REPÚBLICA DEL ECUADOR
Escuela Politécnica Nacional
"E SCIENTIA HOMINIS SALUS"

La versión digital de esta tesis está protegida por la Ley de Derechos de Autor del Ecuador.

Los derechos de autor han sido entregados a la "ESCUELA POLITÉCNICA NACIONAL" bajo el libre consentimiento del (los) autor(es).

Al consultar esta tesis deberá acatar con las disposiciones de la Ley y las siguientes condiciones de uso:

- Cualquier uso que haga de estos documentos o imágenes deben ser sólo para efectos de investigación o estudio académico, y usted no puede ponerlos a disposición de otra persona.

- Usted deberá reconocer el derecho del autor a ser identificado y citado como el autor de esta tesis.

- No se podrá obtener ningún beneficio comercial y las obras derivadas tienen que estar bajo los mismos términos de licencia que el trabajo original.

El Libre Acceso a la información, promueve el reconocimiento de la originalidad de las ideas de los demás, respetando las normas de presentación y de citación de autores con el fin de no incurrir en actos ilegítimos de copiar y hacer pasar como propias las creaciones de terceras personas.

Respeto hacia sí mismo y hacia los demás.

ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DISEÑO, SIMULACIÓN Y COMPARACIÓN DE UN CONTROLADOR PID Y UN CONTROLADOR BASADO EN LYAPUNOV PARA EL DESPLAZAMIENTO DE UN ROBOT HUMANOIDE NAO V6 SOBRE UN CAMINO GENERADO POR UN ALGORITMO DE EXPLORACIÓN RÁPIDA DE ÁRBOL ALEATORIO -RRT

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y CONTROL

EDGAR ROBERTO NUÑEZ QUISHPE

JUAN CARLOS ORTEGA QUEZADA

DIRECTOR: Dr. GEOVANNY DANILO CHÁVEZ GARCÍA

Quito, marzo 2021

AVAL

Certifico que el presente trabajo fue desarrollado por Edgar Roberto Nuñez Quishpe y Juan Carlos Ortega Quezada bajo mi supervisión.

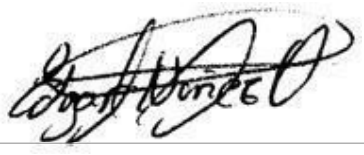
Dr. Geovanny Danilo Chávez García

DIRECTOR DEL TRABAJO DE TITULACIÓN

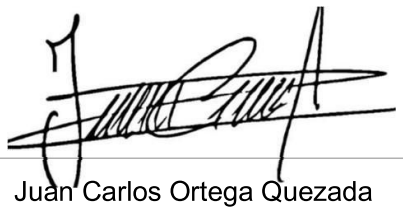
DECLARACIÓN DE AUTORÍA

Nosotros, Edgar Roberto Nuñez Quishpe y Juan Carlos Ortega Quezada, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejamos constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.



Edgar Roberto Nuñez Quishpe



Juan Carlos Ortega Quezada

DEDICATORIA

Dedicado a todos nuestros amigos, compañeros y maestros de la Escuela Politécnica Nacional y a todos aquellos estudiantes en busca de nuevo conocimiento. Esperamos encuentren en este trabajo la motivación y guía académica necesaria que les permita emprender en nuevos proyectos de robótica.

AGRADECIMIENTO

A mis padres Edgar y Carmen, que siempre me han sabido apoyar y aconsejar incondicionalmente en cada paso que he tomado en el trayecto de mi vida, tanto personal como académica, que siempre han esperado lo mejor de mí.

A mi hermana Maricela, con quien comparto tantos recuerdos de mi niñez, y me supo mostrar que a pesar de las dificultades que se presenten, siempre podemos cumplir nuestros objetivos.

A mis mejores amigos Juan, Lesly, Nicole, Salomé y Viviana, con los que no solo he compartido risas y buenos momentos, si no que han estado ahí para darme ánimos, y buscando que sea la mejor versión de mí mismo.

A nuestro tutor de tesis, el Dr. Danilo Chávez y de igual manera al Ing. Kleber Patiño, que nos aconsejaron en la realización de este trabajo de titulación, con sus consejos fue posible su realización.

Finalmente, a todos los amigos que he conocido en la universidad, que me han compartido sus sueños y aspiraciones para su futuro, me han inspirado a siempre buscar el conocimiento y el éxito.

Edgar

A mis padres Carlos y Luz, que siempre han sido mi fuente de inspiración y motivación para todos mis proyectos de vida. No pude pedir mejores padres en esta vida.

A mis hermanas, Soledad, Yomi y Nadia que con sus elocuencias y palabras de aliento siempre me mantuvieron enfocado a finalizar con éxito todas mis metas.

A mis compañeros y mejores amigos Edgar y Viviana con los que compartí muchas anécdotas universitarias y siempre estuvieron ahí en los mejores y peores momentos.

A nuestro asesor de tesis, el Dr. Danilo Chávez que nos supo guiar correctamente hacia la realización de este trabajo. Agradezco de igual manera al Ing. Kleber Patiño quien siempre estuvo pendiente de nuestras dudas y supo aclararlas de la mejor manera. No hubiera sido posible la realización de este trabajo sin ustedes.

A todos mis grandes maestros que siempre me supieron guiar y motivar a seguir adelante con todos mis proyectos.

Y finalmente, a todos mis amigos dentro y fuera de la universidad que me acompañaron en este maravilloso camino universitario. Espero encontrarme con cada uno de ustedes y ver como se cumplen sus sueños.

Juan

ÍNDICE DE CONTENIDO

RESUMEN

En el siguiente documento se presenta el diseño, simulación y comparación de un controlador PID y un controlador basado en Lyapunov para el desplazamiento de un robot humanoide NAO V6 sobre un camino generado por un algoritmo de exploración rápida de árbol aleatorio –RRT tipo FN.

Con el fin de facilitar un espacio seguro en donde se pueda efectuar pruebas de simulación cercanas a la realidad del robot Nao con diferentes algoritmos de control, se ha desarrollado todo este proyecto en un potente simulador de robot llamado CoppeliaSim.

Adicionalmente, se presenta la implementación de una interfaz gráfica desarrollada en Qt Creator, la cual permite al usuario establecer condiciones iniciales y monitorear al robot en su ambiente de trabajo.

PALABRAS CLAVE: robot humanoide, camino, CoppeliaSim, Qt Creator.

ABSTRACT

The following paper presents the design, simulation, and comparison of a PID controller and a Lyapunov-based controller for the displacement of a NAO V6 humanoid robot over a path generated by a rapidly exploring random tree algorithm -RRT FN.

In order to provide a safe space where simulation tests close to the reality of the Nao robot with different control algorithms can be performed, this whole project has been developed in a powerful robot simulator called CoppeliaSim.

Additionally, the implementation of a graphical interface developed in Qt Creator is presented, which allows the user to establish initial conditions and monitor the robot in its work environment.

KEYWORDS: humanoid robot, path, CoppeliaSim, Qt Creator.

1. INTRODUCCIÓN

Gracias a los avances tecnológicos que se han ido produciendo en estas últimas décadas, los robots han ido evolucionando exponencialmente, lo cual ha permitido encontrar soluciones optimas a diferentes problemas e impactar positivamente a la sociedad [1]. En la actualidad, no solamente es común encontrarse con dichos robots en fábricas, sino también en hogares, centros educativos o incluso en residencias para adultos mayores en donde su principal función es brindarles compañía [2]. ¿Cómo es que estos robots llegaron a formar parte de nuestra vida cotidiana? Para que los robots puedan interactuar con humanos, es necesario que estos puedan ser capaces de desenvolverse por sí solos en entornos reales, los cuales tienden a ser complejos. Por tal motivo, se ha desarrollado robots humanoides, cuya estructura física, no solamente provee una mejor relación hombre-máquina, sino que también, les permite movilizarse de manera similar a la de los humanos [3]. Se puede decir, entonces, que estos robots poseen una gran ventaja en comparación a otros robots estándar, puesto que, pueden ejercer sus actividades directamente en un entorno humano [4]. Sin embargo, dicha ventaja conlleva a un diseño de control más complejo y sofisticado, debido a que el robot no solamente debe procurar realizar una tarea en específico, sino que también debe desplazarse a través de su entorno de manera fluida y eficiente, es decir, procurando no colisionar con obstáculos o interrumpir actividades humanas o de otros robots [5].

De igual forma, otra de las dificultades a la que se deben enfrentar dichos robots es poder obtener una localización confiable y precisa en todo momento dentro de entornos interiores complejos arbitrarios. Esto debido a que, por lo general, estos robots ejecutan comandos de movimiento de forma bastante imprecisa [6]. Cabe recalcar, además, que este tipo de robots presentan un movimiento oscilante al momento de caminar, lo cual puede llegar a alterar su desplazamiento [6].

Trabajar directamente con robots en ambientes reales puede llegar a tener repercusiones económicas altas como daños físicos del robot ya que se está trabajando en escenarios no controlados [7]. Por tal razón, es importante contar con un entorno de desarrollo integrado, que facilite un espacio seguro en donde se pueda desarrollar pruebas de simulación cercanas a la realidad de robots con diferentes algoritmos de control, sin que exista riesgo de dañar su estructura física [5]. De esta manera, se asegura beneficios económicos, en tiempo y seguridad [6].

En función de lo mencionado, y debido a que existen pocos controladores viables para la evasión de obstáculos en ambientes reales, se propone en este proyecto de titulación, el diseño, simulación y comparación de un controlador PID y un controlador basado en Lyapunov de un robot humanoide NAO V6 sobre un camino generado por un algoritmo

de exploración rápida de árbol aleatorio –RRT, en un ambiente de simulación generado en CoppeliaSim, basado en un aula de clase con 3 diferentes disposiciones de pupitres.

En el siguiente trabajo, se presenta primeramente una revisión bibliografía de los temas más relevantes tratados durante todo este trabajo con el fin de poder entender e interpretar el desarrollo completo del proyecto y los resultados mostrados al final. Luego, se presenta la metodología utilizada tanto para el diseño de los controladores, como para la generación del camino mediante un algoritmo RRT. Se especifica las técnicas, herramientas e instrumentos empleados para el desarrollo de este Proyecto Técnico. A continuación, se presentan los resultados y discusiones pertinentes que demuestre el funcionamiento completo del trabajo realizado. Finalmente se establece un conjunto de conclusiones y recomendaciones del Proyecto Técnico completo.

1.1 OBJETIVOS

OBJETIVO GENERAL:

Diseñar, simular y comparar un controlador PID y un controlador basado en Lyapunov para el desplazamiento de un robot humanoide Nao V6 sobre un camino generado por un algoritmo de exploración rápida de árbol aleatorio -RRT

OBJETIVOS ESPECÍFICOS:

- Realizar revisión bibliográfica del sistema NAO V6, del software libre de simulación CoppeliaSim Edu., del lenguaje de programación Python y del algoritmo de generación de caminos RRT (Rapidly Exploring Random Tree).
- Diseñar un esquema de control PID y uno basado en Lyapunov aplicado al sistema NAO y un algoritmo RRT para la planificación de caminos.
- Mediante CoppeliaSim Edu, simular el desenvolvimiento del robot Nao en 3 diferentes escenarios basado en aulas de clase.
- Realizar pruebas de cambios de referencia y comparar el rendimiento del controlador tipo PID y el controlador basado en Lyapunov, mediante índices de rendimiento ISE e IAE.
- Diseñar e implementar una interfaz gráfica, para la visualización del comportamiento del sistema con diferentes esquemas de control.

1.2 ALCANCE

El alcance definido en este Proyecto Técnico se lo describe en los siguientes puntos:

- Se realizará la revisión bibliográfica de: las características del robot humanoide NAO V6, así como también, las librerías implementadas para su funcionamiento, los conceptos básicos del lenguaje de programación Python, el funcionamiento básico del software de simulación CoppeliaSim Edu(V-REP), y el algoritmo de generación de caminos RRT*FN.
- Se diseñará esquemas de control PID y se los comparará a un control basado en Lyapunov de velocidad lineal y velocidad angular del robot humanoide NAO mediante el uso de sus funciones implementadas en NAOqi – Módulo principal de Python del robot NAO.
- Se implementará un algoritmo RRT*FN usando Python para la generación de caminos y evasión de obstáculos en el espacio de trabajo del robot humanoide NAO.
- Mediante CoppeliaSim Edu, se simulará y evaluará el desempeño de los controladores implementados en el robot Nao ante cambios de referencia.
- Se desarrollará un ambiente de simulación basado en un aula de clase con 3 diferentes disposiciones de pupitres.
- Se realizará la comparación de los controladores tipo PID y Lyapunov mediante índices de rendimiento ISE e IAE.
- Se diseñará e implementará una interfaz gráfica, para visualizar el comportamiento del sistema con los diferentes esquemas de control.

1.3 MARCO TEÓRICO

A continuación, se presenta la información teórica necesaria para comprender completamente este proyecto técnico.

1.3.1 INTRODUCCIÓN AL ROBOT HUMANOIDE NAO V6.

El robot humanoide Nao, mostrado en la Figura 1.1, es un robot muy conocido a nivel mundial diseñado principalmente para interactuar con personas. Fue creado en 2008 por la compañía Aldebaran Robotics (Francia), la cual en 2015 fue adquirida por la empresa SoftBank Robotics. Nao viene integrado con varios sensores y motores con los cuales puede caminar, bailar, hablar e incluso reconocer caras y objetos. Ha sido utilizado en el campo de la investigación, de la educación y la salud alrededor de todo el mundo [8].

Figura 1. 1 Robot humanoide Nao V6 Edición Estándar [9]

El robot humanoide Nao tiene hasta el momento 6 versiones, siendo V6 la última. A continuación, se expondrá sus características más relevantes.

1.3.1.1 Características generales

El robot Nao V6, en comparación a otro robots humanoides convencionales, es un robot de pequeña estatura con una combinación única entre software y hardware [10]. Su hardware está compuesto de varios sensores, motores y actuadores que le permiten interactuar con el medio externo, mientras que su software es manejado por NAOqi un sistema operativo dedicado[10]. Entre las características más relevantes se puede destacar:

- Como se observa en la Figura 1.2, NaoV6 tiene un cuerpo con 25 grados de libertad – Degrees of Freedom (DoF)(Cabeza: 2DoF, Brazo: 5DoF x 2, Pelvis: 1Dof, Pierna: 5Dof x 2, Mano: 1DoF x 2) en donde sus motores y actuadores son clave para un movimiento libre [9].

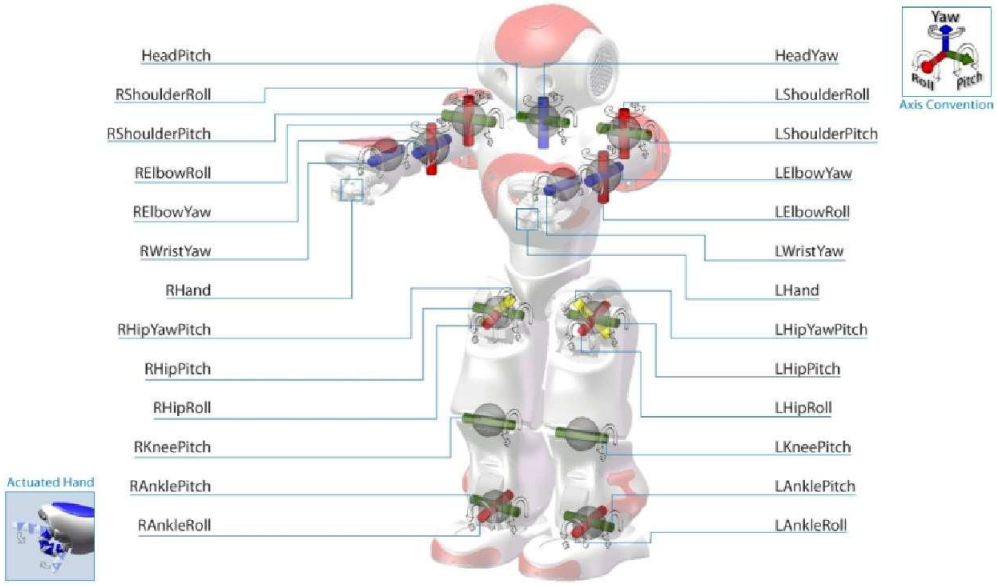


Figura 1. 2 Cadena cinemática y articulaciones (25DoF) del robot humanoide Nao. [11]

- Como se puede observar en la Figura 1.3, NaoV6 posee una red de sensores en todo su cuerpo, de los cuales se puede destacar: dos cámaras, cuatro micrófonos, dos sensores de aproximación, nueve sensores táctiles y ocho sensores de presión [9].
- Nao tiene varios dispositivos de comunicación que incluyen: un sintetizador de voz, luces LED y dos altavoces de alta fidelidad, con los cuales puede oír, hablar y parpadear sus luces [9].
- Nao, en su última versión, cuenta con un procesador Intel ATOM 1.91 GHz de cuatro núcleos, una memoria caché de 2MB, una memoria RAM de 4GB DDR3 y una memora Flash de 32 GB eMMC [9].
- Tiene una batería tipo Litio-Ion, con un voltaje/capacidad nominal de 21.6V/2.9Ah, un tiempo de carga de 90 minutos, y una duración de 60 minutos (uso activo) o 90 minutos (uso normal) [9].

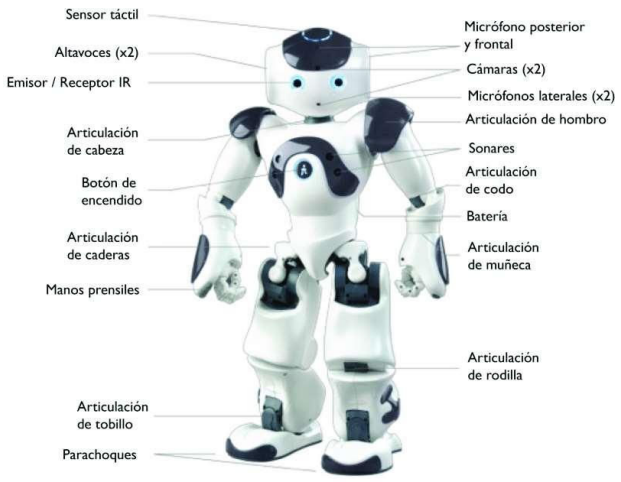


Figura 1. 3 Características técnicas del robot NAO [10].

- Actualmente, el robot Nao V6 posee un reconocimiento de voz y diálogo disponible en 20 idiomas, incluidos inglés, francés, español, alemán, italiano, árabe, holandés, portugués, checo, finlandés, ruso, sueco y turco [9].
- Nao V6 puede ser programado en C++ y Python y es soportado por los sistemas operativos Linux (Ubuntu 16.04 Xenial Xerus – solo de 64 bits), Windows (Microsoft Windows 10 64 bits) y Mac (MAC OS X 10.12 Sierra) [9].
- Como se observa en la Figura 1.4, posee una altura de 58 cm, una longitud de 27.5 cm, un ancho de 31.1 cm y un peso de 5.5 Kg con una velocidad de 0.3 Km/h (velocidad de marcha predeterminada) [9].

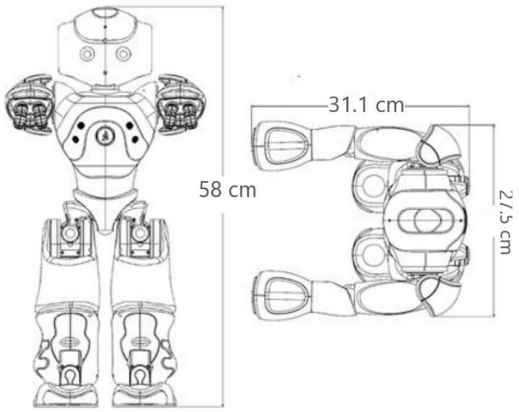


Figura 1. 4 Dimensiones robot humanoide Nao [10].

- Posee tres marcos de referencia: Marco de referencia del torso, marco de referencia del robot y marco de referencia mundo, los cuales se pueden observar en la Figura 1.5.
    - **Marco de referencia del torso:** se adjunta a la referencia del torso del robot, por lo que se mueve con el robot mientras camina y cambia de orientación

cuando se inclina. Este espacio es útil cuando tienes tareas muy locales, que tienen sentido en la orientación del marco del torso[9].

- **Marco de referencia del robot:** este es el promedio de las posiciones de dos pies proyectadas alrededor de un eje z vertical. Este espacio es útil, porque el eje x siempre está hacia adelante, por lo que proporciona una referencia egocéntrica natural [9].
- **Marco de referencia Mundo:** este es un origen fijo que nunca se modifica. Se deja atrás cuando el robot camina y será diferente en la rotación z después de que el robot haya girado. Este espacio es útil para cálculos que requieren un marco de referencia absoluto y externo [9].

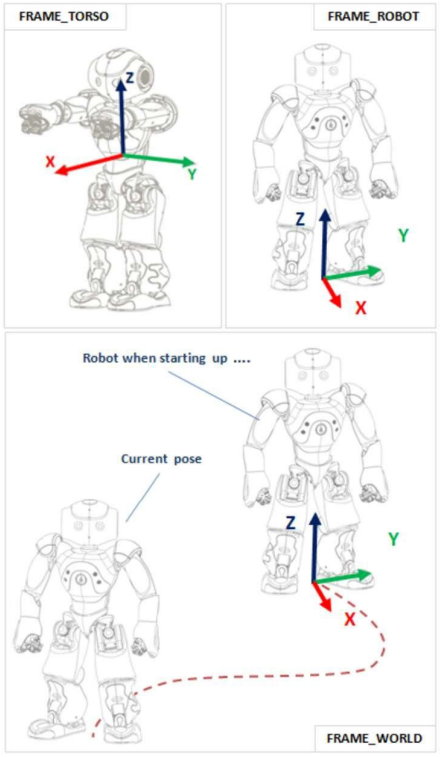


Figura 1. 5 Marco de referencia del robot humanoide Nao [9]

1.3.1.2 Software del robot humanoide Nao V6

En el trayecto de este trabajo, se ha usado diferentes herramientas de software para manejar, programar y simular al robot humanoide Nao V6. A continuación, se describe a los más relevantes:

1.3.1.2.1 NAOqi

NAOqi es el nombre del software principal que corre sobre el robot Nao y hace posible su funcionamiento y control. NAOqi responde a necesidades comunes de robótica que incluyen: paralelismo, recursos, sincronización y eventos. Este sistema permite una

comunicación homogénea entre diferentes módulos del robot (movimiento, audio, video), una programación homogénea y un intercambio de información homogéneo [9]. Entre sus características más importantes se tiene:

- Es multiplataforma, lo cual significa que es posible ejecutarla en diferentes sistemas operativos como Windows, Linux o Mac [9].
- Es de lenguaje cruzado (Figura 1.6), es decir, el software que se ejecuta en el robot puede desarrollarse en C++ o en Python, siendo su lógica de programación similar pero su sintaxis diferentes [9].

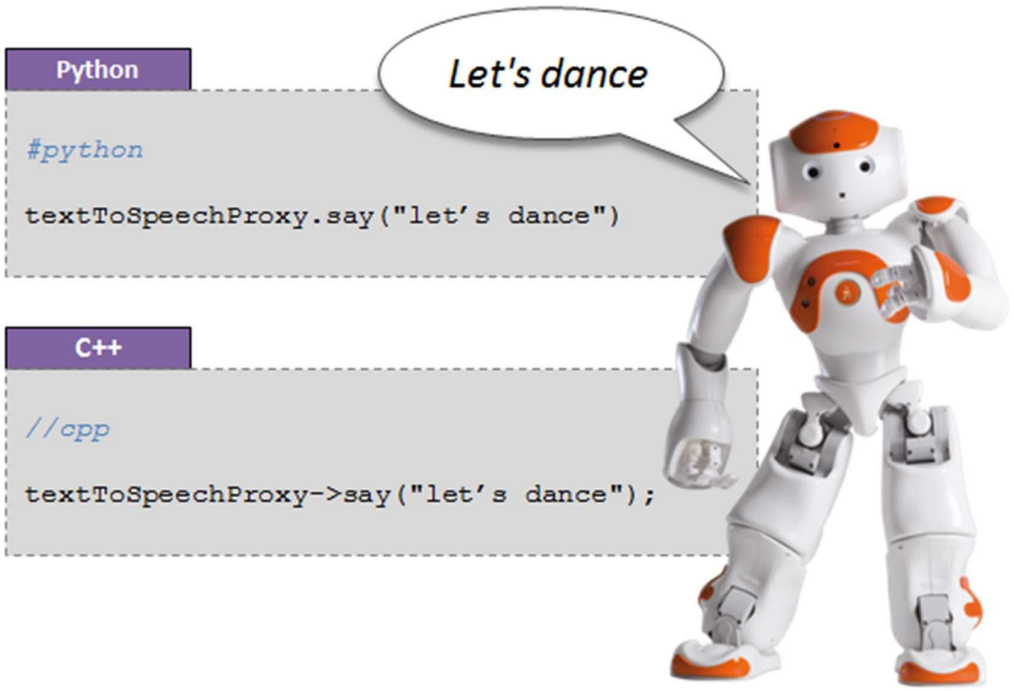


Figura 1. 6 Característica Nao: Lenguaje Cruzado [9].

- El sistema NAOqi proporciona introspección, lo cual significa que el marco sabe qué funciones están disponibles en los diferentes módulos y en dónde.

A continuación, se presenta los conceptos más relevantes necesarios para comprender cómo funciona NAOqi.

Introspección: La introspección es la base o fundamento de la API (Interfaz de programación de aplicaciones) y de las capacidades del robot, así como también, de su monitoreo y acción de las funciones monitoreadas. Gracias a esto, el robot Nao conoce todas las funciones API disponibles [9].

El proceso NAOqi: El ejecutable NAOqi es un bróker (también conocido como intermediario) que cuando se inicia carga un archivo de preferencia llamado autoload.ini,

el cual define qué bibliotecas se debe cargar. Cada biblioteca contiene uno o más módulos que usan dicho bróker para anunciar sus métodos. En la Figura 1.7 se puede apreciar este proceso [9].

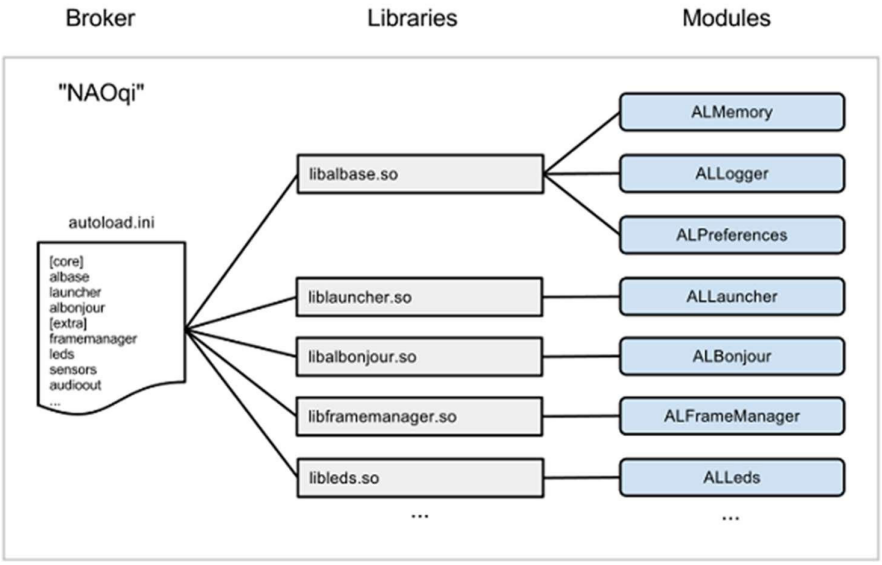


Figura 1. 7 Proceso NAOqi [9].

Bróker: El bróker proporciona servicios de búsqueda para que cualquier método en el árbol o en la red pueda encontrar cualquier método que se haya anunciado. La carga de módulos forma un árbol de métodos adjuntos a módulos y módulos adjuntos a un bróker, como se puede apreciar en la Figura 1.8 [9].

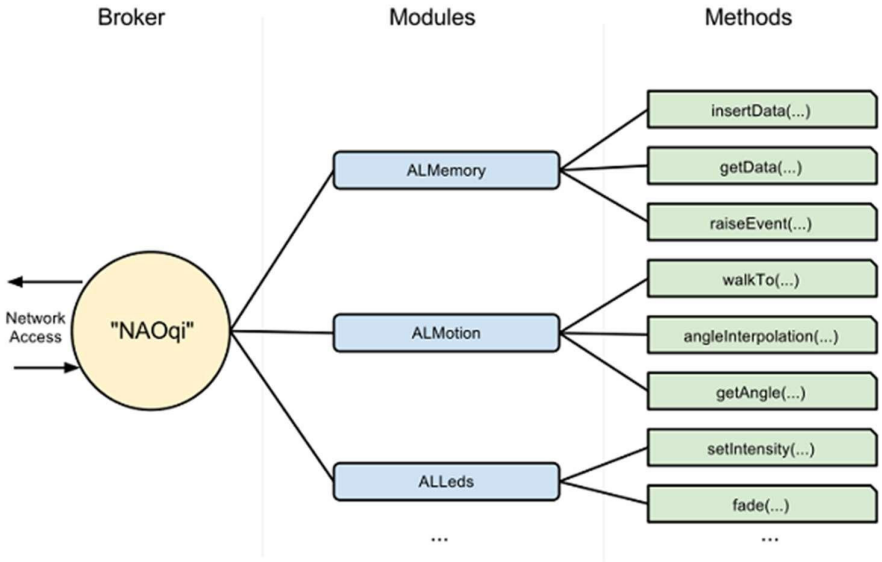


Figura 1. 8 Árbol de métodos adjuntos a módulos y módulos adjunto a bróker [9].

Proxy: Un proxy es un objeto que se comportará tal y como lo hace el módulo al que representa, por ejemplos, si al crear un proxy para el módulo ALMotion, se obtendrá un objeto que contenga todos los métodos ALMotion [9].

Para crear un proxy para un módulo (y por lo tanto llamar a los métodos de un módulo) se tiene dos opciones:

- Usando el nombre del módulo. En este caso, el código que se está ejecutando y el módulo al que se desea conectar deben estar en el *mismo* bróker. A esto se lo conoce como llamada *local* [9].
- Utilizando el nombre del módulo y la IP y puerto de un bróker. En este caso, el módulo debe estar en el bróker correspondiente [9].

Módulos: Normalmente, cada módulo es una clase dentro de una biblioteca. Cuando la biblioteca se carga desde autoload.ini, automáticamente se creará una instancia de la clase del módulo [9].

En el constructor de una clase que deriva de ALModule, se puede "vincular" métodos. Esto anuncia sus nombres y firmas de métodos al bróker para que estén disponibles para otros.

Un módulo puede ser remoto o local.

- Si es remoto, se compila como un archivo ejecutable y se puede ejecutar fuera del robot. Los módulos remotos son más fáciles de usar y se pueden depurar fácilmente desde el exterior, pero son menos eficientes en términos de velocidad y uso de memoria [9].
- Si es local, se compila como una biblioteca y solo se puede usar en el robot. Sin embargo, son más eficientes que un módulo remoto [9].

Cada módulo contiene varios métodos. Entre ellos, algunos métodos están vinculados, lo que significa que se pueden llamar desde fuera del módulo, por ejemplo, dentro de otro módulo, desde un ejecutable, etc. La forma de llamar a estas funciones vinculadas no varía si el módulo es remoto o local: el módulo se adapta automáticamente [9].

Como se ha mencionado anteriormente, es posible desarrollar toda la programación de control del robot Nao V6 desde el lenguaje de programación Python, por lo que es importante conocer sus características principales.

1.3.1.2.2 Python

Python es un lenguaje de programación de propósito general originalmente creado por Guido van Rossum en 1991, el cual se caracteriza por ser independiente de plataforma,

Interactivo, orientado a objetos e interpretado, es decir, no es necesario compilar el código fuente antes de su ejecución. En los últimos años, este lenguaje ha ido ganando popularidad entre programadores alrededor del mundo, ya que presenta varias ventajas frente a otros lenguajes de programación, de las cuales se puede destacar [12] :

- El gran número de librerías que posee, los tipos de datos y funciones incorporadas dentro del mismo lenguaje, las cuales ayudan a realizar numerosas tareas rutinarias o habituales sin la necesidad de tener que programar desde el inicio [12].
- Su velocidad y sencillez con la que crea diferentes programas. Un programa en Python puede llegar a tener de 3 a 5 líneas de código menos a un programa escrito en C o Java [12].
- La cantidad de plataformas en las que se puede desarrollar este lenguaje como Unix, Windows, Mac, entre otras [12].
- Su gratuidad, incluso para propósitos empresariales [12].
- Sintaxis clara en comparación a otros lenguajes de programación [12].

Debido a las ventajas recién establecidas, se ha escogido este lenguaje para programar al robot Nao. Para esto, es necesario hacer uso de un SDK (Software Development Kit - Kit de desarrollo de software) de Python, que no es más que un conjunto de herramientas de software y programas usados por desarrolladores para crear aplicaciones para plataformas específicas. Estas herramientas contienen un gran número de bibliotecas, documentación, ejemplos de código y guías que los programadores pueden usar e integrar en sus propias aplicaciones [13].

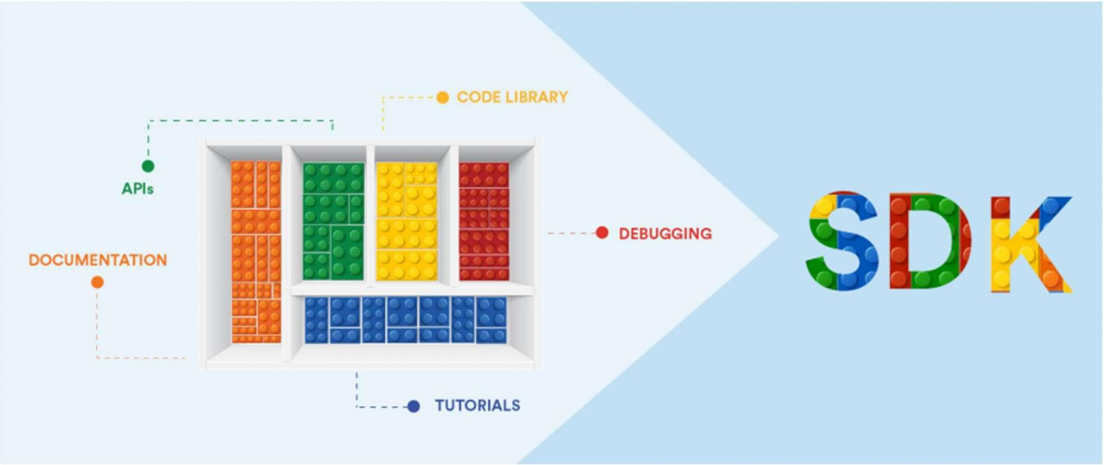


Figura 1. 9 Contenido SDK [13].

Como se puede apreciar en la Figura 1.9, uno de los componentes principales de un SDK son los API's (Application Programming Interfaces - Interfaces de programación de aplicaciones), los cuales están encargados de definir la interacción entre múltiples

softwares intermediarios, es decir, permiten que un software determinado interactúe con otro software [13]. El API de Python para Nao permite usar toda la API de C++ desde una máquina remota, o crear módulos de Python que puedan ser ejecutado de formar remota o directamente en el robot [9]. Cabe recalcar que cada uno de estos módulos hereda los métodos del API correspondiente. Todos y cada uno de los API's del robot NAO cumple con diferentes funciones importantes, y por tal razón, se los ha dividido en diferentes secciones, dependiendo de su funcionalidad: Core (Núcleo), Emotion (Emoción), Interaction engines (motores de interacción), Motion (Movimiento), Audio, Vision (Visión), People Perception (Percepción de Personas), Sensor & LEDs (Sensores y LEDs), LoLA [9].

1.3.1.2.3 CoppeliaSim Edu (V-REP)

CoppeliaSim es un potente simulador de robots multiplataforma, con entorno de desarrollo integrado, que se basa en una arquitectura de control distribuido. Gracias a varias de sus características y ventajas, CoppeliaSim está posesionado como uno de los mejores simuladores en el mercado, siendo su logo y portada, Figura 1.10, reconocido por varios ingenieros aficionados a la robótica. Una de las ventajas más significativas que proporciona ese simulador es que proporciona un marco unificado que combina muchas bibliotecas internas y externas potentes que a menudo son útiles para simulaciones de robótica. Esto incluye motores de simulación dinámica, herramientas de cinemática directa e inversa, bibliotecas de detección de colisión, simulaciones de sensores de visión, planificación de rutas, herramientas de desarrollo de GUI y modelos integrados de muchos robots comunes. CoppeliaSim es un software multiplataforma, en su mayoría de código abierto, el cual proporciona una licencia educativa gratuita [14].

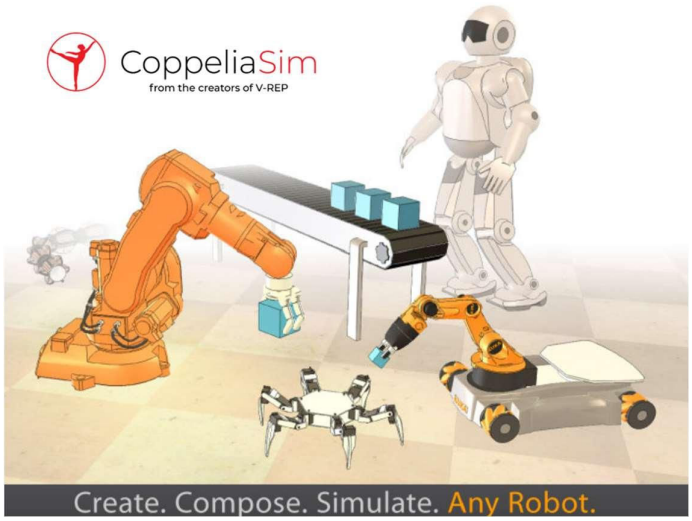


Figura 1. 10 Portada CoppeliaSim [14].

Entre las aplicaciones de CoppeliaSim más significativas, se tiene:

- Simulación de sistemas de automatización industrial
- Monitoreo remoto
- Control de hardware
- Monitoreo de seguridad
- Desarrollo rápido de algoritmos
- Educación relacionada con la robótica
- Presentación de producto

CoppeliaSim se puede utilizar como una aplicación independiente o se puede integrar fácilmente en una aplicación cliente principal: su pequeña huella y su API elaborada hacen de CoppeliaSim un candidato ideal para integrarse en aplicaciones de nivel superior. Un intérprete de scripts Lua integrado (Lua es un lenguaje de programación de extensión que ofrece un buen soporte para la programación orientada a objetos, programación funcional y programación orientada a objetos [15]) hace de CoppeliaSim una aplicación extremadamente versátil, dejando la libertad al usuario de combinar las funcionalidades de bajo / alto nivel para obtener nuevas funcionalidades de alto nivel [14].

CoppeliaSim se basa en una arquitectura de control distribuido, es decir, cada objeto / modelo se puede controlar individualmente mediante un script integrado, un complemento, nodos ROS / ROS2, nodos BlueZero, clientes API remotos o una solución personalizada. Esto hace que CoppeliaSim sea muy versátil e ideal para aplicaciones de múltiples robots. Los controladores se pueden escribir en C / C ++, Python, Java, Lua, Matlab, Octave o Urbi [14].

API CoppeliaSim:

Como se puede observar en la Figura 1.11, el marco o framework de las API's de CoppeliaSim agrupa todas las interfaces alrededor de CoppeliaSim, Se dividen en 5+1 diferentes interfaces:

- La API regular.
- La API remota.
- Las interfaces de ROS.
- La interface de BlueZero.
- La API auxiliar.
- Otras interfaces.

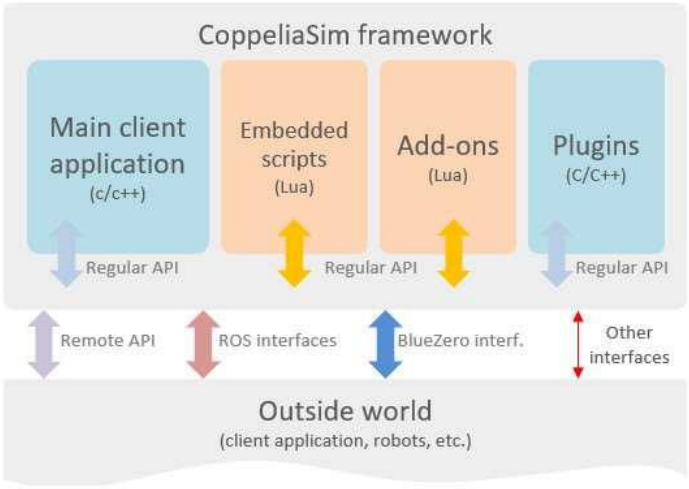


Figura 1. 11 Marco de las API's de CoppeliaSim [14].

Para este proyecto técnico, se ha implementado la API remota o "Remote API", la cual permite la comunicación entre CoppeliaSim y alguna aplicación externa (es decir, una aplicación que se ejecuta en un proceso o en una máquina diferentes) que, en este caso, sería Python (o más específicamente, su intérprete).

1.3.2 ROBÓTICA MÓVIL

Antes de indagar sobre la teoría de controladores que se implementarán sobre el robot humanoide Nao, es necesario primeramente familiarizarse con los conceptos básicos y las ecuaciones que moldean a la robótica móvil. La robótica móvil se caracteriza por estar dotado de un sistema de locomoción (patas, orugas o ruedas) mediante el cual, el robot puede navegar por un ambiente de trabajo determinado. Los robots con ruedas se utilizan en superficies planas, mientras que los robots con orugas o patas se emplean en superficies irregulares o muy irregulares.

Para cualquier tipo de robot móvil, se puede tomar como referencia una partícula en movimiento para poder obtener sus ecuaciones cinemáticas. Con el fin de entender de mejor manera dichas ecuaciones, se va a tomar como referencia a un robot de tracción diferencial (robot de dos ruedas motrices y una o más ruedas locas, Figura 1.12). En dicho caso, las variables de estado del robot serán: posición x, posición y, y su orientación φ, dando como origen al modelo cinemático cartesiano del robot móvil que se representa con la ecuación 1.1, ecuación 1.2 y ecuación 1.3, cuyas variables de entrada son u (velocidad lineal) y w (velocidad angular) [16].

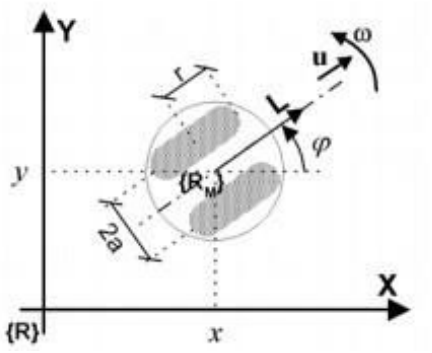


Figura 1. 12. Posición y orientación del robot móvil expresados en coordenadas cartesianas [16].

$$\dot{x} = u\cos(\varphi) \tag{1.1}$$

$$\dot{y} = u\sin(\varphi) \tag{1.2}$$

$$\dot{\varphi} = \omega \tag{1.3}$$

Al tener el modelo cinemático del robot móvil, tanto en coordenadas polares, como en coordenadas cartesianas, Figura 1.13, se emplea una relación que permita representar ambos sistemas de ecuaciones, dando como resultado el conjunto de ecuaciones: ecuación 1.4, ecuación 1.5 y ecuación 1.6 [16].

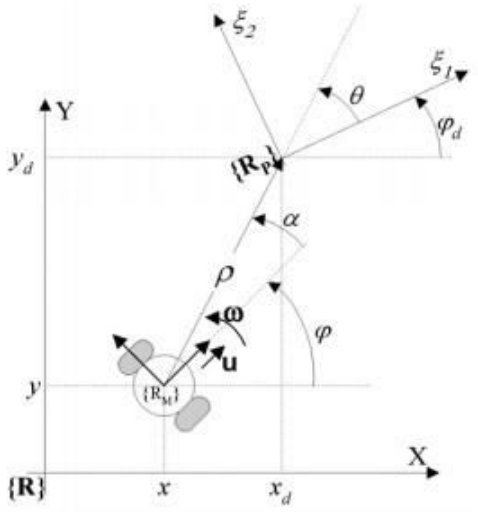


Figura 1. 13 Posición y orientación del robot móvil de dos marcos de referencia {R} coordenadas cartesianas y {Rp} coordenadas polares [16].

$$\rho = \sqrt{(x_d - x)^2 + (y_d - y)^2} \tag{1.4}$$

$$\theta = arctan[(y_d - y), (x_d - x)] - \varphi_d \tag{1.5}$$

$$\alpha = arctan[(y_d - y), (x_d - x)] - \varphi \tag{1.6}$$

1.3.3 VARIABLES Y ESTRATEGIAS DE CONTROL

En el siguiente apartado, se revisa las variables a controlar para poder desplazar al robot humanoide Nao en su espacio de trabajo, así como también el fundamento teórico de cada una de las estrategias de control empleadas.

1.3.2.1 Variables de control de un robot móvil.

Para poder controlar el desplazamiento del robot humanoide Nao desde un punto A hacia un punto B sobre un camino predeterminado, se considera el control a bajo nivel como una caja negra, Figura 1.14, es decir, que tanto su cinemática directa, cinemática inversa y dinámica del robot ya se han tomado en cuenta dentro de las diferentes librerías de NAOqi. Para poder controlar el desplazamiento del robot Nao, se debe monitorear su orientación y posición cada tiempo de muestreo y, de existir algún error, se los corrige mediante las acciones de control tanto de velocidad lineal, como de velocidad angular. Para controlar dichas velocidades, se debe acceder al método "move (x, y, w)", del módulo "ALMotion", donde x es la velocidad lineal del robot en el eje x, con respecto al eje de referencia del torso del robot, y es la velocidad lineal del robot en el eje y con respecto al eje de referencia del torso del robot y w es la velocidad angular del robot en el eje z con respecto al eje de referencia del torso del robot.

Figura 1. 14 Representación del robot como caja negra [17].

Para poder estimar la posición relativa del robot (x, y, θ) dentro del espacio de trabajo, y orientarlo hacia una dirección deseada, se hace uso de la odometría del robot, Figura 1.14.

1.3.2.2 Controlador PID

Para poder controlar cualquier variable de cualquier planta de manera eficiente, es necesario realizar una realimentación de la variable de salida y compararla con la variable de referencia de entrada y corregir el error entre estas dos. A este tipo de control se lo conoce como control en lazo cerrado y existen varias técnicas o herramientas para disminuir o incluso eliminar el error presentado. En la Figura 1.15 se puede observar un esquema clásico de control en lazo cerrado [18].

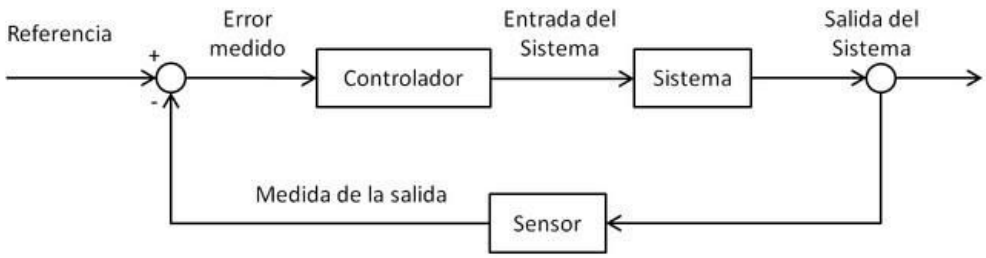


Figura 1. 15 Esquema clásico de control en lazo cerrado.

Uno de los tipos de controladores más conocidos y usados en la mayoría de las aplicaciones de control de procesos automáticos son los controladores tipo PID [19]. El controlador PID es aquel que combina las tres acciones clásicas de control: proporcional, derivativa e integral. Un controlador tipo PID toma el error de la planta y calcula la acción P, PI, PD o PID dependiendo de la calibración de sus parámetros y de la planta [17]. Este controlador es muy conocido a nivel industrial debido a su fácil implementación, y su gran número de métodos de sintonización [18]. La expresión de un controlador con esta acción combinada se obtiene mediante la ecuación 1.7:

$$u(t) = kp * (e(t) + \frac{1}{T_i}\int_0^t e(t)dt + T_d\frac{de(t)}{d(t)}) \qquad (1.7)$$

Donde:

Kp es la constante proporcional del controlador.

Ti es el tiempo de asentamiento de la curva de reacción de la planta.

Td es el tiempo de retardo de la curva de reacción de la planta.

En este proyecto técnico, se ha optado por usar dos tipos de controladores: un controlador proporcional y un controlador proporcional derivativo. A continuación, una explicación breve este tipo de controladores.

1.3.2.2.1. Control Proporcional (P)

El controlador proporcional se aplica cuando se desea corregir el error en estado estable al multiplicar dicho error por una constate proporcional [17]. Es decir, la salida depende directamente del error. Sin embargo, la respuesta de este controlador es muy limitada, por lo que, en mucho de los casos, no se logra eliminar el error por completo, para esto, es necesario el uso de controladores integral, derivativo o incluso una combinación de los tres dependiendo de la planta [18]. La ley de control que define al control proporcional se la expresa en la ecuación 1.8:

$$u(t) = kp * e(t) \qquad (1.8)$$

1.3.2.2.2. Control proporcional derivativo (PD)

Al añadir una acción derivativa a un control proporcional, se está mejorando la respuesta estacionaria del sistema, ya que esta acción anticipa el efecto producido por el controlador proporcional; es decir, esta combinación proporcional, derivativa permite que la variable controlada se estabilice en un menor tiempo [18]. Sin embargo, en estado estable, esta acción derivativa no tiene efecto alguno, ya que solamente actúa cuando el error varía. Se debe tomar en cuenta que, cuando se trabaja con una acción derivativa, existe el riesgo de derivar ruido obtenido en el proceso de control, haciendo que este se incremente y alcance valores muy grandes [18]. La ley de control que define al control proporcional, derivativo se la expresa en la ecuación 1.9:

$$u(t) = kp * (e(t) + T_d \frac{de(t)}{d(t)}) \quad (1.9)$$

En definitiva, una acción de control proporcional, derivativa mejora la respuesta en regímenes transitorios, pero no logra, en la mayoría de los casos, corregir el error en régimen permanente. Para esto, es necesario el uso de otra acción de control conocida como acción integral [18].

1.3.2.2.3. Control proporcional integral (PI)

Al implementar una acción integral a un controlador proporcional, se integran todos los errores pasados y se los usa para una futura ley de control, permitiendo así que la respuesta del sistema tienda cada vez más a cero en cada iteración hasta finalmente tener un error de cero en estado estacionario. Cabe recalcar que, esta acción de control hace al sistema más lento [18].

El controlador PI es usado en un 90% de plantas industriales ya que, como se explicó, elimina el error estacionario por completo y no amplifica el ruido del proceso como es en el caso de un control PD [18]. La ley de control que define al control proporcional, integral se lo expresa en la ecuación 1.10:

$$G_c = kp * (e(t) + \frac{1}{T_i} \int_0^t e(t)dt) \quad (1.10)$$

1.3.2.3 Controlador basado en Lyapunov

Para comprender de mejor manera como se diseña un controlador basado en Lyapunov, es importante primero conocer los fundamentos básicos del análisis de estabilidad de un sistema.

La teoría de Lyapunov juega un papel muy importante en el análisis de estabilidad de los sistemas de control descritos por ecuaciones en espacio de estado [20], por ejemplo, para este caso particular de interés, sistemas de control para robots móviles. El objetivo de realizar un análisis de estabilidad sobre un sistema de control es para conocer si este puede ser aplicado o no sobre una planta real [21]. Sin embargo, la gran mayoría de plantas en la industria tienen un comportamiento no lineal, por lo cual, no es posible determinar dicha estabilidad por medio de métodos tradicionales como el criterio de estabilidad de Jury o el criterio de estabilidad de Routh-Hurwitz [20]. Es ahí, donde el criterio de estabilidad de Lyapunov es de bastante ayuda, ya que, esta herramienta es aplicable tanto a sistemas lineales, como no lineales, variantes e invariantes en el tiempo [21].

Existen dos métodos de análisis de estabilidad de Lyapunov, el método indirecto y el método directo. El método indirecto está formado por procedimientos en los que se utilizan las formas explicitas de las soluciones de las ecuaciones diferenciales o de las ecuaciones en diferencias para el análisis. El método directo no requiere de las soluciones de las ecuaciones diferencia o en diferencias por lo que resulta más útil en la práctica [20]. Para el desarrollo de este proyecto técnico, se enfocará la atención en este segundo método.

La filosofía básica del método directo de Lyapunov es la extensión matemática de una observación física fundamental: si la energía total de un sistema mecánico o eléctrico es disipada en forma continua, entonces el sistema lineal o no lineal debe quedar eventualmente en un punto de equilibrio. Así, se puede determinar la estabilidad de un sistema mediante el análisis de la variación de una simple función escalar $V(t,x)$ conocida como función Lyapunov [20].

Para que un sistema posea un estado de equilibrio en el origen, y este estado sea uniforme y asintóticamente estable, se debe cumplir con las siguientes condiciones [20]:

1. $V(t,x)$ es continua y posee derivadas continuas
2. $V(t,x)$ es definida positiva o $V(x) > 0$
3. $\dot{V}(t,x)$ es definida negativa o $\dot{V}(x) < 0$
4. $V(t,0) = 0$

En resumen, estas condiciones dictan que, si el sistema tiene un estado de equilibrio asintóticamente estable, entonces la energía almacenada en él, desplazada dentro del dominio de atracción, se decrementa al aumentar el tiempo, hasta que por ultimo adopta su valor mínimo en el estado de equilibrio [20].

Ahora que se conoce más a fondo el análisis de estabilidad de un sistema de acuerdo al criterio de Lyapunov y las condiciones que se debe cumplir para que este se encuentre en

un estado de equilibrio asintóticamente estable, se explica a continuación los pasos para el diseño de un controlador basado en la estabilidad de Lyapunov.

Paso 1: Seleccionar una función Lyapunov $V(t,x)$ de prueba (candidata) que cumpla con todas las condiciones anteriormente establecidas[22].

Paso 2: Derivar la ecuación para la derivada $\dot{V}(t,x)$ a lo largo de la trayectoria del sistema:

$$\dot{x} = f(x,u,t) \tag{1.11}$$

Y seleccionar una ley de control de retroalimentación (feedback law control):

$$u = u(x) \tag{1.12}$$

Lo cual asegura que:

$$\frac{dV(x)}{dt} < 0 \; for \; x \neq 0 \tag{1.13}$$

Típicamente, $u(x)$ es una función no lineal de x que contiene algunos parámetros y ganancias que pueden ser seleccionadas para hacer dV/dt<0, y por lo tanto asegurar que el sistema de circuito cerrado sea asintóticamente estable [22].

1.3.3 PLANIFICACIÓN DE CAMINOS

Uno de los más grandes problemas a los que se deben enfrentan los robots móviles es desplazarse libremente a través de espacios de trabajo determinados. Muchas de las veces, dichos espacios no se encuentra libre de obstáculos, por lo que es de suma importancia desarrollar un algoritmo capaz de evadirlos efectivamente, evitando así cualquier tipo de colisión [23].

En los últimos años, se ha ido desarrollando una gran variedad de algoritmos de planificación de caminos, siendo los más conocidos los campos potenciales artificiales, el algoritmo de caminos mínimos, el grafos de visibilidad, los diagramas de Voronoi, los mapas de caminos probabilísticos (Pobabilistic Road Maps - PRM) y la Exploración Rápida de Árbol Aleatorio (Rapidly-exploring Ramdom Trees - RRT) [23]. Para la planificación de caminos del robot Nao sobre su espacio de trabajo en CoppeliSim, se ha decidido trabajar con el algoritmo RRT, el cual se explica a continuación.

1.3.3.1 Algoritmo de exploración rápida de árbol aleatorio –RRT

Se define a un algoritmo de exploración rápida de árbol aleatorio (RRT por sus siglas en inglés) como una estructura de datos aleatorios que se aplican para resolver problemas en la planificación de caminos con restricción holonómicas, no-holonómicas, dinámicas y altos grados de libertad [24]. El objetivo original de este algoritmo es la de construir un árbol de exploración que cubra de manera uniforme gran parte del espacio libre de colisión, como

se puede observar en la Figura 1.16. Para construir este árbol, el algoritmo debe empezar desde un punto de origen, y partir de ahí ir expandiéndose alrededor de su espacio de trabajo C [25].

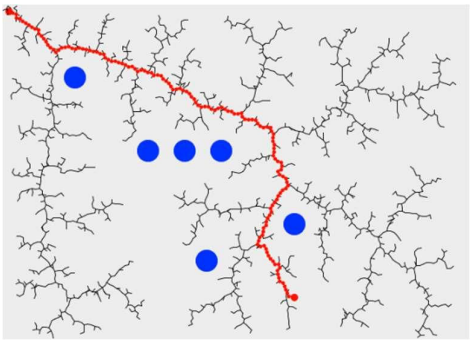


Figura 1. 16 Ejemplo algoritmo RRT en un espacio de trabajo determinado [24]

Con el objetivo de entender de mejor manera cómo funciona este algoritmo, se usarán los siguientes conceptos:

- Se usará "configuración" para denotar un puntos de coordenadas x, y sobre un espacio de trabajo.
- C es el conjunto de todas las posibles configuraciones del robot en un espacio de trabajo determinado.
- C_{free} es el subconjunto de C de aquellas configuraciones que no intersectan o chocan con alguno de los obstáculos existentes en el espacio de trabajo.
- R es la métrica definida dentro del conjunto C. Puede ser cualquier ponderación de proximidad como la distancia euclídea u otra que pueda interesar.
- τ es conocido como el vector árbol en donde se guardan cada una de las configuraciones originadas por el algoritmo.
- q_{init} es la configuración o punto inicial en el que se encuentra ubicado el robot, que en el caso de este trabajo, son las coordenadas x, y del robot con respecto al marco de referencia general y la orientación con respecto al eje de referencia z.
- q_{fin} es el punto o configuración que se desea alcanzar.
- q_{rand} es una configuración aleatoria que es generada por el algoritmo.
- q_{near} es la configuración más próxima a q_{rand}.
- q_{new} es una nueva configuración que se añadirá al árbol.
- ϵ es la longitud del segmento de crecimiento, la cual también se la puede ver como la distancia entre el punto de un determinado árbol con el siguiente al que se está conectado [25].

El algoritmo RRT se encarga de colocar el primer elemento del árbol q_{init}(punto de inicio o de partida) para luego, dentro de un bucle limitado por un número prefijado de iteraciones K_{max}, trate de encontrar la configuración o punto final q_{fin}. Cabe destacar que, en el caso de que no se alcance la configuración final, K_{max} se encargará de detener el algoritmo. Dentro del bucle, se realizan dos tareas en específico [25]:

1. Primeramente, se obtiene un punto o configuración aleatorio q_{rand}, el cual se encuentra dentro del espacio libre de obstáculos o colisiones C_{free}[25].
2. Luego, se hace crecer el árbol en dirección al punto o configuración predefinida con anterioridad [25].

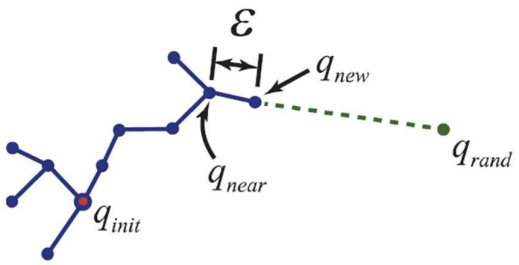


Figura 1. 17 Crecimiento del árbol [25]

En la Figura 1.17 se puede observar de manera resumida cómo funciona el algoritmo RRT. Al aplicar la métrica R a cada uno de los vértices del árbol, se obtiene el punto más cercano a q_{rand} conocido como q_{near}. Luego, se agrega al vector árbol el punto q_{new} siempre y cuando este no se encuentre ubicado dentro del espacio de colisiones. Cabe destacar que ϵ es la distancia entre q_{near} y q_{new} y siempre estará dirigido hacia la dirección q_{rand} [23].

Propiedades de RRTs:

Las principales ventajas de RRT son:

1. La expansión de un RRT se basa principalmente en porciones inexploradas del espacio de configuraciones.
2. La distribución de vértices en un RRT se acerca a la distribución muestral, lo que lleva a un comportamiento consistente o coherente.
3. Un RRT es probabilísticamente completo en condiciones muy generales
4. El algoritmo RRT es relativamente simple, lo que facilita el análisis de desempeño (esta también es una característica preferida de las hojas de ruta probabilísticas)
5. Un RRT siempre permanece conectado, aunque el número de aristas sea mínimo
6. Un RRT puede considerarse como un módulo de planificación de rutas, que puede adaptarse e incorporarse en una amplia variedad de sistemas de planificación.

7. Se pueden construir algoritmos de planificación de trayectos completos sin necesidad de tener la capacidad de dirigir el sistema entre dos estados prescritos, lo que amplía enormemente la aplicabilidad de los RRT [24].

1.3.4 INDICES DE DESEMPEÑO DE UN SISTEMA CONTROLADO

El índice de desempeño considera las características más importantes de la salida del sistema para realizar una medida del rendimiento del proceso. En lazos de control, el desempeño se describe en términos de estabilidad, sensibilidad, exactitud, respuesta transitoria y ruido residual [26].

Dado a que estos índices de desempeño no son determinados con exactitud, es mejor que estos sean lo más insensibles ante variación de parámetros. Es importante recalcar además, que los criterios de desempeño se definen como función de tiempo y el error [26].

Al efectuar un análisis de cada uno de estos indicies de rendimiento, es posible determinar que controlador brinda un mejor rendimiento para el proceso que se está efectuando [27].Estos índices son considerados como una medida cuantitativa del desempeño del sistema, y los que más se utilizan son:

Integral del Error Absoluto IAE

La integral del error absoluto IAE está definida por la ecuación 1.14:

$$IAE = \int_0^\infty [e(t)]\, dt \tag{1.14}$$

Para considerar que el sistema controlador tenga un buen desempeño, el IAE debe tender a cero. Además, cabe recalcar que, a pesar de que este índice proporciona una respuesta aceptable a la salida del lazo de control, no es capaz de optimizar sistemas sobreamortiguados [27].

Integral del Error Cuadrático ISE

El índice de la integral del cuadrado del error se basa en el área del cuadrado de la función de error y se caracteriza por penalizar tanto valores positivos como negativos [27]. Además, no se es muy sensible a la variación de parámetros y presenta como ventaja de que puede ser calculado sin dificultad [26]. Este índice está definido por la ecuación 1.15:

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

Este criterio da mayor ponderación a los errores elevados del sistema, que comúnmente se dan al inicio de la respuesta, y con menos peso significativo evalúa los errores pequeños que se dan al final de la respuesta [27].

TVu (Variaciones totales de esfuerzo de control)

A pesar de que nuestro objetivo principal es de optimizar el desempeño del seguimiento de camino efectuado por el robot NAO, no hay que dejar de lado el análisis del esfuerzo de control, es decir la señal de salida de nuestros controladores. Por lo tanto, se debe procurar que dichas señales de control no posean cambios bruscos ni extremos, para así evitar el deterior prematuro de los elementos final de control los cuales permiten que el robot Nao pueda desplazarse [28]. La evaluación de dichas variaciones de las señales de control se las puede realizar mediante el índice de rendimiento TVu, como se muestrea en la ecuación 1.16:

$$TV_u = \sum_{k=1}^{N} |u_{k+1} - u_k| \qquad (1.16)$$

Donde u_k y u_{k+1} son la señal de control presente y futura respectivamente.

Al calcular el valor de tvu de un controlador determinado, si el valor es pequeño, nos muestra la existencia de variaciones suaves de la señal de control [28].

1.3.5 INTERFAZ GRÁFICA –QT CREATOR.

Hoy en día, una comunicación o interacción hombre-máquina es de gran importancia para poder desarrollar de manera eficiente diferentes procesos. Es por tal motivo, que se debe procurar, mediante una interfaz gráfica, facilitar al máximo dicha comunicación [17].

Para el presente trabajo, se ha decidido utilizar el software Qt Creator para diseñar la interfaz gráfica, ya que su uso es fácil e intuitivo, además de presentar grandes ventajas frente a otras herramientas.
Qt Creator proporciona un entorno de desarrollo integrado (IDE) completo y provee las herramientas necesarias para poder diseñar una interfaz gráfica que cumpla con todos los requerimientos demandados en este trabajo. Qt Creator incluye un depurador visual y un layout de GUI integrado [29].

1.3.5.1. Qt designer

Qt Designer es la herramienta Qt para diseñar y construir interfaces gráficas de usuario (GUI) con Qt Widgets. En él, se puede componer y personalizar las ventanas o cuadros de diálogo en una forma de “lo que ve es lo que obtiene” (WYSIWYG – what you see is what you get) y probarlos utilizando diferentes estilos y resoluciones [30].

Los widgets y formularios creados con *Qt Designer se* integran a la perfección con el código programado, utilizando el mecanismo de ranuras y señales de Qt, para que pueda asignar fácilmente el comportamiento a los elementos gráficos. Todas las propiedades

establecidas en *Qt Designer* se pueden cambiar dinámicamente dentro del código. Cabe mencionar que cada una de las características de promoción de los diferentes widgets y plug-ins permiten usar y crear componentes personalizados con funciones específicas [30].

2. METODOLOGÍA

A continuación, se presenta la metodología aplicada en el trabajo de titulación con el objetivo de describir explícitamente el procedimiento desarrollado durante la investigación. Para la recopilación de información, se utiliza tanto fuentes primarias como libros, revistas técnicas y literatura científica en general, así como también fuentes secundarias como trabajos de titulación, artículos académicos, informes relacionados a robótica móvil, etc.

2.1. ODOMETRÍA DEL ROBOT NAO EN EL ENTORNO DE SIMULACIÓN COPPELIASIM

El objetivo de diseñar e implementar controladores en robots móviles radica en garantizar que estos lleguen a su objetivo deseado. Para lograr este objetivo con el robot humanoide NAO en la plataforma de simulación CoppeliaSim, es importante considerar primeramente ciertos aspectos relevantes [31].

1. Posicionamiento: El objetivo es localizar primeramente la posición relativa del robot con respecto a un marco de referencia establecido en CoppeliaSim (CoppeliaSim World Frame) para luego poder ubicarlo en punto de referencia dado, con una orientación deseada. A esta estimación de posición se la conoce como Odometría [31].
2. Seguimientos de caminos: Se requiere que el robot siga un camino establecido, sin ninguna especificación temporal [31].
3. Seguimiento de trayectorias: Se requiere que el robot siga una referencia de trayectoria determinada parametrizada en el tiempo [31].

En el caso de este proyecto técnico, no se considera el último punto como condición de movimiento del robot humanoide.

A continuación, se explica cómo se ha realizado la odometría del robot Nao V6 en la plataforma de simulación CoppeliaSim.

2.1.1. DESARROLLO DEL ENTORNO DE SIMULACIÓN- COPPELIASIM

Como se ha visto en la sección 1.2, con el objetivo de comprobar el funcionamiento del control de movimiento del robot NAO, se ha desarrollado un ambiente de simulación basado en un aula de clase con 3 diferentes disposiciones de pupitres. Para esto, se ha

hecho uso del software de simulación CoppeliaSim, el cual no solamente cuenta con un modelo estándar del robot NAO (Figura 2. 1), sino también, con todo tipo de diferentes accesorios de entorno, como pupitres, sillas, paredes e incluso personas.

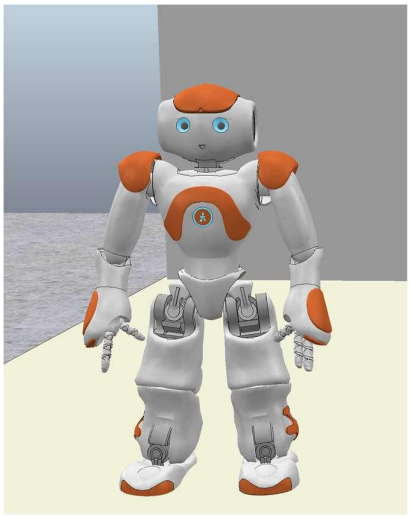


Figura 2. 1 Robot Nao V6 en CoppeliaSim.

Antes de explicar el diseño de cada una de las disposiciones de los pupitres, cabe recalcar que para que este software se pueda comunicar con Python (desde donde se controla al robot), es necesario iniciar el API Remoto dentro del entorno de simulación. Para esto, basta con añadir una línea de código (simRemoteApi.start(19999)) dentro de cualquier objeto, de preferencia estático, en el ambiente de simulación.

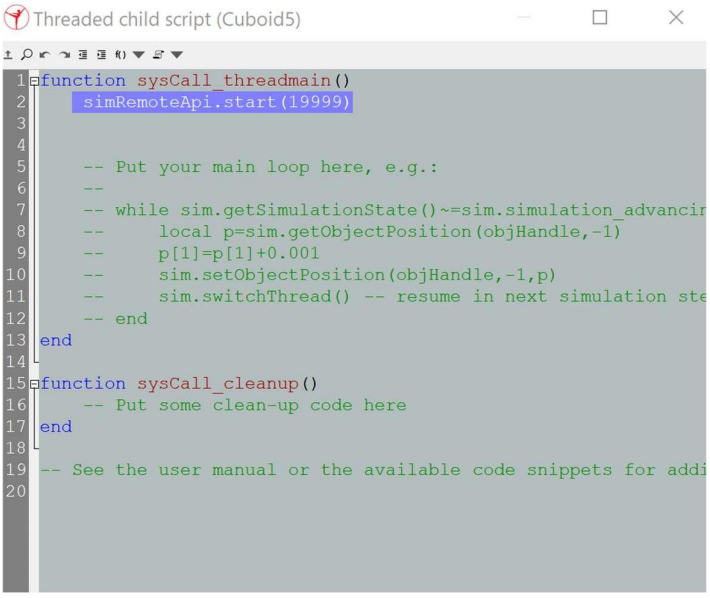


Figura 2. 2 Ejemplo de implementación de código en cuboide.

En la Figura 2. 2, se puede ver un ejemplo de implementación de lo indicado, dentro del código de un cuboide, creado únicamente con el objetivo de dar inicio al API remoto.

Este paso es necesario en cada uno de los ambientes de simulación en donde se vaya a movilizar el robot que, para el caso de este trabajo, serán las tres diferentes disposiciones de pupitres, mostradas a continuación.

Como se puede observar en la Figura 2. 3, para el escenario 1 se ha escogido una disposición tradicional de pupitres ya que la mayoría de las aulas de clase poseen una configuración similar, la cual suele consistir en filas de asientos fijos. Los estudiantes se enfrentan al instructor de espaldas a los demás. Esta disposición de los asientos en el aula es históricamente común en los colegios y universidades, minimizando la comunicación estudiante-alumno y apoyando en gran medida un entorno de aprendizaje "docente en el escenario" [32]. La mayor interacción comunicativa entre profesores y estudiantes ocurre típicamente con los estudiantes en la primera fila o en el medio del aula. Es más probable que los estudiantes de las filas traseras estén menos comprometidos.

Cabe recalcar que, dentro de este espacio, el robot tendrá menos espacio para movilizarse que en los otros dos escenarios.

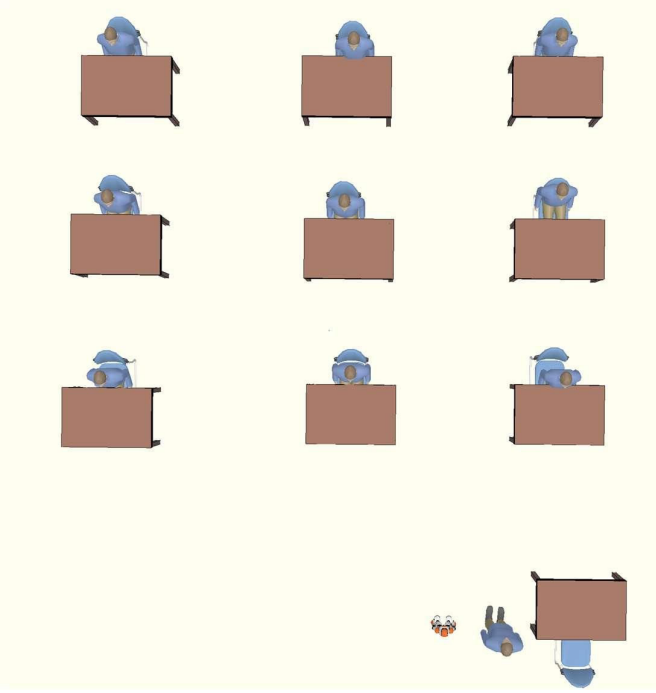


Figura 2. 3 Vista superior de escenario 1.

En el escenario 2, Figura 2. 4, se ha escogido una disposición similar a una sala de reuniones convencional, donde las mesas se encuentran formando un semi-ovalo. Este

escenario ofrece una configuración modificada de mesa redonda, donde todos los participantes se enfrentan entre sí mientras el instructor puede moverse por la sala. La herradura fomenta la discusión entre los estudiantes y con el instructor, aunque esta configuración tiende a fomentar un mayor compromiso entre el instructor y los estudiantes directamente opuestos, con cantidades ligeramente menores para los estudiantes inmediatamente adyacentes al instructor. Una configuración en forma de semi-ovalo puede ser particularmente eficaz cuando el instructor desea proyectar y discutir el material relacionado con el curso en la parte delantera de la clase [32].

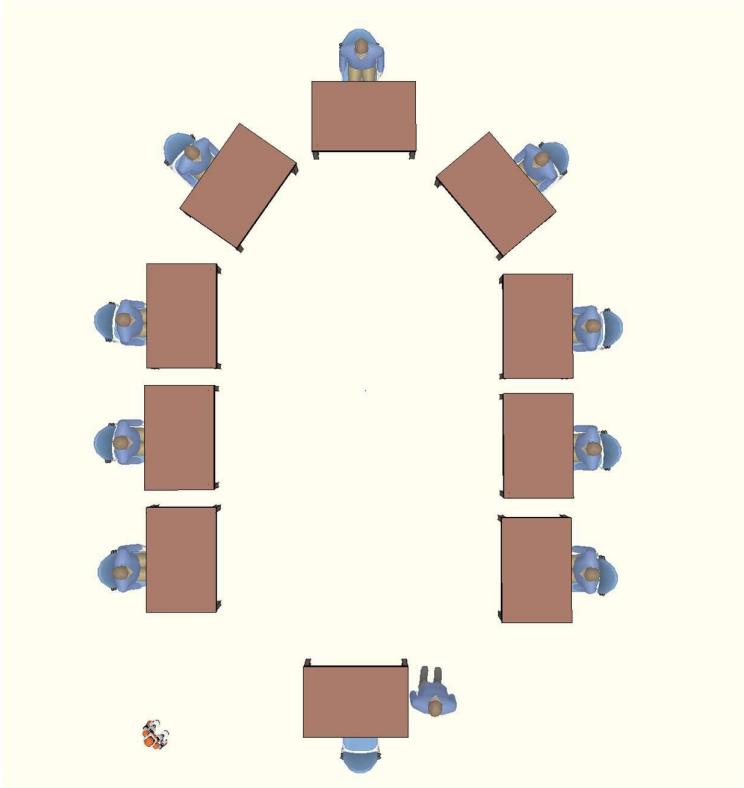


Figura 2. 4 Vista superior de escenario 2.

Finalmente, en la Figura 2. 5, podemos observar el escenario 3, en donde se puede apreciar una disposición de pupitres dividida en dos semicírculos diferentes. Este tipo de disposición puede ser diseñada con mesas rectangulares, circulares o trapezoidales, o escritorios individuales. Los instructores pueden colocar varias mesas juntas para formar grupos de estudiantes (por ejemplo, 3 - 4 estudiantes), o pares. Esta disposición puede ser especialmente ventajosa cuando los estudiantes trabajen en grupos o en parejas con sus compañeros de clase durante una gran parte del tiempo de clase. En términos más generales, este arreglo comunica una comunidad de aprendizaje en la que se espera que los estudiantes trabajen unos con otros [32].

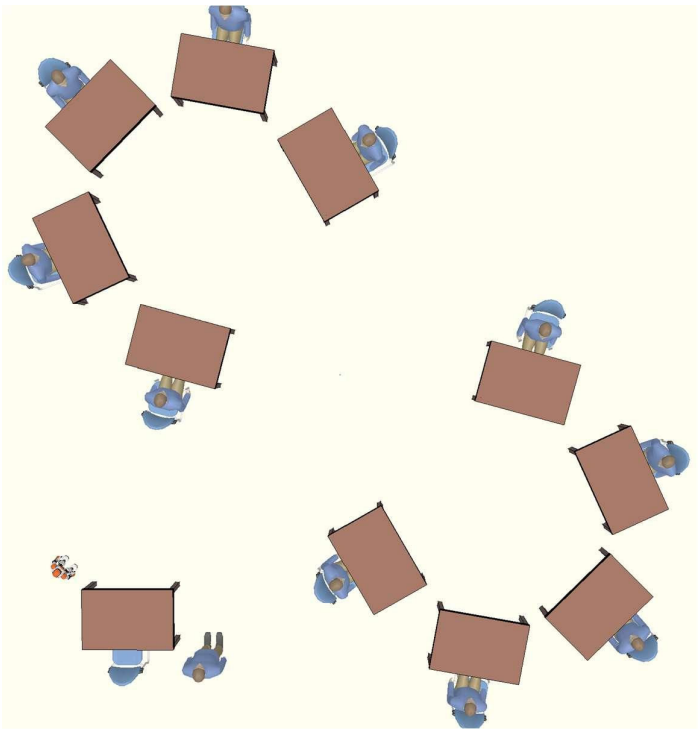


Figura 2. 5 Vista superior de escenario 3.

Cabe recalcar que, en cada uno de los escenarios, el robot Nao tendrá una ubicación inicial de donde podrá desplazarse a cualquier punto que elija el usuario.

2.1.2. ODOMETRÍA DEL ROBOT NAO EN EL ENTORNO DE SIMULACIÓN - COPPELIASIM

La plataforma de simulación CoppeliaSim posee un marco de referencia general (World Frame) por medio del cual se puede conocer la posición de diferentes objetos dentro del espacio de trabajo. Como se puede apreciar en la Figura 2. 6, a este marco de referencia se lo ha dividido en 4 diferentes cuadrantes, dependiendo del signo de cada uno de los ejes (eje x, eje y). Un resumen de esta división se puede encontrar en la tabla 2.1.

Tabla 2. 1 División de cuadrantes del marco de referencia en CoppeliaSim

Cuadrante	Eje x	Eje y
I	Positivo (+)	Positivo (+)
II	Negativo (-)	Positivo (+)
III	Negativo (-)	Negativo (-)
IV	Positivo (+)	Negativo (-)

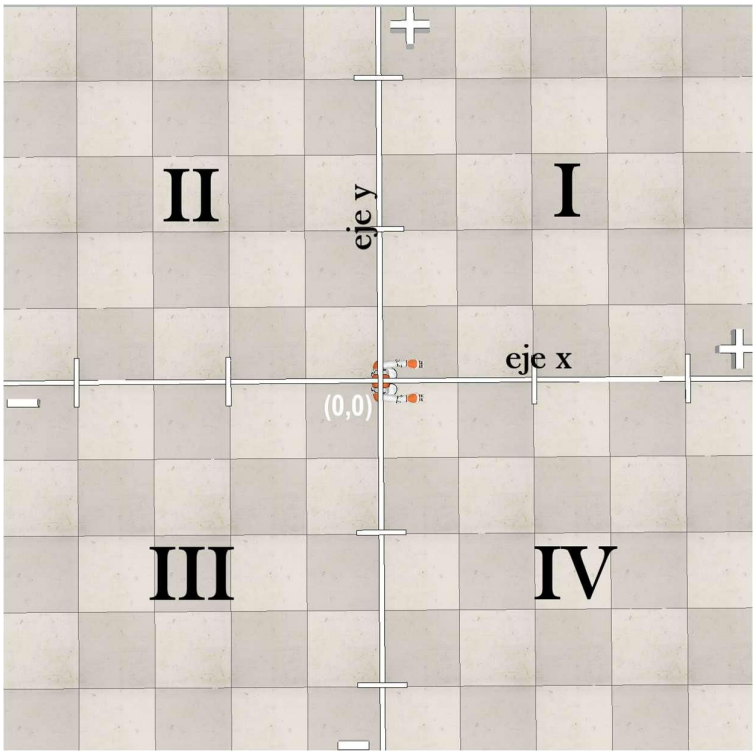


Figura 2. 6 Espacio de trabajo en CoppeliaSim dividido en 4 cuadrantes

Para obtener la posición y orientación real del robot humanoide NAO en CoppeliaSim, se hace uso de las funciones Sim.GetObjectPosition() y Sim.GetObjectAngle(). Con esta información, ya es posible orientar al robot para que pueda desplazarse desde un punto A hacia un punto B, mediante el uso de trigonometría (cambio de coordenadas rectangulares a polares).

Para explicar el procedimiento realizado para que el robot pueda desplazarse de su posición actual a su posición deseada, se hará uso del ejemplo mostrado en la Figura 2. 7 Se denota a X_{Goal} y Y_{Goal} como los puntos que conformar la coordenada a la que desea llegar y γ_{goal} como el ángulo u orientación deseado que debe tomar el robot para llegar a dichas coordenadas. De manera similar, se denota X_{Real} y Y_{Real} como las coordenadas reales del robot y γ_{Real} como el ángulo u orientación real o actual de robot. Como se puede observar, el robot Nao se encuentra en la posición (0,0) u origen y el punto de llegada deseado se encuentra en (1,1), es decir, en este caso $X_{Real} = 0$, $Y_{Real} = 0$, $\gamma_{Real} = 0$ y $X_{Goal} = 1$, $Y_{Goal} = 1$, $\gamma_{goal} = 45\, grados$. Cabe recalcar que se denota a α como la diferencia entre γ_{Real} y γ_{goal}, es decir, el error de ángulo que tiene el robot. El camino más fácil y óptimo para que llegue a su destino se lo representa mediante P, el cual es la hipotenusa del triángulo que se forma entre la diferencia de los puntos de llegada con los puntos de salida.

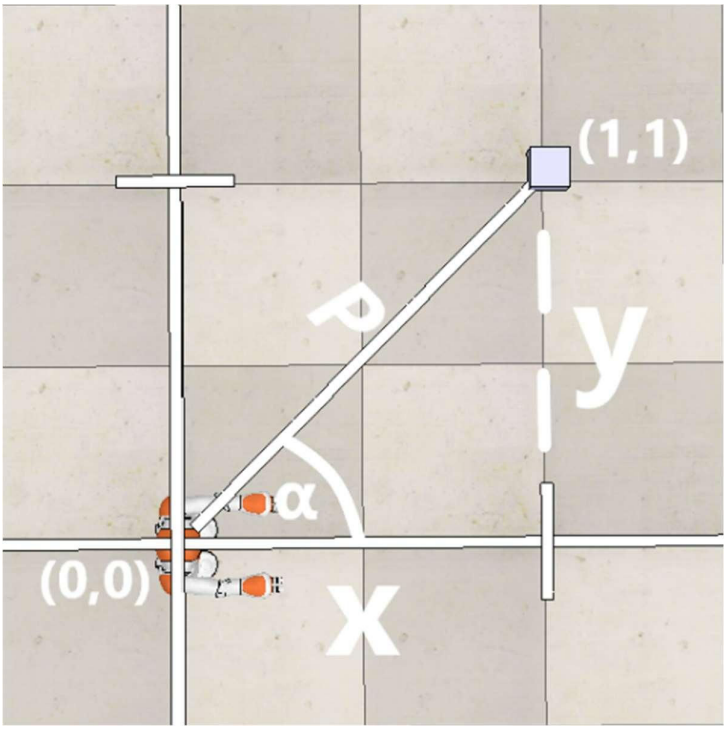


Figura 2. 7 Ejemplo de orientación del robot Nao

Como se puede observar en la Figura 2. 7, para obtener el ángulo de giro necesario del robot, se debe realizar un cambio de coordenadas rectangulares a polares. Es decir, se debe aplicar una relación trigonométrica entre X y Y, en donde ambos representan los catetos del triángulo formado por la diferencia que existe entre la posición deseada y la posición actual del robot ($X = X_{Goal}$-X_{Real} , Y=$Y_{Goal} - Y_{Real}$). La relación trigonométrica utilizada es el arco tangente de los catetos (ecuación 2.23), y su signo y posición X y Y dependerán del cuadrante en donde se encuentre el robot.

$$\gamma_{goal} = arctang\left(\frac{Cateto1}{Cateto2}\right) \tag{2.1}$$

Hay que tomar en consideración que, para el caso de segundo y tercer cuadrante, al ángulo obtenido por el arco tangente se le deberá sumar 90 grados para que así, todos los ángulos de giro necesario sean obtenidos a partir de eje X positivo. En la Tabla 2. 2 se puede observar un resumen de lo explicado anteriormente.

Tabla 2. 2 Relación trigonométrica de catetos y ángulo de giro deseado dependiendo ubicación del robot en eje de coordenadas

Cuadrante	Relación trigonométrica	Ángulo de giro deseado (γ_{goal})
I	$arctang\left(\frac{Y}{X}\right)$	$\gamma_{goal} = arctang\left(\frac{Y}{X}\right)$
II	$arctang\left(\frac{-X}{Y}\right)$	$\gamma_{goal} = 90 + arctang\left(\frac{-X}{Y}\right)$
III	$arctang\left(\frac{-X}{-Y}\right)$	$\gamma_{goal} = -90 - arctang\left(\frac{-X}{-Y}\right)$
IV	$arctang\left(\frac{-Y}{X}\right)$	$\gamma_{goal} = -arctang\left(\frac{-Y}{X}\right)$

En el caso del ejemplo, dado que los puntos de llegada son (1,1) y los de origen son (0,0), el ángulo de giro necesario será de 45 grados. Una vez obtenido este ángulo, se procede, cada cierto tiempo determinado ($t'_m = 0.9\ seg$: tiempo de muestreo de la señal de referencia de ángulo determinada de manera heurística), a obtener el ángulo de giro deseado actual del robot, con el fin de monitorear el error de ángulo existente. Es decir, se monitoreará la diferencia entre el ángulo de giro deseado con ángulo actual del robot hasta que esta tienda a cero mediante la implementación del controlador de velocidad angular. En la Figura 2. 8 se puede observar un esquema de control de movimiento del robot humanoide.

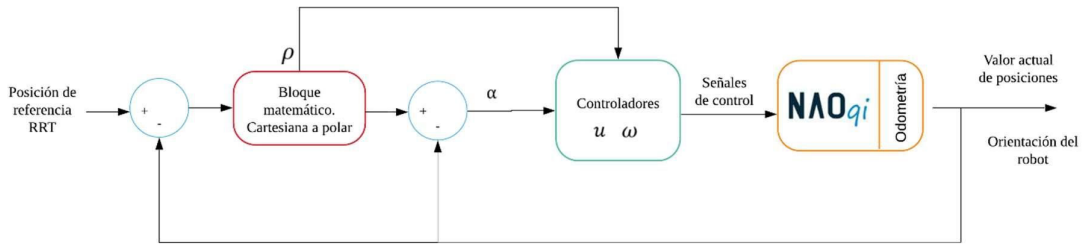


Figura 2. 8 Esquema de control del movimiento del robot Nao

Una vez que el robot haya alcanzado el ángulo deseado, comenzará a caminar rectilíneamente con una velocidad lineal determinada para poder así llegar al punto objetivo. Para monitorear que esto suceda, se hace uso de la ecuación 2.2, ecuación de la distancia. A medida que el robot avance y se acerque al objetivo, el valor de la distancia (ρ) se irá reduciendo hasta tender a cero. Una vez que esto suceda, se dará la orden al robot de que se detenga y espere a la siguiente instrucción.

$$distancia = \sqrt{(X_{Goal} - X_{Real})^2 + (Y_{Goal} - Y_{Real})^2} \quad (2.2)$$

De la ecuación 2.2 se observa que la distancia se la obtiene a partir del valor de X y Y, los cuales, como ya se ha mencionado anteriormente, corresponden a los catetos del triángulo

formado entre las coordenadas reales del robot y las coordenadas deseas. Es decir, la distancia corresponde a la hipotenusa (ρ) de dicho triangulo.

Es importante recalcar que, durante todo el movimiento lineal del robot, aun se estará monitoreando el error de ángulo del robot cada tiempo de muestreo = 0.03 segundos, para mantener el error cercano a cero.

2.2. DESARROLLO DE LOS ALGORITMOS DE CONTROL

El objetivo principal de los robots móviles es poder desplazarse desde un punto A hacia un punto B siguiendo un camino determinado. Para esto, es necesario monitorear su orientación y posición para que, mediante señales de control de velocidad lineal y velocidad angular, se pueda garantizar que el robot siga el camino propuesto. Para el caso de este trabajo de titulación, se ha implementado dos tipos de controladores: un controlador tipo PID y un controlador basado en Lyapunov los cuales se explicarán a continuación:

2.2.1 CONTROL PID

2.2.1.1 Implementación del algoritmo de control

Para poder implementar un controlador tipo PID al robot humanoide NAO, tal y como se muestra en la Figura 2. 8, es necesario primeramente discretizar la estructura de control que se encuentra en el dominio de Laplace (2.3).

$$\frac{U(s)}{E(s)} = k_p + k_i \frac{1}{s} + k_d s \tag{2.3}$$

Cada acción que presenta el controlador tipo PID se discretiza de manera diferente, de acuerdo con su naturaleza. Para la acción integral se utiliza el método de integración trapezoidal o Tustin (ecuación 2.4) que siempre toma un promedio entre el valor anterior y el valor actual.

$$s = \frac{2}{T_o}\left(\frac{z-1}{z+1}\right) \tag{2.4}$$

Para la acción derivativa se utiliza el método de integración hacia atrás (ecuación 2.5), que toma directamente el valor anterior y el valor actual de la variable a ser contralada.

$$s = \frac{1}{T_o}\left(\frac{z-1}{z}\right) \tag{2.5}$$

Reemplazando (2.3) y (2.4) en (2.5) y resolviendo se obtiene el PID en diferencias (ecuación 2.6 – ecuación 2.8).

$$\frac{U(z)}{E(z)} = k_p + k_i\left(\frac{T_o}{2}\frac{z+1}{z-1}\right) + k_d\left(\frac{1}{T_o}\frac{z-1}{z}\right) \tag{2.6}$$

$$\left(z^2 - z\right)U(z) = E(z)\left(z^2\left(k_p + \frac{k_i T_o}{2} + \frac{k_d}{T_o}\right) + z\left(-k_p + \frac{k_i T_o}{2} - \frac{2k_d}{T_o}\right) + \left(\frac{k_d}{T_o}\right)\right) \tag{2.7}$$

$$\left(1 - z^{-1}\right)U(z) = E(z)\left(\left(k_p + \frac{k_i T_o}{2} + \frac{k_d}{T_o}\right) + z^{-1}\left(-k_p + \frac{k_i T_o}{2} - \frac{2k_d}{T_o}\right) + z^{-2}\left(\frac{k_d}{T_o}\right)\right) \tag{2.8}$$

$$U(z) - z^{-1}U(z)$$

$$= E(z)\left(k_p + \frac{k_i T_o}{2} + \frac{k_d}{T_o}\right) + E(z)z^{-1}\left(-k_p + \frac{k_i T_o}{2} - \frac{2k_d}{T_o}\right) + E(z)z^{-2}\left(\frac{k_d}{T_o}\right) \tag{2.9}$$

Aplicando la transformada Z inversa (ecuación 2.10):

$$z^{-n}F(z) = F(k-n) \tag{2.10}$$

Se obtiene finalmente la ecuación en diferencias para la implementación del PID (ecuación 2.11):

$$U(k) - U(k-1) = E(k)\left(k_p + \frac{k_i T_o}{2} + \frac{k_d}{T_o}\right) + E(k-1)\left(-k_p + \frac{k_i T_o}{2} - \frac{2k_d}{T_o}\right)$$

$$+ E(k-2)\left(\frac{k_d}{T_o}\right) \tag{2.11}$$

Con la ecuación 2.11 se realiza la calibración de constantes, luego de tomar en cuenta las consideraciones del modelo detalladas en la siguiente sección.

2.2.1.2 Consideraciones del modelo

Como ya se ha explicado en la sección 1, Figura 1.14, tanto la dinámica como la cinemática del robot son representadas como caja negra, es decir, se da por sentado el control de movimiento a bajo nivel y se aproxima el modelo cinemático a un sistema de primer orden. Para que NAO pueda desplazarse a través de su espacio de trabajo es necesario monitorear su posición y orientación cada tiempo de muestro t_m, y controlarlos mediante su velocidad lineal y angular. Es decir, serán ambas velocidades las acciones de control del sistema que permitirán al robot llegar a un punto determinado.

En el caso de la velocidad lineal, es suficiente implementar únicamente un controlador proporcional (P) más una señal BIAS que permita a la velocidad lineal alcanzar un valor diferente de cero, es decir en un estado de equilibrio cuando el error sea igual a cero y siga a la señal de velocidad lineal de referencia.

En el caso de la velocidad angular, se ha decido implementar un controlador PD (proporcional - derivativo) ya que dicha acción de control es suficiente para corregir el error de ángulo del robot en régimen permanente en el menor tiempo posible sin presentar sobre picos.

Como ya se ha explicado en la sección 1.3.3, es necesario el uso de una acción integral cuando el control actual no es suficiente para corregir el error en régimen permanente, el cual no es el caso para el controlador PD implementando. Por dicha razón no se ha escogido una acción integral para el controlador.

Con el objetivo de realizar una calibración adecuada de constantes en el controlador tipo PID, es necesario primeramente conocer la respuesta del robot sin controlador en lazo cerrado. 2.2.1.2.1

2.2.1.2.1 Prueba a lazo cerrado del ángulo con velocidad máxima permitida por el robot y sin controlador.

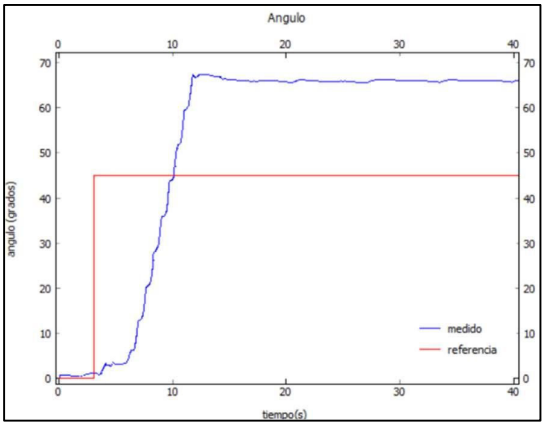


Figura 2. 9 Respuesta de ángulo a velocidad máxima

El objetivo de esta prueba es conocer cuál es la respuesta del ángulo del robot como y consecuentemente, cuáles son los mejores tiempos de establecimiento en esta configuración. De esta manera, al momento de calibrar el controlador tipo PID, se considera que no se pueden disminuir estos tiempos de establecimiento, dado que son intrínsecos del robot.

Para conocer la respuesta de ángulo del robot Nao, se ha dado como referencia una señal paso de 45 grados en t=3 segundos. Como se observa en la Figura 2. 9. En un inicio, el robot se encuentra orientado cero grados en el eje de las x, y ante este cambio de

referencia, gira con una velocidad angular máxima permitida de 8.5 grados/s en sentido antihorario (los valores máximos y mínimos del robot se determinaron de manera heurística dado que los recomendados por el fabricante no reflejaban lo obtenido en el simulador). Se ha dado la instrucción de que el robot se detenga (w=0) cuando el ángulo del robot sea igual o mayor al ángulo de referencia (45 grados). Sin embargo, como se puede evidenciar en la figura, existe un error de posición de aproximadamente 23 grados (ángulo final aproximado = 68 grados), el cual debe ser corregido mediante el controlador propuesto. De igual manera, se puede evidenciar que el robot logra estabilizarse en un tiempo aproximado de 8 segundos a la velocidad angular establecida, ya que inicia a girar en el t=3 segundos, y termina en t=11 segundos.

Al ingresar la velocidad angular máxima permitida por el robot, se asegura que el tiempo de establecimiento es el mínimo que se puede alcanzar. Es decir, el tiempo de estabilización mínima que tendrá el robot ante un cambio de referencia de 45 grados estará rondando los 5 o 6 segundos. Cabe recalcar que este tiempo de estabilización dependerá del cambio de referencia que se le aplique al robot, siendo este mayor, al aplicar un cambio de referencia mayor y viceversa.

2.2.1.2.2Prueba a lazo cerrado del ángulo con velocidades angular mínima permitida por el robot y sin controlador

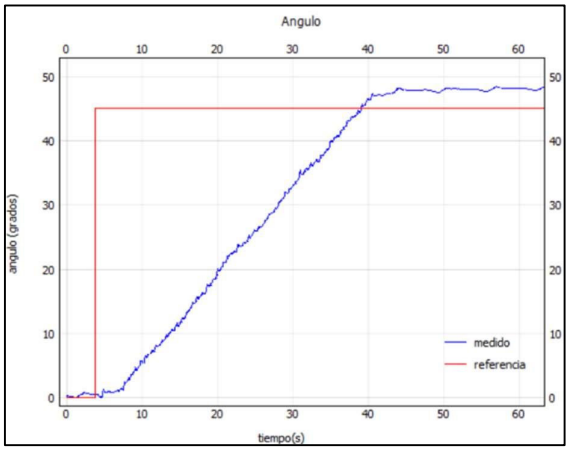


Figura 2. 10 Respuesta de ángulo a velocidad mínima

A diferencia de la prueba realizada anteriormente, en esta prueba, se configura una velocidad angular mínima. Como se observa en la Figura 2. 10 el robot se demora un tiempo considerablemente mayor al caso anterior, debido a que su velocidad angular es menor (1 grados/s). Sin embargo, el error que presenta es menor dado que la inercia del robot es mucho menor a la del primer caso. El objetivo de controlador igualmente es la de reducir este error a cero.

2.2.1.2.3 Prueba a lazo cerrado del ángulo con velocidad angular media y sin controlador.

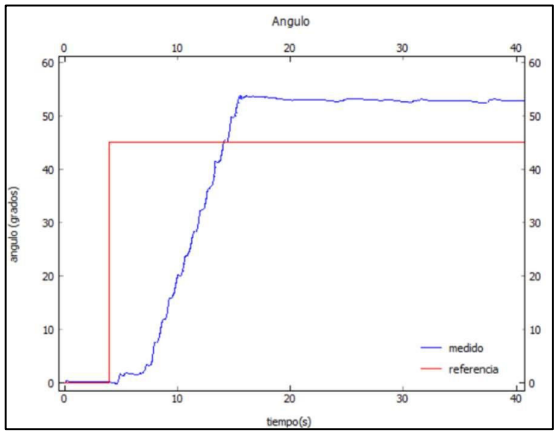


Figura 2. 11 Respuesta de ángulo a velocidad media

Con el objetivo de evidenciar que el robot responde antes diferentes cambios de referencia, se da una referencia de velocidad media de 4.7 grados/s. Se puede observar en la Figura 2. 11 que, al igual que en los anteriores casos, el robot sobrepasa la señal de referencia debido a la inercia de giro del robot. En este caso, se obtuvo un error de posición de 9 grados.

El objetivo del controlador es de que, sin importar el valor de la señal de referencia, el robot pueda alcanzar el ángulo de giro requerido en el menor tiempo posible.

2.2.1.2.4 Prueba a lazo cerrado con velocidad lineal media y sin controlador.

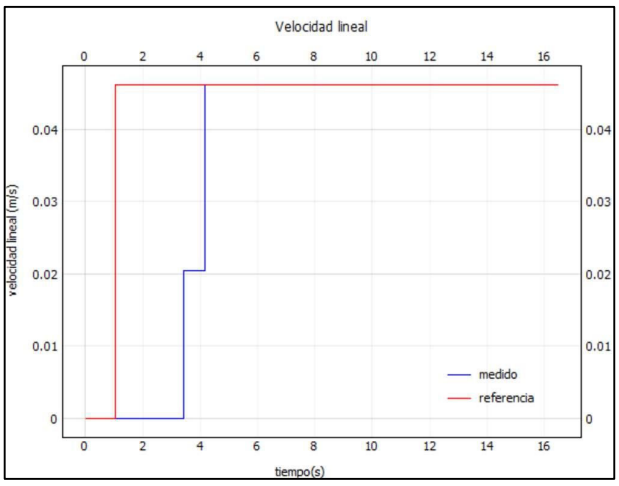


Figura 2. 12 Respuesta de velocidad lineal media

Se puede observar en la Figura 2. 12 que, en el caso de la velocidad lineal, no se tiene un error de posición en estado estable. Sin embargo, en estado transitorio, existe un atraso de señal con respecto a la señal de referencia, el cual es 2.3 [s] aproximadamente. El motivo del comportamiento de la señal medida se explica a continuación.

Como se sabe, para poder diseñar el controlador de la velocidad lineal, es necesario obtener el error de velocidad, es decir, la diferencia entre la velocidad de referencia y a la velocidad actual del robot. Esta última velocidad se la consigue con la función movProxy.getRobotVelocity()[0] proveniente del módulo ALMotion, es decir, es una función propia del robot que se encuentra dentro de la librería NAOqi. Por tal motivo, no es posible eliminar el atraso de la señal de velocidad medida del robot con respecto a la señal de referencia. Cabe mencionar que de igual manera, se tiene un atraso en la velocidad angular ya que se usa la misma función movProxy.getRobotVelocity()[2].

2.2.1.3 Calibración de los controladores

Como se puede evidenciar en la Figura 2. 9, Figura 2. 10 y Figura 2. 11 la respuesta de salida del ángulo del robot en el sistema retroalimentado no cumple con las condiciones suficientes para que pueda seguir a las señales de referencia correspondientes. Esto hace necesario el uso y calibración de un controlador de velocidad angular cuyo objetivo no sea solamente lograr que se alcance la señal de referencia de ángulo, sino también, que lo haga en el menor tiempo posible, evitando al máximo sobre picos de señal. Una vez obtenida la orientación que debe tomar el robot para llegar a su punto objetivo, el robot debe avanzar con una velocidad lineal determinada y así avanzar rectilíneamente a su objetivo, y como se puede observar en la Figura 2. 12, dicha velocidad no presenta un error de posición en estado estacionario. Sin embargo, en caso de que exista alguna perturbación externa (ej. Aumento de algún accesorio al robot que aumente el peso de este) sin controlador alguno no se podrá asegurar que el robot llegue a su punto deseado.

Para calibrar los controladores se hace uso de un método que pueda caracterizar el tiempo de respuesta del sistema; Para esto, se sirve de la ecuación (2.12)

$$\Phi = \int_0^\infty F[e(t), t]dt \tag{2.12}$$

En donde F representa una función del error y de tiempo mientras que Φ es el criterio de comportamiento, el cual será cero únicamente cuando el error es cero durante todo el tiempo. Sin embargo, esto último es prácticamente imposible de alcanzar debido a que el robot suele empezar en condiciones diferentes a las referencias; Además, también existen perturbaciones que afectan y pueden sacar al robot de su punto de trabajo [17].

Como ya se mencionó anteriormente, el objetivo de los controladores es lograr disminuir el error existente entre la señal de referencia y la señal de salida en el menor tiempo posible ante cualquier cambio de referencia o perturbación de la señal. Para lograr esto, se procede a calibrar cada uno de los controladores basándose en el índice de desempeño IAE.

2.1.1.3.1 Integral del Error Absoluto – IAE

Como ya se mencionó en el primer capítulo, este índice de rendimiento es sensible a pequeños errores, y poco sensible a grandes. Además, es fácil de implementar y brinda amortiguamiento y respuestas aceptables a la salida del lazo retroalimentado de control.

Para obtener las mejores constantes del controlador PD de velocidad angular, se ha desarrollado una serie de pruebas experimentales. Las mejores constantes corresponderán a aquellas que provoquen el menor índice de error absoluto. El procedimiento realizado es el siguiente:

- Para el controlador de velocidad lineal solamente se ha considerado parte proporcional, pues es suficiente para que la señal de salida alcance la referencia sin errores o sobre picos.
- Para el controlador de velocidad angular, se ha considerado un controlador PD (proporcional - derivativo) para que la señal de salida alcance a la señal de referencia.

Tabla 2. 3 Valores iniciales constantes del controlador PID

Valores iniciales		
$k_p = 0{,}2$	$k_d = 0$	**IAE=**0,311

Tabla 2. 4 Ensayos para calibración de velocidad angular

Ensayos								
k_p	0,2	0,3	0,35	0,45	0,4	0,5	0,6	0,7
IAE	0,0698	0,0166	0,0294	0,0124	0,01152	0,01167	0,0299	0,01078

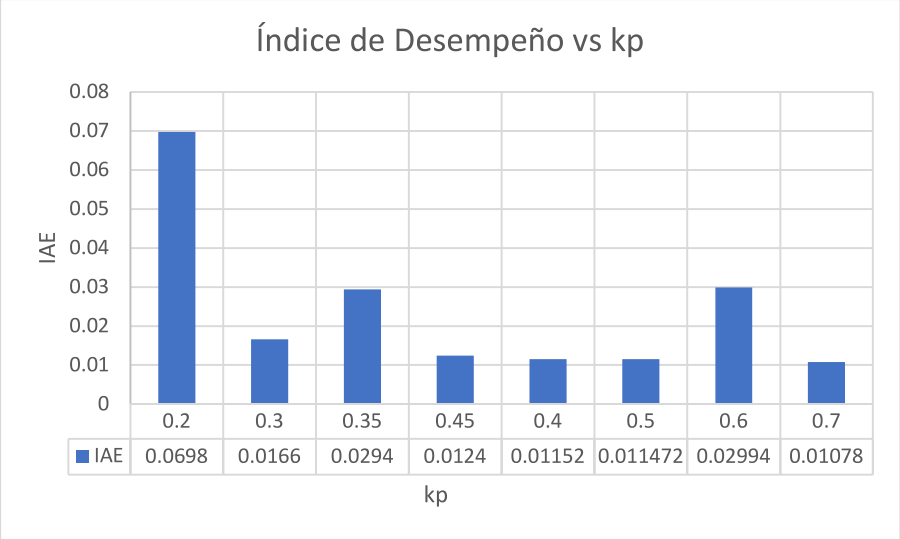


Figura 2. 13 Índice de desempeño vs constante proporcional kp.

Tabla 2. 5 Valores iniciales constante del controlador PID una vez determinado la constante proporcional

Valores iniciales		
$k_p = 0{,}4$	$k_d = 0{,}01$	**IAE=**0,0084

Tabla 2. 6 Ensayos para calibración de velocidad angular una vez determinado la constante proporcional

Ensayos						
k_d	0,01	0,05	0,1	0,15	0,18	0,2
IAE	0,0084	0,00732	0,0122	0,0187	0,0174	0,0162

Tabla 2. 7 Valores finales de contantes del controlador PID

Valores finales		
$k_p = 0{,}4$	$k_d = 0{,}05$	IAE=0,007

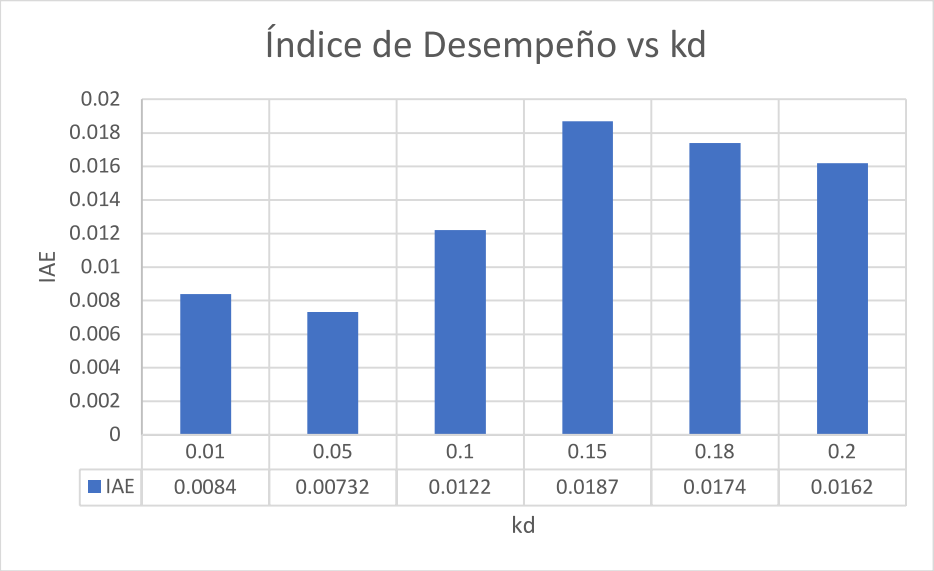


Figura 2. 14 Índice de desempeño vs constante derivativa kd

En el caso del controlador proporcional de velocidad lineal, solamente se ha establecido un valor mínimo en la constante proporcional (Kp=0.01) para no afectar de manera significativa la respuesta en lazo cerrado, ya que, sin perturbaciones externas, el robot no presenta errores en estado estacionario.

2.2.1.4 Representación completa de control - Modelo

Como se menciona en la anterior sección, 2.1.1.3, para realizar la calibración de las constantes de cada uno de los controladores se utiliza solamente la parte proporcional para la señal de control de velocidad lineal, ecuación 2.13, mientras que, para la señal de control de velocidad angular, se emplea un control proporcional – derivativo, ecuación 2.14.

$$U(k) = U(k-1) + E(k)\left(K_p\right) + E(k-1)\left(-K_p\right) \tag{2.13}$$

$$U(k) = U(k-1) + E(k)\left(K_p + \frac{K_d}{T_o}\right) + E(k-1)\left(-K_p - \frac{2K_d}{T_o}\right) + E(k-2)\left(\frac{K_d}{T_o}\right) \tag{2.14}$$

Cabe recalcar que la señal de control de velocidad angular permitirá al robot corregir el error de ángulo que se tenga, es decir, le permitirá al robot orientarse a su próximo objetivo, mientras que la señal de control de velocidad lineal permitirá al robot desplazarse rectilíneamente para poder así corregir el error en posición existente, consiguiendo de esta manera que, punto a punto, el robot siga el camino generado por el algoritmo RRT.

En el diagrama de bloques de la Figura 2. **15** se puede observar el esquema completo de control del ángulo y posición del robot Nao.

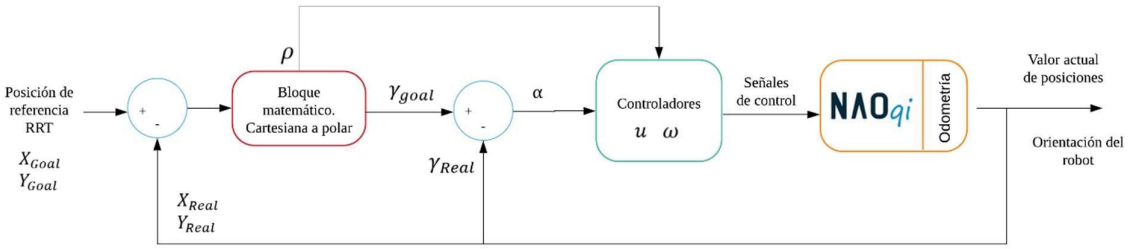


Figura 2. 15 Esquema de control de posición y orientación del robot humanoide Nao.

Dentro del bloque matemático, se realiza el cambio de coordenadas rectangulares a coordenadas polares, así como también la obtención de la distancia al punto de llegada deseado (ρ). Esto se realiza cada $t'_m = 0.9\ seg$.

A continuación, en la Figura 2. 16, se muestra la acción de control PD utilizada para corregir el error de orientación del robot.

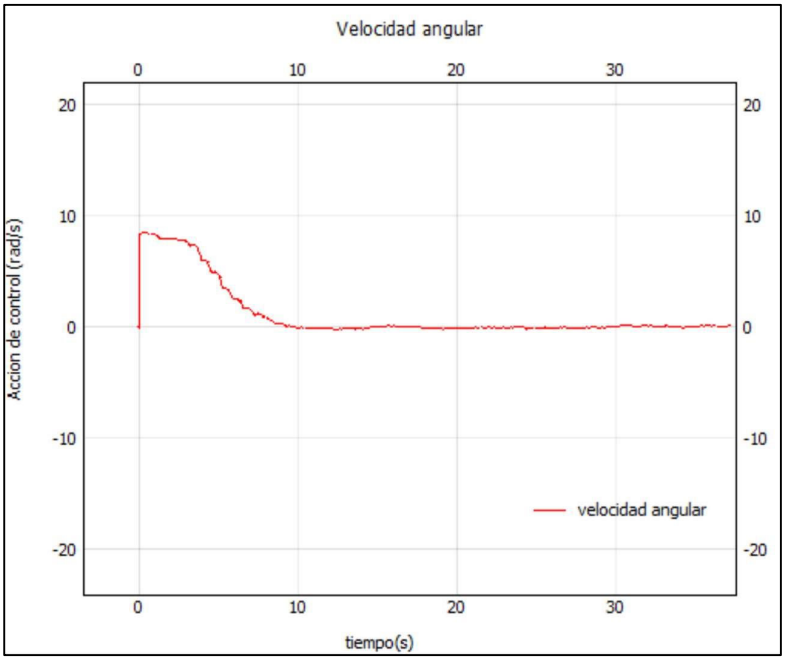


Figura 2. 16 Acción de control - velocidad angular

Se puede observar claramente en la Figura 2. 17 que el ángulo real (señal azul) del robot sigue al ángulo de referencia (señal roja) eliminando así el error de posición que existía en el sistema sin controlador. (Figura 2. 9 - Figura 2. 11).

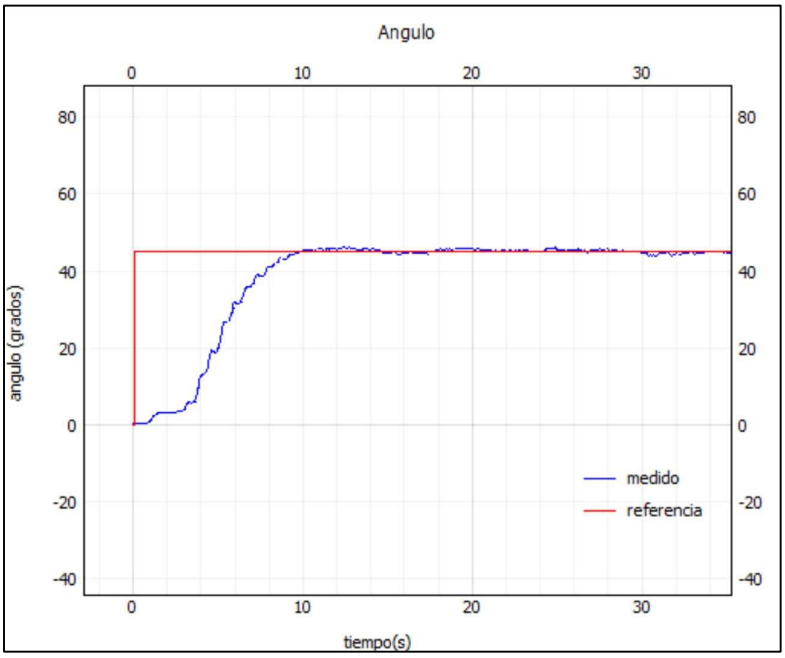


Figura 2. 17 Respuesta de ángulo del robot
al aplicar el controlador PD

2.2.2 CONTROL BASADO EN LYAPUNOV

Como se puede observar en la Figura 2. 18, se considera al robot humanoide NAO posicionado a una distancia $\boldsymbol{\rho}$ de su punto de destino $\{Rp\}$ y se asume como vector de variables de estado a $[\boldsymbol{\rho}\ \ \boldsymbol{\alpha}]^T$, el cual existe para cualquier $\boldsymbol{\rho} > 0$. El sistema de ecuaciones (ecuación 2.15 y ecuación 2.16) que describe el movimiento del robot humanoide es:

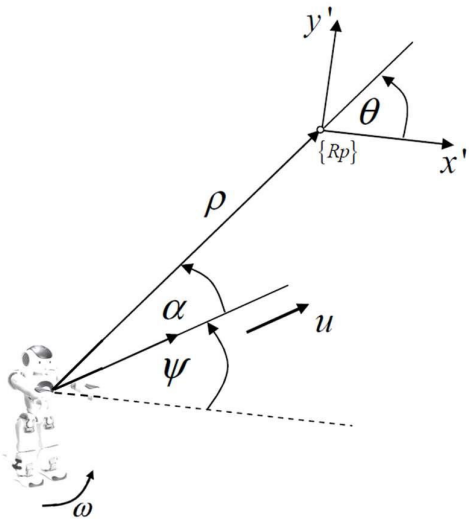


Figura 2. 18 Posición y orientación del robot humanoide respecto a punto destino

$$\dot{\rho} = -ucos(\alpha) \tag{2.15}$$

$$\dot{\alpha} = -\omega + u\frac{sin(\alpha)}{\rho} \tag{2.16}$$

Considere la función candidata de Lyapunov (ecuación 2.17):

$$V(\rho, \alpha) = V_1 + V_2 = \frac{1}{2}\rho^2 + \frac{1}{2}\alpha^2 \tag{2.17}$$

La derivada temporal de la ecuación 2.17 está dada por:

$$\dot{V} = \rho\dot{\rho} + \alpha\dot{\alpha} \tag{2.18}$$

$$\dot{V} = \underbrace{\rho(-ucos(\alpha))}_{\dot{V}_1} + \underbrace{\alpha(-\omega + u\frac{\sin(\alpha)}{\rho})}_{\dot{V}_2} \tag{2.19}$$

En la ecuación 2.19 se puede observar que el primer término $\dot{V}_1$ puede ser no positivo únicamente si la velocidad lineal u tiene la forma de la ecuación 2.20:

$$u = K_u \tanh(\rho)\cos(\alpha) \qquad con\ K_u > 0 \tag{2.20}$$

Donde el coeficiente $K_u = |u_{max}|$

Dada la ecuación 2.18, se reescribe el término $\dot{V}_2$ de la ecuación 2.19 dando como resultado la ecuación 2.21:

$$\dot{V}_2 = \alpha(-\omega + K_u \frac{\tanh(\rho)}{\rho}\sin(\alpha)\cos(\alpha)) \tag{2.21}$$

La ecuación 2.21 puede no ser positiva únicamente si la velocidad angular ω toma la forma:

$$\omega = K_\omega \alpha + K_u \frac{\tanh(\rho)}{\rho}\sin(\alpha)\cos(\alpha) \tag{2.22}$$

Donde $|\omega_{max}|$=$K_\omega \pi + K_u 0.5$, con lo que $\dot{V}_2$ toma la forma (ecuación 2.23):

$$\dot{V}_2 = \alpha\left(-K_\omega \alpha - K_u \frac{\tanh(\rho)}{\rho}\sin(\alpha)\cos(\alpha) + K_u \frac{\tanh(\rho)}{\rho}\sin(\alpha)\cos(\alpha)\right) = -K_\omega \alpha^2 \tag{2.23}$$

Finalmente se obtiene la expresión para la derivada temporal de la función Lyapunov original V (ecuación 2.24):

$$\dot{V} = -K_u \rho \tanh(\rho)\, cos^2(\alpha) - K_\omega \alpha^2 \tag{2.24}$$

De esta manera, se ha cuadrado las ecuaciones para que el sistema sea asintóticamente estable y se pueda garantizar que existe estabilidad. Una vez obtenido u y ω, las acciones de control, ya se puede calibrar para que el robot Nao pueda orientarse y llegar a su posición deseada.

2.2.2.1 Calibración del controlador

Tabla 2. 8 Valores iniciales constantes del controlador basado en Lyapunov

Valores iniciales		
$k_u = 0{,}0512$	$k_w = 0{,}2$	**IAE**=0,00978

Tabla 2. 9 Ensayos para calibración de constantes de Lyapunov

Ensayos						
kw	0,2	0,4	0,6	0,7	0,76	0.8
IAE	0,0098	0,00853	0,021	0,017	0,0112	0,013

Tabla 2. 10 Valores finales de contantes del controlador basado en Lyapunov

Valores finales		
$k_u = 0{,}0512$	$k_w = 0{,}4$	IAE=0,00853

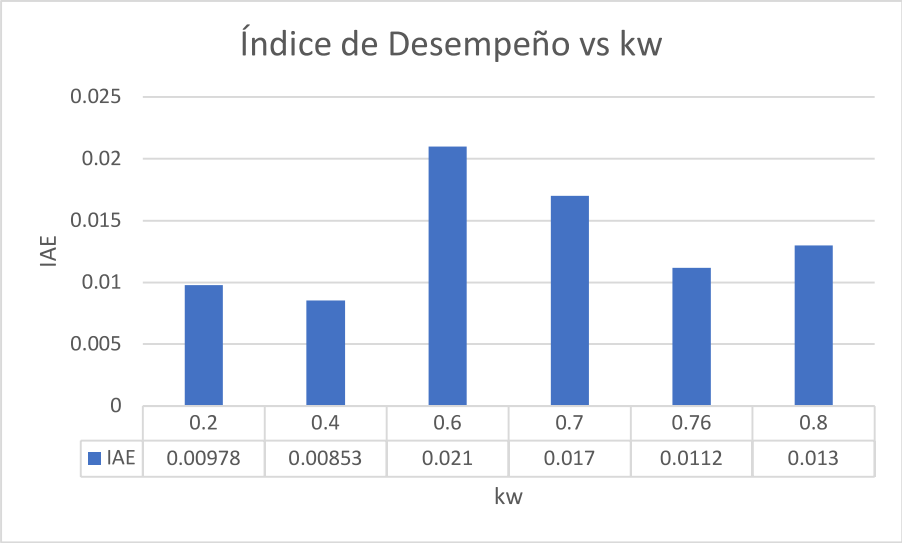


Figura 2. 19 Índice de desempeño vs kw

2.3. IMPLEMENTACIÓN DE LOS ALGORITMOS DE CONTROL

Con el objetivo de visualizar el funcionamiento de ambos controladores, PID y basado en Lyapunov, se propone dar caminos predefinidos al robot y comprobar que efectivamente, si sigue las referencias propuestas.

Es importante tomar en cuenta que, cada vez que el robot Nao comience a caminar, se presenta oscilaciones periódicas en el camino generado. Esto se debe a que la posición del robot solamente es un punto ubicado en su pecho y que, al caminar, este se moverá al ritmo con que se vayan ejecutando los pasos. Esta es una característica inherente que presenta el robot Nao, y como consecuencia, las acciones del controlador también heredarán dicha particularidad.

2.3.1 CONTROLADOR TIPO PID

A continuación, se muestra la respuesta de ángulo del robot Nao y las acciones de control velocidad lineal y angular del robot Nao ante diferentes caminos preestablecidos. En cada una de las pruebas, se estableció una velocidad lineal de referencia del 75%.

2.3.1.1 Línea

Como se puede observar en la Figura 2. 20, el primer camino preestablecido es una línea que va desde el punto (0,0) hasta el punto (1,1). En un inicio, el robot Nao se encontraba con un ángulo de -135 grados con respecto al eje de referencia de CoppeliaSim, para lo

cual tuvo que girar 180 grados en sentido antihorario para poder alinearse al punto (1,1). Se ha decido colocar al robot Nao en esta orientación dado que 180 corresponde al mayor ángulo que tendrá que girar el robot ante cualquier prueba.

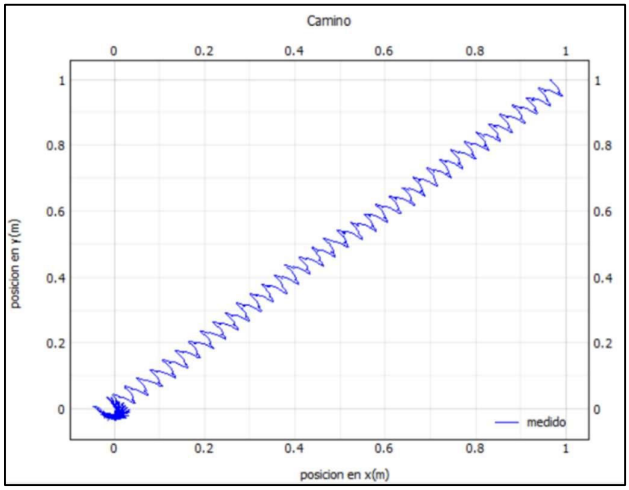


Figura 2. 20 Camino recorrido por el robot humanoide Nao con controlador PID

En la Figura 2. 21 se puede observar cómo en un inicio, mientras el robot Nao realiza el giro de 180 grados, la acción de velocidad lineal se mantiene en cero. Una vez que se haya alineado el robot hacia el punto (1,1), empezará a movilizarse rectilíneamente hasta llegar a su objetivo.

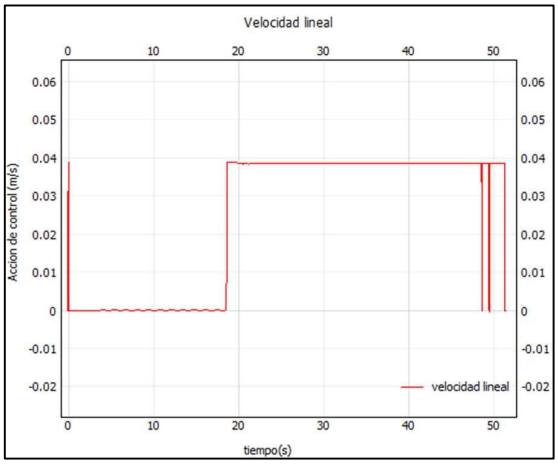


Figura 2. 21 Acción de control - velocidad lineal

En la Figura 2. 22, se puede observar que, al iniciar el giro de 180 grados, la acción de control de velocidad angular acciona una velocidad de 8.5 grados/s (velocidad angular máxima), ya que el error entre el ángulo actual del robot y el ángulo deseado es el máximo. Cuando el error haya disminuido significativamente, la acción de control empezará a disminuir progresivamente hasta que tienda a cero, es decir, hasta que ya no tenga que realizar ningún giro. Sin embargo, como ya se explicó anteriormente, cada 0.9 segundos

se actualiza la señal de referencia para asegurarse que la señal de velocidad angular se mantenga en cero. Es por tal motivo que se puede observar ciertas oscilaciones en estado estable. Además, cabe recalcar que ha medida que el robot llegue a su objetivo, tanto X como Y tenderán a cero, haciendo que el cálculo del ángulo de giro necesario comience a variar significativamente provocando así las oscilaciones que pueden observar al final de la señal de control de velocidad angular. Este fenómeno estará presente cada vez que exista un cambio de referencia de posición o punto objetivo al que se desea que el robot llegue. En el siguiente camino predefinido se explicará de mejor manera dicho fenómeno.

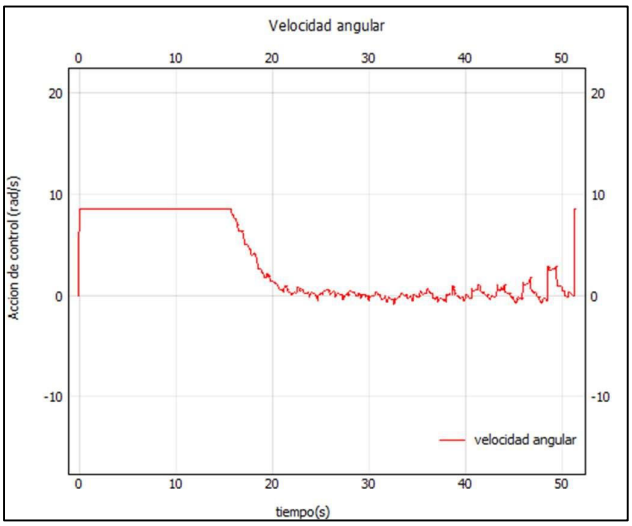


Figura 2. 22 Acción de control - velocidad a angular

En la Figura 2. 23, se puede observar que el ángulo del robot es capaz de alcanzar la referencia sin presentar ningún tipo de sobre pico. Al igual que en el caso anterior, la razón por la que se observa pequeñas oscilaciones en la referencia es debido a que cada 0.9 segundos se actualiza el valor. Con este resultado se demuestra que el robot Nao es capaz de girar un ángulo máximo de 180 grados y alcanzar la referencia propuesta sin presentar ningún tipo de oscilación y en el menor tiempo posible.

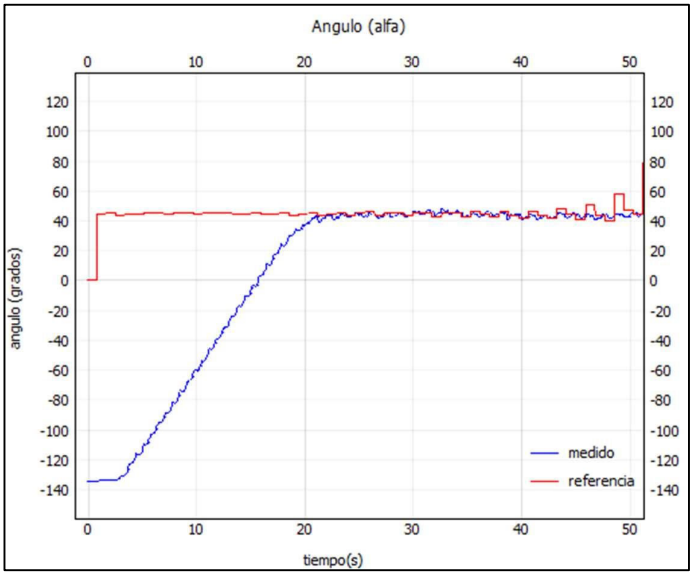


Figura 2. 23 Respuesta de ángulo del robot humanoide Nao con respecto al marco de referencia de CoppeliaSim

2.3.1.2 Rombo

En el caso anterior, se pudo observar la respuesta de ángulo del robot ante un solo cambio de referencia. Sin embargo, es importante conocer cuál es respuesta del robot ante diferentes cambios de referencia ya que en el algoritmo RRT, se verá expuesto a varios de ellos. Es por tal motivo que se ha escogido un rombo como camino predeterminado para poder testear estos cambios de referencia. Como se puede observar en la Figura 2. 24, se han establecido cada uno de los 4 puntos que forman el rombo de manera simétrica, siendo el primero el punto (0.5, -0.5), el segundo (1, 0), el tercero en (0.5, 0.5) y el cuarto en el origen (0,0). El robot inicia en el origen y a partir de ahí, empieza a trazar el camino propuesto.

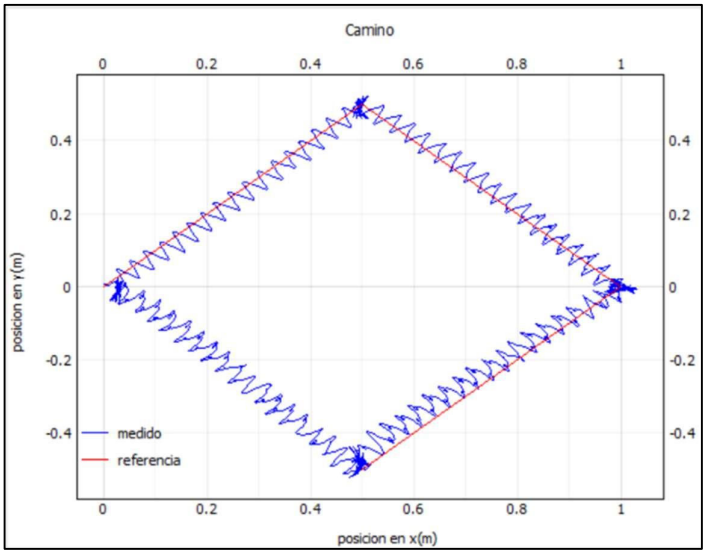


Figura 2. 24 Camino recorrido por el robot humanoide Nao con controlador PID

Como se puede observar en la Figura 2. 25, el robot humanoide, al estar expuesta a cambios de referencia de ángulos rectos, tendrá una acción de control nula en velocidad lineal. Esto, con el objetivo de que el robot no se salga del camino propuesto. Dado que existen 4 ángulos rectos en el rombo, cuatros veces el robot tendrá una acción de control de velocidad lineal nula, es decir, en cada esquina el robot solamente girará sobre su propio eje. Una vez que el robot este alinee a su siguiente punto la acción de control de la velocidad lineal permanece constante, diferente de cero.

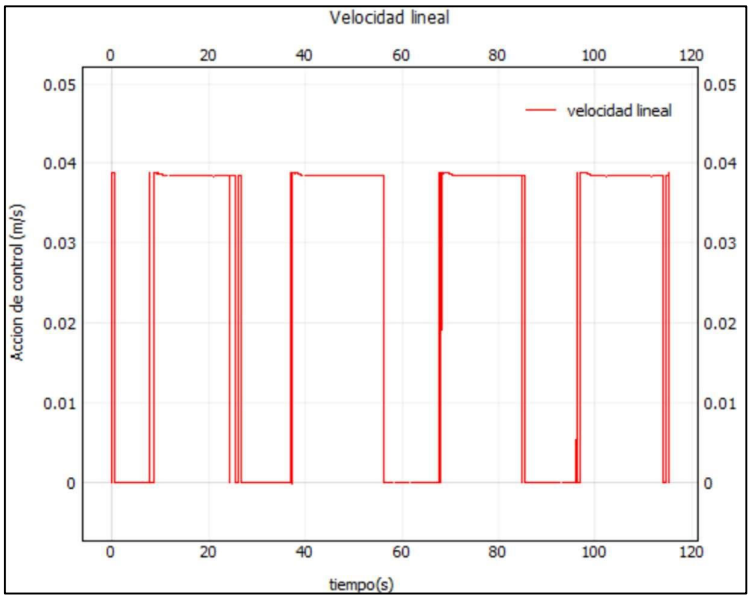


Figura 2. 25 Acción de control - velocidad lineal.

En la Figura 2. 26, se puede evidenciar los cambios presentes en la acción de control de velocidad angular cada vez que el robot debe girar en cada una de las esquinas de rombo.

Como ya se ha explicado en el camino anteriormente expuesto (línea), la acción de control de velocidad angular y el ángulo del robot presentan una particularidad o fenómeno cuando este llega a cada uno de sus puntos objetivo. En este caso en particular, se puede notar como, justo antes de realizar el giro en cualquier de las esquinas, se generan oscilaciones con picos más elevados que los normales. Esto se debe a que tanto X (X_{Goal}-X_{Real}) como $Y(Y_{Goal} - Y_{Real})$ son valores muy pequeños cuando el robot se encuentra a una distancia muy pequeña de su objetivo, y al realizar el cálculo del ángulo del robot mediante el arco tangente, los ángulos empiezan a variar disruptivamente. Para entender este fenómeno de mejor manera, se explicará mediante el siguiente ejemplo. Segundos antes de llegar a la primera esquina del rombo, el robot se encuentra a una distancia menor a 0.1 de su objetivo, es decir que, tanto X como Y tienen valores menores a 0.1 y con cada paso que el robot Nao ejecuta, estos valores cambian disruptivamente. Es decir que, en definitiva, al realizar el cambio de coordenadas de cartesianas a polares, se obtiene por consecuencia cambios disruptivos en los ángulos, haciendo que la acción de control de velocidad angular presente estos picos justo antes de llegar de que el robot llegue a un punto objetivo determinado.

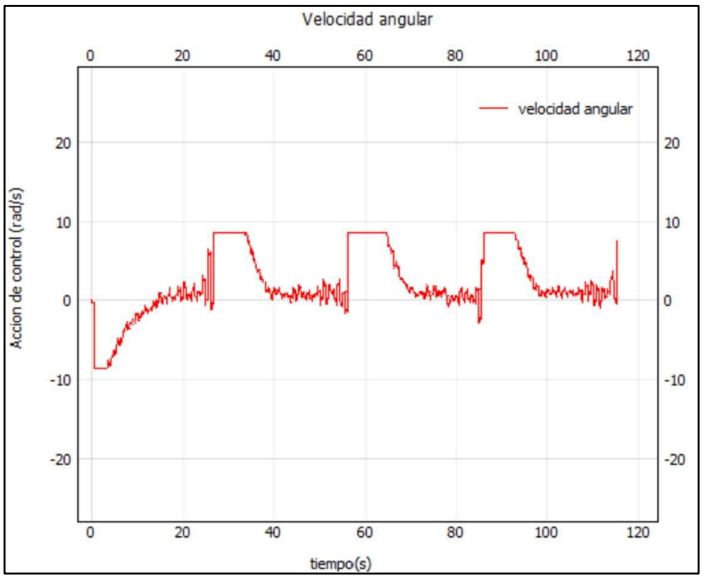


Figura 2. 26 Velocidad angular del robot humanoide Nao con controlador PID

En la Figura 2. 27, se puede observar como la señal de ángulo medido sigue a la referencia sin presentar ningún sobre pico ni error de posición, lo cual evidencia la funcionalidad de los controladores. Al igual que en la figura anterior, aquí también se puede ver reflejado los cambios disruptivos de ángulo segundos antes de que el robot llegue a su objetivo. Vale la pena mencionar que, si el valor de referencia se actualizara cada tiempo de muestreo $t_m = 3mseg$, los picos de ángulo serían mucho mayores a los que se tiene con un tiempo de 0.9

segundos. Es por tal motivo que se ha escogido este tiempo para recalcular cada una de las referencias.

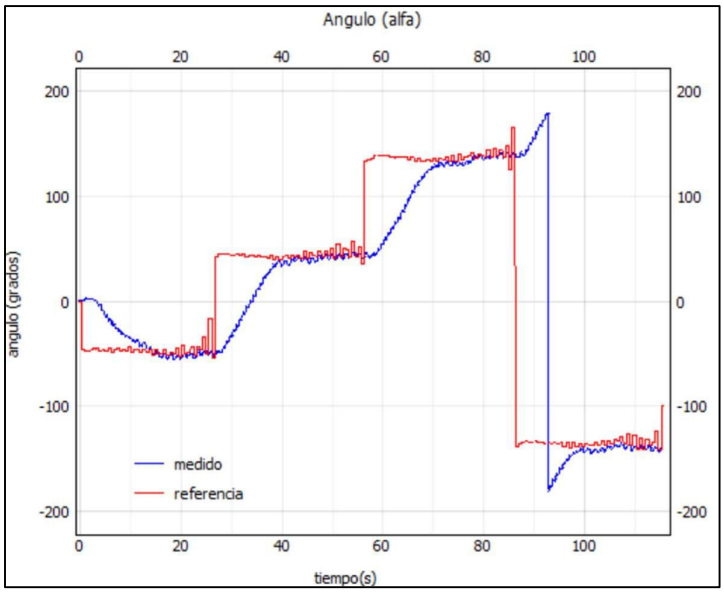


Figura 2. 27 Respuesta de ángulo del robot humanoide Nao con respecto al marco de referencia de CoppeliaSim

2.3.1.3 Circulo

Con el objetivo de observar la respuesta de ángulo ante un cambio de referencia rampa, se ha escogido como tercer camino preestablecido al círculo. Como se observa en la Figura 2. 28, el robot Nao es capaz de seguir sin ningún problema cada uno de los puntos que conforman el circulo. A continuación, se realizará un análisis de la respuesta de ángulo, así como también de sus acciones de control.

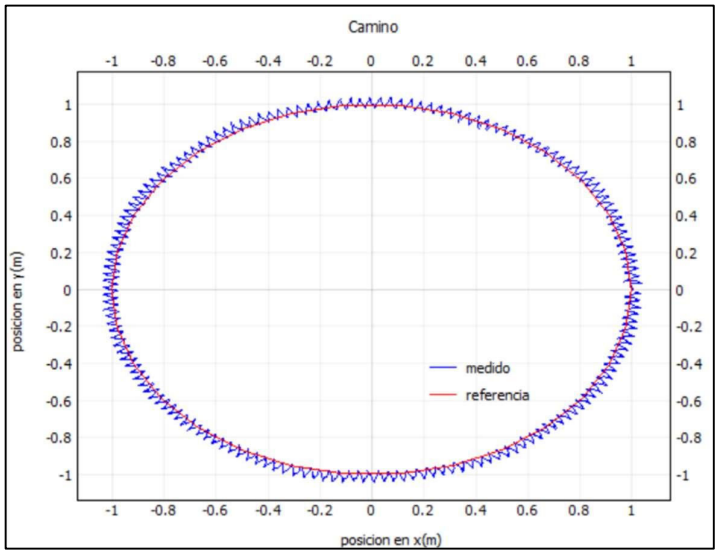


Figura 2. 28 Camino recorrido por el robot humanoide Nao con Controlador PID

A diferencia de los caminos anterior, ahora no se ha aplicado la restricción de dar una acción de control de velocidad lineal nula al estar expuesto a ángulos diferentes. Esto, con el objetivo de no interrumpir el avance del robot al realizar el circulo.

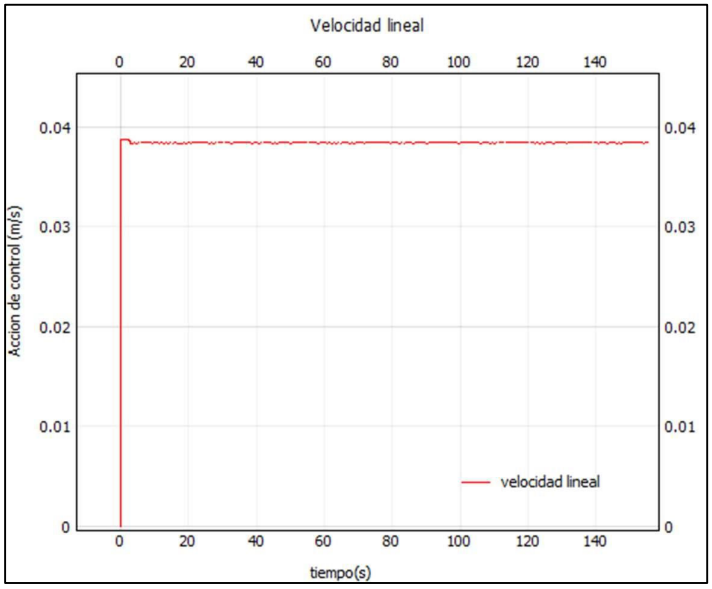


Figura 2. 29 Velocidad lineal del robot humanoide Nao con Controlador PID

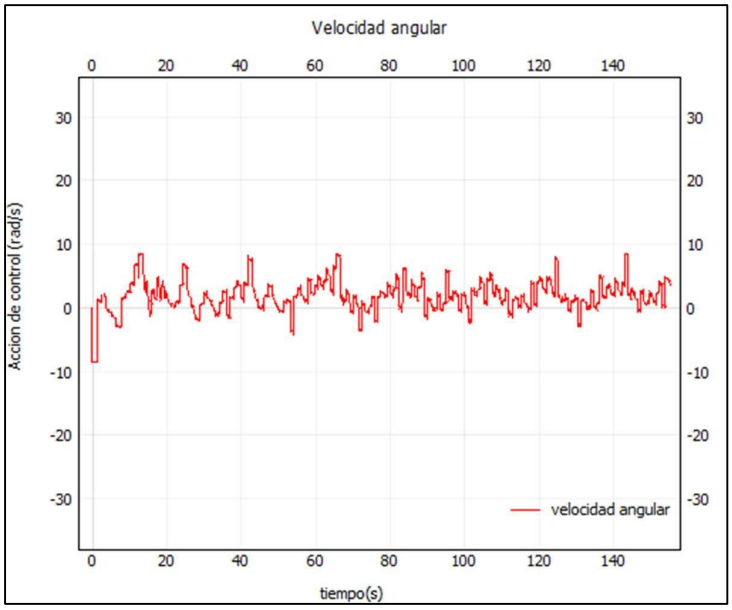


Figura 2. 30 Acción de control – velocidad angular.

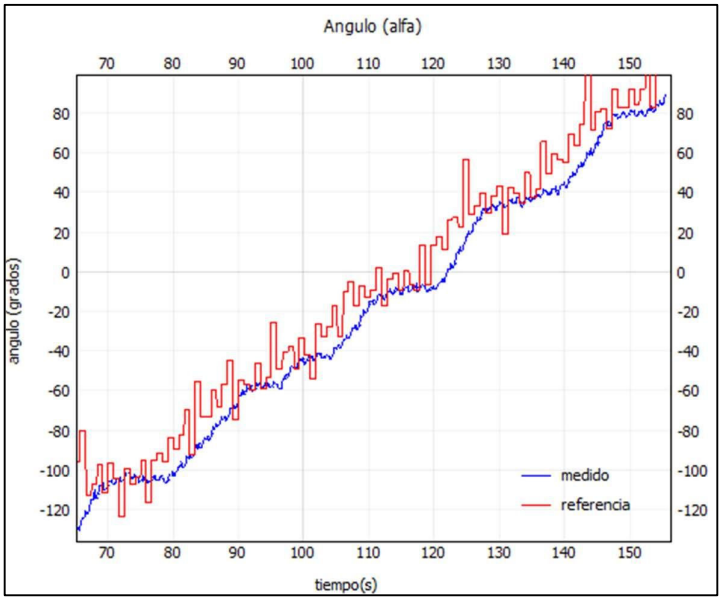


Figura 2. 31 Respuesta de ángulo del robot humanoide Nao con respecto al marco de referencia de CoppeliaSim

En la Figura 2. 31, solamente se muestra una fracción del ángulo de referencia y el ángulo medido dado que toma un tiempo considerable realizar todo el camino. Como se puede notar, el cambio de referencia no se trata de una señal paso, sino de una rampa. Esto debido a que, para realizar la circunferencia, no se necesita cambios bruscos en el ángulo, sino más bien, un cambio progresivo y uniforme. De igual manera, en la Figura 2. 30, se puede observar que la acción de control de la velocidad angular cambia progresivamente, pero de manera menos brusca que en los casos anteriores.

Se puede observar igualmente que la señal medida sigue a la referencia, pero con cierto error de posición. Esto es debido a que, dado que la señal de referencia es una rampa, la señal de respuesta no tiene el tiempo suficiente para alcanzar la referencia en cada intervalo de tiempo. Sin embargo, como se observa en la Figura 2. 28, esto no perjudica en su totalidad al desempeño de robot al realizar el camino.

Como se pudo observar en los tres caminos preestablecidos, tanto el control de velocidad lineal como de velocidad angular permite al robot realizar un seguimiento de camino ya se ante un solo cambio de referencia, diferentes cambios de referencia o incluso cambios de referencia paso.

A continuación, se muestra el resultado del controlador basado en Lyapunov sometido a los mismos caminos preestablecido en PID.

2.3.2 CONTROLADOR BASADO EN LYAPUNOV

A continuación, se muestra la respuesta de ángulo del robot Nao y las acciones de control velocidad lineal y angular del robot Nao ante diferentes caminos preestablecidos (línea, rombo y circulo similar al caso anterior con el controlador tipo PID). En el caso del controlador basado en Lyapunov, la velocidad lineal dependerá de la distancia que exista entre la ubicación actual del robot y el punto objetivo al que se quiera llegar.

2.3.2.1 Línea

Como se puede observar en la Figura 2. 32, tal como se realizó con el controlador tipo PID en la sección 2.3.1.1, el primer camino preestablecido es una línea que va desde el punto (0,0) hasta el punto (1,1). En un inicio, el robot gira de -135 grados a 45 grados orientándose así a su punto de llegada. Se puede evidenciar que entre más se acerca a su punto de llegada, la gráfica trazada se acentúa más. Esto es debido a que, a medida que el robot se acerca a su objetivo, el robot disminuirá su velocidad lineal hasta un mínimo preestablecido de 1.5 mm/s y consecuentemente se tomará un mayor número de datos a plotear.

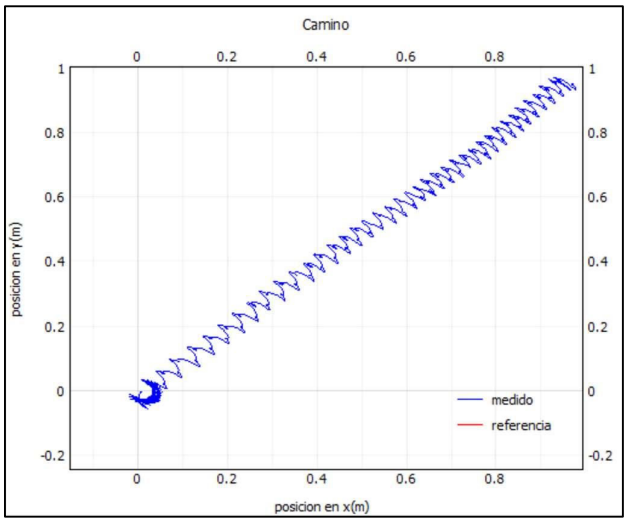


Figura 2. 32 Camino recorrido por el robot humanoide Nao con controlador basado en Lyapunov

En la Figura 2. 33, se pude observar que, mientras el robot se encuentra realizando el giro de 180 grados, el robot tiene una acción de control de velocidad lineal nula. Una vez realizado todo el giro, la velocidad lineal va variando directamente proporcional a la distancia que se encuentra a su objetivo, es decir, mientras más lejos se encuentre el robot de su objetivo, mayor será la velocidad lineal con la que avance, mientras que, mientras más cercano se encuentre, menor será su velocidad. Cabe recalcar que se ha hecho uso

de saturadores en la velocidad lineal mínima del robot, que corresponde a 20mm/s, para que este pueda llegar a su objetivo en el menor tiempo posible.

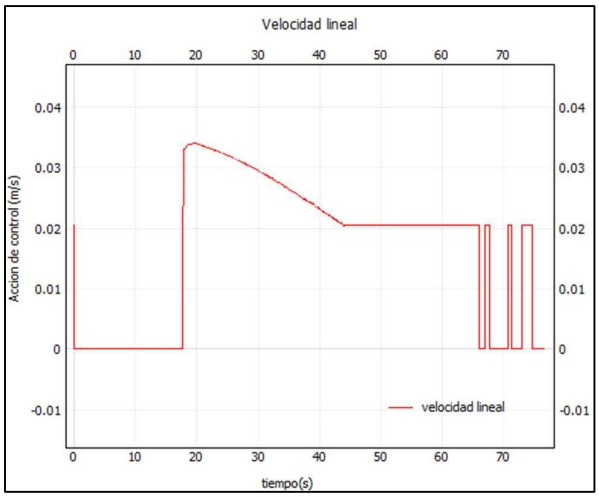


Figura 2. 33 Acción de control – velocidad lineal.

De manera similar que en el controlador tipo PID, la acción de control de la velocidad angular del robot es representativa en el periodo transitorio ya que, durante este tiempo, este se encuentra realizando el giro. Sin embargo, una vez que se encuentre orientado hacia su objetivo, el controlador de Lyapunov trata de mantener este valor cercano a cero, tal y como se puede observar en la Figura 2. 34. El fenómeno de oscilación en la señal del controlador y en el ángulo del robot expuesta en el caso anterior también se encuentra presente en el controlador basado en Lyapunov.

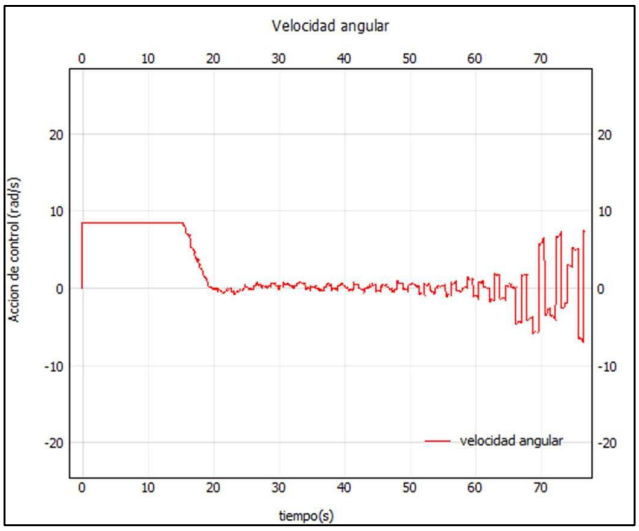


Figura 2. 34 Acción de control – velocidad angular.

Finalmente, en la Figura 2. 35, se observa como la referencia de ángulo del robot alcanza la referencia preestablecida de 45 grados. Se puede notar que existe un pequeño pico y

oscilación del ángulo robot al alcanzar la referencia. Sin embargo, estos valores no afectan significativamente al seguimiento del camino del robot hacia su objetivo.

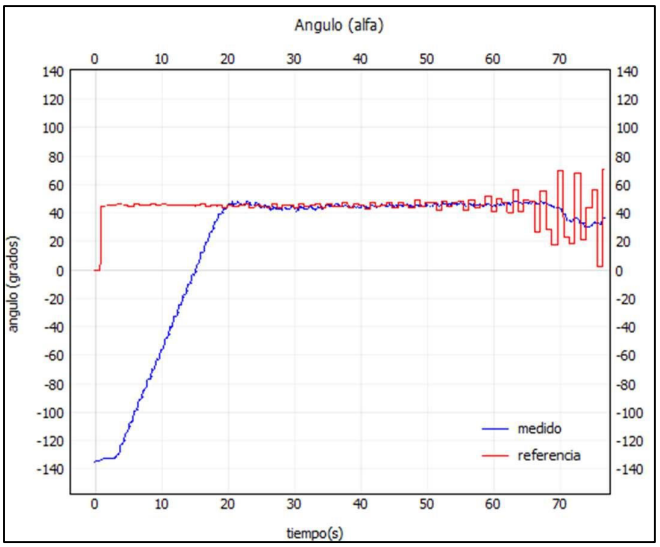


Figura 2. 35 Respuesta de ángulo del robot humanoide Nao con respecto al marco de referencia de CoppeliaSim

2.3.2.2 Rombo

Al igual que en el caso del controlador PID, el robot se ve expuesto a cuatro diferentes cambios de referencia con respecto al punto objetivo de llegada. En un inicio, el robot se encuentra en las coordenadas (0,0) y debe recorrer hacia los puntos (0.5, -0,5), (1,0), (0.5,0.5) y volver al punto de partida (0,0).

Se puede notar en la Figura 2. 36 y Figura 2. 37 que, dado que el punto final u objetivo del robot es el origen, la acción de control de velocidad lineal del robot es baja, pero a medida que se aleje de ese punto, irá aumentando. Cuando se trabaje con el algoritmo RRT, solamente se tomará en cuenta el punto final al que se desea llegar para que, de esa manera, durante gran parte del recorrido del robot, la velocidad lineal sea la mayor posible. Al igual que en el caso anterior, cuando el robot se ve expuesto a giros mayores a 15 grados, primeramente, se detiene linealmente, realiza el giro y luego sigue avanzando.

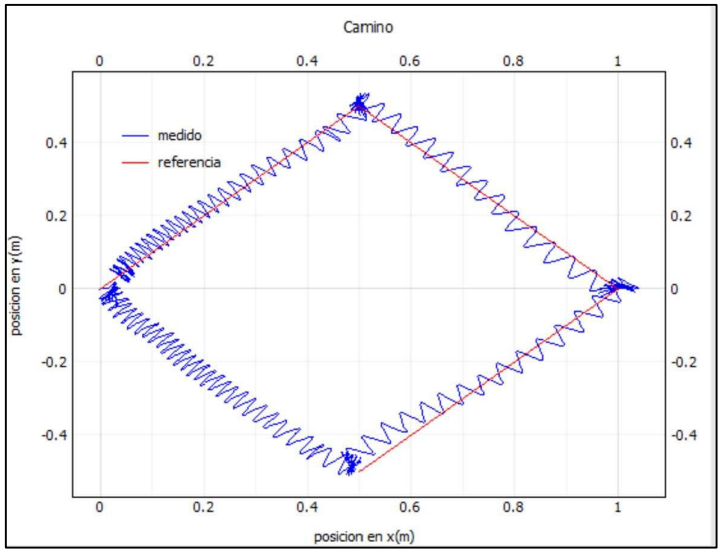


Figura 2. 36 Camino recorrido por el robot humanoide Nao con controlador basado en Lyapunov

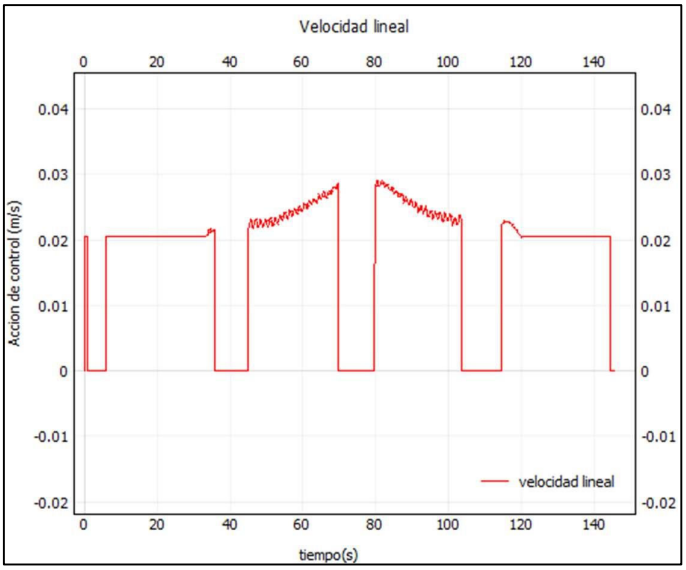


Figura 2. 37 Acción de control – velocidad lineal.

Al igual que en casos anteriores, el robot se ve expuesto a un cambio repentino de ángulo, por lo que la acción de control de velocidad angular será la máxima hasta que el robot logre

orientarse hacia el punto objetivo. Este compartimiento se puede evidenciar en la Figura 2. 38 a continuación.

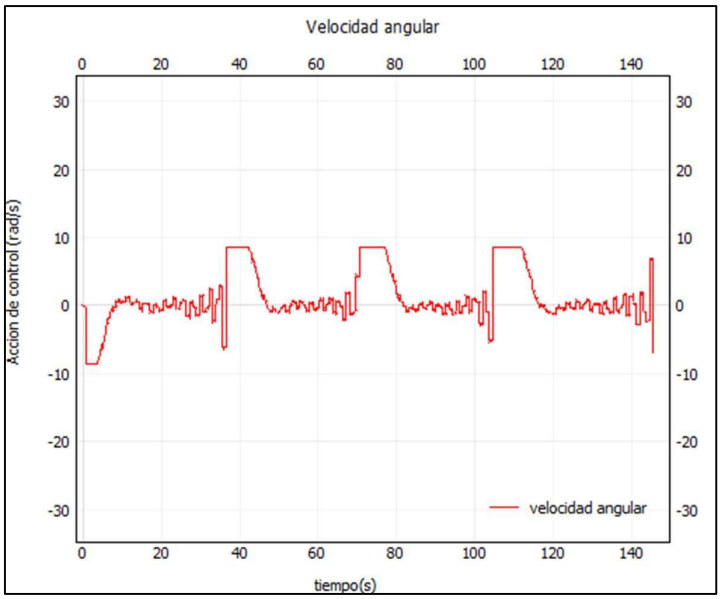


Figura 2. 38 Acción de control – velocidad angular.

Finalmente, en el análisis de ángulo, se puede observar en la Figura 2. 39 que el robot logra obtener el ángulo requerido de referencia sin ningún sobre pico o error de posición. Estas características permiten que el robot, al momento de realizar cualquier giro significativo, lo haga de la mejor manera.

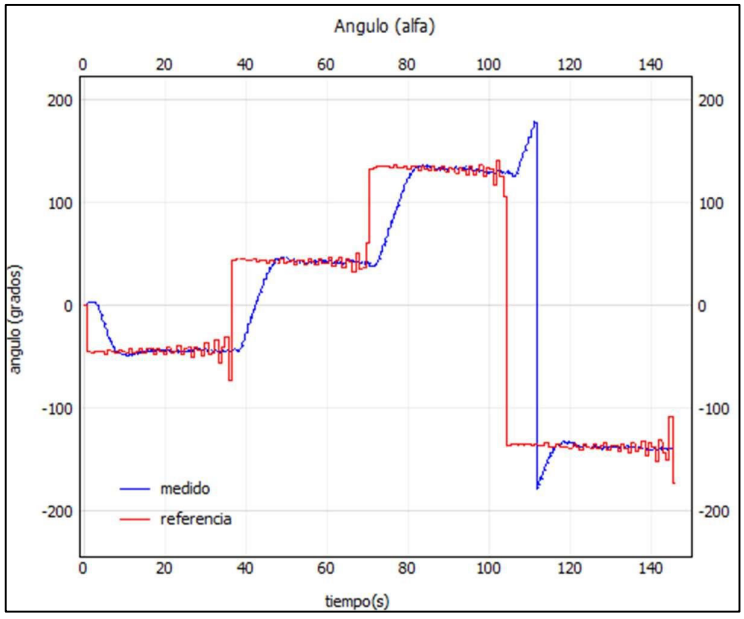


Figura 2. 39 Respuesta de ángulo del robot humanoide Nao con respecto al marco de referencia de CoppeliaSim.

2.3.2.3 Circulo

Como último camino predefinido se ha escogido el circulo debido a que, de esta manera, se expone constantemente a cambios de referencia tanto en ángulo como en velocidad lineal. En la Figura 2. 40 se puede observar el resultado final luego de que el robot ha recorrido todo el camino predefinido. En un inicio, en su punto de partida en (1,0), se puede notar más entonación de los datos debido que su acción de control de velocidad lineal es pequeña. A medida que se aleje el robot de ese punto, su velocidad lineal irá incrementando, hasta llegar al punto más lejano, que corresponde al punto (0,-1). A partir de ahí, el robot nuevamente irá disminuyendo su velocidad lineal hasta llegar nuevamente al punto de partida en (0,1). Dicho comportamiento también se puede evidenciar en la Figura 2. 41, que corresponde a la acción de control de la velocidad lineal del robot. En este caso, se ha configurado una velocidad mínima de 22mm/s.

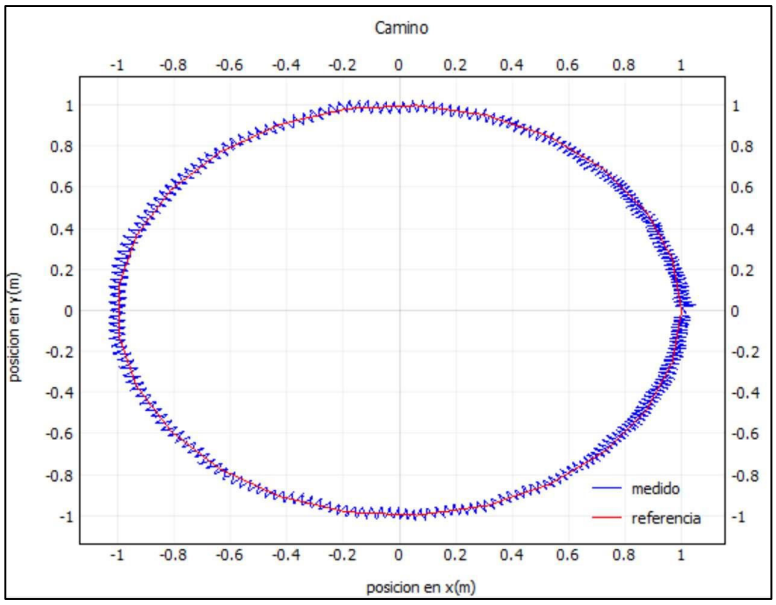


Figura 2. 40 Camino recorrido por el robot humanoide Nao con controlador basado en Lyapunov

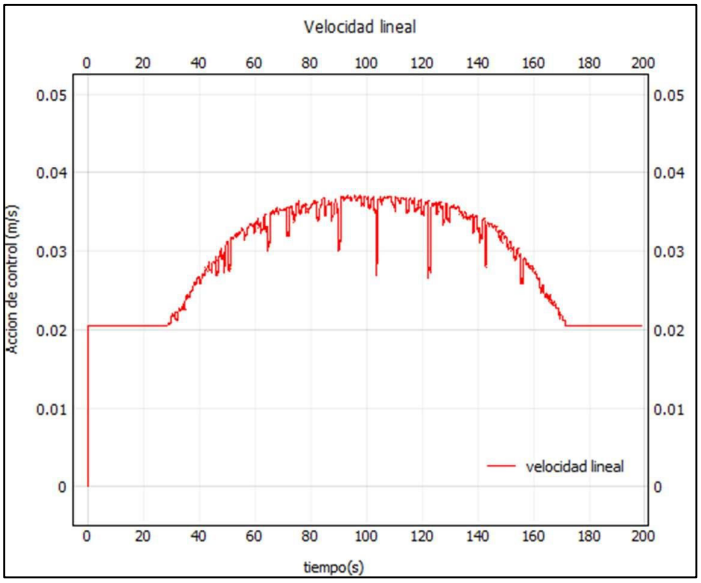


Figura 2. 41 Acción de control – velocidad lineal.

En la Figura 2. 42 se puede observar que la acción de control de la velocidad angular cambia progresivamente, pero de manera menos brusca que en los casos anteriores.

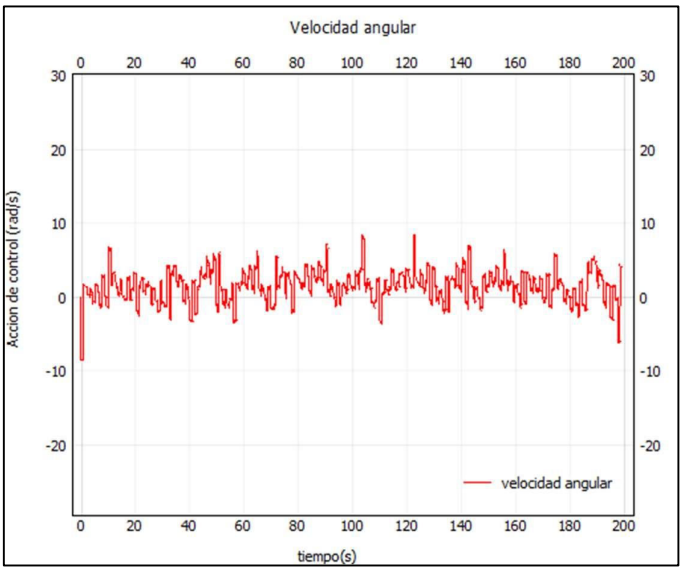


Figura 2. 42 Acción de control – velocidad angular.

Finalmente, en la Figura 2. 43, se puede notar como se ha expuesto al robot Nao a cambios de referencia tipo rampa, y que, sin ningún problema, logra seguir para completar así el circulo predefinido. Al igual que en el caso anterior, solo se presenta una breve sección de la gráfica ya que toma un tiempo considerable para que el robot realice el circulo.

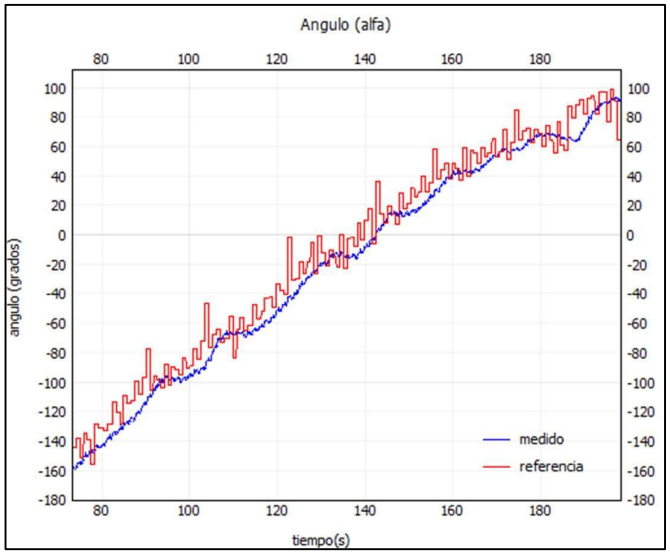


Figura 2. 43 Respuesta de ángulo del robot humanoide Nao con respecto al marco de referencia de CoppeliaSim.

Como se pudo observar en imágenes anteriores, el robot Nao es capaz de seguir diferentes caminos propuestos tanto con el controlador PID como con el controlador basado en Lyapunov, luego de haber calibrado ambos correctamente. En la sección 3, Resultados y Discusión, se podrá evidenciar como ambos controladores funcionan en el aula de clase al darse un camino generado por el algoritmo RRT.

2.4. LÓGICA DE PROGRAMACIÓN

Con el objetivo de poder visualizar el flujo de funcionamiento del proyecto desarrollado en Python, se muestra a continuación el diagrama de flujo de las funciones más relevantes del programa.

En la Figura 2. 44, se muestra la estructura principal del programa, el cual se basa en hilos, los cuales serán explicados en la Figura 2. 45, Figura 2. 46, Figura 2. 47, para su funcionamiento. Primeramente, se llama a la interfaz gráfica, en donde el usuario podrá decidir entre las diferentes opciones de controladores, condiciones iniciales, etc. Al dar inicio al programa mediante esta interfaz gráfica, se ejecuta tres hilos principales, necesarios para el funcionamiento completos del proyecto.

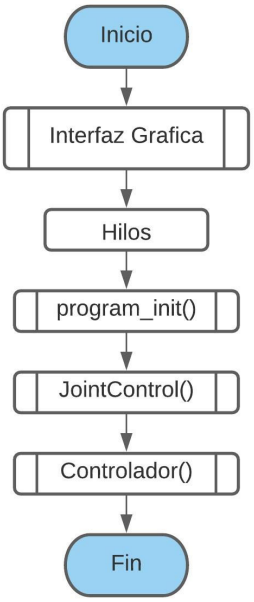


Figura 2. 44 Diagrama de flujo de estructura principal del programa

Como se puede apreciar en la Figura 2. 45, la función *program_init* se encarga primeramente de establecer conexión con el programa CoppeliaSim mediante funciones específicas en Python. Luego, mediante la dirección 127.0.0.1 y puerto 9559 se establece conexión con el módulo NAOqi, el cual fue ejecutado en el mismo momento que se abrió la interfaz, Finalmente, mediante vectores, se define cada una de las articulaciones del robot NAO, se los guarda en una matriz llamada *Body,* y se retorna los datos relevantes de la función.

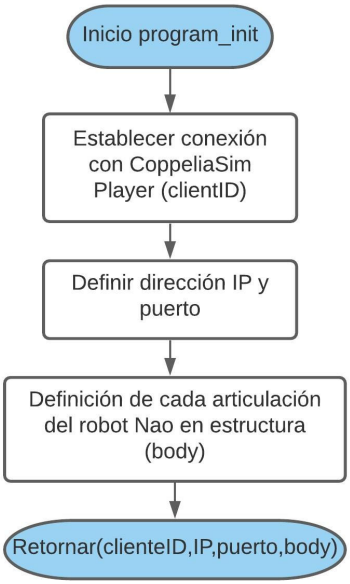


Figura 2. 45 Diagrama de flujo de program_init

En la Figura 2. 46, se muestra el diagrama de flujo de la función *JointControl,* el cual se encarga de inicializar y controlar cada una de las articulaciones del robot NAO desde Python. Mediante un lazo while, esta función servirá de puente entre las instrucciones dictadas por el programa y el movimiento del robot en CoppeliaSim.

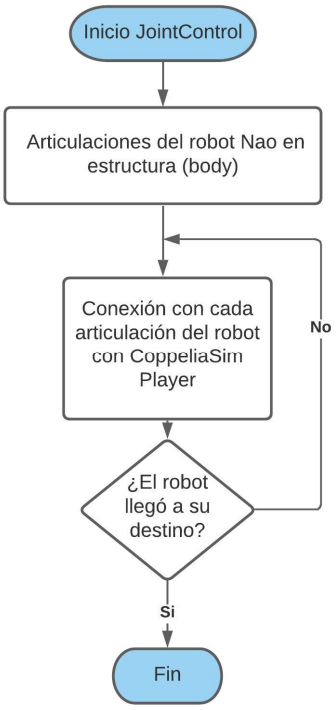


Figura 2. 46 Diagrama de flujo de JointControl

Una de las funciones principales que corre dentro del robot NAO es *Controlador* , cuyo diagrama de flujo esta representada en la Figura 2. 47. Esta función se encarga de obtener la información necesaria del robot en CoppeliaSim, como su ubicación, orientacion, velocidad lineal, angular, etc. así como también ejectuar el planificador de caminos que le dará la ruta o puntos que debe tomar el robot para que llegue a su destino.

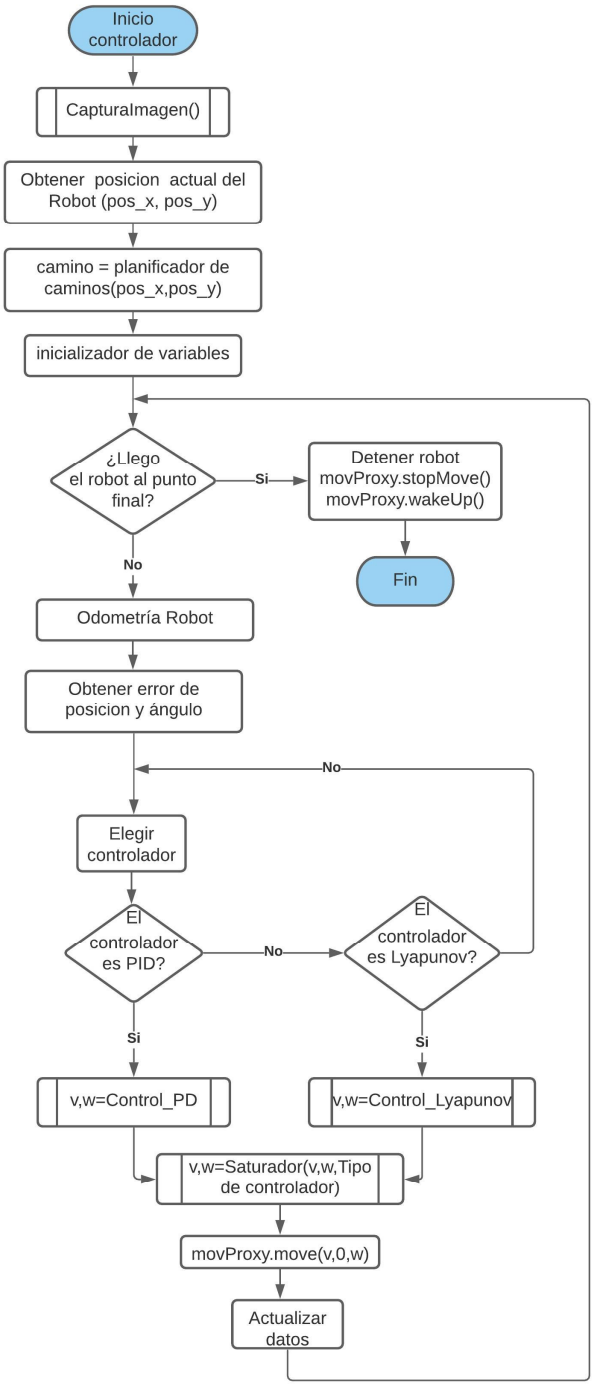


Figura 2. 47 Diagrama de flujo de Controlador

De igual manera, la interfaz gráfica hace uso de esta función para poder elegir entre los dos diferentes tipos de controladores: PID y Lyapunov, cuyos diagramas de flujo están representados por la Figura 2. 48.

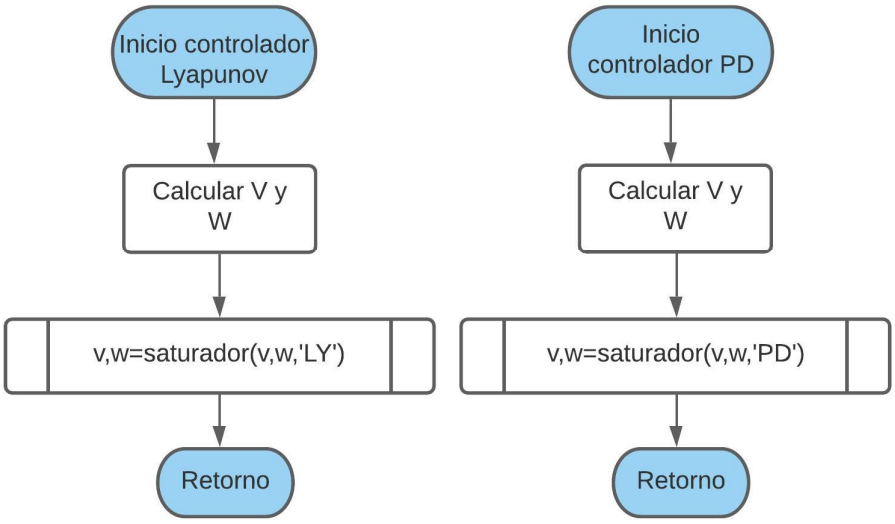


Figura 2. 48 Diagrama de flujo de controladores: Basado en Lyapunov y PD

2.5. DESARROLLO E IMPLEMENTACION DE PLANIFICADOR DE CAMINOS

Como se ha mencionado en el punto 2.2, para que el robot humanoide NAO pueda llegar a su objetivo deseado, debe seguir una sucesión de puntos preestablecido que forman un camino. Dicho camino puede ser generado mediante diferentes algoritmos como Campos Potenciales Artificiales, Mapas de Caminos Probabilísticos (PRM), Exploración Rápida de Árbol Aleatorio (RRT), etc. Para el caso de este Proyecto Técnico, el camino es generado mediante un algoritmo RRT*FN,

Previo a la ejecución del algoritmo RRT, sin embargo, es necesario realizar un procesamiento digital del entorno por donde se va a movilizar el robot Nao, permitiendo así conocer en donde se encuentra ubicados los diferentes obstáculos que se presenten entre el punto de partida y el punto de llegada.

2.5.1 PROCESAMIENTO DIGITAL DEL ENTORNO

Con el objetivo de que el robot Nao pueda desplazarse de un punto A un punto B, es necesario identificar cada uno de los obstáculos que se encuentren en el camino. Para ello,

primeramente, se debe tomar una imagen de la vista superior del aula con la cámara “Vision Sensor”, la cual proporciona una imagen en formato RGB de 512x512 pixeles representada por las ecuaciones 2.24, 2.25, 2.26, 2.27 y en la Figura 2. 49. Luego, se procede a cambiar la imagen a escala de grises, como se muestra en la Figura 2. 50, con el objetivo de facilitar la identificación de los obstáculos.

$$\begin{pmatrix} m_{11} & \cdots & m_{1n} \\ \vdots & \ddots & \vdots \\ m_{m1} & \cdots & m_{mn} \end{pmatrix}_{n_1 x n_2} \tag{2.24}$$

$$m_{ij} = \left(r_{ij}, g_{ij}, b_{ij}\right) \tag{2.25}$$

$$i = 1, 2, 3, ..., m \tag{2.26}$$

$$j = 1, 2, 3, ..., n \tag{2.27}$$

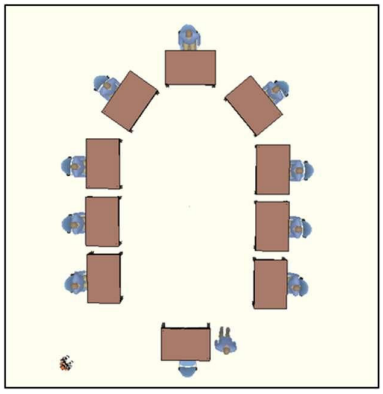


Figura 2. 49 Vista superior del aula de 512 x 512 pixeles.

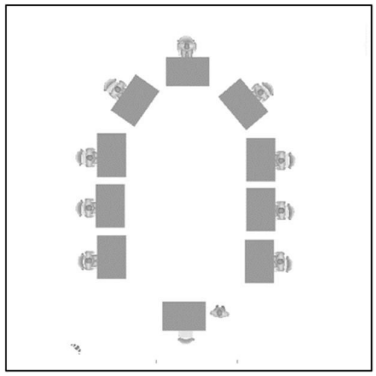


Figura 2. 50 Vista superior del aula de 512 x 512 pixeles en escala de grises.

Al obtener esta imagen representada por la matriz de 512 x 512 pixeles, Figura 2. 50, se puede identificar la ubicación de los obstáculos existentes representados en diferentes escalas de grises. De esta manera el algoritmo de planificación de caminos RRT*FN puede

distribuir cada una de sus ramificaciones y asegurar que el robot no se encuentre con algún obstáculo en su camino. Sin embargo, dicho algoritmo no toma en consideración el grosor del robot, con lo que existe el peligro de que algunos de los puntos generados se ubiquen demasiado cerca de algún obstáculo y el robot choque. Para solucionar este inconveniente, se aplica un redimensionamiento de cada uno de los obstáculos, es decir, se hace a cada uno de los obstáculos de un tamaño mayor al que en realidad son. Esto se puede evidenciar en la Figura 2. 51.

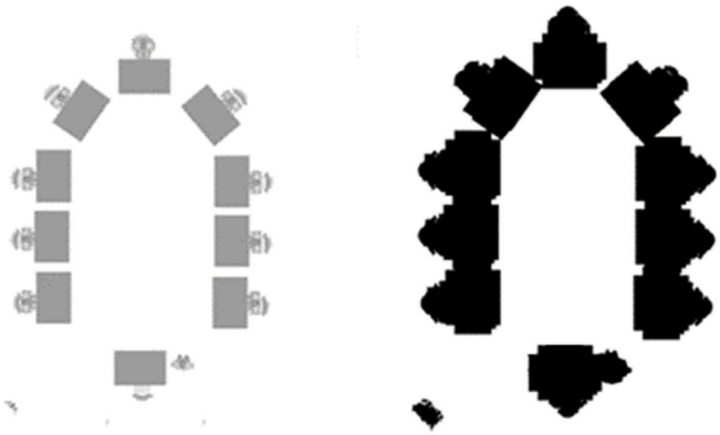


Figura 2. 51 Vista superior del aula sin y con redimencionamiento de obtáculos.

En la Figura 2. 52, se puede apreciar el diagrama de flujo de todo el proceso que se ha descrito anteriormente.

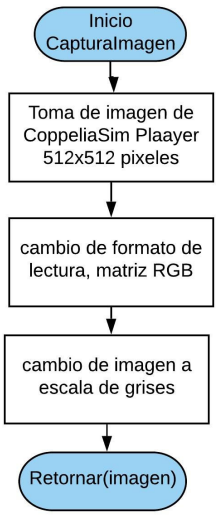


Figura 2. 52 Diagrama de flujo de CapturaImagen

2.5.2 RRT*FN

En el capitulo 1, Subseccion 1.3.3.1 se explicaron los terminos generales utilizados para la implementacion del siguiente algoritmo: ARBOL DE EXPLORACION RAPIDA ALEATORIA DE NODO FIJO (RRT*FN)

RRT*FN, a diferencia de un RRT convencional, se caracteriza no solamente por demandar una menor memoria en el sistema, sino también por imponer una restricción en el número de nodos expandidos [33], es decir, elimina tanto nodos locales como nodos globales.

Tal como se muestra en la Figura 2. 53, solamente si la función de costo de la ruta resultante es mejor, un nodo local elimina los nodos con un solo hijo de un vecino cercano, cada vez que recablea el árbol en busca del camino. De no existir estos nodos, se elimina un nodo global (nodo que se caracteriza de no tener hijos) [33].

La razón para los dos tipos de procedimientos en la eliminación de nodos es debido a que, si un nodo no tiene hijos, significa que no se encuentra en camino de alcanzar el objetivo final. Mediante RRT * FN, una vez establecida conexión con el objetivo final, el algoritmo continúa optimizando el camino resultante, ya sea agregando mejores nodos, eliminando los que nodos que no tienen hijos, y agregando nodos adicionales para mejorar el camino. Durante este procedimiento, se busca posibles caminos a través de toda la imagen, tomando en cuenta que cada uno de los obstáculos permanecen en su posición original al momento de iniciar el algoritmo RRT*FN, y una vez terminado el ciclo de interacciones retornara el camino resultante, por el que se desplazara el robot [34].

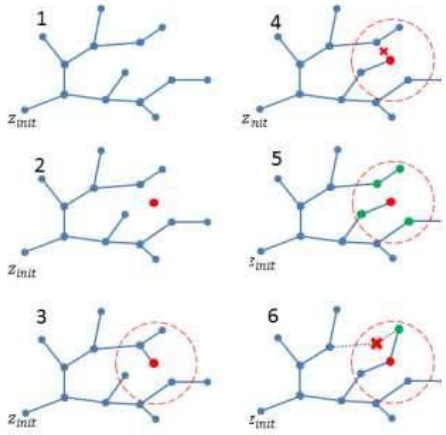


Figura 2. 53 Eliminacion de un nodo local, en funcion de su costo (PATH) *[33]*.

En la Figura 2. 54, Figura 2. 55 y Figura 2. 56 se muestran los algoritmos necesarios para la implementación del RRT*FN utilizada para planificar la ruta que el robot seguirá.

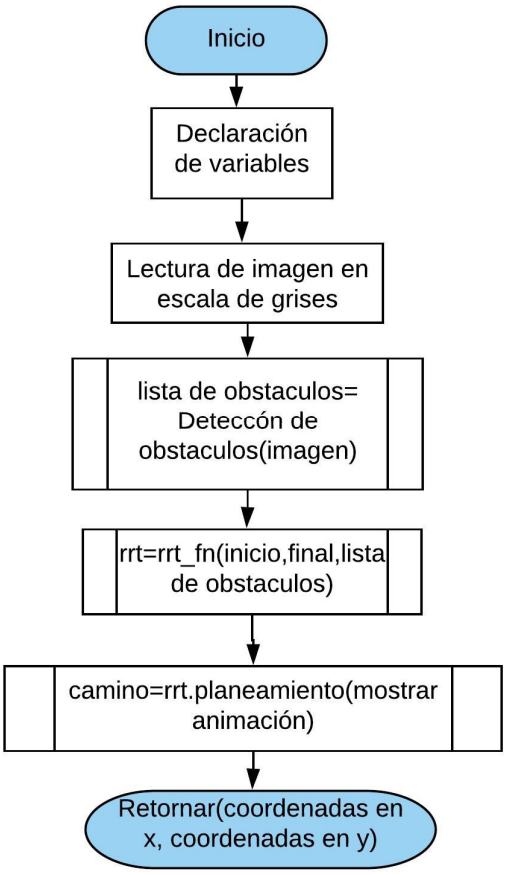


Figura 2. 54 Diagrama de flujo Estructura principal RRT

La Figura 2. 54 muesta la estructura principal del algoritmo RRT*FN utilizada para planificar la ruta que el robot seguirá.

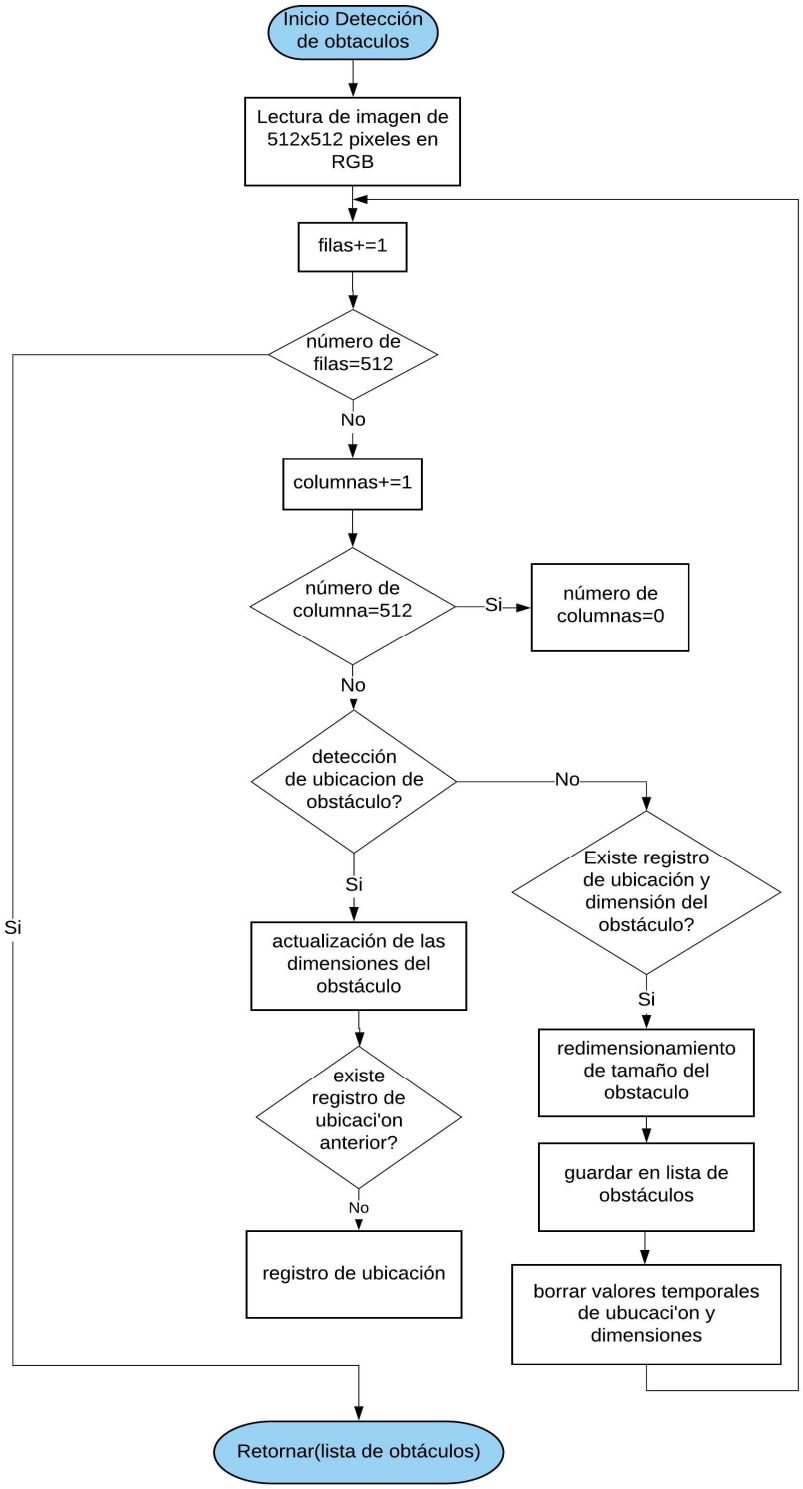


Figura 2. 55 Diagrama de flujo Detección de obstculos

La Figura 2. 55 explica el algoritmo utlizado para la deteccion de obstaculos utilizado como parrte de la planificación de la ruta que el robot seguirá.

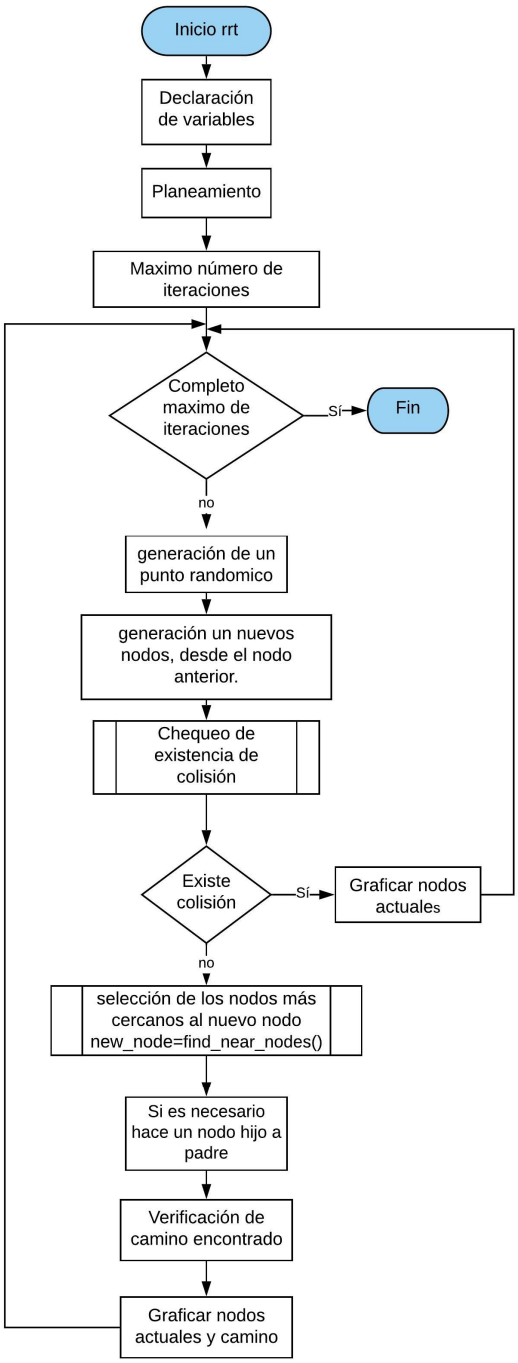


Figura 2. 56 Diagrama de flujo RRT

La Figura 2. 56 explica el algoritmo utlizado para la generacion de nodos y grafico del camino utilizados como parrte de la planificación de la ruta que el robot seguirá.

2.6. DESARROLLO DE INTERFAZ GRÁFICA

La interfaz de usuario se realiza con la finalidad de que cualquier operador humano pueda interactuar con el proceso de manera sencilla y rápida. En el caso de este trabajo de titulación, se ha decidido desarrollar la interfaz gráfica en PQdesigner. La HMI desarrollada contiene cuatro niveles, con una determinada funcionalidad de cada botón, que se explica a detalle en la sección de Anexos.

El primer nivel presenta la información introductoria del trabajo, tal como se muestra en la Figura 2. 57, en cual contiene un botón de ingresar al siguiente nivel, y el botón de salida.

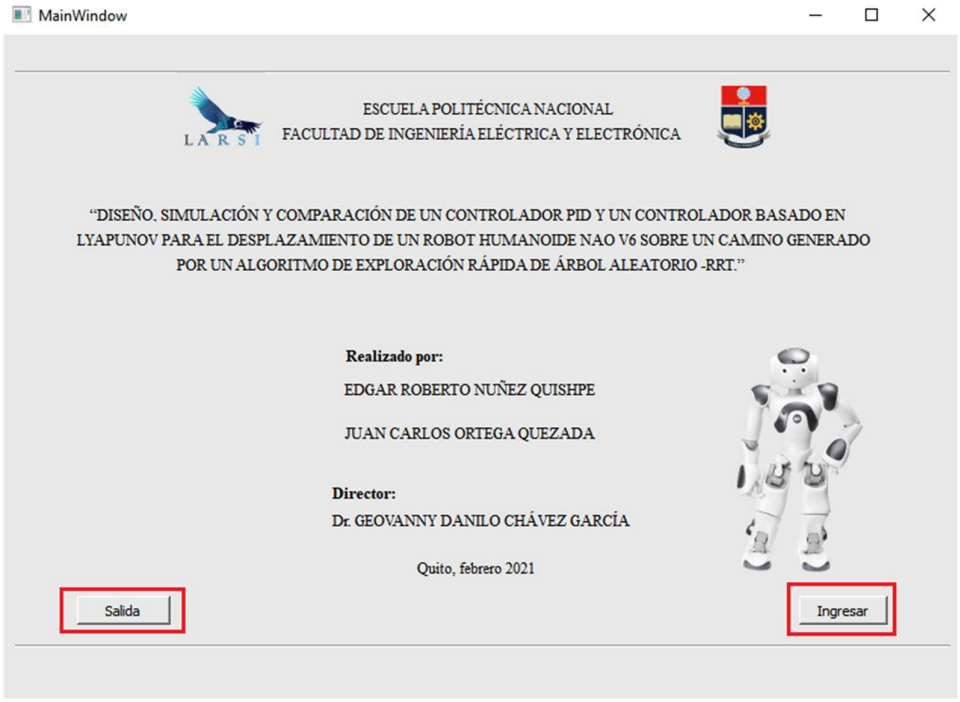


Figura 2. 57. Portada de la interfaz gráfica desarrollada.

En el segundo nivel, contiene tres secciones relevantes: la pantalla de selección del tipo aula ubicado al lado izquierdo, la visualización del camino generado ubicado en el centro de la ventana y las opciones del tipo de controlador (Controlador tipo PID y el Controlador basado en Lyapunov para la interacción con el robot ubicado en el lado derecho), como se presenta en la Figura 2. 58.

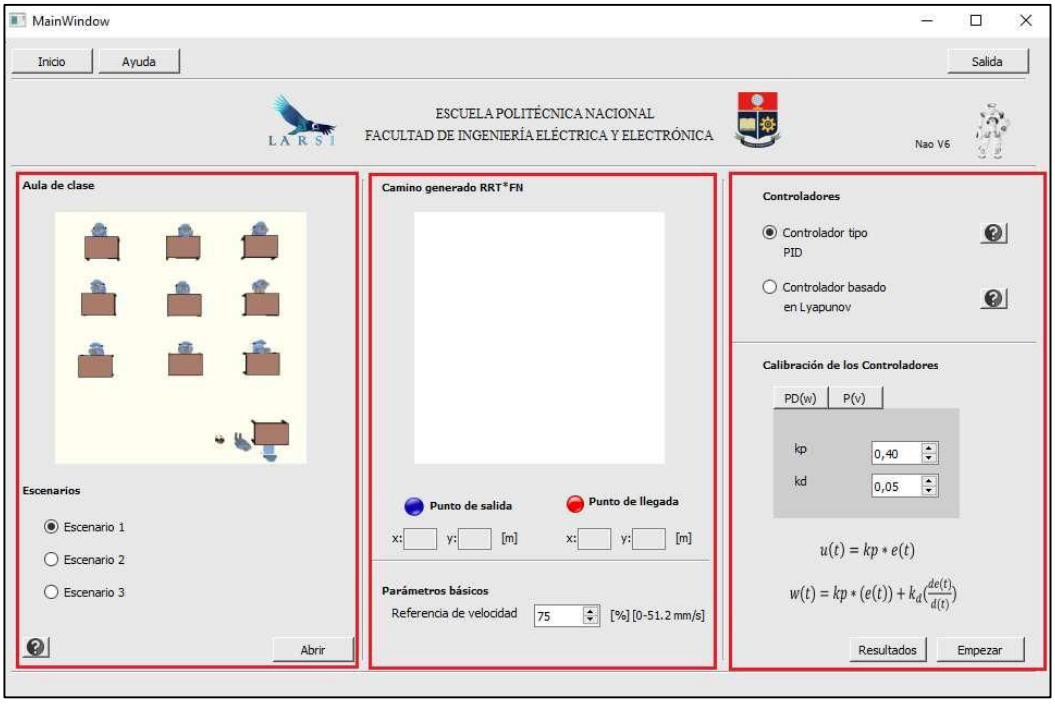


Figura 2. 58 Pantalla de proceso de control del robot NAO.

El tercer nivel es de ayuda al usuario, con la información relevante para poder operar la interfaz del robot y en respuesta de interrogantes de que significa cada variable ingresada en la HMI como se presenta en la Figura 2. 59.

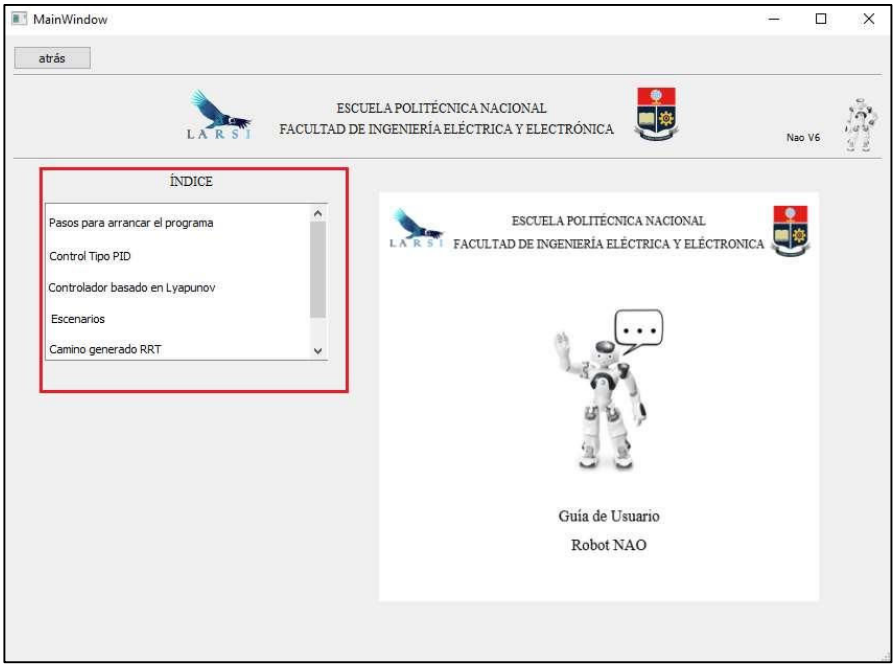


Figura 2. 59 Pantalla de ayuda al usuario.

Finalmente, el cuarto nivel corresponde a la visualización de los resultados obtenidos como se presenta en la Figura 2. 60, con un menú para poder seleccionar el resultado que se desea visualizar en la interfaz.

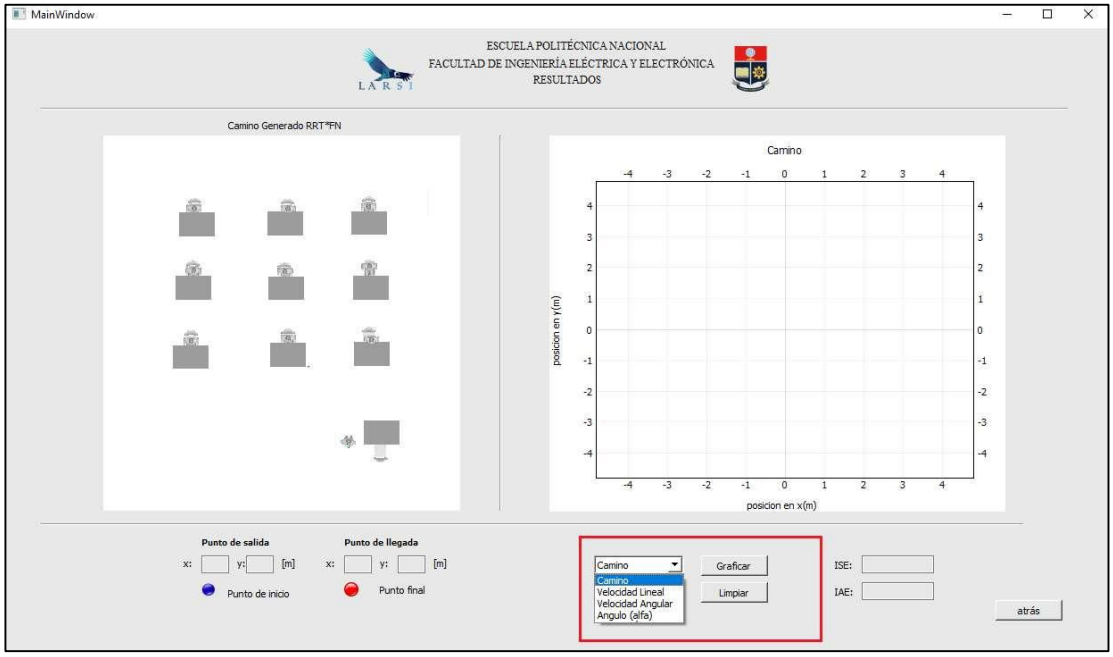


Figura 2. 60 Pantalla de resultados obtenidos.

Las gráficas mostradas en el cuarto nivel son las salidas del proceso, índices de rendimiento, puntos de inicio, punto final, camino generado, ángulo de orientación, velocidad angular, y velocidad lineal.

2.6.1 DIAGRAMA DE LA INTERFAZ.

En esta sección se muestra el diagrama de flujo de la interfaz gráfica propuesta (Figura 2. 61). De igual manera, en la sección de anexos se incluye el manual de usuario de como inicializar la interfaz, y la utilidad de cada elemento. Adicionalmente, se muestra cada uno de los pasos a seguir para la instalación del software necesario para el correcto funcionamiento del proyecto técnico.

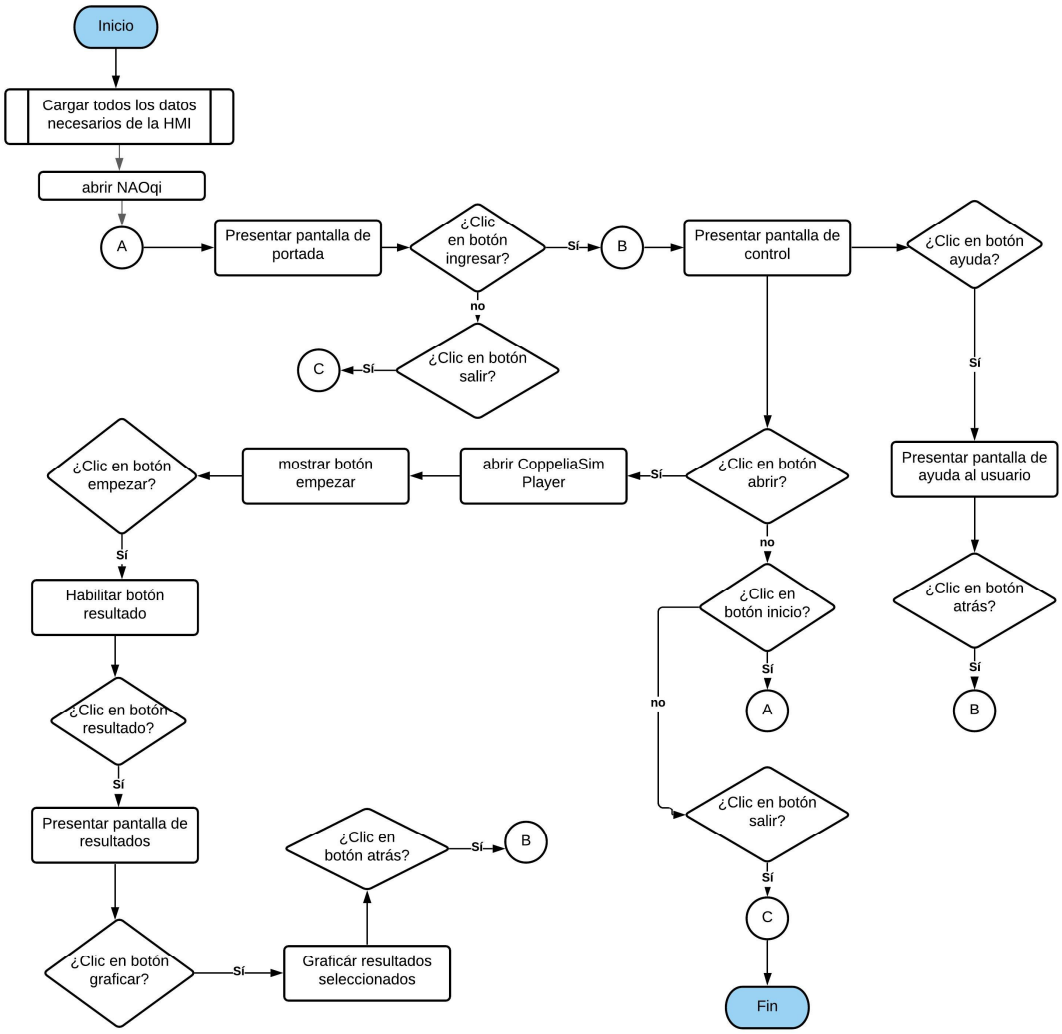


Figura 2. 61 Diagrama de flujo Interfaz gráfica

3. RESULTADOS Y DISCUSIÓN

Como se pudo observar en el capítulo 2, para que el robot Nao pueda movilizarse desde un punto A hacia un punto B en el simulador CoppeliaSim, se ejecuta un proceso ordenado en donde se ven envueltos diferentes programas y librerías. En la Figura 3. 1 se puede apreciar un esquema de bloques simplificado que explica como interactúan los diferentes programas y librerías utilizadas en el proyecto. Al dar inicio al proceso, se ejecuta el software NAOqi y la interfaz gráfica de usuario. A continuación, una vez que se haya ejecutado el simulador CoppeliaSim y se haya elegido algún punto hacia donde se desea que llegue el robot (generando así el camino dado por el algoritmo RRT), se ejecuta la función Inicializador la cual se encarga de realizar la comunicación entre Python y CoppeliaSim mediante la API. Una vez establecida dicha comunicación, el robot ya puede recibir mandos mediante Python a través de NAOqi y posteriormente mediante el uso de la librería JointControl, la cual envía cada una las instrucciones de movimiento al robot en el simulador. Finalmente, para poder efectuar el control tanto de velocidad lineal como de velocidad angular, se necesita enviar la ubicación y ángulo real del robot desde CoppeliaSim al programa cerrando así el lazo y haciendo que se repita nuevamente todo el proceso.

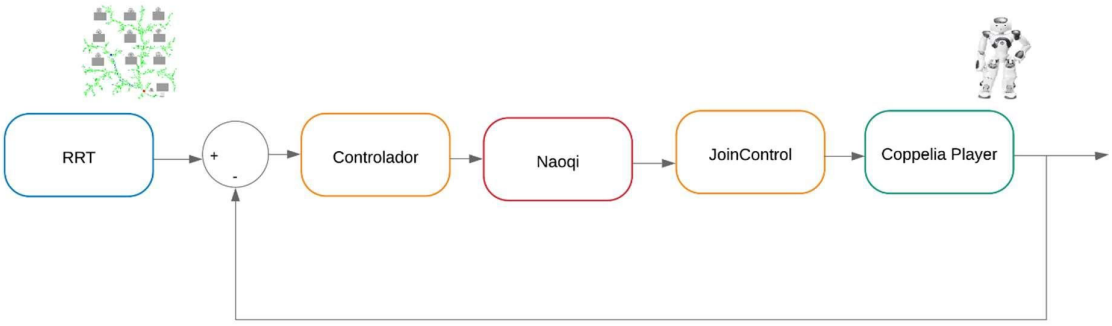


Figura 3. 1 Diagrama de bloques del proceso de ejecución del proyecto

3.1 PRUEBA CON OBSTÁCULOS EN ESCENARIOS CONTROLADOS APLICANDO EL ALGORITMO RRT

A continuación, se presenta los resultados obtenidos por ambos controladores, PID y Lyapunov, al exponer al robot Nao a 3 diferentes escenarios de aulas de clase. Primeramente, se analizará el camino generado por el algoritmo RRT y la respuesta de seguimiento de camino del robot Nao. Luego, se mostrará una tabla resumida de la acción de control, constantes del controlador e índice de rendimiento de la prueba, así como también la respuesta de velocidad lineal, velocidad angular y ángulo del robot. En cada uno de los diferentes escenarios, se ha configurado al robot a 3 velocidades lineales diferentes:

50% para el primer escenario, 75% para el segundo escenario y 100% para el último y tercer escenario

3.1.1 PRUEBA ESCENARIO UNO Y VELOCIDAD LINEAL DE 50 % (25mm/s)

Como ya se ha mencionado en el capítulo 2, el primer escenario se trata de un aula convencional de clases en donde el docente, la mayor parte del tiempo, se encuentra frente a los alumnos y los pupitres se colocan de manera uniforme en filas y columnas. Tal y como se observa en la Figura 3. 2, se ha colocado al robot Nao en su posición inicial (junto al docente) para que desde ese punto se movilice hacia cualquiera de los pupitres. Se puede observar además que el algoritmo RRT genera suficientes ramificaciones a través del aula para que el robot tenga la posibilidad de movilizarse a cualquier punto que el usuario seleccione. Cabe recalcar que ninguna de estas ramificaciones pasa a través de un obstáculo (objetos grises), asegurando así la evasión de obstáculos del algoritmo. En este ejemplo, el robot debe movilizarse desde su punto de inicio predeterminado hacia uno de los estudiantes ubicado en la parte izquierda del aula. El camino generado (línea roja) por el robot se divide en diferentes puntos que serán entregados al robot para su seguimiento.

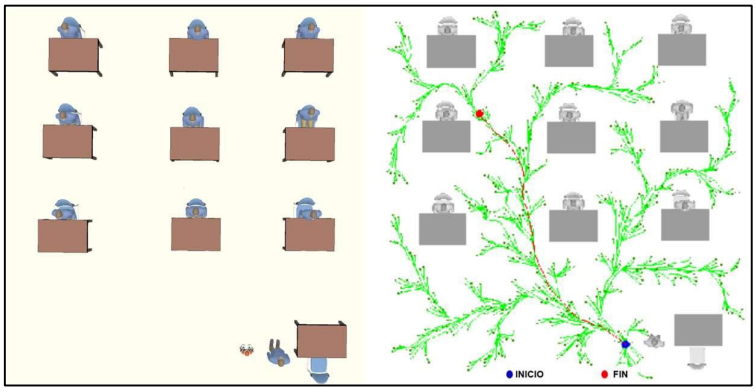


Figura 3. 2 Camino generado por algoritmo RRT – Escenario uno.

A continuación, se muestra los resultados obtenidos con el controlador PID.

3.1.1.1 Prueba con controlador tipo PID

Tabla 3. 1 Datos controlador tipo PID - Escenario uno

Algoritmo de control	**Control tipo PID**	
Acción de control	$u(t) = kp * e(t)$	
	$\omega(t) = kp * (e(t) + T_d \frac{de(t)}{d(t)})$	
Constantes	K_p	0.4
	K_d	0.05
Índices de rendimiento	ISE	0.0026
	IAE	0.0213
	$TVu(u)$	1.938
	$TVu(\omega)$	21.477
Tiempo empleado	$5\,min\,14\,seg$	

Camino:

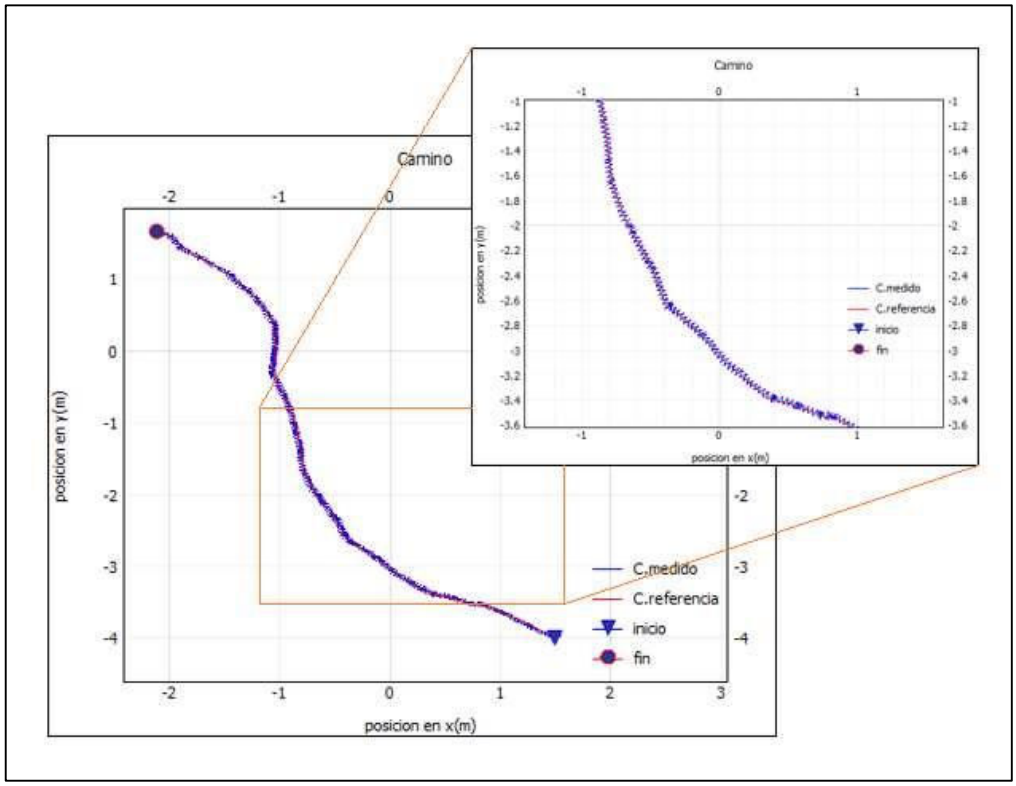


Figura 3. 3 Camino recorrido por el robot humanoide Nao con controlador PID

Como se puede observar en la Figura 3. 3, el robot es capaz de desplazarse desde su punto de inicio hasta el punto final sin desviarse del camino generado por el algoritmo RRT. De igual manera, se muestra una parte del camino generado con el objetivo de observar de mejor manera el comportamiento del robot al caminar. Cabe recalcar que, como ya se ha explicado en secciones anteriores, una de las características inherentes del robot al caminar es el balanceo que se puede notar en la señal medida (señal azul).

Velocidad lineal:

Es importante recordar que, ante curvas pronunciadas, el robot es aún capaz de seguir su camino dado a que, durante el tiempo de giro, el robot se detiene en velocidad lineal durante unos instantes mientras el error de ángulo se mantenga mayor a 15 grados. Gracias a esta configuración el robot no se desvía de su camino o presenta errores de posición. Este comportamiento se puede observar en la Figura 3. 4 en donde, a pesar de que la velocidad de referencia dada desde la interfaz es del 50%, es decir un valor constante, la acción de control de velocidad lineal es igual a cero en los intervalos donde el robot debe realizar al giro mayor a 15 grados.

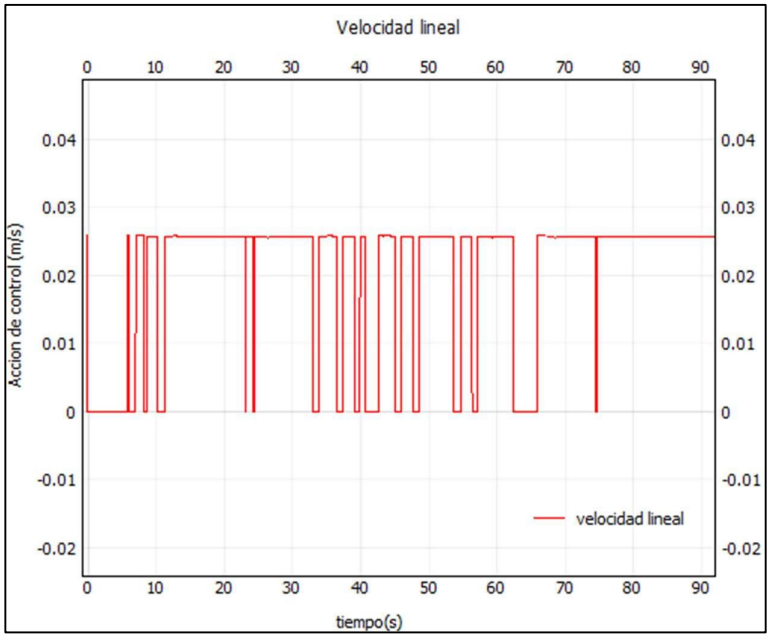


Figura 3. 4 Velocidad lineal del robot humanoide Nao con Controlador PID

Velocidad angular:

En el caso de la velocidad angular, existe una acción de control cada vez que existe un cambio de referencia de nuestro ángulo alfa deseado, observando en la Figura 3. 3 como el robot sigue a la referencia tomando en cuenta todas las consideraciones que se revisó en la sección 2.3 – Implementación de los algoritmos de control. Como se observa en la Figura 3. 5, dado que el robot se toma un tiempo considerable en realizar todo el recorrido del camino (5 minutos con 14 segundos), se muestra solamente una sección de la acción de control de velocidad angular con el objetivo de visualizar de mejor manera su comportamiento.

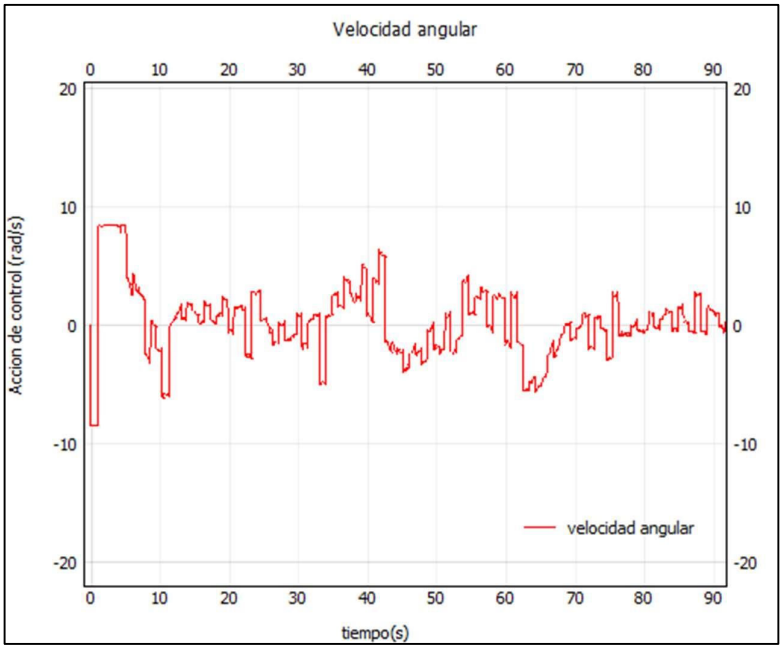


Figura 3. 5 Velocidad angular del robot humanoide Nao con Controlador PID

Ángulo:

Finalmente, en la Figura 3. 6 se puede observar que el robot, en condiciones iniciales, se encuentra con un ángulo de 90 grados con respecto al eje de referencia de su espacio de trabajo y, a partir de ese punto, alcanza la orientación de referencia para poder llegar al siguiente punto.

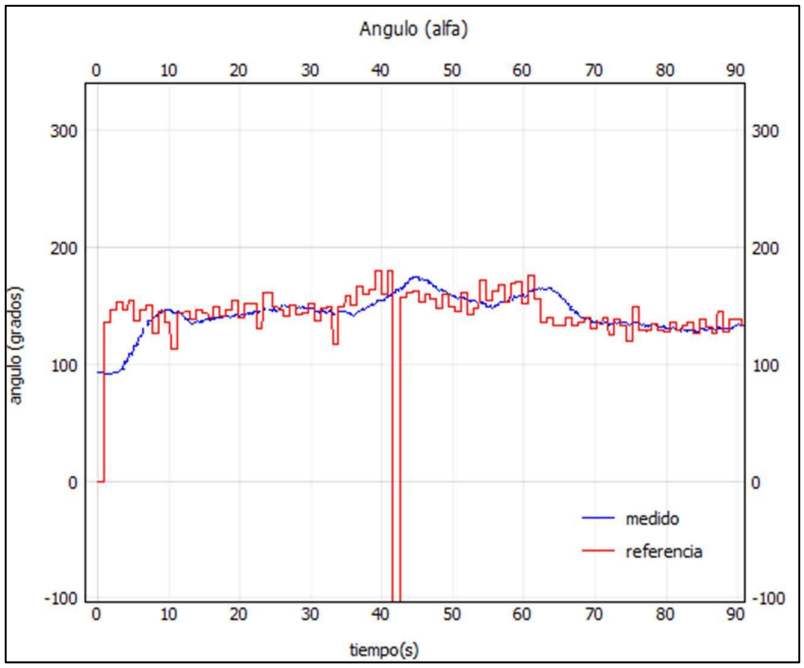


Figura 3. 6 Angulo del robot humanoide Nao con respecto al marco de referencia de CoppeliaSim

A continuación, se muestra los resultados obtenidos con el controlador basado en Lyapunov.

3.1.1.2 Prueba con controlador basado en Lyapunov

Tabla 3. 2 Datos controlador basado en Lyapunov - Escenario uno

Algoritmo de control	**Control basado en Lyapunov**	
Acción de control	$u = K_u \tanh(\rho)\cos(\alpha)$	
	$\omega = K_\omega \alpha + K_u \frac{\tanh(\rho)}{\rho}\sin(\alpha)\cos(\alpha)$	
Constantes	K_u	0.0512
	K_w	0.4
Índices de rendimiento	ISE	0.00023
	IAE	0.00833
	$TVu(u)$	4.170
	$TVu(\omega)$	12.980
Tiempo empleado	$5\,min\,12\,seg$	

Camino:

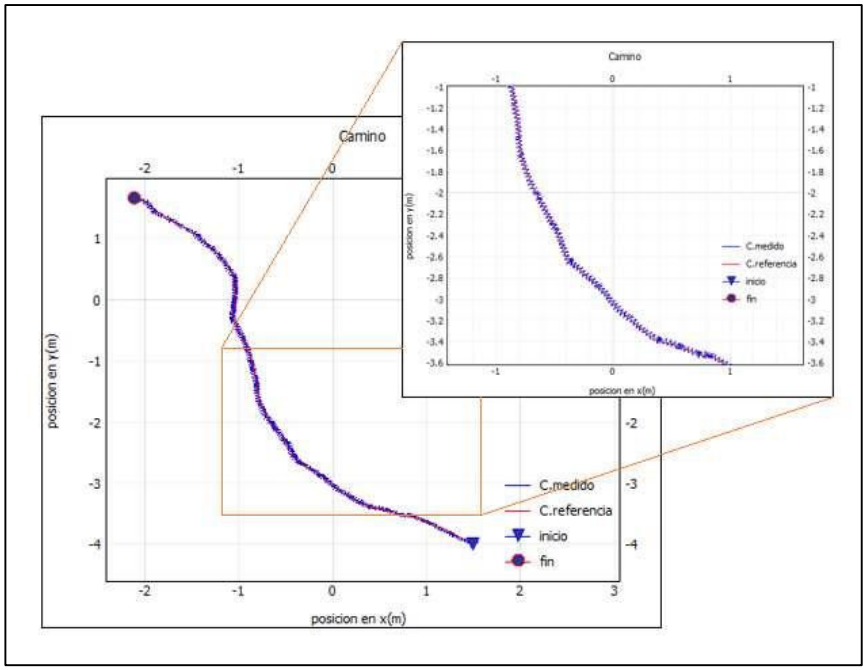


Figura 3. 7 Camino recorrido por el robot humanoide Nao con controlador basado en Lyapunov

En el caso del controlador basado en Lyapunov, el robot también es capaz de seguir el camino generado por el algoritmo RRT tal y como se observa en la Figura 3. 7. En este caso, el robot logra alcanzar su objetivo en un tiempo similar al alcanzado con el controlador PID. Sin embargo, este tiempo podría disminuir considerablemente al aumentar la velocidad lineal del robot.

Velocidad lineal:

Al igual que en el caso anterior, la acción de control de velocidad lineal del robot tenderá a cero cada vez que el error del ángulo del robot supere los 15 grados. Sin embargo, tal y como se observa en la Figura 3. 8, este comportamiento solamente es momentáneo y bastante útil para que el robot pueda seguir su camino y no desviarse al encontrarse con curvas pronunciadas. Cabe recalcar que, a diferencia del primer caso, la acción de control de velocidad lineal del robot disminuirá a medida que el robot llegue a su destino final. Solamente se muestra una fracción del comportamiento de la acción de control de velocidad lineal ya que, como se mencionó anteriormente, el robot toma más de 5 minutos llegar a su objetivo.

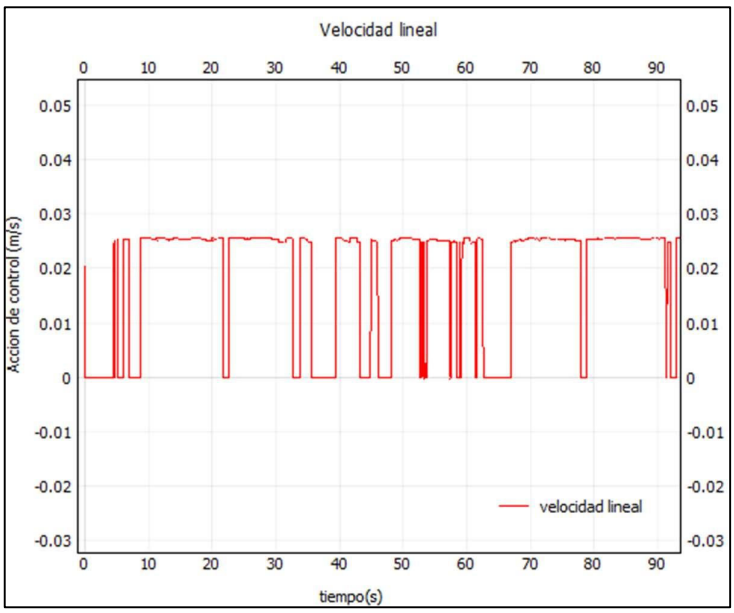


Figura 3. 8 Velocidad lineal del robot humanoide Nao con controlador basado en Lyapunov

Velocidad angular:

En el caso de la velocidad angular, se puede evidenciar que el controlador funciona correctamente ya que logra seguir la referencia obtenida a partir de la odometría del robot como se puede observar en la Figura 3. 10. La señal de referencia será mayor cada vez que sea un necesario un giro más pronunciado del robot.

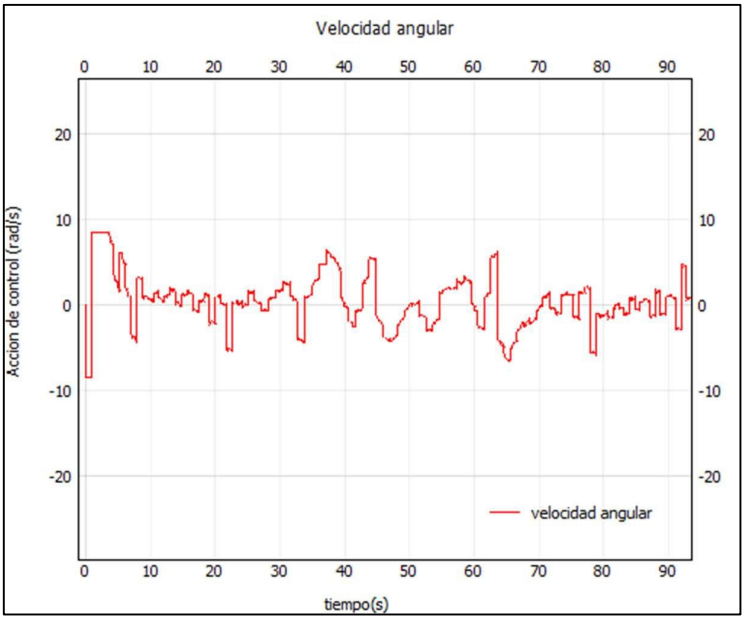


Figura 3. 9 Velocidad angular del robot humanoide Nao con controlador basado en Lyapunov

Ángulo:

Finalmente, en la Figura 3. 10, se puede observar la tendencia del ángulo a seguir su referencia. Como ya se ha explicado anteriormente, las oscilaciones presentes en la señal de referencia se deben a características inherentes en la odometría del robot.

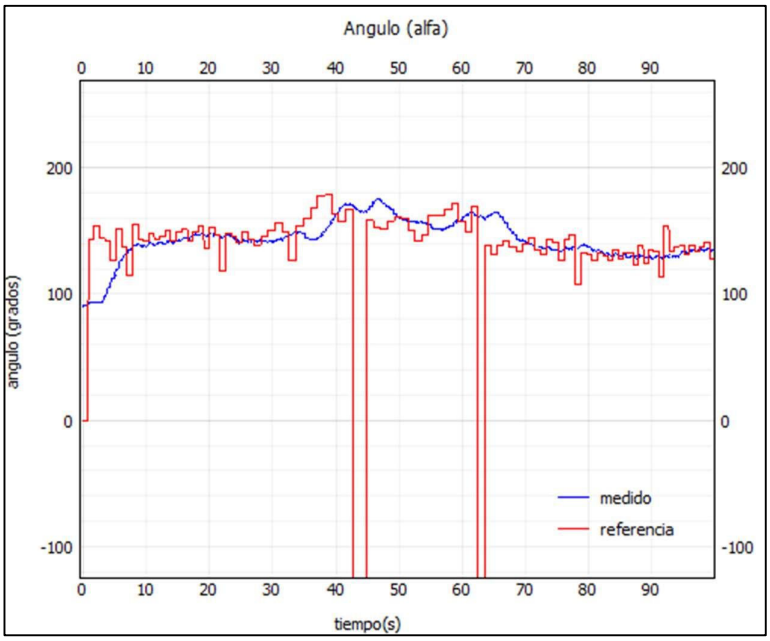


Figura 3. 10 Angulo del robot humanoide Nao con respecto al marco de referencia de CoppeliaSim

3.1.1.3 Comparación entre ambos controladores

Tabla 3. 3 Comparación de ISE, IAE y TVu de ambos controladores

	Controlador tipo PID	Controlador basado en Lyapunov
Tiempo empleado	$5min\ 14seg$	$5min\ 12seg$
$\boldsymbol{ISE}$	0.00266	0.000236
$\boldsymbol{IAE}$	0.02135	0.00833
$\boldsymbol{TVu(u)}$	1.938	4.170
$\boldsymbol{TVu(\omega)}$	21.477	12.980

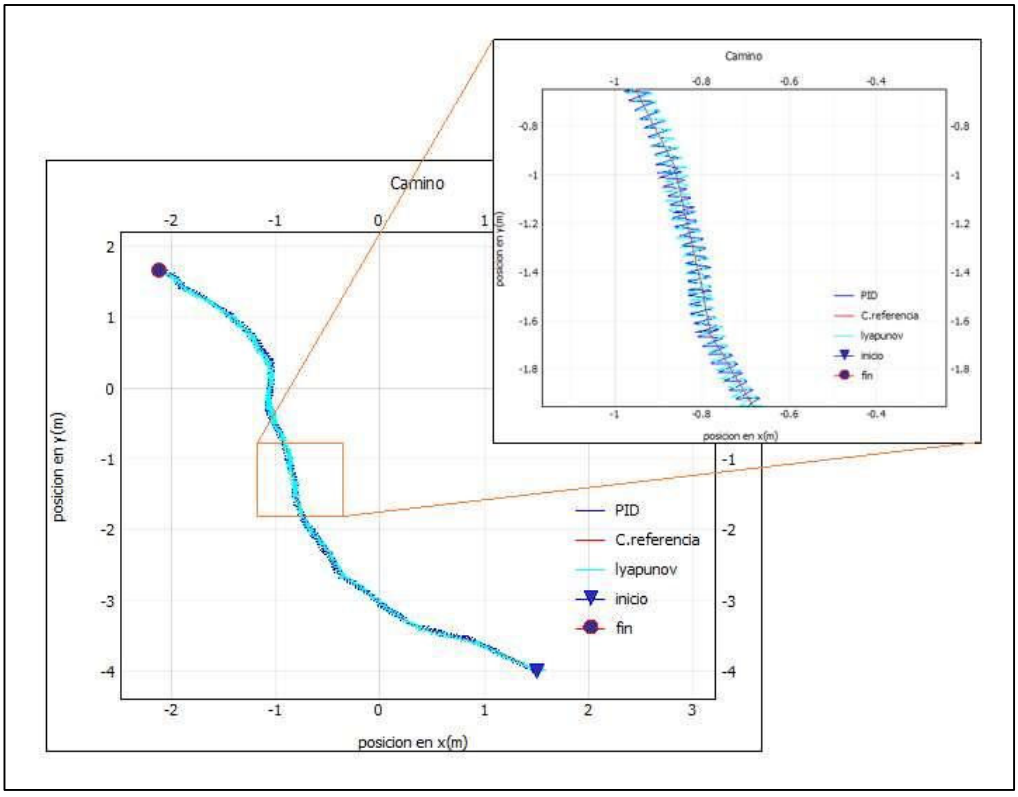


Figura 3. 11 Comparación entre el algoritmo de control PID y basado en Lyapunov en el camino efectuado por el robot Nao

Al observar la Figura 3. 11 se puede evidenciar que ambos controladores cumplen con el objetivo de seguimiento de camino propuesto. Sin embargo, como se puede notar en la Tabla 3.3, el controlador basado en Lyapunov posee un índice de rendimiento menor al del controlador PID y de hecho, tomar unos segundos menos a que llegue su objetivo, lo cual resulta en un mejor desempeño general a velocidades del 50%.

3.1.2 PRUEBA ESCENARIO DOS Y VELOCIDAD LINEAL DE 75 % (37.5mm/s)

Como se puede notar en la Figura 3. 12, se ha colocado al robot Nao en la esquina inferior izquierda del escenario dos ya que desde este punto puede, con más facilidad, ingresar al

centro del aula o desplazarse alrededor de los pupitres, permitiendo así poder acercarse a cualquiera de los estudiantes o hacia el docente. De manera similar al caso anterior, el algoritmo RRT genera suficientes ramificaciones dentro de su espacio de trabajo para que el usuario pueda seleccionar la ubicación a donde quiere que el robot se movilice sin que choque con ningún obstáculo presente. En este ejemplo, se ha optado que el robot se desplace hacia uno de los estudiantes ubicado en uno de los pupitres de la derecha del aula ya que, para ello, debe evadir ciertos obstáculos en su camino. En la parte derecha de la Figura 3. 12 se puede notar como el camino generado (línea roja) evade los obstáculos en el camino y permite al robot llegar a su objetivo. Este camino se divide en diferentes puntos los cuales van a ser entregados al robot para su seguimiento.

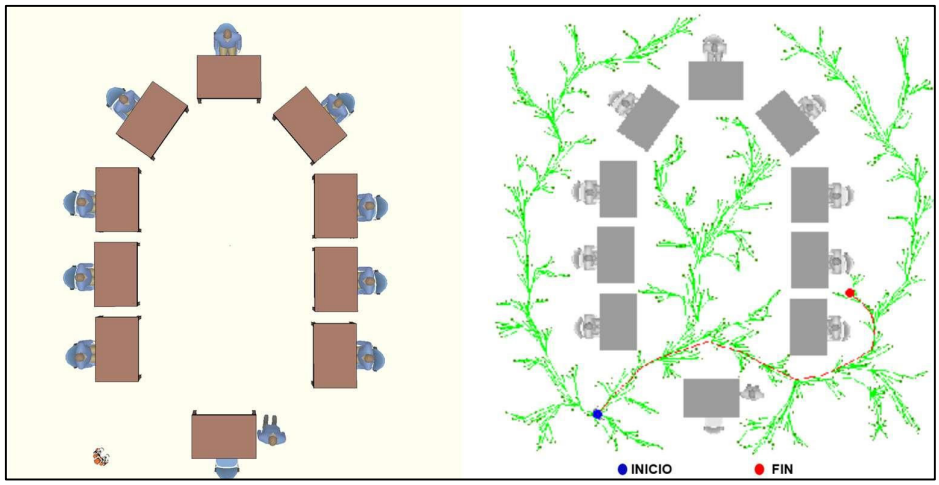


Figura 3. 12 Camino generado por algoritmo RRT – Escenario dos.

3.1.2.1 Prueba con controlador PID

Tabla 3. 4 Datos controlador tipo PID - Escenario dos

Algoritmo de control	**Control tipo PID**	
Acción de control	$u(t) = kp * e(t)$	
	$\omega(t) = kp * (e(t) + T_d \frac{de(t)}{d(t)})$	
Constantes	K_p	0.4
	K_d	0.05
Índices de rendimiento	ISE	0.00189
	IAE	0.02802
	$TVu(u)$	7.001
	$TVu(\omega)$	16.580
Tiempo empleado	$4\,min\,16\,seg$	

Camino:

Como se puede observar en la Figura 3. 13, el robot logra seguir el camino propuesto por el algoritmo RRT tomando en cuenta que en esta ocasión, este se encuentra expuesto a algunas curvas pronunciadas.

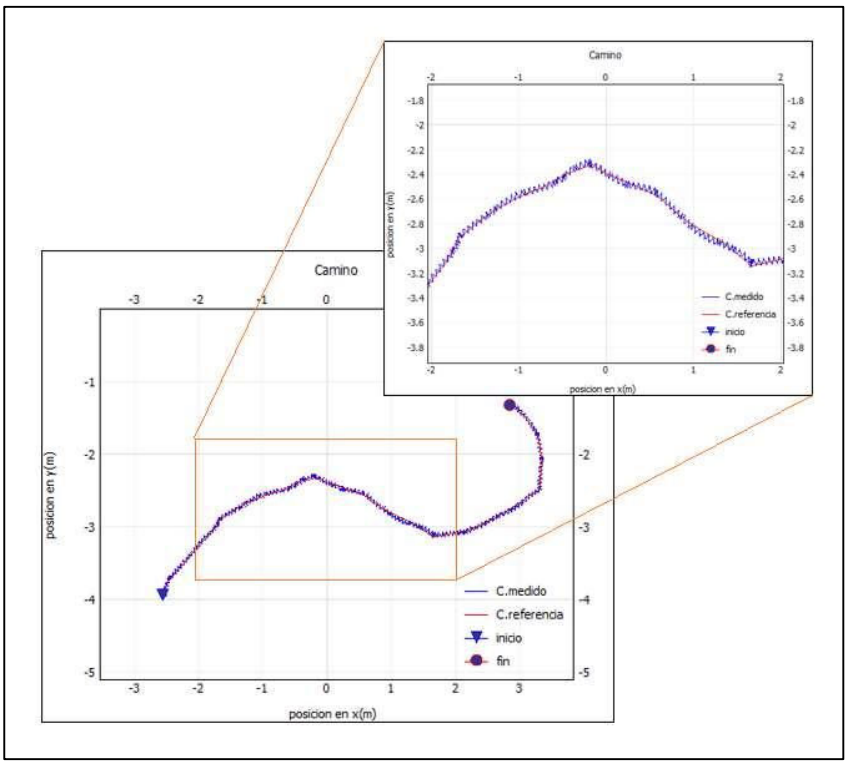


Figura 3. 13 Camino recorrido por el robot humanoide Nao con controlador PID

Velocidad lineal:

Al incrementar la velocidad a un 75% el robot es capaz de llegar a su objetivo en menor tiempo. Sin embargo, esto conlleva más probabilidad de que el robot se desvié de su camino especialmente en curvas pronunciadas. Es por tal motivo que en este caso también se ha decidido implementar la configuración que, en el primer escenario, el cual consiste en que cuando se presente errores de ángulo mayores a 15 grados, el robot detendrá momentáneamente su velocidad lineal, es decir su acción de control será igual a cero hasta poder orientarse nuevamente hacia su punto objetivo siguiente. Tal comportamiento se puede evidenciar en la Figura 3. 14.

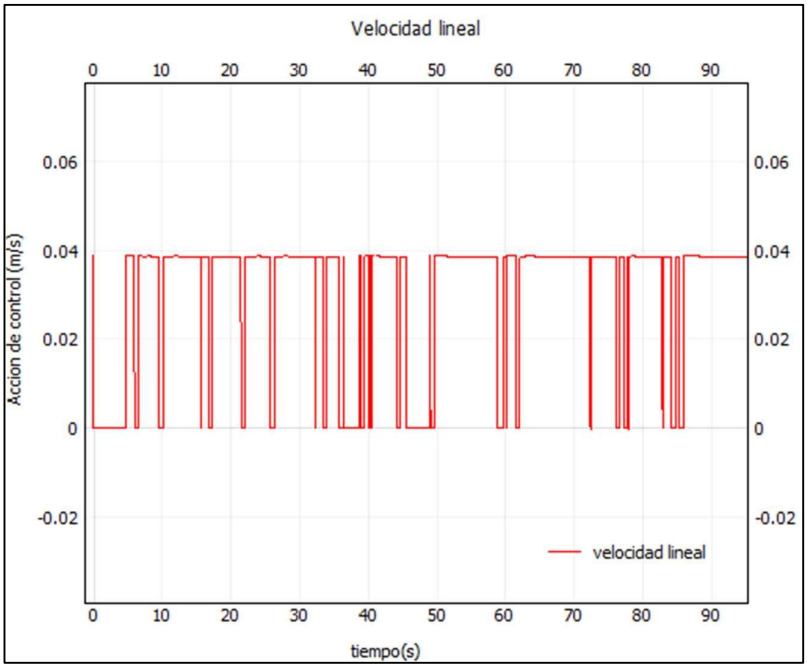


Figura 3. 14 Velocidad lineal del robot humanoide Nao con Controlador PID

Velocidad angular:

Al igual que en el primer escenario, las acciones de control de velocidad angular son el resultado del error en el ángulo de orientación, logrando así que el robot gire a su ángulo deseado de referencia sin ninguna complicación, lo cual se refleja en que siga al camino sin ningún error, especialmente en las curvas pronunciadas como se puede evidencia en la Figura 3. 13 . Cada vez que se presente este tipo de curvas, la acción de control de velocidad angular será mayor tal y como se observa en la Figura 3. 15.

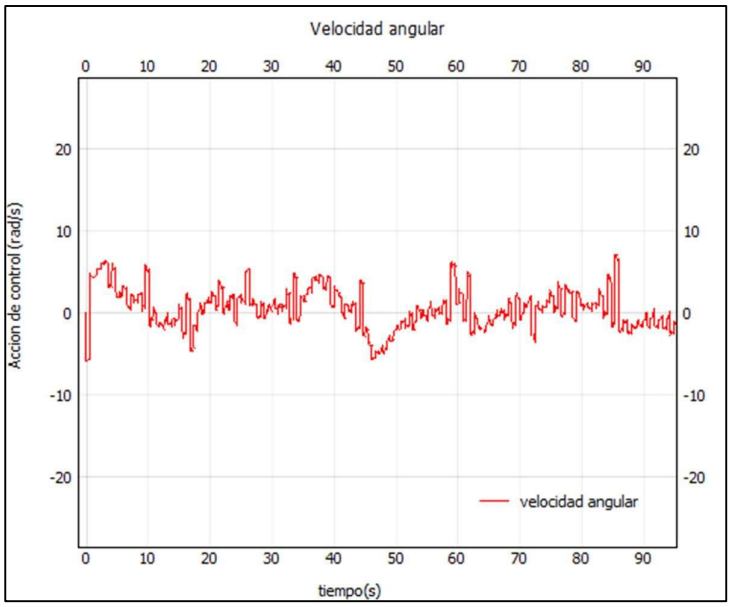


Figura 3. 15 Velocidad angular del robot humanoide Nao con Controlador PID

Ángulo:

Finalmente, en la Figura 3. 16 se puede observar que el robot en condiciones iniciales se encuentra con un ángulo de 45 grados con respecto al eje de referencia de CoppeliaSim y, partir de ese punto, alcanza la orientación de referencia para poder llegar al siguiente punto.

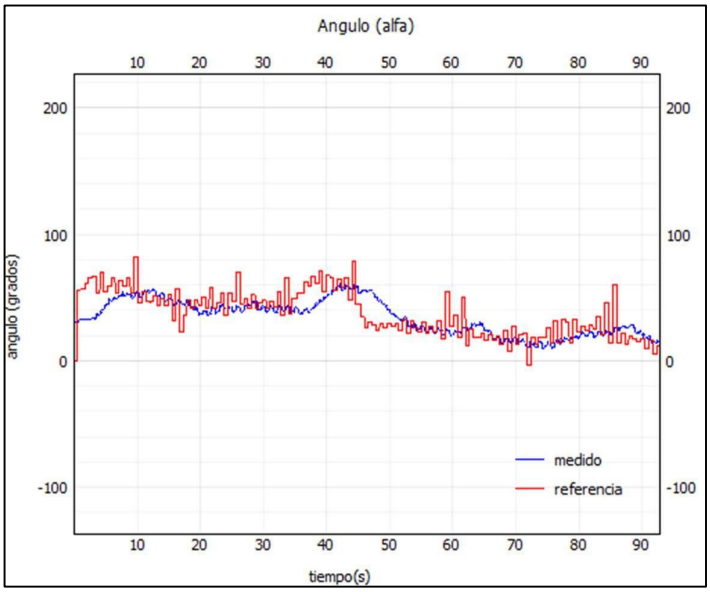


Figura 3. 16 Angulo del robot humanoide Nao con respecto al marco de referencia de CoppeliaSim

3.1.2.2 Prueba con controlador basado en Lyapunov

Tabla 3. 5 Datos controlador basado en Lyapunov - Escenario dos

Algoritmo de control	**Control basado en Lyapunov**	
Acción de control	$u = K_u \tanh(\rho)\cos(\alpha)$	
	$\omega = K_\omega \alpha + K_u \frac{\tanh(\rho)}{\rho}\sin(\alpha)\cos(\alpha)$	
Constantes	K_u	0.0512
	K_w	0.4
Índices de rendimiento	ISE	0.002760
	IAE	0.02242
	$TVu(u)$	8.466
	$TVu(\omega)$	19.639
Tiempo empleado	$5\,min\,0.0\,seg$	

Camino:

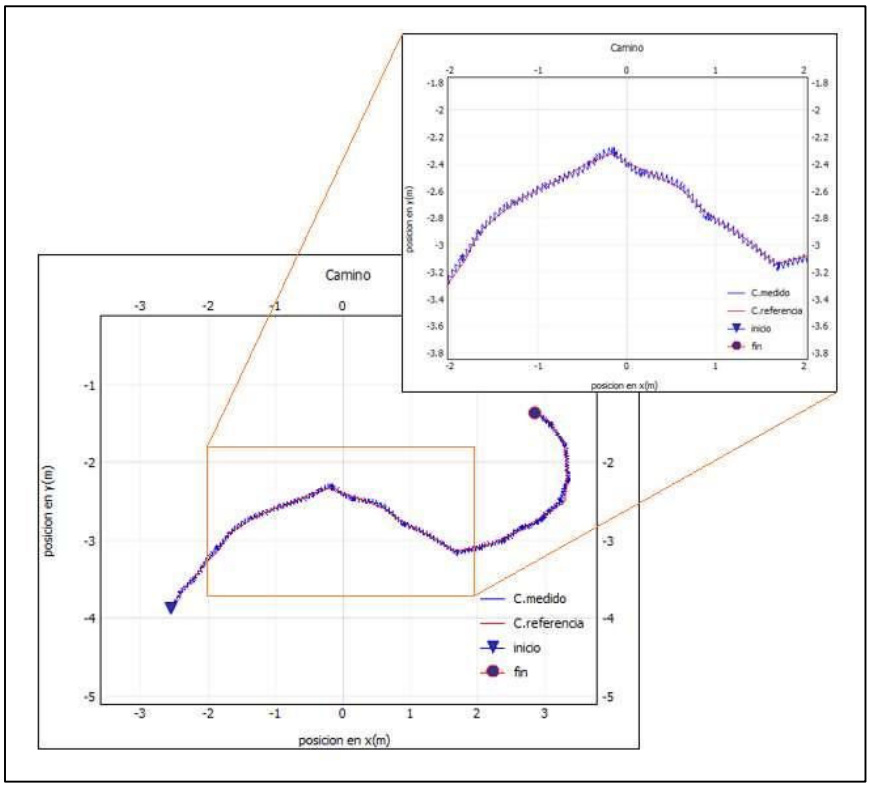


Figura 3. 17 Camino recorrido por el robot humanoide Nao con controlador basado en Lyapunov

En el caso del controlador basado en Lyapunov, el robot también es capaz de seguir el camino generado por el algoritmo RRT tal y como se observa en la Figura 3. 17.

Velocidad lineal:

Como ya se mencionó anteriormente, la acción de control de velocidad lineal en Lyapunov disminuirá a medida que se acerque a su objetivo y consecuentemente su velocidad lineal provocando así que aumente en algunos segundos su tiempo de llegada. En este caso, también se puede evidenciar en la Figura 3. 18 que la acción de control de velocidad lineal es igual a cero al presentarte errores de ángulo mayores a 15 grados, donde el robot momentáneamente disminuye su velocidad lineal con el objetivo de poder realizar giros pronunciados.

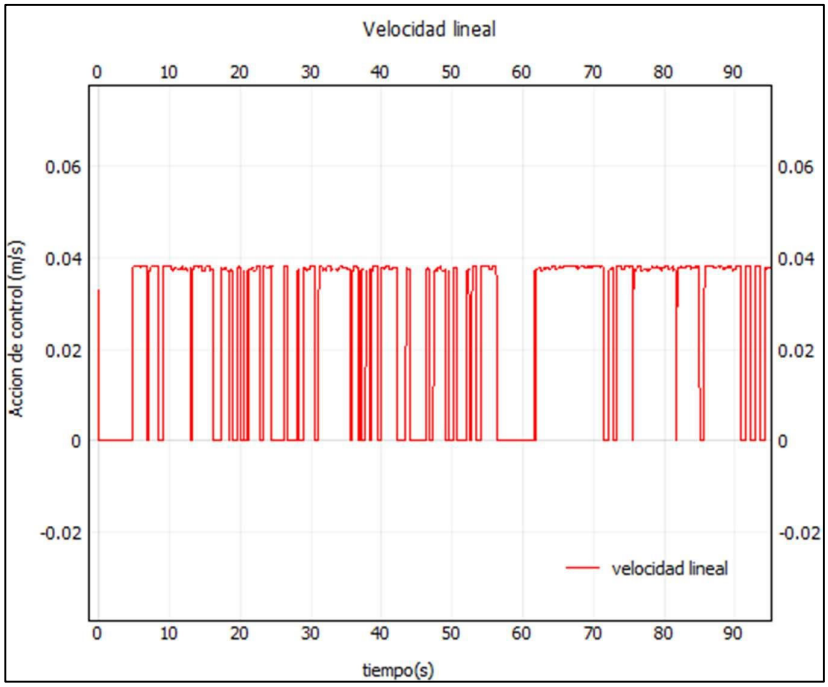


Figura 3. 18 Velocidad lineal del robot humanoide Nao con controlador basado en Lyapunov

Velocidad angular:

Al igual que en el primer y segundo escenario, se observa como la acción de control de velocidad angular aumenta cada vez que existe un nuevo cambio de referencia de ángulo, lo cual se refleja finalmente en que siga al camino sin ningún error, especialmente en las curvas pronunciadas como se puede notar en el acercamiento de la Figura 3. 17. Cada vez que se presente este tipo de curvas, la acción de control de velocidad angular será mayor tal que la velocidad lineal tendera a cero.

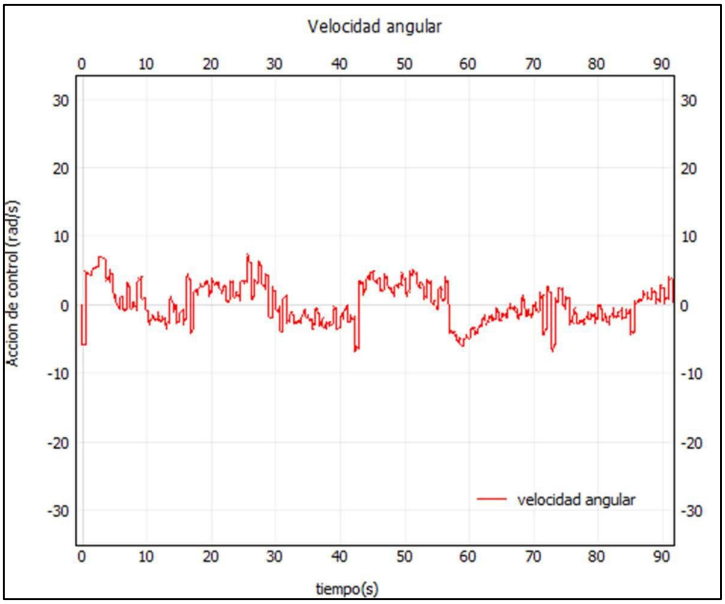


Figura 3. 19 Velocidad angular del robot humanoide Nao con controlador basado en Lyapunov

Ángulo:

Al observar la Figura 3. 20, se puede observar cómo el ángulo de referencia presenta cierta oscilación, cuya causa ya ha sido explicada en secciones anteriores. Sin embargo, se puede notar que la señal medida no se ve afectada por esta oscilación y logra seguir un promedio de la señal de referencia lo cual significa que el robot gira sin ningún tipo de interferencia.

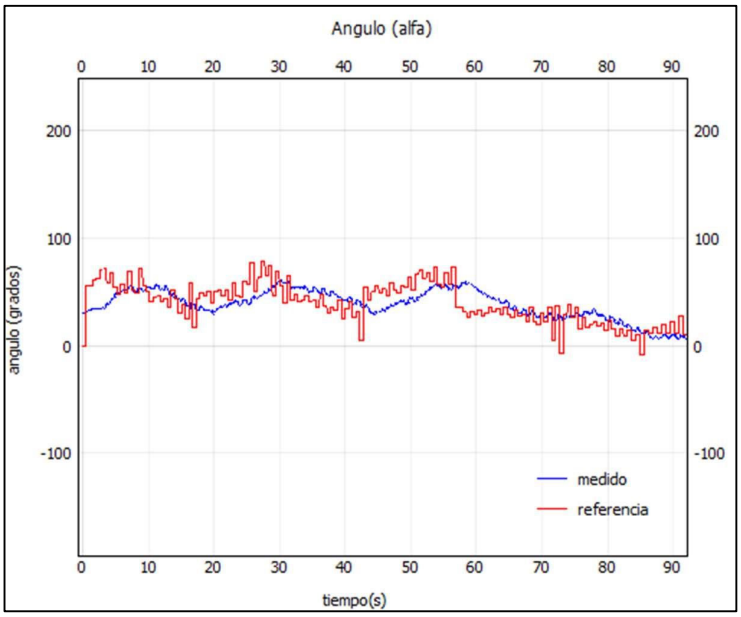


Figura 3. 20 Angulo del robot humanoide Nao con respecto al marco de referencia de CoppeliaSim

3.1.2.3 Comparación entre ambos controladores

Tabla 3. 6 Comparación de ISE, IAE y TVu de ambos controladores

	Controlador tipo PID	Controlador basado en Lyapunov
Tiempo empleado	$4\,\mathrm{min}\,16seg$	$5min$
$\boldsymbol{ISE}$	0.00189	0.00276
$\boldsymbol{IAE}$	0.01875	0.02242
$\boldsymbol{TVu(u)}$	7.001	8.466
$\boldsymbol{TVu(\omega)}$	16.580	19.639

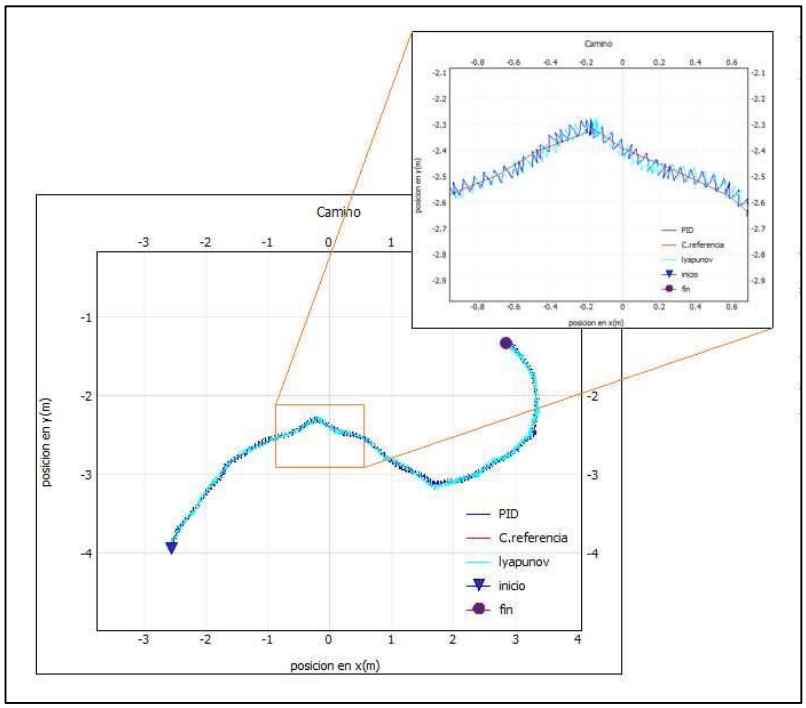


Figura 3. 21 Comparación entre el algoritmo de control PID y basado en Lyapunov en el camino efectuado por el robot Nao

Al observar ambos caminos se puede evidenciar que ambos cumplen con el objetivo de seguimiento de camino propuesto. Sin embargo, como se puede notar en la Tabla 3.6, el controlador PID posee un índice de rendimiento menor al del controlador basado en Lyapunov, además de que toma 44 segundos menos a que llegue su objetivo, lo cual resulta en un mejor desempeño general a velocidades del 75%.

3.1.3 PRUEBA ESCENARIO TRES Y VELOCIDAD LINEAL DE 100 % (50mm/s)

Como se puede notar en la Figura 3. 22, se ha colocado al robot Nao en la esquina inferior izquierda del escenario tres ya que desde este punto puede, con más facilidad, ingresar al centro del aula o desplazarse alrededor de los pupitres, permitiendo así poder acercarse a cualquiera de los estudiantes o hacia el docente. De igual manera al caso anterior, el algoritmo RRT genera suficientes ramificaciones dentro de su espacio de trabajo para el usuario pueda seleccionar la ubicación a donde quiere que el robot se movilice sin que choque con ningún obstáculo presente. En este ejemplo, se ha escogido que el robot se desplace hacia al centro de un grupo de pupitres de la izquierda del aula ya que, para ello, debe evadir ciertos obstáculos en su camino. En la parte derecha de la Figura 3. 22 se puede notar como el camino generado (línea roja) evade los obstáculos en el camino y permite al robot llegar a su objetivo. Este camino se divide en diferentes puntos los cuales van a ser entregados al robot para su seguimiento.

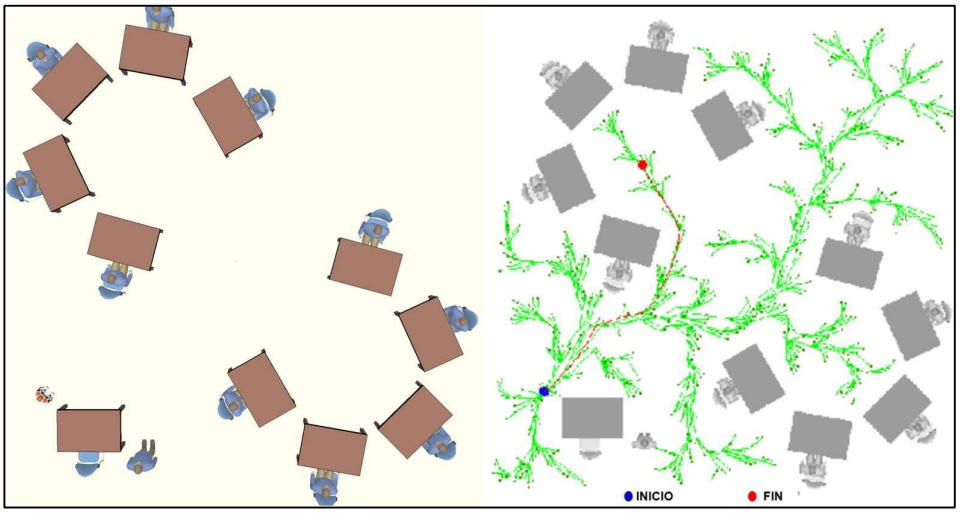


Figura 3. 22 Camino generado por algoritmo de planificador de caminos RRT - Escenario tres

3.1.3.1 Prueba con controlador PID

Tabla 3. 7 Datos controlador tipo PID - Escenario tres

Algoritmo de control	**Control tipo PID**	
Acción de control	$u(t) = kp * e(t)$	
	$\omega(t) = kp * (e(t) + T_d \frac{de(t)}{d(t)})$	
Constantes	K_p	0.4
	K_d	0.05
Índices de rendimiento	$TVu(u)$	4.255
	$TVu(\omega)$	8.257
Tiempo empleado	$2\,min\,26\,seg$	

Camino:

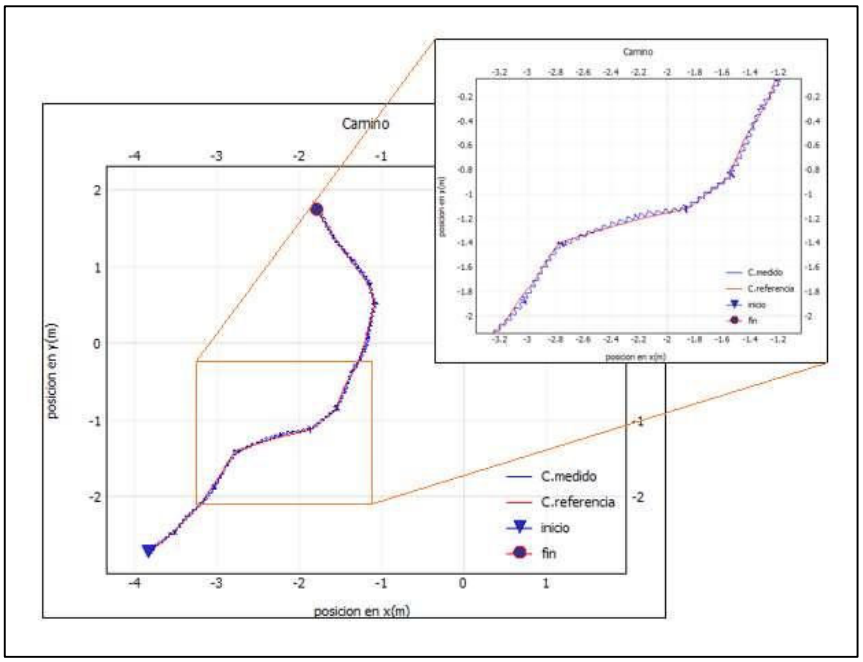


Figura 3. 23 Camino recorrido por el robot humanoide Nao con controlador PID

Como se puede observar en la Figura 3. 23, el robot es capaz de desplazarse a través del camino generado por el algoritmo RRT. En este caso, se puede notar que las oscilaciones inherentes del robot al caminar han disminuido ya que la velocidad del robot es mayor a la presentada en primer y segundo escenario. Esto quiere decir que el robot permanecerá menor intervalo de tiempo en un mismo lugar y por consecuencia llegando más rápido a su objetivo. Se puede notar igualmente que, ante algunas curvas pronunciadas, el robot se detiene durante un intervalo de tiempo para poder realizar el giro necesario para seguir el camino propuesto.

Velocidad lineal:

Como se observa en la Figura 3. 14 Figura 3. 24, el robot posee el mismo comportamiento que en los casos anteriores, es decir, que disminuye su velocidad lineal en puntos donde las curvas son pronunciadas. De igual manera, se puede evidenciar en la Figura 3. 25 que en los mismos puntos donde la acción de control de velocidad lineal es cero, la acción de control velocidad angular es mayor. Se toma como ejemplo el intervalo de tiempo 43 segundos. Se puede observar como en la gráfica de velocidad lineal, la señal medida va a cero, mientras que en la gráfica de velocidad angular, se genera una señal mayor al promedio, lo cual radica en un giro más rápido del robot.

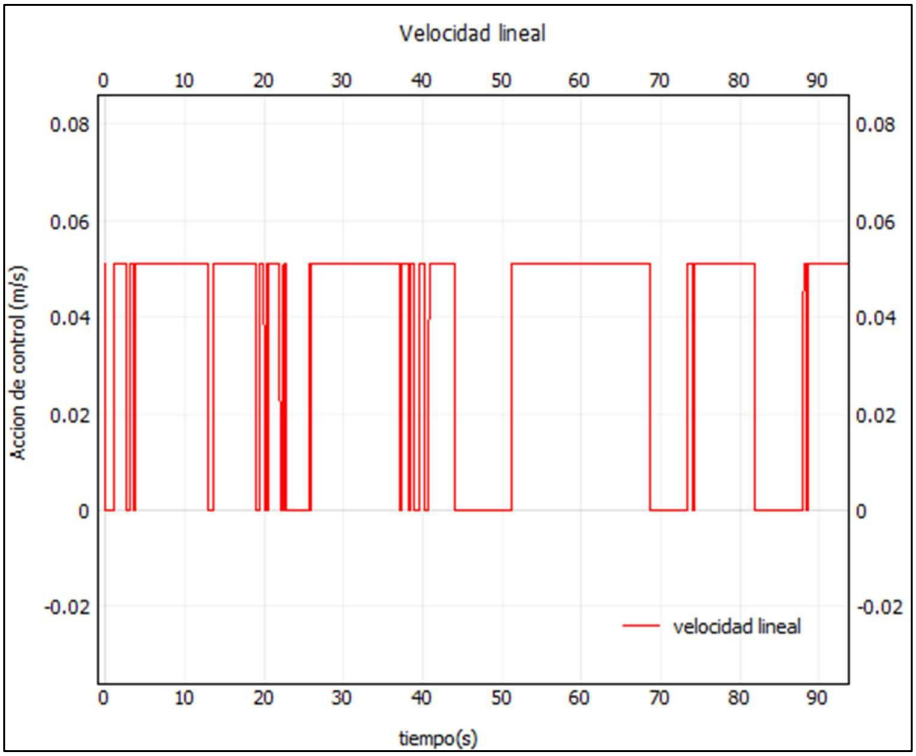


Figura 3. 24 Velocidad lineal del robot humanoide Nao con Controlador PID

Velocidad angular:

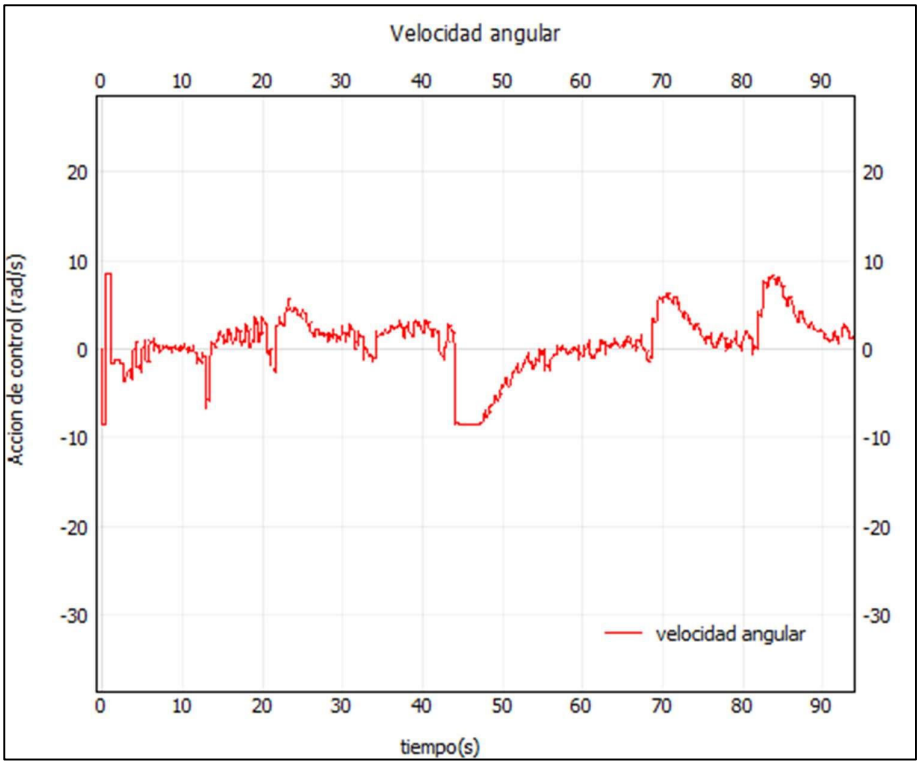


Figura 3. 25 Velocidad angular del robot humanoide Nao con Controlador PID

Ángulo:

Se observa en la Figura 3. 26 una fracción de la señal para evidenciar la tendencia del seguimiento de la señal de ángulo del robot con la señal de referencia generada.

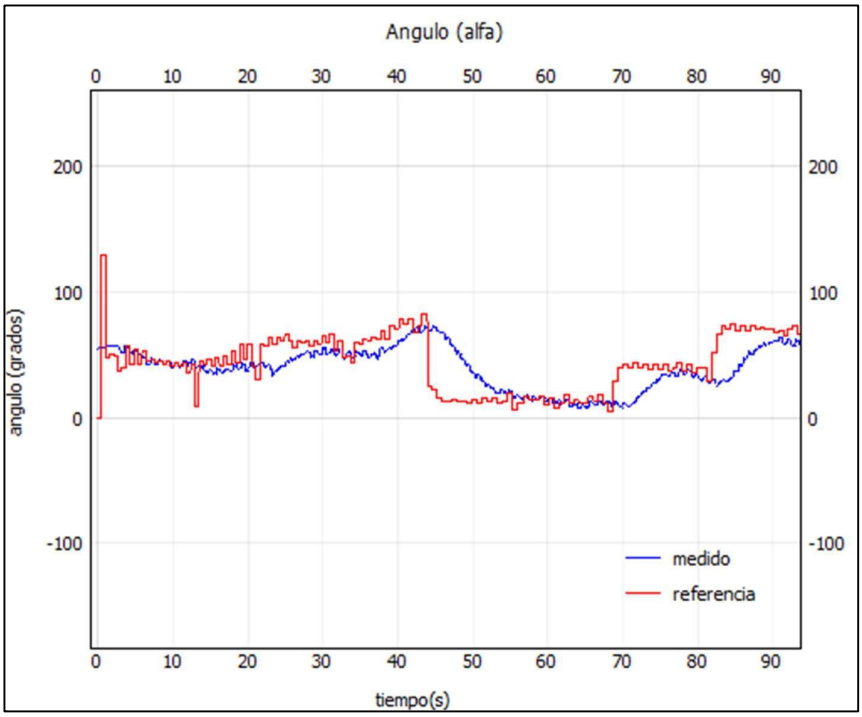


Figura 3. 26 Angulo del robot humanoide Nao con respecto al marco de referencia de CoppeliaSim

3.1.3.2 Prueba con controlador basado en Lyapunov

Tabla 3. 8 Datos controlador basado en Lyapunov - Escenario tres

Algoritmo de control	**Control basado en Lyapunov**	
Acción de control	$u = K_u \tanh(\rho)\cos(\alpha)$	
	$\omega = K_\omega \alpha + K_u \frac{\tanh(\rho)}{\rho}\sin(\alpha)\cos(\alpha)$	
Constantes	K_u	0.0512
	K_w	0.4
Índices de rendimiento	ISE	0.00089
	IAE	0.01018
	$TVu(u)$	5.773
	$TVu(\omega)$	9.627
Tiempo empleado	$3\,min\,20\,seg$	

Camino:

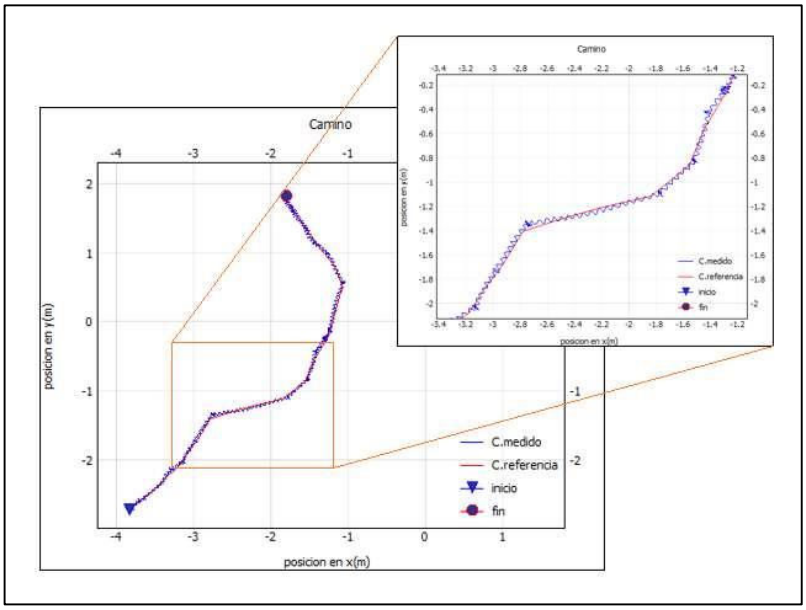


Figura 3. 27 Camino recorrido por el robot humanoide Nao con controlador basado en Lyapunov

En el caso del controlador basado en Lyapunov, el robot también es capaz de seguir el camino generado por el algoritmo RRT tal y como se observa en la Figura 3. 27. Al igual que en el controlador PID, se puede notar un decremento en las oscilaciones inherentes del robot al caminar ya que permanece menor tiempo en cada punto ya que su velocidad es mayor comparando al caso del primer y segundo escenario.

Velocidad lineal:

En este caso, la acción de control de velocidad lineal a diferencia del controlador PID, es directamente proporcional a la distancia del robot a su objetivo; es decir a mayor distancia de su objetivo, mayor velocidad lineal. Se puede observar en la Figura 3. 28, que la acción de control de velocidad lineal del robot tiende a cero durante ciertos intervalos de tiempos ya que durante esos momentos el robot se ve expuesto a errores de ángulos mayores a 15 grados y se desea que el robot se oriente al siguiente punto objetivo correctamente antes de que siga avanzando linealmente. En la Figura 3. 29 se puede claramente notar como la acción de control de velocidad angular aumenta cada vez que existe un cambio de referencia, dado por un nuevo punto dado por el algoritmo RRT*FN.

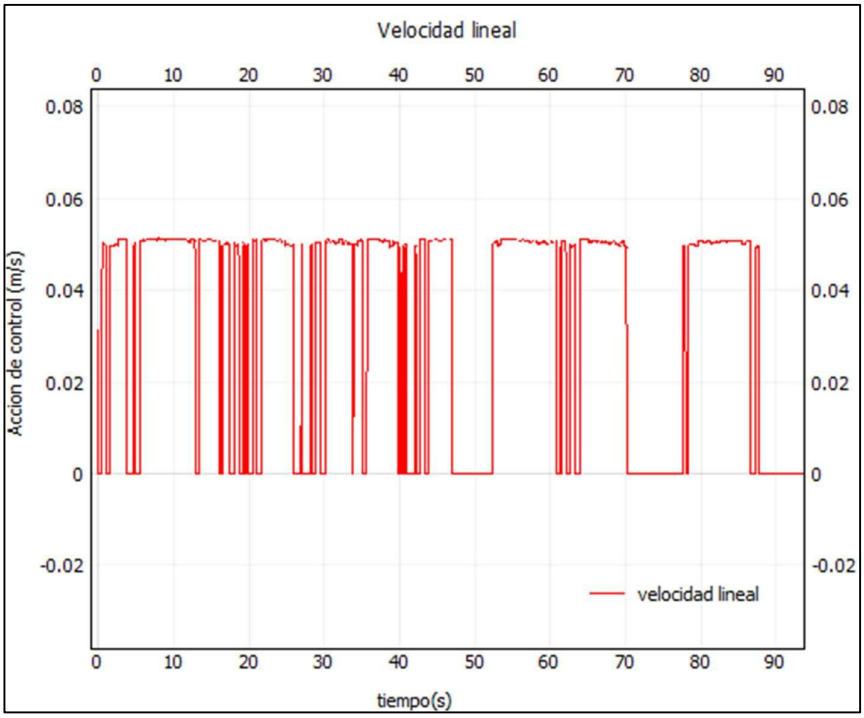


Figura 3. 28 Velocidad lineal del robot humanoide Nao con controlador basado en Lyapunov

Velocidad angular:

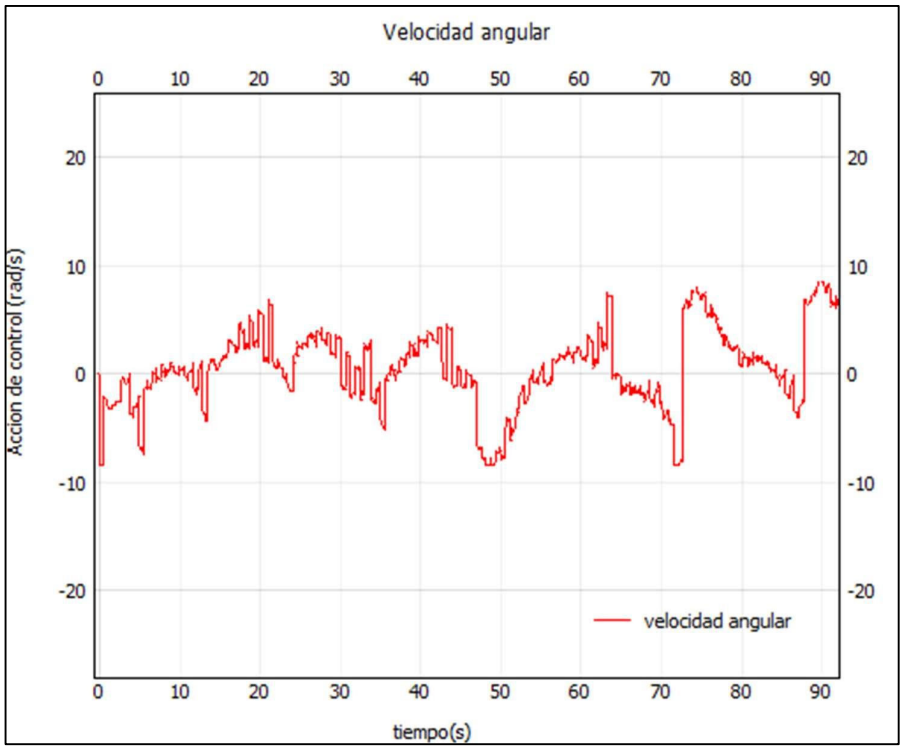


Figura 3. 29 Velocidad angular del robot humanoide Nao con controlador basado en Lyapunov

Ángulo:

Finalmente, en la Figura 3. 30 se puede observar la tendencia de seguimiento del ángulo del robot a la señal de referencia

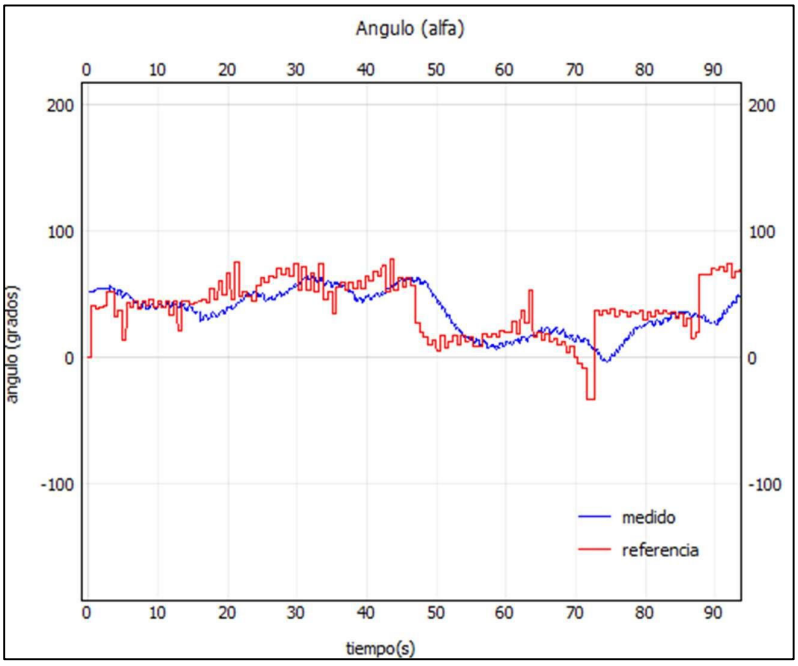


Figura 3. 30 Ángulo del robot humanoide Nao con respecto al marco de referencia de CoppeliaSim

3.1.3.3 Comparación entre ambos controladores

Tabla 3. 9 Comparación de ISE, IAE y TVu de ambos controladores

	Controlador tipo PID	**Controlador basado en Lyapunov**
Tiempo empleado	$2min\ 26seg$	$3min\ 20seg$
$\boldsymbol{ISE}$	0.002120	0.000899
$\boldsymbol{IAE}$	0.007894	0.010188
$\boldsymbol{TVu(u)}$	4.255	5.773
$\boldsymbol{TVu(\omega)}$	8.257	9.627

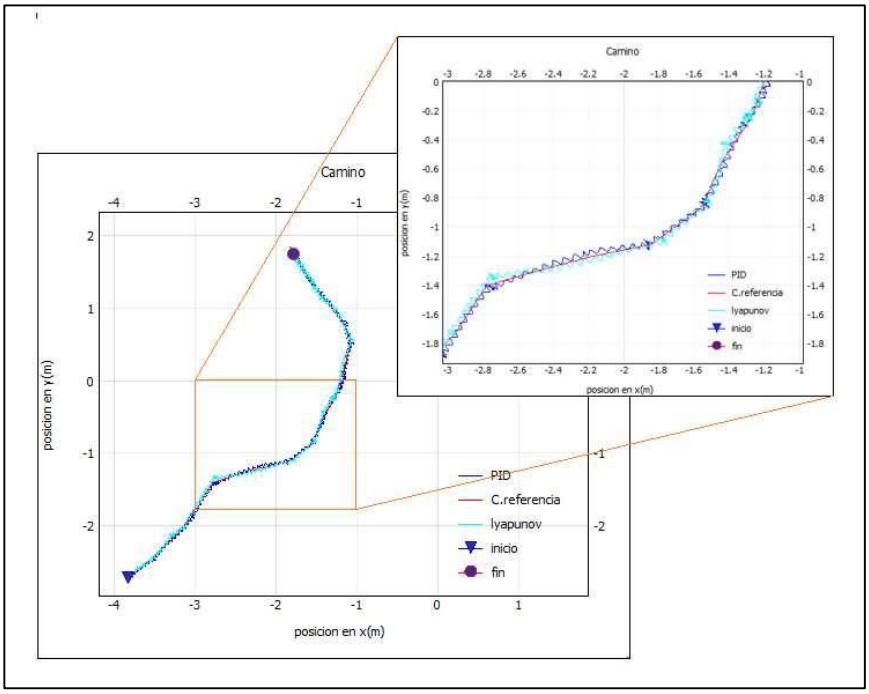


Figura 3. 31 Comparación entre el algoritmo de control PID y basado en Lyapunov en el camino efectuado por el robot Nao

Al observar ambos caminos en la Figura 3. 31, se puede evidenciar que cumplen con el objetivo de seguimiento de camino propuesto. Sin embargo, como se puede notar en la Tabla 3.9, el controlador tipo PID posee un índice de rendimiento menor al del controlador basado en Lyapunov, además de que toma aproximadamente un minuto menos a que llegue su objetivo, lo cual resulta en un mejor desempeño general a velocidades del 100%.

4. CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

- La evidencia presentada en al capítulo 3 demuestra que el robot Nao es capaz de realizar diferentes seguimientos de camino generado por el algoritmo RRT en diferentes escenarios. Sin embargo, a velocidades muy bajas, le toma al robot demasiado tiempo en llegar a su objetivo mientras que a velocidades cercanas al 100%, el robot presenta ligeras desviaciones del camino, especialmente ante curvas pronunciadas. Por tales razones, se ha escogido como velocidad predeterminada del robot en un 75% de su velocidad máxima.
- A pesar de que con ambos controladores el robot fue capaz de realizar el camino propuesto por el algoritmo RRT, se pudo evidenciar mediante los diferentes índices de rendimiento empleados que, al emplear el controlador P en velocidad lineal y el controlador PD en velocidad angular, el robot llegaba a su objetivo de manera más eficiente.
- La implementación del algoritmo de planificación de caminos RRT usado en el robot humanoide, presento un desempeño satisfactorio debido a que no existió la presencia de colisión con los obstáculos estáticos existentes en su entorno. Sin embargo, en la vida real, gran número de obstáculos tienden a estar en movimiento por lo que se propone para futuros trabajos tomar en cuenta un algoritmo que permita al sistema evadir obstáculos móviles.
- Para poder controlar el desplazamiento del robot Nao, se debe monitorear tanto su orientación como su posición cada tiempo de muestreo y, de existir algún error, se los corrige mediante las acciones de control tanto de velocidad lineal, como de velocidad angular.
- Con el fin de que el robot pueda movilizarse eficientemente, se ha utilizado saturadores tanto en valores máximo como en valores mínimos en los controladores de la velocidad lineal y la velocidad angular del robot Nao.
- En cada uno de los resultados de camino del robot, se puede observar como este presenta una oscilación constante al seguir la referencia de camino. Esta oscilación se debe a que el punto de medida de la ubicación de Nao se encuentra ubicada en su pecho y, al momento de caminar, tiende a variar esta ubicación de manera constante. Este comportamiento es común en cualquier robot bípedo o incluso en una persona cuando camina en su entorno.
- En las gráficas de velocidad lineal y angular se puede apreciar oscilaciones en la señal de referencia causadas por la propia odometría del robot humanoide. A medida que Nao se acerque a su punto objetivo, la diferencia entre su posición actual y su posición

deseada tenderá a cero, causando así que al calcular el ángulo de referencia, esta señal oscile.

- Al realizar las diferentes pruebas de seguimiento de camino en un ambiente simulado y controlado, no se ha considerado diferentes factores físicos o externos del robot que puedan influir en su desempeño tales como el nivel de batería de Nao, el calentamiento de sus diferentes motores por uso continuo o incluso la contextura del piso en el que se moviliza.
- La programación del proyecto se lo ha realizado en lenguaje Python debido a que presenta múltiples ventajas con respecto a otros lenguajes de programación. Cuando se trabaja con Python no solamente se encuentra numerosas fuentes de información o tutoriales de programas ya implementados en Nao, sino que también existe numerosas librerías ya escritas que pueden ser completo del programa original. Un claro ejemplo de esto se observa en las librerías usadas para la implementación de la interfaz gráfica.

4.2. RECOMENDACIONES

- Antes de utilizar el robot Nao, se recomienda revisar a detalle las características físicas del robot, así como los lenguajes de programación con los que se puede trabajar, para lograr tener un mejor desenvolvimiento al trabajar con el robot, y adaptaciones con antiguos proyectos.
- Se recomienda que, antes de utilizar la interfaz desarrollada para el robot Nao que trabaja simultáneamente con CoppeliaSim Player y Python, se revise las características mínimas que debe tener el computador, pues caso contrario puede presentarse complicaciones al efectuar la comunicación con el robot.
- Una de las tareas más importantes en el desplazamiento de un robot humanoide, es la evasión de obstáculos que pueda presentarse en su camino, por lo que se debe tener en cuenta las dimensiones del robot, así como tener en consideración el ambiente en el que se va a desenvolver.
- Se recomienda usar los parámetros prestablecidos en cada uno de los controladores del robot ya que estos han sido previamente calibrados tomando en cuenta el índice de rendimiento IAE. Si se desea cambiar alguno de estos valores, se recomienda leer con anterioridad el trabajo realizado para tener un entendimiento básico de cómo afecta el cambiar algún parámetro.
- Se recomienda usar la velocidad lineal del robot prestablecida en la interfaz gráfica, ya que, mediante las diferentes pruebas realizadas, se determinó que esta velocidad es con la que se obtuvo la mejor relación entre precisión de seguimiento de camino y rapidez con la que se llega al objetivo.

- El usuario, mediante la interfaz (al correr el simulador junto con el planificador de caminos RRT) puede ir creando el camino por donde específicamente se quiere que el robot camine, dando puntos sucesivos cercanos al robot, mientras se encuentre abierta la ventana de selección de punto de llegada.
- Como trabajos futuros de investigación se podría hacer uso de los sensores de proximidad que posee el robot Nao, así como los sensores de fuerza contenidos en los pies, con el fin de que el robot Nao pueda tener la opción de evadir obstáculos poniendo en uso los sensores ya mencionados.
- Se recomienda investigar otros algoritmos de planificación de caminos que puedan ajustarse a la evasión de obstáculos en movimiento, tomando en cuenta también la carga computacional que puede llevar a exigir de un sistema.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] R. Reddy, “Robotics and Intelligent Systems in Support of Society”, *IEEE Intell. Syst.*, vol. 21, núm. 3, pp. 24–31, may 2006, doi: 10.1109/MIS.2006.57.

[2] F. Tanaka, A. Cicourel, y J. R. Movellan, “Socialization between toddlers and robots at an early childhood education center”, *Proc. Natl. Acad. Sci.*, vol. 104, núm. 46, pp. 17954–17958, nov. 2007, doi: 10.1073/pnas.0707769104.

[3] S. Kajita, H. Hirukawa, K. Harada, y K. Yokoi, “Biped Walking”, en *Introduction to Humanoid Robotics*, S. Kajita, H. Hirukawa, K. Harada, y K. Yokoi, Eds. Berlin, Heidelberg: Springer, 2014, pp. 105–158.

[4] A. Ude, C. G. Atkeson, y M. Riley, “Programming full-body movements for humanoid robots by observation”, *Robot. Auton. Syst.*, vol. 47, núm. 2, pp. 93–108, jun. 2004, doi: 10.1016/j.robot.2004.03.004.

[5] A. Hornung, K. M. Wurm, y M. Bennewitz, “Humanoid robot localization in complex indoor environments”, en *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Taipei, oct. 2010, pp. 1690–1695, doi: 10.1109/IROS.2010.5649751.

[6] S. E. M. Rozas, “Modelado y simulación de robots terrestres para la inspección del alcantarillado”, Proyecto Fin de Máster, Escuela Técnica Superior de Ingeniería Universidad de Sevilla, Sevilla, 2018.

[7] L. Poubel, “From Robot Simulation to the Real World”, *InfoQ*, jun. 28, 2019. https://www.infoq.com/presentations/robot-simulation-real-world/ (consultado oct. 20, 2020).

[8] IEEE, “Nao - ROBOTS: Your Guide to the World of Robotics”, *ROBOTS*, 2018. https://robots.ieee.org/robots/nao/ (consultado sep. 05, 2020).

[9] B. Maisonnier, “NAO - Documentation — Aldebaran 2.8.6.23e documentation”, *Softbank Robotics Documentation*. http://doc.aldebaran.com/2-8/home_nao.html (consultado sep. 05, 2020).

[10] C. Wang, “BEHAVIOUR DESIGN OF NAO HUMANOID ROBOT: Playing Tic Tac Toe Game”, Tesis de pregrado, VAASAN AMMATTIKORKEAKOULU UNIVERISTY OF APPLIED SCIENCES, Korsholm, 2015.

[11] N. Kofinas, E. Orfanoudakis, y M. G. Lagoudakis, “Complete Analytical Forward and Inverse Kinematics for the NAO Humanoid Robot”, *J. Intell. Robot. Syst.*, vol. 77, núm. 2, pp. 251–264, feb. 2015, doi: 10.1007/s10846-013-0015-4.

[12] M. A. Alvarez, “Qué es Python”, *DesarrolloWeb*, nov. 19, 2003. https://desarrolloweb.com/articulos/1325.php (consultado sep. 07, 2020).

[13] L. Valdellon, “What is an SDK? Everything You Need to Know”, *CleverTap*, jul. 30, 2020. https://clevertap.com/blog/what-is-an-sdk/ (consultado sep. 08, 2020).

[14] CoppeliaSim, “CoppeliaSim User Manual”, *Coppeliarobotics*. https://www.coppeliarobotics.com/helpFiles/ (consultado ago. 19, 2020).

[15] R. Lersalimschy y W. Celes, “Manual de Referencia de Lua 5.1”, 2008. https://www.lua.org/manual/5.1/es/manual.html (consultado oct. 20, 2020).

[16] H. Secchi, *Una introducción a los robots móviles*. Buenos Aires: AADECA: Facultad de Ingeniería UNLP, 2008.

[17] L. Capito y P. Proaño, “SEGUIMIENTO DE TRAYECTORIAS MEDIANTE CUATRO TÉCNICAS DE CONTROL UTILIZANDO UNA PLATAFORMA ROBÓTICA PIONEER 3DX Y EL SISTEMA OPERATIVO ROBÓTICO ROS”, Tesis de pregrado, Escuela Politécnia Nacional, Quito, 2015.

[18] J. Heredia y S. Mena, “IMPLEMENTACIÓN DE UN MANIPULADOR MÓVIL PARA DESARROLAR TAREAS DE SEGUIMIENTO DE TRAYECTORIAS CON UN CONTROLADOR PID”, Tesis de pregrado, Escuela Politécnia Nacional, Quito, 2017.

[19] L. Payne, “The Modern Industrial Workhorse: PID Controllers”, jul. 01, 2014. https://www.techbriefs.com/component/content/article/tb/pub/features/articles/20013 (consultado oct. 20, 2020).

[20] M. Garcıa y A. Barreiro, “An´alisis de la Estabilidad segu´n Lyapunov de un Control Borroso en Tiempo Discreto”, p. 6.

[21] K. Ogata, *Ingeniería de Control Moderna*, 5ta Edición. Madrid: Pearson Education, 2010.

[22] S. G. Tzafestas, “5 - Mobile Robot Control I: The Lyapunov-Based Method”, en *Introduction to Mobile Robot Control*, S. G. Tzafestas, Ed. Oxford: Elsevier, 2014, pp. 137–183.

[23] J. Villacrés y M. Viscaíno, “DISEÑO E IMPLANTACIÓN DE TRES ESQUEMAS DE CONTROL: PID, LQR Y MODOS DESLIZANTES PARA LA ESTABILIZACIÓN DEL PÉNDULO INVERTIDO PARA ESTABILIZACIÓN DEL PÉNDULO INVERTIDO SOBRE DOS RUEDAS DEL LEGO MIDSTORMS CON APLICACIÓN DE UN PLANIFICAR DE RUTAS MEDIANTE EL ALGORITMO RRT”, Tesis de pregrado, Escuela Politécnia Nacional, Quito, 2016.

[24] S. LaValle, “Rapidly-Exploring Random Trees: A New Tool for pathing Planning”, Illinois, 1998.

[25] D. López, Gómez Fernando, F. Cuesta, y A. Ollero, “Planificación de Trayectorias con el Algoritmo RRT. Aplicación a Robots No Holónomos”, *ResearchGate*, 2008. https://www.researchgate.net/publication/28141977_Planificacion_de_Trayectorias_con_el_Algoritmo_RRT_Aplicacion_a_Robots_No_Holonomos (consultado sep. 20, 2020).

[26] M. Shinners, *Modern Control System Therory and Design*. John Wiley & Sons, 1998.

[27] V. Orosco, “Desarrollo de una herramienta computacional para la sintonización de parámetros de controladores PID y SMC pra el seguimiento de trayectorias de una cuadricóptero basad en algoritmos genéticos.”, Escuela Politécnia Nacional, Quito, 2018.

[28] V. Alfaro y R. Vilanova, “Sintonización de los controladores PID de 2GdL: desempeño, robustez y fragilidad”, ene. 2010.

[29] Qt User Manual, “Qt Creator Manual”. https://doc.qt.io/qtcreator/index.html (consultado sep. 20, 2020).

[30] Qt User Manual, “Qt Designer Manual”. https://doc.qt.io/qt-5/qtdesigner-manual.html (consultado sep. 20, 2020).

[31] G. Andaluz, “Modelación, identifiación y control de robots móviles.”, Escuela Politécnia Nacional, Quito, 2011.

[32] Yale University, “Classroom Seating Arrangements | Poorvu Center for Teaching and Learning”, *Classroom Seating Arrangements*, 2018. https://poorvucenter.yale.edu/ClassroomSeatingArrangements?fbclid=IwAR0F8n0zK_96Wnixh4j8pcFtVcyMptTk9kjDqlT8rq70kDAACxeIlcBZpfw (consultado dic. 30, 2020).

[33] M. Garcıa y A. Barreiro, “Análisis de la Estabilidad según Lyapunov de un Control Borroso en Tiempo Discreto”, p. 6.

[34] B. Cardenas, “An Overview of the Class of Rapidly-Exploring Random Trees”, B.Sc Tesis, Utrecht University, Países Bajos, 2018.

ANEXO A

A.1. INTRODUCCION

El computador necesita una lista de requerimientos mínimos para que el proyecto desarrollado pueda funcionar de manera óptima, es decir, que sea capaz de no solamente de establecer una comunicación entre el módulo de NAOqi, CoppeliaSim y Python 2.7, sino también, de realizar las acciones de control en Python 2.7. Los requerimientos mínimos del computador, determinados por medio de pruebas experimentales, son los siguientes:

- Procesador: Intel Core i7 8va generación (o cualquier procesador similar), mínimo 5 núcleos, a 2.6 GHz.
- Memoria RAM: 16Gb
- Sistema Operativo: Windows 10 o superior.
- Tarjeta gráfica NVIDIA GTX 1650Ti

A.2. INSTALACIONES PREVIAS

Los programas mencionados en la introducción deben ser correctamente instalados previo al uso del proyecto desarrollado, por lo que, a continuación, se detalla los pasos a seguir:

A.2.1. Instalación de Anaconda 2.7

Primeramente, se procede a instalar Python 2.7 contenido en Anaconda.

Para instalar anaconda 2.7 se debe seguir los siguientes pasos:

1. En la carpeta Instaladores (carpeta que se adjunta en el CD del proyecto técnico), se encuentra el ejecutable Anaconda 2. Hacer doble clic para ejecutar el instalador.
2. Dejar la dirección sugerida por el instalador: C:\Users\USER\Anaconda2 y hacer clic en **Next,** como se muestra en la Figura A.1.
   USER: nombre de la maquina (ej.: JuanOrtega, PC2020, etc.)

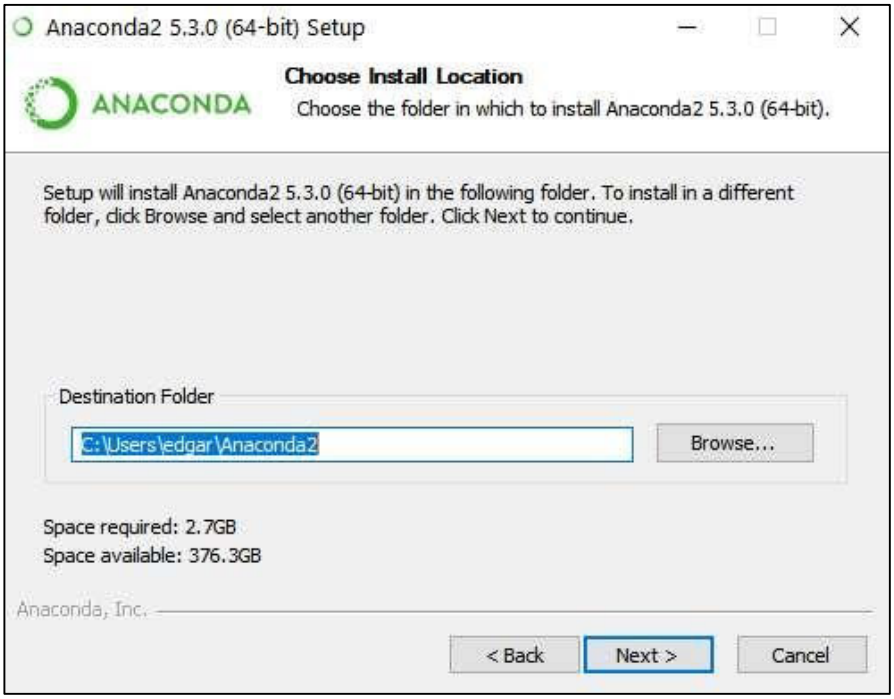


Figura A.1 Ventana de instalación de Anaconda.

3. Seleccionar la opción: **Register Anaconda as my default Python 2.7** y hacer clic en **Next**, como se muestra en la Figura A.2.

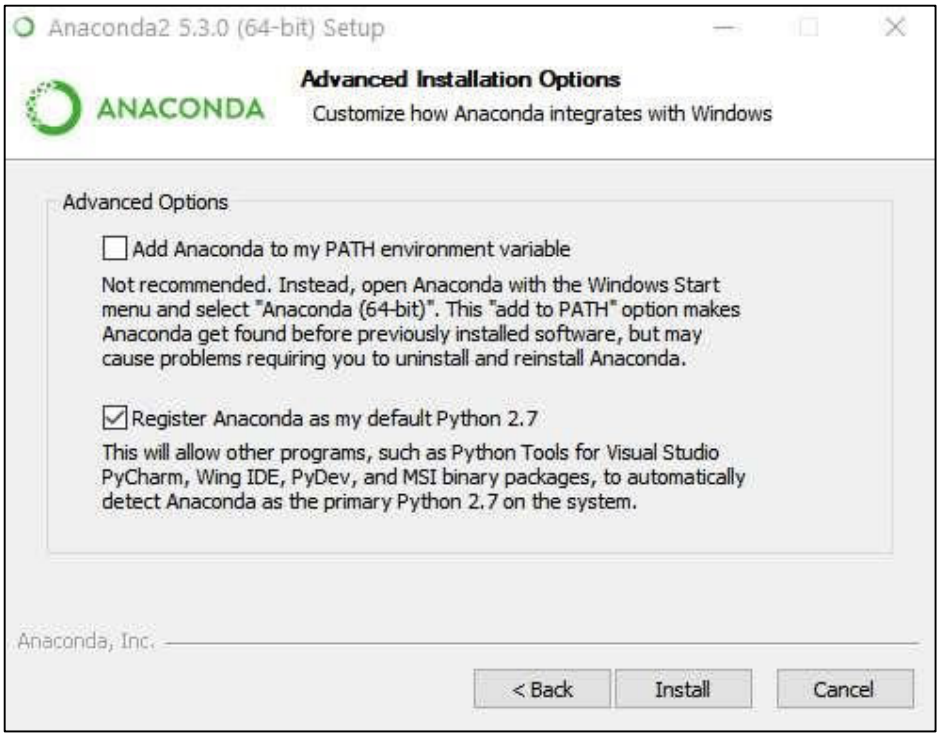


Figura A.2 Ventana de instalación de Anaconda.

4. Dar clic en **Install** y esperar hasta que el programa se haya instalado total por completo. Este proceso puede durar de 5 a 10 minutos, dependiendo de la velocidad del computador.

A.2.2. Instalación de Python SDK

Para establecer una comunicación entre NAOqi y Python 2.7, es necesario instalar previamente el Python SDK. Su instalación se lo hace a través del siguiente procedimiento:

1. En la carpeta Instaladores abrir la carpeta Python SDK.
2. Copiar todos los archivos contenidos en la carpeta.
3. Abrir la carpeta Anaconda 2, ubicado en la dirección C:\Users\USER\Anaconda2.
4. Pegar todos los archivos copiados previamente, de la carpeta Python SDK.
5. Abrir, **Editar las variables de entorno del sistema**, que se puede encontrar desde la barra de tareas de Windows 10.
6. En la pestaña **Avanzado**, hacer clic en **Variables de entorno**, como se muestra en la Figura A.3.

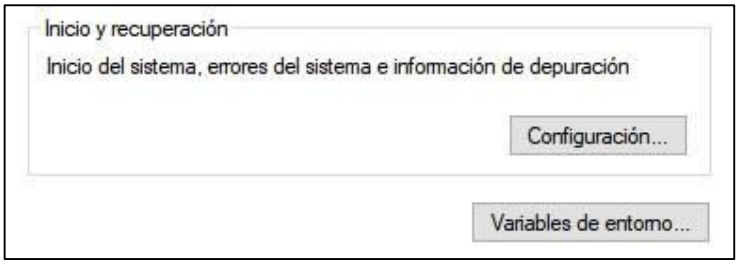


Figura A.3 Ventana principal de Propiedades del sistema/Opciones avanzadas.

7. En la ventana **Variables de entorno**, agregue una nueva variable de usuario: PYTHONPATH y configúrelo en: C:\Users\USER\Anaconda2\lib, como se muestra en la Figura A.4.

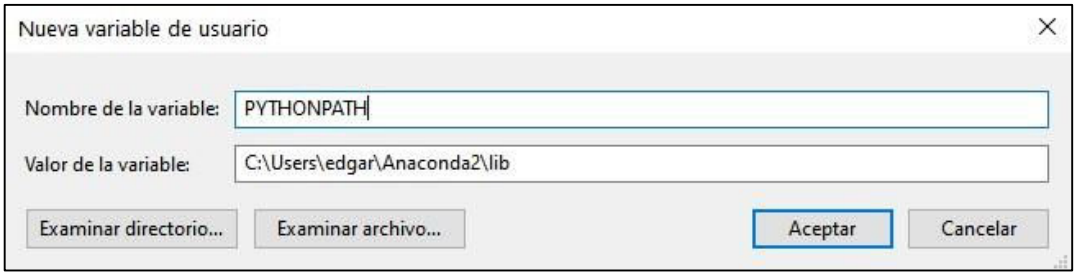


Figura A.4 Ventana Variables de entorno/nueva.

A.2.3. Instalación de cada una las librerías

1. Abrir anaconda Prompt.
2. Digitar cada uno de los siguientes comandos detallados a continuación, y esperar a su instalación correspondiente.

- Matplotlib: es una librería que se usa para la generación de gráficos, además de gráficos con visualización interactiva. Para instalarlo se debe ingresar los siguientes comandos en la terminal de Anaconda Prompt.
  - conda update --all
  - pip install numpy
  - pip install –upgrade numpy
  - conda install matplotlib
- Pygame: se encuentra conformado por módulos realizados en lenguaje de Python, el cual permite la creación de gráficos interactivos. Para instalarlo se debe ingresar el siguiente comando en la terminal de Anaconda Prompt.
  - pip install --user pygame
- La instalación de Qt creator, entorno donde se realiza la creación de la interfaz, que permita la interacción con la simulación del Robot NAO. Para instalarlo se debe ingresar los siguientes comandos en la terminal de Anaconda Prompt.
  - conda update --all
  - conda install qt
  - conda install pyqt
  - conda install pyqt5
- Pyqtgraph: es una biblioteca de interfaz de usuario, que proporciona los requerimientos para aplicaciones científicas y de ingeniería, con gráficos de alto rendimiento junto con el uso de numpy para la interpretación de datos

numéricos. Para instalarlo se debe ingresar el siguiente comando en la terminal de Anaconda Prompt.

  - ❖ pip install --user pyqtgraph

- OpenCV-Python: es una biblioteca diseñada para resolver problemas de visión por computador y admitir algoritmos relacionados con visión por computador.
  - ❖ pip install opencv-python==4.2.0.32
- PIL, la biblioteca de imagenes, se encuentra diseñada para el acceso a datos almacenados en algunos formatos de pixeles básicos, lo que proporciona una base sólida como herramienta general del Procesamiento de imágenes.
  - ❖ conda install Pil

A.2.4. Instalación de Choregraphe Suite

Para simular el Robot NAO desde CoppeliaSim, es necesario la ejecución previa del archivo *naoqi-bin*, además de la selección de la versión del Robot, con el que se quiere trabajar. Para cumplir ambos requerimientos, es necesario la instalación de Choregraphe Suite, a través del siguiente procedimiento:

1. En la carpeta Instaladores encuentra el ejecutable con el nombre de **Choregraphe Suite**, hacer doble clic para proseguir con la instalación.
2. Seleccionar la Opción **Quick**, y seguir con la instalación como se muestra en la Figura A.5.

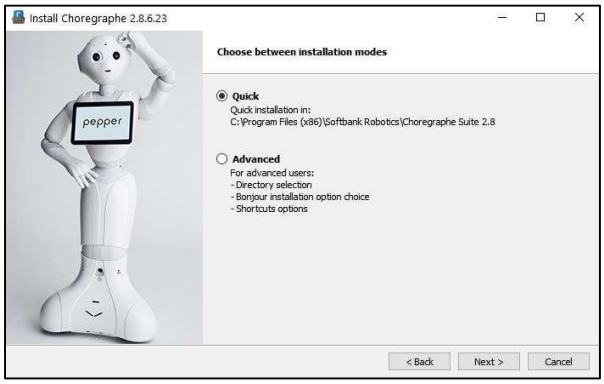


Figura A.5 Ventana de instalación de Choregraphe Suite.

3. Una vez finalizada la instalación, ejecutar Choregraphe Suite. De existir un mensaje de error, al final de la sección A.2.4 se encuentra como solucionar el posible problema.

A.2.4.1. Configuracion del Choregraphe

1. Ejecutar Choregraphe Suite
2. Hacer clic en la pestaña **Edit\Preferences**, ubicada en la parte superior izquierda del programa.
3. Hacer clic en **Virtual Robot**, y luego en **Robot model**, para así poder seleccionar **NAO H25 (V6)**, como se muestra en la Figura A.6.
4. Hacer clic en **OK.**

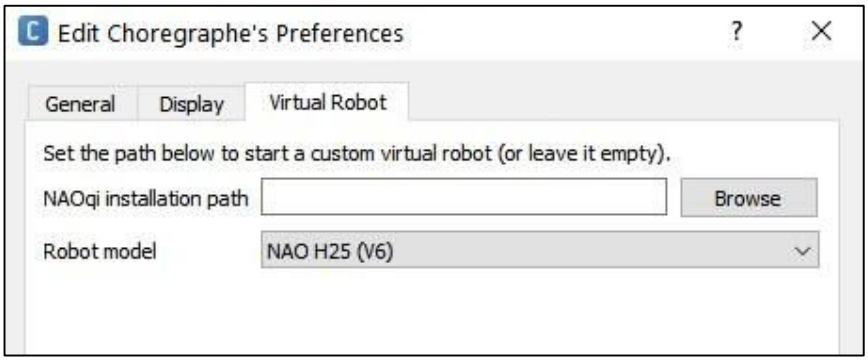


Figura A.6 Ventana de edición de preferencias de Choregraphe.

4. Hacer clic en la pestaña **Connection** y seleccionar la opción **Connect to…**
5. Seleccionar **Use Fixed port** (9559), como se muestra en la Figura A.7.

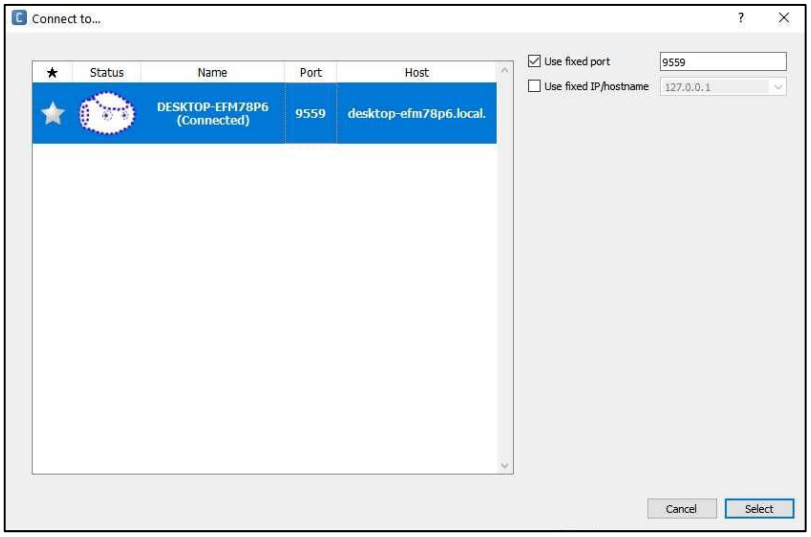


Figura A.7 Ventana de Connect to… de Choregraphe.

6. Finalizar haciendo clic en **Select** y cerrar Choregraphe.

Nota: En caso de que no aparezca la alguna opción de selección como se muestra en la Figura A.7, se puede seleccionar **Use fixed IP/hostname,** ubicado en la parte superior derecha de la ventana.

Solución ante posible problema al ejecutar Choregraphe Suite: Hay dos archivos que es posible que se deba agregar manualmente a la carpeta del sistema para que Choregraphe vuelva a estar operativo.

1. En la carpeta **Solución** existen dos carpetas: windows-system32 y windows-syswow64

- Copiar los archivos windows-system32y pegarlos en la carpeta C: \ Windows \ System32.
- Copiar los archivos windows-system64 y pegarlos en la carpeta C: \ Windows \ SysWOW64

Nota: C: \ Windows \ SysWOW64. (Si esa carpeta no existe, probablemente no tenga Windows de 64 bits).

A.2.5. Instalación de CoppeliaSim Edu

La instalación de CoppeliaSim, será el entorno donde se simulará el Robot NAO. La instalación de CoppeliaSim Edu se lo hace a través del siguiente procedimiento.

1. En la carpeta Instaladores encuentra el ejecutable con el nombre de CoppeliaSim Edu. Hacer doble clic sobre el instalador para proseguir con la instalación.
2. Hacer clic en **Next**, y finalizar instalación.
3. Una vez finalizado la instalación, abrir la carpeta instaladores, y copiar la carpeta Aulas en la dirección: C: \

A.2.6 Instalación de PyCharm Community

La instalación de PyCharm Community, será el entorno donde se desarrolla la programación de los algoritmos de control del Robot NAO. La instalación de PyCharm Community se lo hace a través del siguiente procedimiento:

1. En la carpeta Instaladores encuentra el ejecutable con el nombre de PyCharm Community. Hacer doble clic sobre el ejecutable para proseguir con la instalación.
2. Hacer clic en **Next**, sin modificar la dirección de destino, como se muestra en la Figura A.8.

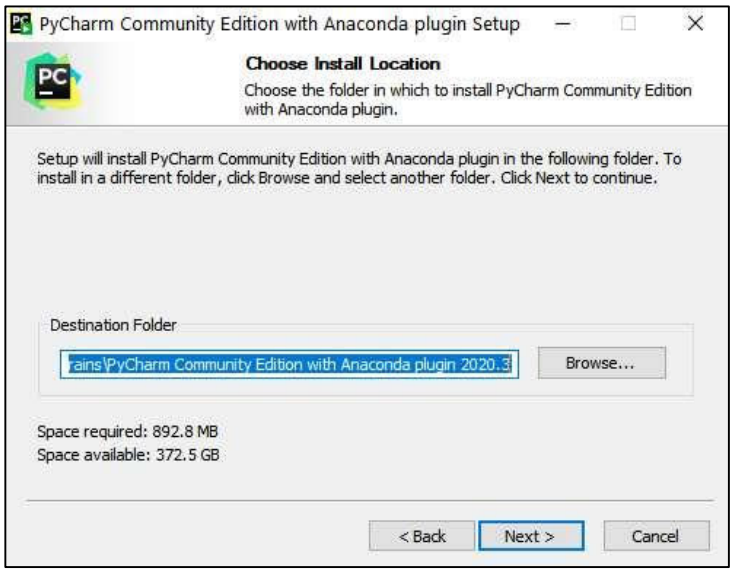


Figura A.8 Ventana de instalación de PyCharm Community.

3. Seleccionar cada una de las opciones mostradas en la ventana de instalación de PyCharm Community como se muestra en la Figura A.9.

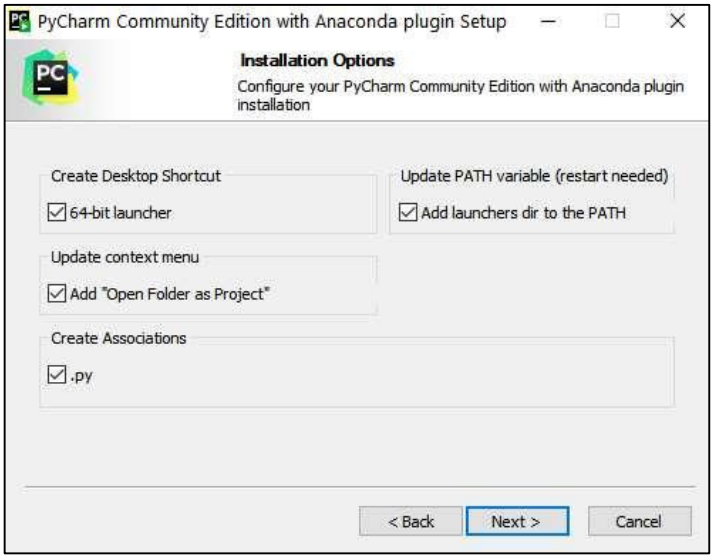


Figura A.9 Ventana de instalación de PyCharm Community.

4. Seleccionar **Reboot now** y finalizar instalación como se muestra en la Figura A.10.

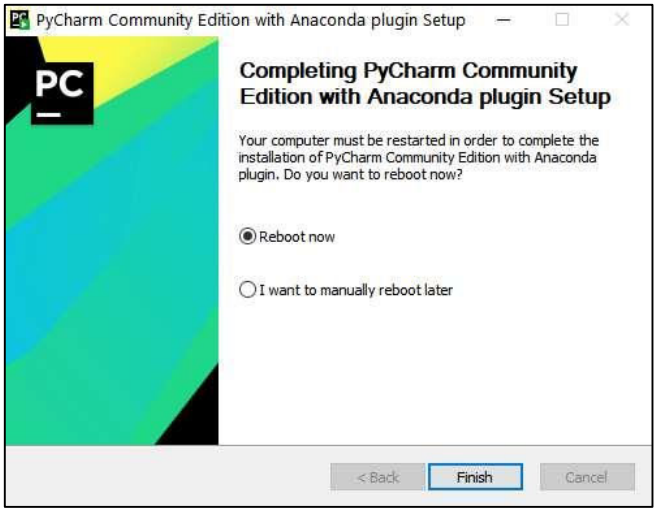


Figura A.10 Ventana de instalación de PyCharm Community.

A.2.6.1. Crear un ambiente Conda

1. Asegúrese de que Anaconda este descargado e instalado en su computadora, y que conozca una ruta a su archivo ejecutable.
2. Crear carpeta con el nombre **Proyecto** en el escritorio.

3. Abra el programa PyCharm Community Edition with Anaconda plugin 2020.1.2 x64.
4. Seleccionar **New Proyect**.

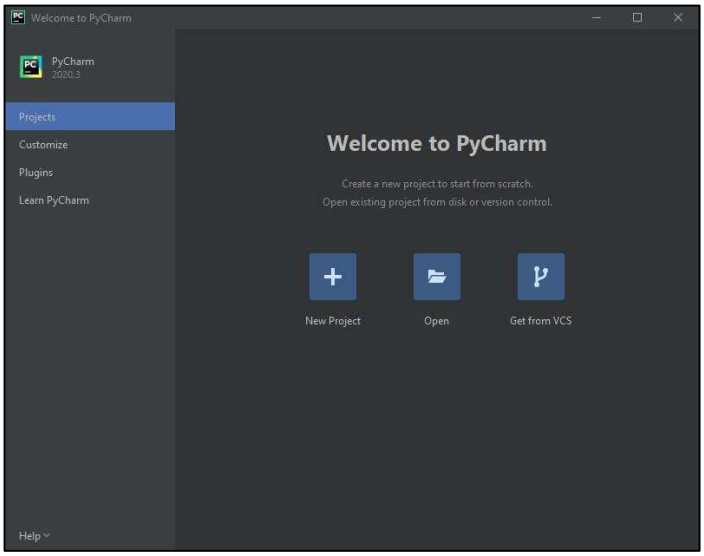


Figura A.11 Ventana de inicio de PyCharm Community.

5. Una vez seleccionado la opción crear nuevo proyecto, configurar el nuevo proyecto, como se muestra en la Figura A.12.

5.1 Seleccionar la localización del proyecto a crear: C:\Users\USER\Desktop\Proyecto

5.2 Seleccionar la opción **New Enviroment using Conda** y localización: C:\Users\USER\Anaconda2\envs\Proyecto.

5.3 Colocar la versión de Python **2.7**.

5.4 Seleccionar **Create**.

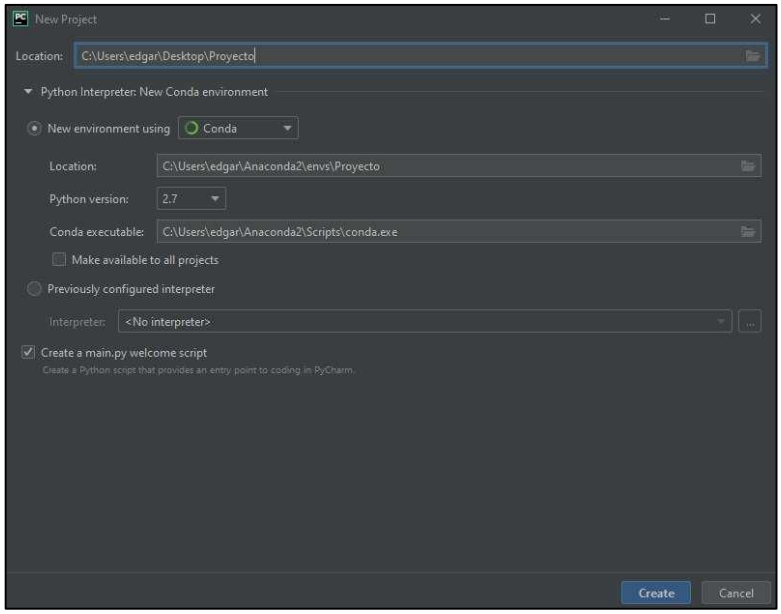


Figura A.12 Ventana New Proyect de PyCharm Community.

6. Configurar el intérprete con el que trabajara el proyecto.

6.1 Presione Ctrl+Alt+S para abrir Settings/ Proyect:/ Python Interpreter.

6.2 Hacer clic en configuración y clic en **add**, como se muestra en la Figura A.13.

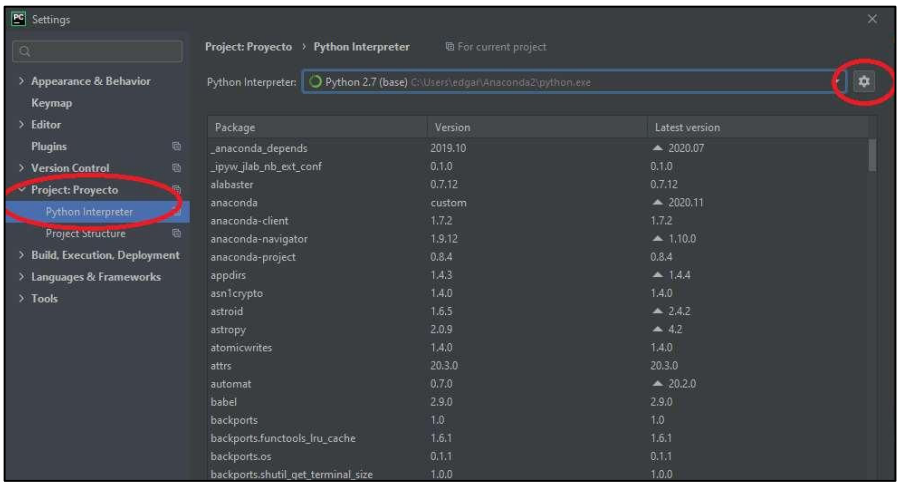


Figura A.13 Ventana Settings de PyCharm Community.

6.3 Hacer clic en **Virtualenv Enviroment** y seleccionar **Existing Enviroment** con la dirección C:\Users\USER\Anaconda2\python.exe, como se muestra en la Figura A.14.

6.4 Hacer clic en **Ok**.

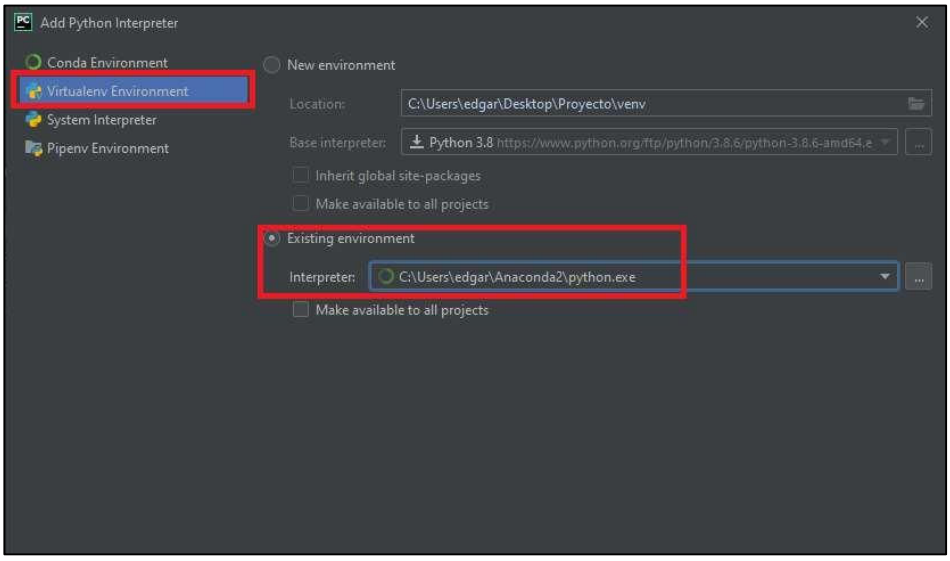


Figura A.14 Ventana Add Python Interpreter de PyCharm Community.

7. Copiar archivos del Proyecto

7.1 Abrir la Carpeta instaladores\Proyecto

7.2 Copiar todos los archivos contenidos a excepción de la carpeta. idea en la dirección

C:\Users\USER\Desktop\Proyecto

8. Selección del Interprete creado.

8.1 Hacer clic flecha con dirección hacia abajo, y seleccionar Edit Configurations.

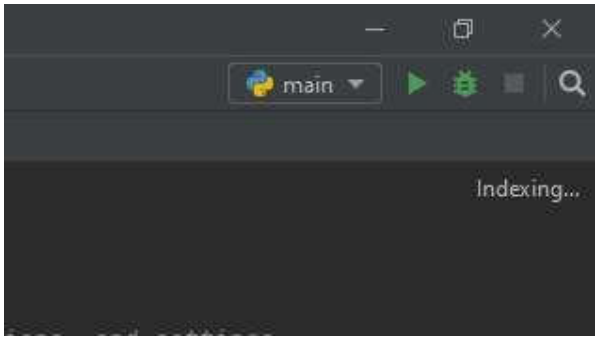


Figura A.15 Acercamiento de ventana de trabajo de PyCharm Community.

8.2 Hacer clic en **Add New Configuration**, como se muestra en la Figura A.16.

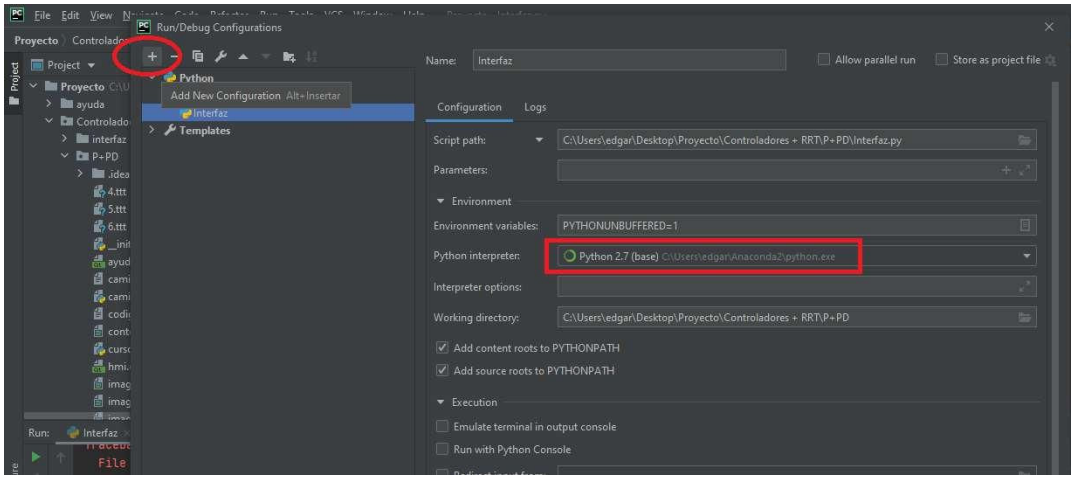


Figura A.16 Ventana Run/Debug Configutations de PyCharm Community.

8.3 Hacer clic en Tipo de archivo **PYTHON**.

8.4 Seleccionar la localización del proyecto y el archivo .py que desea ejecutar en el espacio del **Script Path**, en este caso el archivo .py con el nombre de "**Interfaz**", C:\Users\USER\Desktop\Proyecto\Controladores + RRT\P+PD\Interfaz.py.

8.5 Seleccionar interprete creado anteriormente de Python, se debe escoger estrictamente **Python 2.7(base)**.

8.6 Finalmente hacer clic en **Ok**.

8.7 Para ejecutar el Proyecto se debe dar clic en **Run**.

ANEXO B

B. MANUAL DE USUARIO DE LA INTERFAZ HMI

B.1. INTRODUCCIÓN

La interfaz hombre-máquina HMI es importante para poder realizar pruebas con cada controlador junto con el algoritmo de planificador de rutas RRT, poder modificar los parámetros de control y observar los resultados obtenidos.

B.2. ARRANQUE DE LA INTERFAZ

Para el arranque de la interfaz basta con ejecutar el programa PyCharm Community y hacer clic en el proyecto como ve en la Figura B1, que fue cargado anteriormente. Puede referirse al Anexo A, subsección A.2.6.1. para más información.

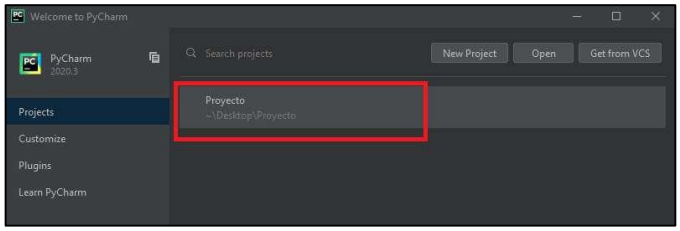


Figura B1. Ventana Welcome to PyCharm de PyCharm Community.

Una vez abierto el proyecto existente, se debe hacer clic en **Run**, ejecutando así la interfaz del Proyecto y mostrando la pantalla de Nivel I (presentación), con el que se interactua con el usuario, como se muestra en la Figura B.2.

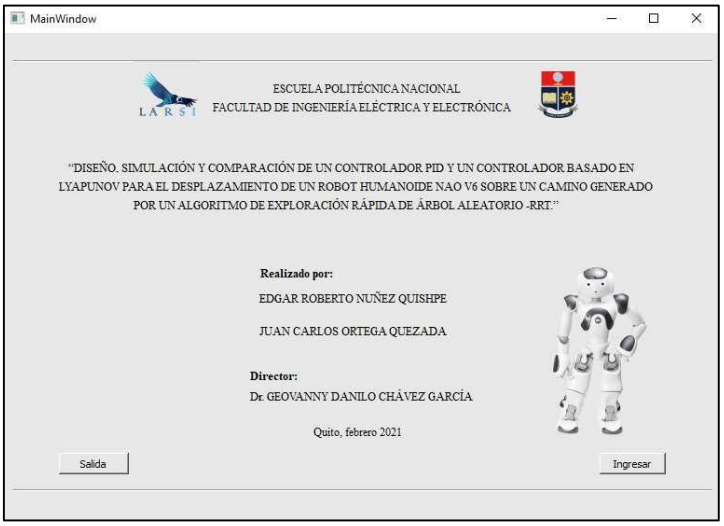


Figura B.2 Interfaz para el usuario Nivel I

B.3. SELECCION DE PARAMETROS

B.3.1. SELECCIÓN DEL AMBIENTE DE SIMULACIÓN

Una vez se dé clic en el botón **Ingresar**, se mostrará la ventana de Nivel II, para la selección de cada parámetro de la simulación del robot NAO. Primero se debe escoger el ambiente de simulación que contará con tres diferentes escenarios, los cuales se pueden observar al seleccionar alguno de los tres escenarios en la parte inferior izquierda. Una vez seleccionado el aula, hacer clic en el botón abrir, como se muestra en la Figura B.3.

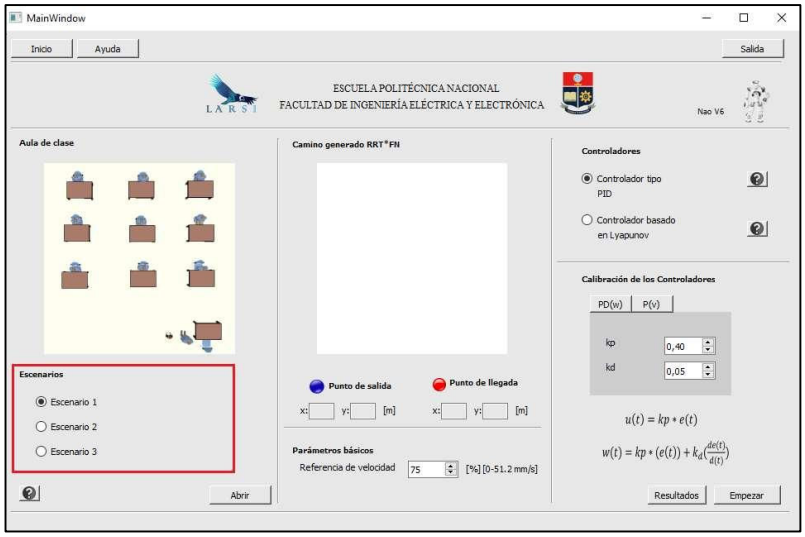


Figura B.3 Interfaz para el usuario Nivel II, selección de escenario.

B.3.2. CONFIGURACION DE PARÁMETROS

Una vez seleccionado el ambiente de simulación, el usuario puede escoger el tipo de controlador a usar, ingresar los parámetros de dicho controlador o simplemente dejar la configuración por defecto. El primer parametro a modificar es el valor de referencia de velocidad, como se muestra en la Figura B.4, que puede ser modificada desde 5 al 100% de su maximo valor.

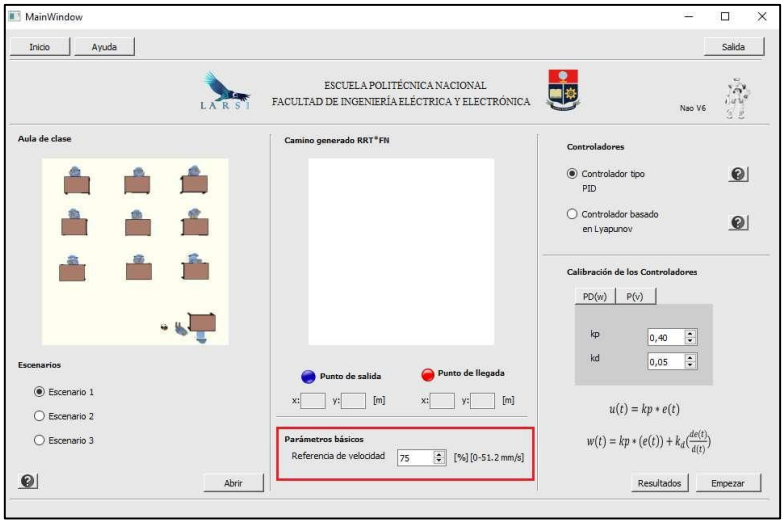


Figura B.4 Interfaz para el usuario Nivel II, selección de velocidad lineal.

Una vez seleccionado el valor de referencia de velocidad lineal, se elije el tipo de controlador, tal y como se puede ver en la Figuara B.5. Se puede escoger entre un controlador tipo PID para la velocidad lineal y velocidad angular, o un controlador basado en lyapunov.

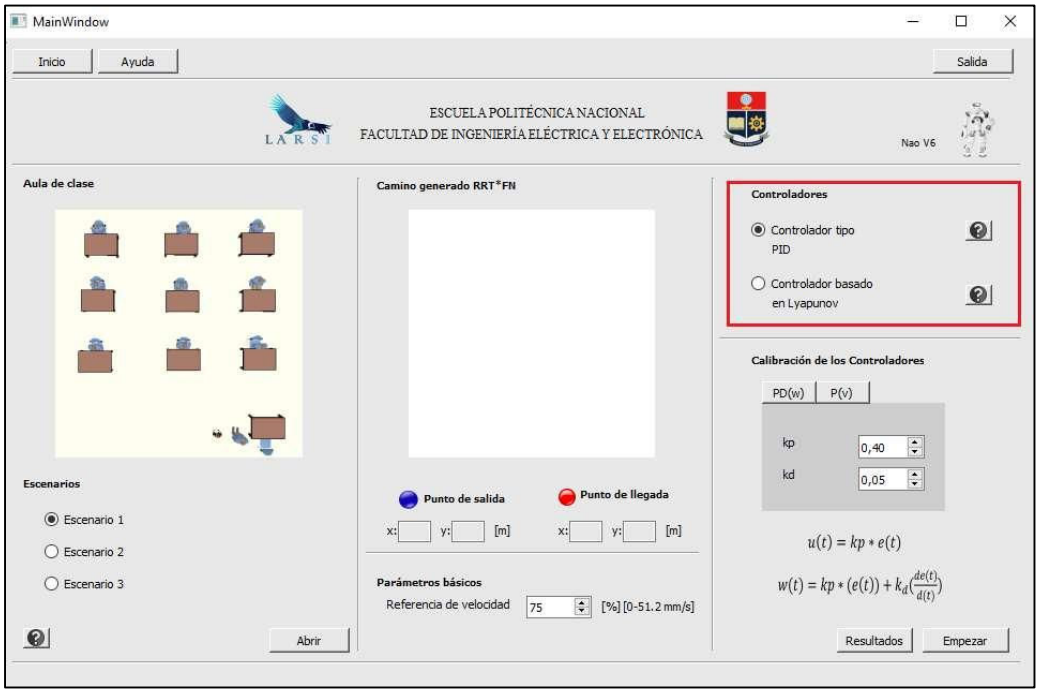


Figura B.5 Interfaz para el usuario Nivel II, selección de tipo de controlador.

Al seleccionar el tipo de controlador, automáticamente se mostrará los parámetros a modificar, como se ve en la Figura B.6, dependiendo del controlador que se escogió con anterioridad. El usuario puede modificar los parámetros del controlador y optar por dejar los parámetros recomendados preestablecidos.

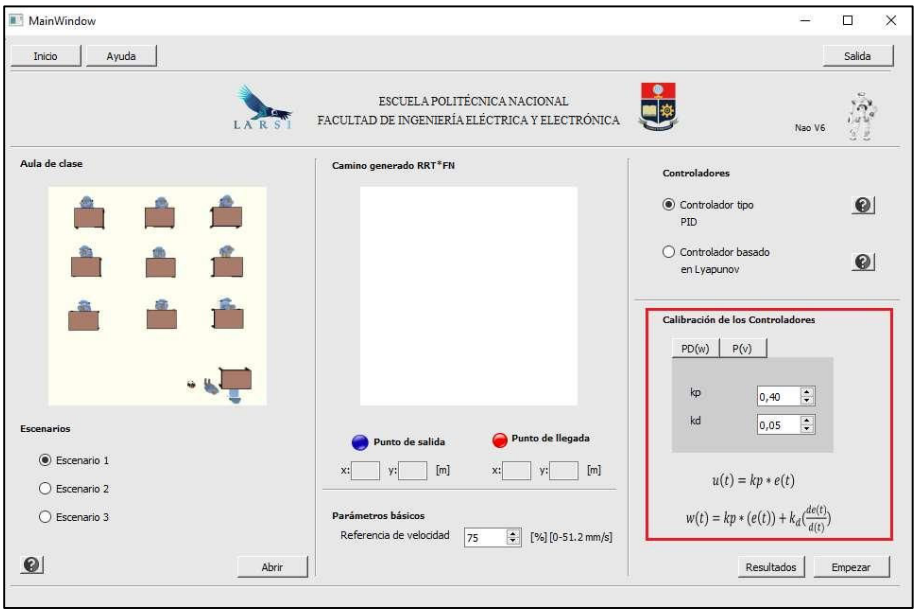


Figura B.6 Interfaz para el usuario Nivel II, parametros del controlador.

B.4. ARRANQUE DEL CONTROLADOR Y PLANIFICADOR DE CAMINOS

Una vez seleccionados cada uno de los parámetros de control y escenario del robot, se debe hacer clic en el botón **Empezar**, tomando en cuenta que previamente se debe dar **play** en la ventana de simulación en CoppeliaSim Player para que la interfaz pueda interactuar con el robot ubicado en el ambiente simulado, como se ve en la Figura B.8.

Al dar clic en el botón empezar, se abrirá una ventana emergente, como se muestra en la Figura B.9, dependiendo del escenario seleccionado con anterioridad. El objetivo es que el usuario seleccione del destino final, al que se desea que el robot se desplace. En caso de no seleccionar ningún punto al que el robot debe llegar, simplemente se cerrará la ventana y terminará la simulación.

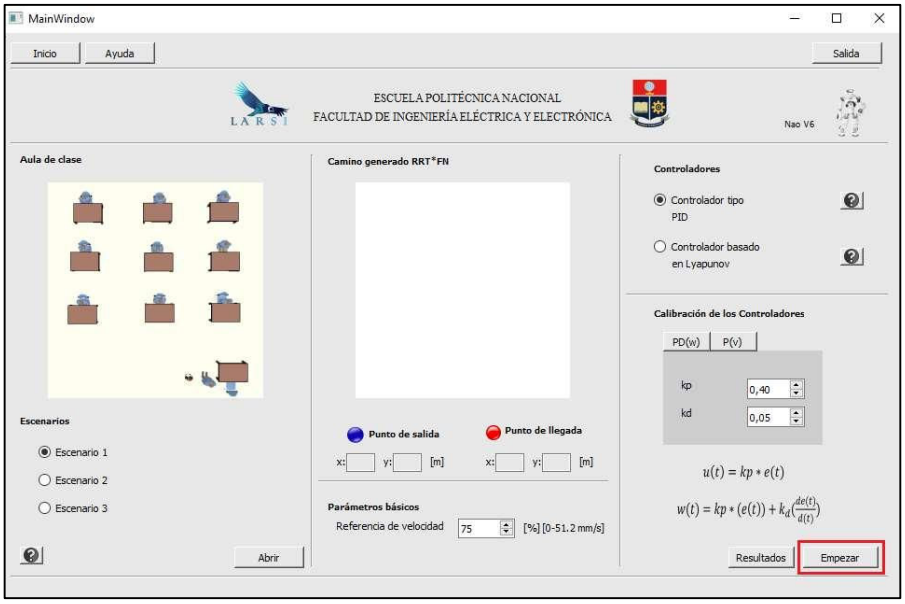


Figura B.8 Interfaz para el usuario Nivel II, arranque del controlador y el planificador de caminos.

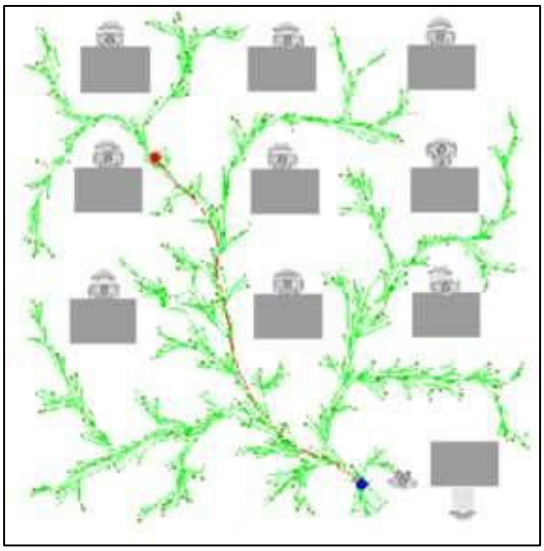


Figura B.9 Ventana, selección del destino del robot.

Al finalizar el desplazamiento del robot NAO por todo el camino generado por el planificador de caminos RRT, se mostrará el botón de resultados, Figura B.9.

Figura B.9 Habilitación del botón resultados.

Para poder visualizar los resultados obtenidos, se hace clic en el botón **Resultados**, ubicado junto al botón **Empezar**, como se puede observar en la Figura B.8, con lo que se abrirá la ventana de Nivel 4. En la ventana de resultados, se puede visualizar el camino generado por el planificador de caminos RRT, y cada una de las gráficas tanto de velocidad lineal, angular, ángulo de referencia, y camino recorrido. Al seleccionar alguna de las opciones, se debe procurar dar clic sobre la opción **graficar** como se visualiza en la Figura B.10.

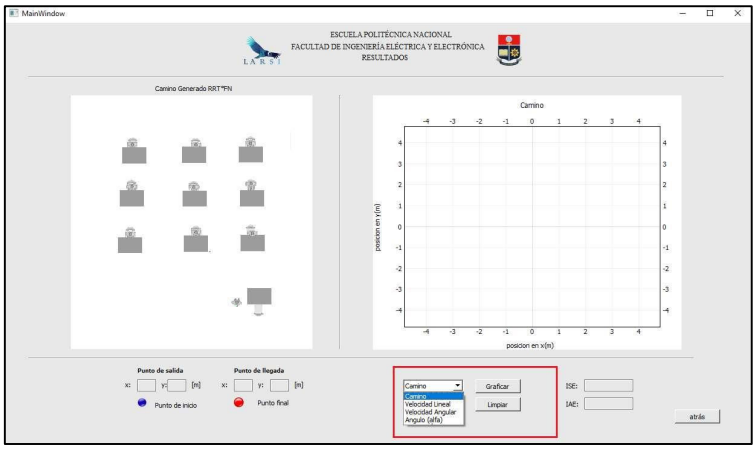


Figura B.10 Ventana de resultados.

B.5 AYUDA

En caso de ser necesario, existe botones de ayuda en la ventana de Nivel 2 de la interfaz (Figura B.11), con la información de los parámetros a configurar antes de arrancar el controlador. Al hacer clic, se abrirá una ventana de ayuda con un índice de la información que puede tener acceso el usuario, como se muestra en la figura B.12.

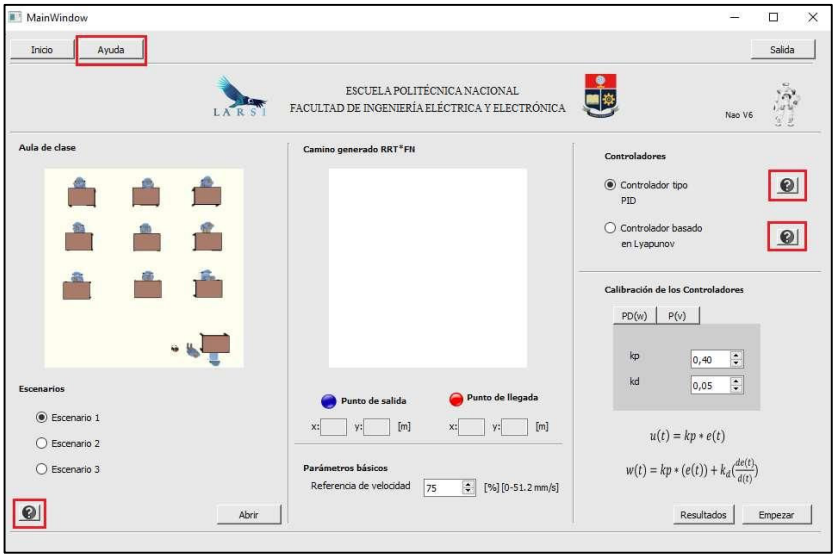


Figura B.11 Ventana con todos los botones existentes de ayuda al usuario.

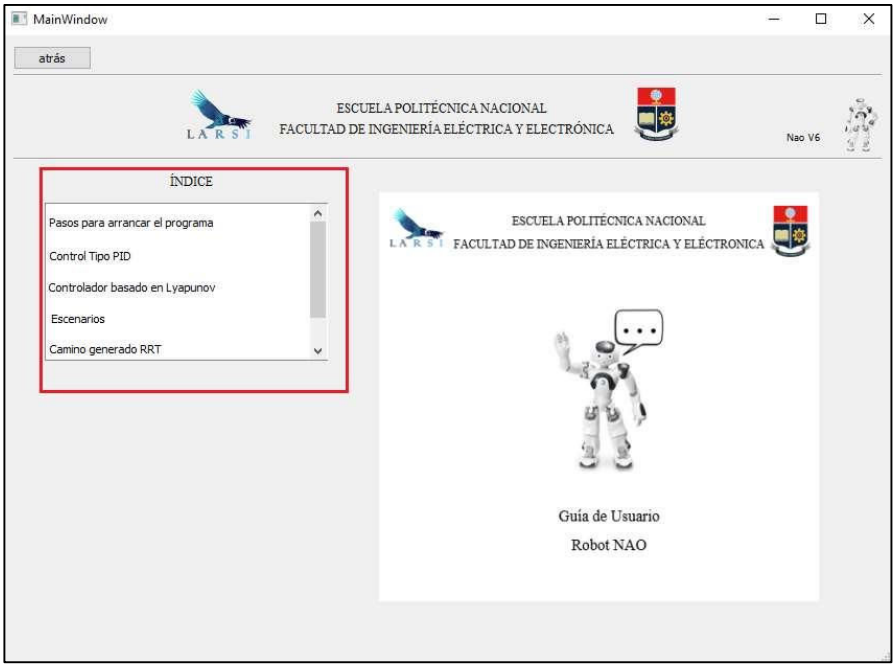


Figura B.12 Ventana de ayuda para el usuario.