ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DISEÑO Y SIMULACIÓN DE UN SISTEMA DE TELEOPERACIÓN DE UN ROBOT HUMANOIDE NAO

DISEÑO Y SIMULACIÓN DEL CONTROL DE POSICIONAMIENTO DE UN ROBOT HUMANOIDE NAO

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y AUTOMATIZACIÓN

ALEX JAVIER ENRÍQUEZ SALINAS

alex.enriquez@epn.edu.ec

DIRECTOR: GEOVANNY DANILO CHÁVEZ GARCÍA

danilo.chavez@epn.edu.ec

DMQ, octubre 2022

CERTIFICACIONES

Yo, Alex Javier Enríquez Salinas declaró que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

Alex Javier Enríquez Salinas

Certifico que el presente trabajo de integración curricular fue desarrollado por Alex Javier Enríquez Salinas, bajo mi supervisión.

Dr. GEOVANNY DANILO CHÁVEZ GARCÍA

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

NOMBRE_ESTUDIANTE

ALEX JAVIER ENRÍQUEZ SALINAS

NOMBRE_DIRECTOR

GEOVANNY DANILO CHÁVEZ GARCÍA

NOMBRE_COLABORADOR(ES)

VIVIANA ISABEL MOYA GONZÁLEZ

KLEBER DARIO PATIÑO CAIZA

DEDICATORIA

Dedico este trabajo a Dios, a mi familia, amigos, compañeros, educadores de la Escuela Politécnica Nacional y a todas las personas que buscan, debaten y crean el conocimiento, espero que este trabajo sea una ayuda que les sirva de guía para futuros proyectos y sea una base para crear nuevas ideas que puedan hacer el futuro de la humanidad más próspero y lleno de innovación.

AGRADECIMIENTO

En primer lugar, agradezco a Dios quien ha sido mi fuente de apoyo e inspiración durante toda mi vida por haberme dado la fortaleza, salud, vida y su bendición en cada momento feliz o triste que he tenido.

A mi familia por siempre apoyarme y darme todo lo necesario para llegar hasta donde he llegado.

A mi madre Mónica Salinas por darme su apoyo incondicional, por su gran amor y acompañarme en cada etapa de mi carrera y sobre todo por ser una gran amiga y siempre saberme guiar en cada situación.

A mi Padre Fernando Enríquez por siempre estar pendiente de mí y apoyarme en todo lo que he necesitado y por aconsejarme y corregirme cuando no he ido por el camino correcto.

A mi hermano David Salinas quien ha sido mi modelo para seguir, por ser una fuente de conocimientos que han influenciado en mi persona y por ser mi compañero de juegos desde el inicio de mi vida.

Al Dr. Danilo Chávez quien ha sido mi tutor de tesis que ha sabido guiarme y transferirme su conocimiento para la elaboración de este documento.

Al Ing. Kleber Patiño y la Dr. Viviana Moya quienes me han brindado consejos y me han sabido dar sus puntos de vista y correcciones para la elaboración de este trabajo.

A la Escuela Politécnica Nacional por haberme brindado todo el conocimiento que poseo y sobre todo por brindarme una educación profesional de excelencia.

A todos los amigos que he conseguido en el transcurso de mi vida estudiantil Jonathan, Hans, José, Sebastián y Adrián que han aportado con su conocimiento y grandes experiencias que han hecho la vida estudiantil más divertida.

A todos los amigos que he hecho en el transcurso de mi vida en el colegio, universidad y trabajo que siempre me han sacado una sonrisa.

ÍNDICE DE CONTENIDO

RESUMEN

Este documento muestra cómo controlar al robot humanoide NAO mediante el desarrollo de un algoritmo de control de posición basado en Lyapunov, se da conocer como simular y como entablar la comunicación entre diferentes programas para la transferencia de datos y comandos. En primera instancia se expone que la cinemática y dinámica del robot NAO se encuentra en el Software de CoppeliaSim.edu y los datos de posición con respecto al marco de referencia del robot NAO serán enviados a Python, se muestra como la transferencia de datos entre Python-CoppeliaSim edu se realiza mediante la utilización de APIs remotos.

En segunda instancia se da a conocer las funcionalidades de la librería de NAOqi la cual permite ejecutar de manera sencilla los comandos que dispone el software de Choregraphe dentro del software de Python mediante la utilización de un Proxy, los comandos del proxy permiten que el robot ejecute la acción de control proveniente del controlador que se encuentra en Simulink. En tercera instancia se muestra el algoritmo del controlador que se ejecuta en Simulink y como transferir los datos entre Matlab/Simulink-Python mediante la utilización de UDP.

Finalmente se expone como desarrollar el control de posición basado en Lyapunov del robot NAO a través de un modelo reducido basado en el modelo cinemático de un robot Uniciclo y como este se comporta en el robot humanoide NAO ante múltiples cambios de referencia, se analiza los resultados que se obtiene mediante el uso índices de desempeño como lo son el ISE e ISU.

PALABRAS CLAVE: Lyapunov, Controlador de posición, NAO, Robot humanoide, NAOqi.

ABSTRACT

This document shows how to control the humanoid robot NAO through the development of a Lyapunov-based position control algorithm, how to simulate and how to establish communication between different programs for data and command transfer. In the first instance, it is exposed that the kinematics and dynamics of the NAO robot is found in the CoppeliaSim.edu Software and the position data with respect to the reference frame of the NAO robot will be sent to Python, it is shown as the data transfer between Python - CoppeliaSim edu is done using remote APIs.

In the second instance, the functionalities of the NAOqi library are disclosed, which allows the commands available in the Choregraphe software to be easily executed within the Python software using a Proxy, the proxy commands allow the robot to execute the control action coming from the controller found in Simulink. In the third instance, the controller algorithm that runs in Simulink and how to transfer data between Matlab/Simulink-Python using UDP is shown.

Finally, it is exposed how to develop the Lyapunov-based position control of the NAO robot through a reduced model based on the kinematic model of a Unicycle robot and how it behaves in the NAO humanoid robot before multiple changes of reference, the results are analyzed which is obtained using performance indices such as the ISE and ISU.

KEYWORDS: Lyapunov, Position Controller, NAO, Humanoid Robot, NAOqi.

1. INTRODUCCIÓN

La robótica es una rama de la ingeniería que en los últimos años ha ido tomando un impulso cada vez más fuerte esto debido a la gran cantidad de tareas que esta puede realizar de manera eficaz y autónoma por lo que estudiar y entender el desarrollo de esta rama es de vital importancia tanto para la vida cotidiana como para el nivel industrial.

Se conoce como robot humanoide a la maquina la cual imita las funciones y comportamientos del ser humano, esto se hace con fin de poder ejecutar las mismas tareas que una persona hace y poder utilizarlos en la vida diaria e industrial.

En la actualidad existen algunos prototipos que tratan de imitar comportamientos humanos como lo es el Robot HOAP de Fujitsu el cual incorpora gran cantidad de funcionalidades pero su alto costo hace que sea poco accesible al público general, por ello han salido nuevos modelos más económicos y accesibles tanto para el ámbito industrial y educacional como lo es el robot humanoide NAO, con este robot se puede estudiar sus diferentes funciones de voz, audio y movimiento (camina, posturas o tomar cosas).

Todas las funcionalidades que incorpora el robot NAO o robots similares tienen el objetivo de hacer operativo al robot y cada una de dichas cualidades esta desarrollada para implementar nuevos algoritmos de control que permitan crear nuevas funciones con los elementos que este brinda.

Uno de los algoritmos mayormente desarrollados para este tipo de robot son los algoritmos de control de posición, los cuales permiten que el robot se sitúe en una posición especifica o deseada de manera autónoma, para ello se debe entender como ejecutar el algoritmo, que tipo de control poner y en que plataforma implementarlo.

La teoría de control enseña una gran variedad de estrategias que permitirán controlar al robot, una que es novedosa por su funcionalidad son los algoritmos basados en Lyapunov, el desarrollo de este es una propuesta diferente ya que la gran mayoría de teorías de control se centran en controladores del tipo PID.

El desarrollo del algoritmo de control de posición que en este caso será el de Lyapunov es importante pero también lo es conocer las plataformas que permiten implementar y simular al robot NAO, para ello es fundamental estar al tanto de programas como: Matlab/Simulink, Python y sus diferentes módulos o librerías externas como lo es NAOqi, la librería que permite ejecutar comandos de Choregraphe en Python o CoppeliaSim edu el cual permite simular al robot y probar la cinemática y dinámica del mismo.

Conocer tanto como desarrollar el controlador y como implementar el mismo mediante diferentes programas de simulación da un abanico de posibilidades para el desarrollo de algoritmos de control pero también es necesario entender como el algoritmo se comporta ya que dentro del mismo hay diferentes controladores que pueden hacer que un sistema sea mejor o peor, por esto es importante comprender los resultados de los índices de desempeño ya que los mismos ayudarán a optimizar el algoritmo de control que se desarrolle.

Es necesario vislumbrar todas estas temáticas ya que en un futuro no muy lejano la robótica tomará un impulso en todos los ámbitos de la vida diaria, el conocimiento de cómo crear y entender estas nuevas teorías impulsará a entrar en una nueva era del desarrollo tecnológico.

1.1. OBJETIVO GENERAL

Diseñar y simular el control de posición de un robot humanoide Nao V6 mediante el uso de diferentes programas de simulación como: Matlab-Simulink, Python, Choregraphe y CoppeliaSim edu para que el robot sea capaz de llegar a una coordenada de posición XY deseada.

1.2. OBJETIVOS ESPECÍFICOS

1. Revisar el contenido bibliográfico con respecto al manejo y simulación del robot Nao V6.
2. Establecer la comunicación para el envío y recepción de datos entre Matlab-Simulink y Python.
3. Entablar una conexión entre Python, Choregraphe y CoppeliaSim edu para la ejecución de comandos.
4. Desarrollar un controlador de posición basado en Lyapunov para el robot Nao V6 e implementarlo en Simulink.
5. Probar el correcto funcionamiento del sistema desarrollado verificando que el robot llegue a su posición final deseada.

1.3. ALCANCE

El Alcance del proyecto queda definido a partir de los siguientes puntos:

- Revisar el contenido bibliográfico relacionado a la utilización y simulación del Robot Nao V6.

- Estudiar y entender los programas necesarios para la simulación del Robot como lo son Matlab-Simulink, Python, Choregraphe y CoppeliaSim edu.
- Diseñar un controlador de posición basado en Lyapunov el cual se deberá de implementar en Simulink.
- Establecer la comunicación bidireccional en el envío y recepción de información, para así poder cerrar el lazo de control y de esta manera enviar la información de posición XY del usuario, los datos del controlador y los comandos de ejecución para que el robot se mueva.
- Desarrollar una interfaz gráfica la cual permita observar el funcionamiento de todo el sistema desarrollado.
- Comprobar el funcionamiento del controlador y del sistema elaborado ante diferentes cambios de referencia mediante la evaluación de índices de desempeño como el ISE e ISU.

1.4. MARCO TEÓRICO

1.4.1. ROBOT HUMANOIDE NAO

El Robot Humanoide NAO es un robot creado por la empresa francesa Aldebaran el cual fue diseñado para abarcar a la mayor cantidad de público disponible ya sea en el ámbito educacional, tecnológico o del entretenimiento su creación se debe a la necesidad de las personas de obtener un Robot que sea asequible al precio del consumidor debido a que este tipo de Robots pueden exceder un costo de hasta 50 mil euros como es el caso del Robot HOAP de Fujitsu. Con la creación del Robot Humanoide Nao se puede tener un precio al consumidor mucho más reducido por valores aproximados de hasta 10 mil euros, este precio se obtiene debido a que este es fabricado en masa y para un consumidor más general [1].

El Robot es muy versátil y dinámico para su costo, este dispositivo posee una altura de 58 centímetros y consta de 25 grados de libertad y una gran cantidad de sensores distribuidos en toda su estructura dentro de los cuales encontramos que posee sensores de presión, 2 cámaras del tipo HD, 4 micrófonos y 9 sensores táctiles también puede trabajar con una gran cantidad de softwares como lo son Python, C++, Matlab, JAVA y .NET [1].

1.4.1.1. Características Principales

En la Figura 1. 1 se puede apreciar el aspecto físico del robot Nao el cual goza de una altura de 58 cm con un peso de 4.5 Kg, posee 25 grados de libertad los cuales están

distribuidos en sus diferentes articulaciones de estos 11 grados de libertad se encuentran disponibles en sus dos piernas, 5 grados de libertad por cada brazo en conjunto un total de 10 grados por ambos brazos, 1 grados de libertad por cada mano en conjunto 2 grados por ambas manos y 2 grados en el cuello, en las articulaciones de las piernas se debe considerar cada una como 6 articulaciones ya que se tiene un motor general ubicado en la cadera que interactúa con ambas [2].

El Robot cuenta con un procesador AMD GEODE x86 de 500 MHz ubicado en la cabeza más una unidad SDRAM con capacidad de 256 Mb, adicional a esto también se incorpora una memoria Flash de 1Gb como soporte extra, también consta de un microcontrolador ARM7-60 MHz el cual se encuentra en el torso cuya función principal es la de distribuir la información disponible al resto de microcontroladores, el robot se puede comunicar mediante WIFI 802.11 o mediante su puerto Ethernet, se debe recordar que el sistema principal del robot es basado en Linux pero este puede ser modificado [1].

El robot NAO V6 se lo puede conseguir por un precio de $ 10.500 USD haciéndonos acreedores de un SDK, API y manuales completos con un rango de garantía de 2 años y capacidad de trabajar con distintos programas como Python, Java o C++ y de poder interactuar con sus múltiples sensores: 8 sensores de posición, 4 micrófonos, 2 receptores y emisores de ultrasonido, 2 cámaras HD, 9 sensores táctiles, giroscopios y acelerómetros [3].

Figura 1. 1 Robot Humanoide Nao V6 [3].

1.4.2. PYTHON

Python es un lenguaje de programación de software libre creado por el programador holandés Guido Van Rossum a finales de la década de los 80 se caracteriza por ser un lenguaje de alto nivel ya que permite realizar un programa en pocas líneas de código esto debido a la gran cantidad de funcionalidades ya implícitas dentro de su estructura como son conjuntos, listas o tuplas lo que hace la programación más sencilla [4].

1.4.2.1. Sintaxis

Python es un lenguaje de programación el cual se caracteriza por tener una programación muy sencilla como se aprecia en la Figura 1. 2 se puede decir que para ejecutar una acción en este caso la de imprimir un Hola mundo basta con escribir su comando seguido de la acción, esto resulta relativamente sencillo ya que en otros lenguajes como es el caso de C++ la tarea puede ser un poco más demorosa, se debe tener en cuenta que Python utiliza un sistema de identación o sangrado que no es más que un sistema jerárquico que reemplaza las llaves o corchetes como en C++ para dar lugar a los espacios, es decir la función de un código principal estará sin espacios y si se pusiera otra línea de código que depende de la primera esta deberá tener una identación o espacio y así sucesivamente abra más espacios entre mayor dependencia tenga el código de otro lo mismo que se hacía con las llaves en otros lenguajes solo que con este se verá de manera más organizada [4].

```
print "Hello World"
```

Figura 1. 2 Programación Hola mundo en Python [4].

1.4.2.2. Sentencias condicionales

Este tipo de sentencia se caracteriza por la línea de código que empieza con if y sirve para hacer cumplir una condición específica que el programador considere necesaria y que a través de esta condición se tome una acción como se puede apreciar en la Figura 1. 3 se puede ver que la estructura viene dada por un if o si, luego la condición e inmediatamente por dos puntos para concluir la misma, las demás líneas de código muestran un nivel de espaciado hacia la derecha lo que se denomina como identación y sirve para dar un orden jerárquico [5].

```
4 if a != 0:
5     x = -b/a
6     print 'Solución:␣', x
```

Figura 1. 3 Sentencia if [5].

1.4.2.3. Sentencias Iterativas

Este tipo de sentencias permiten realizar una acción una y otra vez en forma de bucle mientras una condición se cumpla, el caso más común dentro del código es el de la instrucción while o mientras un ejemplo de esto se puede visualizar en Figura 1. 4 [5].

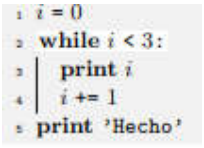

Figura 1. 4 Sentencia iterativa while [5].

1.4.2.4. Funciones

Este tipo de elementos son muy útiles a la hora de realizar líneas de código que se necesita volver a utilizar en otras partes del código, al definir una función se puede utilizar la traza de código varias veces dentro del código principal según se necesite llamar un ejemplo de una función se puede ver en la Figura 1. 5.

```
def multiplica(val1, val2):
  return val1 * val2
```

Figura 1. 5 Ejemplo de función en Python [5].

1.4.2.5. Comunicación basada en hilos

También llamada comunicación en paralelo o comunicación de procesos es fundamental para ejecutar procesos que requieren que se ejecute datos al mismo tiempo para este tipo de acción uno de los mejores módulos es el comando threading, este tipo de comando ayuda a definir que los objetos del thread se ejecuten a la vez dentro de un proceso mayor, esto es útil por ejemplo a la hora de tomar datos de entrada o salida de varios programas que se quiera que los datos se procesen y ejecuten al mismo tiempo [6].

1.4.3. SDK DE PYTHON NAOQI

El módulo de NAOqi es uno de los softwares principales para poder controlar y ejecutar diversos comandos desde Python hacia el robot Nao, a través de este módulo se puede ejecutar funciones ya sea de audio, video o movimiento a través de un intercambio de información homogéneo, NAOqi actúa como un intermediario que al iniciarse ejecuta todas las bibliotecas necesarias para ejecutar los comandos [7].

Broker: es el encargado de proporcionar un servicio de búsqueda, en este sentido podrá encontrar cualquier módulo que se encuentre en la red o árbol si este ha sido llamado [7].

Proxy: se lo define como un objeto el cual adquirirá todas las características del módulo que haya sido definido, en otras palabras, si se define un proxy con el método AlMotion este conseguirá todas las funciones que brinda este método de movimiento, un ejemplo de esto se puede ver en la Figura 1. 6, lo primero que hace el código es llamar al módulo de NAOqi para que se importen todas las librerías necesarias para manipular al robot, luego

se define un objeto mediante ALProxy y lo que se llena es la cualidad que se quiere dar al proxy en este caso ALMotion o de movimiento luego se pondría la dirección IP en nao.local seguido de su número de puerto en 9559, con esto definido se puede ejecutar el comando moveTo que permitirá mover al robot a un punto específico en metros utilizando coordenadas cartesianas (x, y , ángulo de orientación) u ocupar otros comandos como el move que permitirá controlar la velocidad en m/s del robot igualmente en coordenadas cartesianas [8].

```
from naoqi import ALProxy
motion = ALProxy("ALMotion", "nao.local", 9559)
motion.moveInit()
motion.moveTo(0.5, 0, 0)
```

Figura 1. 6 Ejemplo de código con NAOqi en Python mediante el módulo ALProxy con AlMotion [8].

1.4.4. COPPELIASIM

Es un simulador que ofrece la capacidad de tener un entorno de simulación ya sea para la industria como automatización de fábricas, elaboración de prototipos de robots como verificación de uso de actuadores o como monitores remotos entre otras muchas opciones, esto se da gracias a su arquitectura de control distribuido, esto permite controlar cada objeto o modelo de manera independiente ya sea mediante la elaboración de un script, un cliente API remoto, un complemento y un ROS, el programa también tiene la facilidad de poder desarrollar los controladores mediante Python, C/C++, Lua, Java, Octave y Matlab [9].

Cliente API remoto: es uno de los 6 métodos de programación que existen en CoppeliaSim los cuales son Scripts, Plugins, Nodo BueZero, Nodo ROS y Complementos (add-on), este método permite que una aplicación o programa externo se pueda conectar con CoppeliaSim mediante el uso de API remotos [9].

Se puede apreciar un ejemplo de Cliente API remoto en la Figura 1. 7, en este caso se ocupa este método para abrir un hilo de comunicación para una IP y puerto determinados, en esta situación específica se entabla una conexión entre Python y CoppeliaSim otro ejemplo de esto se puede apreciar en Figura 1. 8, este caso una vez abierto el hilo de comunicación podemos acceder a diferentes funciones como puede ser tomar el valor de las coordenadas de posición que se encuentra nuestro objeto o robot dentro del entorno de simulación [10].

```
clientID = sim.simxStart('127.0.0.1', 19997, True, True, 5000, 5)
```

Figura 1. 7 Ejemplo de Cliente API remoto creación de un hilo de comunicación entre dos programas.

```
NAO_pos = sim.simxGetObjectPosition(clientID, NAO_Handle[1], -1, sim.simx_opmode_streaming)[1]
```

Figura 1. 8 Ejemplo de Cliente API remoto obtención de coordenadas o posición de un objeto o robot dentro del entorno de simulación.

1.4.5. CHOREGRAPHE

Choregraphe es un software muy fácil de utilizar en este entorno se puede realizar cualquier acción que se desee ya sea que el robot camine, gire, se acueste, baile o se ponga de pie mediante la selección de cajas las cuales ya tienen programado el código en alto nivel de dichas acciones, al realizar la programación mediante bloques o cajas lo hace un programa muy intuitivo y fácil de usar a la hora de querer efectuar una acción en el Robot [11].

Se debe tener en cuenta que en Choregraphe no solo se puede programar en bloques sino que también se puede hacer utilizando el SDK de Python con NAOqi por ambos métodos se puede hacer prácticamente las mismas funciones destacando que la primera es más intuitiva pero el código se ejecuta más lento y la segunda es un poco más compleja de utilizar pero el código se ejecuta más rápido, NAOqi será de gran ayuda ya que mediante este se puede importar todas las funcionalidades que brinda Choregraphe empleadas en Python lo que ayudará a realizar una programación más fácil, por ejemplo programar al Robot solo con V-REP sería programar cada motor y actuador lo que haría la programación muy difícil mientras que con Choregraphe no [11].

Se puede apreciar en la Figura 1. 9 como es la programación mediante bloques de Choregraphe se elige un bloque y este tendrá una función específica en este caso la función del bloque hace que el robot diga Hello [12].

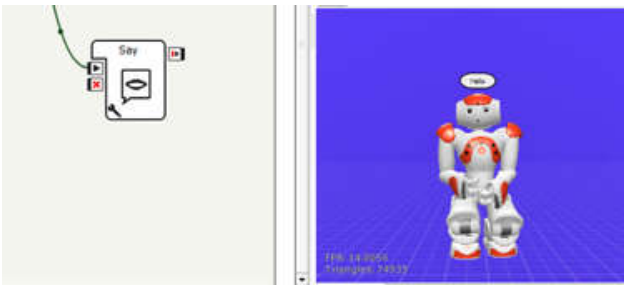

Figura 1. 9 Ejemplo de programación en Choregraphe [12].

1.4.6. MATLAB/SIMULINK

Matlab: es un lenguaje de programación el cual se caracteriza por ser de alto nivel fue creado en 1970 con el objetivo de realizar cálculos técnicos y matriciales en este entorno se pueden desarrollar algoritmos, adquisición de datos, modelamiento y simulación, cálculo y matemática, desarrollo de gráficos científicos entre otras funciones [13].

Simulink: es un entorno de simulación en el cual la programación que se realiza está basada en diagrama de bloques de multidominio, en este entorno se podrá simular sistemas dinámicos, sistemas de control entre otras. Simulink ofrece la capacidad de probar y verificar la simulación de manera continua a parte tiene una compatibilidad de enlazarse con Matlab para exportar gráficos o crear bloques con líneas de código de Matlab para trabajar en Simulink [13].

UDP Send y Receive: cómo se puede apreciar en la Figura 1. 10 este tipo de bloques son de gran utilidad ya que permiten enviar y recibir información de una aplicación a otra ya sea de Simulink a Python o viceversa mediante la configuración de una dirección IP remota y un puerto remoto [14].

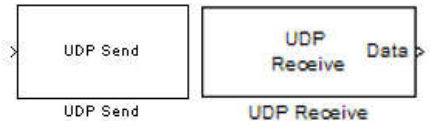

Figura 1. 10 Bloques UDP de Symulink para envío y recepción de datos.

Matlab Function: es un bloque muy útil ya que permite escribir funciones con el lenguaje de programación de Matlab y utilizar dichas funciones en los bloques de Simulink este bloque siempre tendrá entradas y salidas como se muestra en Figura 1. 11 [14].

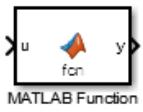

Figura 1. 11 Bloque MATLAB Function para Simulink [14].

1.4.7. ROBÓTICA MÓVIL

La robótica móvil se encarga de estudiar el desplazamiento de un robot que tienen cierto grado de libertad desde un punto A hasta un punto B para ello existen varios tipos según su sistema de locomoción, se pueden encontrar con ruedas, patas u orugas los cuales le permiten desplazarse por su entorno de trabajo ya sea este una superficie plana o irregular [15].

Los mecanismos por los cuales están dispuestas sus ruedan pueden ser muy variados y clasificarse en varios tipos como lo son los Robots omnidireccionales, uniciclo, triciclo y cuatriciclo, destacando que el robot uniciclo es uno de los robots más utilizados por los investigadores de robótica para realizar ensayos o pruebas a las estrategias de control debido a que posee una cinemática relativamente sencilla al solo estar compuesto por dos ruedas fijas y una rueda loca para dar estabilidad [16].

Se debe tomar en cuenta que para cualquier tipo de robot móvil se puede considerar al mismo como una partícula en movimiento y de esta manera poder conseguir sus ecuaciones cinemáticas, para ello se puede tomar en cuenta un robot uniciclo o robot diferencial para el desarrollo de las técnicas de control debido a su sencilla cinemática, como se puede ver en la Figura 1. 12 las ecuaciones que describen el movimiento del robot están dadas por ((1. 1), ((1. 2) y ((1. 3) las cuales muestran el moviendo en función de sus velocidades lineales en x, y, el módulo de la distancia a desplazarse u, el ángulo de giro φ y velocidad angular w [15].

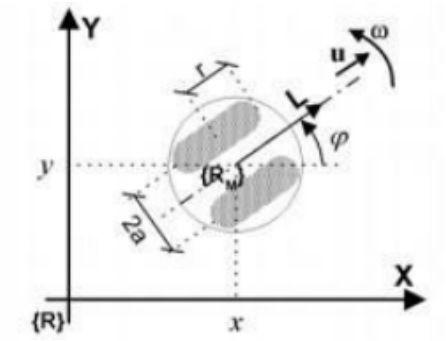

Figura 1. 12 Cinemática de un Robot Móvil [16].

$$\dot{x} = u * cos\,(\varphi) \tag{1. 1}$$

$$\dot{y} = u * sin\,(\varphi \tag{1. 2}$$

$$\dot{\varphi} = w \tag{1. 3}$$

1.4.8. SISTEMAS DE CONTROL

Los sistemas son caracterizados por presentar un conjunto de elementos que son capaces de influir en el trabajo del sistema, su finalidad es modificar las variables de salida de modo que estas alcancen una consigna o referencia de entrada impuesta como condición esto se logra mediante la manipulación de las variables de control [17].

Control en lazo abierto: este tipo de sistema se caracteriza por no tener una realimentación o sensor que permita medir o conocer el estado de la salida del sistema por lo cual no se puede comparar con la variable de entrada del sistema [18].

Control en lazo cerrado: se caracteriza por tener una realimentación de la variable de salida hacia la variable entrada mediante la utilización de un sensor que efectué la medición de esta, mediante esto se puede comparar la variable de salida con la referencia o variable de entrada y generar una acción de control que permita disminuir el error y corregir la salida [18].

Control de posición: un control de posición tiene como objetivo que el robot o elemento móvil se dirija hacia una posición o punto del espacio específico que se ha determinado con anterioridad de manera autónoma, para ello se deberá de emplear cierta estrategia de control las cuales pueden ser un control PID, P, PI, PD, Lyapunov o control difuso entre otras estrategias, con el objetivo de minimizar el error hasta que la posición deseada y la posición actual tienden a ser cero [19].

1.4.9. CONTROLADOR BASADO EN LYAPUNOV

Lyapunov fue el primer científico en introducir la idea de una función de energía ficticia a la que llamó función de Lyapunov, la misma se puede aplicar a cualquier función que utilice ecuaciones diferenciales ya sean estas un sistema lineal o no lineal. Es un método que tiene dos variantes con respecto a su estabilidad, el primer método llamado método indirecto necesita las soluciones de las ecuaciones diferenciales que describen a el sistema de manera explícita mientras que el segundo método no necesita trabajar con dichas soluciones explicitas lo que le hace un método general y extendido en el campo por ello se le da el nombre de método directo de Lyapunov [20].

El segundo método o método directo de Lyapunov se basa en un principio básico el cual dice que si la energía total de un sistema eléctrico o mecánico se disipa de manera continua entonces el sistema sea lineal o no lineal eventualmente quedará en un punto de equilibrio es decir se puede conocer la estabilidad mediante el cumplimiento de ciertos criterios de una función o función de Lyapunov [15].

La teoría de Lyapunov dice que la función debe cumplir 4 condiciones para que el sistema escalar se considere asintóticamente estable, posea un equilibrio en el origen y tenga un estado uniforme, dichas condiciones son:

1. $V\ (x)\ es\ continuo\ y\ tiene\ derivadas\ continuas$

2. $V\ (0) =\ 0\ la\ función\ evaluada\ en\ cero\ es\ cero$

3. $V\ (x) > 0\ es\ positiva\ mayor\ que\ cero\ para\ todas\ las\ x\ a\ excepción\ del\ cero$

4. $\frac{dV(x)}{dt} < 0\ la\ derivada\ de\ la\ función\ es\ menor\ que\ cero$

Una vez que se sabe cuáles son las condiciones que se debe cumplir para definir una función de Lyapunov los pasos para hacer un sistema estable son:

Paso 1: se debe elegir una función de prueba que cumpla las tres condiciones de estabilidad anteriormente vistas, la primera estabilidad asintótica uniforme, la segunda estabilidad asintótica total y la tercera estabilidad de Lyapunov [21].

Paso 2: derivar la función de prueba a lo largo de la trayectoria del sistema como se ve en (1. 4.

$$\dot{x} = f(x, u, t) \tag{1. 4}$$

Seleccionar una ley de control u como ((1. 5) la cual otorga una estabilidad al sistema normalmente esta función viene acompañada de ganancias y parámetros que cumplen con ((1. 6) esto asegura que el sistema sea asintóticamente estable en lazo cerrado [21].

$$u = u(x) \tag{1. 5}$$

$$\frac{dV(x)}{dt} < 0\ para\ x \neq 0 \tag{1. 6}$$

1.4.10. ÍNDICES DE DESEMPEÑO DE LOS SISTEMAS DE CONTROL

Los índices de desempeño tienen la función de indicar que tan bien se comporta el sistema mediante el resultado de una medida cuantitativa la que mostrará si un sistema se comporta de manera eficiente esta medida permite evaluar al sistema frente a permutaciones o cambios en los parámetros del proceso [22].

1.4.10.1. Tipos de Índices de desempeño

ISE: Integral del error cuadrático, es un índice que da mayor peso cuando el error es grande y menor peso cuando este es pequeño el objetivo es minimizar en lo posible este índice [23].

$$J_{e_1} = \sum_{k=0}^{N} e(k)^2 \tag{1. 7}$$

ITSE: Integral del tiempo por el error cuadrático, es el que penaliza el tiempo de convergencia de un estado estacionario [23].

$$J_{e_2} = \sum_{k=0}^{N} k * e(k)^2 \quad (1.\ 8)$$

IAE: Integral del valor absoluto del error, es el encargado de penalizar la acción de seguimiento [23].

$$J_{e_3} = \sum_{k=1}^{N} |r_k - y_k| \quad (1.\ 9)$$

ISU: Integral de la salida de control cuadrática, nos muestra el esfuerzo del control o la medida de la energía que emplea el controlador [23].

$$J_{u_1} = \sum_{k=0}^{N} (u(k) - u_{ss})^2 \quad (1.\ 10)$$

TVU: Integral de la variación total de control, nos indica la evolución que tiene el controlador a la salida [23].

$$J_{u_2} = \sum_{k=1}^{N} |u_k - u_{k-1}| \quad (1.\ 11)$$

1.4.11. INTERFACES DE USUARIO

1.4.11.1. Normativa ISO 9241-151

Es una normativa que indica como deben de estar estructuradas las interfaces de usuario ya sea para páginas web u otros tipos, en esta normativa nos señala que la interfaz debe estar centrada, focalizar los comandos a los objetos principales sin poner demasiado contenido que puede hacer que el usuario se pierda, ser lo más simple posible sin dejar de lado las propiedades principales que cumple la interfaz, debe tener una presentación coherente y colores los cuales dependiendo de la aplicación deberán de ser o muy vistosos para advertencia o pálidos para evitar el desgaste visual [24].

1.4.11.2. Guide de Matlab/Simulink

Una interfaz de usuario permite presentar un programa el cual no es muy vistoso para el usuario, pero con una interfaz si lo es, para ello existen varios tipos de programas que permiten hacer eso, uno de ellos es el Guide de Matlab mediante este se puede desarrollar múltiples aplicaciones útiles y lo más importante poder conectarnos a Simulink y controlar los comandos que este posee, se puede ver en la Figura 1. 13 como es la interfaz de desarrollo que este posee en donde se puede crear botones, textos, tablas y otras funcionalidades útiles en una interfaz [25].

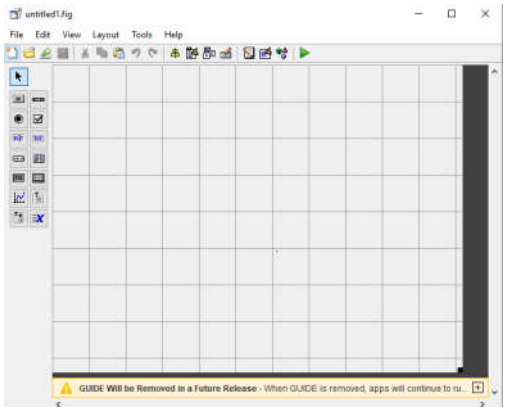

Figura 1. 13 Desarrollo de interfaz de usuario en Guide de Matlab/Simulink.

2. METODOLOGÍA

A continuación, se expone la metodología empleada en el trabajo de integración curricular la cual nos muestra de manera explicita todo el trabajo y procedimiento que se elaboró en la investigación.

La información se recopilo de fuentes primarias y secundarias, las fuentes primarias están constituidas por: libros, artículos científicos y revistas técnicas, mientras que las fuentes secundarias están conformadas por: manuales o guías de usuario de programas del Robot NAO que se encuentran en sitios web, trabajos de integración curricular previos e informes técnicos.

2.1. ENTORNO DE SIMULACIÓN COPPELIASIM PARA EL ROBOT NAO

2.1.1. MARCO DE REFERENCIA DE COPPELIASIM CON RESPECTO AL ROBOT NAO

El robot NAO realizará sus acciones de movimiento en el entorno de simulación de CoppeliaSim Edu por lo cual es importante conocer como es el ambiente que brinda el programa para que el robot NAO interactúe, en este caso consta de un marco de referencia característico el cual es una cuadrícula subdividida por cuadros más pequeños, este entorno tiene sus ejes de referencia como un plano cartesiano en donde el eje x se encuentra a lo largo de una línea horizontal, el eje y a lo largo de la línea vertical y el eje z en este caso estaría proyectado hacia fuera de la pantalla o perpendicular a ambos ejes (x, y). Conociendo esto es relativamente fácil ubicar el origen de coordenadas (0,0) en el centro del plano cartesiano como se muestra en la **Figura 2. 1**.

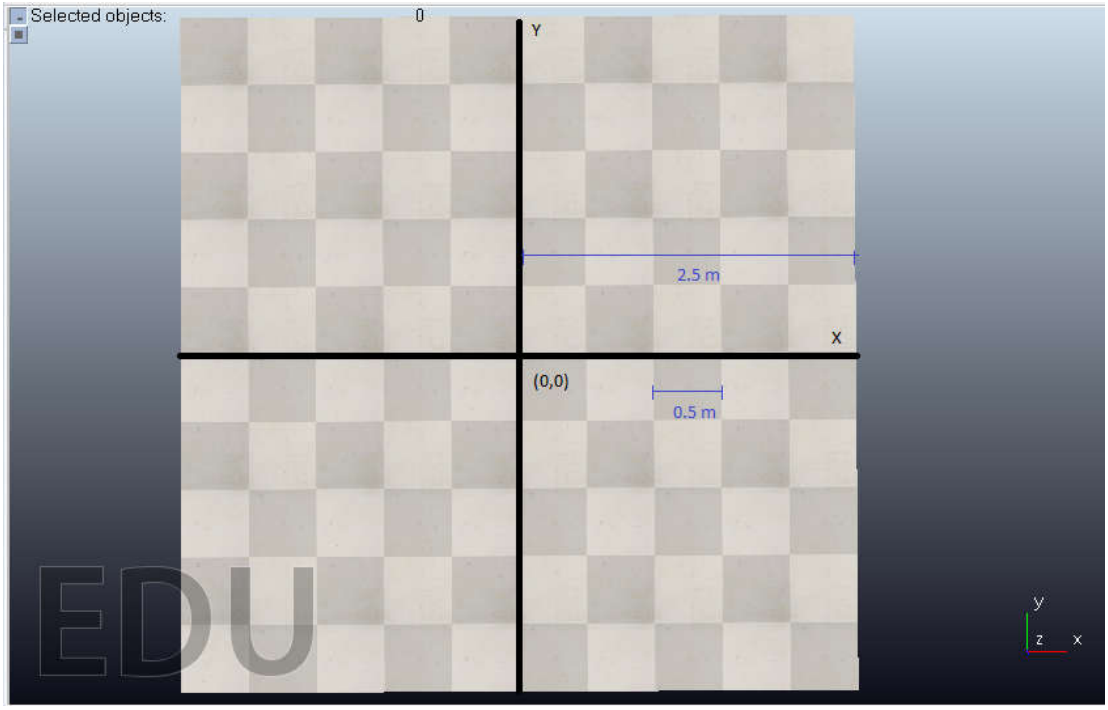

Figura 2. 1 Marco de Referencia de CoppeliaSim Edu.

Por defecto CoppeliaSim brinda una cuadrícula de 2.5 metros de largo y 2.5 metros de ancho en donde cada subdivisión del cuadrado tiene una dimensión de 50x50 cm con esto se puede dar una idea de la cantidad de espacio que se tiene para probar el control de posición del robot.

Por cuestiones de facilidad el robot NAO estará ubicado en el origen de coordenadas como se aprecia en la **Figura 2. 2** desde esta posición el robot podrá moverse a cualquier punto del plano para ello se dividirá la referencia en cuatro cuadrantes, de esta manera el usuario podrá ingresar la coordenada específica a la que se desea que el robot vaya de acuerdo al cuadrante en que se encuentre dicha posición, en este sentido es fácil observar que si se ingresara una posición (1,1) bastará con trazar la línea imaginaria en la mente y contar el número de cuadros en (x, y) para ver el punto deseado de llegada como se ve en la **Figura 2. 3**.

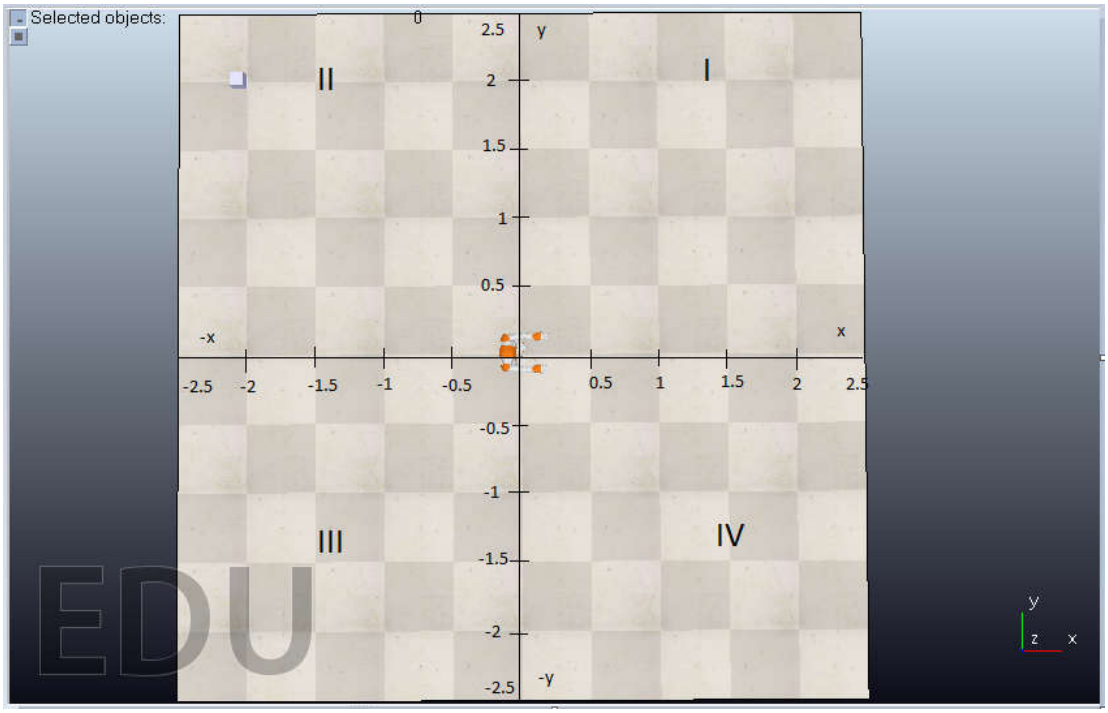

Figura 2. 2 Distribución de cuadrantes del marco de referencia de CoppeliaSim.

Para que el Robot NAO se desplace de un punto a otro es necesario tomar en cuenta ciertos parámetros que son necesarios conocer para realizar este objetivo, como se puede ver en la **Figura 2. 3** el primer parámetro que se necesita conocer es el valor de la posición real (Xreal, Yreal) el cual vendría a ser la posición en la que se encuentra en ese instante el robot, otro parámetro necesario son las coordenadas (Xdeseada, Ydeseada) las cuales son ingresadas por el usuario y dicen a qué punto del marco de referencia previamente mostrado se quiere llevar al robot NAO.

Finalmente se necesita conocer los parámetros que tomará nuestro controlador, los cuales son el módulo o la distancia por recorrer representada por D y el ángulo de orientación Theta que debe seguir para llegar al punto deseado.

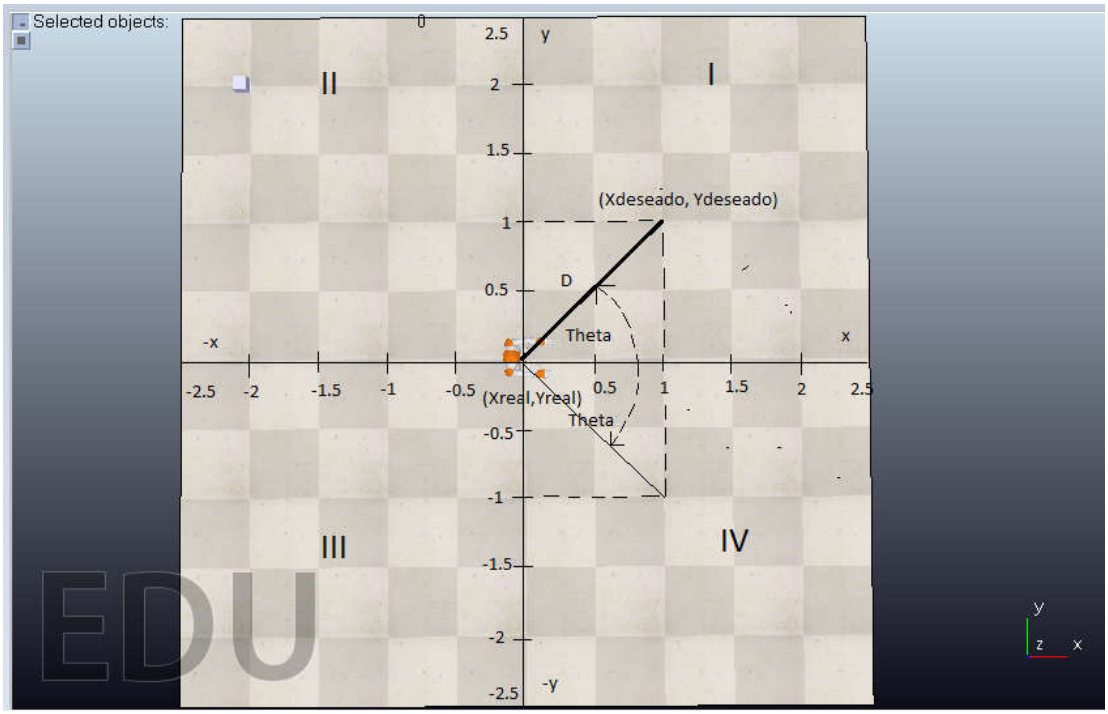

Figura 2. 3 Variables necesarias para que el robot se desplace en CoppeliaSim.

Se tiene que considerar que el ángulo Theta siempre estará referido para el eje X, para obtener dicho ángulo en los cuadrantes 1 y 2 se lo tomará desde el eje X positivo en sentido antihorario como se ve en (2. 1), (2. 2) y para los cuadrantes 3 y 4 se tomará desde el eje x positivo en sentido horario como se ve en (2. 3) y (2. 4) de esta manera se puede obtener el ángulo adecuado al cual se debe dirigir el robot.

Cuadrante I

$$\theta = \text{tang}^{-1}\left(\frac{y}{x}\right) \tag{2. 1}$$

Cuadrante II

$$\theta = \Pi + \text{tang}^{-1}\left(\frac{y}{-x}\right) \tag{2. 2}$$

Cuadrante III

$$\theta = -\Pi + \text{tang}^{-1}\left(\frac{-y}{-x}\right) \tag{2. 3}$$

Cuadrante IV

$$\theta = \text{tang}^{-1}\left(\frac{-y}{x}\right) \quad (2.\ 4)$$

Para obtener el módulo de la distancia a la que debe de ir el controlador es necesario considerar que se debe restar las coordenadas deseadas menos las coordenadas reales como en (2. 5) con ello se obtiene la distancia que falta para que el robot llegue mientras el controlador le guíe en cada punto, se debe recordar que cuando el error o el módulo de la distancia tienda a cero será cuando el robot llegue a la posición deseada.

$$D = \sqrt{(Xdeseada - Xreal)^2 + (Ydeseada - Yreal)^2} \quad (2.\ 5)$$

2.1.2. COMUNICACIÓN DE DATOS ENTRE COPPELIASIM-PYTHON PARA EL ROBOT NAO

El robot NAO tendrá que interactuar en el software de simulación de CoppeliaSim pero todos los comandos y algoritmos de control se los hace mediante otros programas entre ellos Python, por tal motivo es importante enviar y recibir los datos necesarios para efectuar la acción de control de Python-CoppeliaSIm, como se puede ver en la **Figura 2. 4** lo primero que se debe de hacer es tomar un cuboide de la plataforma de CoppeliaSim y ponerlo en una parte no muy visible del entorno de simulación ya que este cubo no tendrá más que la función de albergar el código que permite realizar el enlace de datos entre ambos programas.

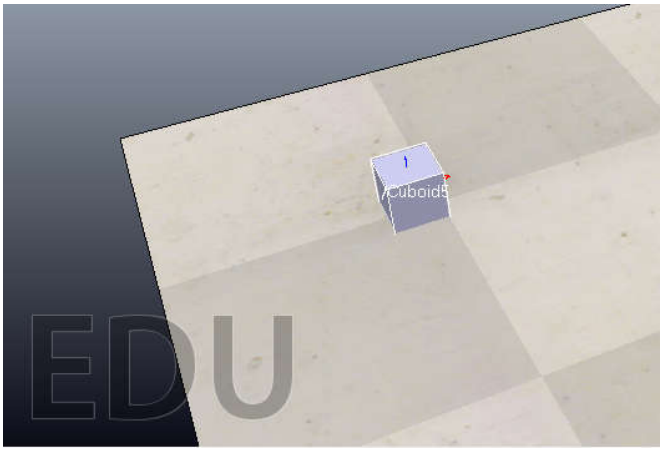

Figura 2. 4 Cuboide de CoppeliaSim.

La línea de código que permite entablar la comunicación de CoppeliaSim a Python es la que se muestra en la **Figura 2. 5** esta línea de código deberá estar puesta en la estructura de código del cuboide de V-rep y no en el código del robot NAO y con esto se tendrá creado una API remota.

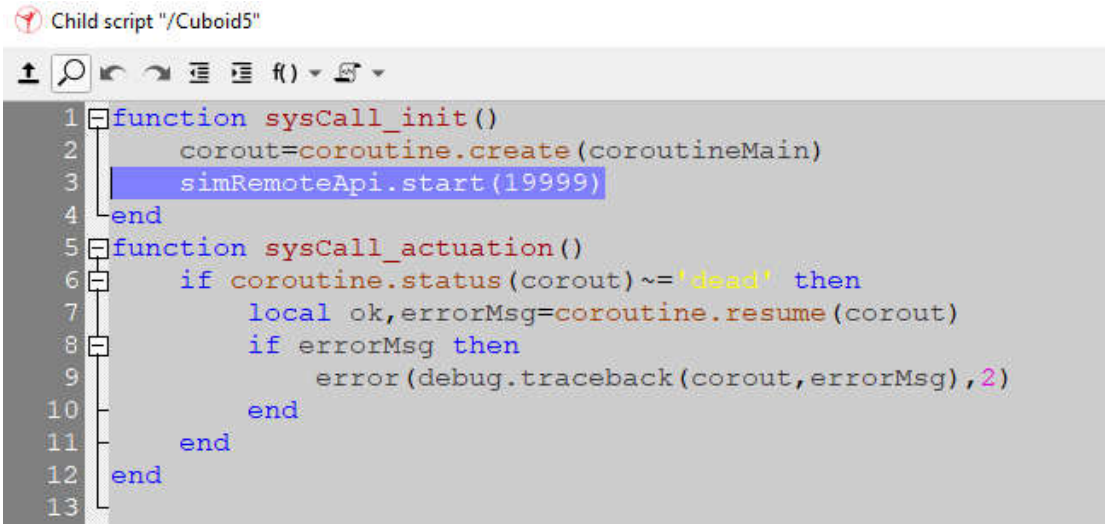

Figura 2. 5 Código para entablar la conexión de CoppeliaSim a Python mediante el Cuboide.

De la parte de la programación de Python lo que se debe de agregar es la línea de código que se muestra en **Figura 2. 6** con esto se inicia la comunicación entre ambos programas mediante la dirección IP y el número de puerto, con esto se podrá enviar y recibir datos los cuales son necesarios para elaborar el controlador.

```
clientID = sim.simxStart('127.0.0.1', 19997, True, True, 5000, 5)  # conexion al V-rep.
```

Figura 2. 6 Código para conexión de Python a CoppeliaSim.

2.1.3. COMANDOS DE EJECUCIÓN DE COPPELIASIM-PYTHON PARA EL ROBOT NAO

Una vez entablada la comunicación entre ambos programas se obtiene una gran lista de comandos útiles que ofrece CoppeliaSim en Python, para verificar que ambos programas están enlazados es útil probar ambos realizando una acción que sea relativamente sencilla de ejecutar como puede ser un Hola mundo, pero en la versión de CoppeliaSim como se muestra en la **Figura 2. 7**, esta función es importante ya que con esto se puede verificar que ambos programas se encuentran conectados y por consiguiente se podrá realizar funciones más pesadas.

```
sim.simxAddStatusbarMessage(clientID,'Hello CoppeliaSim!',sim.simx_opmode_oneshot)
```

Figura 2. 7 Hola mundo de CoppeliaSim.

Se debe tener en cuenta que el controlador necesita los parámetros de posición del Robot dentro de CoppeliaSim ya que con dichos datos se puede cerrar el lazo de control, como se aprecia en la Figura 2. 8 para obtener las coordenadas de posición de los sensores del robot se utiliza el comando simGetObjectPosition el cual da la posición en metros y para obtener el ángulo de orientación se utiliza simxGetObjectOrientation el cual da en radianes.

```
#COMUNICACIÓN RECEPCION DE DATOS DE PARTE DE COREOGRAPHE-COPPELIASIM A PYTHON-Naoqi
NAO_pos = sim.simxGetObjectPosition(clientID, NAO_Handle[1], -1, sim.simx_opmode_streaming)[1]
pos_x = NAO_pos[0]
pos_y = NAO_pos[1]
NAO_orient = sim.simxGetObjectOrientation(clientID, NAO_Handle[1], -1, sim.simx_opmode_streaming)[1]
gamma_real = NAO_orient[2]
```

Figura 2. 8 Comandos para cerrar el lazo de control con Python-CoppeliaSim.

2.2. ENTORNO DE SIMULACIÓN PYTHON PARA EL ROBOT NAO

2.2.1. COMANDOS DE EJECUCIÓN SDK NAOQI PARA EL ROBOT NAO

El software de Python sirve como un intermediario entre CoppeliaSim y Matlab/Simulink ya que este tomará los datos de ambos programas y los enviará de una plataforma a otra, aparte de efectuar una acción de intermediario de datos Python tiene una función sumamente importante y es que este programa permite conectar las librerías que posee Choregraphe a Python mediante la importación del SDK de NAOqi, esto es sumamente útil ya que Choregraphe puede ejecutar funciones ya sea de caminar, voz, audio, posturas de una manera sumamente fácil, al tener todos estos comandos en Python permite enviar los datos del controlador y hacer que se ejecuten mediante NAOqi al Robot Nao.

Para poder tomar estas funcionalidades es importante descargar la librería de NAOqi con ella en ejecución lo que se hace es crear un Proxy mediante la dirección IP y número de puerto del robot real o de Choregraphe y dar un atributo del proxy creando a una función de Python, como se puede ver en **Figura 2. 9** se toma dos comandos de proxy el primero contiene funciones de movimiento y el segundo funciones de posturas.

```
movProxy = ALProxy("ALMotion", NaoIP, NaoPort)
postureProxy = ALProxy("ALRobotPosture", NaoIP, NaoPort)
```

Figura 2. 9 Declaración del Proxy de movimiento y posición de NAOqi.

Definido el atributo del proxy como se mostró previamente Python será el encargado de enviar los datos del controlador desarrollado es por ello que se ocupa el proxy de move que permite enviar datos de velocidad en m/s, se puede enviar los datos de dos formas

una es enviar las velocidades lineales pero en coordenadas (x, y) con su respectiva velocidad angular y la segunda es enviar el módulo de la velocidad lineal en este caso ucont y la velocidad angular wcont como se muestra en la **Figura 2. 10** estas dos formas de envió dependerá de cómo sean los datos de salida del controlador, también se debe tener en cuenta que hay otro comando de moviendo el cual es el moveTo este a diferencia del otro recibirá los datos ya en metros y permitirá mover al robot a una posición específica que desee el usuario sin la necesidad de elaborar otro controlador ya que este efectuará el propio control PID que posee el robot por default por lo cual para probar el control de Lyapunov u otro control se debe efectuar la acción con el comando move.

```
movProxy.move(ucont, 0, wcont) # ENVIA LOS DATOS DE VELOCIDAD EN m/s
```

Figura 2. 10 Comandos de NAOqi para efectuar las órdenes del controlador.

2.3. ENTORNO DE SIMULACIÓN DE MATLAB-SIMULINK

2.3.1. COMUNICACIÓN DE DATOS ENTRE SIMULINK-PYTHON PARA EL ROBOT NAO

El controlador se desarrolla en Simulink por lo cual es necesario enviar los datos de este a Python para ello se ocupa los bloques de comunicación UDP, los datos a enviar son velocidad lineal, velocidad angular y el error, estos se empaquetan en un array y dichos valores se discretizan a un tiempo de muestreo de 0.02s que es el tiempo de respuesta del robot NAO, finalmente se envía los datos a la dirección IP y número de puerto específico como se ve en la **Figura 2. 11**.

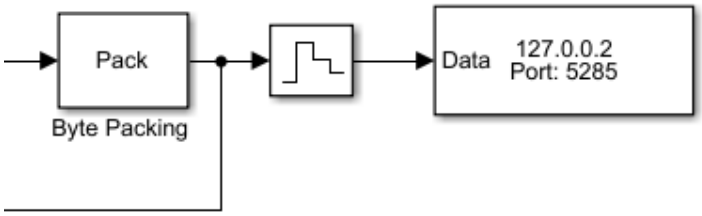

Figura 2. 11 Transmisión de datos de Simulink a Python.

Python enviará los datos de posición actual del robot que a su vez los tomó de CoppeliaSim para ello se desempaqueta el bloque de datos en un array de tres elementos posición en (x, y) y ángulo de orientación como se aprecia en **Figura 2. 12**.

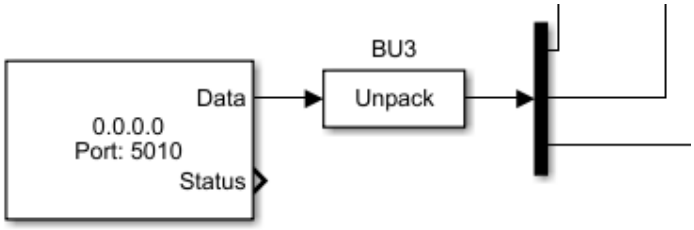

Figura 2. 12 Recepción de datos de Python a Simulink.

Para el caso de la programación en Python simplemente se declara la comunicación UDP con la dirección IP y número de puerto especificados en Simulink y se podra tomar los datos de Simulink como se ve en **Figura 2. 13**.

```
#COMUNICACIÓN-RECEPCÓN DE DATOS DE MATLAB-SYMULINK A PYTHON
UDP_IP = "127.0.0.2"
UDP_PORT = 5285
sock = socket.socket(socket.AF_INET,    # Internet
                     socket.SOCK_DGRAM) # UDP -- # Create Datagram Socket

sock.bind((UDP_IP, UDP_PORT))
Controlador, addr = sock.recvfrom(1024) # buffer size is 1024 bytes
Control=unpack('>ddd', Controlador)
```

Figura 2. 13 Recepción de datos de Simulink a Python.

En el caso del envío de los datos de posición de Python a Simulink el trabajo es idéntico a la recepción se declara la dirección IP y número de puerto especifico igual que el bloque UDP de Simulink y se procede a enviar los datos como se muestra en **Figura 2. 14**.

```
#COMUNICACIÓN-TRANSMISIÓN DE DATOS DE MATLAB-SYMULINK A PYTHON
Realimentacion=[pos_x,pos_y,gamma_real]
mensajebody=pack('>ddd',*Realimentacion)
sock.sendto (mensajebody, ("127.0.0.2",5010))
```

Figura 2. 14 Transmisión de datos de Python a Simulink.

2.4. DESARROLLO DEL CONTROLADOR DE POSICIÓN

2.4.1. CONTROL DE POSICIÓN BASADO EN LYAPUNOV

Para el desarrollo del controlador se debe de tomar en cuenta al robot NAO como una partícula móvil en el cual su dinámica y cinemática se encuentran en las funciones propias de las librerías de NAOqi es decir se asume que el mismo es una caja negra, es por ello que se puede desarrollar el controlador de posición basándose en el modelo más básico y frecuentemente usado para el desarrollo de algoritmos de control como lo es un robot

uniciclo con dicha cinemática se puede lograr desarrollar el controlador de posición que sea funcional para el robot NAO.

Al asumir que el mismo es una caja negra y una partícula móvil las ecuaciones que describen el movimiento del robot se las puede deducir a partir de la **Figura 2. 15** las cuales muestran la velocidad lineal en coordenadas cartesianas y la velocidad angular como se aprecia en (2. 6), (2. 7) y (2. 8).

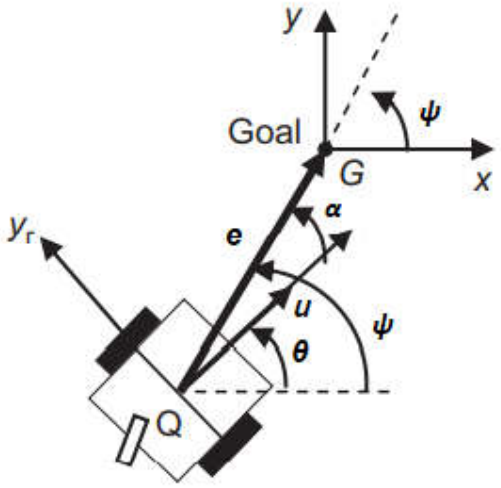

Figura 2. 15 Cinemática robot uniciclo.

$$\dot{x} = u\cos(\theta) \tag{2. 6}$$

$$\dot{y} = u\sin(\theta) \tag{2. 7}$$

$$\dot{\phi} = w \tag{2. 8}$$

Para el desarrollo del controlador se pasa la cinemática del robot a coordenadas polares las cuales quedan expresadas en módulo del error, ángulo de orientación, velocidad lineal y velocidad angular como se aprecia en (2. 9), (2. 10) y (2. 11).

$$\dot{e} = -u\cos(\alpha) \tag{2. 9}$$

$$\dot{\alpha} = -w + \left(\frac{u}{e}\right) \sin(\alpha) \tag{2. 10}$$

$$\dot{\psi} = \left(\frac{u}{e}\right) \sin(\alpha) \tag{2. 11}$$

Una vez descrito el modelo cinemático del sistema se puede empezar a elaborar el diseño del controlador de Lyapunov para ello lo primero que se debe seleccionar es la función candidata o función de Lyapunov que permite llevar a la estabilidad el sistema por ello se selecciona la función (2. 12), con dicha función se procede a comprobar que las tres primeras reglas de Lyapunov se cumplan.

La primera ley se cumple al tener una función polinómica de grado 2 la cual a su vez está basada en sin y cos los cuales son funciones continuas en todos sus puntos como se ve en (2. 12).

La segunda ley se cumple al evaluar x = 0 en (2. 12) lo que daría como resultado que nuestra función sea cero siempre que x sea cero.

La tercera ley se cumple por el hecho de que la función candidata de Lyapunov sea un polinomio de grado 2 es decir x al cuadrado, con esto se asegura de que la función (2. 12) evaluada en cualquier x distinta de cero sea siempre positiva

Como el controlador a ser diseñado es un control de posición x estará en función de (2. 9) y (2. 10) y se puede observar en (2. 14) es decir el usuario ingresa una coordenada específica en (x, y) y el robot tratará de llevar el módulo del error o distancia recorrida al punto deseado juntamente con el ángulo que produce dicho desplazamiento a cero.

$$v(x) = \frac{1}{2}\, x^T\, Q\, x \tag{2. 12}$$

$$Q = \begin{bmatrix} q1 & 0 \\ 0 & 1 \end{bmatrix} \; donde\; q1 > 0 \tag{2. 13}$$

$$x = [e, \alpha]^T \tag{2. 14}$$

Comprobado el cumplimiento de las tres primeras leyes de Lyapunov se procede a comprobar la cuarta ley la cual dice que la derivada de la función candidata de Lyapunov debe ser menor o igual a cero, se desarrolla la función (2. 15) y queda (2. 18) la cual se divide en dos funciones v1 prima y v2 prima para facilitar el cumplimiento de dicha ley.

$$\dot{v(x)} = x^T \, Q \, \dot{x} \tag{2. 15}$$

$$\dot{v(x)} = [e \; \alpha] \begin{bmatrix} q1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{e} \\ \dot{\alpha} \end{bmatrix} \tag{2. 16}$$

$$\dot{v(x)} = [q1 \; e \quad \alpha] \begin{bmatrix} \dot{e} \\ \dot{\alpha} \end{bmatrix} \tag{2. 17}$$

$$\dot{v(x)} = q1 \; e \; \dot{e} + \; \alpha \; \dot{\alpha} \tag{2. 18}$$

$$\dot{v(x)} = \; \dot{v1} + \; \dot{v2} \tag{2. 19}$$

Se toma la primera función (2. 20) y se desarrolla dicha función teniendo en cuenta que u y w serán las variables para controlar y las que harán que el sistema se estabilice, para v1 prima se tiene que hacer que u cambie a la función de tal manera que esta sea menor o igual a cero por lo que al encontrar dicho cambio u queda como (2. 24), se debe de recordar que en este paso se pueden tener múltiples desarrollos y el desarrollo que se propone no es el único que puede hacer que se estabilice la función.

$$\dot{v1} = q1 \; e \; \dot{e} = -q1 \; e \; u \cos(\alpha) \tag{2. 20}$$

La expresión ((2. 21) es el resultado de haber dividido la función candidata de Lyapunov en dos partes como se muestra en (2. 18) y (2. 19) el valor de Alpha prima se saca de (2. 10) y la expresión (2. 23) muestra el desarrollo de esta para luego ocupar dicha solución.

$$\dot{v2} = \alpha\, \dot{\alpha} \tag{2. 21}$$

$$\dot{v2} = -\alpha\, w + \alpha \left(\frac{u}{e}\right) \sin(\alpha) \tag{2. 22}$$

$$\dot{v2} = \alpha\, [-w + \left(\frac{u}{e}\right) \sin(\alpha)] \tag{2. 23}$$

$$u = uc = k1\, e \cos(\alpha)\; donde\; k1 > 0 \tag{2. 24}$$

Encontrado el módulo u de la velocidad lineal se procede a comprobar que v1 prima cumpla con el requerimiento como se aprecia en (2. 25) tener en cuenta que las constantes k1 siempre serán mayores y estas serán probadas en el controlador para ver cual estabiliza mejor el sistema.

$$\dot{v1} = -k1\, q1\, \mathrm{e}^2\, \cos^2(\alpha)\; donde\; q1 > 0\; y\; k1 > 0 \tag{2. 25}$$

Para el caso de v2 prima se realiza lo mismo partiendo de (2. 23) se desarrolla (2. 26) pero en este caso se debe encontrar la velocidad angular w que hace que el sistema cumpla la cuarta ley como se muestra en (2. 27) de la misma manera k2 será una constante siempre positiva para asegurar el cumplimiento de la condición y se verifica en (2. 28).

$$\dot{v2} = \alpha\, [-w + k1\, \cos(\alpha)\, \sin(\alpha)] \tag{2. 26}$$

$$w = wc = k2\, \alpha + k1\, \cos(\alpha)\, \sin(\alpha)\; donde\; k1 > 0\; y\; k2 > 0 \tag{2. 27}$$

$$\dot{v2} = -k2\, \alpha^2 \tag{2. 28}$$

Finalmente se comprueba los controladores elaborados en base de u, w en (2. 29) y se puede apreciar que cumple con la cuarta ley de Lyapunov por lo cual el controlador está listo para su prueba.

$$\dot{v(x)} = -k1\ q1\ \mathrm{e}^2\ \cos^2(\alpha) - k2\ \alpha^2\ \leq\ 0 \qquad (2.\ 29)$$

Tener en cuenta que cuando se simule el controlador de Lyapunov (u, w) en Simulink para encontrar el módulo del error (e) se debe sacar el módulo entre las coordenadas deseadas (xd, yd) o las que ingresa el usuario menos las coordenadas reales (xr, yr) o las que provienen de la realimentación de los sensores del Robot NAO en CoppeliaSim como se muestra en (2. 30).

El ángulo Alpha (α) viene dado por el mismo principio el ángulo deseado (αd) menos el ángulo real (αr), en el ángulo deseado (αd) se ocupa el comando atan2 ya que este comando permite sacar el ángulo de la arco tangente referido siempre al eje x es decir de 0 a 180 grados en sentido antihorario y de 0 a -180 grados en sentido horario, el ángulo real (αr) viene a ser el ángulo que proviene de los sensores del Robot Nao o el dato de realimentación de la orientación del robot como se puede ver en (2. 31) y (2. 32).

$$e = \sqrt{(xd - xr)^2 + (yd - yr)^2} \qquad (2.\ 30)$$

$$\alpha =\ \alpha d - \alpha r \qquad (2.\ 31)$$

$$\alpha =\ \mathrm{atan2}\left(\frac{yd - yr}{xd - xr}\right) - \alpha r \qquad (2.\ 32)$$

2.4.2. SIMULACIÓN DEL CONTROLADOR MEDIANTE EL ROBOT UNICICLO

Como se ha venido mencionando con anterioridad al robot se toma como una caja negra o una partícula en movimiento por esto se utiliza el modelo cinemático del robot uniciclo ya que este es el más sencillo y utilizado para desarrollar algoritmos de control que sean funcionales y relativamente fáciles de implementar.

El controlador que se desarrolle se recomienda simular, primero con el modelo cinemático del robot uniciclo ya que si este es funcional también lo será en el robot NAO cualquier control se puede hacer una pre-simulación con este modelo y luego corregir errores o fallas de este.

EL modelo uniciclo resulta mucho más fácil de simular y consumirá menos recursos del computador, recordar que una vez que el controlador sea funcional con este modelo se

puede probar con seguridad en el robot NAO y hacer las pruebas y modificaciones necesarias que requiera el controlador.

Como se puede apreciar en la **Figura 2. 16** se toma las referencias o coordenadas deseadas y se resta con las coordenadas reales o de la realimentación para sacar el error, este se pasa a coordenadas polares mediante (2. 30) y (2. 32) para que el controlador desarrollado de u (2. 24) y w (2. 27) pueda efectuar su acción de control y finalmente esta ira al modelo cinemático del robot que se expresa con (2. 6), (2. 7) y (2. 8).

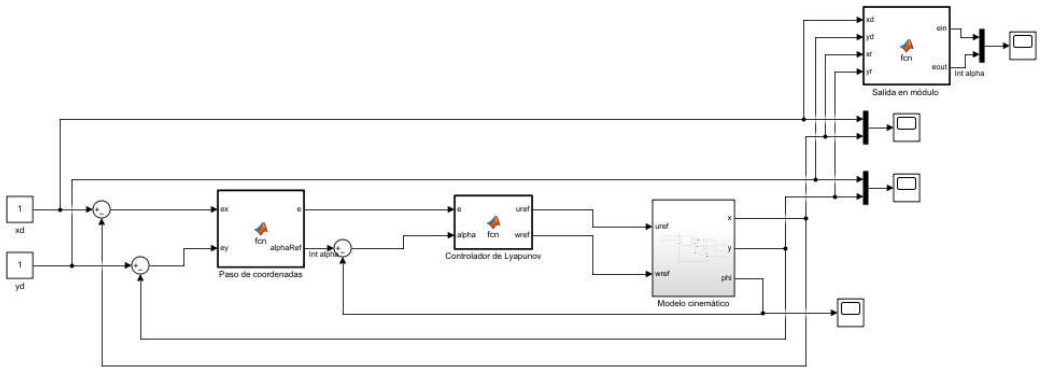

Figura 2. 16 Simulación del controlador mediante el modelo cinemático uniciclo.

Para probar el controlador se ha puesto sus constantes de k1 y k2 en un valor de 0.5 y la posición tanto en x como y en 1, podemos apreciar en la **Figura 2. 17** y **Figura 2. 18** que el controlador dirige adecuadamente a la posición en (x, y) a 1 sin tener sobre picos y con un tiempo de establecimiento de 12s.

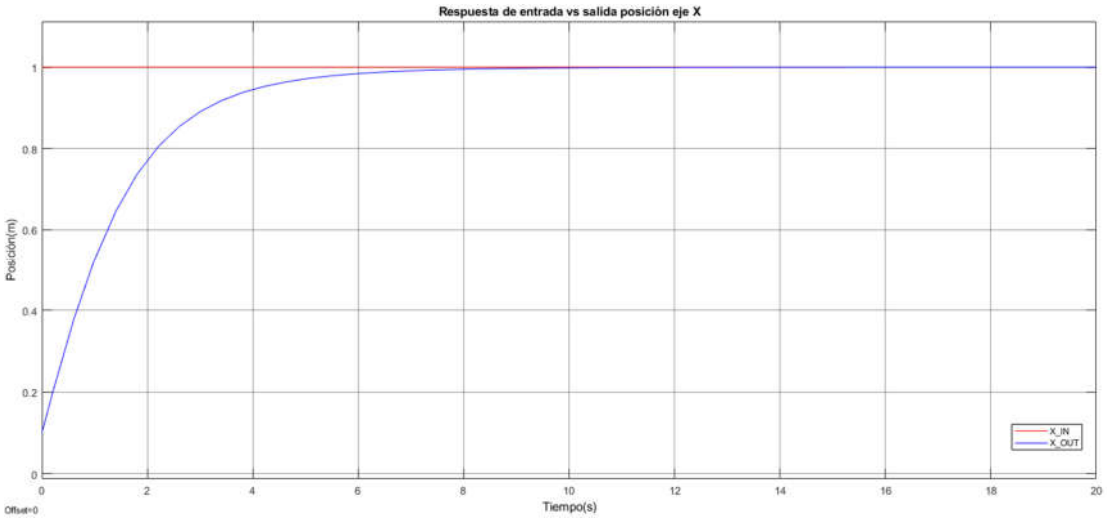

Figura 2. 17 Respuesta de la entrada vs la salida de la posición X

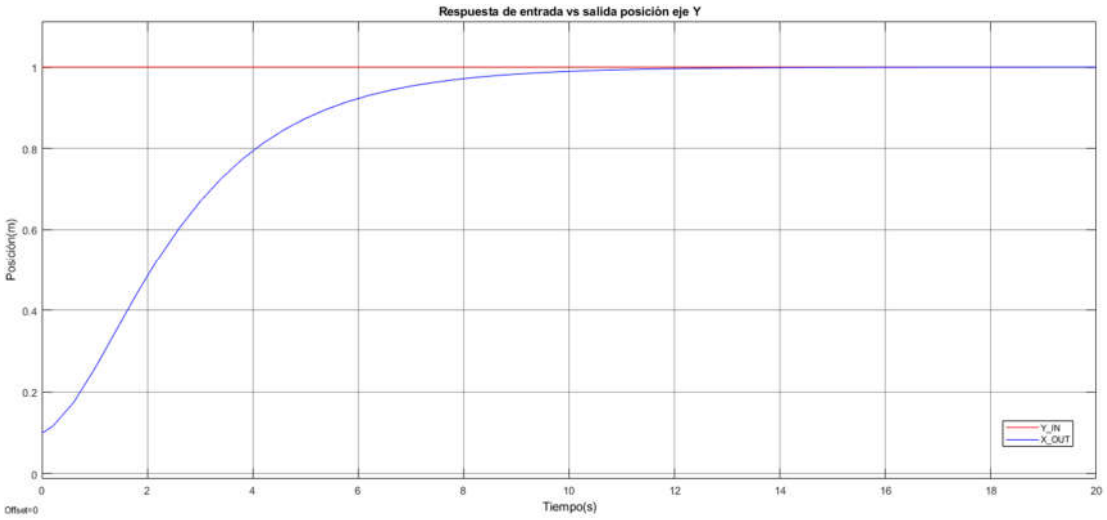

Figura 2. 18 Respuesta de la entrada vs la salida de la posición Y.

En la **Figura 2. 19** se puede apreciar el comportamiento del sistema, pero en módulo, se observa que el mismo tiende a recorrer la distancia requerida sin sobre picos y con un tiempo de establecimiento de 12s, este tiempo se da debido a la propia naturaleza del algoritmo de control a la hora de encender y ejecutar sus actuadores.

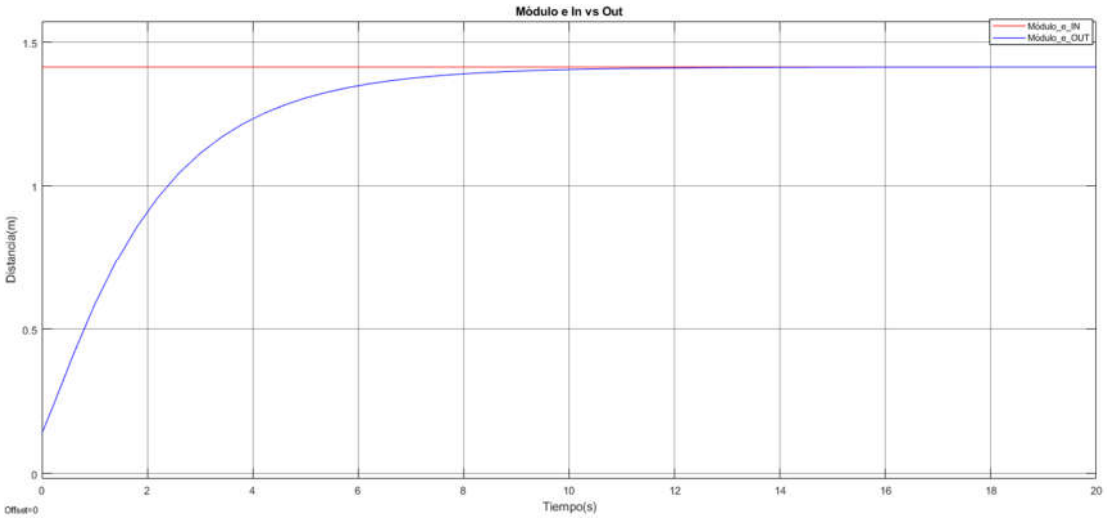

Figura 2. 19 Respuesta de la entrada vs la salida del módulo de la distancia e.

2.4.3. SIMULACIÓN DEL CONTROLADOR EN EL ROBOT

Desarrollado el controlador de posición y probado el funcionamiento de este en el modelo cinemático del robot uniciclo se procede a implementarlo en el robot NAO como se puede ver en la **Figura 2. 20** las señales de control son enviadas al módulo de NAOqi de Python y este a su vez ejecutara la acción de este en el modelo que contienen estas librerías con CoppeliaSim, el lazo de realimentación proviene de dichos módulos y da la posición y

orientación del robot para luego calcular el error y pasarlas a coordenadas polares, para que de esta manera el controlador efectué su cálculo, el esquema propuesto se puede ver en Simulink en la **Figura 2. 21**.

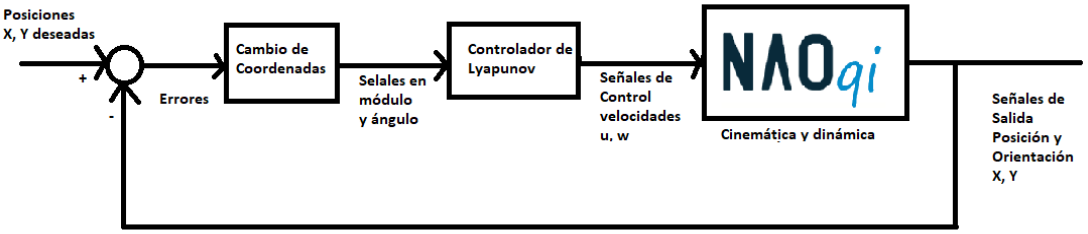

Figura 2. 20 Esquema en lazo cerrado.

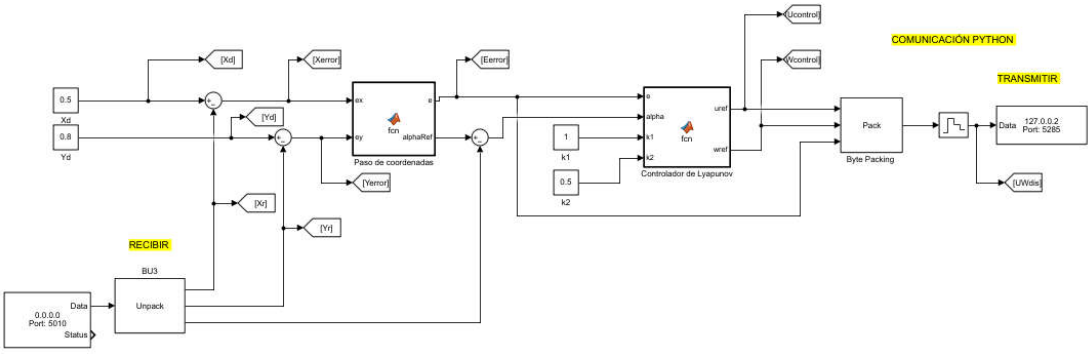

Figura 2. 21 Esquema en lazo cerrado en Simulink.

Se puede ver en la **Figura 2. 22**, **Figura 2. 23** y **Figura 2. 24** la respuesta del controlador desarrollado en el robot NAO en este caso al dirigirlo a la posición (x, y) de (1, 1) con constantes del controlador de k1 y k2 de 0.5, se puede apreciar que su comportamiento es adecuado, tanto en la **Figura 2. 22** y **Figura 2. 23** se observa cual es la respuesta de salida en (x, y) y ambas siguen a la referencia mientras que **Figura 2. 24** se puede ver el resultado, pero con el módulo de esta manera se puede apreciar a la vez el comportamiento de (x, y).

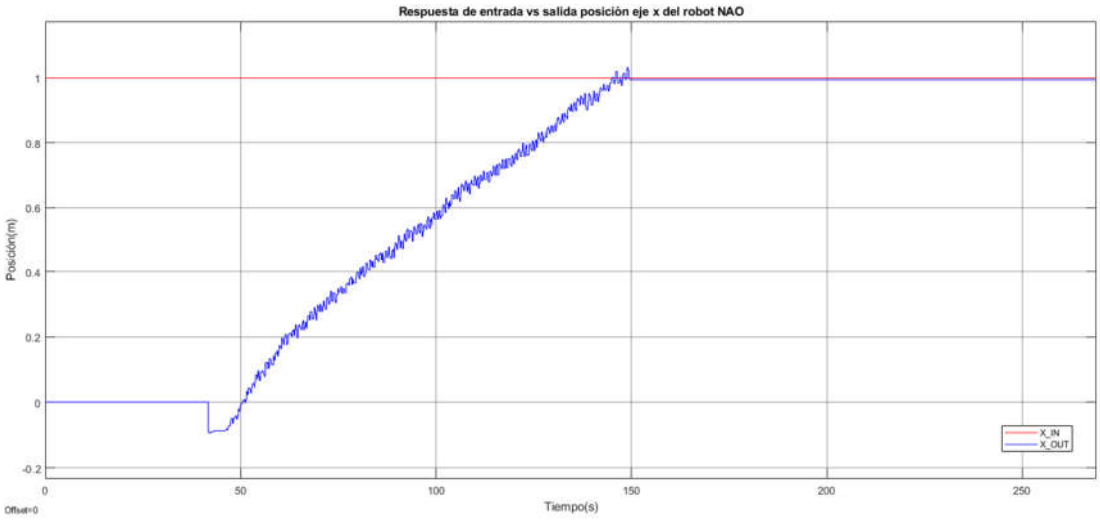

Figura 2. 22 Respuesta de la simulación de entrada vs salida en el eje X del robot NAO.

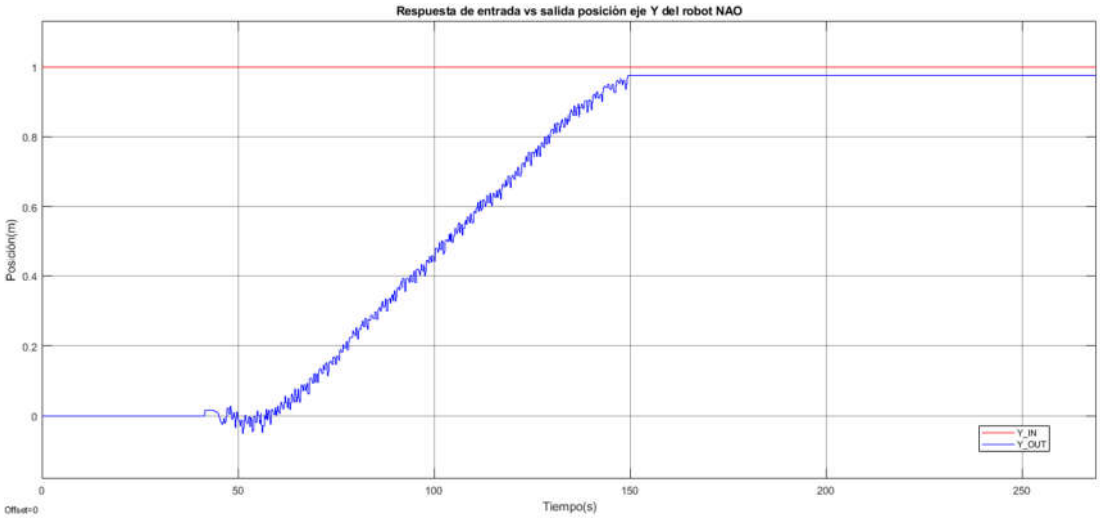

Figura 2. 23 Respuesta de la simulación de entrada vs salida en el eje Y del robot NAO.

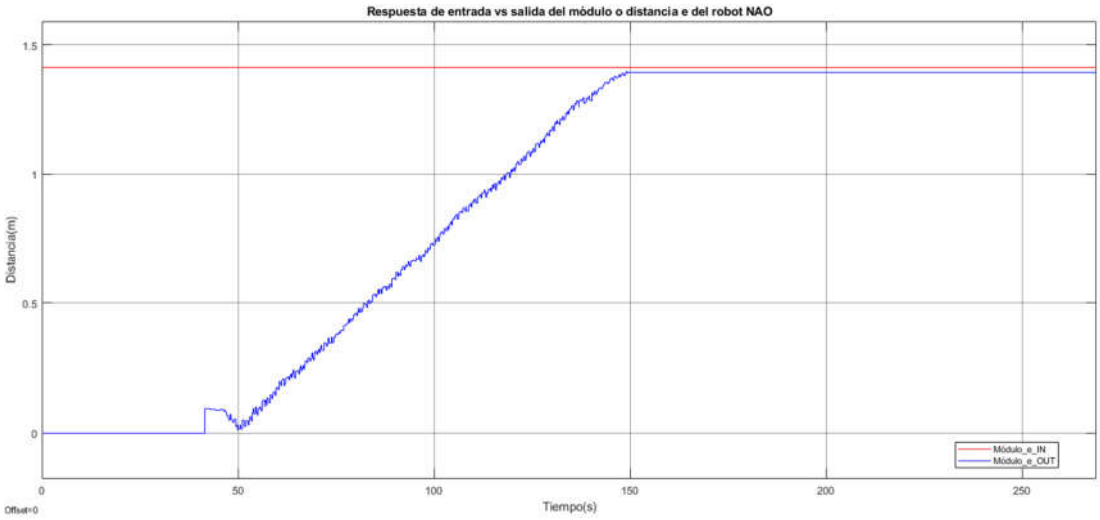

Figura 2. 24 Respuesta de la simulación de entrada vs salida del módulo e del robot NAO.

Se debe considerar que el módulo del error tiende a ser cero cuando el Robot NAO llegue a la posición deseada, eso se puede apreciar en las **Figura 2. 25** y **Figura 2. 26** tanto el error de X como el de Y tienden a ser cero cuando la posición deseada llega a la referencia mientras que en la **Figura 2. 27** se puede apreciar lo mismo pero expresado en módulo.

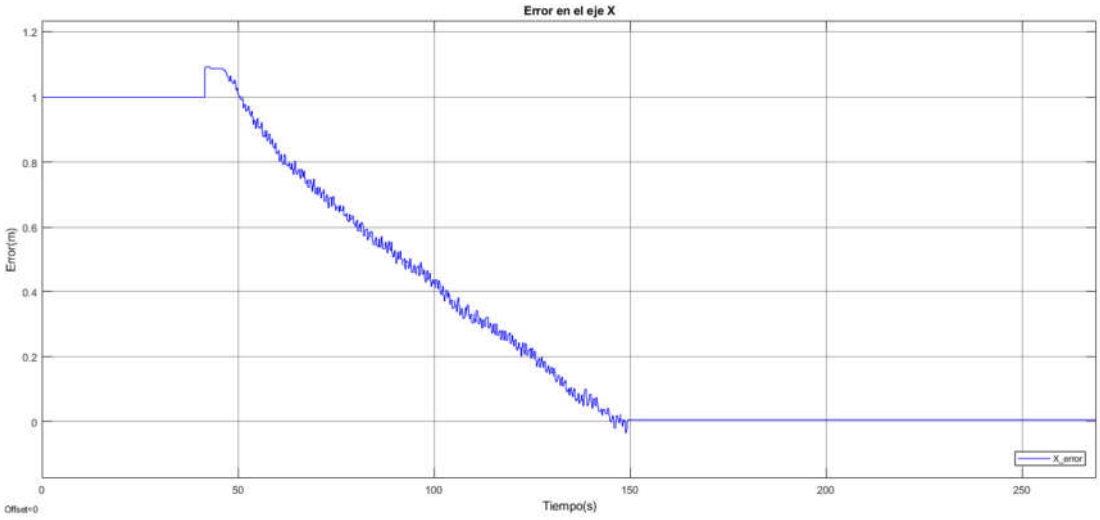

Figura 2. 25 Respuesta de la simulación error en el eje X del robot NAO.

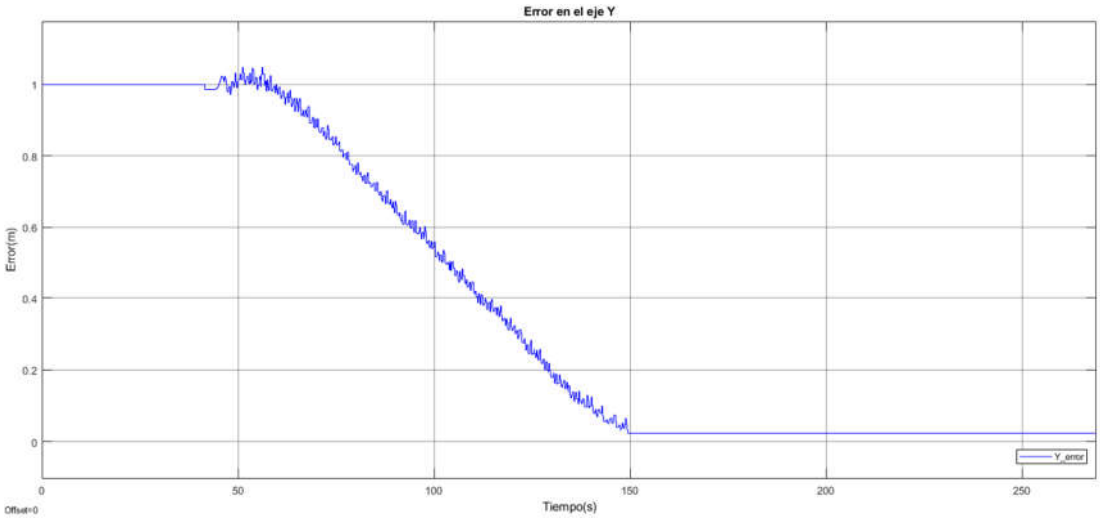

Figura 2. 26 Respuesta de la simulación error en el eje Y del robot NAO.

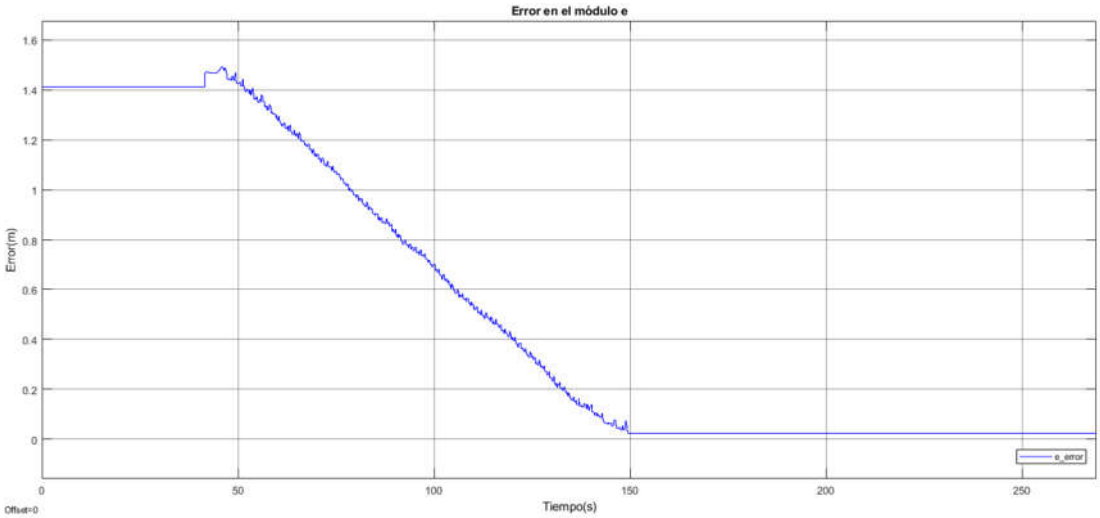

Figura 2. 27 Respuesta de la simulación error en el módulo e del robot NAO.

Se puede apreciar en las **Figura 2. 28** y **Figura 2. 29** las señales de las acciones de control tanto de la velocidad lineal u como la velocidad angular w, estas señales son las que se enviará al módulo de NAOqi de Python juntamente con CoppeliaSim para que el modelo del robot NAO que poseen estos programas realicen las acciones de control, se puede observar que las acciones de control también tienden a cero conforme el robot llega a su posición deseada.

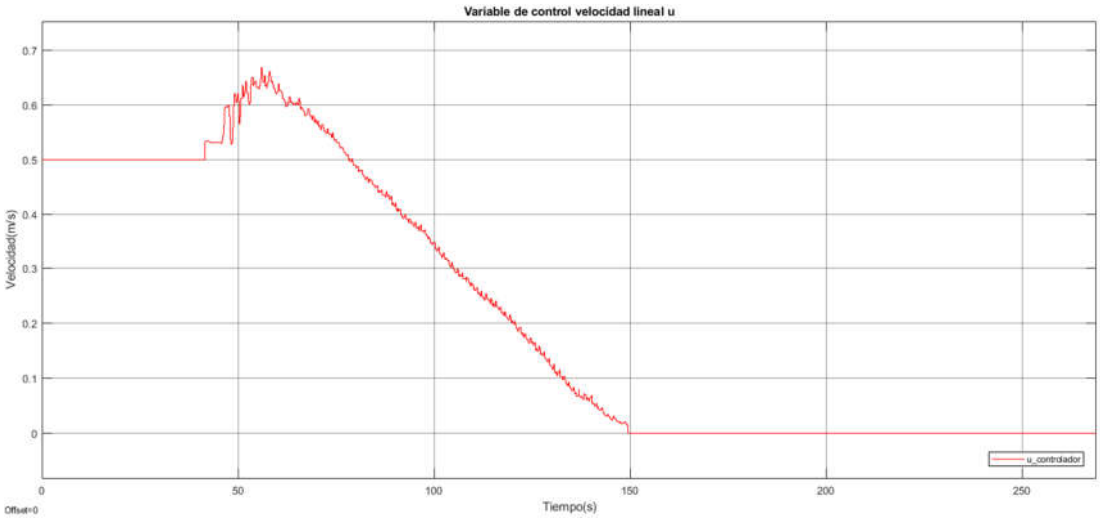

Figura 2. 28 Respuesta de la simulación de la variable de control velocidad lineal u del robot NAO.

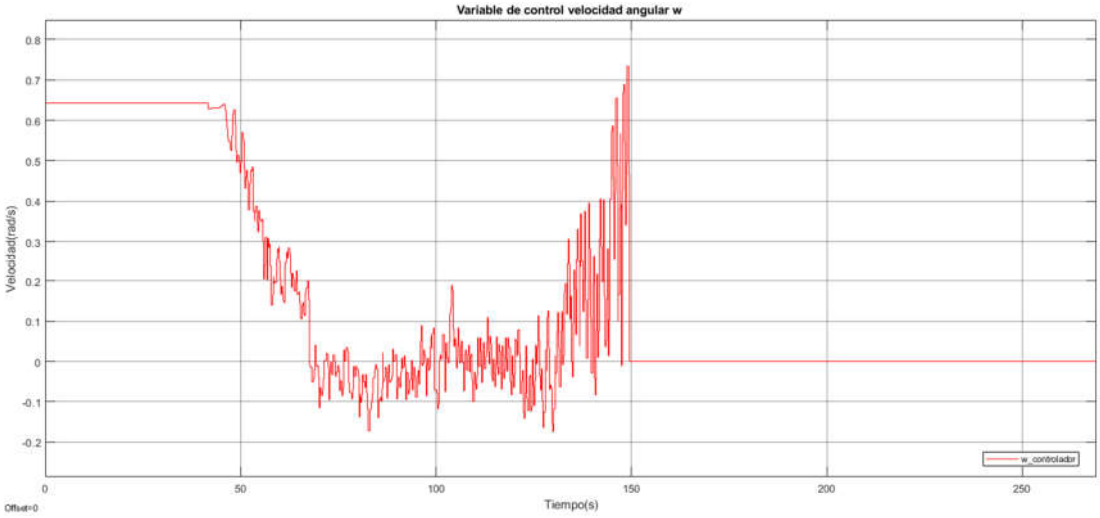

Figura 2. 29 Respuesta de la simulación de la variable de control velocidad angular w del robot NAO.

En la **Figura 2. 30** se puede apreciar el movimiento que realizó el robot desde su posición inicial (x, y) de (0, 0) hasta su posición deseada de (1, 1) dicho movimiento dio como resultado las gráficas que se expusieron con anterioridad.

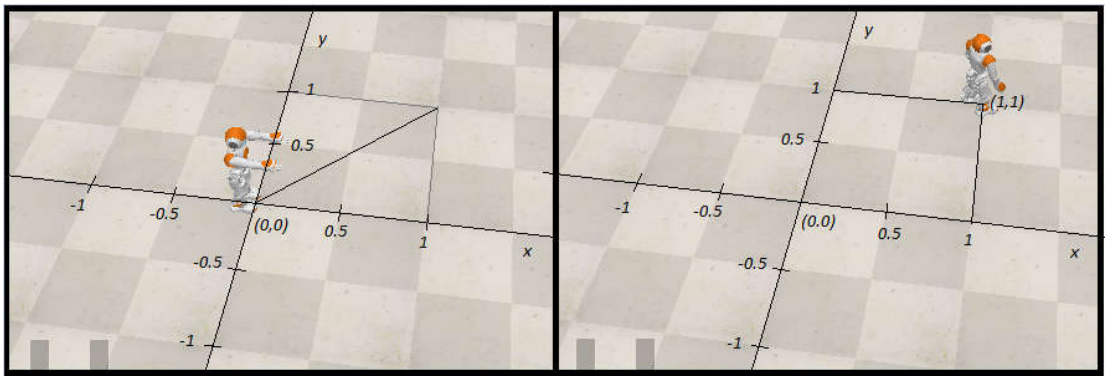

Figura 2. 30 Movimiento efectuado por el robot NAO a la posición (1, 1).

2.5. INTERFAZ GRÁFICA

La interfaz de usuario tendrá como objetivo controlar los comandos y parámetros de Simulink que necesita el controlador del robot para su funcionamiento dentro de una presentación más amistosa, la interfaz se clasifica en tres pestañas la primera que se observa en Figura 2. 31 sería la presentación o portada, la segunda pantalla muestra las instrucciones para iniciar, detener y volver a iniciar los programas como se ve en la Figura 2. 32 y finalmente la tercera pestaña como se ve en la Figura 2. 33 permite controlar los parámetros de Simulink en los cuales se puede ingresar la posición deseada y los valores del controlador a modo de control remoto para poder ver su desarrollo en CoppeliaSim.

Figura 2. 31 Interfaz gráfica pantalla principal.

Figura 2. 32 Interfaz gráfica pantalla de instrucciones.

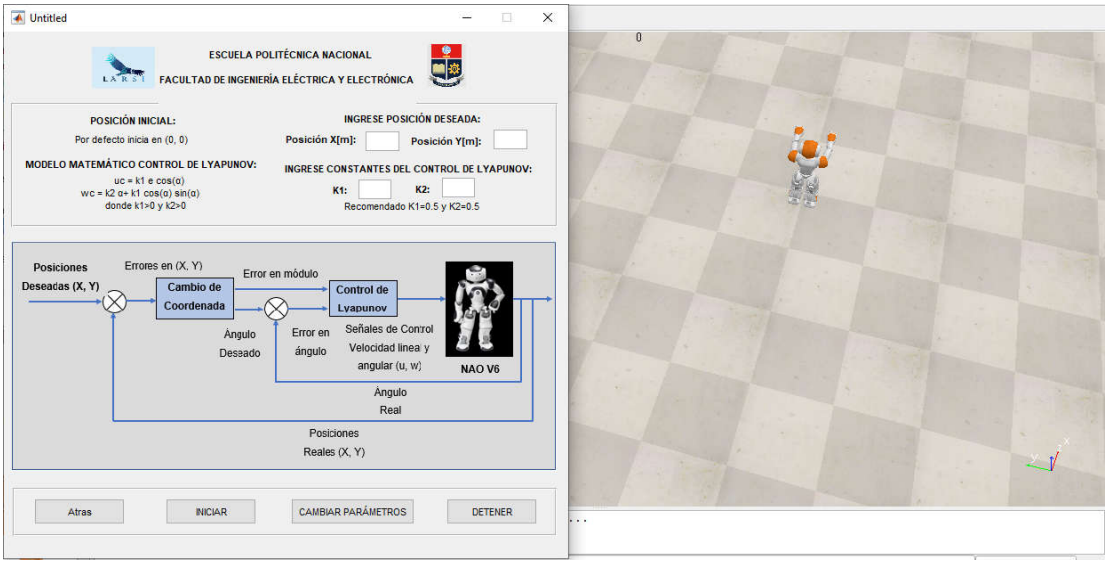

Figura 2. 33 Interfaz gráfica pantalla de control de Simulink.

2.6. LÓGICA DE PROGRAMACIÓN

2.6.1. DIAGRAMA DE FLUJO DE LA ESTRUCTURA DEL PROGRAMA

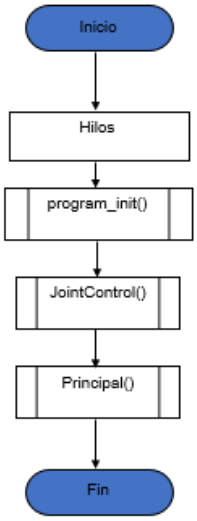

Figura 2. 34 Diagrama de flujo de la estructura del programa.

2.6.2. DIAGRAMA DE FLUJO DEL PROGRAMA PROGRAM_INIT

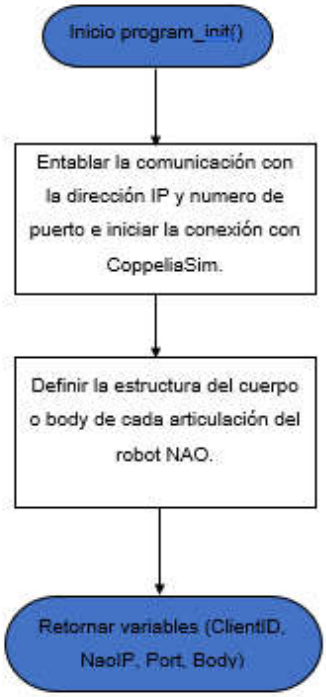

Figura 2. 35 Diagrama de flujo de la estructura del programa program_init.

2.6.3. DIAGRAMA DE FLUJO DEL PROGRAMA JOINTCONTROL

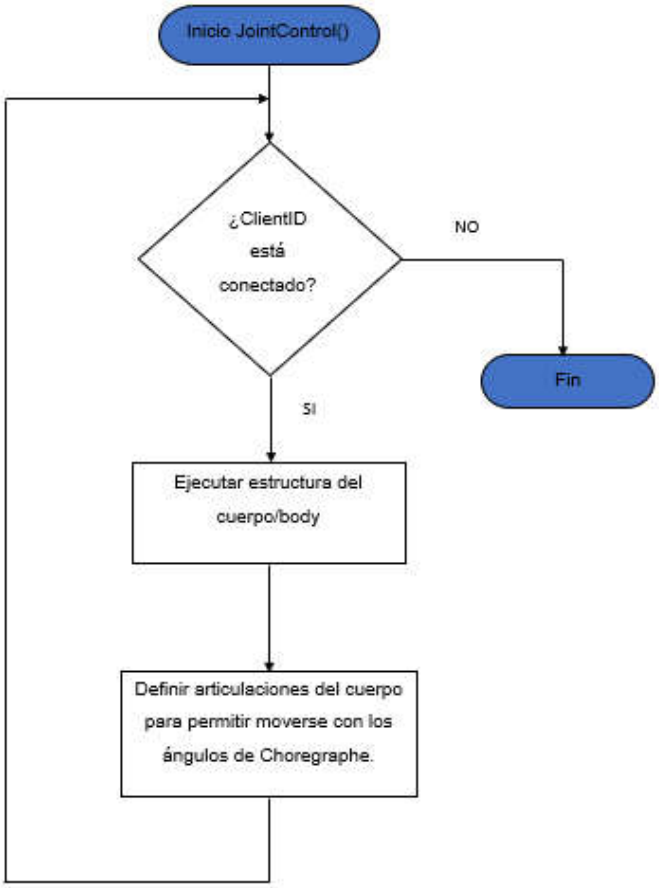

Figura 2. 36 Diagrama de flujo de la estructura del programa jointcontrol.

2.6.4. DIAGRAMA DE FLUJO DEL PROGRAMA PRINCIPAL

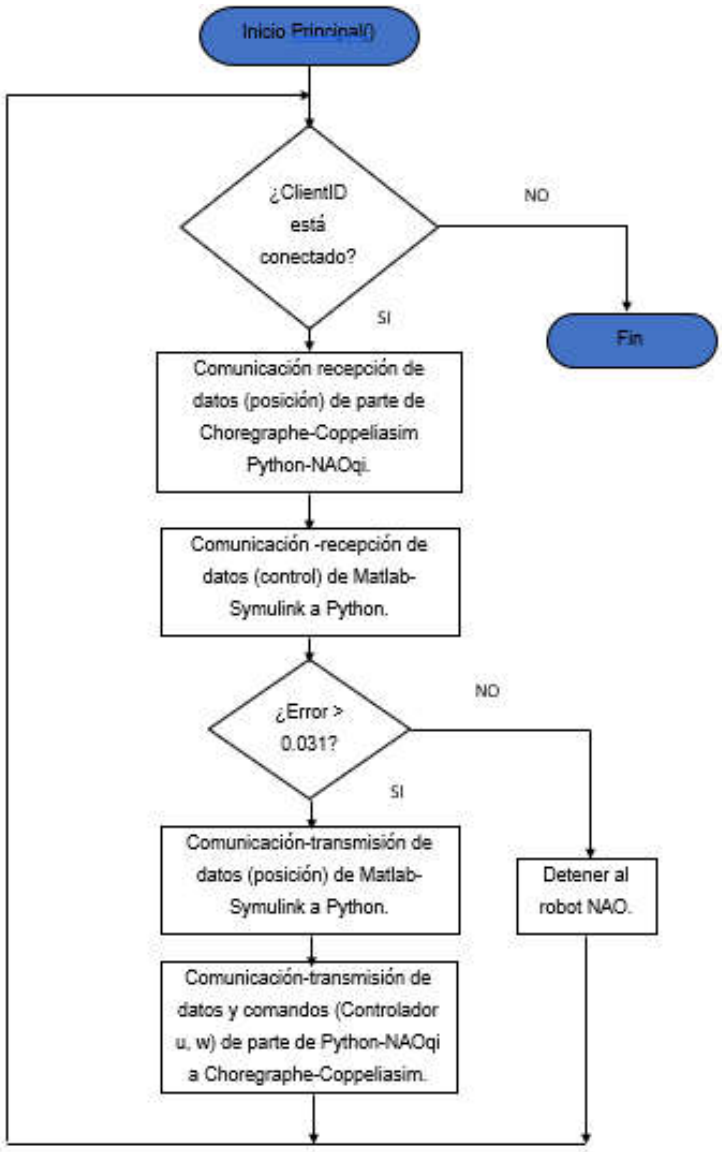

Figura 2. 37 Diagrama de flujo de la estructura del programa principal.

2.6.5. DIAGRAMA DE FLUJO DEL CONTROL BASADO EN LYAPUNOV

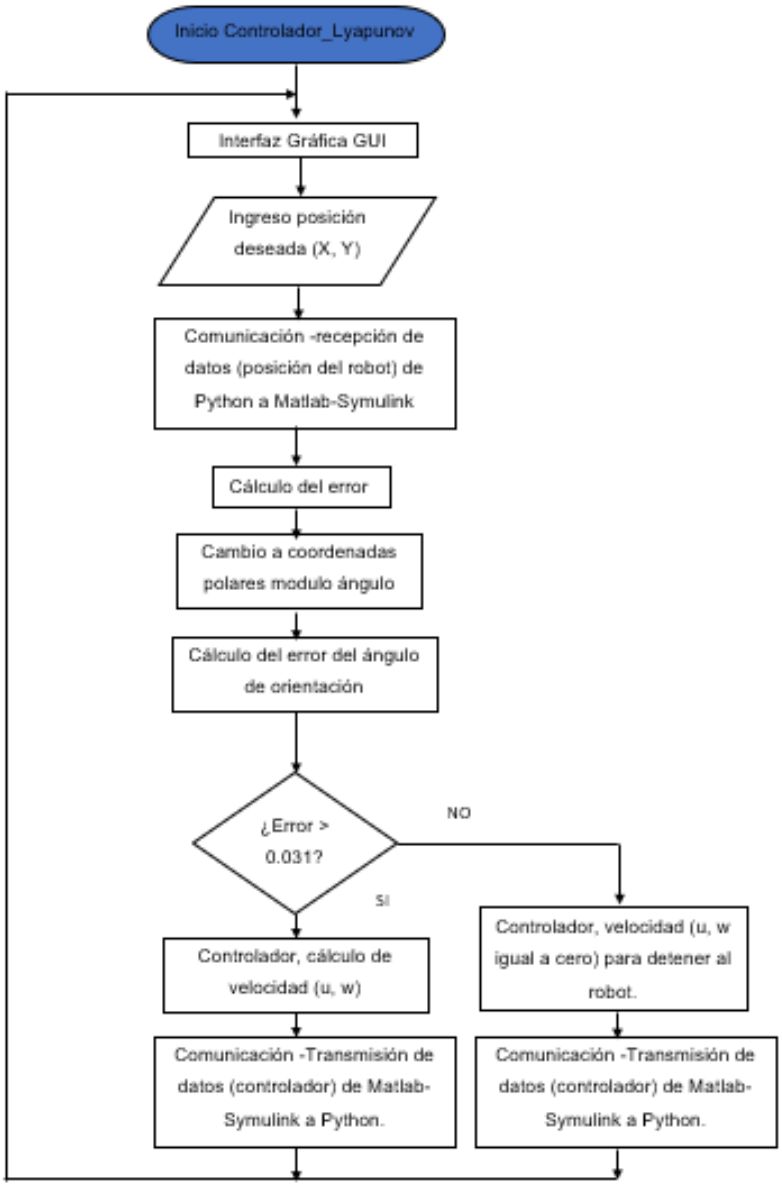

Figura 2. 38 Diagrama de flujo de la estructura del controlador basado en Lyapunov.

3. RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

3.1. RESULTADOS

3.1.1. ÍNDICES DE DESEMPEÑO Y CONSTANTES DEL CONTROLADOR

Se puede apreciar en la Tabla 3. 1 cómo se comporta el controlador mediante los índices de desempeño ISE e ISU y como estos cambian mediante la variación de la constante K1 del controlador de Lyapunov mientras que la constante K2 se mantiene constante, se puede observar que entre menor sea el valor de la constante K1 su tiempo de establecimiento y tiempo de simulación será mayor de la misma manera el índice del error ISE y el índice de la acción de control ISU son los más elevados recordando que entre menor sean los valores de estos índices mejor será el comportamiento del controlador, al ir subiendo el valor K1 se puede ver que estos parámetros mejoran disminuyendo su valor dándonos los mejores resultados en K1 de 0.5 y 0.7, valores superiores a este empiezan nuevamente a distorsionar la respuesta.

Tabla 3. 1 Índices de desempeño con variación de la constante K1 del controlador.

K2 = 0.5										
K1	**0.1**		**0.3**		**0.5**		**0.7**		**0.9**	
Índices	**ISE**	**ISU**	**ISE**	**ISU**	**ISE**	**ISU**	**ISE**	**ISU**	**ISE**	**ISU**
Eje X	105.3	279.4	71.15	124.1	84.03	82.19	59.37	86.97	72.34	88.14
Eje Y	85.97	379.9	96.98	92.77	83.08	77.41	65.59	80.97	81.91	86.68
Módulo E	191.3	659.3	168.1	216.8	167.1	159.6	125	167.9	154.2	174.8
Ts	414.26(s)		174.61(s)		154.42(s)		126.13(s)		149.23(s)	
Tiempo de simulación	466.90(s)		228.96(s)		198.52(s)		178.54(s)		202.50(s)	

En la **Tabla 3. 2** se puede apreciar cómo se comporta el controlador pero esta vez variando la constante K2 y K1 siendo constante de la misma manera que el anterior análisis entre

menor sea el valor de K2 tanto los índices de desempeño ISE e ISU como el tiempo de establecimiento son mucho más elevados que cuando el K2 es mayor, los mejores resultados se pueden apreciar cuando K2 tiene un valor de 0.5 y 0.7 especialmente el primero ya que este obtiene el ISE e ISU menor a todas las variaciones, lo que demuestra que la acción de control y el error que el controlador presenta es el más óptimo que los demás valores por ello poner un valor de K1 y K2 de 0.5 en ambos puede ser una buena opción para que el robot muestre una buena respuesta.

Tabla 3. 2 Índices de desempeño con variación de la constante K2 del controlador.

K1 = 0.5										
K2	**0.1**		**0.3**		**0.5**		**0.7**		**0.9**	
Índices	**ISE**	**ISU**	**ISE**	**ISU**	**ISE**	**ISU**	**ISE**	**ISU**	**ISE**	**ISU**
Eje X	270.9	2425	71.45	317.4	64.19	91.53	67.53	98.4	89.68	115.7
Eje Y	108.5	1592	141.7	140.7	80.11	79.45	78.4	87.1	107.9	107.3
Módulo E	379.4	4017	213.2	458.1	144.3	171	145.9	185.5	197.6	223
Ts	1393.12(s)		256.02(s)		150.54(s)		145.26(s)		199.18(s)	
Tiempo de simulación	1441.82(s)		314.06(s)		196.18(s)		198.14(s)		255.14(s)	

3.1.2. RESPUESTA A MÚLTIPLES CAMBIOS DE REFERENCIA

Para observar el correcto funcionamiento del controlador desarrollado se probó el mismo ante múltiples cambios de referencia y es lo que se puede apreciar en la **Figura 3. 1** y **Figura 3. 2** para los ejes (x, y) y en la **Figura 3. 3** para el módulo o distancia recorrida, en este caso al robot se hizo que recorra una vuelta completa por todos los ejes del plano cartesiano, inicialmente estando en una posición inicial de (0, 0) para luego dirigirse a la posición de (1, 1) y así consecutivamente a las posiciones (0.5, -0.5), (-1, -1), (-0.5, 0.5) y (1, 1) se puede apreciar en las imágenes mencionadas que el controlador sigue a cada una de las referencias de manera correcta.

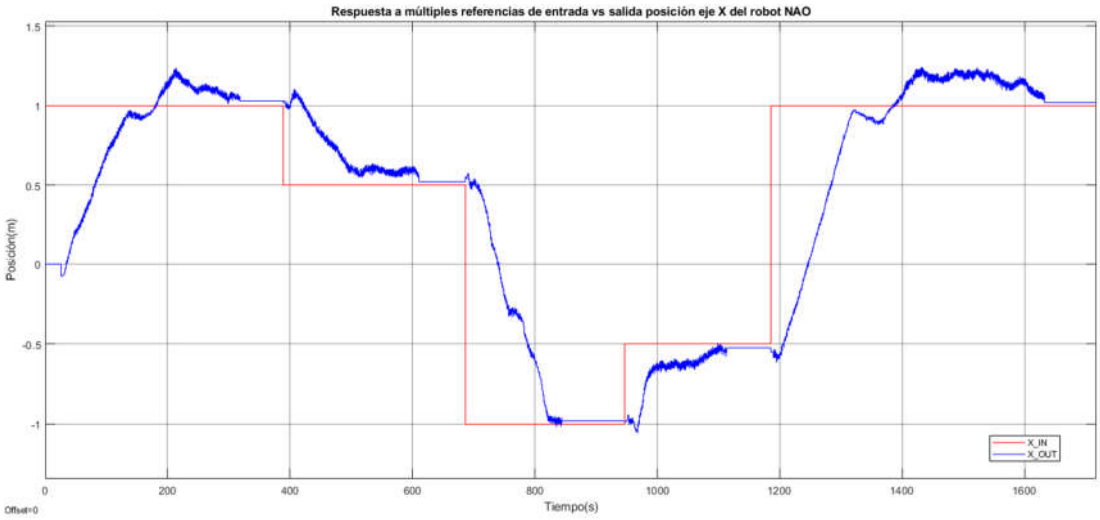

Figura 3. 1 Respuesta a múltiples referencias de entrada vs salida posición eje X del robot NAO.

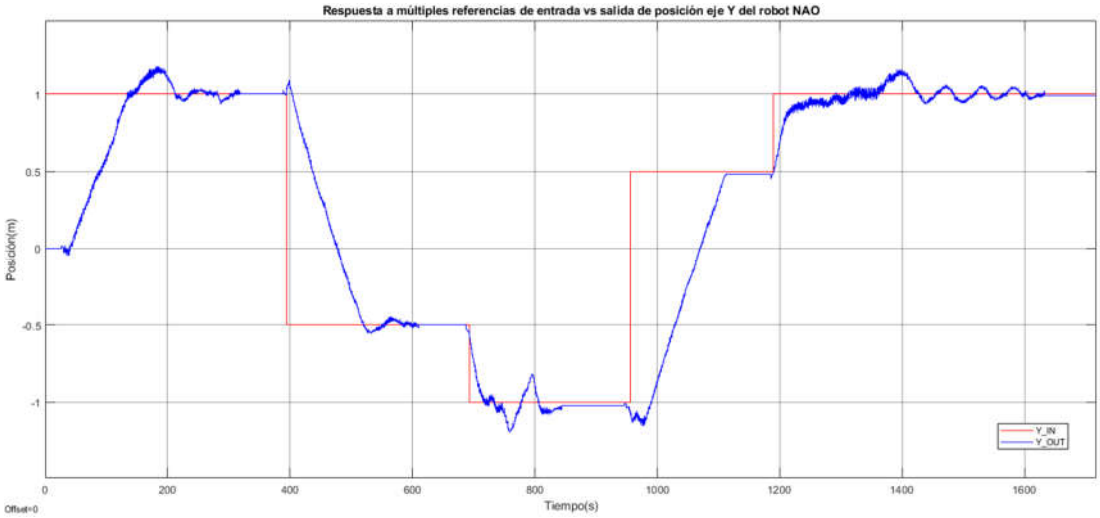

Figura 3. 2 Respuesta a múltiples referencias de entrada vs salida posición eje Y del robot NAO.

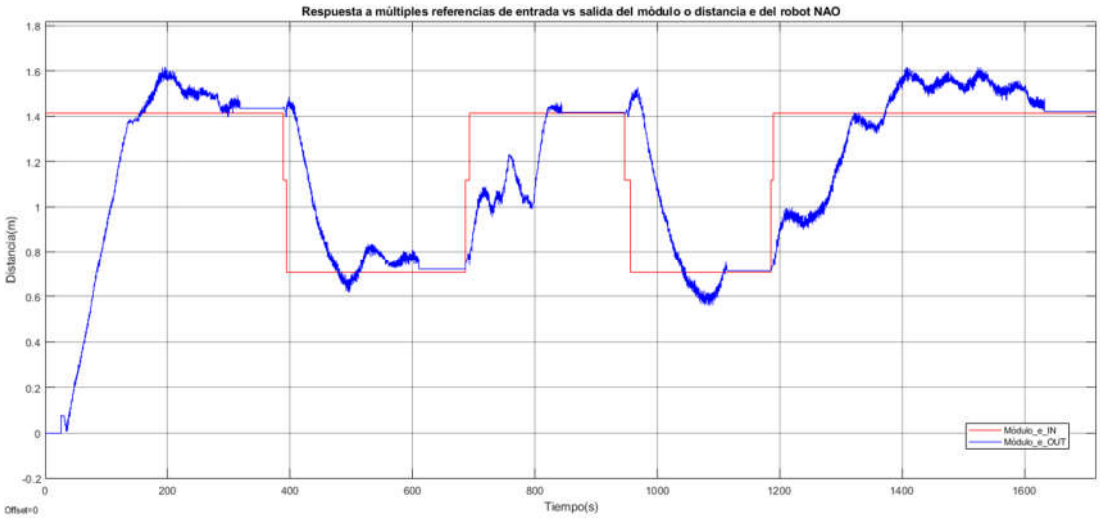

Figura 3. 3 Respuesta a múltiples referencias de entrada vs salida del módulo o distancia del robot NAO.

El error siempre tiende a llegar a cero al analizar tanto la respuesta en x, y o con el módulo como se puede ver en las Figura 3. 4, Figura 3. 5 y Figura 3. 6 con cada cambio de referencia la señal tiene un tiempo transitorio que finalmente termina próximo a cero.

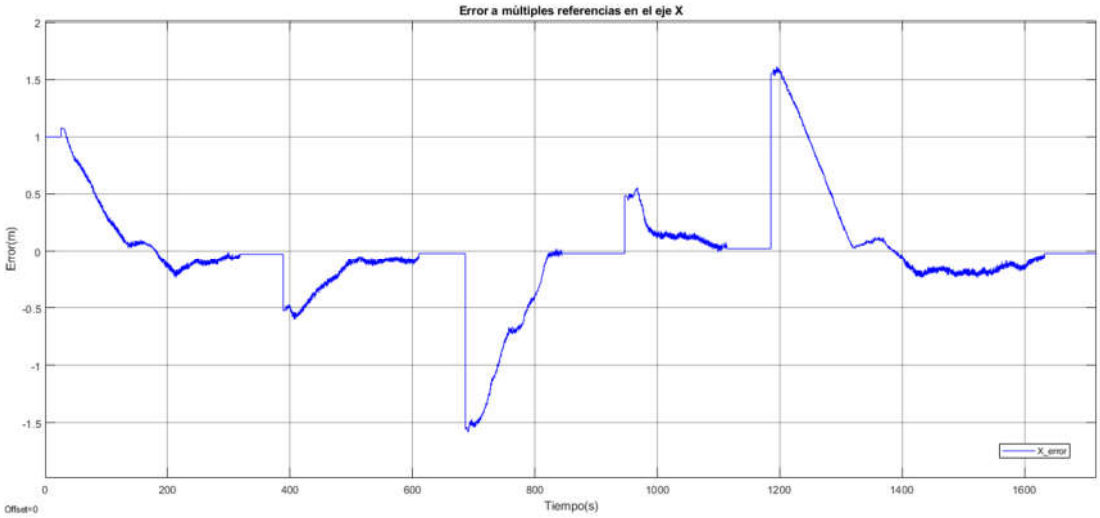

Figura 3. 4 Error a múltiples referencias en el eje X.

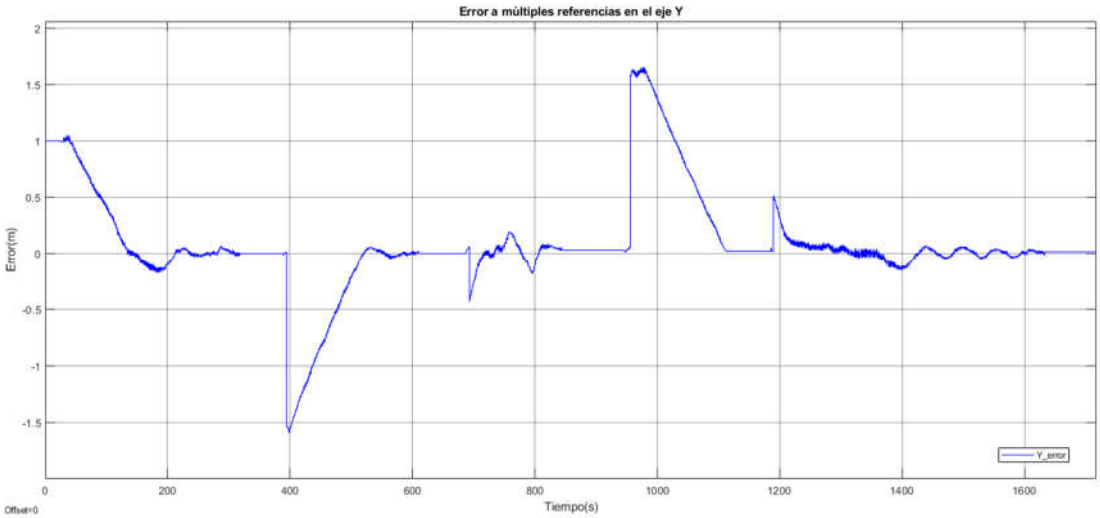

Figura 3. 5 Error a múltiples referencias en el eje Y.

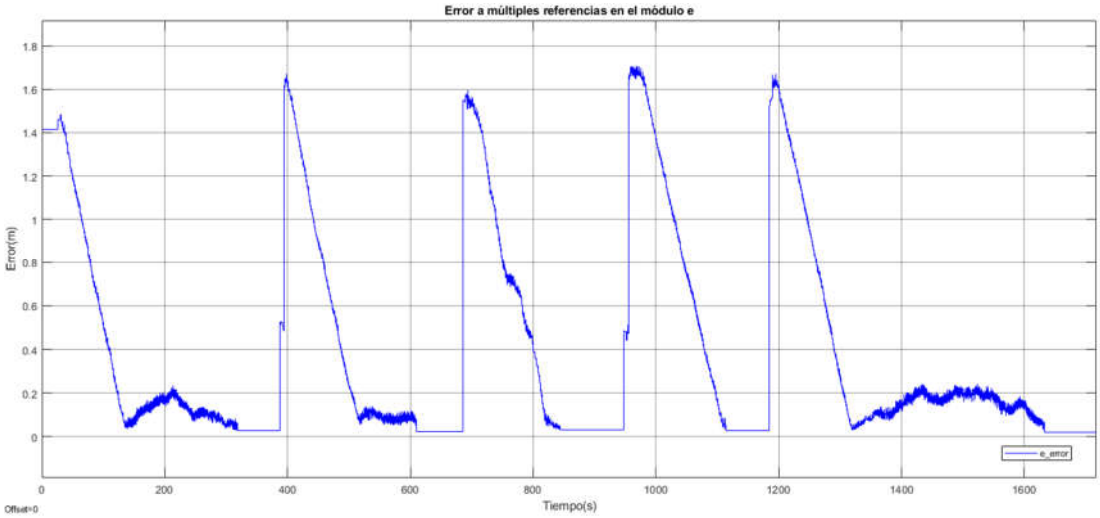

Figura 3. 6 Error a múltiples referencias en el módulo e.

Para múltiples cambios de referencias las acciones de control trabajan adecuadamente ya que las mismas tienden a cero cuando la salida llega a la referencia o posición deseada y esto se puede apreciar tanto en la velocidad lineal u de la **Figura 3. 7** como en la velocidad angular w de la **Figura 3. 8**.

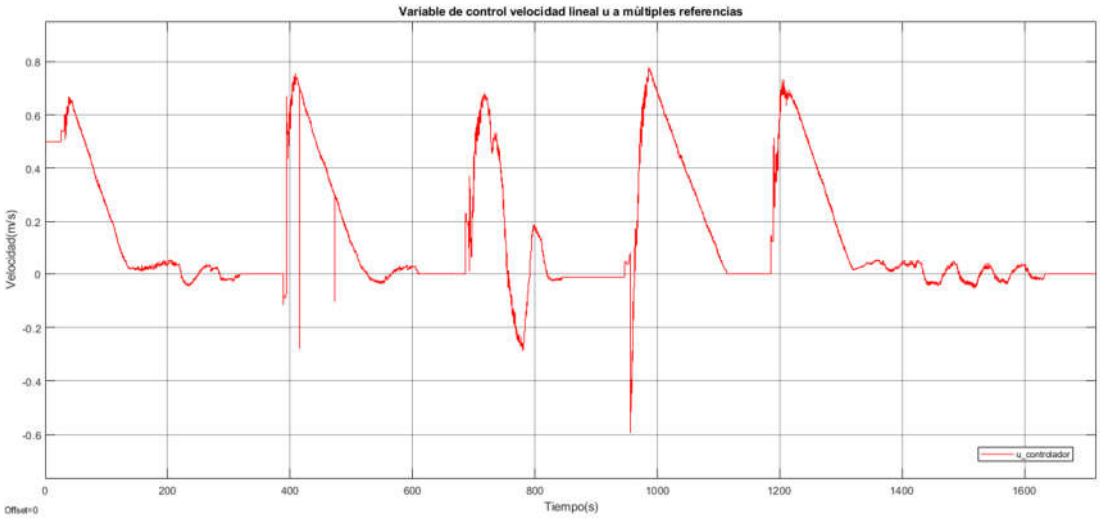

Figura 3. 7 Variable de control velocidad lineal u a múltiples referencias.

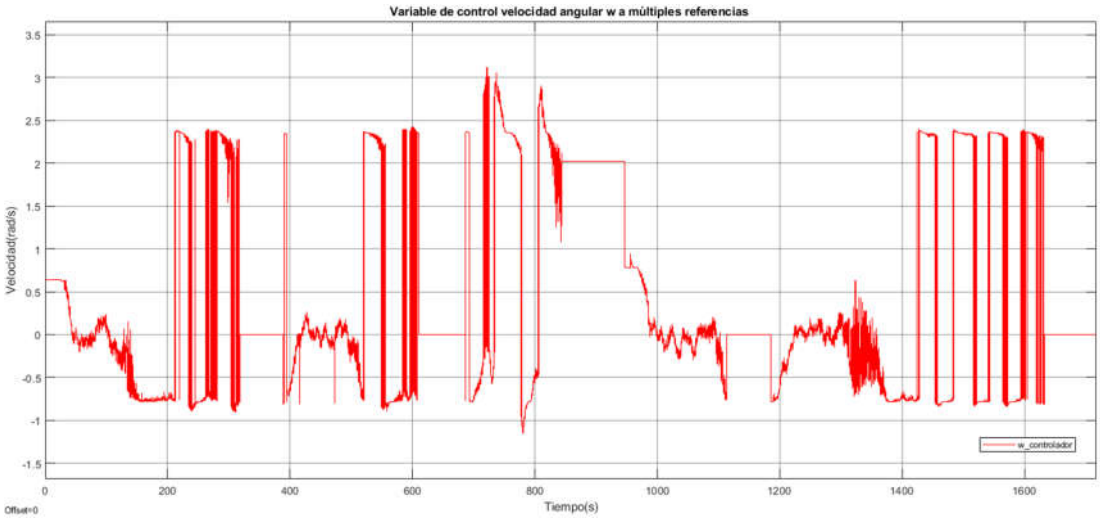

Figura 3. 8 Variable de control velocidad angular w a múltiples referencias.

La acción del movimiento del robot ante múltiples cambios de referencia se puede observar en la **Figura 3. 9** en donde se tiene 6 recuadros que muestran los estados o movimientos del robot conforme se cambiaba su set point, el estado 0 representa la posición inicial del robot dentro de CoppeliaSim esta posición viene ser el origen de coordenadas o posición (0, 0) y es la posición en la que siempre iniciara el robot NAO, el estado 1 muestra el movimiento del robot al primer cuadrante es decir la posición (1, 1), el estado 2 muestra la movilización al cuarto cuadrante o (0.5, -0.5), el estado 3 deja ver el movimiento del robot al tercer cuadrante es decir (-1, -1), el estado 4 muestra el desplazamiento al segundo cuadrante o (-0.5, 0.5) y finalmente para dar una vuelta completa el estado 5 muestra el

movimiento del robot nuevamente al primer cuadrante o (1, 1), el movimiento que observamos a múltiples referencias son los resultados que se obtuvieron en las gráficas expuestas con anterioridad.

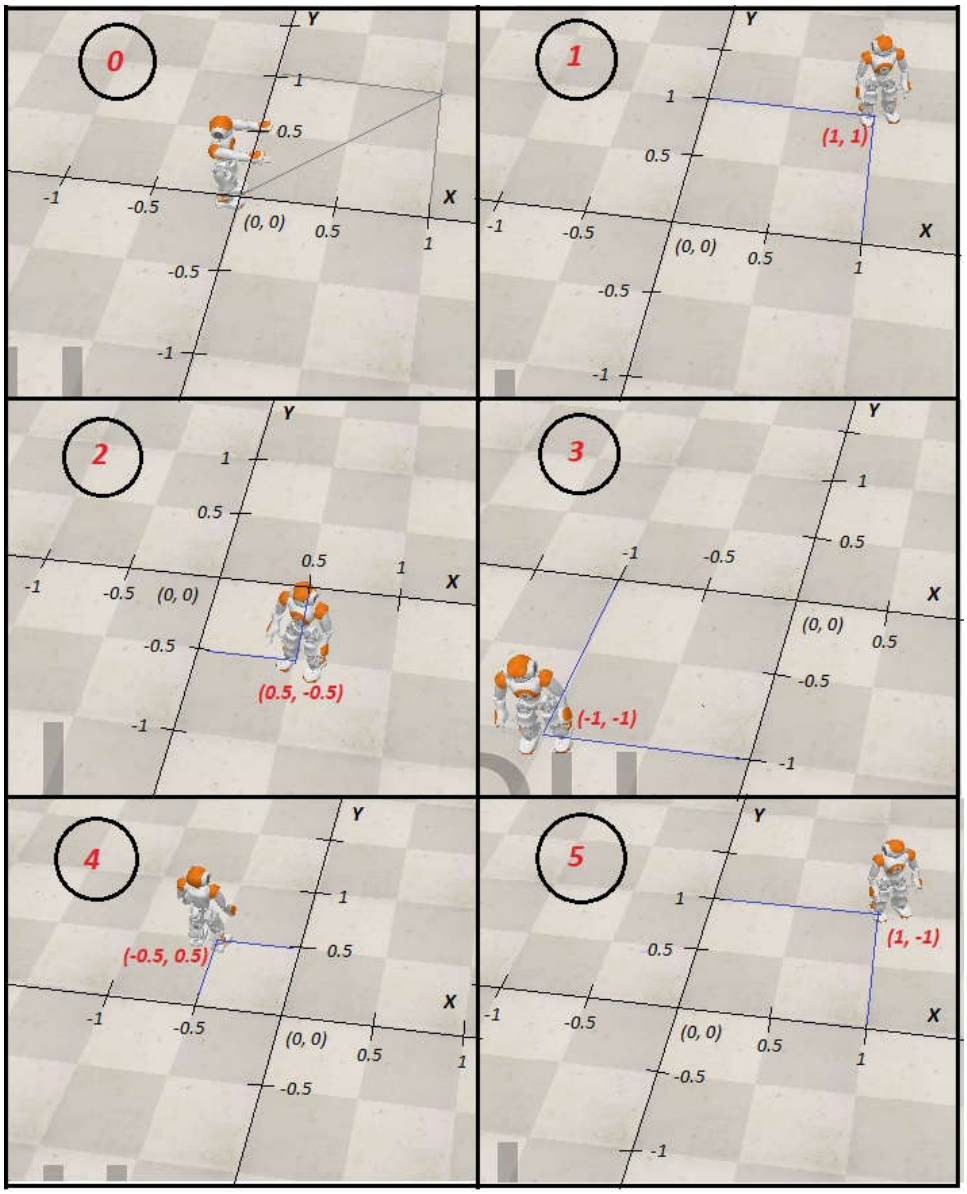

Figura 3. 9 Movimiento efectuado por el robot NAO a múltiples referencias de posición.

3.2. CONCLUSIONES

- El control de Lyapunov muestra un comportamiento que puede ser adecuado o no de acuerdo con los valores de sus constantes k1 y k2 propias del modelo matemático del control desarrollado, cuando el valor de k1 y k2 es menor a 0.5 el controlador tiende a conseguir un tiempo de establecimiento mayor y a demorarse

más la simulación mientras que sus índices de desempeño tanto el ISE como el ISU son mayores con respecto al resto de valores que se probaron lo que indica que el comportamiento es peor. Valores de k1 y k2 que sean de 0.5 o 0.7 dan los mejores resultados de ISE e ISU siendo un valor óptimo para ejecutar el controlador mientras que valores mayores a estos empieza a distorsionar la respuesta ya que se empieza a tener índices de desempeño y tiempo de establecimiento mayores.

- Ante la entrada de múltiples cambios de referencia se pudo observar que el controlador desarrollado para unas constantes de k1 y k2 de 0.5 el comportamiento de este es óptimo ya que el mismo sigue cada posición deseada que se introdujo en este caso se probó su funcionamiento introduciendo posiciones deseadas en todos los cuadrantes es decir haciendo que se mueva una vuelta completa en sentido horario por todos sus 4 cuadrantes y el mismo responde de manera eficaz y adecuada.

- La implementación del controlador en la plataforma de Matlab/Simulink resulto ser adecuada para ejecutar y probar el controlador ya que la misma permite implementar gran cantidad de tipo de controladores de una manera muy fácil e intuitiva.

- El controlador fue desarrollado suponiendo un modelo sencillo del robot NAO es decir asumiendo la cinemática y dinámica como una caja negra y ver al robot como una partícula en movimiento, para ello se asumió el modelo del robot como un robot uniciclo, el desarrollo del controlador para este modelo reducido resulto ser eficaz ya que nos ofrece la creación y prueba de varios controladores los cuales funcionan de manera correcta en el robot NAO sin una dificultad matemática extrema para el cálculo del mismo.

3.3. RECOMENDACIONES

- Se recomienda ocupar el comando move ya que este permite mover al robot mediante la velocidad expresada en m/s y rad/s a través de los tres parámetros que este utiliza (xp, yp, tethap) que son velocidad lineal en (x, y) y su velocidad angular. Tener en cuenta que el controlador transmite las señales de control solo mediante dos parámetros, la velocidad lineal en módulo y la velocidad angular (u, w) por lo que se aconseja enviar el comando de velocidad lineal u en el eje de la velocidad lineal X mientras que la velocidad lineal Y ponerla en cero de esta manera (u, 0, w). Recordando que si se pone el parámetro u en el eje de velocidad lineal Y así (0, u,

w) el robot se movilizara de manera extraña semejante a un cangrejo por lo cual no es muy aconsejable esta combinación.

- Se recomienda no utilizar en NAOqi de Python el comando moveTo ya que el mismo ya contiene un controlador PID y modo de ejecución interno lo cual hará que las acciones del controlador de Lyapunov desarrollado presenten problemas, este comando sirve para poner de manera directa una posición desea y este de manera autónoma se encargará de dirigir a la posición referencia sin controladores externos. Para probar nuevos controladores se deberá de utilizar el comando move el cual no dispone de ningún control, recordando que el comando moveTo sus parámetros están en metros y el comando move están en m/s.

- Se recomienda ocupar como tiempo de muestreo el mismo tiempo de ejecución de instrucciones que tienen el robot NAO el cual es de 20ms esto se debe poner en los bloques de Simulink y hará que no se pierda instrucciones del controlador ya que entre mayor este sea le tomara más tiempo al controlador llegar a la posición deseada por la pérdida de datos.

- Se recomienda utilizar el modelo cinemático de un robot que sea relativamente sencillo, en este caso el de un robot uniciclo asumiendo el movimiento del robot NAO como si fuera este tipo de robot, haciendo esto la creación de un algoritmo de control que sea viable para el robot NAO será mucho más sencillo de elaborar y se podrá probar múltiples controles.

- Es aconsejable utilizar las librerías de NAOqi que se encuentra disponibles en el programa de Python ya que las mismas permiten utilizar las funcionalidades que brinda el programa Choregraphe, este permite enviar y ejecutar acciones de manera muy sencilla de lo contrario nos tocaría programar en V-REP de CoppeliaSim en donde los comandos son muy complejos de ejecutar y tocaría programar las instrucciones articulación por articulación.

4. REFERENCIAS BIBLIOGRÁFICAS

[1] IEEE Robotics and Automation Society., ICRA 2009: 2009 IEEE International Conference on Robotics and Automation: Kobe, Japan: May 12-17, 2009. [IEEE], 2009.

[2] J. E. Fierro, “BipITLag robot View project Motion planning of robotic manipulators View project,” 2016. [Online]. Available: www.indautor.gob.mx

[3] Inc. RobotLAB, “Robot NAO V6 EDICIÓN ESTÁNDAR.” https://www.robotlab.com/tienda-de-robots/robot-nao-para-la-educacion (accessed May 24, 2022).

[4] C. Holguín, Y. Díaz-Ricardo, and R. Antonio Becerra-García, “El lenguaje de programación Python,” Ciencias Holguín, vol. 20, pp. 1–13, 2014, [Online]. Available: http://www.linuxjournal.com/article/2959

[5] A. Marzal and I. Gracia, “Introducción a la programación con Python,” 2003.

[6] P. Software Foundation, “threading — Paralelismo basado en hilos — documentación de Python - 3.8.13,” Mar. 17, 2022. https://docs.python.org/es/3.8/library/threading.html (accessed May 25, 2022).

[7] Aldebaran, “Marco NAOqi — Manual de documentación de Software NAO - 1.14.5.” http://doc.aldebaran.com/1-14/dev/naoqi/index.html (accessed May 25, 2022).

[8] Aldebaran, “Manual de Parallel Tasks - Making NAO move and speak — Aldebaran 2.4.3.28-r2 documentation.” http://doc.aldebaran.com/2-4/dev/python/making_nao_move.html (accessed May 25, 2022).

[9] D. Díaz, L. Leopoldo, and A. Ángel, “SIMULACIÓN DE UN ENTORNO INDUSTRIAL MEDIANTE LA HERRAMIENTA DE TRABAJO COPPELIASIM (V-REP),” UNIVERSITAT POLITÈCNICA DE VALÈNCIA, VALÈNCIA, 2019.

[10] CoppeliaSim, “Manual de Remote API functions (Python).” https://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsPython.htm (accessed May 26, 2022).

[11] E. Pot, J. Monceaux, R. Gelin, B. Maisonnier, and A. Robotics, “Choregraphe: A graphical tool for humanoid robot programming,” in Proceedings - IEEE International Workshop on Robot and Human Interactive Communication, 2009, pp. 46–51. doi: 10.1109/ROMAN.2009.5326209.

[12] Aldebaran, “Qué es Choregraphe-Manual de documentación de Aldebaran 2.4.3.28-r2.” http://doc.aldebaran.com/2-4/software/choregraphe/choregraphe_overview.html (accessed May 26, 2022).

[13] M. Ataurima, “MATLAB y Simulink para Ingeniería.” Universidad de Ciencias y humanidades UCH, Vol. 1., 2013

[14] MathWorks, “Centro de ayuda de MATLAB, Simulink y otros productos de MathWorks.” https://la.mathworks.com/help/ (accessed May 26, 2022).

[15] E. NUÑEZ and J. ORTEGA, “DISEÑO, SIMULACIÓN Y COMPARACIÓN DE UN CONTROLADOR PID Y UN CONTROLADOR BASADO EN LYAPUNOV PARA EL DESPLAZAMIENTO DE UN ROBOT HUMANOIDE NAO V6 SOBRE UN CAMINO GENERADO POR UN ALGORITMO DE EXPLORACIÓN RÁPIDA DE ÁRBOL ALEATORIO -RRT,” Escuela Politécnica Nacional, Quito, 2021.

[16] G. Andaluz, “MODELACIÓN, IDENTIFICACIÓN Y CONTROL DE ROBOTS MÓVILES,” Escuela Politécnica Nacional, Quito, 2011.

[17] X. Álvarez Brotons, “Control predictivo de canales de riego utilizando modelos de predicción de tipo Muskingum (primer orden) y de tipo Hayami (segundo orden),” Universidad Politécnica de Catalunya, Barcelona, 2004.

[18] M. Alberto Pérez Ing. Analía Pérez Hidalgo Bioing Elisa Pérez Berenguer, “‘INTRODUCCION A LOS SISTEMAS DE CONTROL Y MODELO MATEMÁTICO PARA SISTEMAS LINEALES INVARIANTES EN EL TIEMPO.,’” 2007.

[19] G. H. Millán, L. H. Ríos Gonzales, and M. Bueno López, “Implementación de un controlador de posición y movimiento de un robot móvil diferencial,” Tecnura, vol. 20, no. 48, pp. 123–136, Feb. 2016, doi: 10.14483/udistrital.jour.tecnura.2016. 2.a09.

[20] M. García and A. Barreiro, “Análisis de la Estabilidad según Lyapunov de un Control Borroso en Tiempo Discreto,” Vigo.

[21] S. G. Tzafestas, Introduction to mobile robot control, First., vol. 1. London: Elsevier, 2014.

[22] V. Orosco, “DESARROLLO DE UNA HERRAMIENTA COMPUTACIONAL PARA LA SINTONIZACIÓN DE PARÁMETROS DE CONTROLADORES PID Y SMC PARA EL SEGUIMIENTO DE TRAYECTORIA DE UN CUADRICÓPTERO BASADO EN ALGORITMOS GENÉTICOS,” Escuela Politécnica Nacional, Quito, 2018.

[23] A. Rodríguez Mariano, G. Reynoso Meza, D. E. Páramo Calderón, E. Chávez Conde, M. A. García Alvarado, and J. Carrillo Ahumada, “ANÁLISIS DEL DESEMPEÑO DE CONTROLADORES LINEALES SINTONIZADOS EN DIFERENTES ESTADOS ESTACIONARIOS DEL BIORREACTOR DE CHOLETTE MEDIANTE TÉCNICAS DE DECISIÓN MULTI-CRITERIO,” Revista Mexicana de Ingeniería Química, vol. 14, no. 1, pp. 167–204, May 2015.

[24] ISO and INTECO, INTE/ISO 9241-151:2018 Ergonomía de la interacción persona-sistema Parte 151: Directrices para las interfaces de usuario Web. ICS, 2018, pp. 1–68.

[25] D. Barragán and B. Guerrero, “Manual de Interfaz Gráfica de Usuario en Matlab,” 2018. [Online]. Available: www.matpic.com