ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELECTRÍCA Y ELECTRÓNICA

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

DISEÑO Y SIMULACIÓN DE CONTROL DE TRAYECTORIA DE UN ROBOT HUMANOIDE NAO

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

HENRY RAMIRO CALDERÓN CHANGO

henry.calderon01@epn.edu.ec

DIRECTOR: GEOVANNY DANILO CHÁVEZ GARCÍA

danilo.chavez@epn.edu.ec

DMQ, 13 de abril de 2023

CERTIFICACIONES

Yo, Henry Ramiro Calderón Chango declaro 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.

Henry Ramiro Calderón Chango

Certifico que el presente trabajo de integración curricular fue desarrollado por Henry Ramiro Calderón Chango, bajo mi supervisión.

Geovanny Danilo Chávez García
DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el producto resultante 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.

Henry Ramiro Calderón Chango

ESTUDIANTE

Geovanny Danilo Chávez García

DIRECTOR

DEDICATORIA

Este trabajo está dedicado a todos aquellos que han creído en mí, me han brindado su amor y su apoyo incondicional. A mi familia, por siempre estar a mi lado, brindándome su amor y su paciencia. A mis amigos, por su constante ánimo y por ser una fuente de inspiración.

Este trabajo es un reflejo del esfuerzo y dedicación que he puesto en mi formación académica. Espero que sirva como una inspiración para futuros estudiantes de ingeniería electrónica, y que los resultados obtenidos contribuyan de manera positiva al desarrollo de esta fascinante disciplina.

Gracias a todos por formar parte de este logro y por su constante apoyo.

ÍNDICE DE CONTENIDO

RESUMEN

En el siguiente trabajo de investigación, se presenta el diseño y simulación de un controlador basado en inteligencia artificial para el desplazamiento de un robot humanoide NAO V6 sobre una trayectoria fija.

Se realizó la comparativa entre el controlador diseñado con inteligencia artificial, un PID y uno basado en Lyapunov implementados en trabajos anteriores, para esto se utilizó tres tipos de trayectorias: circular, cuadrada y una lemniscata rotada 90°.

Adicionalmente, se presenta la implementación de una interfaz gráfica desarrollada en Simulink\Matlab, la cual permite al usuario la visualización de la trayectoria seleccionada y la realizada por el robot, así como del comportamiento general de los controladores.

PALABRAS CLAVE: NAO, Lyapunov, Inteligencia Artificial, Simulink, PID, Trayectoria

ABSTRACT

In the following research work, the design and simulation of a controller based on artificial intelligence for the displacement of a NAO V6 humanoid robot on a fixed trajectory is presented.

The comparison was made between the designed controller of artificial intelligence, a PID and one based on Lyapunov implemented in previous works, for these three types of trajectories: circular, square and lemniscate trajectories were used.

Additionally, the implementation of a graphical interface developed in Simulink\Matlab is presented, which allows the user to visualize the selected trajectory and that performed by the robot, as well as the general behavior of the controllers.

KEYWORDS: NAO, Lyapunov, Artificial Intelligence, Simulink, PID, Trajectory

1 INTRODUCCIÓN

La robótica ha crecido aceleradamente en los últimos años, esta tecnología ya no solo se encuentra en centros de investigación, laboratorios o industria, este avance tecnológico ha permitido que los robots también puedan encontrarse en ambientes más comunes para la sociedad como el hogar, escuelas y diferentes trabajos [1]. Sus funcionalidades han ido en aumento, cada vez se diseñan robots más potentes, autónomos y con aplicaciones más relacionadas con el común vivir de la sociedad [2]. Dentro del catálogo de robots disponibles o desarrollados, los robots sociales son los que buscan reproducir o simular las interacciones humanas, por lo que actualmente las empresas han desarrollado robots humanoides los cuales tienen una estructura antropomórfica y más apegada a la de un ser humano real [3], permitiéndoles tener una mayor aceptación al momento de la interacción con los seres humanos.

Los robots humanoides se han desarrollados primordialmente para la investigación ya que pueden realizar movimientos y acciones muy apegadas a las que realizan los humanos gracias a su gran cantidad de sensores y actuadores que le conceden altos grados de libertad [4]. La ventaja principal de este tipo de robots consiste en la capacidad de trabajar con los seres humanos sin tener que realizar modificaciones en su entorno, contrariamente a lo que se realiza en robots manipuladores y móviles [5]. Actualmente existen empresas dedicadas a la investigación de este tipo de robots; Honda ha desarrollado el robot ASIMO el cual puede es capaz de imitar varias acciones humanas como el subir y bajar escaleras, caminar trayectorias rectas, bailar y realizar otro tipo de actividades [6], así como otros robots que son capaces de interactuar con los seres humanos respondiendo a conversaciones u órdenes básicos simulando ser otras personas o incluso mascotas.

Así también la empresa japonesa SoftbankRobotics ha desarrollado el robot humanoide llamado Nao, el cual puede interactuar con personas de forma natural, también puede escuchar, observar, hablar, caminar, sentarse y muchas más acciones dependiendo de la programación que lo gobierna [7]. La complejidad del desarrollo de sus aplicaciones no tiene límite ya que, gracias a su cantidad de grados de libertad, puede realizar movimientos que van desde caminar por trayectorias complejas, jugar un partido de futbol y rodear obstáculos de una manera muy simple [8].

La locomoción bípeda en este tipo de robots resulta de gran complejidad, por lo que se ha desarrollado un sin número de investigaciones para lograr cada día un control mejor y más parecido al movimiento humano [9]. El problema más presente en el desarrollo de este tipo de robots es la extensa y complicada dinámica no lineal y alta inestabilidad. Por tal motivo

se han desarrollado diversas técnicas con el fin de tener un buen seguimiento de las trayectorias sin perder estabilidad [10].

Con el desarrollo de la inteligencia artificial se han podido solucionar problemas de alta complejidad gracias a que poseen un grupo de algoritmos lógicos que, en conjunto con un entrenamiento previo, son capaces de tomar decisiones en situaciones concretas [11]. Una de las aplicaciones principales de la inteligencia artificial es del desarrollo de controladores inteligentes, haciendo un reemplazo a los controladores tradicionales como el PID [12] o controladores basados en modelos matemáticos no lineales. El diseño de controladores inteligentes no requiere de una matemática exacta, ya que permiten resolver los problemas de la no linealidad basándose en el aprendizaje del controlador. Se han aplicado diversas técnicas de control para la simulación de trayectorias o mejora del seguimiento de los robots NAO [13]. Se han utilizado técnicas como controladores PID, Control basado en en el Criterio de Estabilidad de Lyapunov, y un algoritmo muy novedoso llamado Sammy Walk el cual utiliza técnicas tradicionales para la mejora del movimiento [14]. Sin embargo, no se han desarrollado controladores inteligentes para mejorar el control de movimiento y seguimiento de trayectoria en los robots NAO.

Por tal motivo, en el presente trabajo de titulación se plantea el desarrollo de un sistema de control utilizando técnicas de inteligencia artificial para la simulación de un robot humanoide NAO V6 dentro del software CoppeliaSim, con el fin de analizar y comparar el controlador diseñado en este proyecto, con los tradicionales PID y el basado en Lyapunov realizados en trabajos anteriores, para determinar la eficiencia y error de cada uno en trayectorias de tipo circular, cuadrada y lemniscata.

1.1 OBJETIVO GENERAL

Diseñar y simular sistema de control de trayectoria de un robot humanoide NAO

1.2 OBJETIVOS ESPECÍFICOS

1. Realizar una revisión bibliográfica de las técnicas de control para el seguimiento de trayectorias de robots bípedos, modelos cinemáticos de robots tipo uniciclo, así también, del robot humanoide NAO V6, del software de simulación CoppeliaSim, del lenguaje de programación Python, de la implementación de controladores en el entorno de Simulink/MATLAB y técnicas modernas de control basadas en inteligencia artificial.

2. Diseñar un esquema de control basado en inteligencia artificial e implementado en Simulink/MATLAB que aplicado al robot NAO permita el seguimiento de trayectorias previamente definidas y con el menor error.
3. Implementar la comunicación entre el controlador implementado en Simulink/MATLAB, Python y CoppeliaSim para la realización de simulaciones y análisis de resultados.
4. Simular el comportamiento del robot utilizando el software de CoppeliaSim para las trayectorias clásicas de prueba: circular, cuadrada y en 8.
5. Realizar pruebas de desempeño del esquema de control diseñado con otros controladores diseñados en trabajos anteriores.
6. Diseñar e implementar una interfaz gráfica en el entorno de MATLAB para la visualización del comportamiento del robot NAO con los esquemas de control probados y los resultados que permitan compararlos.

1.3 ALCANCE

El alcance definido para el presente Proyecto Técnico se lo especifica en los siguientes puntos:

- Se realizará una revisión bibliográfica de las características principales del robot humanoide NAO V6, del SDK del robot para su programación en Python, de los conceptos básicos y las librerías implementadas en el lenguaje de programación Python, el funcionamiento básico del software de simulación CoppeliaSim desde su propio entorno y desde un entono de Python, la comunicación bidireccional entre Python y Simulink/Matlab.
- Se realizará una revisión de las tendencias actuales de los controladores implementados en robots bípedos para el seguimiento de trayectorias tanto con esquemas de control clásicos como modernos, especialmente los basados en inteligencia artificial y redes neuronales, su programación y entrenamiento en el software de Simulink/Matlab.
- Se diseñará un esquema de control en Simulink/Matlab basado en inteligencia artificial utilizando los resultados de trabajos previos como base de datos para el entrenamiento del controlador.

- Se implementará un sistema de intercambio de información entre Simulink/Matlab y Python para la simulación del comportamiento del robot, el análisis y la visualización de los resultados obtenidos.
- Con el software de CoppeliaSim se realizará la simulación del comportamiento del robot para el esquema de control diseñado y otros de trabajos anteriores.
- Se diseñará e implementará una interfaz gráfica que permita la visualización de la trayectoria prevista y la realizada por robot, además de su comportamiento general con los esquemas de control probados.

1.4 MARCO TEÓRICO

1.4.1 ROBOT HUMANOIDE NAO

El robot NAO mostrado en la Figura 1.1, es el primero robot humanoide desarrollado por la empresa francesa Softbanks Robotics, el cual tiene una altura de 58 centímetros. Este robot posee 25 grados de libertad lo que le permite caminar de una forma natural y muy similar a la bípeda que realizamos los humanos [15]. Este robot posee una gran cantidad de sensores y actuadores lo que permite realizar acciones tales como caminar, bailar, reconocer objetos y personas, por lo que se lo utiliza para diversas investigaciones en el campo de la educación e interacciones sociales [1].

Figura 1.1 Robot Humanoide NAO [16]

Este robot humanoide consta hasta le fecha actual de 6 versiones, siendo la última la V6 y de la cual se indicará sus principales características.

1.4.1.1 Características Generales

El robot NAO tiene unas dimensiones de 57.32cm de alto, 27.33cm de ancho, 29cm de profundidad y un peso de 5.4kg [17]. Está compuesto por un procesador ubicado en el torso del robot, siendo un ATOM Z530 de 1,6GHZ, con una memoria RAM DE 1GB, memoria Flash de 2GB y una Micro SDHC de 8GB [17]. Otras de las características más importantes se presentan a continuación:

- **Batería:** Posee una batería de 21.6V con una corriente nominal de 2.25 A/H lo que le otorga una autonomía de entre 60-90 minutos dependiendo de las funciones utilizadas.
- **Conectividad:** Tiene una conexión Ethernet RJ45, Wi-Fi IEEE 802.11 b/g/n y conexión USB para la actualización del sistema operativo.
- **Accesorios:** Está compuesto por 4 micrófonos, 2 altavoces, 54 leds, 1 sensor FSR, sensores inerciales de tres ejes, 2 sensores ultrasónicos utilizados como sonares y 3 sensores táctiles de tipo capacitivo ubicados en la cabeza del robot.
- **Visión:** Posee una cámara MT9M114 con una resolución de 1.22Mp y un campo de visión 72.6°. Tiene una distancia de enfoque fijo de 30cm y una resolución de salida de video de 1280x960 a 30fps.

1.4.1.2 Cinemática del Robot

El robot NAO en su versión V6 posee 25 grados de libertad (DOF), los cuales se distribuyen en 5 cadenas cinemáticas las cuales forman como efector final las dos manos, los dos pies y la cámara ubicada en la cabeza del robot [4]. Como se observa en la Figura 1.2, la posición inicial o de referencia se la presenta con las piernas rectas, los brazos y cabeza apuntando hacia delante, también se observa la orientación en Pitch, Roll y Yaw de cada uno de los DOF.

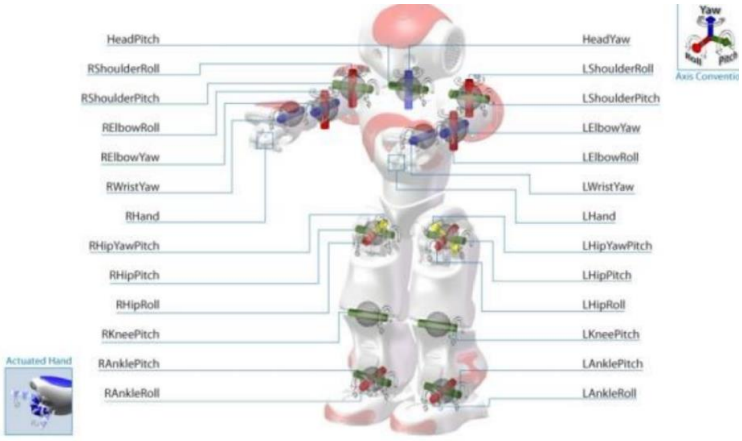

Figura 1.2 Posición Inicial del Robot NAO con sus respectivos grados de libertad [4].

El robot posee tres de marcos de referencia, del torso, del robot y del mundo como se observa en la Figura 1.3, cada uno de estos permite observarlo desde cada referencia para poder seguir las órdenes deseadas.

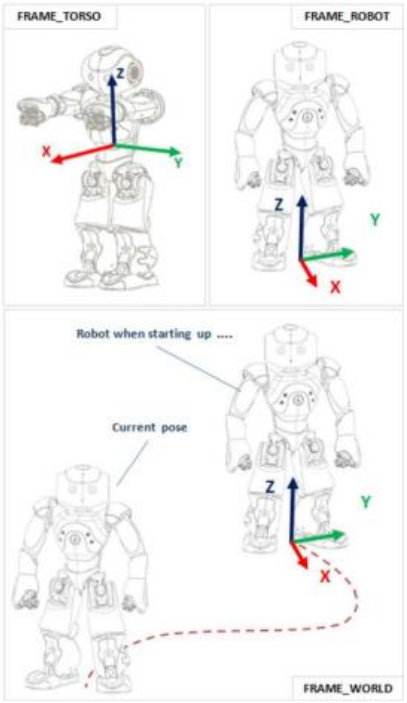

Figura 1.3 Sistemas de referencia del robot NAO [1].

1.4.1.3 NAOqi Framework

NAOqi es el sistema operativo con el cual trabaja el robot humanoide NAO, por lo que NAOqi framework son la diferentes funciones en las cuales se permite el control de los diversos módulos del robot. NAOqi framework permite el desarrollo de aplicaciones en diversos lenguajes de programación como Python o C++ [7].

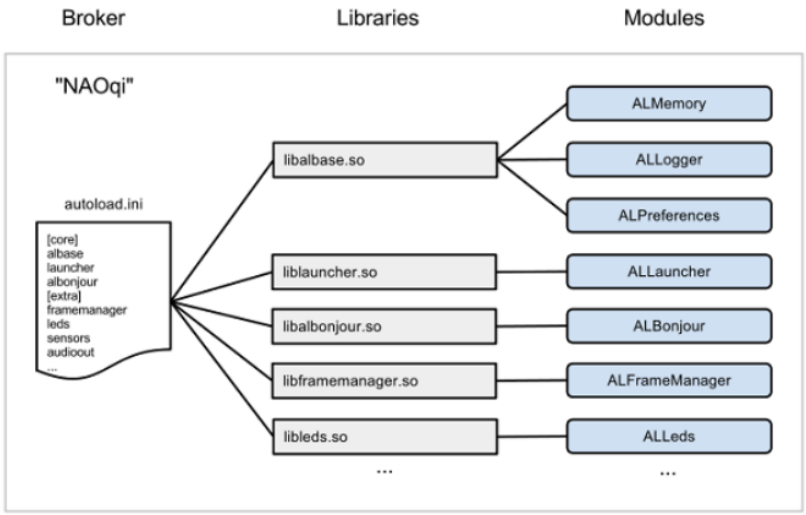

Figura 1.4 Estructura del NAOqi [18].

El archivo ejecutable de NAOqi, es el bróker principal *autoload.ini*, el cual define las librerías preparadas para la ejecución de cada una de las aplicaciones, como se observa en la Figura 1.4, la estructura de cada una de las librerías está compuesta por módulos los cuales utilizan al broker como intermediario para la ejecución de cada uno de estos métodos [19].

Este broker entrega servicios de búsqueda que permiten que cada uno de los módulos de la red encuentren a cualquiera de los métodos que se requieran y hayan sido publicados. Como se observa en la Figura 1.5 los módulos cargados conforman un árbol de métodos los cuales a su vez están ligados a un broker principal [19].

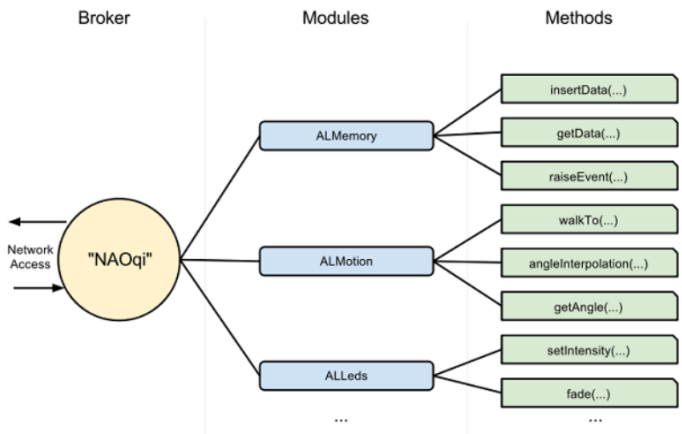

Figura 1.5 Esquema de Árbol del sistema NAOqi [18].

El framework de NAOqi está dividido en siete áreas principales las cuales tienen diferentes funcionalidades dentro del robot las cuales son: NAOqi Core, NAOqi Motion, NAOqi Audio, NAOqi Vision, NAOqui PeoplePerception, NAOqi Sensors y NAOqi Trackers. De los módulos antes mencionados, será el NAOqi Motion el que permita controlar los

movimientos del robot, se integra de instrucciones para el movimiento de las articulaciones, el caminata y posturas[20].

1.4.2 PYTHON

Python es un lenguaje de programación de alto nivel que posee estructuras implícitas de datos, conjuntos, listas y diccionarios los cuales permiten realizar tareas de alta complejidad con pocas líneas de código [21]. Este lenguaje puede ser utilizado en diversas plataformas como Windows, MacOS y Linux, sin embargo, la principal característica de este sistema es que es de código abierto por lo que los usuarios siguen desarrollando módulos que pueden ser utilizados por las personas que lo requieran y editarlos a su conveniencia [22]. Otras de las características importantes que engloban a este lenguaje de programación son las siguientes:

- **Simplicidad del código fuente:** La sintaxis de este lenguaje posee pocas palabras reservadas y una mínimas cantidad de tokens que incluyen símbolos, sin contar los operadores básicos.
- **Tipos de datos:** Posee datos dinámicos que no requieren declaración.
- **Indentación:** Se refiere a la forma de especificar las estructuras condicionales debido a la falta de delimitadores.
- **Estructuras de datos nativas:** Ofrece listas, tablas de hash y conjuntos las cuales poseen una sintaxis tan simple como la de los arreglos.
- **Multiparadigma:** Permite desarrollar algoritmos de manera estructurada, orientada a objetos la cual permite el desarrollo de aplicaciones en diversas áreas [23].

Por todas estas ventajas mencionadas anteriormente, se han desarrollado varias aplicaciones que pueden ser programadas en el robot NAO, para lo cual se requiere de un Software Development Kit (SDK) para Python. Un SDK no es más que un conjunto de herramientas que permiten la programación de ciertas aplicaciones en un entorno en específico [24]. Estas herramientas poseen bibliotecas, documentación, código de ejemplo y manuales que pueden utilizar los usuarios como guía para poder obtener integrar en sus propias aplicaciones [1].

Como se puede observar en la Figura 1.6, el componente que permite la interacción entre varias aplicaciones o softwares ya sean intermediarios o diferentes se conoce como Application Programming Interfaces (API's). Existe una API determinada para NAO que permite usar la API definida para C++, la cual, a través de una máquina virtual puede crear diversos módulos en Python que pueden ser ejecutados por el robot.

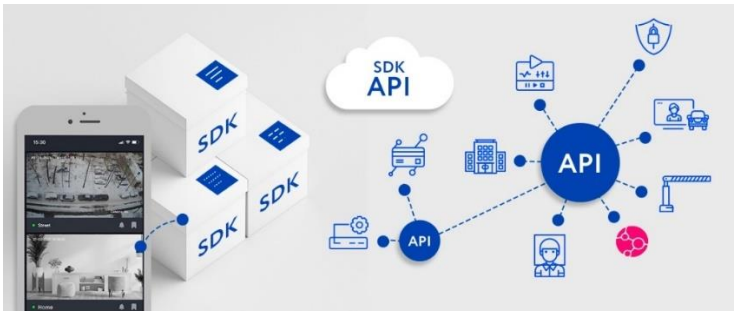

Figura 1.6 Esquema y relación entre SDK y API [25].

1.4.3 COPPELIASIM EDU

CoppeliaSim es un potente entorno que permite realizar simulaciones, pruebas y verificaciones de robots multiplataforma. Este software está considerado uno de los más importantes y mejores en el mercado debido a sus características que podemos mencionar a continuación:

- El simulador es flexible para realizar varias funcionalidades que pueden ser difícil en tiempo real.
- Los scripts se pueden codificar para manejar múltiples módulos dependiendo de los requisitos.
- Permite realizar aplicaciones de automatización de fábricas, monitoreo remoto, desarrollo de hardware, verificación, monitoreo seguro, desarrollo de algoritmos.
- Puede simular terrenos complejos, evitación de obstáculos, distribución de carga, estabilidad general [26].
- Herramientas de desarrollo GUI que permiten la integración de diversos robots comúnmente conocidos.

Como se observa en la Figura 1.7, existen métodos programación o codificación diferentes, los cuales tienen sus ventajas a la hora de realizar una simulación, los métodos incluidos en CoppeliaSim son: Scripts Embebidos, Pluggins, APIs, Nodo BlueZero, Nodo ROS y Add-ON[27].

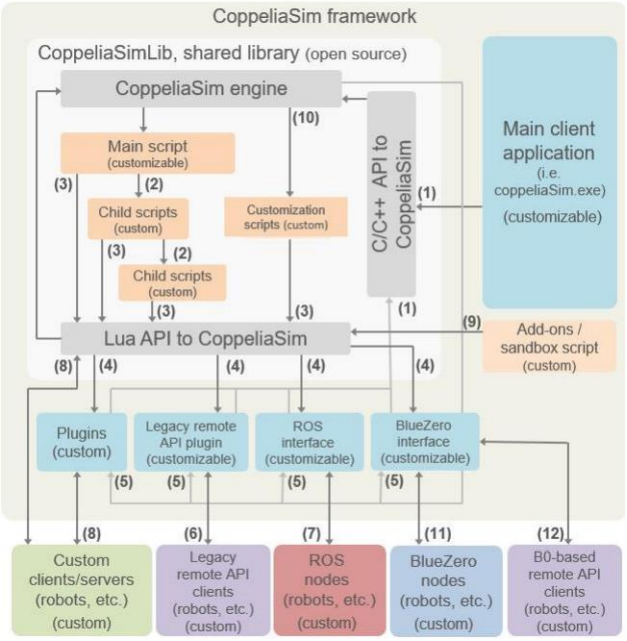

Figura 1.7 Diagrama de bloques para la codificación en CoppeliaSim

La API es la herramienta que se utilizará para el presente proyecto, ya que permite la conexión y entendimiento del sistema CoppeliaSim con los controladores diseñados en C++, Python, Java, Lua, Matlab, Octave o Urbi [28].

1.4.4 MATLAB / SIMULINK

Matlab es un entorno de programación el cual se basa en la realización de cálculos a través de matrices. Dentro de este programa se puede manipular datos basados en vectores, escalares, matrices pudiendo manipularlas con las diferentes operaciones que se realizan. Existen diferentes tipos de variables, arreglos y funciones dentro de este entorno las cuales pueden realizar aplicaciones estadísticas, de cálculos básicos, graficas de funciones y cálculos complejos [29].

Por otro lado, Simulink es un toolbox o herramienta de Matlab que sirve para realizar simulaciones, modelar y analizar diferentes sistemas. Se puede simular modelos lineales o no lineales, en tiempo continuo o no continuo, gracias una interfaz gráfica sencilla se pueden representar datos, operaciones y visualizar resultados en tiempo real de las aplicaciones diseñadas [29].

Dentro del ToolBox DSP System de Simulink, los bloques de UDP Send y UDP Receive que se observan en la Figura 1.8, son de gran importancia ya que permiten el envío y recepción de información de una aplicación a otra, por ejemplo, de Simulink a Python o viceversa, realizando la configuración de un puerto y una dirección IP remota [30].

Figura 1.8 Bloques de control UDP Send y Receive de Simulink [30].

1.4.5 VARIABLES Y ESTRATEGIAS DE CONTROL

En esta sección se dará una introducción y breve explicación de las variables que se involucran en la robótica móvil, así como los diferentes controladores que se utilizan para lograr alcanzar el control de trayectoria para un robot NAO.

1.4.5.1 Variables de un robot móvil

Para lograr comprender de la mejor forma las posiciones y orientaciones de en el espacio de los robots móviles, se utilizan los movimientos de rotación y translación, con el objetivo de conocer la posición y orientación del robot en el plano en el cual se está desplazando. Esta comprensión del movimiento del robot desde una posición inicial a una final nos indica el estudio de la cinemática móvil [31].

Esta cinemática permite plantear las trayectorias en los robots de tipo humanoide las cuales realizan sus movimientos tridimensionales en los planos sagital y frontal como se observa en el diagrama de bloques de la Figura 1.9. El plano sagital replica la locomoción humana, mientras que el plano frontal muestra la dinámica del balanceo que combinados forman el movimiento [32].

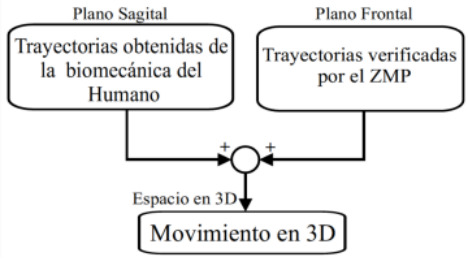

Figura 1.9 Diagrama de Bloques de los planos de marcha en robots humanoides [32].

Para este tipo de robots existen dos métodos de control los cuales son por posición y por fuerza o torque. En el robot NAO, se utilizan los servomotores y giroscopios para poder determinar la posición y el ángulo de postura del cuerpo. Como se va a analizar la locomoción en una trayectoria fija, se requiere únicamente el estudio del plano frontal para movimientos de aducción o abducción, y el plano sagital para movimientos de flexión o extensión [32].

Las variables que se introducen en este estudio, se pueden determinar debido a que se conoce los ángulos que requerimos para el plano sagital los cuales son: LHipPitch, LKneePitch, LAnklePitch, RHipPitch, RKneePitch, RAnklePitch; mientras que para el plano frontal tenemos los ángulos LHipRoll, LAnkleRoll, RHipRoll y RAnkleRoll [32].

1.4.5.2 Controladores PID

La estructura de un lazo de control retroalimentado se puede observar en la Figura 1.10, en donde el controlador es el encargado de efectuar los modos o acciones de control que interactúan sobre la señal de error $e(t)$; misma que es la diferencia entre el valor buscado de la variable controlada $r(t)$ o conocido como *setpoint* y el valor real $y(t)$. Estas características que se aplican en el sistema de control permiten que la variable controlada cambie su valor en caso de que exista la presencia de una perturbación $z(t)$, en la menor cantidad de tiempo, errores y oscilaciones. Las formas en las que el controlador realiza estas acciones depende de la configuración y variación de los parámetros fundamentales del mismo [33].

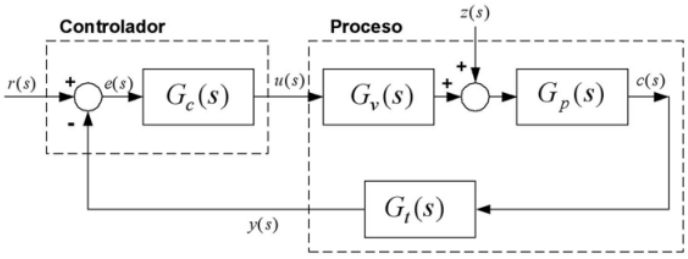

Figura 1.10 Diagrama de bloques básico de un sistema de control tradicional [33].

La mayoría de los controladores se basa en este principio, sin embargo, el control proporcional integral derivativo (PID), es el más utilizado en la industria, siendo la ecuación 1.1 la encargada de representar su funcionalidad.

$$u(t) = kp * \left(e(t) + \frac{1}{T_i}\int_0^t e(t)dt + T_d \frac{de(t)}{d(t)}\right) \quad (1.1)$$

Donde:

kp es la constante proporcional del controlador.

T_i es el tiempo de asentamiento de la curva de respuesta de la planta.

T_d es el tiempo de retardo de la curva de respuesta de la planta.

Dependiendo de la necesidad y de la planta o sistema que se está analizando se pueden considerar las respectivas variaciones de esta ecuación. Estas variaciones no permiten tener controladores de tipo proporcional (P), proporcional derivativo (PD) y proporcional

integral (PI). El término proporcional de la ecuación permite la reducción del error en régimen permanente, sin embargo, el uso de este puede provocar sobre oscilaciones y estabilidad relativa en el sistema. La acción integral genera un efecto sobre el error en régimen permanente garantizando la eliminación de este cuando la entrada es de tipo escalón. Finalmente, la acción derivativa permite cierta estimación del futuro error por lo que permite tener una idea anticipativa del funcionamiento. A continuación, se presenta una breve explicación de las variantes del control PID [34].

1.4.5.2.1 Control Proporcional (P)

En algunos procesos se puede trabajar con una ganancia muy elevada sin provocar en el sistema problemas de estabilidad. Estos procesos generalmente tienen un constante de tiempo dominante o tienen un término integral dentro de su esquema. Una ganancia alt dentro de un controlador P, quiere decir que el error en estado estacionario será pequeño y no requiere incluir una acción integral. Un ejemplo de estos se puede observan en el bucle interno de los controladores de cascada. La ecuación 1.2 es la encargada de representar un controlador tipo P [34].

$$u(t) = kp * e(t) \tag{1.2}$$

1.4.5.2.2 Control Proporcional Integral (PI)

Esta estructura de control es la más utilizada en los procesos industriales, El colocar una acción integral dentro del controlador es la mejor forma en la que se puede eliminar el error en régimen permanente. Otro caso en el que se puede utilizar este tipo de controladores es cuando el desfase que se coloca en el proceso es moderado (procesos con integradores puros). Cuando existen retardos también se recomienda la utilización de la acción integral ya que al contrario de la acción derivativa no amplifica la frecuencia y el ruido existente en el sistema. La ecuación 1.3 que se muestra a continuación, representa la forma básica de un control PI.

$$u(t) = kp * \left(e(t) + \frac{1}{T_i}\int_0^t e(t)dt\right) \tag{1.3}$$

1.4.5.2.3 Control Proporcional Derivativo (PD)

El control PD es utilizado cuando el proceso que se va a controlar incorpore dentro de su estructura un término integrador. Un ejemplo de esto son los procesos térmicos, ya que toda la energía que se le es suministrada se puede emplear para elevar la temperatura de un horno y evitar el desecho de las pérdidas que son despreciables. En esta clase de procesos se puede trabajar con elevadas ganancias en el controlador sin que requiera la adición del término integral. La acción derivativa tiene una alta sensibilidad al ruido debido

a las altas frecuencias que requiere una ganancia elevada, por este motivo cuando existen niveles altos de ruido se debe limitar la ganancia o evitar colocar la acción derivativa. Por otro lado, en procesos que posean grandes tiempos muerto el término derivativo no tiene efecto ya que la aproximación lineal que entrega solo es válida cuando los valores de T_d son pequeños. La ecuación 1.4 es la encargada de representar a un controlador de tipo PD [34].

$$u(t) = kp * \left(e(t) + T_d \frac{de(t)}{d(t)}\right) \quad (1.4)$$

1.4.5.3 Controlador Lyapunov

La teoría de Lyapunov es una base fundamental para el análisis de la estabilidad en sistemas de control definidos por sus respectivas ecuaciones en el espacio de estados [35], por ejemplo, para este caso en específico en el que se está analizando el comportamiento de robots móviles. El objetivo de análisis de estabilidad de un sistema de control es determinar si se puede aplicar el controlador o no a una planta real. Debido a que la mayoría de los sistemas o plantas de la industria tienen un comportamiento no lineal, no es posible la determinación de la estabilidad del sistema por los métodos tradicionales de Jury o Routh – Hurwitz. En esos casos, se aplica de manera más concreta el criterio de estabilidad de Lyapunov porque es utilizado para sistemas variantes, invariantes en el tiempo, lineales y no lineales [36].

Para poder realizar este análisis de Lyapunov existe un método directo y un método indirecto. El método directo no requiere de la obtención de las ecuaciones diferenciales o de las ecuaciones de diferencias por lo que resulta mejor dentro de una implementación práctica. El método indirecto, por lo contrario, requiere utilizar las formas explícitas de las soluciones de sus respectivas ecuaciones diferenciales o ecuaciones de diferencias [35].

La manera más clara de entender el método directo de Lyapunov, es la extensión matemática de una observación física fundamental; por ejemplo, cuando la energía total de un sistema eléctrico, térmico o mecánico tiene pérdidas continuas, entonces el sistema lineal o no lineal debe quedar en algún momento en un punto de equilibrio. De esta manera se puede determinar la estabilidad de un sistema mediante la variación de una función escalar $V(t, x)$ conocida como función de Lyapunov [35].

Para que a un sistema se considere estable en un punto de equilibrio en el origen debe cumplir las siguientes condiciones:

1. $V(t, x)$ es continua de igual manera que sus derivadas.
2. $V(t, x)$ es positiva $V(x) > 0$

3. $\dot{V}(t,x)$ es negativa $V(x) < 0$
4. $V(t,0) = 0$

Estas condiciones implican que, si el sistema es asintóticamente estable, entonces la energía que se encuentra almacenada en él, disminuye al transcurrir el tiempo, hasta que llega a un valor mínimo el cual es considerado el estado de equilibrio [37].

Una vez introducido el concepto de Lyapunov de estabilidad se procederá a mencionar los pasos para el diseño de un controlador basado en este criterio de estabilidad.

- Escoger una función Lyapunov $V(t,x)$ de prueba que cumpla con las condiciones antes mencionadas.
- Obtener la derivada $\dot{V}(t,x)$ a lo largo de la trayectoria del sistema como se observa a en la ecuación 1.5.

$$\dot{x} = f(x,u,t) \tag{1.5}$$

- Seleccionar una ley de control retroalimentada

$$u = u(x) \tag{1.6}$$

- Asegurarse que se cumpla la condición

$$\frac{dV(x)}{dt} < 0 \; para \; x \neq 0 \tag{1.7}$$

Por lo general, $u(x)$ es una función no lineal que posee parámetros y ganancias las cuales pueden provocar que la condición $\frac{dV(x)}{dt} < 0$ se cumpla, por lo que asegura que el sistema retroalimento sea asintóticamente estable [37].

1.4.5.4 Controladores basados en Inteligencia Artificial

En la actualidad existen diversos métodos basados en inteligencia artificial, machine learning y lógica difusa para los cuales se puede obtener un controlador para diversas aplicaciones ya que estos permiten controlar sistemas lineales y no lineales.

1.4.5.4.1 Controladores Basados en Redes Neuronales

Las redes neuronales artificiales han sido utilizadas de manera común para el control y modelamiento de sistemas con un grado de complejidad alto, ya que tiene la capacidad de aproximar funciones $f: \Re^M \rightarrow \Re^N$. De manera más simple, estas redes están compuestas por una capa de entrada, un conjunto de capas intermedias ocultas y una capa de salida. Cada capa está conformada por un conjunto de neuronas, mismas que se encuentran

interconectadas unas con otras, es decir, las salidas de cada neurona se conectan con las entradas de la siguiente capa; este enlace formado entre dos neuronas posee un coeficiente asociado denominado peso.

Cada una de estas neuronas posee una función matemática asociada $f: \Re^N \rightarrow \Re$, en donde "N" es la cantidad de neuronas que posee la capa previa. La cantidad de neuronas en cada capa no requiere ser la misma en todas, depende del diseño de la red neuronal. Por otro lado, el algoritmo de aprendizaje permite el ajuste de los pesos y coeficientes de cada neurona para que logre ejecutar la función deseada. La clasificación o tipo de red neuronal depende de las funciones matemáticas de cada neurona y la estructura de la red; entre los que destacan la red neuronal convolucional (CNN), perceptrón multicapa, red neuronal recurrente (RNN), red de base radial (RBF) [38].

Las redes perceptrón multicapa o también conocidas como "feedforward", son redes que presentan una o varias capas ocultas en su estructura tal y como se puede observar en la Figura 1.8.

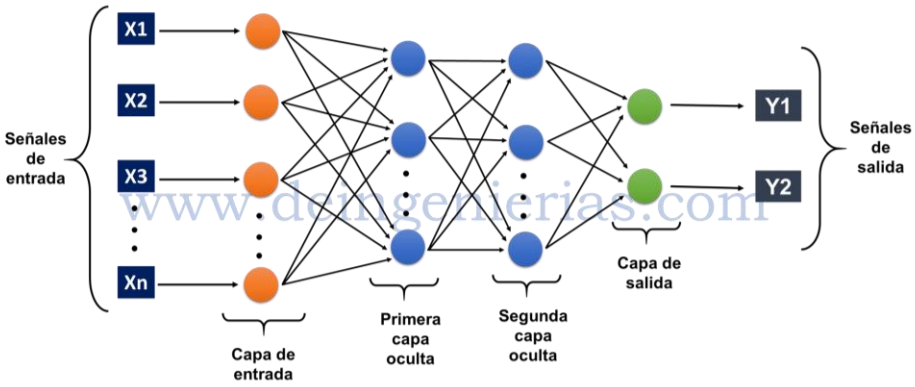

Figura 1.8 Arquitectura de un perceptrón multicapa con dos capas ocultas [39].

Entre los elementos que constituyen la red de este tipo están: señales de entrada $x_1, x_2, x_3 \ldots x_n$, conexiones sinápticas ponderadas w_i, conocida como la matriz de pesos, el *bias* b_i un sumador, función de transferencia y las señales de salida y_1, y_2. El proceso que sigue la red consiste en que cada neurona recibe el valor de la suma de sus entradas más el *bias,* y transporta el resultado de esta operación a través de la función de activación para obtener la salida. Existen varias funciones de activación entre las cuales se destaca la sigmoide, la tangente y la hiperbólica [40].

Este tipo de redes han generado buenos resultados en problemas de clasificación de patrones y en estrategias de aprendizaje supervisado y no supervisado; siendo el primer caso, en donde se requiere conocer los valores esperados para cada una de las entradas,

por lo que es indispensable tener previamente un conjunto de datos de entrenamiento representados por la ecuación 1.11:

$$\{x_1, t_1\}, \{x_2, t_2\}, \dots, \{x_n, t_n\} \tag{1.11}$$

En donde cada valor de x_i que se tiene en la entrada, se tiene un valor de t_i en la salida. Por consiguiente, cuando se aplica el valor de p_i a la red, se obtiene un valor de salida y_i, el mismo que está determinado por la ecuación 1.12, y el cual se compara con el valor esperado t_i.

$$y = f(\sum_i w_i x_i) \tag{1.12}$$

Los valores de los pesos w_i, establecen el funcionamiento de la red, por lo que se lo pueden fijar empleando diferentes algoritmos de entrenamiento de la red neural, entre los más conocidos está la propagación del error hacia atrás o conocida como BackPropagation (BP).

El entrenamiento en una red perceptrón multicapa se estima como un problema de optimización, en cual requiere encontrar una función de coste en donde el error sea mínimo. El entrenamiento por BP básico es un algoritmo en donde los pesos w_i y el *bias* de la red se ajusten en la dirección para la cual el error de la función de coste disminuya más rápidamente, lo que está representado por la ecuación 1.13.

$$p_{i+1} = p_i - \alpha_i g_i \tag{1.13}$$

En donde p_i es un vector conformado por pesos y en el instante actual, g_i representa al gradiente actual, y α_i es la velocidad de aprendizaje. Este algoritmo se utiliza en redes de aprendizaje supervisado las cuales generan un ciclo de propagación – adaptación hacia delante y hacia atrás.

Cuando se entrega un modelo x en la entrada de la red, este se transporta desde la capa de entrada hasta la capa de salida de la red. La variable que otorga la salida se compara con la señal deseada, y se procede al cálculo del error entre ellas; una vez obtenido este error, esta señal se transporta hacia atrás, desde la capa de salida como se observa en la Figura 1.9, hacia las neuronas de la capa oculta [40].

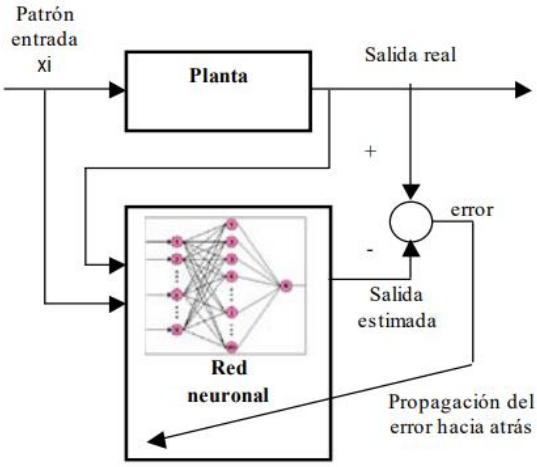

Figura 1.9 Estructura de un sistema de control entrenado por BackPropagation [40].

Este procedo se reproduce en cada una de las capas que conforman la red, hasta que cada neurona obtenga una señal de error que represente su aporte al error total. Una vez obtenidas estas señales de error los pesos se actualizan en cada neurona, para que la red se dirija hacia un estado en el cual se pueda clasificar de forma correcta todos los patrones de entrenamiento [40].

1.4.6 ÍNDICES DE DESEMPEÑO DE UN SISTEMA CONTROLADO

1.4.6.1 Integral del Error Absoluto IAE

Este índice es uno de los más sencillos en ser implementados, por lo que en sistemas sub amortiguados o altamente amortiguados no se puede llegar a tener un valor óptimo. Un sistema óptimo bajo este criterio debe ser uno que posea un amortiguamiento razonable y una correcta respuesta transitoria [41]. La ecuación 1.14 es la encargada de determinar este indicador, lo que indica que el controlador posee un buen desempeño es cuando este valor calculado se aproxima a cero [1].

$$IAE = \int_0^\infty |e(t)|dt \qquad (1.14)$$

1.4.6.2 Integral del Error Cuadrático ISE

Un sistema que logra disminuir en su mayoría este indicador ISE se puede considerar que es un sistema óptimo. Este índice se utiliza con regularidad cuando se activan entradas escalón y estadísticas por su facilidad en la implementación análoga y digital. Así también, puede discriminar entre sistemas con exceso de sobre amortiguamiento y sub-amortiguamiento. Este índice está representado por la ecuación 1.15, en donde la integral llega a un valor mínimo dependiendo de la cantidad de amortiguación, por lo que este

criterio tiene mayor valor cuando son errores grandes y menor valor cuando los errores son pequeños [41].

$$ISE = \int_0^\infty e(t)^2 dt \tag{1.15}$$

Una de las ventajas de este índice es que no es sensible a los cambios de parámetros y su cálculo es muy sencillo. En la práctica, es muy significativo su uso, ya que puede ser indicador de la minimización del consumo de energía en cierto tipo de sistemas como por ejemplo en los aeroespaciales [1].

2 METODOLOGÍA

2.1 SIMULACIÓN DEL ROBOT NAO EN COPPELIASIM

2.1.1 SISTEMA DE REFERENCIA DEL ROBOT NAO EN EL ENTORNO COPPELIASIM

El simulador CoppeliaSim Edu es un entorno que permite reproducir las acciones de movimiento que realiza el robot NAO, para lo cual es necesario conocer el entorno con el que se interactúa. Para este proyecto se requiere conocer el marco de referencia característico, mismo que es una cuadrícula dividida por cuadros de menor tamaño. El entorno posee ejes de referencia de igual forma que un plano cartesiano tradicional, en donde el eje “x” se ubica a lo largo de una recta horizontal, mientras que el eje “y” se sitúa a lo largo de una recta vertical, finalmente el eje “z” es una proyección perpendicular con respecto a los ejes (x, y) de tal forma que pareciera que el eje sale a través de la pantalla. Una vez entendida la distribución de los ejes de referencia se puede identificar el origen de coordenadas (0,0), en la intersección de los ejes como se muestra en la *Figura 2.1*, así también se observa que cada una de las cuadrículas que lo subdividen tienen un tamaño de 50x50cm.

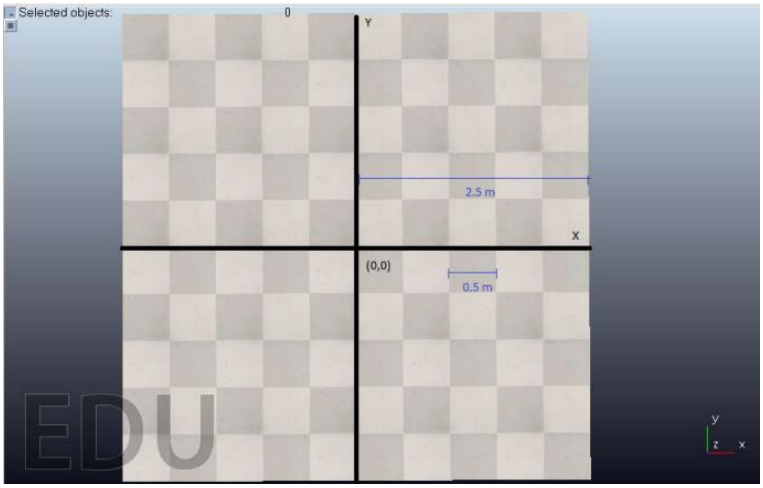

Figura 2.1 Sistema de Referencia del entorno CoppeliaSim Edu.

Para lograr una mayor sencillez en el seguimiento de la trayectoria, se ubica al robot NAO en el origen (0,0) como se muestra en la *Figura 2.2*, desde esta posición se pueden realizar movimientos hacia cualquier punto del plano, para ello se divide este en cuatro cuadrantes; de esta manera se puede ingresar las coordenadas (x, y) específicas a las que se requiere que el robot se traslade dependiendo del cuadrante en el que está situado.

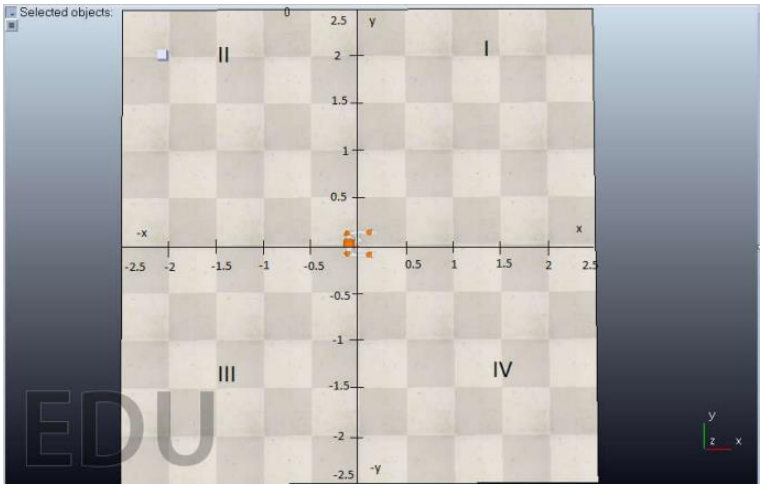

Figura 2.2 Cuadrantes del Marco de Referencia en simulador CoppeliaSim Edu.

Para que el robot Nao realice un desplazamiento de un punto A hacia un punto B, se requiere conocer los parámetros necesarios para efectuar este movimiento, como se observa en la Figura 2.3., uno de los parámetros es la posición real que viene dada por la expresión $(X_{real},\ Y_{real})$, la cual es la posición en la que se encuentra en robot en el instante actual. El segundo parámetro requerido son las coordenadas a las que se requiere que el robot se traslade dentro del marco de referencia, estas vienen dadas por la expresión $(X_{deseada},\ Y_{deseada})$. Finalmente se requiere conocer las variables que el controlador va a recibir, siendo estas la distancia hacia el punto deseado D y el ángulo de orientación $Theta$

requerido para realizar el desplazamiento requerido. En la *Figura 2.3*, se muestran las variables requeridas para realizar el movimiento del robot Nao desde el origen de coordenadas, hacia un punto situado en el primer cuadrante con las coordenadas $(1,\ 1)$.

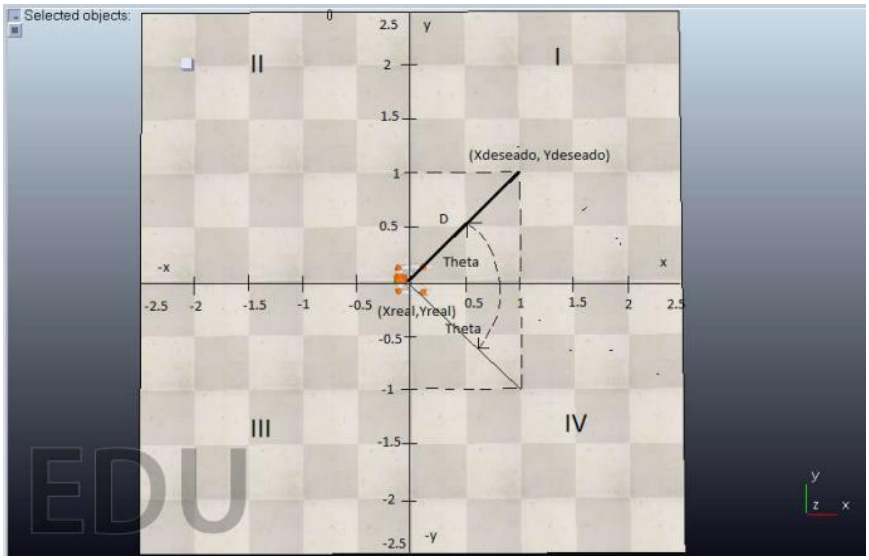

Figura 2.3 Parámetros para el desplazamiento del robot en el entorno CoppeliaSim Edu.

Como se observa en la *Tabla 2.1*, el ángulo $Theta$ siempre está dirigido hacia el eje X, para lo cual en los cuadrantes I y II se lo toma a partir del eje X positivo en sentido antihorario, mientras que para los cuadrantes III y IV se mide el ángulo desde el eje X positivo pero en sentido horario, obteniendo de esta forma el ángulo θ adecuado para determinar la dirección del robot.

Tabla 2.1 Relación Trigonométrica de ángulo θ en cada cuadrante.

Cuadrante	Relación Trigonométrica	Ángulo θ Deseado
I	$arctang\left(\frac{Y}{X}\right)$	$\theta = arctang\left(\frac{Y}{X}\right)$
II	$arctang\left(\frac{Y}{-X}\right)$	$\theta = \pi + arctang\left(\frac{Y}{-X}\right)$
III	$arctang\left(\frac{-Y}{-X}\right)$	$\theta = -\pi + arctang\left(\frac{-Y}{-X}\right)$
IV	$arctang\left(\frac{-Y}{X}\right)$	$\theta = arctang\left(\frac{-Y}{X}\right)$

Finalmente, para obtener el valor del módulo de la distancia se requiere realizar el cálculo de distancia entre dos puntos, la cual realiza una diferencia entre las coordenadas $(X_{deseada},\ Y_{deseada})$ y $(X_{real},\ Y_{real})$ como se observa en (2.1). Con esta distancia se puede medir la distancia restante que existe entre el robot y el punto final, esto es enviado al

controlador para ir evaluando en cada punto esta distancia y el error respectivo, mismos que cuando tiendan a 0 su valor se considera que el robot ha llegado a la posición final de llegada.

$$D = \sqrt{(X_{deseada} - X_{real})^2 + (Y_{deseada} - Y_{real})^2} \quad (2.1)$$

2.1.2 COMUNICACIÓN ENTRE PYTHON – COPPELIASIM.

El robot NAO simulado requiere la comunicación con el software de simulación CoppeliaSim, sin embargo, para lograr esto se requiere que las instrucciones y algoritmos de control sean realizados mediante otros lenguajes de programación como lo es Python, por esta razón es indispensable tener un sistema de envío y recepción de información entre Python y CoppeliaSim. Para esto se requiere colocar un cuboide en el entorno de la plataforma CoppeliaSim como se observa en la **Figura *2.4***, y ubicarlo en un sitio lo menos visible para que no moleste al usuario, ya que esto únicamente va a permitir albergar el código que enlaza los datos entre estos programas.

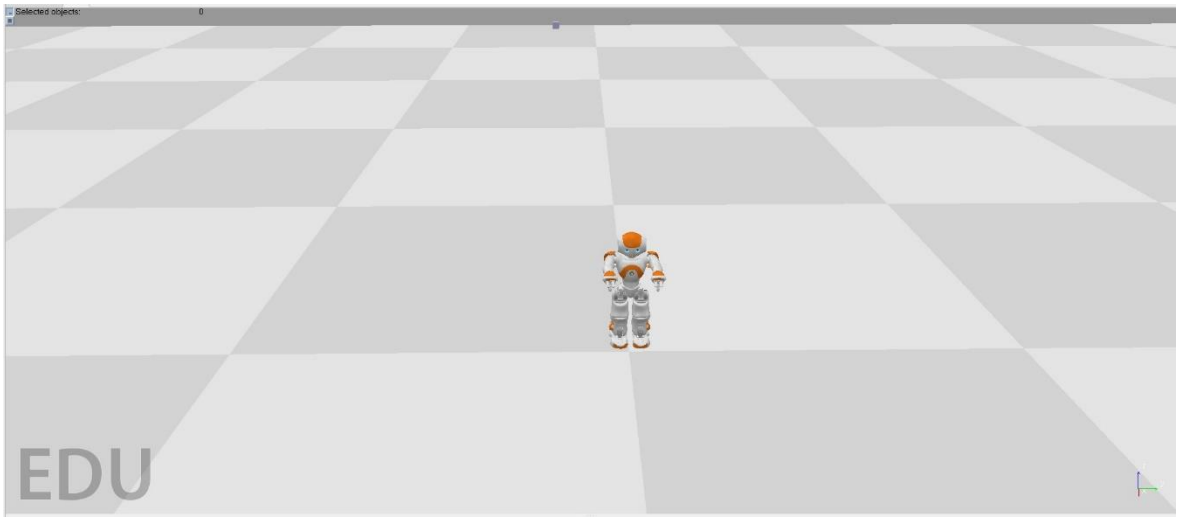

Figura 2.4 Cuboide de comunicación entre CoppeliaSim y Python.

La instrucción que se observa en la **Figura *2.5***, logra establecer la comunicación entre Python y CoppeliaSim, misma que no requiere estar colocada dentro del robot NAO, si no que se debe colocar en la estructura del cuboide de V-rep, de esta forma se establece la API remota para la simulación.

```
simRemoteApi.start(19999)
```

Figura 2.5 Instrucción que permite la comunicación entre CoppeliaSim y Python.

Por otro lado, para lograr el entendimiento por parte de Python se requiere establecer la comunicación mediante una dirección IP y el número de puerto respectivo, como se

observa en la *Figura 2.6*, una vez realizados estos pasos se consigue el envío y recepción de datos, que es indispensable para transmitir la información desde y hacia el controlador.

```
clientID=sim.simxStart('127.0.0.2',19999,True,True,5000,5) # Connect to CoppeliaSim
```

Figura 2.6 Instrucción de Python para envío y recepción de datos por IP.

Una vez logrado el enlace entre ambos programas se requiere la comprobación del enlace entre ellos. Para lograr verificar esta conexión se genera una línea de código que muestra que la conexión fue establecida y generada con éxito. Como se observa en la *Figura 2.7*, la instrucción que comprueba la comunicación muestra un mensaje que indica que se estableció la conexión entre Python y CoppeliaSim.

```
sim.simxAddStatusbarMessage(clientID,'Conexión aceptada',sim.simx_opmode_oneshot)
```

Figura 2.7 Comprobación de conexión establecida.

El controlador diseñado, requiere recibir los parámetros de posición del Robot NAO dentro de CoppeliaSim para tener un lazo cerrado de control. Como se observa en la *Figura 2.8*, para lograr extraer las coordenadas de la posición que entregan los sensores del robot se emplea la instrucción $simGetObjectPosition$, misma que da la posición actual en metros; y se utiliza la instrucción $simxGetObjectOrientation$ para obtener el ángulo de orientación en radianes, sin embargo se lo convierte en grados para ser posteriormente visualizado en dentro de la venta de comandos mientras se ejecuta el script.

```
NAO_Pos= sim.simxGetObjectPosition(clientID, NAO_Handle[1], -1, sim.simx_opmode_streaming)[1]
NAO_orient = sim.simxGetObjectOrientation(clientID, NAO_Handle[1], -1, sim.simx_opmode_streaming)[1]
#NAO_Vel=sim.simxGetObjectVelocity()
NAO_Vel=motionProxy.getRobotVelocity()
NAO_orientDEG=(NAO_orient[2]*180)/pi
aux=[NAO_Pos[0],NAO_Pos[1],NAO_orient[2]]
send_UDP(aux)
print ('Pos NAO: {:.3f} {:.3f} {:.2f}°'.format(NAO_Pos[0],NAO_Pos[1],NAO_orientDEG))
```

Figura 2.8 Instrucciones para obtener la posición y orientación del robot NAO.

Finalmente, en la *Figura 2.9*, se puede observar la función $threading$, misma que es importante para la ejecución de las funciones en paralelo, dentro de las cuales se encuentra la inicialización de los comandos del robot, para así poder ejecutar los hilos o funciones respectivas.

```
#p1 = threading.Thread(target=process)
#p1.start()
p2 = threading.Thread(target=JointControl, args=(clientID, motionProxy, Body,))
p4 = threading.Thread(target=caminata)
p4.start()
p2.start()
#p3=threading.Thread(target=getposicion)
#p3.start()
p4.join()
p2.join()
#p3.join()
#p1.join()
# Como finalizar el hilo
```

Figura 2.9 Función threading para la ejecución de funciones en paralelo.

2.1.3 SDK NAOQI

El script implementado en Python también se encarga de la transmisión de información entre CoppeliaSim y Simulink, ya que los datos generados en el primero se envían hacia Simulink, en este se calcula y ejecuta la acción de control que se envían de nuevo hacia CoppeliaSim para que pueda accionar los actuadores del robot y este interactuar con el entorno diseñado. Python no solo actúa como intermediario entre estos programas, sino que también utiliza instrucciones que permiten conectar las librerías de Choregraphe (Software de Programación NAO) a Python a través del SDK de NAOqi. Esto posibilita la ejecución de funciones como caminar, emitir sonidos, grabar audio, entre otras; de forma más sencilla como si se estuviera programando el robot en su sistema original. Para lograr esta comunicación se requiere la descarga previa de la librería de NAOqi junto con el software de Choregraphe, misma que ejecuta un Proxy a través de una IP y puerto de comunicación de un robot virtual. En la *Figura 2.10* se pueden observar las instrucciones que permiten crear el atributo para obtener las funciones de movimiento y posturas del robot NAO mediante el sistema NAOqi.

```
motionProxy = ALProxy("ALMotion",naoIP, naoPort)
postureProxy = ALProxy("ALRobotPosture", naoIP, naoPort)
```

Figura 2.10 Instrucciones para crear Proxy de posición y postura de NAOqi.

Una vez definidos los atributos proxy de NAOqi, se requiere transmitir los datos de controlador, por lo que se utiliza el recurso proxy $MOVE$ el cual, establece las velocidades lineales en el eje X y Y del plano del robot, además de la velocidad de giro; las dos primeras en m/s y la tercera en rad/s siempre evitando valores mayores a las velocidades máximas que son: *0.09259 m/s* y *0.5808 rad/s.* La forma de envío seleccionada depende del tipo de datos que requiere la salida del controlador diseñado, sin embargo se debe tomar en

cuenta la existencia de una segunda instrucción la cual es $moveTo$, misma que permite recibir los datos en metros y logra mover al robot hacia una ubicación que el usuario haya seleccionado utilizando el controlador PID que viene por default, con el objeto de poder realizar una comparación entre este, y los controladores diseñados de Lyapunov, Redes Neuronales, y efectuados mediante la instrucción $MOVE$.

```
motionProxy.move(DataR[0],0,DataR[1])
#motionProxy.moveToward(DataR[0],0,DataR[1])
```

Figura 2.11 Instrucción para efectuar movimiento en el robot NAO.

2.1.4 COMUNICACIÓN ENTRE SIMULINK Y PYTHON.

Los controladores se diseñan en la herramienta de Matlab llamada Simulink, para lo cual se requiere un sistema de bloques que permitan entregar los datos del controlador hacia Python para posteriormente ejecutar las acciones dentro de CoppeliaSim. Para esto se usan los bloques de comunicación conocidos como UDP, a través de estos se va a enviar la velocidad lineal, velocidad angular y error calculado; se almacenan en un arreglo y se discretizan estos valores con un tiempo de muestro de 0.02s, mismo que es el tiempo en el que el robot NAO responde a una acción, para finalizar se envían estos datos por medio de una dirección IP y el puerto establecido con anterioridad, que serán diferentes a los utilizados para la comunicación con el robot virtual y CoppeliaSim. Como se observa en la *Figura 2.12*, se pueden identificar los bloques que corresponden a la Comunicación para envío y recepción de datos.

Figura 2.12 Bloque de Simulink para envío y recepción de datos por UDP.

Python transmite los datos de la posición en la que se encuentra actualmente el robot obtenido a través de CoppeliaSim, se extraen los datos en un arreglo de tres variables las cuales son (X_{actual}, Y_{actual}) y el ángulo de orientación θ como se observa en la *Figura 2.13*.

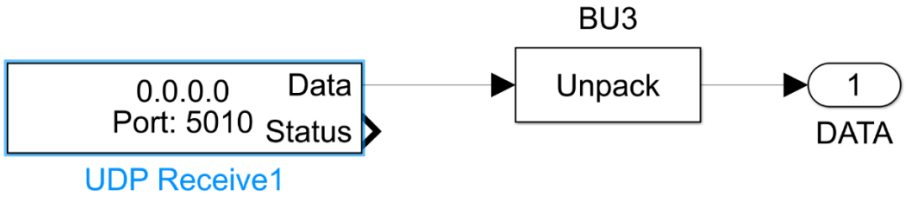

Figura 2.13 Bloque de Recepción de datos de Python hacia Simulink.

La programación de Python requerida para esta comunicación únicamente declara la dirección IP y número de puerto establecidos en Simulink y en el proxy de CoppeliaSim, estas instrucciones se pueden observar en la **Figura *2.14***.

```
def recieve_UDP(size):
    sock=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
    #print 'Objeto creado'
    sock.bind(('127.0.0.3',5285))
    #print 'Objeto "Bindado"'
    aux='>'+('d'*size)
    #print 'Iniciando recepcion con bufer de 1024'
    data, addr = sock.recvfrom(1024) # buffer size is 1024 bytes
    #print 'Mensaje recibido'
    mensaje=struct.unpack(aux, data)
    #print 'desempacando el paquete'
    sock.shutdown(0)
    #print 'Método shutdown REALIZADO'
    sock.close
    #print 'Metodo close REALIZADO'
    return mensaje
```

Figura 2.14 Función en Python para recepción de Datos de Simulink.

Para el caso del envío de información de posición a Simulink, se procede de igual forma, estableciendo la dirección IP y número de puerto declarados en el bloque UDP de Simulink, esta función se muestra en la *Figura 2.15*.

```
def send_UDP(data):
    sock=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
    sock.bind(('127.0.0.1',5011))
    aux='>'+('d'*len(data))
    mensajebody=struct.pack(aux,*data)
    sock.sendto (mensajebody, ('127.0.0.3',5010))
    #sock.shutdown(0)
    #sock.close
```

Figura 2.15 Función de Python para envío de Datos hacia Simulink.

2.2 DISEÑO DE CONTROLADORES

2.2.1 CONTROL DE POSICIÓN BASADO EN LYAPUNOV

Para el diseño del controlador se requiere considerar al robot NAO como una partícula móvil, en donde su cinemática y dinámica están establecidas en la librería de NAOqi. Por esta razón se diseña el controlador basándose en el modelo más básico, es decir considerando que es un robot uniciclo, para utilizar su cinemática y lograr obtener las ecuaciones que describen el movimiento de una caja negra y una partícula móvil como se observa en la *Figura 2.16*, obteniendo la velocidad lineal en coordenadas cartesianas y la velocidad angular como se indican en las ecuaciones (2.2), (2.3), (2.4).

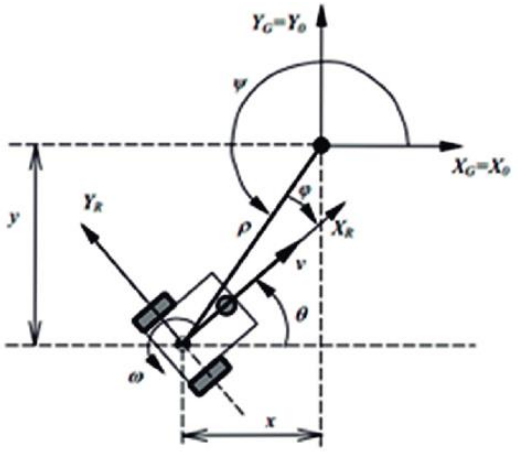

Figura 2.16 Cinemática del Robot Uniciclo *[43]*.

$$\dot{x} = u\dot{\cos}(\varphi) \tag{2.2}$$

$$\dot{y} = u\dot{\sin}(\varphi) \tag{2.3}$$

$$\dot{\varphi} = \omega \tag{2.4}$$

Para el diseño del controlador se transforman las ecuaciones de la cinemática del robot a coordenadas polares, y se obtienen las expresiones de ángulo de orientación, distancia hacia el punto deseado, velocidad lineal y angular mismas que están representadas en las ecuaciones (2.5), (2.6) y (2.7).

$$\dot{p} = -\dot{u}\cos(\alpha) \tag{2.5}$$

$$\dot{\alpha} = -w + \dot{\left(\frac{u}{p}\right)}\sin(\alpha) \tag{2.6}$$

$$\dot{\psi} = \left(\frac{u}{p}\right)\sin(\alpha) \tag{2.7}$$

Una vez establecidas las ecuaciones que representan el modelo cinemático se debe seleccionar la función de Lyapunov que consigue llevar al sistema hacia la estabilidad, la

función que se observa en (2.5) debe cumplir con el criterio establecido por las siguientes reglas:

- La primera ley se cumple debido a que las funciones sin y cos son continuas en todos sus puntos (ecuación 2.5).
- La segunda ley evalúa $x = 0$ en (2.5) lo que resulta en que la función es cero siempre que x sea cero.
- La tercera ley cumple la condición de tener un valor positivo para cualquier valor diferente de 0, debido a que la función de Lyapunov escogida es un polinomio de grado 2.

Al estar diseñando un controlador de trayectoria, en las ecuaciones (2.5) - (2.7) se identifican que los estados que se requieren reducir son: la distancia hacia el punto deseado (ρ), y la diferencia entre la orientación actual del robot y el vector hacia el siguiente punto (α).

$$\dot{V}(\bar{x}) = \frac{1}{2} x^T Q = V(p, \propto) \tag{2.8}$$

$$Q = \begin{bmatrix} q1 & 0 \\ 0 & 1 \end{bmatrix} \ donde \ \ q1 > 0 \tag{2.9}$$

$$x = [p, \alpha]^T \tag{2.10}$$

Una vez comprobadas las leyes antes mencionadas, se procede a realizar la comprobación de la cuarta ley de Lyapunov, misma que indica que la derivada de la función escogida (2.8) debe ser menor o igual a cero. Se procede a desarrollar la ecuación (2.11) hasta llegar a (2.14) que se divide en dos funciones $\dot{V}1$ y $\dot{V}2$.

$$\dot{V}(p, \propto) = x^T Q \dot{x} \tag{2.11}$$

$$\dot{V}(p, \propto) = [p \quad \alpha] \begin{bmatrix} q1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{p} \\ \dot{\alpha} \end{bmatrix} \tag{2.12}$$

$$\dot{V}(p, \propto) = p\dot{p} + \alpha \, \dot{\alpha} \tag{2.13}$$

$$\dot{V}\ (p, \propto) = \dot{V}1 + \dot{V}2 \tag{2.14}$$

Se selecciona la ecuación (2.14) para desarrollarla, tomando en cuenta que las variables controladas son u y w, mismas que producirán la estabilidad del sistema. La variable $v1'$ requiere que el valor de u modifique la función y se obtenga la ecuación (2.16). Se pueden obtener varias soluciones para la misma función por lo que el desarrollo que se propone no es el único que garantice que la función consiga la estabilidad.

$$\dot{V}1 = \rho\,\dot{\rho} = \rho[-u\cos(\alpha)] \tag{2.15}$$

$$\dot{V}2 = \alpha\,\dot{\alpha} = \alpha\left[-\omega + \left(\frac{u}{\rho}\sin(\alpha)\right)\right] \tag{2.16}$$

Una vez ya establecidas las funciones candidatas de Lyapunov, se requiere asegurar que cada una cumpla las tres leyes antes mencionadas, para lograrlo se analizaran V_1 y V_2 por separado.

Para V_1, se asume que la velocidad lineal se comporta de acuerdo con la ecuación (2.17) [37].

$$u = k_u \tanh\rho\cos\alpha \tag{2.17}$$

$$\dot{V}_1 = -k_u\rho\tanh\rho\cos^2\alpha \tag{2.18}$$

Para V_2, se parte de (2.16) reemplazando (2.17) en esta para así obtener una expresión de ω con la que V cumpla las leyes.

$$\dot{V}_2 = \alpha\left[-\omega + \frac{k_u\tanh\rho\cos\alpha\sin\alpha}{\rho}\right] \tag{2.19}$$

Se asume:

$$\omega(\rho,\alpha) = k_\omega\alpha + \frac{k_u\tanh\rho\cos\alpha\sin\alpha}{\rho} \tag{2.20}$$

Con (2.20) se desarrolla (2.19) hasta obtener:

$$\dot{V}_2 = -k_\omega\alpha^2 \tag{2.21}$$

Con las ecuaciones (2.18) y (2.21) es posible unirlas en (2.14) de tal manera que resulta en (2.22):

$$\dot{V}\,(p,\propto) = \dot{V}1 + \dot{V}2 = -k_u\rho\tanh\rho\cos^2\alpha - k_\omega\alpha^2\,, donde\ k_u > 0\ y\ k_\omega > 0 \tag{2.22}$$

Se debe tomar en consideración que para la simulación del controlador en función de $(u,\ w)$ en el software Simulink, el módulo del error ρ debe ser calculado obteniendo las coordenadas deseadas $(x_{deseada}, y_{deseada})$ proporcionadas por la trayectoria, menos las coordenadas reales del robot NAO (x_{real}, y_{real}) obtenidas de la realimentación de los sensores que es entregado por CoppeliaSim como se observa en la ecuación (2.23).

$$\rho = \sqrt{(x_{deseada} - x_{real})^2 + (y_{deseada} - y_{real})^2} \tag{2.23}$$

El ángulo α sigue el mismo principio en donde se realiza la diferencia del ángulo deseado $\alpha_{deseado}$ menos el ángulo real del robot α_{real}. Para obtener estos datos, se utiliza la función $atan2$ para obtener el ángulo del arco tangente desde el eje x, es decir 0 a $180°$ en sentido antihorarios y 0 a $-180°$ en sentido horario para el $\alpha_{deseado}$; mientras que el α_{real} es la realimentación entregada por los sensores simulados del robot NAO como se observa en las expresiones (2.24) y (2.25).

$$\alpha = \alpha_{deseado} - \alpha_{real} \tag{2.24}$$

$$\alpha = \text{atan2}\left(\frac{y_{deseado} - y_{real}}{x_{deseado} - x_{real}}\right) - \alpha_{real} \tag{2.25}$$

2.2.2 CONTROL DE SEGUIMIENTO DE TRAYECTORIA POR REDES NEURONALES

Para realizar el control de seguimiento de trayectoria mediante el uso de redes neuronales se ocupa una red de tipo propagación hacia atrás o BackPropagation que es un subtipo de las redes de tipo FeedForward. Este tipo de redes están conformadas por una capa de entrada, una capa de salida y n capas ocultas o intermedias. Para este caso, se ha estructurado la red neuronal que se observa en la Figura 2.17, misma que consta con 10 capas de entrada, 2 capas intermedias de 20 neuronas cada una y dos capas de salida.

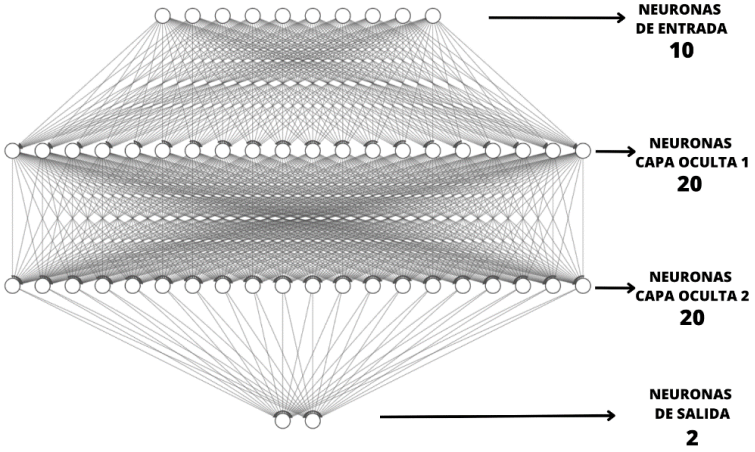

Figura 2.17 Red Neuronal para Control de Posición.

La capa de entrada no realiza ninguna operación como se muestra en el diagrama de bloques de la *Figura 2.18*, únicamente distribuye cada una de las entradas X_k hacia los pesos W_{jk} que se encuentran en la primera capa oculta. En cada una de las neuronas la capa oculta 1, se realiza primero la suma de entradas ponderadas como se observa en la ecuación 2.26.

$$h_{jk} = \sum W_{jk} X_k \tag{2.26}$$

Figura 2.18 Operaciones de la capa de entrada de la red neuronal.

Se aplica una función de transferencia no lineal también llamada función de activación dentro de cada capa oculta como se observa en la *Figura 2.19*, para cada uno de los elementos de salida de las 20 neuronas correspondientes a las dos capas ocultas, en este caso la función utilizada es la sigmoide que está representada por la ecuación 2.27.

$$F(z) = \frac{1}{1+\exp(-z)} \tag{2.27}$$

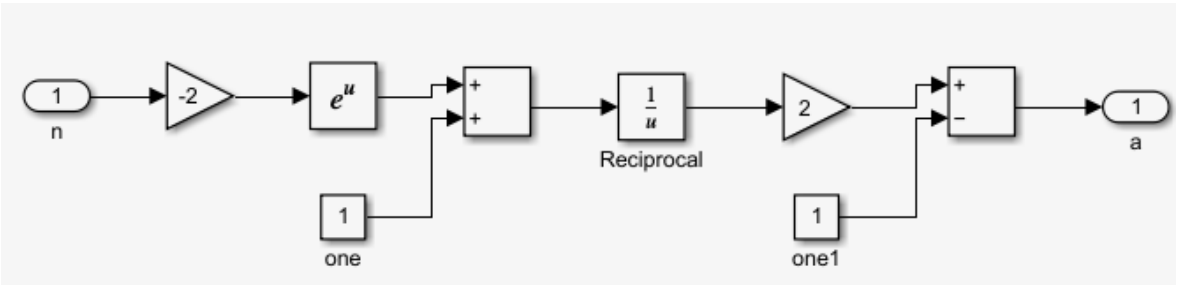

Figura 2.19 Función de activación de las capas ocultas

Una vez calculados los valores de la segunda capa oculta, pasan hacia la capa de salida como se observa en la *Figura 2.20*, en donde se obtiene la salida de la red neuronal multiplicando los valores obtenidos por los pesos como indica la ecuación 2.28.

$$h_i = \sum_j W_{ij} X_j = \sum_j W_{ij}\, f\left(\sum_k W_{jk} X_k\right) \tag{2.28}$$

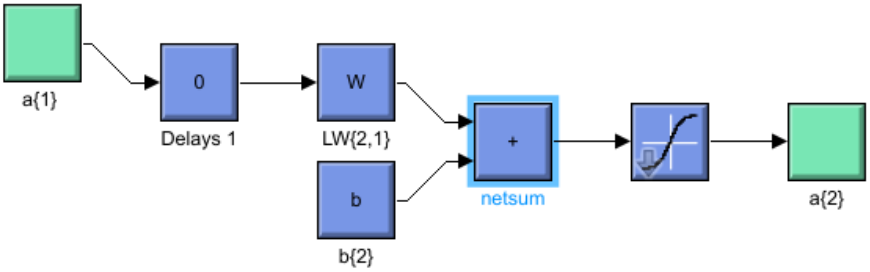

Figura 2.20 Operaciones para obtener valores de salida en las capas.

Después se obtener la estructura de la red neuronal que se ocupa en este caso, se requiere el entrenamiento de la red, mismo que consiste en ajustar los pesos W_{ij} para que el error entre la salida deseada y la salida de la red tenga un valor mínimo. El proceso consiste en

que a partir de la entrada se computa el valor en cada neurona para que la salida $Y_i(X_n)$ tenga un valor como se observa en la *Figura 2.21*. Este valor de salida se diferencia con el valor deseado y se obtiene el error que posteriormente permite el ajuste de los pesos, primero en la capa de salida, seguido de la capa oculta 2 para finalizar en la capa oculta 1.

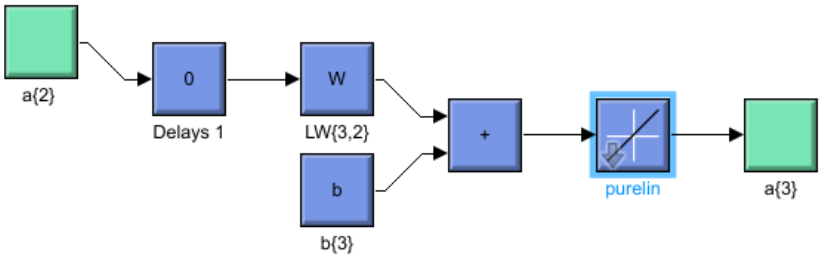

Figura 2.21 Calculo de Valor de Salida de la Red Neuronal

Para lograr obtener estos valores, se requiere tener una base de datos de entrenamiento previo. Estos datos son recopilados de la información del controlador de Lyapunov, en donde se realizó las pruebas necesarias para obtener una base de datos de los valores necesarios para entrenar esta red.

Las señales de entrada son los errores de posición actual hasta el error de posición en tiempo $t-4$, así como los errores de orientación del estado actual hasta el tiempo $t-4$, obteniendo las 10 entradas mencionadas en la estructura, para esto se ocupan valores de memoria que se van almacenando y desplazando continuamente como se observa en la *Figura 2.22*. Finalmente, la capa de salida de la red neuronal tiene dos neuronas mismas que no realizan ningún cálculo matemático y entregan los valores de velocidad lineal u y velocidad angular w necesarios para el control del robot.

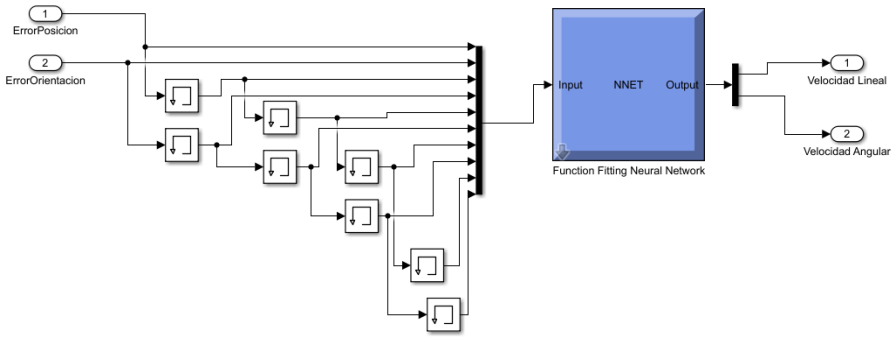

Figura 2.22 Estructura de la Red Neuronal Diseñada Simulink.

2.3 GENERACIÓN DE TRAYECTORIAS

Las trayectorias del robot que se van a generar para este proyecto son tres: círculo, cuadrado y lemniscata. Estas están representadas por el diagrama de bloques de Simulink que se observa en la Figura *2.23*.

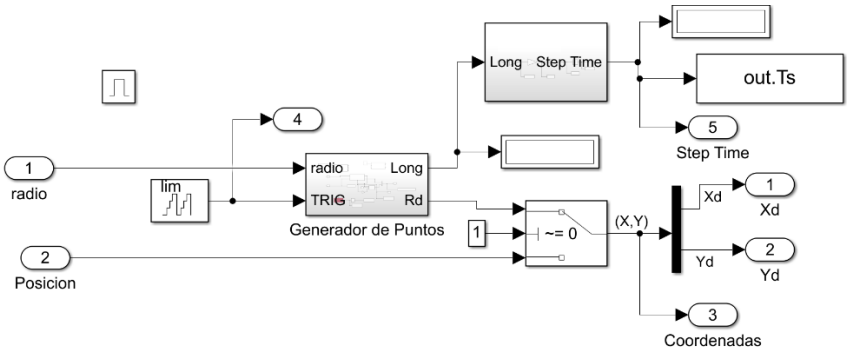

Figura 2.23 Diagrama de Bloque para la Generación de Trayectorias.

Para lograr generar cualquiera de las trayectorias, el bloque generador de puntos requiere de un bloque de señal, mismo que indica el tiempo respectivo para generar el siguiente punto de trayectoria, en este caso como se observa en la Figura 2.24 es de 28 segundos.

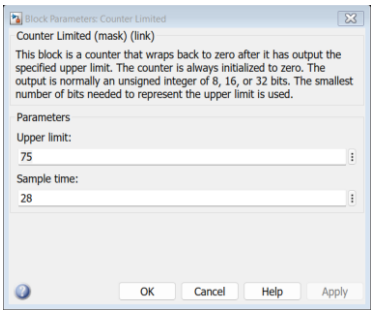

Figura 2.24 Tiempo de Generación de Puntos de Trayectoria.

Dentro del sistema de generación de puntos se puede seleccionar el tamaño de la trayectoria a realizarse, valor que se encuentra en la variable $radio$. Para el caso del círculo esta variable indica el radio, en el cuadrado es el valor del lado y para el caso de la lemniscata es la distancia entre los centros que forman cada círculo, como se observa en la *Figura 2.25*.

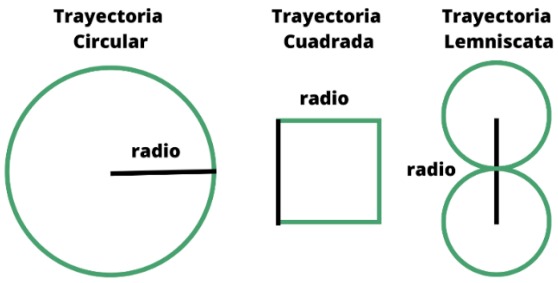

Figura 2.25. Tamaño de la Trayectoria para las diferentes formas.

Una vez obtenidos los datos para la generación de las trayectorias se procede a dibujar los puntos utilizando ecuaciones parametrizadas para cada una de estas y como se presenta en los siguientes puntos.

2.3.1 TRAYECTORIA CIRCULAR

Para generar la trayectoria circular se ha implementado el diagrama de bloques que se muestra en la Figura 2.26, observando que se transforma la señal de entrada de radio en funciones paramétricas de seno y coseno dadas por las ecuaciones (2.29). Las señales de salida son las coordenadas de x, y correspondiente a cada punto que se recorre.

$$\begin{cases} x = x_0 + r\cos t \\ y = y_0 + r\sin t \end{cases} \tag{2.29}$$

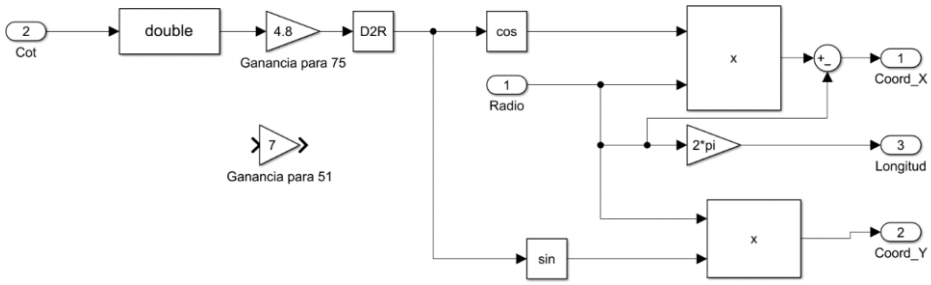

Figura 2.26 Diagrama de Bloques Generación de Trayectoria Circular.

2.3.2 TRAYECTORIA CUADRADA

Para generar la trayectoria cuadrada se implementa el diagrama de bloques que se muestra en la Figura *2.27*, observando que se transforma la señal de entrada de radio en funciones paramétricas, mismas que generan cada punto en secciones del trazado de un círculo como se muestra en el código de la Figura *2.28*. Las señales de salida son las coordenadas de x, y correspondiente a cada punto que traza la trayectoria.

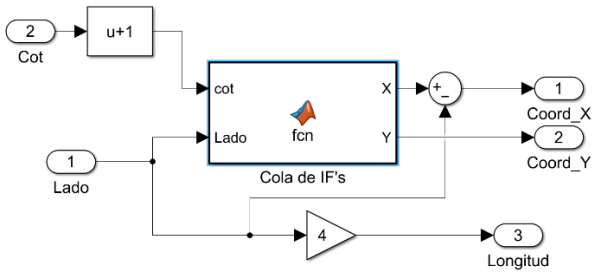

Figura 2.27 Diagrama de Bloques de la Generación de Trayectoria Cuadrada.

```
if aux1<=pi/4
    aux_x=a;
    aux_y=a*tan(aux1);
elseif (aux1>pi/4)&&(aux1<=pi/2)
    aux1=aux1-(pi/4);
    aux_x=a*(1-tan(aux1));
    aux_y=a;
elseif (aux1>pi/2)&&(aux1<=3*pi/4)
    aux_x=-a*tan(aux1-pi/2);
    aux_y=a;
elseif (aux1>3*pi/4)&&(aux1<=pi)
    aux_x=-a;
    aux_y=-a*tan(aux1);
elseif (aux1>pi)&&(aux1<=5*pi/4)
    aux_x=-a;
    aux_y=-a*tan(aux1);
elseif (aux1>5*pi/4)&&(aux1<=3*pi/2)
    aux_x=-a*(1-tan(aux1-5*pi/4));
    aux_y=-a;
elseif (aux1>3*pi/2)&&(aux1<=7*pi/4)
    aux_x=a*tan(aux1-3*pi/2);
    aux_y=-a;
elseif aux1>7*pi/4
    aux_x=a;
    aux_y=a*tan(aux1);
end
```

Figura 2.28 Código para Generar la Trayectoria Cuadrada.

2.3.3 TRAYECTORIA LEMNISCATA

Para generar la trayectoria cuadrada se implementa el diagrama de bloques que se muestra en la Figura 2.29, observando que se transforma la señal de entrada de radio en las ecuaciones paramétricas que se muestra en la ecuación 2.30, mismas que se encuentran en función de seno y coseno. Las señales de salida son las coordenadas de x, y correspondiente a cada punto que traza la trayectoria.

$$\begin{cases} x = d\sqrt{2}\dfrac{\sin\theta}{1+\cos^2\theta} \\ y = d\sqrt{2}\dfrac{\sin\theta\cos\theta}{1+\cos^2\theta} \end{cases} \tag{2.30}$$

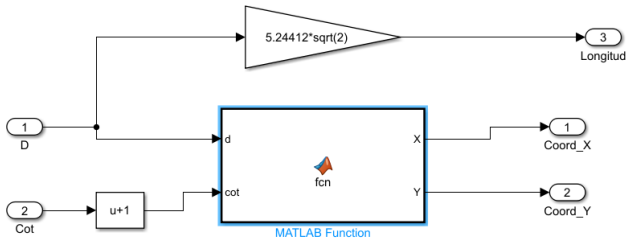

Figura 2.29 Diagrama de Bloques de la Trayectoria Lemniscata

2.4 SIMULACIÓN DE LOS CONTROLADORES

2.4.1 SIMULACIÓN DEL CONTROLADOR DE LYAPUNOV

Una vez diseñado el controlador y generada la trayectoria deseada, se procede a realizar la simulación del de Matlab / Simulink del controlador de Lyapunov. Para este controlador se realiza el cálculo del error como se muestra en la Figura 2.30, en donde se toman los valores de x, y actual y deseada; se calcula la distancia y mediante la función $atan2$ se

calcula el ángulo deseado y el error angular. Se toma en cuenta que se realiza el cambio de formatos de los ángulos de $[-180°, 180°]$ a $[0°, 360°]$.

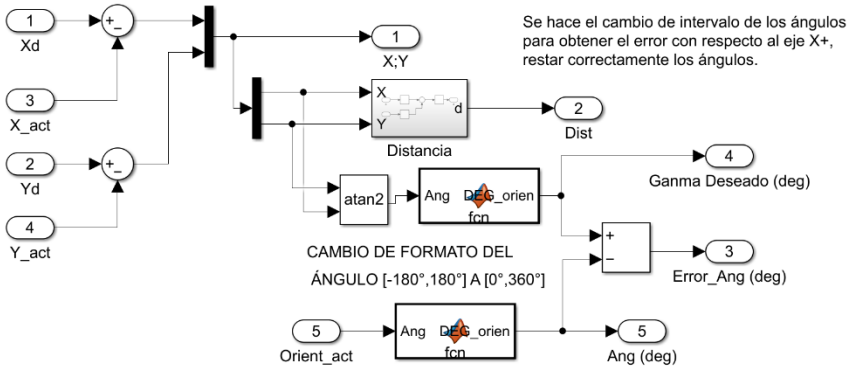

Figura 2.30 Diagrama de Bloques del Cálculo del Error.

Después de calculado los errores de distancia y angular, el sistema ingresa las variables al bloque de controladores que se muestra en la Figura 2.31.

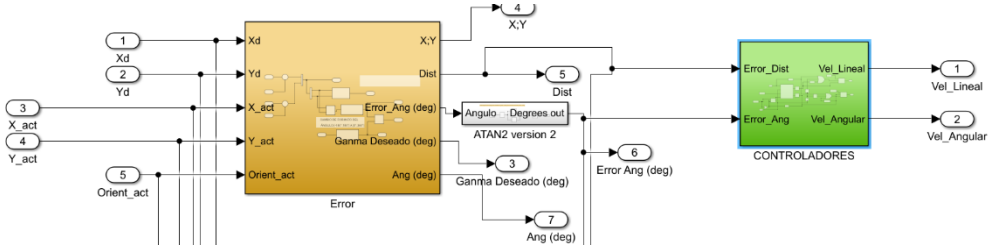

Figura 2.31 Diagrama de Bloques de Control Lyapunov.

Estas variables son procesadas siguiendo el diagrama de control que se muestra en la Figura 2.32, en se limita mediante un saturador los valores tanto de u y w para que no supere los valores máximos de velocidades que soporta el robot. Las salidas de este bloque son la Velocidad Lineal y la Velocidad Angular.

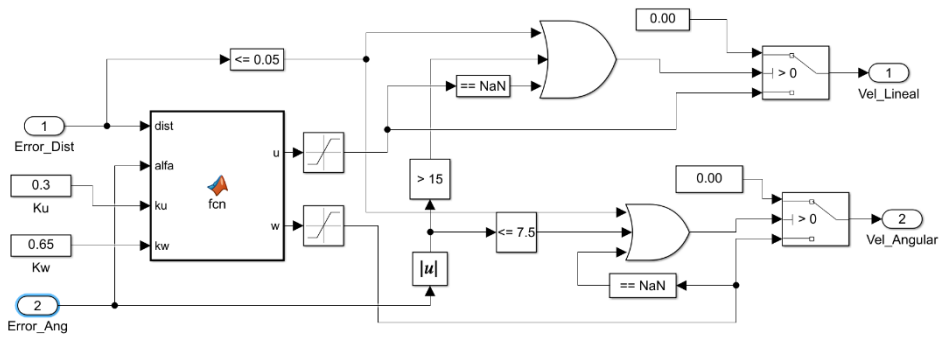

Figura 2.32 Diagrama de Control Lyapunov.

Una vez implementado el controlador se procede a realizar las pruebas de funcionamiento para poder comprobar el comportamiento del controlador, para esto se colocó los valores de $radio = 1.5$, $Ku = 0.3$ y $Kw = 0.65$ para generar la trayectoria cuadrada, obteniendo la repuesta que se observa en la *Figura 2.33*, en donde se verifica que tiene una excelente respuesta a partir de su salida del punto inicial.

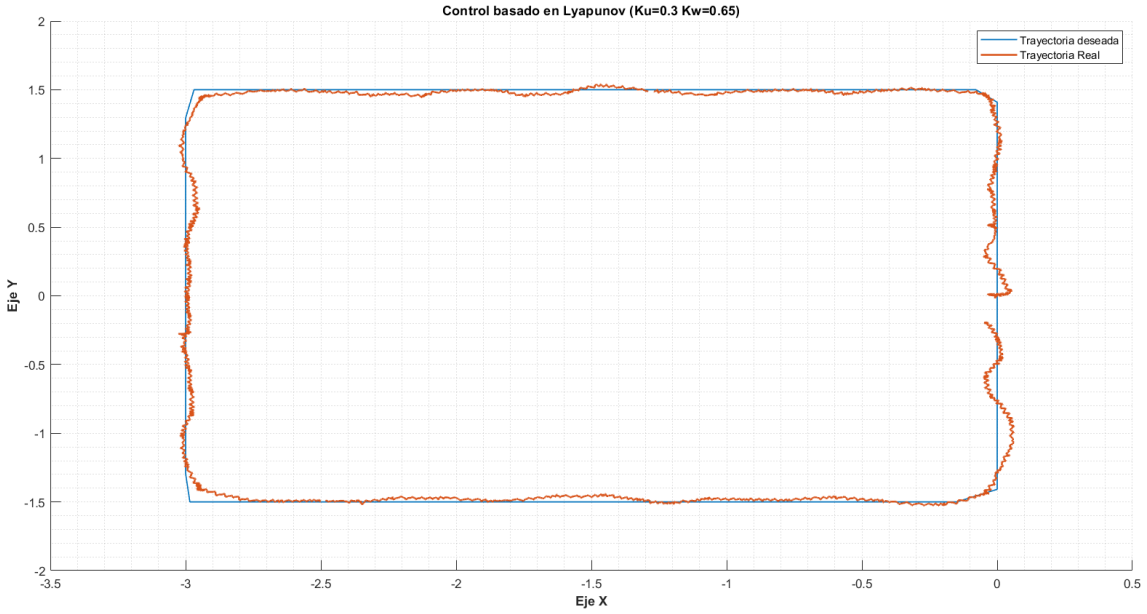

Figura 2.33 Respuesta de Trayectoria Cuadrada de Controlador Lyapunov.

2.4.2 SIMULACIÓN DEL CONTROLADOR CON REDES NEURONALES

El controlador con Redes Neuronales al igual que el Lyapunov, utiliza el mismo diagrama que se observa en la Figura 2.30 para el cálculo del Error de Posición y Error de Orientación. Una vez obtenidos los valores, se envían los datos al bloque Red Control Nao, mismo que se observa en la Figura *2.34*. y el cual contiene la red neuronal diseñada para el sistema.

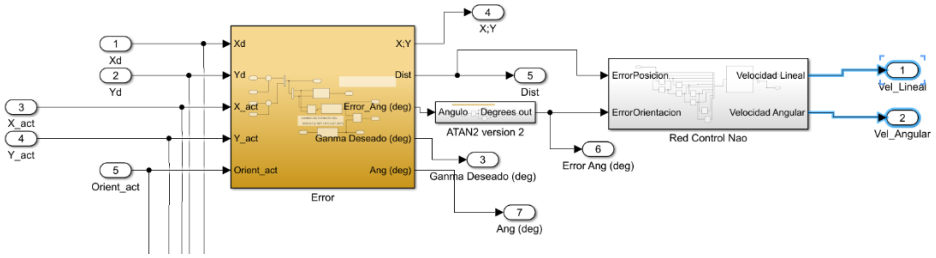

Figura 2.34 Diagrama de Bloques para Control con Redes Neuronales.

Como se menciona anteriormente, la red neuronal está entrenada con los datos de Error Posición, Error Orientación, Velocidad Lineal y Velocidad Angular obtenidas del controlador

de tipo Lyapunov, mismos que se exportan hacia el sistema con los comandos que se muestran en la Figura *2.35*.

```
files=dir('Senales de Control');
datal=0;

for k=1:numel(files)
    fileName=files(k).name;
    if endsWith(fileName,'.mat')
        load("Senales de Control/"+fileName);
        datal=datal+ length(uwda);
    end
end
```

Figura 2.35 Código para Obtener los datos de entrenamiento de la red.

Una vez obtenidos los datos de entrenamiento se ajustan los vectores correspondientes para obtener a partir de estos, las 10 entradas y las dos salidas que ingresan a la red neuronal, eso se obtiene con el código que se muestra en la Figura *2.36*.

```
for k=1:numel(files)
    fileName=files(k).name;
    if endsWith(fileName,'.mat')
        load("Senales de Control/"+fileName);
        entrada=zeros(10,1);
        salida=zeros(2,1);
        for i=1:length(uwda)
            columnaUwda=uwda(:,i);
            entrada=circshift(entrada,2);
            entrada(1)=columnaUwda(4);
            entrada(2)=columnaUwda(5);

            salida(1)=columnaUwda(2);
            salida(2)=columnaUwda(3);

            entradasRed(:,i+iAnt)=entrada;
            salidasRed(:,i+iAnt)=salida;
        end

    iAnt=iAnt+i;

    end
end
```

Figura 2.36 Código para separar datos de entrenamiento.

Después de obtenidos los datos de entrenamiento se procede a agregarlos dentro de la red neuronal diseñada mediante el comando $fitnet$ y se la entrena mediante el comando $train$ como se observa en **Figura *2.37***.

```
net = fitnet([20,20]);
net = train(net,entradasRed,salidasRed);
```

Figura 2.37 Código para entrenar la red neuronal.

Para que la red pueda calcular los valores de Velocidad Lineal y Velocidad Angular requeridos por el robot en el tiempo t, se utilizan los bloques de memoria que se observa en la **Figura 2.38**. Estos contienen la información de Error de Posición $Ep(t), Ep(t-1), Ep(t-2), Ep(t-3), Ep(t-4)$ y Error de Orientación $Eo(t), Eo(t-1), Eo(t-2), Eo(t-3), Eo(t-4)$.

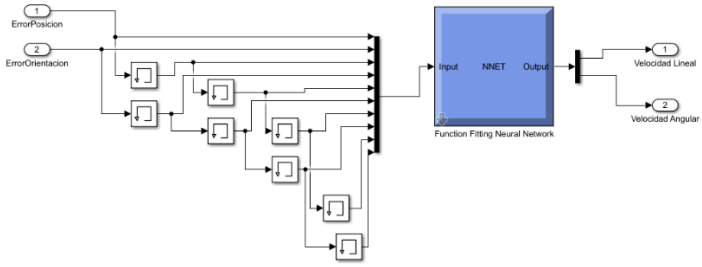

Figura 2.38 Diagrama de Bloques de la Red Neuronal para el Control del Robot

Dentro del bloque $Funtion\ Fitting\ Neural\ Network$, se encuentran el conjunto de neuronas $a(1)$ y $a(2)$, las cuales se conectan a las capas $Layer\ 2$ y $Layer\ 3$ respectivamente tal y como se observa en la *Figura 2.39*; dentro de las cuales se encuentran las operaciones requeridas por el proceso para obtener las salidas correspondientes de velocidad lineal y angular en el instante, calculando el valor de los pesos necesarios a partir de los datos de entrenamiento mencionados anteriormente.

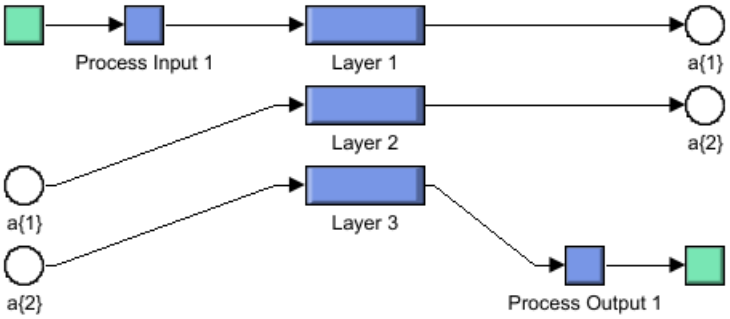

Figura 2.39 Diagrama de bloques del Controlador Neuronal Simulink.

Finalmente, los valores de las memorias se van actualizando, desplazándose a las posiciones $t \to t-1, t-1 \to t-2, t-2 \to t-3, t-3 \to t-4$ para volver a calcular el valor actual necesario de salida de la red neuronal. Para comprobar el funcionamiento del controlador, se realiza la prueba de seguimiento de trayectoria cuadrada, con valor de $lado = 3$, observando que tiene un comportamiento muy parecido al controlador de Lyapunov, sin embargo en el desplazamiento de $(-3, 1.5)$ a $(-3, -1.5)$ poseen ligeras variaciones con respecto a la trayectoria deseada como se observa en la **Figura 2.40**.

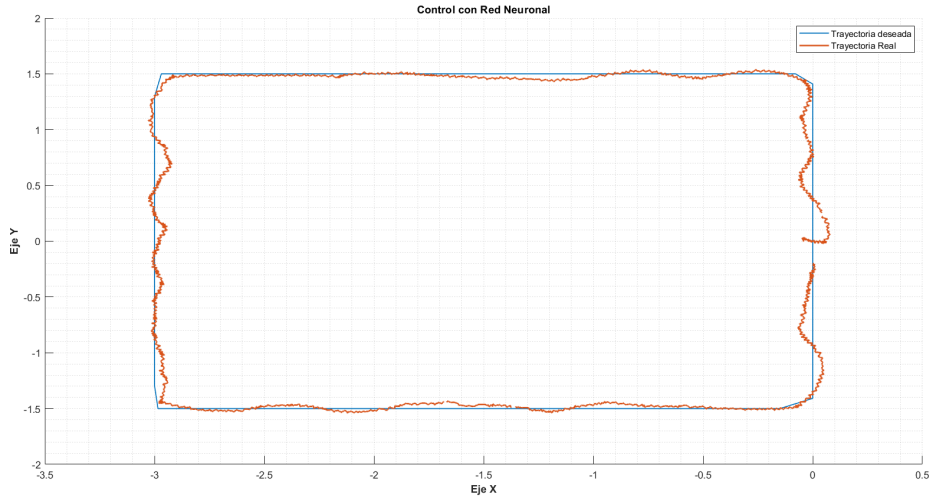

Figura 2.40. Respuesta de Trayectoria Cuadrada del Controlador Neuronal.

2.5 LÓGICA DE PROGRAMACIÓN

2.5.1 DIAGRAMA DE FLUJO DEL PROGRAMA

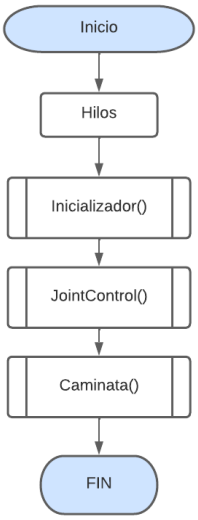

Figura 2.41 Diagrama de Flujo del Programa Principal

2.5.2 DIAGRAMA DE FLUJO DE LA FUNCIÓN INICIALIZADOR

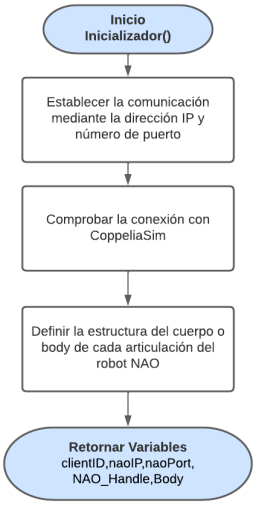

Figura 2.42 Diagrama de Flujo de la Función Inicializador.

2.5.3 DIAGRAMA DE FLUJO DE LA FUNCIÓN JOINTCONTROL

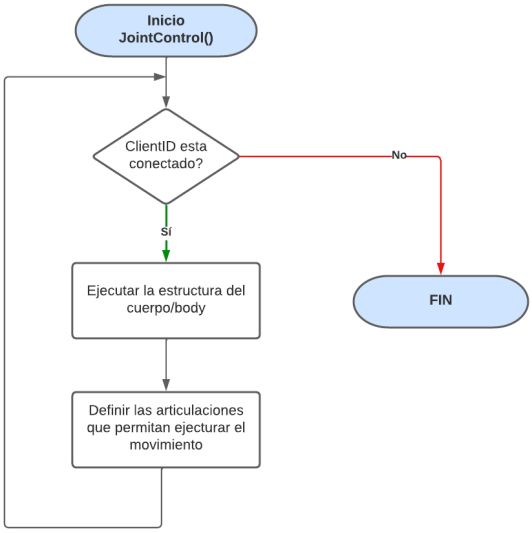

Figura 2.43 Diagrama de Flujo de Función JointControl.

2.5.4 DIAGRAMA DE FLUJO DE LA FUNCIÓN CAMINATA

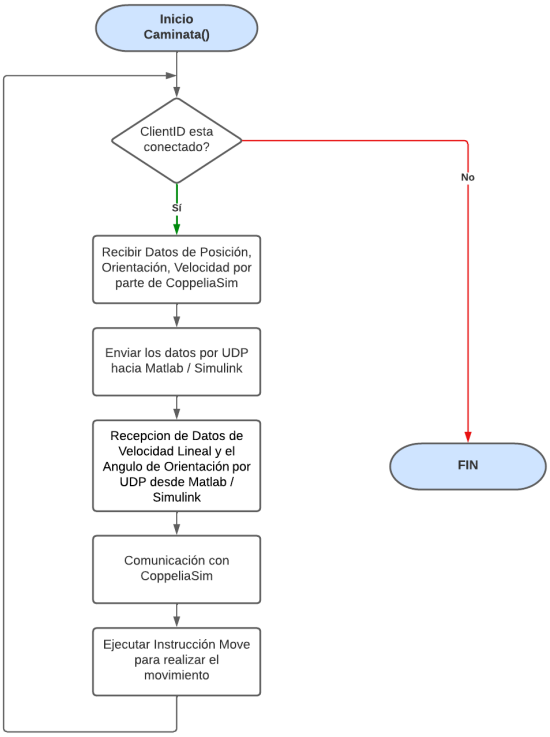

Figura 2.44 Diagrama de Flujo de Función Caminata

2.5.5 DIAGRAMA DE FLUJO DEL CONTROLADOR CON REDES NEURONALES

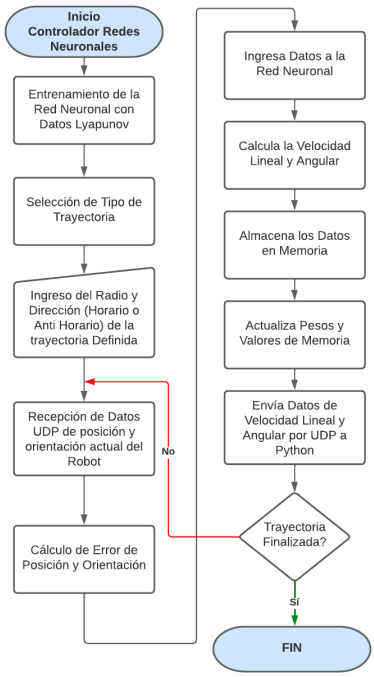

Figura 2.45 Diagrama de Flujo del Controlador de Redes Neuronales

3 RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

3.1 RESULTADOS

3.1.1 CONTROLADOR PID

3.1.1.1 Trayectoria Circular PID

La prueba de seguimiento de trayectoria circular del controlador PID se realiza con los siguientes parámetros: $Pu = 2.1$, $Pw = 1.9$ y $radio = 1.5$. Se puede observar en la Figura 3. que la respuesta del controlador PID tiene problemas para controlar la trayectoria en los puntos $(0, 0)$ hasta llegar al punto 0.5 en y (**Figura 3.2**), posteriormente en las coordenadas de -2.5 a -3 en el eje x (Figura 3.) de igual forma tiene problemas al ejecutar el control.

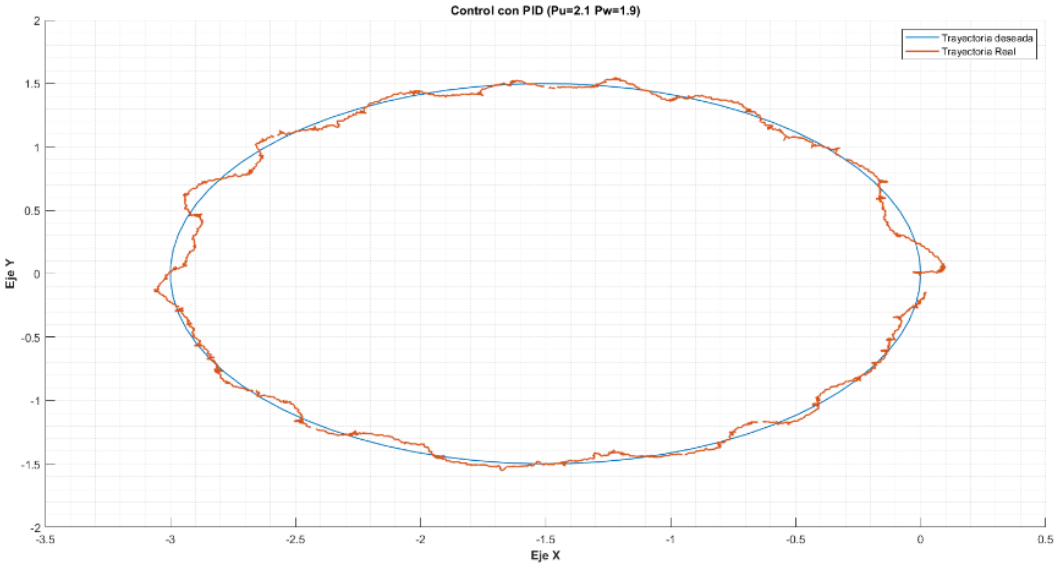

Figura 3.1 Respuesta de Trayectoria Circular Control PID

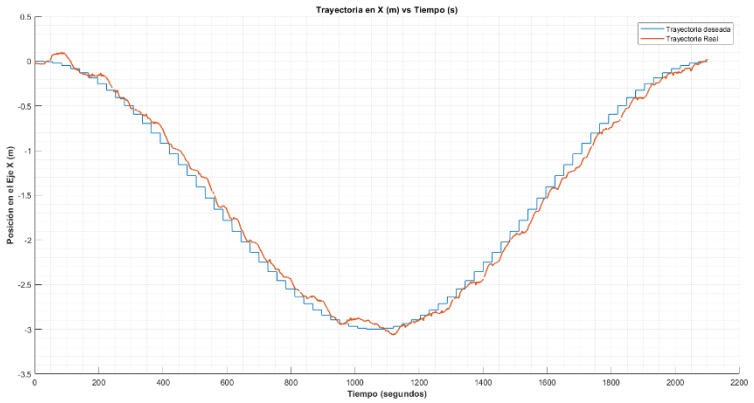

Figura 3.2 Respuesta en eje "x" Trayectoria Circular Control PID.

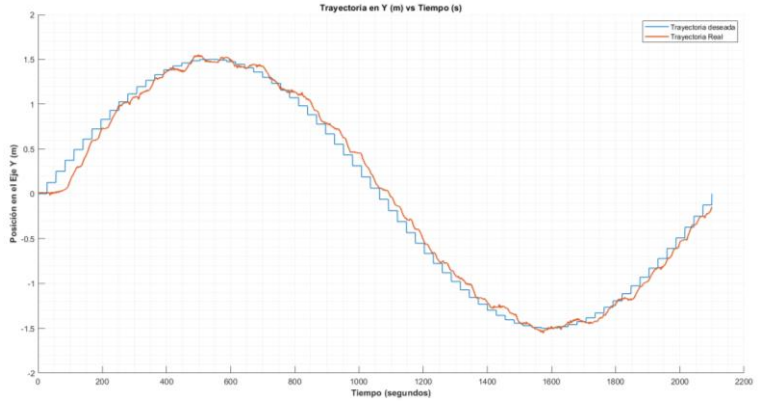

Figura 3.3 Respuesta en eje "y" Trayectoria Circular Control PID.

3.1.1.2 Trayectoria Cuadrada PID

La prueba de seguimiento de trayectoria cuadrada del controlador PID se realiza con los siguientes parámetros: $Pu = 2.1$, $Pw = 1.9$ y $lado = 3$. Se puede observar en la Figura 3.5 Respuesta en eje "x" Trayectoria Cuadrada Control PID que la respuesta del controlador PID tiene problemas para controlar la trayectoria en los puntos $(0, 0)$ hasta llegar al punto $(0, 1.5)$, también se encuentran errores significativos en las posiciones de $(-3, 1.5)$ a $(-3, 0)$ (). Posteriormente en las coordenadas de -0 a -1.5 y -3 a -2.5 en el eje x (Figura 3.5) de igual forma tiene problemas al ejecutar el control.

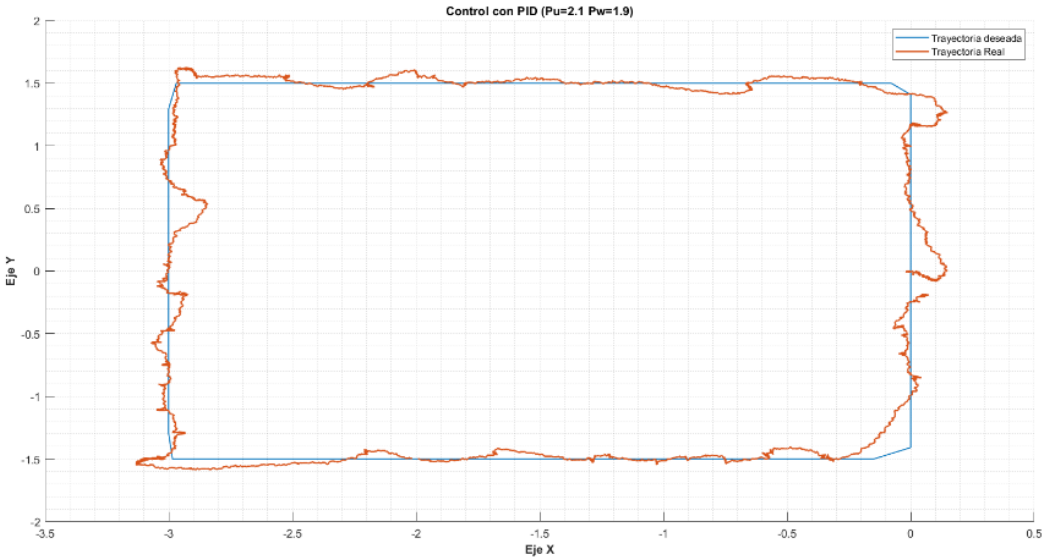

Figura 3.4 Respuesta de Trayectoria Cuadrada del Control PID

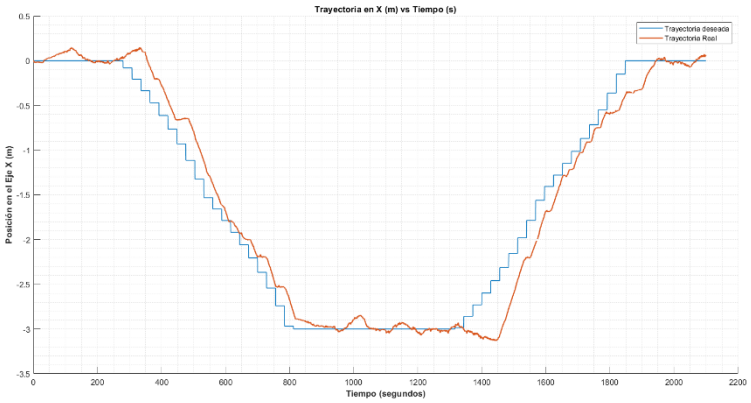

Figura 3.5 Respuesta en eje "x" Trayectoria Cuadrada Control PID

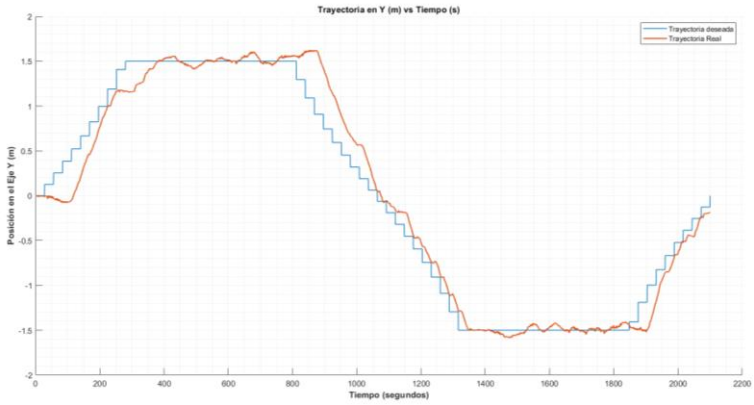

Figura 3.6 Respuesta en eje "y" Trayectoria Cuadrada Control PID

3.1.1.3 Trayectoria Lemniscata PID

La prueba de seguimiento de trayectoria lemniscata del controlador PID se realiza con los siguientes parámetros: $Pu = 2.1$, $Pw = 1.9$ y $radio = 3$. Se puede observar en la Figura 3.8 que la respuesta del controlador PID tiene problemas para controlar la trayectoria en los puntos en los que la trayectoria cambia especialmente en el eje x (Figura *3.9*), por otro lado, en el eje y se tiene una respuesta mucho más estable (Figura *3.9*).

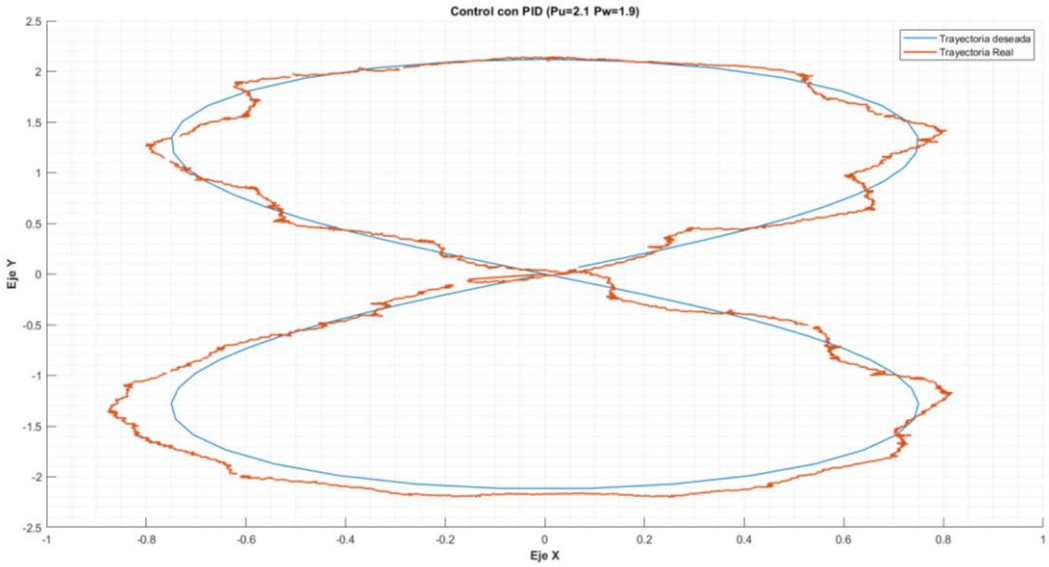

Figura 3.7 Respuesta Trayectoria Lemniscata del Control PID

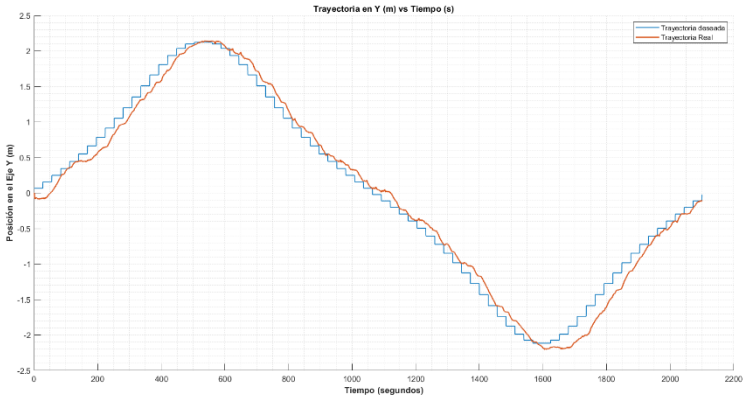

Figura 3.8 Respuesta en eje "y" Trayectoria Lemniscata Control PID.

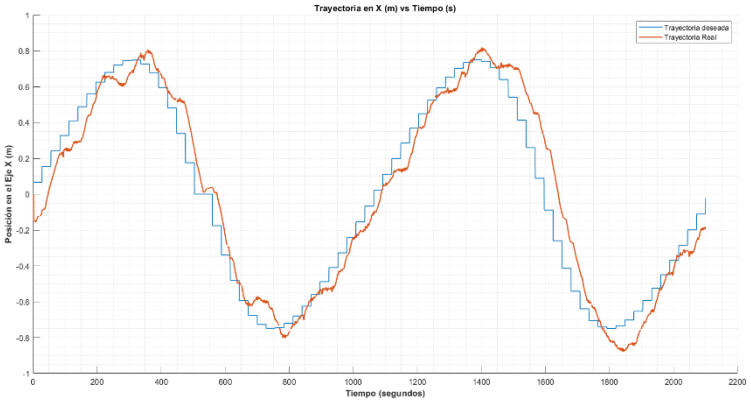

Figura 3.9 Respuesta en eje "x" Trayectoria Lemniscata Control PID.

3.1.2 CONTROLADO LYAPUNOV

3.1.2.1 Trayectoria Circular Lyapunov

La prueba de seguimiento de trayectoria circular del controlador Lyapunov se realiza con los siguientes parámetros: $Ku = 0.3$, $Kw = 0.65$ y $radio = 1.5$. Se puede observar en la Figura 3.10 que la respuesta del controlador Lyapunov no tiene problemas para controlar la trayectoria, únicamente posee un error mínimo en eje y en el punto $(0, 0)$ (Figura 3.11) durante la salida del robot, mientras que en eje x la respuesta es mucho más estable (Figura 3.12).

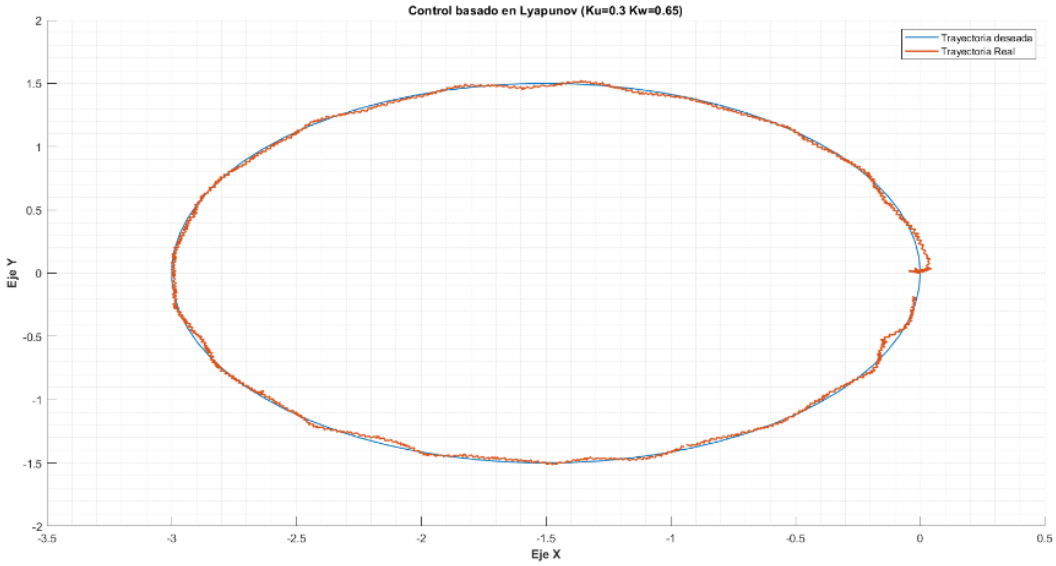

Figura 3.10 Respuesta Trayectoria Circular del Control Lyapunov.

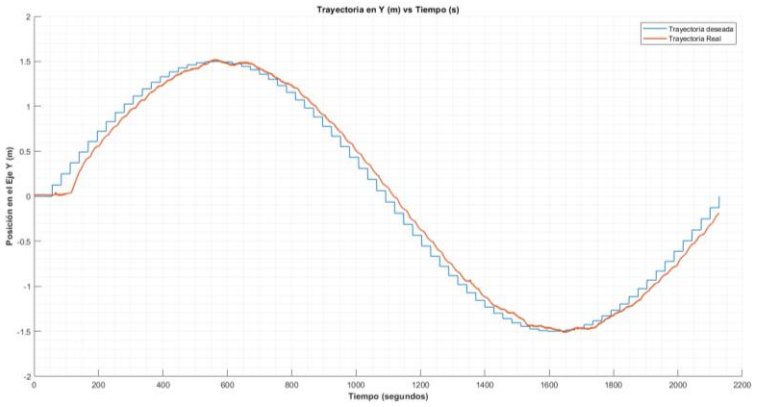

Figura 3.11 Respuesta en eje "y" de Trayectoria Circular del Control Lyapunov.

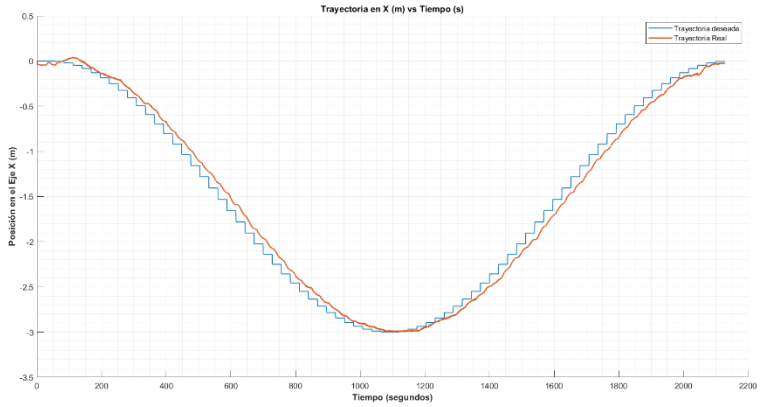

Figura 3.12 Respuesta en eje "x" de Trayectoria Circular del Control Lyapunov.

3.1.2.2 Trayectoria Cuadrada Lyaponov

Para la trayectoria cuadrada, al igual que la anterior, la respuesta del robot es bastante satisfactoria, se mantiene sobre la trayectoria en las secciones paralelas al eje X y en las esquinas gira al momento de llegar hasta volver a colocarse en la trayectoria.

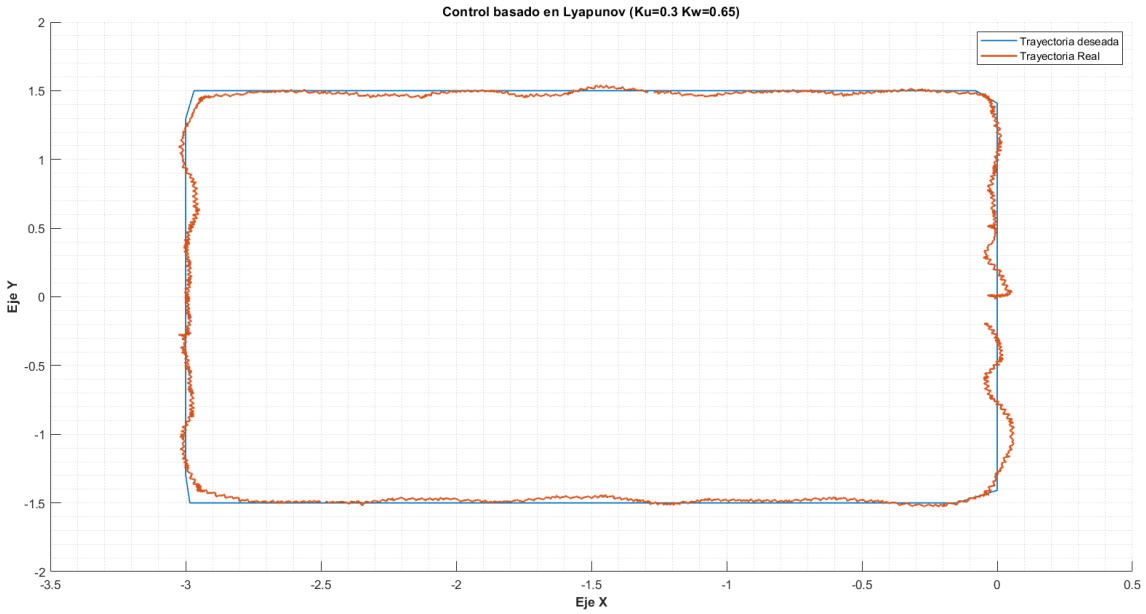

Figura 3.13 Respuesta de Trayectoria Cuadrada de Controlador Lyapunov.

En la Figura 3.14y Figura 3.15, se puede observar la respuesta tanto en el eje x como en el eje y de la trayectoria real (azul) vs la trayectoria realizada (rojo), en donde se observa que en el eje y se genera un mayor error con respecto a la trayectoria real en los cambios de dirección que se generan en las esquinas.

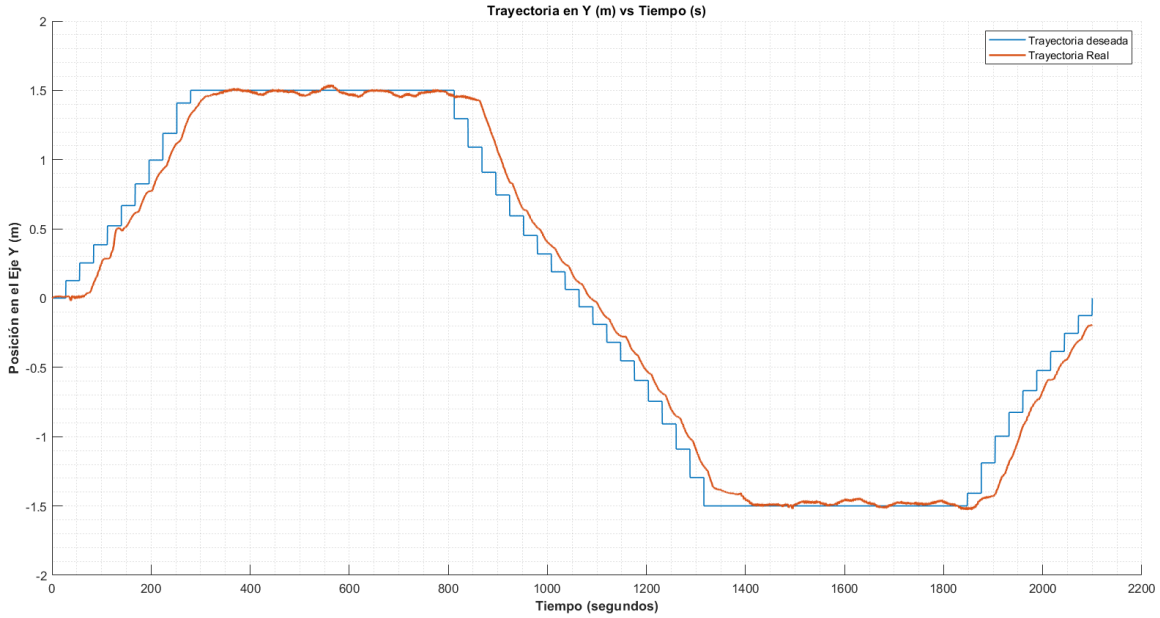

Figura 3.14 Respuesta en Eje "y" de la Trayectoria Cuadrada con Controlador Lyapunov.

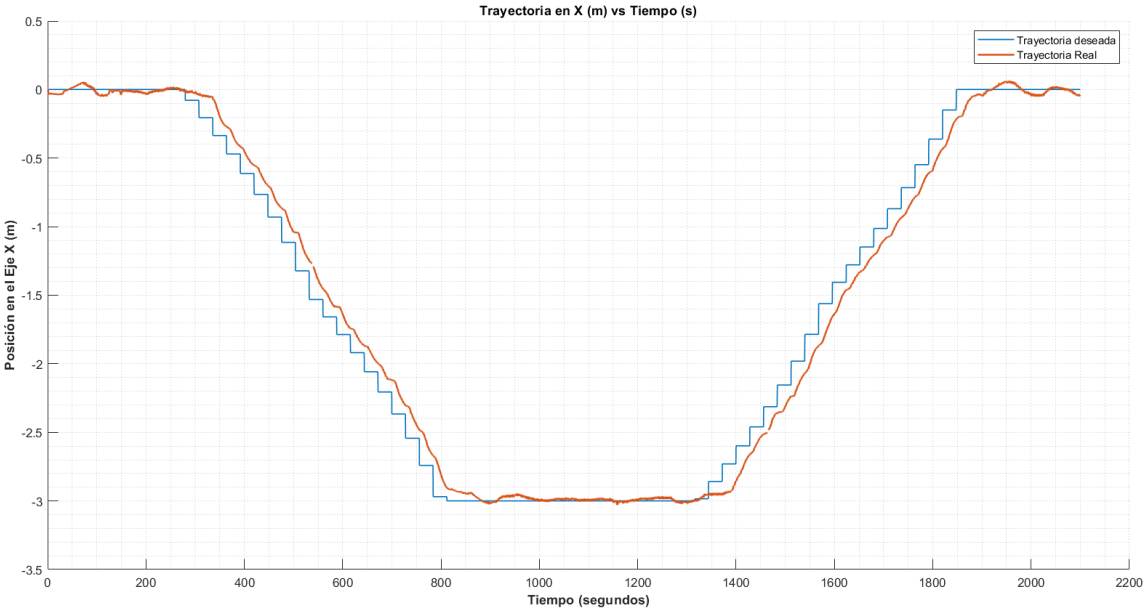

Figura 3.15 Respuesta en Eje "x" de la Trayectoria Cuadrada con Controlador Lyapunov.

3.1.2.3 Trayectoria Lemniscata Lyapunov

La prueba de seguimiento de trayectoria circular del controlador Lyapunov se realiza con los siguientes parámetros: $Ku = 0.3$, $Kw = 0.65$ y $radio\ centros = 3$. Se puede observar en la Figura 3.16 que la respuesta del controlador Lyapunov tiene problemas para controlar la trayectoria en los cambios de dirección que se ejecutan en el eje x de -0.8 a -0.6 y 0.6 a 0.8 (Figura 3.17); mientras que en el eje y la respuesta es mucho más estable (Figura 3.18).

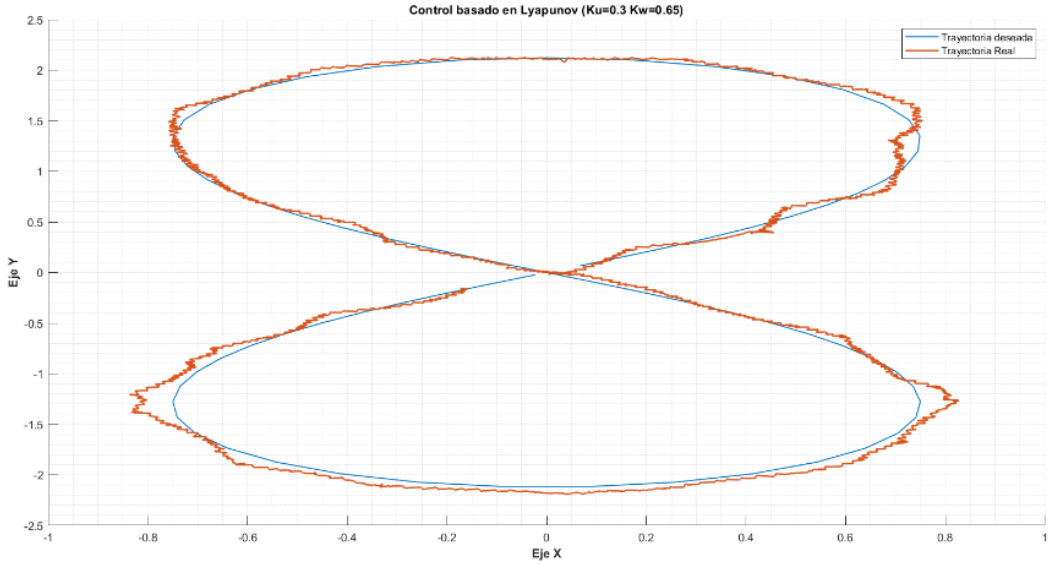

Figura 3.16 Respuesta Trayectoria Lemniscata del Control Lyapunov.

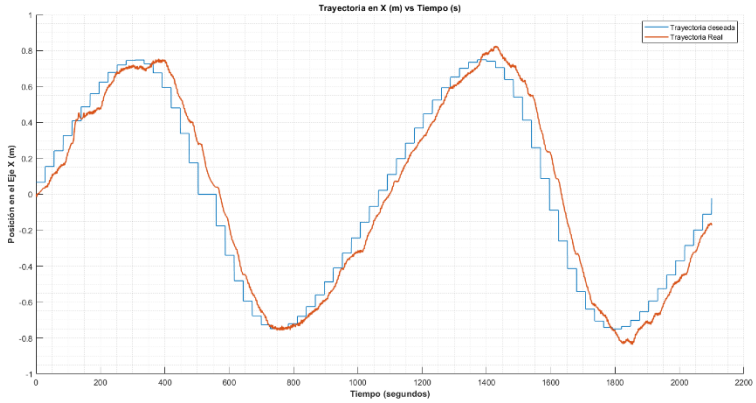

Figura 3.17 Respuesta en el eje "x" de la Trayectoria Lemniscata del Control Lyapunov.

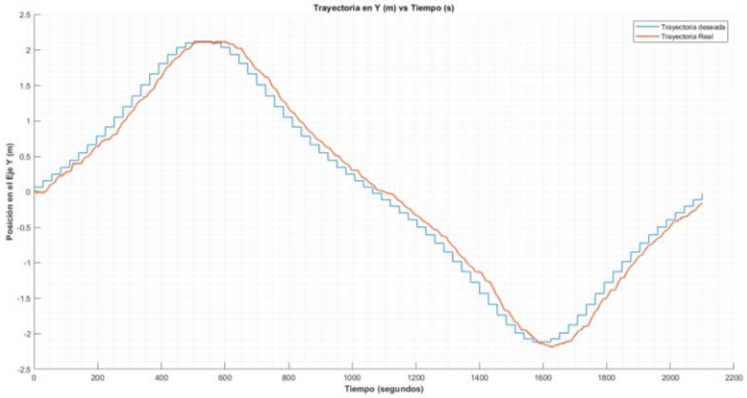

Figura 3.18 Respuesta en el eje "y" de la Trayectoria Lemniscata del Control Lyapunov.

3.1.3 CONTROLADOR CON REDES NEURONALES

3.1.3.1 Trayectoria Circular

La prueba de seguimiento de trayectoria circular del controlador de Red Neuronal FeedForward se realiza con el entrenamiento basado en el controlador Lyapunov con los parámetros $Ku = 0.3$, $Kw = 0.65$ y se realiza la simulación a partir de un $radio = 1{,}5$. Se puede observar en la Figura 3.19 que la respuesta del controlador tiene problemas de control únicamente en la partida del robot $(0, 0)$ hasta llegar al punto $(0.8, 0.8)$, en el eje x (Figura 3.20) y en el eje y (Figura 3.21) se observa que el mayor error se encuentra en estas coordenadas.

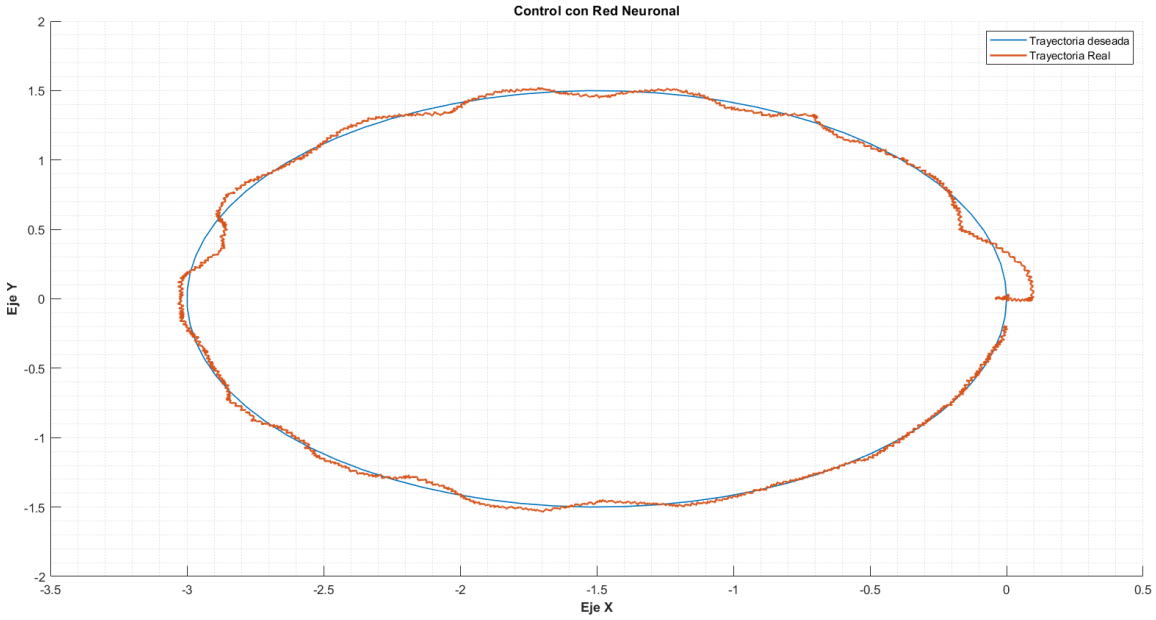

Figura 3.19 Respuesta de Trayectoria Circular del Control Red Neuronal.

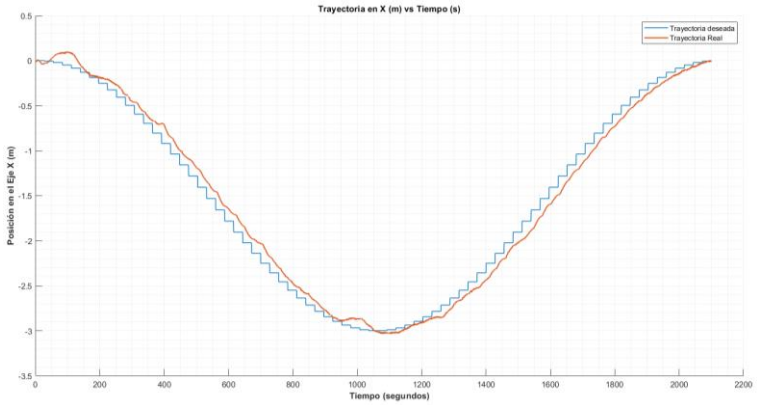

Figura 3.20 Respuesta en eje "x" Trayectoria Circular del Control Red Neuronal.

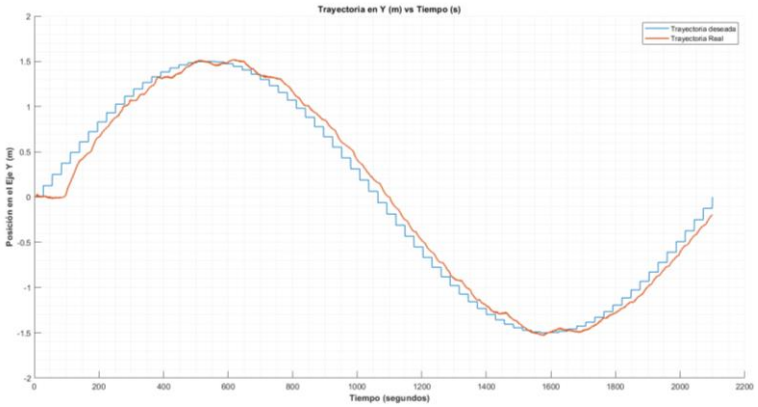

Figura 3.21 Respuesta en eje "y" Trayectoria Circular del Control Red Neuronal.

3.1.3.2 Trayectoria Cuadrada

Para la trayectoria cuadrada se utilizó la misma red neuronal entrenada, la trayectoria tiene cada lado de 3 metros. En la Figura *3.22* se observa como robot sigue la trayectoria de mejor manera en las secciones en las que la coordenada en Y es constante o se mueve paralelo al eje X.

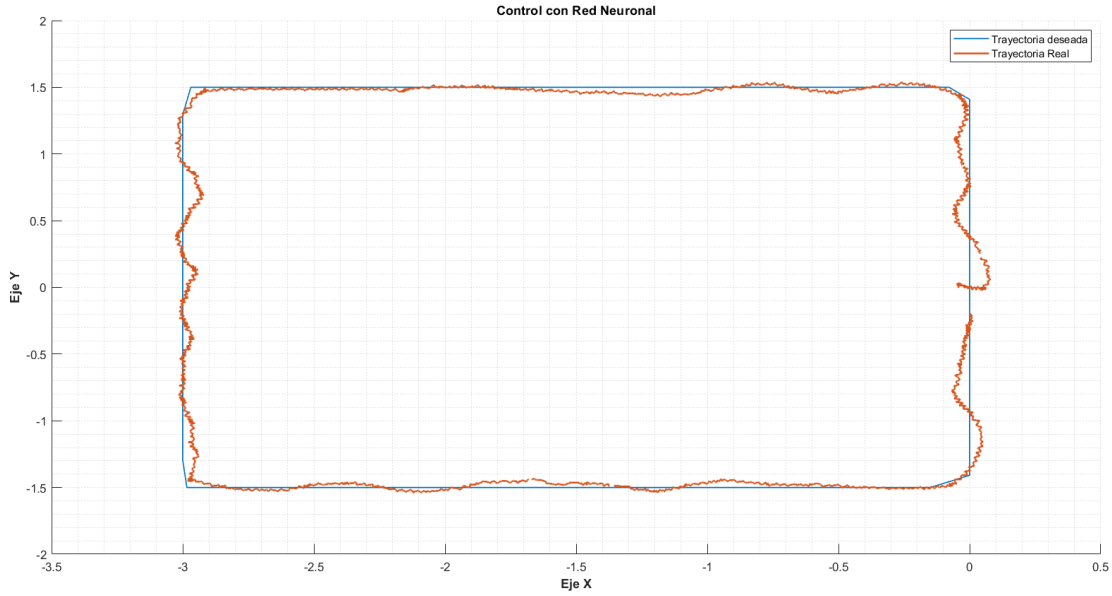

Figura 3.22 Respuesta en el plano de la Trayectoria Cuadrada del Control Neuronal.

En la Figura *3.23* y Figura 3.24, se puede observar la respuesta tanto en el eje x como en el eje y de la trayectoria real (azul) vs la realizada por el robot (rojo), en donde se identifica que en el eje y se genera un mayor error con respecto a la trayectoria real, al inicio, en el cambio de dirección de -1.5 a -1.5 y al finalizar la trayectoria.

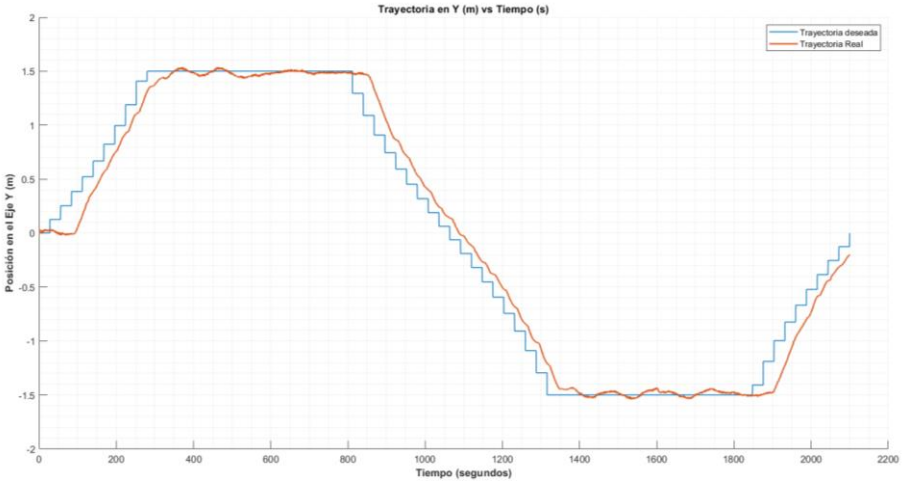

Figura 3.23 Respuesta en eje "y" de la Trayectoria Cuadrada del Control Neuronal.

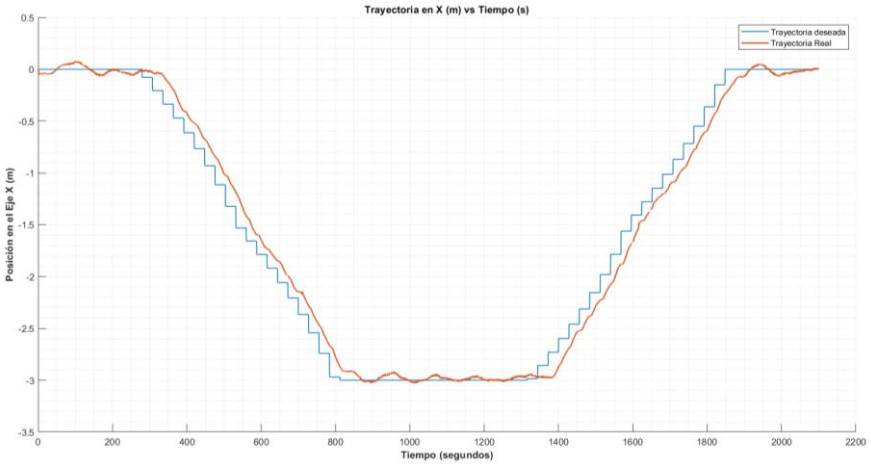

Figura 3.24 Respuesta en eje "x" de la Trayectoria Cuadrada del Control Neuronal.

3.1.3.3 Trayectoria Lemniscata

La prueba de seguimiento de trayectoria lemniscata del controlador de Red Neuronal FeedForward se realiza con el entrenamiento basado en el controlador Lyapunov con los parámetros $Ku = 0.3$, $Kw = 0.65$ y se realiza la simulación a partir de un $radio\ centros = 3$. Se puede observar en la Figura 3.25 que la respuesta del controlador tiene problemas de control en los centros de la lemniscata, sobre todo en el eje x en donde en la se observa en las coordenadas 0.8 a 0.6 y -0.6 a 0.8 (Figura 3.26), por otro lado en el eje y (Figura 3.27) se observa una respuesta más estable por parte del controlador.

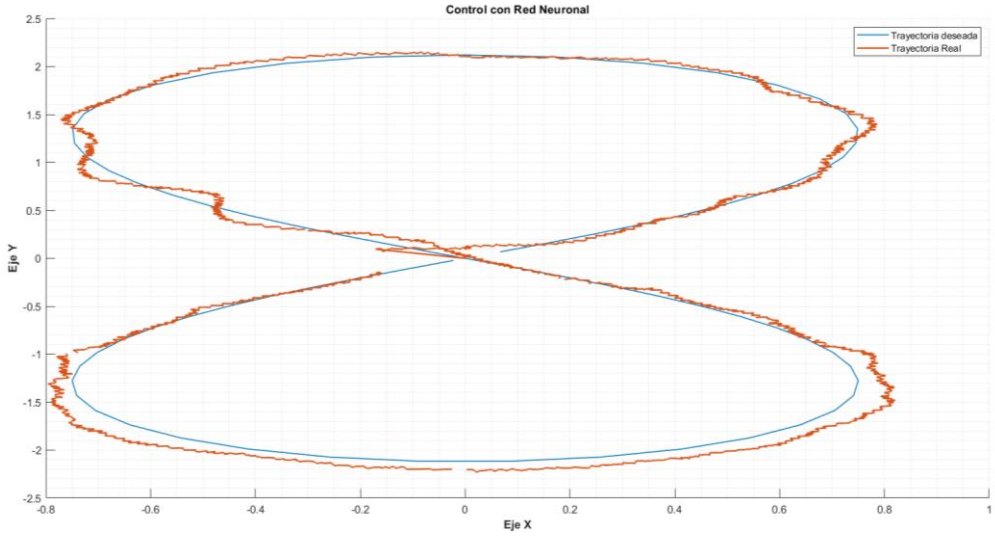

Figura 3.25 Respuesta Trayectoria Lemniscata del Control Red Neuronal.

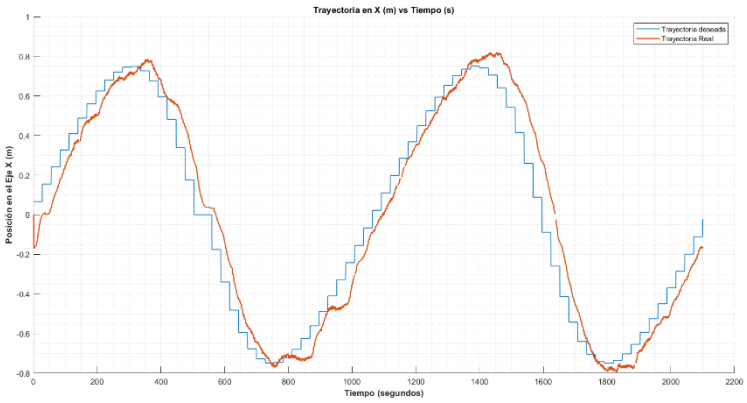

Figura 3.26 Respuesta en el eje "x" Trayectoria Lemniscata del Control Red Neuronal.

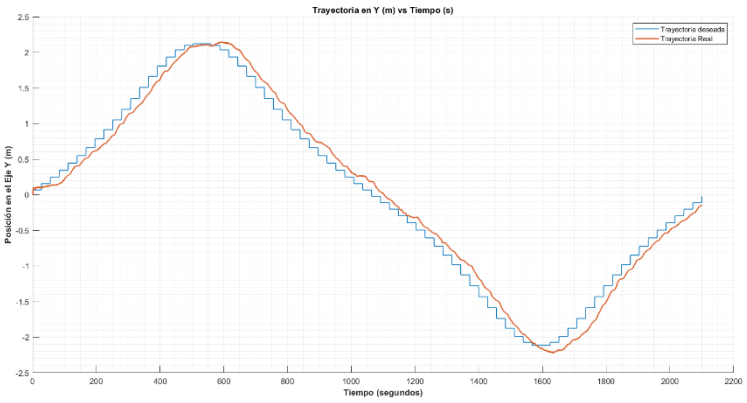

Figura 3.27 Respuesta en el eje "y" Trayectoria Lemniscata del Control Red Neuronal.

3.1.4 ÍNDICES DE DESEMPEÑO DE LOS CONTROLADORES

Se realizaron los cálculos de los indicadores de desempeño de cada controlador como se observa en la Tabla 3.1, el controlador PID tiene un peor comportamiento que Lyapunov y Red Neuronal, siendo estos últimos los que poseen indicadores muy similares, sin embargo

el controlador basado en Lyapunov tiene un mejor comportamiento en todas las trayectorias analizadas.

Tabla 3.1 Indicadores de Desempeño de los Controladores

		Indicador			
		ISU	**ISW**	**IAep**	**IAeo**
PID (Pu=2,1 Pw=1,9)	**Circular**	8,1087	326,6304	235,9558	773,9695
	Cuadrado	10,2537	311,477	539,9577	694,7231
	Lemniscata	10,1201	326,4178	356,4036	605,717
Lyapunov (Ku=0,3 Kw=0,65)	**Circular**	3,0755	63,6934	265,4632	300,0018
	Cuadrado	4,4963	129,333	336,3523	392,2451
	Lemniscata	4,826	108,9096	341,3299	334,3099
Red Neuronal (Entrenamiento Lyapunov)	**Circular**	3,469	90,6285	297,5302	365,001
	Cuadrado	4,9452	126,831	381,1913	416,9524
	Lemniscata	4,374	123,5492	358,042	360,5432

3.2 CONCLUSIONES

- Actualmente se desarrollan métodos de control más complejos aplicados a los sistemas a los que las teorías de control clásico ya no pueden ser aplicados, estos métodos incluyen la implementación de redes neuronales artificiales que llegan a simular el proceso de toma de decisiones y aprendizaje del ser humano. Estas características son las que permiten su aplicación en el control de sistemas muy variantes o cuyas representaciones matemáticas son difíciles de abordar. Una de sus aplicaciones se ha demostrado en este trabajo, utilizando este concepto de aprendizaje en su forma más básica y obteniendo resultados iniciales favorables que avalan su uso en las aplicaciones de control del movimiento de robots complejos.

- El controlador PID pese a ser uno de los más utilizados en la industria y mayoría de trabajos de investigación, no tiene un buen comportamiento en sistemas no lineales multivariable como el que representa el sistema de seguimiento de trayectoria; en las pruebas realizadas se observa que este controlador genera errores muy altos en los ejes x, y, sobre todo cuando existen cambios de giro por parte del robot.

- El controlador Lyapunov tiene un comportamiento muy estable a diferencia del controlador PID. En este se pudo verificar que para sistemas no lineales multivariable como el analizado en este proyecto, el control realizado tiene un bajo

porcentaje de error, sin embargo, en situaciones donde hay cambios de orientación muy altos se puede generar errores más altos como en el caso de la lemniscata.

- El controlador de Redes Neuronales tiene un comportamiento casi idéntico al controlador de Lyapunov debido a que su entrenamiento está basado en este; siendo así los errores que genera este controlador son muy parecidos tanto en el eje x, y, sin embargo tiene mucha más escalabilidad ya que no solo requiere el cambiar parámetros si no que se pueden usar sus datos de salida para entrenar posteriores redes neuronales más complejas y con muchas más redes ocultas que permitan reducir el error a un valor mínimo en todos los casos.
- Los indicadores de rendimiento muestran que el controlador de redes neuronales y el controlador Lyapunov tienen mejor respuesta de control en cada uno de los parámetros analizados en este proyecto, así también, se observó que las trayectorias en las que mejor controla el sistema es la circular, debido que sus cambios de orientación no son tan bruscos como en los otros casos.
- El uso de plataformas como el robot humanoide NAO v6 permiten a los estudiantes e investigadores probar nuevas técnicas de control de forma fácil y directa, ya que se vinculan con software como MatLab/Simulink y Python que son utilizados ampliamente en carreras de ingeniería y afines.

3.3 RECOMENDACIONES

- Se recomienda realizar pruebas con diferentes condiciones de tamaño y dirección para cada prueba que se ha presentado en este proyecto, ya que se pueden analizar y calibrar los controladores dependiendo de la respuesta que se consiga en diferentes circunstancias.
- Se recomienda el diseño de controladores alternos al PID para sistemas que sean no lineales y que controlen más de una variable a la vez, ya que el comportamiento de este se ha observado muy bajo con respecto a la respuesta obtenida por los otros controladores analizados.
- Se recomienda realizar pruebas más complejas sobre los controladores tanto de Lyapunov como de Redes Neuronales para observar la respuesta de este en situaciones donde se puedan realizar cambios mucho más complejos de orientación que los analizados en el presente trabajo.

- Se recomienda utilizar los datos de la Red Neuronal diseñada en el presente proyecto para lograr un control mucho más sólido y complejo, con varias capas ocultas que permitan reducir el error al mínimo no solamente en trayectorias fijas, si no en trayectorias aleatorias que puedan ser incluidas.

4 REFERENCIAS BIBLIOGRÁFICAS

[1] E. Nuñez, «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 aleatoreo -RRT,» Escuela Politécnica Nacional, Quito, 2021.

[2] U. A y D. A, «Diseño e implementación de un escenario mecatrónico interactivo con el uso de robots Nao como herramienta tecnológica de apoyo a la enseñanza en niños,» Universidad de las Fuerzas Armadas ESPE, Sangolquí, 2021.

[3] S. Shamsuddin, H. Yussof y F. Akhtar, «Initial Response of Autistic Children in Human-Robot Interaction Therapy with Humanoid Robot NAO,» de *2012 IEEE 8th International Colloquium on Signal Processing and its Applications*, Malaysia, 2012.

[4] B. Conterón, «DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE IMITACIÓN CORPORAL COMPLETA EN TIEMPO REAL CON UN ROBOT HUMANOIDE NAO,» Universidad de las Fuerzas Armadas ESPE, Sangolquí, 2020.

[5] Z. Llansola, «Robots Humanoides,» Universitat Jaume I, Castellón, 2006.

[6] M. Barceló, «¿Robots humanoides?,» BYTE 195, 2012.

[7] J. Valarezo, «IMPLEMENTACIÓN DE UN SISTEMA BASADO EN EL ROBOT NAO PARA LECTURA DE TEXTOS EN APLICACIONES DE HUMAN ROBOT INTERACTION,» Universidad de las Fuerzas Armadas ESPE, Sangolquí, 2019.

[8] F. Vilatuña, «DESARROLLO DE UN SISTEMA BÁSICO DE ROBOTICA DE ENTRETENIMIENTO PERSUASIVO BASADO EN EL SISTEMA HUMANOIDE NAO,» Universidad de las Fuerzas Armadas ESPE, Sangolquí, 2018.

[9] F. Rivas, J. Cañas y J. González, «Aprendizaje automatico de modos de caminar para un robot humanoide,» *Proceedings of Robot,* pp. 120-127, 2011.

[10] M. Velásquez, «Control en Modo Deslizante con Estimación de la Perturbación Aplicado a la Marcha de un Robot Bípedo.,» INAOE, Puebla, 2013.

[11] J. Avila, M. Mayer y V. Quesada, «La inteligencia artificial y sus aplicaciones en medicina I: introducción antecedentes a la IA y robótica,» *Atención Primaria,* vol. 52, nº 10, pp. 778-784, 2020.

[12] O. Aguilar, R. Tapia y I. Rivas, «Diseño de un controlador de velocidad adaptativo para un MSIP utilizando inteligencia artificial.,» *Computación y Sistemas,* vol. 1, nº 20, pp. 41-54, 2016.

[13] J. Nandar, «Diseño y Simulación de un Sistema de Teleoperación de un Robot Humanoide NAO,» Escuela Politécnica Nacional, Quito, 2022.

[14] S. Iglesias, «Locomoción bípeda del robot humanoide NAO,» Universidad Politécnica de Catalunya, Barcelona, 2009.

[15] F. Calvopiña y P. Valladares, «INTERPRETACIÓN DE EXPRESIONES FACIALES EN ADULTOS MAYORES UTILIZANDO LA VISIÓN ARTIFICIAL DEL ROBOT HUMANOIDE NAO,» Universidad Politécnica Salesiana, Quito, 2017.

[16] S. Supplies, «NAO V6 Programmable Robot,» Stem Supplies, [En línea]. Available: https://stem-supplies.com/nao-v6-programmable-robot. [Último acceso: 01 02 2023].

[17] C. Rodriguez, «IMPLEMENTACIÓN DE TÉCNICAS DE VISIÓN ARTIFICIAL EN UN ROBOT HUMANOIDE PARA BENEFICIO DE NIÑOS CON DÉFICIT DE ATENCIÓN E HIPERACTIVIDAD A TRAVÉS DE UN EJERCICIO PSICOMOTRIZ,» Escuela Superior Politécnica de Chimborazo, Riobamba, 2018.

[18] K. Enriquez, «Diseño e implementación de un sistema de clasificación de objetos basado en el robot humanoide NAO,» Universidad de las Fuerzas Armadas, Sangolquí, 2019.

[19] G. Gallud, «Reconocimiento de emociones humanas y su aplicación a la Robótica Social,» Universidad de Alicante, Alicante, 2019.

[20] D. Rodriguez, «Control de Robots Humanoides a través de agentes Jason,» Universidad Politécnica de Valencia, Valencia, 2017.

[21] C. Pérez, D. Ricardo y B. Roberto, «El lenguaje de programación Python,» Ciencias Holguín, Cuba, 2014.

[22] A. Fernandez, Python 3 al descubierto, México: AlfaOmega, 2013.

[23] V. Frittelli, D. Serrano, R. Teicher, F. Steffolani, M. Tartabini, J. Fernández y G. Bett, «Uso de Python como Lenguaje Inicial en Asignaturas de Programación,» *Artículos de las III Jornadas de Enseñanza de la Ingeniería,* vol. 2, nº 3, pp. 132 - 137, 2013.

[24] A. O. Escobar, «ESTUDIO DE SDK DE REALIDAD AUMENTADA (VUFORIA, WIKITUDE Y ARTOOLKIT) PARA EL RECONOCIMIENTO DE OBJECT TARGET BASADO A LA ESCALA Y DISTANCIA EN DISPOSITIVOS MÓVILES CON SISTEMAS OPERATIVOS ANDROID.,» Universidad Técnica del Norte, Ibarra, 2021.

[25] Watcher, «API/SDK,» Watcher, [En línea]. Available: https://watcher.flussonic.com/en-es/features/api-sdk/. [Último acceso: 01 02 2023].

[26] S. Rooban, S. D. Suraj, S. B. Vali y N. Dhanush, «CoppeliaSim: Adaptable modular robot and its different locomotions simulation framework,» *Elsevier,* p. 6, 2020.

[27] D. Díaz, «SIMULACIÓN DE UN ENTORNO INDUSTRIAL MEDIANTE LA HERRAMIENTA DE TRABAJO COPPELIASIM (V-REP),» UNIVERSITAT POLITÈCNICA DE VALÈNCIA, Valencia, 2020.

[28] CoppeliaSim, «CoppeliaSim User Manual,» CoppeliaRobotics, [En línea]. Available: https://www.coppeliarobotics.com/helpFiles/. [Último acceso: 01 02 2022].

[29] Matlab, «Matlab/Simulink,» 2016.

[30] A. Enriquez, «DISEÑO Y SIMULACIÓN DEL CONTROL DE POSICIONAMIENTO DE UN ROBOT HUMANOIDE NAO,» Escuela Politécnica Nacional, Quito, 2022.

[31] A. Bañó, «Análisis y Diseño del Control de Posición de un Robot Móvil con Tracción Diferencial,» Universitat Rovira I Virgili, Virgili, 2003.

[32] R. Ambrocio, «Control basado en el ZMP usando bases de Groebner para la marcha de un robot humanoide.,» UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA, Oaxaca, 2019.

[33] V. A. Ruíz, «ECUACIONES PARA CONTROLADORES PID UNIVERSALES,» *Ingeniería,* vol. 12, nº 1,2, pp. 11-20, 2002.

[34] T. Á. Cantarero, «Diseño del Controlador PID,» Universidad de Sevilla, Sevilla, 2015.

[35] M. García y A. Barreiro, «Análisis de Estabilidad segpun Lyapunov de un Control Borroso en Tiempo Discreto,» p. 6.

[36] K. Ogata, Ingeniería de Control Moderna, Madrid: Pearson Education, 2010.

[37] S. G. Tzafestas, «5 - Mobile Robot Control I: The Lyapunov - Based Method,» *Introduction to Mobile Robot Control,* pp. 137-183, 2014.

[38] J. E. Sierra-García y M. Santos, «Redes neuronales y aprendizaje por refuerzo en el control de turbinas eólicas,» *Revista Iberoamericana de Automática e Informática Industrial,* vol. 18, pp. 327-335, 2021.

[39] Deingenierias.com, «Redes neuronales en inteligencia artificial,» De Ingenierias, 13 06 2019. [En línea]. Available: https://deingenierias.com/inteligencia-artificial/redes-neuronales-en-inteligencia-artificial/. [Último acceso: 2023 02 10].

[40] S. Pérez, J. Mora y G. Morales, «ESTRATEGIA DE RE-ENTRENAMIENTO DE REDES NEURONALES PARA MEJORAR EL CONTROL DE LA EXCITACIÓN DE UNA MÁQUINA SÍNCRONA,» *Scientia Et Technica,* vol. 12, nº 32, pp. 37-42, 2006.

[41] S. Castaño, «Índices de Desempeño,» Control Automático Educación, [En línea]. Available: controlautomaticoeducacion.com/control-realimentado/indices-de-desempeno/. [Último acceso: 10 02 2023].

[42] E. A. P. Flores, «Diagrama del Robot Móvil,» Research Gate, [En línea]. Available: https://www.researchgate.net/figure/Diagrama-del-robot-movil_fig1_28272353. [Último acceso: 13 02 2023].

[43] S. Tzafestas, Introduction to mobile robot control, Amsterdam: Elsevier, 2014.