = " + val);// se imprime el resultado en consola
    }

    public void rotateGiro2(float val)
    {
        Giro2.localRotation=Quaternion.Euler(0,-90,val);
        x2= val: // se guarda el valor de la nueva posicion del brazo robotico
        Debug.Log ("El angulo es  = " + val);// se imprime el resultado en consola
    }

    public void rotateGiro3(float val)
    {
        Giro3.localRotation=Quaternion.Euler(0,180,-90 + val);
        x3= val: // se guarda el valor de la nueva posicion del brazo robotico
        Debug.Log ("El angulo es  = " + val);// se imprime el resultado en consola
    }
```

FIGURA 4.12: Implementación de rotaciones utilizando la función "Quaternion.Euler"

Como se demostró en el desarrollo de la UI se concluye que las herramientas de desarrollo que ofrece *Unity* satisfacen las necesidades del proyecto logrando los resultados esperados en el aspecto funcional de la UI y además brindando atributos visualmente atractivos.

4.4 Estudio Cinemático

El estudio cinemático consiste en el estudio analítico de la geometría del movimiento de los elementos robóticos con respecto a un sistema de coordenadas fijo, esto sin considerar la fuerza o momentos que el robot requiere para realizar sus movimientos.

Para realizar este analisis de posición se debe de dividir en 2 problemas el cuales son los problemas de cinemática directa e inversa. En la figura 5.1 se puede observar la relación que existe entre la cinemática directa e inversa. En este capítulo se mostrara el desarrollo matemático del brazo robótico PUMA TQ2000 abordando el problema de la cinemática directa de este.

4.4.1 Estudio Cinemático directo

El problema de la cinemática directa consiste en determinar la posición (x, y, z) y orientación de efector final o herramienta (Φ, π, δ) en términos de variables de las articulaciones (θ1, θ1…. θn) encontrando la posición cartesiana y orientación tomando en consideración el sistema robótico.

4.4.2 Estudio Inversa

El problema de la cinemática inversa consiste en determinar el movimiento de una cadena de articulaciones (θ1, θ1…. θn) teniendo como objetivo encontrar el valor que debe tomar cada una para ubicar el extensor final una posición en el espacio (x, y, z).

En nuestro caso, para solución el problema en el proyecto se opta por utilizar el estudio de cinemática directa para asi poder caracterizar el espacio de trabajo que el brazo robótico PUMA TQ2000 y asi obtener el modelo matemático de este para asi poder estudiar su comportamiento.

4.5 Método de análisis de la cinemática directa

En la robótica, una de las formas para obtener el modelado cinemático directo de un brazo robótico es la metodología de Parametrización de *Denavit-Hartenberg* (D-H), desarrollada por Jacques Denavit y Richard Hartenberg en 1955 con el propósito de estandarizar la ubicación de los sistemas referenciados a los eslabones espaciales, pero no fue hasta 1971 que Richard Paul demostró su valor para el análisis cinemático de sistemas

robóticos. Este método matricial que permite establecer de manera sistemática un sistema de coordenadas {Si} ligado a cada eslabón i de una cadena articulada.

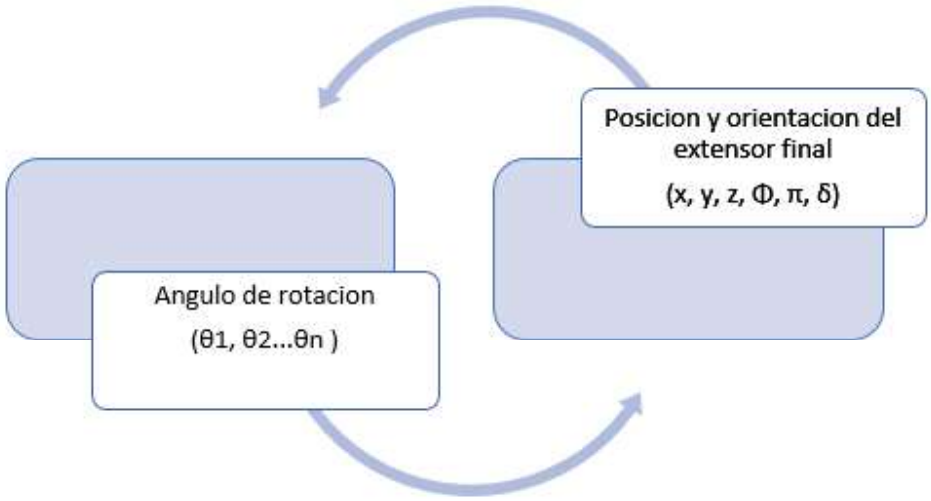


FIGURA 4.13: Relación entre cinemática directa e Inversa

Parámetros D-H

Para definir un sistema implementando la metodología D-H es necesario realizar los siguientes parámetros:

- Rotación alrededor del eje z_{i-1} un ángulo θ_i.
- Traslación a lo largo de z_{i-1} una distancia d_i.
- Traslación a lo largo de x_i una distancia a_i.
- Rotación alrededor del eje x_i un ángulo α_i

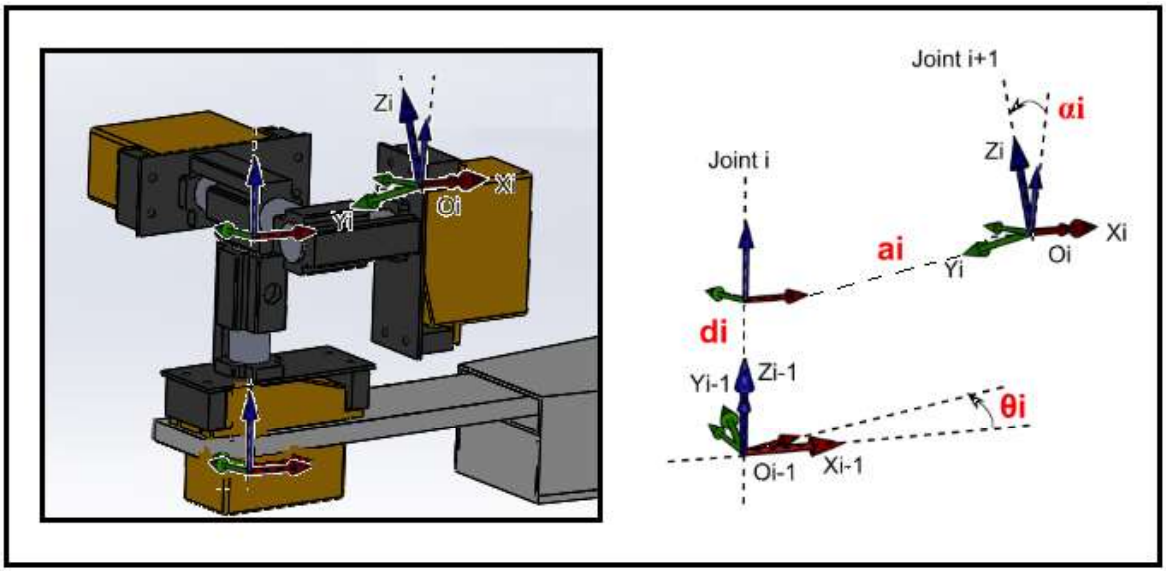


FIGURA 4.14: Representación de los parámetros necesarios para aplicar el método de D-H en los últimos 3GDL del efector final.

Definir la matriz D-H

Para realizar el objetivo del proyecto es necesario determinar el algoritmo cinemático directo por medio de la matriz D-H. Para aplicar el método se usa el procedimiento Denavit-Hartenberg que se muestra a continuación:

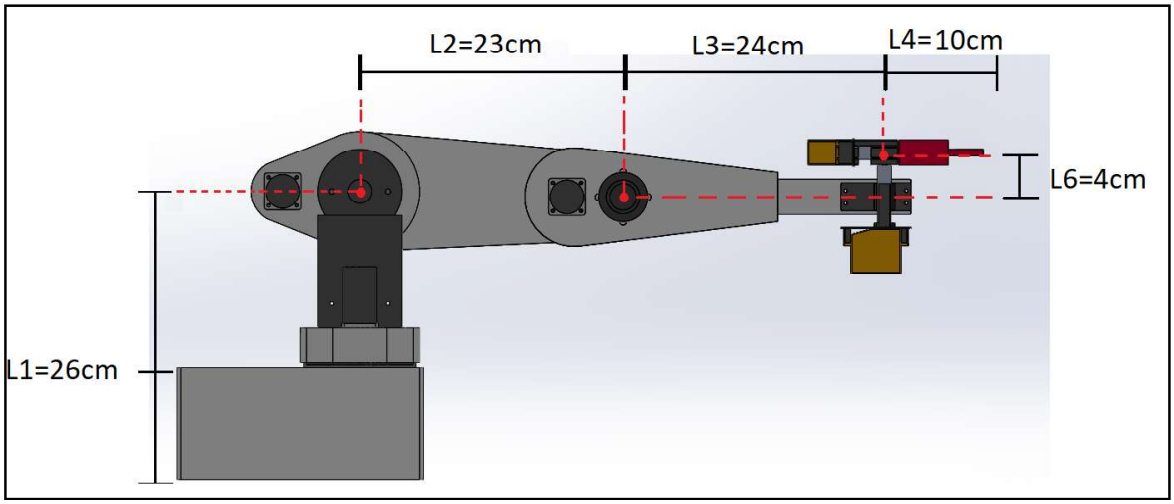
1. **Numerar los eslabones**: La base del robot se enumerara como "0" y se enumerara como "1" el primer eslabón móvil y "2" al siguiente eslabón móvil, así hasta terminar la cadena cinemática.
2. **Numerar las articulaciones**: Se enumera cada una de las articulaciones del robot comenzando desde "1" este siendo el primer grado de libertad, y como "n" la última articulación.
3. **Localizar el eje de cada articulación**: Para articulaciones rotativas, el giro será en su eje. En caso de ser articulación prismática el eje será a lo largo de esta produciendo movimiento lineal en el eslabón.

4. **Ejes Z**: Empezamos a colocar los sistemas cartesianos {Si} (xyz). Situamos los Z_{i-1} en los ejes de las articulaciones i, con i=1 hasta n. Es decir, Z_0 va sobre el eje de la articulación "1", Z_1 va sobre el eje "2" hasta n donde Z será igual a Z_{n-1}, etc.
5. **Sistema de coordenadas 0**: Se sitúa el origen del sistema $\{S_0\}$ en cualquier punto a lo largo de Z_0. La orientación de X_0 y Y_0 puede ser arbitraria, siempre que se respete evidentemente que $\{S_0\}$ sea un sistema dextrógiro.
6. **Resto de sistemas**: Para el resto de sistemas desde i=1 hasta n_{i-1}, colocar el punto origen en la intersección de Z_i con la normal común a Z_i y Z_{i+1}. En caso de cortarse los dos ejes Z, colocarlo en ese punto de corte. En caso de ser paralelos, colocarlo en algún punto de la articulación i_{+1}.
7. **Ejes X**: Situar cada Xi en dirección de la normal común a Z_{i-1} y Z_i, en la dirección de Z_{i-1} hacia Z_i.
8. **Ejes Y**: Una vez situados los ejes Z y X, el eje Y tiene su direcciones determinadas por la restricción de formar un sistema {Si} dextrógiro.
9. **Sistema del extremo del robot**: Situar el sistema {Sn} en el extremo del robot (Herramental), con su eje Z paralelo a Z_{n-1} y X y Y en cualquier dirección válida.
10. **Ángulos teta**: Cada θi es el ángulo desde X_{i-1} hasta X_i girando alrededor de Z_i.
11. **Distancias d**: Cada *d*i es la distancia desde el sistema $\{S_{i-1}\}$ hasta la intersección de las normales común de Z_{i-1} hacia Z_i, a lo largo de Z_{i-1}.
12. **Distancias a**: Cada *a*i es medida a lo largo del eje x_i para que el origen del eje z_{i-1} coincida con el origen del eje z_i.
13. **Ángulos alfa**: Ángulo que hay que rotar a Z_{i-1} para alinearse con Z_i, rotando alrededor de X_i.

Esta metodología permite obtener el modelo cinemático directo donde cada articulación estará relacionada al movimiento angular o de desplazamiento de cada uno de los eslabones del brazo robótico, para asi obtener una representación de su posición y orientación con el sistema coordenado del brazo robótico.

Definir la matriz D-H

Para realizar el objetivo del proyecto es necesario determinar el algoritmo cinemático directo por medio de la matriz D-H. Para aplicar el método se usa el procedimiento Denavit-



Hartenberg que se mencionó anterior mente donde en las siguientes figuras se pude observar la morfología del robot, sus medidas y cada una de sus articulaciones.

FIGURA 4.15: Robot Puma TQMA 2000 vista lateral.

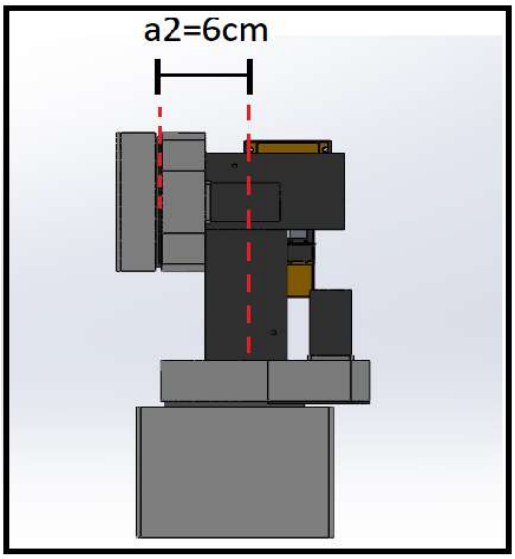


FIGURA 4.16: Robot Puma TQMA 2000 vista Frontal.

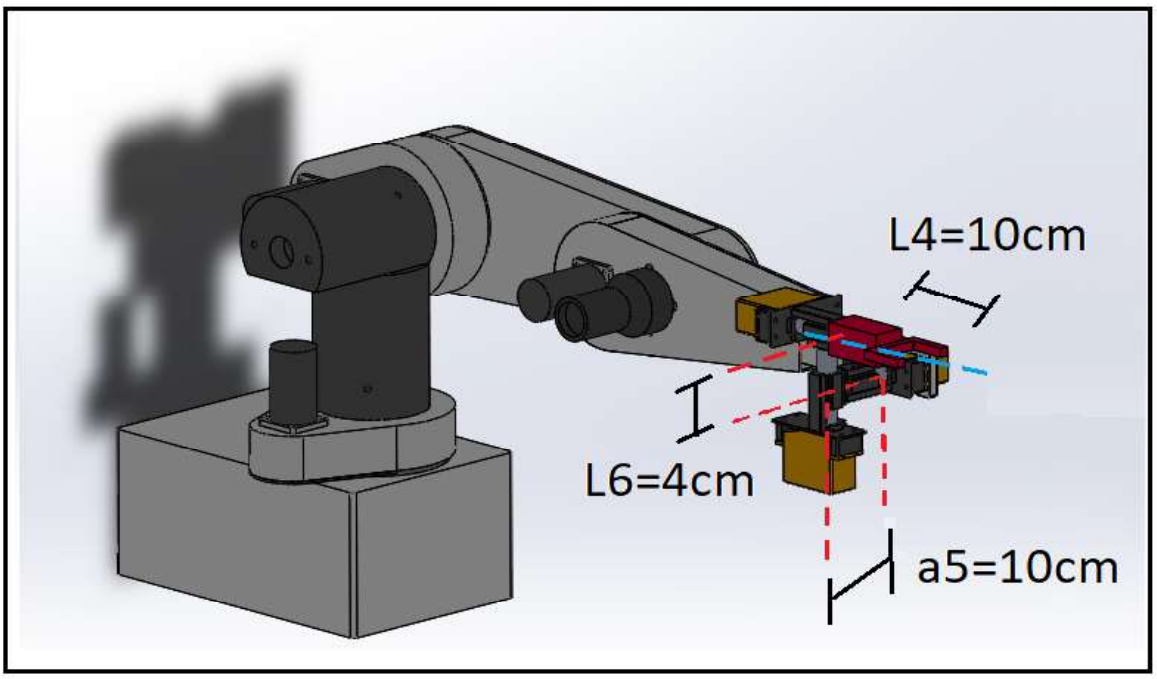


FIGURA 4.17: Robot Puma TQMA 2000 vista isométrica.

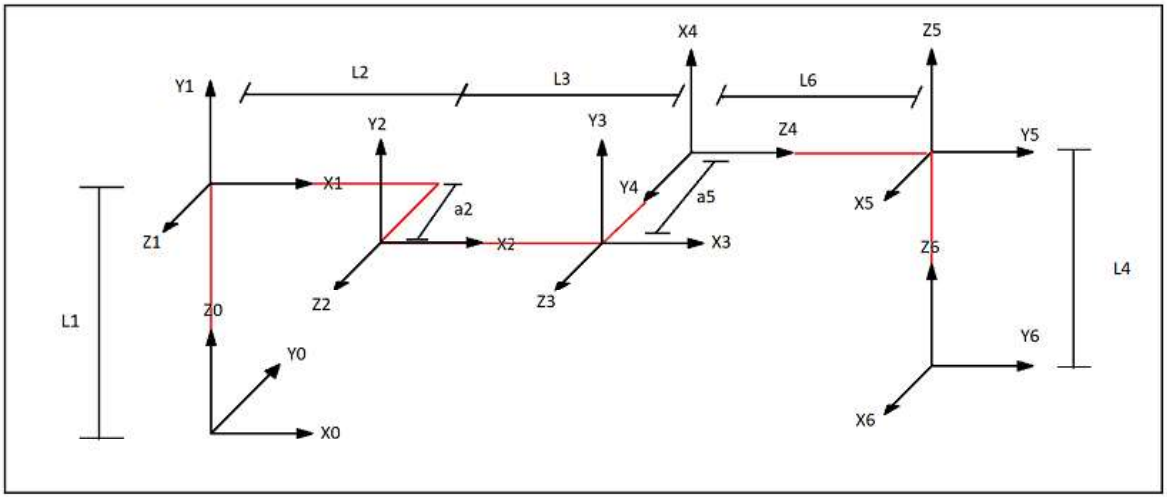


FIGURA 4.18: Representación de ejes coordenados del Robot Puma TQMA 2000.

Donde L1= 26 cm, a2= 6 cm, L2=23 cm, L3=24 cm, a5=10 cm, L4=10cm, L6=4 cm.

En conjunto con la representación de los ejes coordenados y la metodología D-H se puede obtener el análisis de modelo cinemático directo, donde se toman en cuenta

eslabones, articulaciones y distancias. Esto con el fin de obtenerla matriz DH que es la cual resume los datos del modelo cinemático de la morfología del brazo robótico tal cual como se muestra en la Tabla 4.1.

Tabla 4.1: Resultados de la matriz DH del brazo robótico puma TQMA2000

i	θ_i	d_i	a_i	α_i
1	Θ_1	L1	0	90^0
2	Θ_2	a2	L2	0
3	Θ_3	0	L3	0
4	$\Theta_4 + 90°$	-a5	0	90^0
5	$\Theta_5 + 90°$	L6	0	90^0
6	Θ_6	-L4	0	0

Una vez obtenido los parámetros DH como se muestra en la Tabla 4.1 se procede a calcula las matrices que determinaran la posición y orientación del brazo robótico resolviendo el modelo cinemático directo, donde la matriz homogénea T nos indica la posición del efector final $\{S_6\}$ del brazo robótico con respecto al eje coordenado $\{S_0\}$ donde nos indica que la posición de *home* del robot será en X = 51 cm, Y= 4 cm y Z=16 cm.

$$A01 = \begin{bmatrix} C\theta1 & -C\alpha1S\theta1 & S\alpha1S\theta1 & a1C\theta1 \\ S\theta1 & C\alpha1C\theta1 & -S\alpha1C\theta1 & a1S\theta1 \\ 0 & S\alpha1 & C\alpha1 & d1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & L1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A12 = \begin{bmatrix} C\theta2 & -C\alpha2S\,\theta2 & S\alpha2\,S\theta2 & a2C\,\theta2 \\ S\theta2 & C\alpha2C\theta2 & -S\alpha2C\theta2 & a2S\theta2 \\ 0 & S\alpha2 & C\alpha1 & d1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & L2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & a2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A23 = \begin{bmatrix} C\theta3 & -C\alpha3S\,\theta3 & S\alpha3\,S\theta3 & a3C\theta3 \\ S\theta3 & C\alpha3C\theta3 & -S\alpha3C\theta3 & a3S\theta3 \\ 0 & S\alpha3 & C\alpha3 & d3 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & L3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A34 = \begin{bmatrix} C\theta4 & -C\alpha4S\theta4 & S\alpha4S\theta4 & a4C\theta4 \\ S\theta4 & C\alpha4C\theta4 & -S\alpha4C\theta4 & a4S\theta4 \\ 0 & S\alpha4 & C\alpha4 & d4 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -a5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A45 = \begin{bmatrix} C\theta5 & -C\alpha5S\,\theta5 & S\alpha5\,S\theta5 & a5C\,\theta5 \\ S\theta5 & C\alpha5C\,\theta5 & -S\alpha5C\,\theta5 & a5S\theta5 \\ 0 & S\alpha5 & C\alpha5 & d5 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & L6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A56 = \begin{bmatrix} C\theta6 & -C\alpha6S\,\theta6 & S\alpha6\,S\theta6 & a6C\,\theta6 \\ S\theta4 & C\alpha6C\,\theta6 & -S\alpha6C\,\theta6 & a6S\theta6 \\ 0 & S\alpha6 & C\alpha6 & d6 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & L4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T = A01 * A12 * A23 * A34 * A45 * A56 = \begin{bmatrix} 0 & 1 & 0 & L2 + L3 + L6 \\ -1 & 0 & 0 & a5 - a2 \\ 0 & 0 & 1 & L1 - L4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.1)$$

$$T = \begin{bmatrix} 0 & 1 & 0 & 51 \\ -1 & 0 & 0 & 4 \\ 0 & 0 & 1 & 16 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.2)$$

Una vez obtenida las matrices que caracterizan la posición y orientación de los sistemas de coordenadas de cada una de las articulaciones del robot y haber resuelto el problema cinemático directo del brazo robótico se procede a elaborar una simulación en Matlab para comprobar el comportamiento del modelo obtenido, para asi tener un marco de referencia en comparación al ambiente de simulación de *Unity* obteniendo los resultados mostrados en las figuras 4.19, figuras 4.20 y figuras 4.21.

La demostración matemática obtenida con la matriz DH y Matlab fue validada con la simulación dentro de *Unity 3D* y obteniendo resultados similares en tolerancias de +/- 0.001 además de una excelente fluidez en el entorno de simulación obteniendo entre 70 a 220 FPS

sin ningún problema de retraso cumpliendo con los requerimientos que el proyecto demanda.

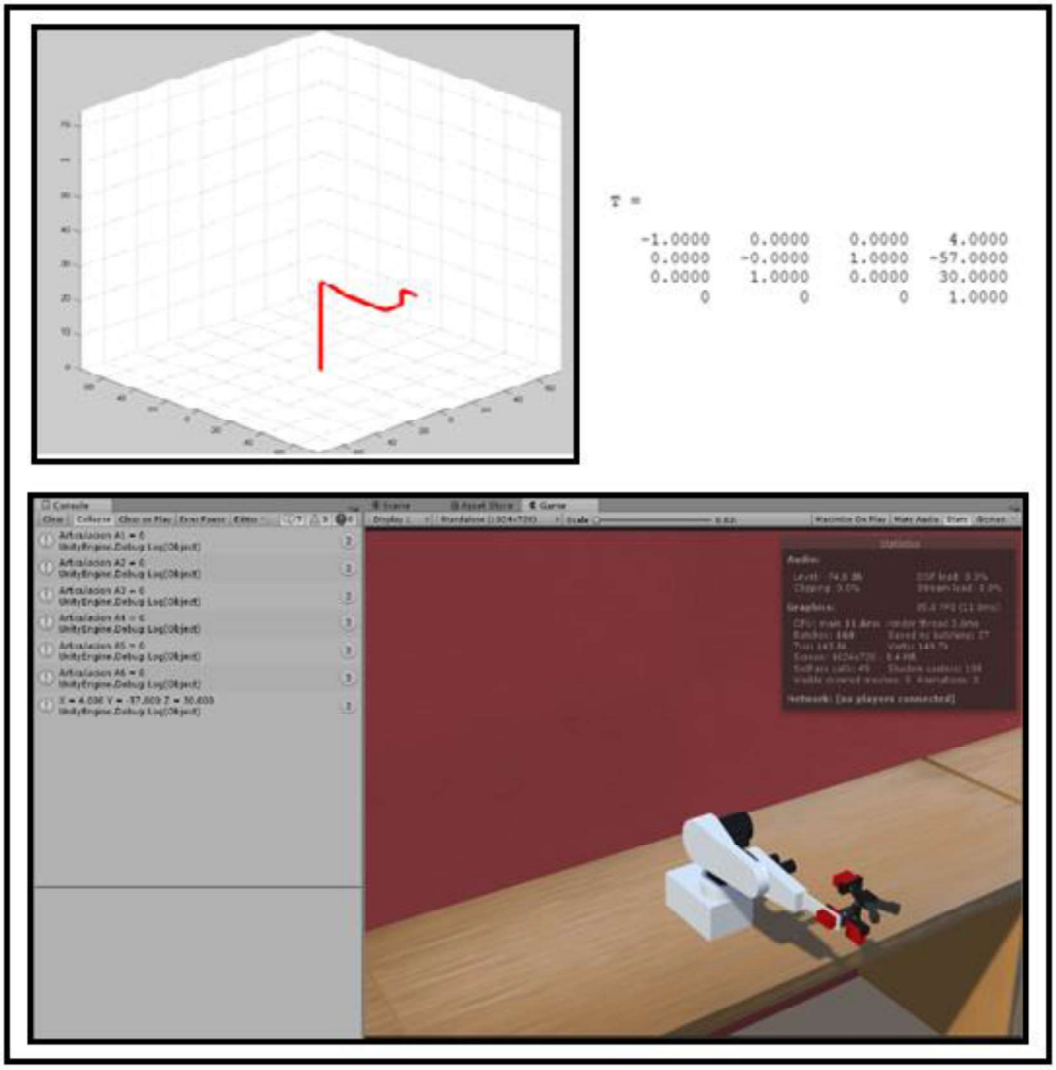


FIGURA 4.19: Simulación y comparación del robot Brazo robótico posición *home*.

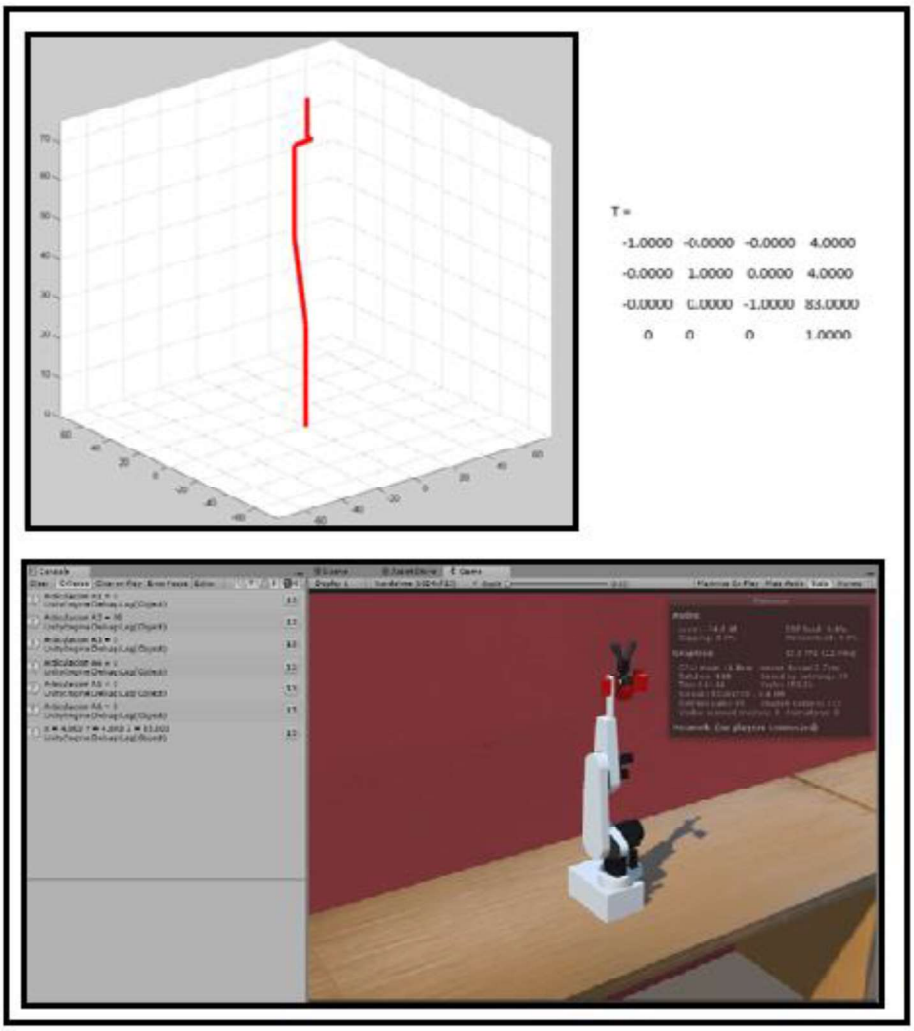


FIGURA 4.20: Simulación y comparación del Brazo robótico (Rotación en: $q_2 = 90°$).

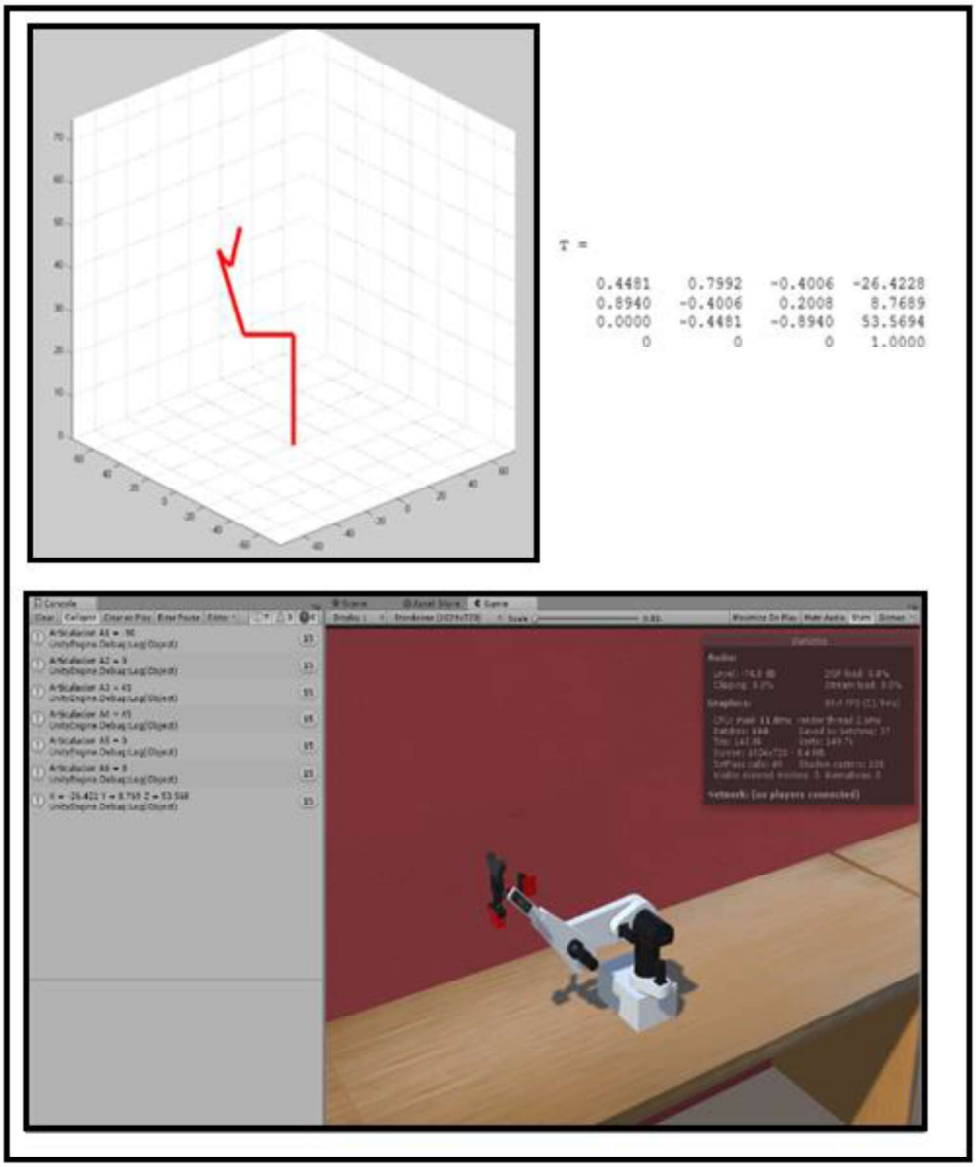


FIGURA 4.21: Simulación y comparación del Brazo robótico (Rotación en: q_1 = -90 °, q_3 = 45 ° y q_4 = 45).

CAPÍTULO 5

Sistemas de control

En este capítulo se enfocara en los sistemas y sub-sistemas que son necesarios para el control del brazo robótico Puma TQ2000 para el cual, como se muestra en la figura 5.1 se debe de tratar cada una de las articulaciones del robot como un sistema independiente, los cuales son coordinadas por una unidad de control principal, cuya función es procesar y distribuir las señales de control proporcionadas por la interface de usuario para así desarrollar los movimientos del brazo articulado de forma sincronizada y en forma armónica al realizar el posicionamiento y orientación de cada eslabón en el robot.

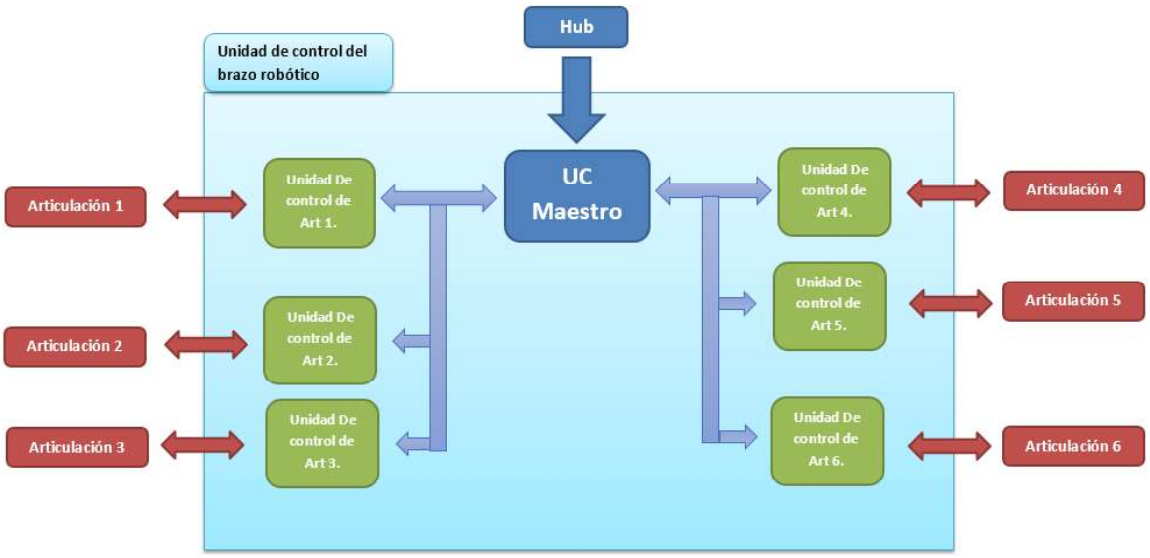
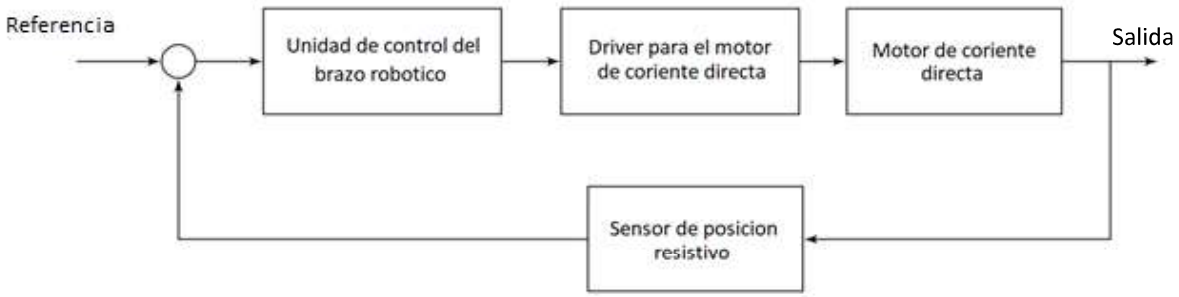


FIGURA 5.1: Diagrama a bloques del funcionamiento de la U.C. del brazo robótico.

Dentro del diagrama de la figura 5.1 se puede apreciar el apartado de la unidad de control para las articulaciones de la 1 a la 6, donde para fines prácticos se puede representar de la misma manera para las 6 articulaciones ya que cuentan con los mismos elementos que son:

- Motor de corriente directa.
- Trasmisión del motor.

- Unidad de control del motor.
- Driver del motor.



- Sensor de posición resistivo.

FIGURA 5.2: Diagrama de control por articulación del brazo robótico.

Todos estos elementos en cada una de las articulaciones del robot tienen la estructura de un servomotor y trabajan en conjunto de manera sincronizada para desarrollar el posicionamiento de cada uno de los eslabones del brazo articulado. Un servomotor tiene como función principal proporcionar un movimiento preciso en el posicionamiento de la articulación, en donde para fines orientados a las necesidades del proyecto cumple con los requisitos necesarios para ser implementado en este.

5.1 Elementos del sistema de control

5.1.1 Motor de corriente directa Maxon 2332.968-51.236-200 24VDC

Este motor se encuentra en cada una de las articulaciones del brazo robótico el cual, según las especificaciones del fabricante como se muestra en la figura 5.3.

Dato	Letra	Valor	Unidad
Tensión nominal	Va	24	V
Velocidad	ω	8410	RPM
Par nominal	Tm	29.1	mNm
Resistencia entre terminales	Ra	7.94	Ω
Inductancia en terminales	La	0.821	mH
Constante de par	Kt	39.3	mNm/A
Constante de velocidad	Ke	243	rpm/V
Momento de inercia del rotor	J	0.00000194	gcm^2
Fricción Viscosa	B	9.558E-07	$Nmsrad^{-2}$

FIGURA 5.3: Parámetros del motor Maxon según especificaciones de fabricante.

5.1.2 Driver del motor L298N

El circuito integrado L298N es un doble controlador de puente H completo fabricado por IC STMicroelectronics, el cual, tiene como función proporcionar alimentación al motor de cd, así como la dirección de giro.

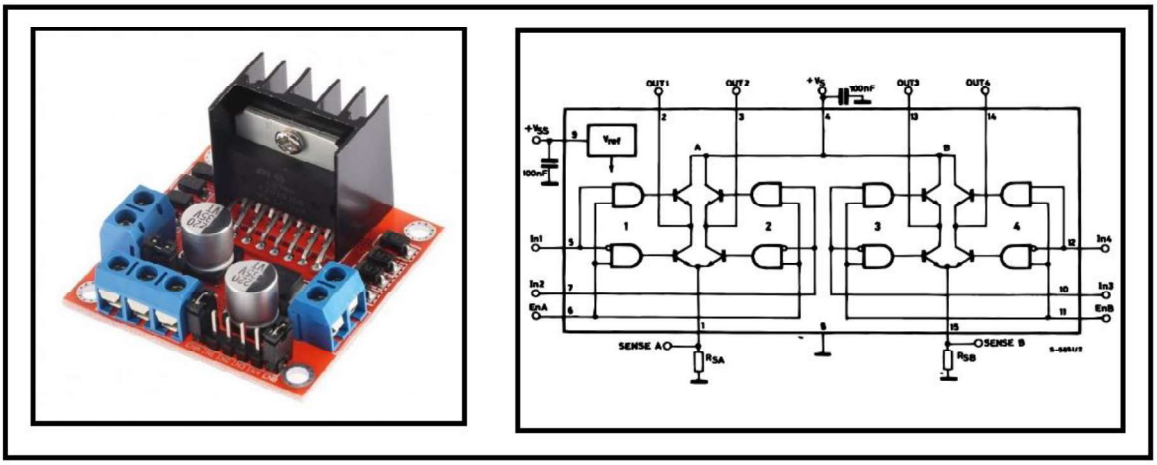


FIGURA 5.4: Tablilla armada con chip L298N y su diagrama interno.

5.1.3 Unidad de control del motor.

Para desarrollar la unidad de control se optó una arquitectura Maestro/esclavo así como se mostró en la figura 5.1 donde el circuito integrado ATmega2560 fue seleccionado como el maestro cumpliendo la función de gestionar y distribuir los datos de la HMI y los

circuitos esclavos de la unidad de control siendo el ATmega328 seleccionado, uno por cada articulación de del brazo robótico cumpliendo la función de posicionar las articulaciones según la información que estos reciban del maestro.

Tabla 5.1: Características de los circuitos integrados ATmega2560 y ATmega328

Device	Flash	EEPROM	RAM	General Purpose I/O pins	16 bits resolution PWM channels	Serial USARTs	ADC Channels
ATmega328	32K Bytes	1K Bytes	2K Bytes	12	6	2	8
ATmega2560	256KB	4KB	8KB	86	12	4	16

5.1.4 Sensor de posición resistivo

Los sensores de posición o de desplazamiento resistivo son un elemento electromecánico que posee un conductor eléctrico conectado a una resistencia sobre el cual desliza, está en nuestro caso establece una resistencia de acuerdo al Angulo en la cual este posicionada, donde este dato representa la posición de la articulación del brazo robótico.

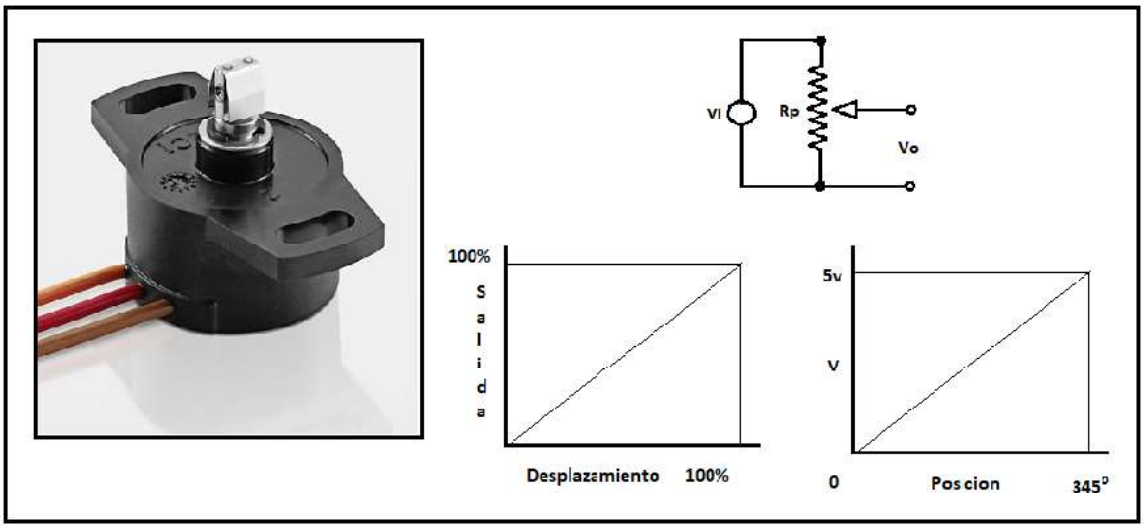


FIGURA 5.5: Sensor de posición resistivo, su diagrama de conexión y su relación voltaje/posición.

5.1.5 Trasmisión del Motor

En el caso particular del brazo robótico PUMA TQ2000 cuenta en cada una de sus primeras 3 articulaciones un motor de corriente directa Maxon con su reductor acoplado

directamente a su flecha, los reductores consisten en ser un sistema mecánico conformado a base de engranes, mecanismos circulares o serrados con geometrías diferentes, según su tamaño y la función con el propósito de modificar el par y la velocidad de salida de un motor. En nuestro caso las articulaciones del brazo robótico cuentan con 3 relaciones de salida diferente, siendo de 25:1, 15:1 y de 10:1. Además de los reductores cada articulación cuenta con una trasmisión de banda dentada independiente por articulación siendo para la articulación 1 y 2 una relación de conversión de 25:1 y para la articulación 3 de 15:1, esto con la finalidad de que cada elemento de transferencia de fuerza mecánica se capaz de mover cada articulación de manera precisa.

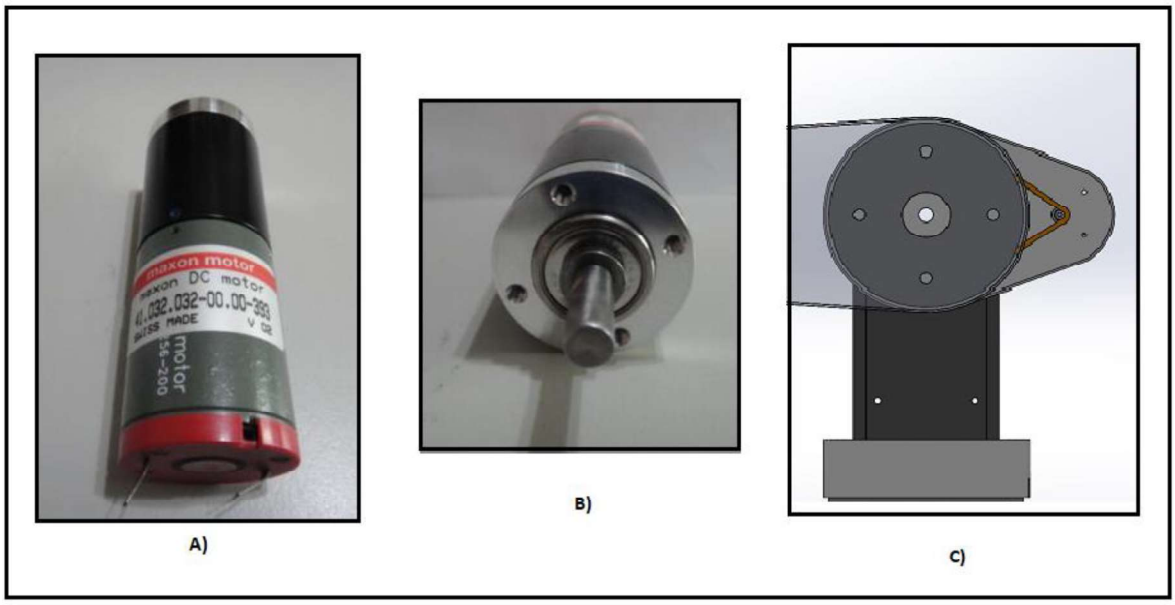


FIGURA 5.6: Motor de corriente directa Maxon con reductor, B) Reductor C) Trasmisión de banda dentada de la segunda articulación del brazo robótico

Como se ve en la figura 5.2 es necesario que tanto como los sistemas electrónicos y los sistemas mecánicos trabajen de manera conjunta para alcanzar el objetivo de accionar de manera coordinada el movimiento del brazo robótico articulado, para esto es necesario obtener el modelo matemático del motor de corriente directa Maxon 2332.968-51.236-200

24VDC basándose de los parámetros mostrados en la figura 5.3 que son entregados según fabricante. Dicho modelo matemático se obtiene de forma clásica atreves del circuito equivalente

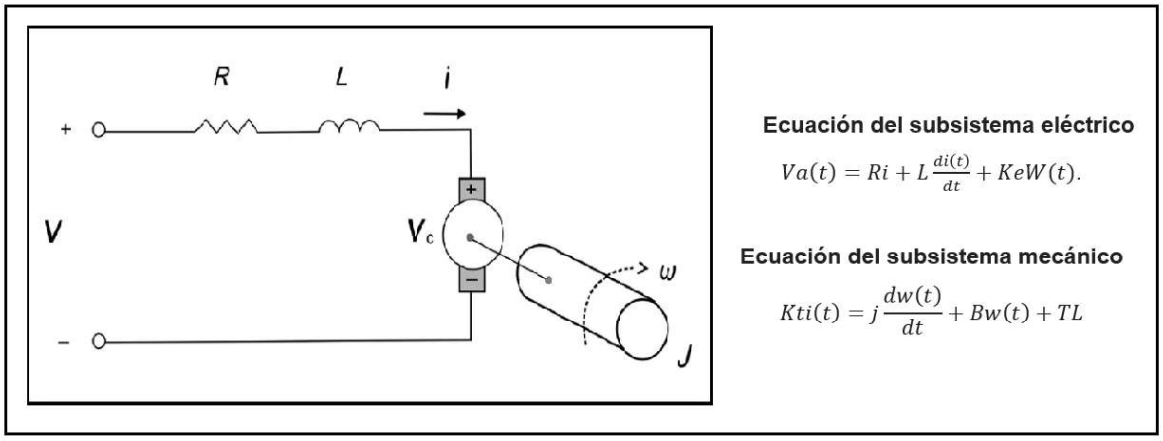


FIGURA 5.7: Modelo eléctrico del motor MAXON y modelo matemático.

Tomando en consideración los datos entregados por el fabricante y con ayuda de la transformada de Laplace se procede a sustituir de las ecuaciones diferenciales dichos valores.

$$I_a(s) = \frac{J}{K_t}\omega(s)s + \frac{\beta}{K_t}\omega(s) = \left(\frac{Is+\beta}{K_t}\right)\omega(s)$$

$$V_a = R_a\left(\frac{Js+\beta}{K_t}\right)\omega(s) + L_aS\left(\frac{js+\beta}{K_t}\right)\omega(s) + K_e\omega(s)$$

$$V_a = \left(\frac{R_aJs}{K_t} + \frac{R_a\beta}{K_t}\right)\omega(s) + \left(\frac{L_ajs^2}{K_t} + \frac{L_a\beta s}{K_t}\right)\omega(s) + K_e\omega(s)$$

$$\frac{V_a}{\omega(s)} = \left(\frac{R_a J s}{K_t} + \frac{R_a \beta}{K_t}\right) + \left(\frac{L_a j s^2}{K_t} + \frac{L_a \beta s}{K_t}\right) + (K_e)$$

$$\frac{V_a}{\omega(s)} = \left(\frac{R_a J s}{K_t} + \frac{L\beta}{K_t}\right) + \left(\frac{L_a j s^2}{K_t}\right) + \frac{(R_a\beta + K_e K_t)}{K_t}$$

$$\frac{V_a}{\omega(s)} = \frac{1}{\left(\frac{L_a j s^2}{K_t}\right) + \left(\frac{R_a J s}{K_t} + \frac{L\beta}{K_t}\right) + \frac{(R_a\beta + K_e K_t)}{K_t}}$$

$$\frac{V_a}{\omega(s)} = \frac{\frac{K_t}{L_a J}}{S^2 + \left(\frac{R_a}{L_a} + \frac{\beta}{j}\right)s + \frac{(R_a\beta + K_e K_t)}{L_a J}}$$

$$\frac{V_a}{\omega(s)} = \frac{\frac{39.3}{0.821 * 0.00000194}}{S^2 + \left(\frac{7.94}{0.821} + \frac{9.558E-07}{0.00000194}\right)s + \frac{(7.94 * 9.558E-07 + 39.3 * 243\ )}{0.821 * 0.00000194}} \quad (5.1)$$

$$\frac{V_a}{\omega(s)} = \frac{13059256.37}{S^2 + 6748.36\,S + 405557.2912} \quad (5.2)$$

5.2 Control PID

Una vez que se obtiene la función de transferencia del motor, en la cual, el denominador representa la ecuación característica del motor, se procede a realizar los cálculos de controlador PID, el cual es un esquema de control clásico utilizado universalmente en sistemas de control de lazo cerrado. Este lazo cerrado se obtiene por medio de la retroalimentación de la variable de salida a controlar y en comparación la señal de referencia se calcula el error o la desviación entre el valor medido y el valor deseado. Para lograr operar adecuadamente el control PID se deben de seleccionar cuidadosamente tres parámetros del control: la ganancia proporcional (*Kp*) que depende del error actual, la ganancia integral (*ki*) que depende del error pasado y la ganancia derivativa (*kd*) que

depende de una predicción del valor futuro, de tal manera que es posible desarrollar un algoritmo con el valor de estas tres variables para poder realizar una acción de control ajustándose a los requerimientos de un proceso especifico reduciendo el error con respecto al valor de entrada.

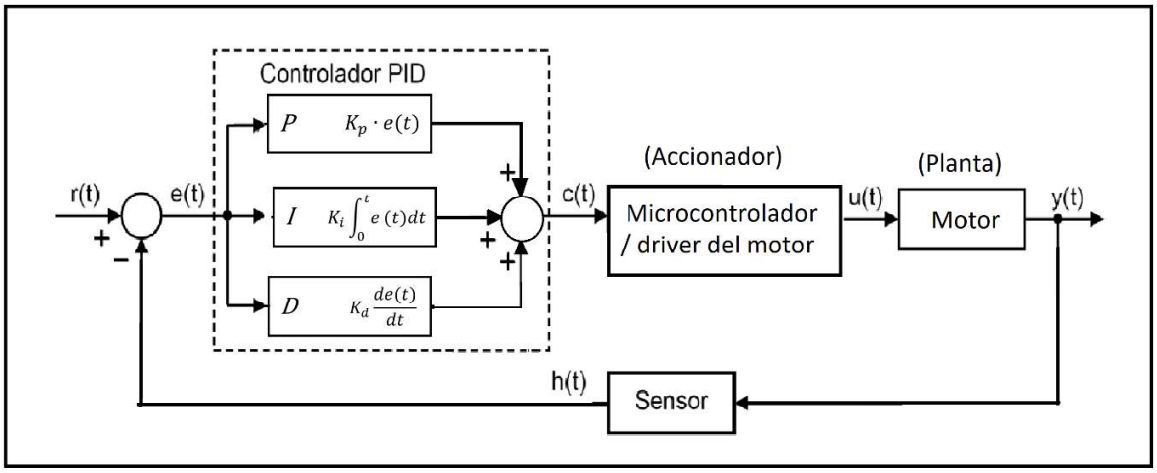


FIGURA 5.8 Diagrama de bloques de un sistema PID.

En este caso, el elemento a controlar es el motor de cada articulación del brazo robótico por medio de un microcontrolador que generara la señal de control que será enviada al convertidor electrónico de potencia del motor, este conjunto de elementos ayudaran a ajustar la velocidad y el torque del motor y con ayuda del valor de referencia y del sensor podremos posicionar la articulación del brazo robótico en la posición del valor de referencia.

El método utilizado para calcular los valores de *Kp, Ki* y de *Kd* será el método basado en la curva de reacción de Ziegle y Nichols donde se menciona que muchas plantas pueden ser descritas satisfactoriamente con el siguiente modelo:

$$G_0(s) = \frac{K_0 e^{-S\tau 0}}{v_0 S + 1} \quad donde \;\; v_0 > 0 \tag{5.3}$$

Una versión cuantitativamente lineal del modelo se puede obtener mediante un experimento de lazo abierto y siguiendo los siguientes paso:

1. Con la planta a lazo abierto, se lleva a la planta a un punto de operación normal, es decir, que la salida de la planta se estabiliza en *y (t) = y_0* para una entrada constante *u (t) = u_0*.
2. En el instante inicial *t1*, aplicar un cambio en la entrada escalón, desde u_0 a u_1 (esto debería ser en un rango de 10 al 20% de rango completo).
3. Registrar la salida hasta que se estabilice en el nuevo punto de operación. Suponiendo que la curva que se obtiene es la que se muestra en la Figura 5.9 Esta curva se llama curva de reacción del proceso.

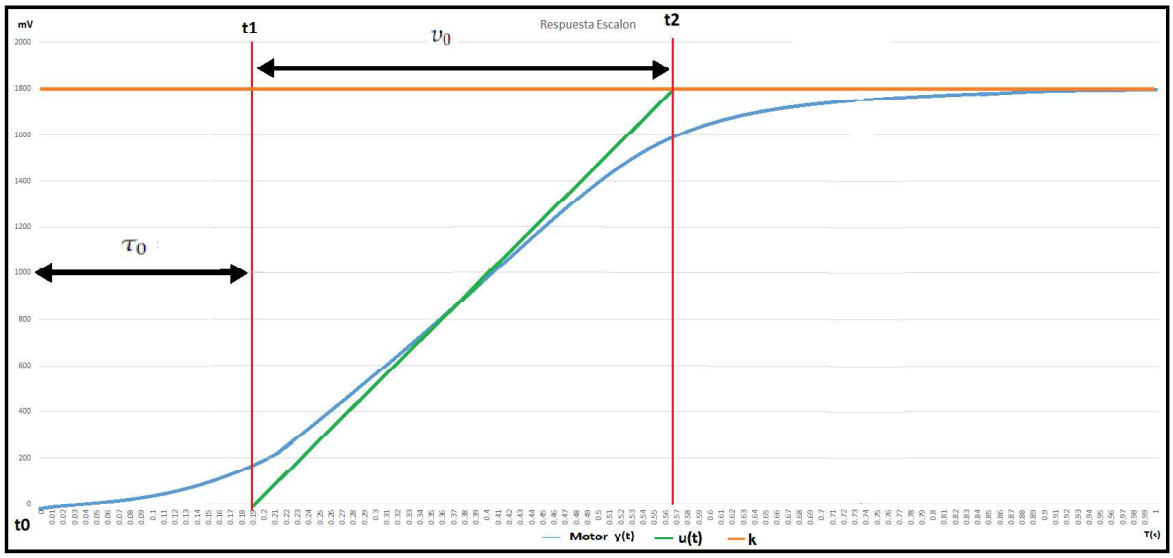


FIGURA 5.9 Respuesta escalón a la planta y aplicación del método Z-N basado en la curva de reacción

Donde una vez que se obtiene la gráfica de la curva de reacción se procede a calcular los parámetros del modelo con las siguientes formulas:

$$K_0 = \frac{y_\infty - y_0}{y_\infty - u_0} \quad ; \quad \tau_0 = t_1 - t_0 \quad ; \quad v_0 = t_2 - t_1 \tag{5.4}$$

(5.5)

$$K_0 = \frac{1658.776 - 167.87}{1800 - 0} = 0.82785$$
$$\tau_0 = 0.1539 - 0 = 0.1539$$
$$v_0 = 0.56869 - 0.1898 = 0.37889 \quad (5.6)$$

El modelo que se obtiene puede ser utilizado para varios métodos de sintonización de controladores PID. Uno de ellos fue propuesto por Ziegle y Nichols y el cual se muestra en la tabla 5.1 con los resultados presentados en la Tabla 5.2

Tabla 5.2: Parámetros de ajuste para el método de curva de reacción

	K_p	T_i	T_d
PI	$\frac{v_0}{K_0\tau_0}$	--	--
PI	$\frac{0.9\,v_0}{K_0\tau_0}$	$3\tau_0$	--
PID	$\frac{1.2\,v_0}{K_0\tau_0}$	$2\tau_0$	$0.5\,\tau_0$

Tabla 5.3: Resultados de los parámetros de ajuste.

	K_p	T_i	T_d
PI	2.8698	--	--
PI	2.582888	0.4617	--
PID	3.4438	0.3078	0.07695

5.3 Control de posición

Una vez obtenidos los valores K_p, T_i y T_d tal como se muestra en la tabla 6.2 se desarrolló el algoritmo de control de posición de cada una de las articulaciones del brazo robótico, para lo cual se utiliza el circuito integrado ATmega328 exclusivamente para realizar los cálculos de posición, velocidad y torque necesario para funcionar de manera correcta, ya que la planta estará constantemente trabajando con perturbaciones externas como lo son las cargas externas y la carga de trabajo.

Para realizar la programación del ATmega328 se opta por utilizar el entorno de desarrollo integrado (*Integrated Development Environment* - IDE) proporcionado por la marca Arduino, este es una aplicación multiplataforma que está basada para trabajar en Java, este permite programar el ATmega328 y ATmega2560 utilizando el lenguaje C y C++ utilizando reglas especiales de estructura de código.

Para el desarrollo del programa de control como se menciona anteriormente, se utiliza un ATmega328 por cada una de las articulaciones, esto por su facilidad de uso, bajo costo y buena respuesta, características requeridas para para las necesidades del proyecto.

El diseño del programa principalmente se puede dividir en tres funciones las cuales son:

- **Función PID:** que se encarga de realizar los cálculos de los valores K_p, T_i y T_d para lograr que la variable de salida siga la trayectoria definida por la señal de referencia.
- **Función Dirección:** Esta se encarga de definir el sentido de giro del motor según la señal de retroalimentación comparándolo con la señal de referencia.
- **Función de Ciclo**: Es la encargada de realizar un bucle infinito donde constantemente estará leyendo la señal de referencia y realiza un llamado a las funciones anteriormente mencionadas para generar el movimiento correcto de cada articulación.

```
// FUNCION ENCARGADA DE DETERMINAR EL GIRO DEL MOTOR

void Direccion()
{
  //  Serial.println("Direccion");
  if (w >= .1 ) //Saturacion de la accion de control u en un tope maximo y minimo
    {

    analogWrite(IN2, Speed); // salida del PWM
    analogWrite(IN1, LOW);

    }

   if (w <=  -.1 ) //Saturacion de la accion de control u en un tope maximo y minimo
    {

    analogWrite(IN1, Speed); // salida del PWM
    analogWrite(IN2, LOW);

    }
}
```

FIGURA 5.10 Código de la función dirección para determinar el giro del motor.

```
bool PID::Calculo()
{
   unsigned long now = millis();
   unsigned long timeChange = (now - lastTime);
   if(timeChange>=SampleTime)
   {
      /* Calcular todas las variables de error de trabajo */
      double input = *myInput;
      double error = *mySetpoint - input;
      double dInput = (input - lastInput);
      outputSum+= (ki * error);

      /*Agregar proporcional en la medición*/
      if(!pOnE) outputSum-= kp * dInput;

      if(outputSum > outMax) outputSum= outMax;
          else if(outputSum < outMin) outputSum= outMin;
          double output;
          if(pOnE) output = kp * error;
                else output = 0;

      /*  Calcular resto de salida PID */
      output += outputSum - kd * dInput;

        if(output > outMax) output = outMax;
              else if(output < outMin) output = outMin;
              *myOutput = output;

       /* Recordar algunas variables para la próxima vez */
      lastInput = input;
      lastTime = now;
        return true;
   }
   else return false;
}
```

FIGURA 5.11 Función de cálculos del PID para cada articulación de brazo robótico.

```
// FUNCION ENCARGADA DE REALIZAR Y REFRESZAR LOS CALCULOS DEL PID
void PIDLoop()
{
  w1 = abs(w); // set input
  // calcular PID DEL MOTOR
  PIDMOTOR.Compute();
  Speed = abs(Output);
}
```

FIGURA 5.12 Función PIDLoop necesaria para el correcto funcionamiento de la función PID C.

```
void loop()
{
 // si se envia un valor por el puerto serial esta funcion procesa el dato y cambia el set point
  if (Serial.available() > 0)
    {
    // lee el byte de entrada:
    Setpoint = Serial.parseInt();
    Setpoint = Setpoint * ConGrado;
    Setpoint = (int)Setpoint;
    contador = Serial.read();
    Serial.print("valor de Setpoint a cambiado = ");
    Serial.println(Setpoint);
    }

    x = analogRead(PIN_INPUT);
    w = (x - Setpoint); //Posicion o error

PIDLoop(); // entra ciclo del PID  para los calculos
Direccion();
}
```

FIGURA 5.12 Función Bucle encargada donde se estable los nuevos parámetros de la señal de referencia y se ejecutan las funciones PIDLoop y Dirección.

Como se puede notar en las Figuras 5.10, 5.11 y 5.12 las funciones de PID, Dirección y Ciclo son las encargadas de hacer el control de posición de cada una de las articulaciones del brazo robótico.

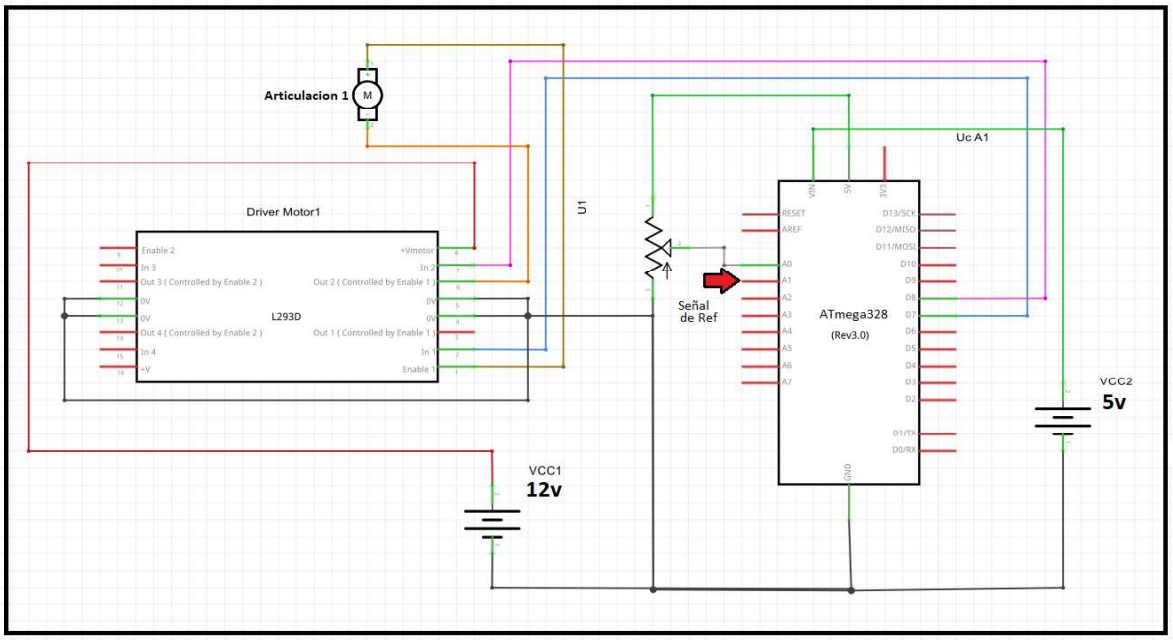


FIGURA 5.13: Diagrama de conexiones de la unidad de control maestro, Uc esclavo, Driver puerto H y Articulación 1.

Como se muestra en la figura 5.13 el diagrama de conexiones eléctricas presenta cómo interactúan cada uno de los componentes anteriormente mencionados para realizar el movimiento de cada una de las articulaciones al igual que los cálculos de velocidad y la posición que tendrá cada articulación dependiendo de la señal de referencia, este circuito se encarga únicamente del control de un motor por cada articulación y por lo tanto, se requieren de seis circuitos iguales para lograr el funcionamiento completo del brazo robótico.

5.4 Comunicación

Para el funcionamiento correcto del proyecto se investiga que protocolos de comunicación maneja *Unity* puesto que este es el programa donde se diseña la *HMI* del brazo robótico, ya que *Unity* usa una variante de C# para ser programado se pueden utilizar las librerías ya existentes, además de utilizar diferentes protocolos de comunicación como lo son:

- TCP(*Transport Control Protocol / Internet Protocol*)
- UDP (*User Datagram Protocol*)
- Modbus
- LAN (*Local Area Network*) (*Ethernet*)
- Puerto serie (*Serial port*)

De los cuales se elige el puerto serie para establecer la comunicación. Este, es una interfaz cuya función principal es la comunicación en serie, transmitiendo un bit a la vez. Es también llamado puerto COM (puerto de comunicaciones) o puerto RS-232. Esto se debe a que *Unity* en su documentación explica de manera más detallada la forma en la cual se puede implementar esta comunicación ya que este al ser una plataforma de desarrollo de videojuegos los periféricos como los *mouse*, teclados y *joystick* son comúnmente utilizados bajo este protocolo.

Para esto es necesario utilizar la librería "*System.IO.Ports*" que *Unity* proporciona en su entorno de programación la cual contiene las clases para controlar los puertos serie. La clase más importante *"SerialPort"* proporciona un marco de control de I/O (*Input/output*) dando acceso a las propiedades de objeto a controlar, que en nuestro caso sería el circuito integrado ATmega2560, el cual se encargara de gestionar los datos de la HMI a las unidades de control de cada una de las articulación, como se puede observar en la figura 5.15 las funciones que se encargan de enviar las posiciones al *HUB* (Concentrador).

Por otro lado dentro el programa en el circuito integrado ATmega2560 utiliza la función *"Serial.begin(speed)"* para inicializar la comunicación serial entre el circuito integrado y otro dispositivo que en este caso se trata del ordenador que contiene la *HMI* del brazo robótico y la función "*Serial.available()"* que se encarga de obtener un numero de bytes (caracteres) cuya lectura se encuentra disponible en el puerto serie, estos datos se almacenan en el buffer del puerto serie cuyo memoria es de 64 bytes donde una vez recibida la información esta es gestionado por función "datosResividos()" la cual se encarga de gestionar los datos recibidos y los distribuye a cada uno de los ATmega328 para dar la posición de referencia de cada articulación así como se puede observar en la

```
void setup()
{
  Serial.begin(9600); // se inicializa el puerto serie
  lcd.begin(16, 2);   // se inicializa  la libreria para manipular el display lcd
  lcd.clear();
}
```

```
void loop()
{

  while  (Serial.available() > 0) // verifica si existe algun dato en el buffer
    {
      art1 = Serial.parseInt();
      art2 = Serial.parseInt();
      art3 = Serial.parseInt();
      art4 = Serial.parseInt();
      art5 = Serial.parseInt()
      art6 = Serial.parseInt()
      datosResividos() ;
      break;
    }

}
```

figura 5.14.

FIGURA 5.14: Funciones encargadas de la comunicación serial entre la HUB la HMI.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO.Ports;


public class ComMov : MonoBehaviour {


    public float speed;
    public float movimiento;
    public int x1 =0,x2=0,x3=0,x4=90,c5=90,c6=0; //Variables de las artigulaciones en posicion actual
    public char Name;
    public char NombrePuerto; // Variable  que determina el puerto de com serial a utilizar
    public float Slider1;
    public Transform Cube;
    public string c1,c2,c3,c4,c5,c6;

// se inicializa la comunicacion serial entre la HMI  y el HUB
    SerialPort serialTest = new SerialPort(NombrePuerto , 9600);

    void Start ()
    {
        serialTest.Open (); // se abre comunicacion serial
        Debug.Log ("Inicio");
    }


    public void EnviarDatosHubRobot (int Direccion)
    {
        Debug.Log ("Inicio del boton");// Mensaje que se imprime en consola para indicar que se enviaran los datos
        c1 = x1.ToString ();
        c2 = x2.ToString ();
        c3 = x3.ToString ();
        c4 = x4.ToString ();
        c5 = x6.ToString ();
        c6 = x6.ToString ();
        if (serialTest.IsOpen)
        {
            try
            {
                    serialTest.WriteLine (c1);  //se envia dato de articulacion 1
                    serialTest.WriteLine (c2);  //se envia dato de articulacion 2
                    serialTest.WriteLine (c3);  //se envia dato de articulacion 3
                    serialTest.WriteLine (c4);  //se envia dato de articulacion 4
                    serialTest.WriteLine (c5);  //se envia dato de articulacion 5
                    serialTest.WriteLine (c6);  //se envia dato de articulacion 6
            }

            Debug.Log ("Enviando"); // mensaje que se enviara en consola indicando que se enviaron los datos correctamente

            catch(System.Exception)
            {
                Debug.Log ("No-Enviado"); // mensaje que se enviara en consola indicando que no se enviaron los datos
            }

        }
    }
```

FIGURA 5.15: Funciones encargadas de la comunicación serial entre la HMI y el HUB.

Donde teniendo en cuenta el diagrama de la figura 5.13 donde se muestra que este es el control independiente de cada uno de los motores es necesario mencionar que la unidad maestro de control es la encargada de darle su valor de referencia para que esta pueda ejecutar el algoritmo de control del motor y realizar los cambios de posición, pero como

se muestra en la figura 5.16 se puede observar un diagrama completo de todos los elementos relacionados encargados de dotar de movimiento al brazo robótico.

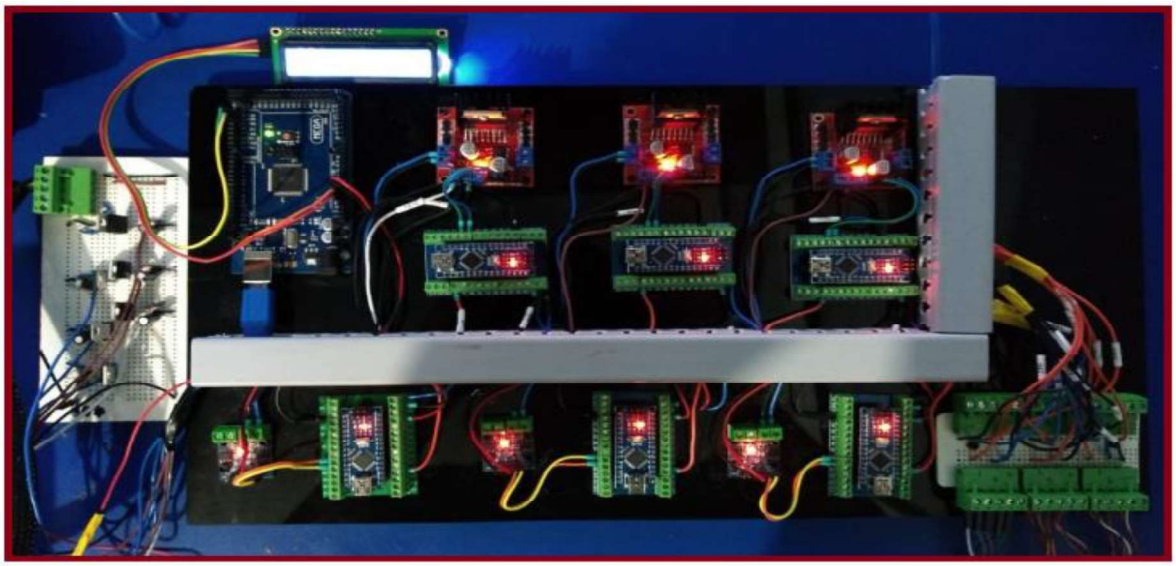


FIGURA 5.16: Conexión y relación de los elementos de control de maestro/esclavo.

Una vez realizadas las conexiones se procedió a validad la interconexión entre la HMI, el sistema de control y el brazo robótico, como se puede observar en las figura 18, la conexión entre la HMI y el sistema de control se logró de manera satisfactoria además de seleccionar la posición de Home del brazo robótico que se muestra en la misma figura. Como se observa en las figura 18, figura19 y figura 20 se pueden observar el correcto movimiento de cada una de las articulaciones, sometiéndolas a 3 posiciones diferentes donde se ponen a prueba el movimientos de la articulación todas las articulaciones del brazo robótico ejecutando correctamente su algoritmo de posición. También se puede observar con más detalle las posiciones que se le dieron al robot dentro de la HMI validando los resultados en Matlab, donde se obtuvo una precisión de +/- 0.001. Obteniendo los resultados deseados en esta investigación.

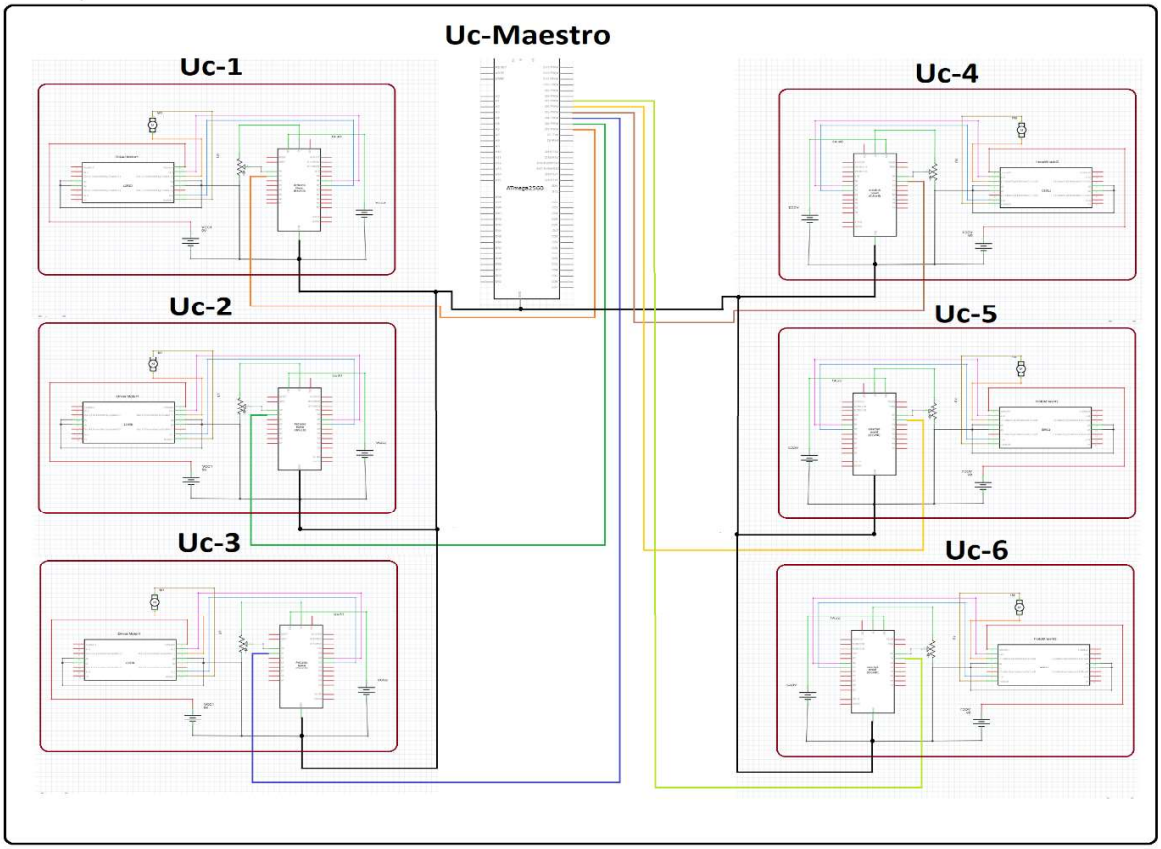


FIGURA 5.17: Sistema de control Maestro/Esclavo del brazo robótico.

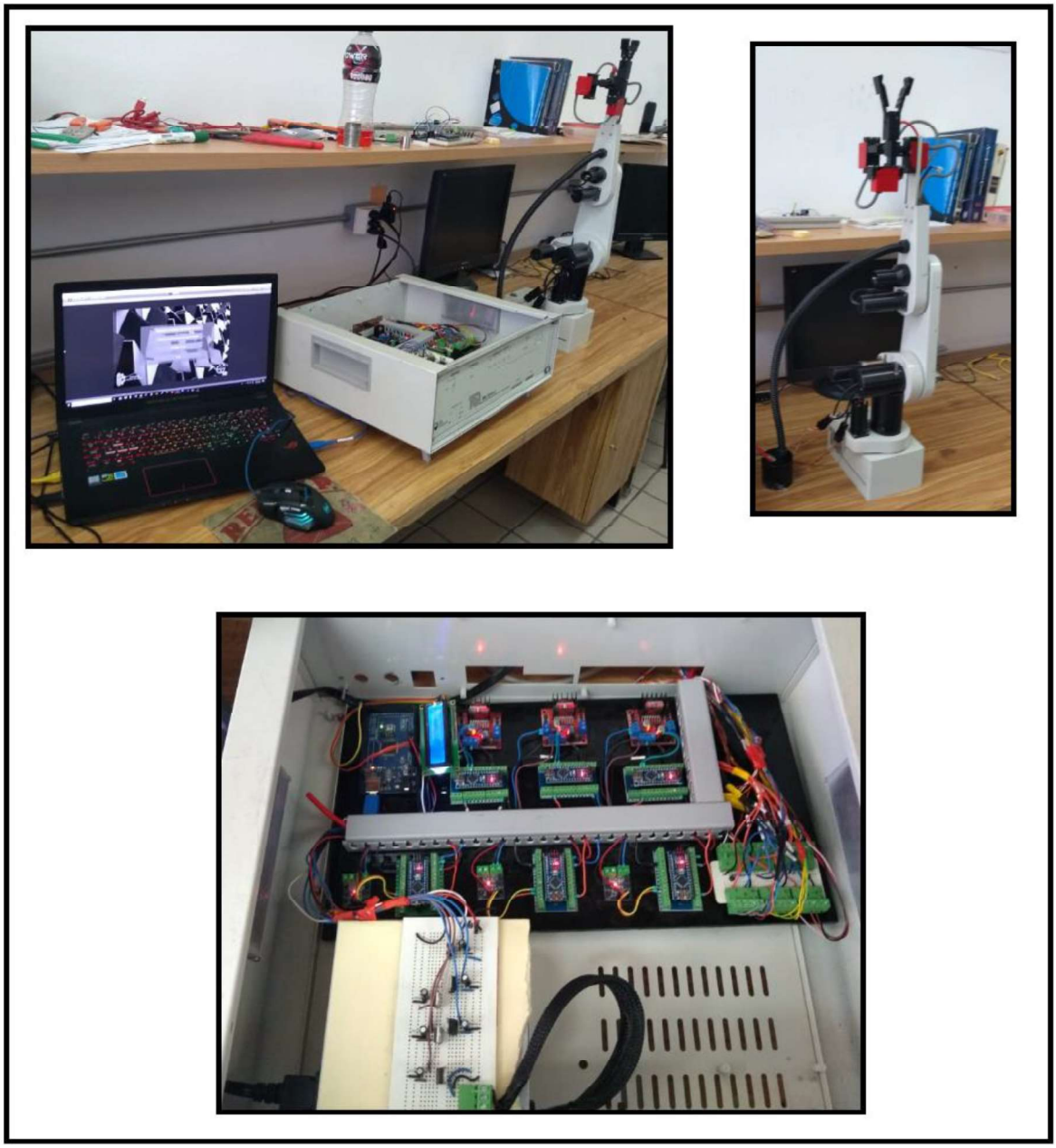


FIGURA 5.18: Resultados de la interconexión de la HMI, sistema de control y del brazo robótico

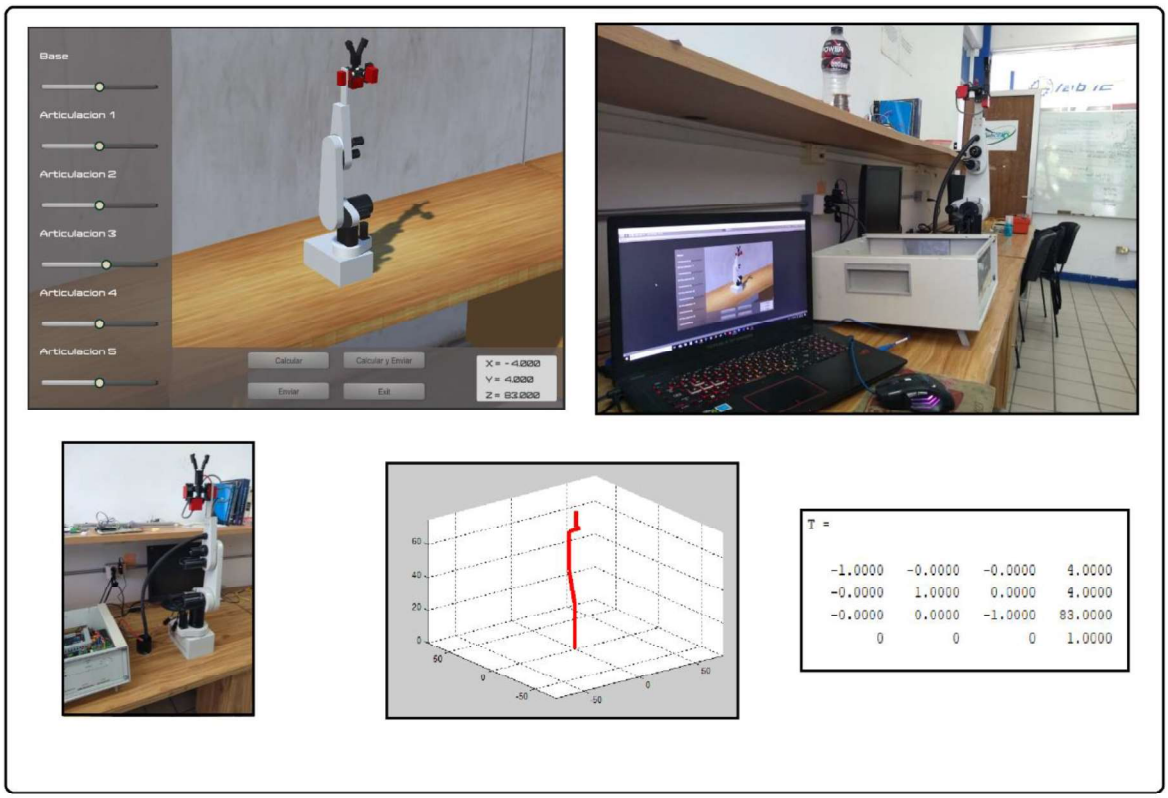


FIGURA 5.19: Prueba de monitoreo 1 del brazo robótico (Rotación en: $q_2 = 90°$).

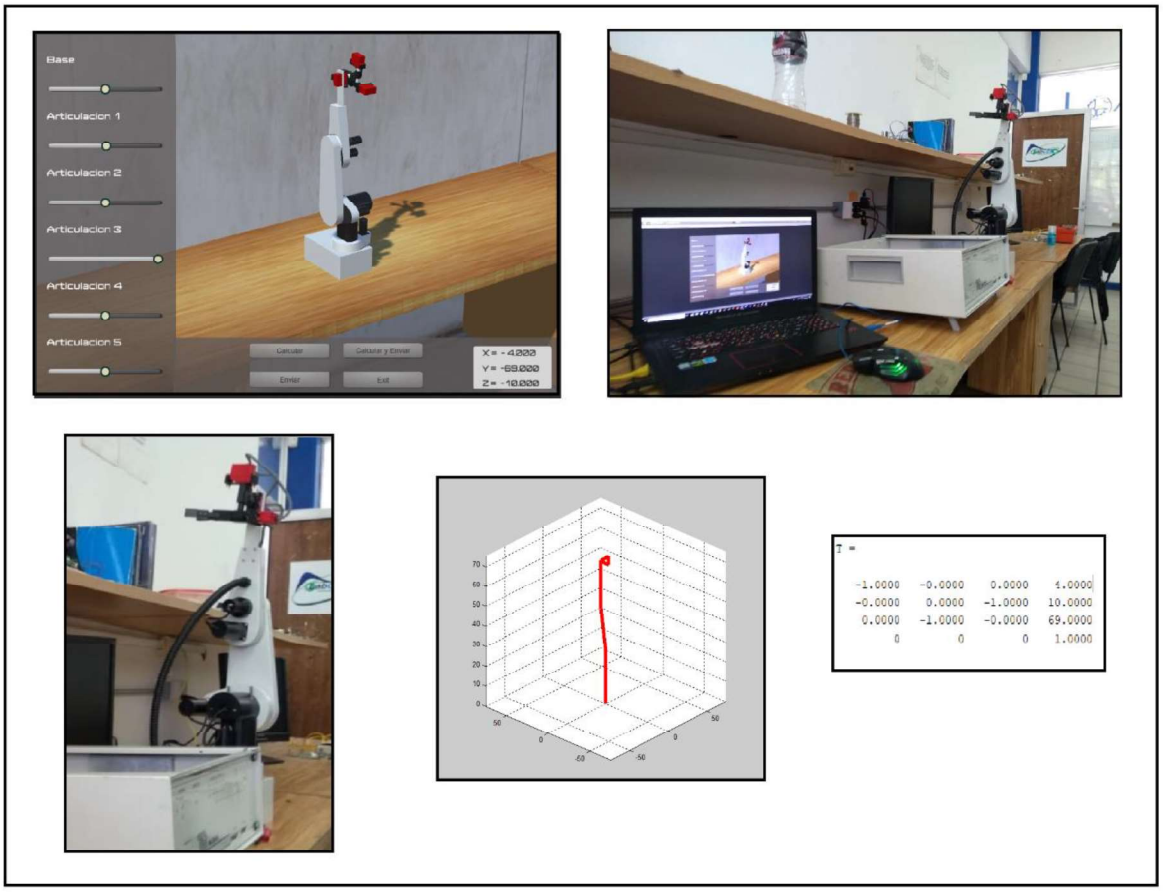


FIGURA 5.20: Prueba de monitoreo 2 del brazo robótico (Rotación en: $q_2 = 90°$, $q_4 = 90°$).

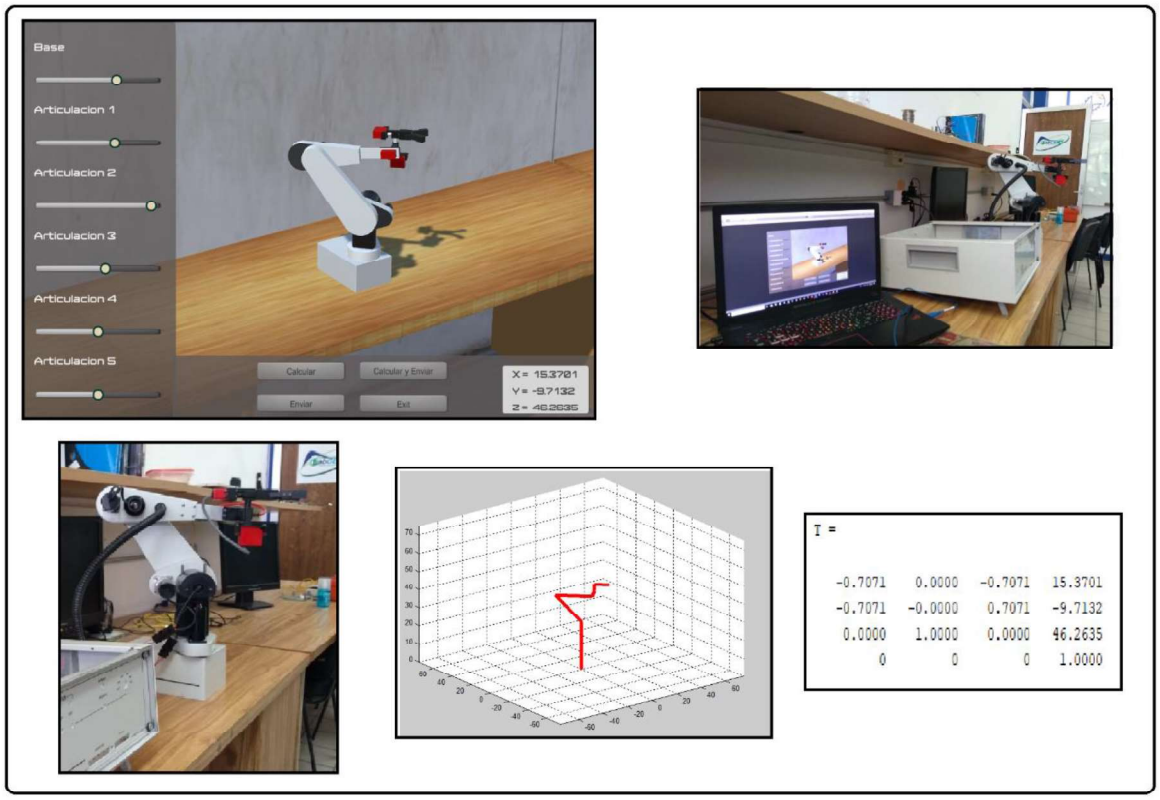


FIGURA 5.21: Prueba de monitoreo 3 del brazo robótico (Rotación en: q_1 = 45°, q_2 = 135°, q_3 = -135°).

CAPÍTULO 6

Conclusiones

En este capítulo se presentara una síntesis de los elementos de los cuales están conformados la HMI y las Unidad de control al igual el cómo interactúan estos sistemas entre sí para lograr el correcto funcionamiento del brazo robótico PUMATQ2000, También se hace mención para posibles continuaciones del trabajo debido a que el los cálculos de la cinemática inversa no se implementaron en esta investigación.

1. Se ha desarrollado un modelo y una interfaz de usuario con elementos visuales. La interfaz interactiva es un simulador para el brazo robótico PUMATQ2000.
2. La interfaz fue diseñada dentro del ambiente de desarrollo proporcionado por *Unity* y contiene funciones de manipulación de elementos 3D que corresponden al brazo robótico.
3. Los elementos 2D dentro de la HMI permiten al usuario acceder a diferentes configuraciones dentro de la interfaces tales son como el acceder al módulo de simulación, realizar cálculos de cinemática directa, configurar el puerto de comunicación y establecer y enviar comandos de control a las unidades de control.
4. Todos los *sprites* de la HMI utilizados en los elementos como los botones, indicadores, recuadros de texto, sliders, imágenes de fondo y las texturas fueron desarrolladas desde 0.
5. Se desarrolló un algoritmo encargado de realizar los cálculos de la cinemática directa del brazo robótico, obteniendo una precisión de -/+.001 comparándolas con el modelo matemático del brazo robótico comparándolo con la simulado desarrollada en el ambiente de Matlab.
6. Se desarrolló una función la cual se encarga de detectar las colisiones entre los elementos del brazo robótico, cambiando de color las partes que chocan entre

ellas al igual que esta misma impide enviar al HUB dicha posición evitando que el brazo robótico real colisione.

7. Se desarrolló una función la cual es la encargada de establecer la comunicación entre la HMI y el HUB bajo el protocolo de comunicaron serial a un nivel de compatibilidad API (*Application Programming Interface)* .NET 2.0 para reducir el tamaño de la aplicación pero aumentando el tiempo de arranque.
8. Se realizó el estudio dela morfología del brazo robótico desarmando todos los elementos mecánicos de este, donde se analizaron todos los elementos del cual estaban conformado y también se tomaron medidas de estos para obtener los planos 2D de cada elemento.
9. Se realizó la virtualización del brazo robótico en el *software Solidworks* obteniendo el modelo 3D de este donde a su vez se realizaron los planos 2D de los elementos principales de este para futuras documentaciones.
10. Se realizó la virtualización del laboratorio donde el brazo robótico se ubica por medio de un scanner 3D, posteriormente se renderizo por medio del *software 3DMAX.*
11. Se logró satisfactoriamente la exportación de los modelos 3D del laboratorio y del brazo robótico a la plataforma de desarrollo de *Unity* al igual que se pudo lograr la simulación de movimiento correspondiente del brazo robótico gracias al uso correcto de jerarquías durante la unión de los eslabones y la función de *"Quaternion"* para los cálculos de giro de cada articulación.
12. Se desarrolló en el circuito integrado ATmega2560 el HUB que se encarga de recibir los valores de posición de cada una de las articulaciones del brazo robótico distribuyendo para cada unidad de control del brazo robótico la referencia al cual deben estar posicionada cada articulación.
13. Por medio del método basado en la curva de reacción de Ziegle y Nichols se obtuvieron los parámetros de *Kp, Ki* y de *Kd* para desarrollar el control PID en el

circuito integrado ATmega328, para controlar la velocidad y posición de cada articulación.

14. Se realizaron pruebas de rotación en sentido horario y anti horario en cada articulación de manera independiente para colaborar la robustez del cada uno de los sistemas obteniendo los resultados de alta repetibilidad y precisión.
15. Se realizaron pruebas de interacción entre la HMI, el HUB y las unidades de control de cada una de las articulaciones para validad que el brazo robótico funcionara de manera correcta, de igual manera se validaron los resultados de dichas pruebas con el algoritmo previamente desarrollado en Matlab, obteniendo resultados aceptables.
16. Se obtuvieron los resultados deseados de cada uno de los elementos que conforman el proyecto brindando un sistema de control robusto de posición y velocidad del robot como una HMI que controla dichos sistemas y además proporciona una simulación 3D de movimientos del brazo robótico obteniendo una excelente fluidez en el entorno de simulación obteniendo entre 70 a 220 FPS.

6.1 Trabajos Futuros

El desarrollo de este trabajo puede servir de base para ejecutar ciertas mejorar de expansión del sisma, como pueden ser:

1. Anadir un módulo que se encargue de realizar los cálculo de cinemática inversa del brazo robótico ya que por limitaciones de tiempo y el alcance de la tesis no se pudo implementar.
2. Anadir un módulo que encargue simular la dinámica del robot, donde se pueda observar las perturbaciones que este puede sufrir dependiendo del tipo de carga al que sea sometido.
3. Se puede mejorar la velocidad de procesamiento de las unidades de control de las articulaciones utilizando un microcontrolador con más capacidad al igual que se puede compactar las conexiones utilizando menos micro controladores.

4. Se puede cambiar el circuito integrado del HUB a uno con mayor capacidad y con mayor robustez al igual que cambiar el protocolo de comunicación que hay entre este y la HMI.
5. Dentro de la HMI se pueden mejorar los paquetes de texturas para mejorar el aspecto visual de entorno 3D, aunque esto implica que aumentara la carga al momento de procesar información volviendo al sistema más lento obligando al usuario a utilizar un equipo más potente.
6. Se pueden optimizar los códigos que se encargar de los cálculos de la cinemática directa y detección de colisiones para que funcionen.
7. Incorporación a un servidor para que este cuente con comunicación remota y pueda tener acceso a múltiples clientes que estén dados de alta dentro de la plataforma.

REFERENCIAS BIBLIGRAFICAS

[1] «Open Dynamics Engine - home». [En línea]. Disponible en: http://www.ode.org/. [Accedido: 29-nov-2017].
[2] «Real-Time Physics Simulation». .
[3] S. Ivaldi, J. Peters, V. Padois, y F. Nori, «Tools for simulating humanoid robot dynamics: a survey based on user feedback», en *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, 2014, pp. 842–849.
[4] «Gazebo». [En línea]. Disponible en: http://gazebosim.org/. [Accedido: 29-nov-2017].
[5] «The ARGoS Website». [En línea]. Disponible en: http://www.argos-sim.info/. [Accedido: 29-nov-2017].
[6] «Webots: features». [En línea]. Disponible en: https://www.cyberbotics.com/features. [Accedido: 29-nov-2017].
[7] «Coppelia Robotics V-REP: Create. Compose. Simulate. Any Robot.» [En línea]. Disponible en: http://www.coppeliarobotics.com/. [Accedido: 29-nov-2017].
[8] «Simulation Solutions for Mechatronic Research - CM Labs», *CM Labs Simulations*. .
[9] C. Chen, M. M. Trivedi, y C. R. Bidlack, «Simulation and graphical interface for programming and visualization of sensor-based robot operation», en *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, 1992, pp. 1095–1101.
[10] B. Kehoe, S. Patil, P. Abbeel, y K. Goldberg, «A Survey of Research on Cloud Robotics and Automation», *IEEE Trans. Autom. Sci. Eng.*, vol. 12, n.º 2, pp. 398-409, abr. 2015.
[11] M. Stenmark, M. Haage, E. A. Topp, y J. Malec, «Making Robotic Sense of Incomplete Human Instructions in High-level Programming for Industrial Robotic Assembly», en *the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17), Human-Machine Collaborative Learning workshop (accepted)*, 2017.
[12] M. Campusano y J. Fabry, «Live Robot Programming: The language, its implementation, and robot API independence», *Sci. Comput. Program.*, vol. 133, pp. 1-19, ene. 2017.
[13] M. A. Olivares-Mendez, S. Kannan, y H. Voos, «Vision based fuzzy control autonomous landing with UAVs: From V-REP to real experiments», en *Control and Automation (MED), 2015 23th Mediterranean Conference on*, 2015, pp. 14–21.
[14] «ROS.org | Powering the world's robots». .
[15] M. Qi, Y. Li, K. Xiang, y Y. Ge, «A wireless inertial measuring system for human motion analysis», en *Information and Automation (ICIA), 2016 IEEE International Conference on*, 2016, pp. 1021–1025.
[16] J. J. O. Barros, V. M. F. dos Santos, y F. M. T. P. da Silva, «Bimanual Haptics for Humanoid Robot Teleoperation Using ROS and V-REP», 2015, pp. 174-179.
[17] T.-M. Shih y J.-J. Huang, «Applying hierarchical fuzzy theory on inverse kinematic multi-DOF robot arm control», en *Control Automation Robotics & Vision (ICARCV), 2014 13th International Conference on*, 2014, pp. 468–473.
[18] «Unity». [En línea]. Disponible en: https://unity3d.com/es. [Accedido: 21-mar-2018].
[19] Game physis engine development ----- ia
[20] Handbook of Industrial Robotic DOI:10.1002/9780470172506