# ESCUELA POLITÉCNICA NACIONAL

## FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

# INTEGRACIÓN DE UN SISTEMA DE CONTROL Y MONITOREO DE UN GRUPO DE ROBOTS MÓVILES TERRESTRES BASADO EN VISIÓN ARTIFICIAL Y ENFOCADO AL MAPEO Y NAVEGACIÓN DENTRO DE UN ÁREA DE TRABAJO PROVISTA DE OBSTÁCULOS

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

**LEANDRO ISAAC PONCE CEVALLOS**

leandro.ponce@epn.edu.ec

**DIRECTOR: ING. PATRICIO JAVIER CRUZ DAVALOS, PhD.**

patricio.cruz@epn.edu.ec

**Quito, octubre 2021**

# AVAL

Certifico que el presente trabajo fue desarrollado por Leandro Isaac Ponce Cevallos, bajo mi supervisión.

**ING. PATRICIO JAVIER CRUZ DAVALOS, PhD.**

**DIRECTOR DEL TRABAJO DE TITULACIÓN**

# DECLARACIÓN DE AUTORÍA

Yo Leandro Isaac Ponce Cevallos, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

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

Leandro Isaac Ponce Cevallos

# DEDICATORIA

*Dedicada a mis padres, por ser quienes han guiado toda mi trayectoria personal durante mis buenos y malos momentos de manera incondicional. A mis hermanas, por ser quienes han inspirado mi dedicación y esfuerzo mediante su ejemplo personal y profesional.*

.

# AGRADECIMIENTO

Este trabajo de titulación ha sido posible gracias al inmenso apoyo incondicional de mis padres. El esfuerzo y dedicación de toda mi trayectoria ha sido posible solo gracias a la guía constante, a la inspiración diaria y al ejemplo de excelencia que me han brindado durante toda mi vida. Este agradecimiento es también una promesa de superación continua en honor a la incansable confianza que mis padres han depositado en mí.

Agradezco también a mis hermanas, por su ejemplo de excelencia personal y profesional. Han inspirado en mi un profundo deseo de alcanzar lo más alto y de compartir con ellas todos mis éxitos y alegrías.

A mi tutor de tesis, Patricio Cruz, por su constante guía durante este proceso final de mi carrera universitaria. Gracias a su tiempo y dedicación ha sido posible la finalización de este trabajo que refleja su excelencia profesional y personal.

A mis amigos, que han acompañado mi camino de manera desinteresada y han sido un apoyo indispensable en mi travesía. Gracias por todas las anécdotas y alegrías vividas.

A Evelyn, por su afecto y apoyo constante durante la etapa final de mi carrera, quien ha logrado inspirar confianza y serenidad en mi camino.

Finalmente, a la Escuela Politécnica Nacional y a todos los profesores que han brindado su esfuerzo y dedicación para forjar profesionales de calidad académica y personal.

# ÍNDICE DE CONTENIDO

# RESUMEN

El presente trabajo de titulación consiste en una extensión de la plataforma de pruebas experimental para robots móviles de tamaño reducido desarrollada en [1] y actualizada en [2]. En particular, se expande la funcionalidad del sistema de visión artificial para permitir la identificación de posibles obstáculos. Adicionalmente, se detalla la implementación del algoritmo RRT* (Rapid Exploring Random Trees) para la planificación de rutas, tomando en cuenta obstáculos de cualquier forma geométrica presentes en el área de trabajo de la plataforma y se implementa el lazo de control de movimiento para el seguimiento de las rutas generadas. Además, se incluyen cambios de hardware respecto a los módulos de comunicación y la adición de un cuarto robot móvil a los tres ya existentes.

El monitoreo del sistema, respecto a la planificación de rutas y seguimiento de las mismas, se realiza mediante nodos ROS (Robotic Operating System) que incluyen dos interfaces gráficas diseñadas en Qt Designer. Dichas interfaces facilitan la interacción del usuario con el sistema, permitiendo conocer datos importantes, así como la configuración de parámetros relevantes para el cálculo de rutas y para el control de movimiento de los robots móviles de tracción diferencial.

Finalmente, se demuestran las nuevas capacidades del sistema mediante la obtención de gráficas que ilustran su desempeño, así como de la ejecución del algoritmo de planificación de rutas.

**PALABRAS CLAVE:** ROS, Planificación y Seguimiento de Rutas, Control de Movimiento, Visión Artificial, Mapeo.

# ABSTRACT

This project develops an extension of the experimental testbed platform for mini-sized mobile robots presented in [1] and updated in [2]. The artificial vision system is expanded to allow the identification of potential obstacles. Furthermore, the implementation of the RRT* (Rapid Exploring Random Trees) algorithm for path planning is detailed which considers obstacles of any geometric shape present in the task space. Also, a motion control-loop for route-tracking is presented. In addition, a fourth mobile robot and hardware changes regarding the communication modules of the system are added.

The system monitoring, focused on path-planning and tracking tasks, is performed by using ROS (Robotic Operating System) nodes that launch two graphical user interfaces created in Qt Designer. These interfaces facilitate the user interaction with the system since they show relevant system data and allow the setup of relevant parameters to the route calculation and for the motion control of the four differential-drive mobile robots.

Finally, the new capabilities of the system are tested, and the results show the system's performance for both the path-planning and route-tracking goals.



**KEYWORDS** ROS, Route Planning and Tracking, Motion Control, Computer Vision, Mapping.

# 1 INTRODUCCIÓN

Industrialmente, las aplicaciones que pueden ser desarrolladas con sistemas de grupos de robots son cada vez más relevantes dada la inmensa diversidad de problemas complejos que pueden ser resueltos al utilizar estas plataformas trabajando de manera coordinada.

Justamente, los sistemas multi-agentes son una solución que se ha popularizado progresivamente tras haber sido aplicados en algunas aplicaciones como distribución dinámica de sensores, movilización de objetos de manera cooperativa, exploración y rescate, etc.

Posiblemente, la principal característica de estos sistemas es la disponibilidad de numerosos robots que individualmente son capaces de realizar tareas relativamente simples como movilización, monitoreo y mapeo local; pero que al utilizarlos en grupos comandados de forma organizada, ya sea centralizada o distribuida, las tareas posibles de cumplir exitosamente se escalan de manera proporcional. Por ejemplo, de esta manera se puede lograr la movilización de objetos de gran tamaño, monitoreo de áreas extensas o mapeo de ambientes completos a través de la combinación de los datos de cada robot.

El desarrollo de aplicaciones para grupos robóticos generalmente demanda extensas pruebas de funcionamiento, compatibilidad y desempeño. Con este objetivo, se ha venido desarrollando en la Escuela Politécnica Nacional una plataforma de pruebas que permite realizar ensayos y pruebas prácticas sobre un grupo de tres robots de tamaño reducido con capacidades básicas de movilidad. Dada la complejidad y costo que acompaña la implementación de este tipo de plataformas, el desarrollo se ha realizado de manera continua y progresiva. Una primera iteración se presentó en [1] donde se implementó la arquitectura básica de hardware y software para el control y seguimiento de trayectorias de dos robots móviles diferenciales de manera individual, los mismos que eran identificados a través de marcadores ArUco. Uno de los mayores aportes de dicha iteración fue la iniciación de un proyecto con capacidad de expansión y mejoramiento continuo, acompañado de una aplicación práctica que demostraba la utilidad de la plataforma. En una segunda iteración [2], se realizó una modificación crucial para llevar al sistema a un sistema operativo abierto que permita una fácil expansión sin necesidad de contar con licencias de software para su funcionamiento. Justamente en [2], se migró el sistema al sistema operativo Linux y se desarrolló el software de seguimiento de rutas sobre el Sistema Operativo Robótico (ROS por sus siglas en inglés) para un grupo de tres robots que se desplazan mientras mantienen

una formación triangular. Esta migración abrió las puertas para una expansión de manera modular que no requiera desmontar las aplicaciones previamente implementadas para poder crear nuevas.

El presente documento detalla los nuevos avances que se han desarrollado en este proyecto de titulación sobre la plataforma y sus potenciales aplicaciones. Como primer aporte se cuenta con la adición de un cuarto robot a los tres ya existentes; de esta manera se da continuidad en la escalabilidad respecto al número de robots que es uno de los objetivos permanentes del desarrollo de la plataforma. En segundo lugar, se realiza la implementación de un sistema de planificación de rutas que proporciona una solución viable dentro de un entorno de trabajo provisto de obstáculos. Además, para ejecutar la movilización de cada robot, se implementa una arquitectura de control que asegure que cada robot móvil cumpla el seguimiento de la ruta generada. Se debe tener en cuenta que como parte del objetivo de contar con una movilidad segura y planificada se implementa también un esquema de prevención de colisiones entre robots.

Finalmente, el desarrollo de la aplicación de planificación de rutas y control de movilidad está acompañado de la implementación de una interfaz de usuario que permite la planificación, comando y monitoreo del sistema, así como la obtención de resultados de desempeño.

## 1.1 OBJETIVOS

El objetivo general del presente trabajo es:

Integrar un sistema de control y monitoreo de un grupo de robots móviles terrestres basado en visión artificial y enfocado al mapeo y navegación dentro de un área de trabajo provista de obstáculos.

Para cumplir con dicho objetivo, se han establecido los siguientes objetivos específicos:

- Realizar una revisión bibliográfica de algoritmos de mapeo, planificación de rutas y evasión de obstáculos
- Acoplar el sistema de visión artificial para obtener el mapa del área de trabajo donde se consideren obstáculos dentro del mismo.
- Incrementar un robot móvil adicional a los 3 ya existentes en la plataforma e implementar un algoritmo de planificación de rutas y evasión de obstáculos sobre la plataforma de pruebas conformada por 4 robots.

- Desarrollar una interfaz gráfica de usuario para realizar el monitoreo y control del grupo de robots móviles, así como para la visualización del mapa y las rutas planificadas y ejecutadas.
- Evaluar el desempeño del sistema completo, analizando a través de pruebas, el funcionamiento del algoritmo de navegación implementado, así como la generación y seguimiento de rutas con evasión de obstáculos.

## 1.2 ALCANCE

- Se presenta una revisión bibliográfica de al menos tres algoritmos de planificación de rutas y evasión de obstáculos, así como de un método de mapeo para el área de trabajo.
- Se determina el algoritmo de navegación y planificación de rutas, así como el método de mapeo bidimensional a implementar sobre la plataforma de pruebas basándose en los requerimientos y limitaciones del sistema.
- Se utilizan los elementos existentes del sistema presentado en[2], incluyendo la cámara, los 3 robots existentes y el espacio físico de trabajo (120cm x 80cm). Se agrega un robot móvil adicional para demostrar la capacidad de expansión del sistema que se propone implementar en este proyecto.
- El sistema de visión artificial es modificado y adaptado para reconocer, además de los marcadores ArUco que se encuentren sobre cada robot, distintas figuras geométricas que se coloquen sobre el espacio de trabajo, las cuales serán identificadas como obstáculos a evadir durante la navegación de cada robot.
- Una vez implementado el sistema de mapeo bidimensional, a través del módulo de visión artificial, se implementa el algoritmo seleccionado para la navegación de cada robot desde un punto de salida a un punto de llegada, seleccionados por el usuario.
- Se implementa el seguimiento de las rutas generadas por cada robot móvil de acuerdo con el algoritmo de navegación.
- Se desarrolla una interfaz gráfica para comandar los robots móviles y monitorear el funcionamiento del sistema.
- Se realizan pruebas para evaluar la eficacia del algoritmo de generación de rutas mediante la modificación del escenario respecto a la ubicación de obstáculos en el área de trabajo, así como los puntos de partida y llegada designados para cada robot móvil.
- Se realizan pruebas para evaluar la eficacia del método de mapeo del área de trabajo mediante la modificación del tamaño y ubicación de los obstáculos presentes.

## 1.3 MARCO TEÓRICO

En el presente capítulo se describen los aspectos más importantes que forman parte del fundamento teórico necesario para la implementación de los objetivos planteados.

### 1.3.1 SISTEMAS BASADOS EN GRUPOS ROBÓTICOS

Los sistemas conformados por grupos robóticos, en algunos casos llamados sistemas multi-agentes, son esquemas que permiten completar tareas complejas empleando las capacidades individuales de cada robot o agente para que, de forma cooperativa o grupal, logren completar labores que serían complejas o imprácticas para un solo robot [17].

Un grupo robótico puede ser identificado por las características de sus individuos, como por ejemplo: reactividad, proactividad, comunicación, sociabilidad, movilidad, autonomía y adaptabilidad, entre otras [16].

Estos sistemas robóticos pueden ser clasificados en dos grupos:

**Sistemas con control descentralizado**

Son descritos como esquemas robóticos donde ningún agente, por sí solo, tiene la capacidad de resolver el problema. El control se realiza de manera descentralizada, sin contar con un sistema que conozca el estado del grupo robótico de manera completa. La computación de datos se realiza de manera asíncrona y de manera local en cada agente [4].

Las mayores ventajas de estos sistemas es la independencia de un sistema central de control, la capacidad de adaptabilidad frente a condiciones dinámicas del entorno, la diversidad de posibles tareas a completar de manera cooperativa y la resiliencia a la perdida de agentes; es decir, en caso de perder uno o varios agentes, los demás robots pueden retomar las tareas asignadas a los robots perdidos.

Justamente, una de sus fortalezas es la posibilidad de ser desplegados de manera independiente en ambientes donde la comunicación con un nodo central sea imposible o limitada, sea por interferencia o por seguridad de operación [19].

Por otro lado, el desempeño de estos sistemas está dispuesto por las capacidades individuales de cada agente. Sus limitaciones, al igual que sus fortalezas están dadas por las limitaciones individuales de los agentes. Al diseñar aplicaciones con sistemas multi-agentes descentralizados se debe tener en cuenta que existirá siempre limitaciones de poder computacional, alcance de comunicación entre agentes, sensibilidad de sensores y limitaciones de los actuadores en cada robot. [18]

Adicionalmente, una característica inherente a estos sistemas es la complejidad y costo de cada agente, puesto que es indispensable que estos cuenten con una variedad de sensores, así como de un microcontrolador capaz de procesar todos los datos de manera local.

**Sistemas con control centralizado**

Si bien son sistemas que no ofrecen todas las ventajas que ofrecen los sistemas descentralizados, son muy útiles en casos donde las tareas requeridas sean más simples y pueden ser completadas mediante la centralización del monitoreo y control.

Este tipo de sistemas permiten cierta flexibilidad dado que aún son capaces de ser configurados para realizar tareas de manera cooperativa. Si bien el control y datos están centralizados, también se puede configurar a cada agente para que procese tareas de manera local.

Una de sus ventajas es la facilidad de implementación, puesto que es más sencillo realizar un control centralizado del sistema, con algoritmos más simples que los necesarios en sistemas multi-agentes descentralizados.

Además, con un sistema central de control, la limitación en capacidad computacional es menor. El nodo de control central puede realizar cálculos más complejos y solamente transferir a los individuos las ordenes necesarias para el cumplimiento de la tarea.

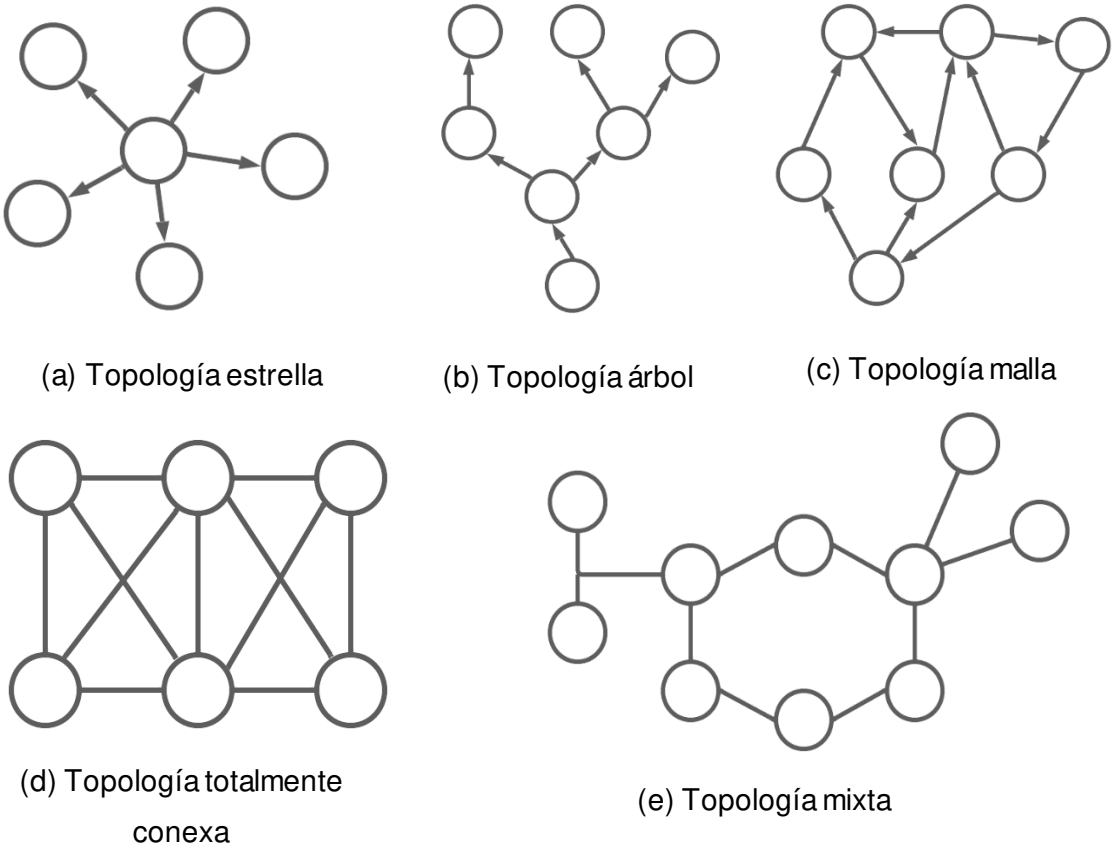
Una ventaja adicional es que cada agente no requiere ser extremadamente complejo ni contar con una gran cantidad o variedad de sensores, el procesamiento más complejo se realizara de manera remota por lo que el procesador de cada individuo puede ser simple y de costo reducido.

Una de las mayores desventajas es la dependencia del nodo central de control, que, en caso de una falla, todo el sistema queda incapacitado. Esta debilidad hace que la aplicación de estos sistemas sea limitada a tareas que no sean de naturaleza crítica.

Otra limitación de estos sistemas, es la capacidad de no poder ser desplegados en lugares remotos de manera segura. Puesto que se requiere un sistema de comunicación entre el control central y cada agente, el grupo de robots no puede ser desplegado a grandes distancias del nodo central o en ambientes donde interferencias puedan interrumpir dicha comunicación.

**Comunicación en sistemas robóticos grupales**

Los esquemas de comunicación dependen en gran medida de la aplicación y del tipo de sistema. Los esquemas de comunicación más comunes, ilustrados en la Figura 1.1 son [20]:



(a) Topología estrella (b) Topología árbol (c) Topología malla

(d) Topología totalmente conexa (e) Topología mixta

**Figura 1.1** Topologías de red

- *Estrella*: Cada nodo o agente se conecta con un nodo central quien comanda y envía instrucciones a todos los demás nodos. Esta topología puede ser utilizada tanto en sistemas descentralizados como centralizados. La principal desventaja de esta topología de red es la dependencia del sistema sobre un nodo central. Adicionalmente, en caso de requerir enviar datos entre nodos periféricos, la información deberá pasar a través del nodo central. Ver Figura 1.1 (a)
- *Tipo árbol*: Se caracteriza por tener un nodo principal al que se conectan de manera directa e indirecta a todos los demás nodos. Esta topología puede ser usada en sistemas centralizados y descentralizados, pero es más útil en sistemas descentralizados, donde se cuenta con un agente coordinador que monitorea el estado de los demás agentes. Ver Figura 1.1 (b)
- *Malla*: Es una topología extremadamente útil para agentes de un sistema descentralizado puesto que permite la comunicación entre varios nodos de

manera directa teniendo en cuenta agentes coordinadores que pueden conectar con otras mallas. Una ventaja del uso de esta topología es la resiliencia a cortes de comunicación directa entre agentes puesto que los datos pueden ser direccionados a través de otros nodos hasta que los datos lleguen a su destino. Ver Figura 1.1 (c)

- *Totalmente conexa*: Consiste en una comunicación directa entre todos los nodos del sistema. Es una topología que requiere de muchos recursos computacionales pero que a su vez ofrece total robustez ante cortes de comunicación. Ofrece caminos directos e indirectos entre nodos para su comunicación. Ver Figura 1.1 (d)
- *Mixta*: Dependiendo de los requerimientos de la aplicación, se pueden unir varias topologías para crear esquemas mixtos, donde una parte de la red puede estar configurada como malla, árbol o estrella. Estos esquemas son utilizados en redes de gran escala y en aplicaciones que se pueden beneficiar de las distintas características de cada topología. Ver Figura 1.1 (e)

**Sistema Operativo Robótico (ROS) [21]**

Una herramienta que ha facilitado, desde su creación, la implementación de aplicaciones robóticas es ROS, el cual se define como un sistema operativo robótico que establece un conjunto de reglas y parámetros para lograr mantener un esquema lógico y organizado de comunicación y control. Una de las grandes ventajas de ROS es que es de código abierto, por lo que innumerables usuarios y empresas pueden desarrollar código para ROS que puede ser utilizados por otros usuarios con necesidades similares.

Se basa en la creación e interconexión de nodos. Cada nodo se define como un programa, escrito en Python o C++, que puede recibir y enviar datos a través de un sistema de mensajes y servicios. Estos últimos son un esquema de comunicación entre nodos que se basa en la publicación de mensajes o datos a tópicos que mantienen dichos datos disponibles para cualquier nodo que los requiera. La información en mensajes y servicios pueden contener varios tipos de datos, incluyendo tipos de datos especificados por el usuario. Mientras que un tópico puede ser comprendido como un espacio de memoria que permite la lectura de datos a nodos que se suscriban a él mediante su configuración interna. De la misma manera, permite la publicación o escritura de datos a los nodos que se definan como publicadores de determinado tópico. La estructura básica de un sistema ROS se muestra en la Figura 1.2.

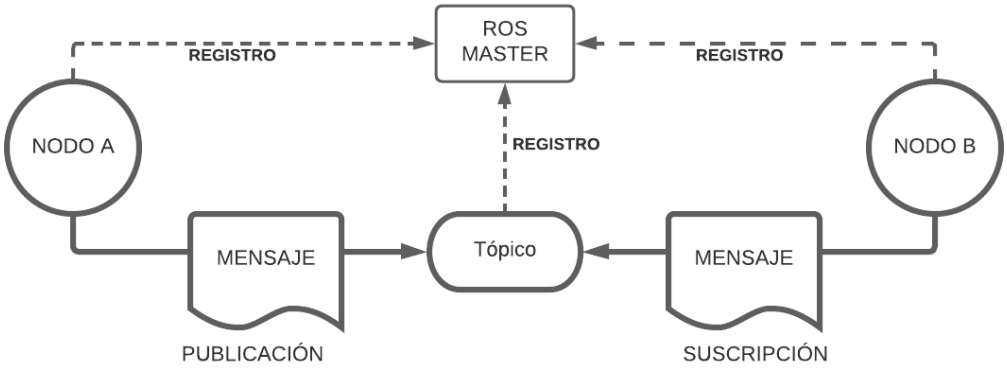



**Figura 1.2** Estructura básica de ROS

Uno de los beneficios al usar ROS, es la posibilidad de diseñar sistemas modulares que pueden ser fácilmente modificados y adaptados a nuevas necesidades. Justamente, en la plataforma de pruebas que es parte de este proyecto, existen distintas funciones que se benefician de esta característica, ya que el monitoreo y control puede ser dividido en módulos que se encargan de tareas específicas como el control de movimiento, monitoreo a través de la cámara, tareas de comunicación, etc.

### 1.3.2 ESTRUCTURA DE LA PLATAFORMA DE PRUEBAS

El presente proyecto se basa en los avances presentados en [1] y [2]; ambas iteraciones anteriores del banco de pruebas han realizado aportes indispensables para la continuidad y mejoramiento del sistema.

La estructura básica de la plataforma fue establecida en [1], con un sistema centralizado de control y dos robots móviles de tracción diferencial. Se implementó una aplicación de seguimiento de trayectorias como demostración de la funcionalidad del sistema. Una de las limitaciones de esta primera iteración fue la necesidad de contar con licencias de uso para programas como MATLAB y VisualStudio, sobre los cuales la aplicación fue desarrollada.

La segunda versión de la plataforma fue desarrollada en [2], donde se realizaron avances indispensables para aumentar la funcionalidad, modularidad y escalabilidad del proyecto. Uno de estos avances fue la migración de la plataforma a un Sistema operativo Linux, donde se lo implementó sobre el sistema operativo robótico ROS. Una segunda mejora fue desarrollada en el esquema de control de los robots móviles, implementando un control de velocidad interno para cada robot y así mejorar su desempeño. En el presente trabajo, dicho control interno de velocidad no será

modificado puesto que permite mantener una precisión adecuada en los movimientos que se comandan a cada robot.

**Estructura de control**

En la segunda iteración del proyecto, el control se realizó en una configuración de cascada, con un control interno de velocidad para cada llanta y un control externo de posición para cada robot.

De hecho, el control externo fue un control de formación para un grupo de tres robots ; tal que una formación triangular se mantenía mientras se realizaba el seguimiento de distintas trayectorias.

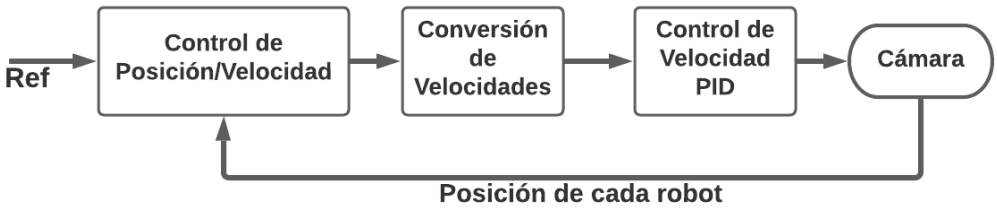



**Figura 1.3** Estructura de control en cascada en la plataforma de pruebas.

La Figura 1.3 muestra la estructura de ambos controles, externo e interno, que comandan a los robots en la plataforma de pruebas.

El control externo de posición o velocidad actúa para llevar de un punto a otro a cada robot, mientras que el control interno de velocidad actúa localmente, en cada robot, para mantener la velocidad de cada rueda en el valor deseado. El bloque de conversión representa la transformación de variables de velocidad lineal y angular a variables de velocidad en revoluciones por minuto para cada llanta.

El controlador PID fue sintonizado a través de pruebas de funcionamiento en el trabajo presentado en [2], por lo que en el presente proyecto se utilizarán las ganancias ya establecidas, mismas que ofrecen un adecuado desempeño.

**Robots móviles de tracción diferencial**

La plataforma de pruebas está basada en un grupo de robots móviles de tracción diferencial de bajo costo, que pueden ser construidos de manera rápida y sencilla. Una ventaja del uso de robots diferenciales es que su control es relativamente simple, respecto a otros tipos de robots móviles.

Ya que uno de los objetivos de la plataforma de prueba es mantener un costo bajo de implementación y escalabilidad, los robots móviles utilizados desde la primera versión

del proyecto han sido inspirados en los agentes de la plataforma MicroMVP [3]. Su estructura física se basa en componentes impresos en 3D y con componentes electrónicos de bajo costo; los cuales son los siguientes:

- Microprocesador Fio v3 - ATmega32U4: Microprocesador encargado de la recepción de datos a través del módulo Xbee y del control de velocidad PID de cada llanta.
- Módulo DRV8835: Módulo de control para dos motores DC.
- Módulo Xbee: Módulo de comunicación inalámbrica.
- Batería Li-Po
- Motores con caja reductora
- Encoders magnéticos en cuadratura: Utilizados como sensores de rapidez para cada llanta del robot.

**Modelamiento cinemático**

Dada la escala de cada robot y los requerimientos del sistema, es suficiente emplear el modelo cinemático de estos para controlarlos exitosamente. La Figura 1.4 ilustra el modelamiento cinemático del robot. Asumiendo que el punto de referencia a ser considerado para propósitos de control es $(x_c, y_c)$, su modelo viene dado por las ecuaciones (1), (2) y (3).

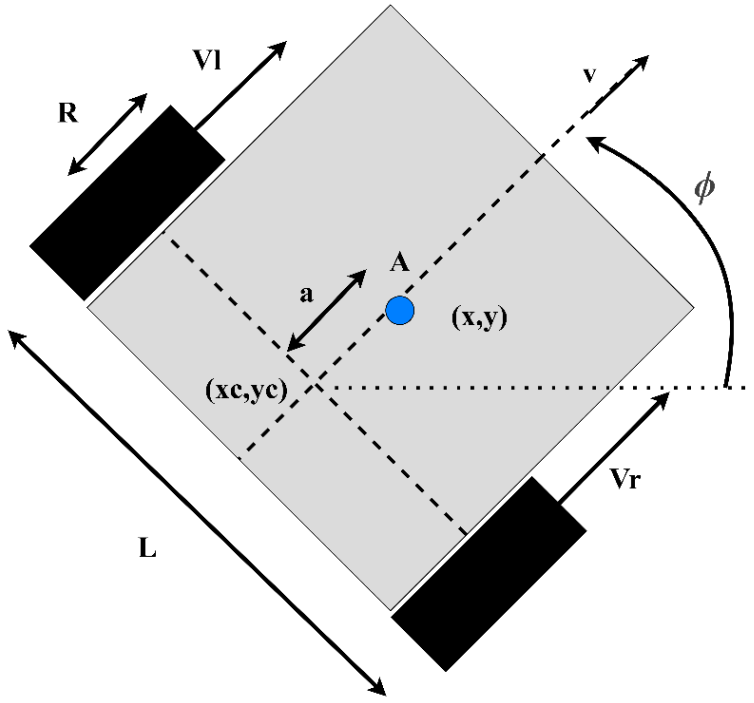



**Figura 1.4** Modelo cinemático del robot de tracción diferencial.

$$\dot{x}_c = \frac{R}{2}(V_r + V_l)\cos(\phi) \quad (1)$$

$$\dot{y}_c = \frac{R}{2}(V_r + V_l)\sin(\phi) \quad (2)$$

$$\dot{\phi} = \frac{R}{2}(V_r - V_l) \quad (3)$$

Donde $R$ es el radio de las llantas, $L$ es la distancia entre llantas y $\phi$ es el ángulo de orientación del robot diferencial. $V_r$ y $V_l$ son la rapidez de cada llanta, derecha e izquierda respectivamente.

Tomando en cuenta las características de movimiento del robot diferencial, se conoce que su modelo cinemático puede ser reducido al de un robot uniciclo. Este modelo es muy útil para simplificar su control, por lo que el robot puede ser modelado en base a su velocidad lineal y angular de la siguiente manera.

$$\dot{x} = v\cos(\phi) \quad (4)$$

$$\dot{y} = v\,sin(\phi) \quad (5)$$

$$\dot{\phi} = w \quad (6)$$

Donde el punto $(x, y)$ es el punto central del uniciclo, v es la rapidez lineal y w es la rapidez angular.

Como se puede observar, los dos modelos cinemáticos pueden ser combinados, para así obtener las velocidades individuales de cada llanta en función de la velocidad lineal y angular del robot.

$$V_r = \frac{2v + wL}{2R} \quad (7)$$

$$V_l = \frac{2v - wL}{2R} \quad (8)$$

Para el caso de los robots móviles de banco de pruebas, el punto de control es desplazado cierta distancia a través de una línea perpendicular al eje de sus llantas. Cuando este nuevo punto de control $A$ es ubicado a cierta distancia $a$ del punto central $(x_c, y_c)$, véase Figura 1.4, se puede expresar el modelo del robot en función de su rapidez lineal y angular.

$$\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} = \begin{bmatrix}\cos(\phi) & -a\sin(\phi)\\ \sin(\phi) & a\cos(\phi)\end{bmatrix}\begin{bmatrix}v\\ w\end{bmatrix} \quad (9)$$

Donde $\phi$ es el ángulo de orientación del robot diferencia, $v$ es su rapidez lineal y $w$ es su rapidez angular. La coordenada $(x, y)$ es la posición del nuevo punto de control ubicado en $A$.

**Marcadores de identificación ArUco**

Como se mencionó en la introducción, los sistemas de control para grupos robóticos pueden ser de tipo centralizado o descentralizado. Para el proyecto de la plataforma de pruebas, tanto el control como el monitoreo se ha realizado de manera centralizada. Como se indicó anteriormente, una de las principales ventajas de este esquema de control es que no se requieren sensores complejos y de alto costo para mantener un monitoreo constante sobre los robots. Adicionalmente, el contar con una cámara como sensor, permite conocer el estado del sistema completo en cualquier momento mediante la utilización de marcadores de referencia ArUco.

En la plataforma de pruebas, la cámara utilizada es la Logitech C920, la cual permite recoger imágenes a 30 cuadros por segundo a una resolución máxima de 1080p. A pesar de no ser una cámara que ofrece las máximas prestaciones en calidad de imagen, resolución o velocidad de obturación, es una solución de bajo costo que para los requerimientos actuales de la plataforma no incide como limitante del sistema en su desempeño o funcionalidad.

En el presente trabajo, se utilizará los marcadores de identificación ArUco ya empleados en [1] y [2]. Estos marcadores son de tipo rectangular y ofrecen varios diccionarios que presentan algunas ventajas dependiendo de los requerimientos y limitaciones del sistema de visión artificial.

La integración de la librería ArUco con OpenCV permite utilizar las funciones integradas para realizar el reconocimiento de los identificadores o marcadores de manera sencilla. Dado que las condiciones de trabajo para la aplicación del presente proyecto no presentan dificultades de visualización de los marcadores, cualquier diccionario de ArUco puede ser utilizado. Para los robots móviles se ha seleccionado el diccionario “Original ArUco”. [38]

A través de cada marcador, se realiza una estimación de posición en función de las coordenadas de las cuatro esquinas reconocida por la librería *aruco* de *OpenCV*. La Figura 1.5 muestra ejemplos de marcadores ArUco.

**Figura 1.5** Marcadores ArUco usados en la plataforma de pruebas.

**Red Xbee: topología y método de comunicación**

Como parte del esquema centralizado de control y monitoreo, cada robot recibe órdenes de un solo módulo coordinador que se encuentra conectado a la computadora del usuario. Este módulo envía de forma secuencial, a cada robot, los datos de velocidad que se obtienen del controlador de movimiento, sea este de formación, posición u orientación.

Por ser una plataforma de pruebas de una escala relativamente pequeña, la topología seleccionada es una configuración estrella, donde el nodo central es el módulo coordinador y los nodos periféricos son los módulos a bordo de cada robot.

Como parte del sistema de comunicación, y por requerir transmitir distintos datos desde el coordinador hasta cada robot, la comunicación se realiza a través de tramas API, las cuales permiten configurar un destinatario específico y un campo de datos que pueden ser empaquetados antes de ser transmitidos y desempaquetados por cada robot.

Las ventajas de usar comunicación Xbee en modo API, en lugar de transmitir datos de manera directa simulando una conexión serial son las siguientes [35]:

- Permite configurar cada módulo de la red en caso de requerir una modificación en sus parámetros.
- Identificación del módulo transmisor en cada trama de comunicación.
- Recepción de trama de reconocimiento de éxito o falla en la comunicación.
- Recepción del dato de intensidad de la señal RF en cada trama de comunicación.
- Permite realizar análisis de la red y diagnóstico del estado de la comunicación.
- Permite realizar actualizaciones de firmware remotas en caso de ser requeridas.

### 1.3.3 PLANIFICACIÓN DE MOVIMIENTO

En todo sistema de robots móviles se requiere algún método de navegación que permita al robot moverse de forma segura en un entorno determinado. El control

manual no es una forma viable de control dado que el objetivo de los sistemas robóticos es la automatización de procesos. La planificación de movimiento para agentes robóticos es un campo que ha estado en estudio durante mucho tiempo, por lo que muchas soluciones han sido propuestas dependiendo del tipo de robot, aplicación y características de los agentes robóticos.

La planificación de movimiento se compone principalmente de tres partes: el mapeo del entorno, la planificación de rutas y el control del robot para que siga la ruta planificada.

**Mapeo de entornos**

La planificación de movimiento requiere de manera esencial del mapeo del entorno del robot o agentes robóticos. Para esto, se pueden considerar varios métodos de mapeo dependiendo del sistema. En sistemas avanzados, el mapeo puede requerir ser tridimensional, teniendo en cuenta distancia, profundidad y dimensión de los obstáculos y objetos presentes en el entorno. En otros casos, un mapeo bidimensional del entorno es suficiente y puede ser implementado de manera más sencilla [25].

**Mapeo y posicionamiento simultaneo (SLAM)**

El Mapeo y posicionamiento simultaneo (SLAM) es un método de mapeo que consiste en recrear un mapa de un entorno desconocido a través de la combinación de información adquirida por diversos sensores a bordo de un robot [26]. SLAM es ampliamente usado por robots individuales que requieren conocer su entorno y navegar de forma autónoma, en caso de requerir conocer una gran área de determinado entorno. También, el uso de grupos de robots es una solución viable.

En el caso de sistemas multi-agentes, el tipo de agentes puede determinar qué solución de mapeo es la más acertada. En caso de contar con agentes que incluyan sensores como transductores ultrasónicos, sensores LIDAR [14], o detectores infrarrojos o cámaras, un mapeo basado en la detección de obstáculos de manera individual, como SLAM, puede ser implementado. Se debe tener en cuenta que en este caso, la planificación de rutas solo puede ser posible una vez se haya completado el mapeo de todo el entorno. En muchos casos, al utilizar agentes complejos con variedad de sensores, el mapeo y planificación de rutas son dos procesos que se realizan de manera separada. Por lo que, para crear el mapa de toda un área, la información recolectada por cada agente a través del conjunto de sensores a bordo, es combinada y ordenada para crear una versión virtual del entorno.

### Mapeo por visión artificial

El campo de visión artificial se ha desarrollado de manera acelerada en los últimos años y el análisis de imágenes en tiempo real es una tarea que se puede realizar cada vez de manera más rápida y eficiente. Esta velocidad de procesamiento para establecer un mapa del entorno abre las posibilidades de mapeo en tiempo real y permite una actualización rápida de los cambios en el entorno.

Justamente, un método de mapeo más complejo y eficiente puede ser realizado a través de visión artificial. El uso de cámaras permite una adquisición de datos mucho más completa pero que requiere de una mayor complejidad computacional. En la mayoría de los casos, una cámara estéreo es una solución para determinar la profundidad de los objetos presentes en el entorno. El uso de cámaras monoculares suele ser limitado a mapeo bidimensional de entornos. Uno de los beneficios de usar visión artificial como método de mapeo del entorno es que los datos son adquiridos de manera instantánea y se puede cubrir una gran área del entorno [27].

Si bien el uso de cámaras también es común en métodos de mapeo SLAM, estas pueden realizar un mapeo del entorno sin necesidad de combinar información de cada agente. Cámaras que se posicionen para capturar una extensa área de trabajo ofrecen un mapeo de forma directa e instantánea, permitiendo realizar la planificación de rutas de manera rápida.

### Recursos y herramientas computacionales

Existen numerosas herramientas computacionales dirigidas a solucionar y simplificar el problema del análisis de imágenes a través de visión artificial. El desarrollo de librerías completas orientadas a la manipulación y análisis de imágenes en tiempo real han llevado la complejidad de implementación de sistemas de visión artificial a niveles cada vez más bajos.

#### *OpenCV*

OpenCV es una librería de código abierto desarrollada desde 1999 como una iniciativa de investigación de Intel. A partir de 2006 se publicó la primera gran versión disponible a todo público. A través de los años, y con el soporte de la comunidad, la librería ha crecido de manera acelerada y hoy en día provee de más de 2500 algoritmos optimizados sobre visión artificial y aprendizaje automático. [28]

#### *Marcadores de referencia*

Los marcadores de referencia, como son los ArUco, son objetos usados ampliamente en sistemas de visión computarizada para obtener puntos de referencia y estimar posiciones, distancias y orientación. En algunas aplicaciones, son usados para realizar

estabilización de imagen con alta precisión [29]. En la Figura 1.6 se muestran distintos tipos de marcadores referenciales a bordo del robot Perseverance.

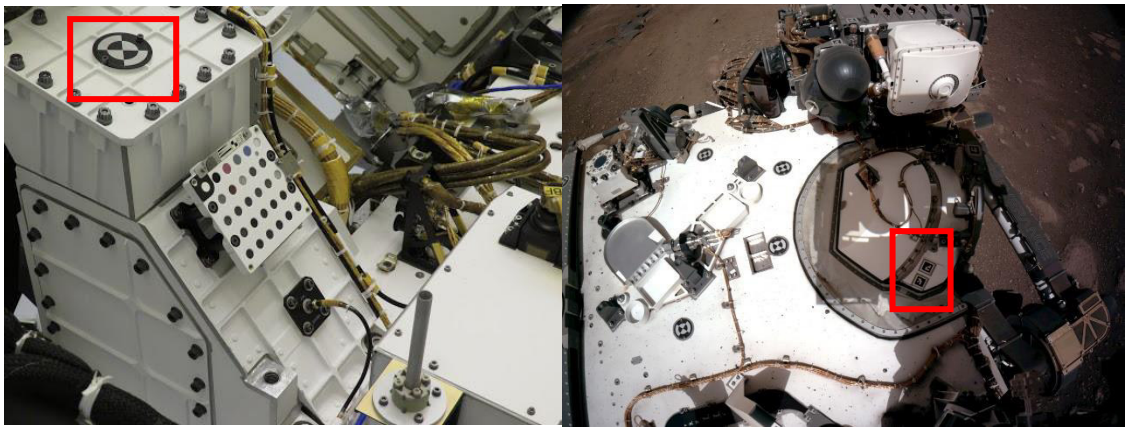


**Figura 1.6** Marcador de referencia a bordo del robot Perseverance [40].

Existen muchos tipos de marcadores, con distintas formas geométricas y basados en distintos conceptos de reconocimiento de imágenes. Existen marcadores circulares, amorfos y rectangulares. Los más comunes, por un gran margen, son los rectangulares, pues ofrecen cuatro esquinas que facilitan la estimación de orientación. Dependiendo de la aplicación, algunos marcadores pueden ser de mayor utilidad que otros. En la mayoría de las aplicaciones de visión artificial, marcadores rectangulares o cuadrados son más útiles pues pueden proveer información acerca de la distorsión presente en las imágenes [30].

**Planificación de rutas**

Una vez que se obtiene información sobre el entorno en el que los agentes van a desempeñar sus tareas, se requiere que cada uno de ellos pueda desplazarse de manera segura, sin comprometer la integridad física de sí mismos o de los demás robots. Justamente, la planificación de rutas es una herramienta basada en la ejecución de algoritmos que permiten determinar caminos por los que cada agente puede movilizarse. Estos caminos deben tener ciertas características para que sean considerados como soluciones viables; entre las principales se tienen:

- *Costo de operación*: El costo que requiera el completar la ruta puede ser entendido como la distancia que esta tiene desde el punto de origen hasta el destino. En la mayoría de los casos, se busca que el camino o ruta sea de la menor distancia posible, optimizando el tiempo de operación y la energía requerida. El obtener el camino más corto u óptimo, manteniendo un tiempo computacional relativamente bajo ha sido estudiado en varias ocasiones y se

han desarrollado distintos algoritmos que pretenden encontrar una solución viable de manera eficiente.

- *Tiempo computacional*: Si bien se pueden invertir muchos recursos computacionales en la determinación de una ruta para cada agente, siempre se busca que el tiempo computacional sea el mínimo posible. Por esto, se busca un balance entre la ruta generada y el tiempo computacional invertido. Dependiendo de la aplicación, ciertos compromisos pueden realizarse para obtener rutas adecuadas en un tiempo computacional relativamente bajo. El desarrollo de los algoritmos de planificación no solo se centra en la calidad del camino generado si no en la eficiencia con la que se logra determinar la solución.
- *Precisión y exactitud*: Dado que el objetivo principal de todos los algoritmos de planificación de rutas es llevar de un punto de partida a un punto de llegada a cada robot, la precisión y exactitud que se logre una vez completada la ruta son también parámetros por considerar. El objetivo en general es que el robot termine lo más cerca posible de la posición deseada. Existen algoritmos que siempre garantizan dicha característica y otros que garantizan llevar al robot lo más cera posible dependiendo de ciertos parámetros.

**Tipos de entornos**

El algoritmo que se utilice para la planificación de rutas dependerá en cierta medida del tipo de entorno que se requiere procesar [31]:

- *Entornos estáticos*: Son entornos que se consideran invariables durante la operación de las tareas del grupo de robots. Si bien la ubicación de cada agente puede cambiar, la ubicación de obstáculos y espacios transitables no será cambiante.
- *Entornos dinámicos*: Son aquellos espacios que pueden cambiar durante la operación del grupo de robots. La posición y orientación de obstáculos será dinámica por lo que en estos casos se necesita realizar un mapeo continuo o periódico para determinar los cambios y ajustar las rutas planificadas de manera acorde.

**Algoritmos de Planificación de Rutas**

Como parte del desarrollo de la robótica, el avance en algoritmos de planificación de rutas ha sido constante y progresivo. Uno de los primeros algoritmos desarrollados con el propósito de guiar a un robot móvil fue A*, desarrollado en la década de los años 60 como parte del proyecto del robot Shakey [22]

*Algoritmo A**

Es un algoritmo de búsqueda que busca el camino más corto desde un punto inicial hasta una meta. Se basa en el uso del costo directo y un parámetro heurístico adicional. No siempre garantiza una solución óptima pero en la mayoría de los casos logra encontrarla [5].

El parámetro heurístico que usa A* es la distancia mínima entre el nodo que se analiza y el nodo meta. Este parámetro es denominado una heurística admisible, el cual no debe en ningún caso sobreestimar el costo de alcanzar la meta.

El algoritmo se basa en el cálculo de un parámetro que califica a cada nodo. La ecuación (10) expresa dicha calificación expresada en función de los parámetros $g$ y $h$.

$$f(n) = g(n) + h(n) \tag{10}$$

Donde $h(n)$ representa el valor heurístico del nodo a evaluar desde dicho nodo hasta la meta. Es considerado como un parámetro optimista puesto que no sobreestima la distancia hasta la meta, esto se logra estimando la distancia más corta posible, usualmente la línea recta entre el nodo a evaluar y el nodo meta. Mientras que el término $g(n)$ es el costo real del camino recorrido y es usualmente expresado como la suma de distancias entre el nodo inicial hasta el nodo evaluado.

La función $f(n)$ es conocida como el mérito del nodo analizado, y expresa la probabilidad del nodo de estar en el camino más corto. Cuanto menor sea el mérito de un nodo, más probable es que este sea parte del camino más corto.

El funcionamiento de A* se basa en mantener una lista ordenada por valores crecientes del mérito de cada nodo analizado y se selecciona el nodo de menor valor, o primero de la lista ordenada.

El algoritmo inicia analizando el nodo origen o inicial, calcula su mérito y continua con analizar todos los nodos que se encuentren conectados a él. Una vez calculado $f(n)$ de cada nodo vecino, se seleccionará el que tenga menor valor de $f(n)$, y de ese nodo se analizaran a su vez sus vecinos, para elegir nuevamente el nuevo nodo que tenga menor valor de $f(n)$. El algoritmo analizará sucesivamente los nodos con menor mérito hasta llegar al nodo meta, cuando se tendrá la lista de todos los nodos con menor $f(n)$, los cuales comprenden la solución o camino más corto encontrado por el algoritmo.

Considerando el conjunto de nodos presentados en la Figura 1.7, donde el nodo inicial es A y el nodo meta el J, los números en rojo son el parámetro heurístico $h(n)$ y los números en azul son el costo directo nodo a nodo $g(n)$.

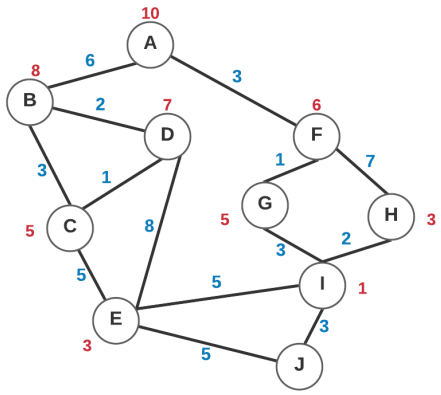


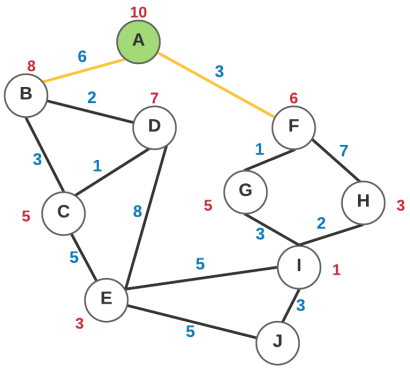

**Figura 1.7** Nodos, costo y valor heurístico en algoritmo A*

Se inicia analizando el nodo inicial A y se calcula $f(n)$ de sus vecinos. Este ejemplo se muestra en la Figura 1.8 (a).

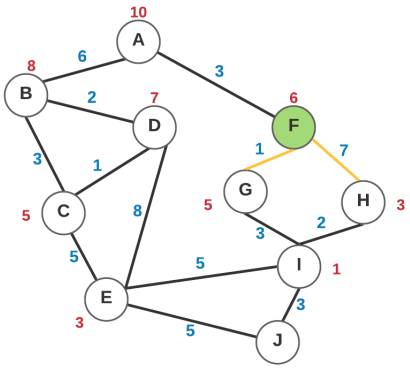
$$f(B) = 6 + 8 = 14$$

$$f(F) = 3 + 6 = 9$$

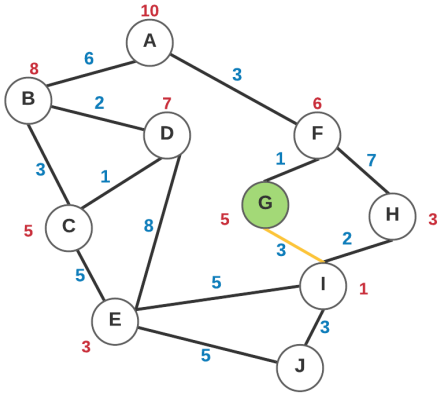
En base a esto, se selecciona el nodo con menor mérito. Dado que $f(F) < f(B)$ , el siguiente nodo a analizar será el nodo F, ver Figura 1.8 (b).



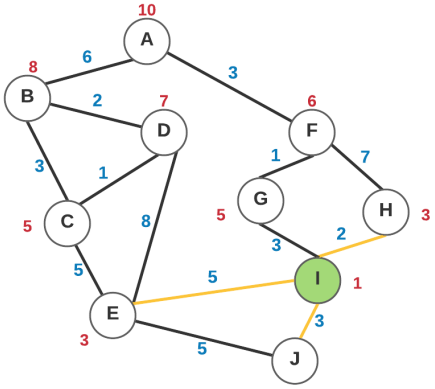

**(a)** Primera iteración - Ejemplo de A*




**(b)** Segunda iteración - Ejemplo de A*




**(c)** Tercera iteración - Ejemplo de A*




**(d)** Última iteración - Ejemplo de A*

**Figura 1.8** Iteraciones en algoritmo A*

$$f(G) = (3 + 1) + 5 = 9$$
$$f(H) = (3 + 7) + 3 = 13$$

De forma similar, se selecciona el nodo con menor mérito. Dado que $f(G) < f(H)$, el siguiente nodo a analizar será el nodo G, ver Figura 1.8 (c).

$$f(I) = (3 + 1 + 3) + 1 = 8$$

Dado que no existen más vecinos, el siguiente nodo a analizar será el nodo I. ver Figura 1.8 (d).

$$f(E) = (3 + 1 + 3 + 5) + 3 = 15$$
$$f(H) = (3 + 1 + 3 + 2) + 3 = 12$$
$$f(J) = (3 + 1 + 3 + 3) + 0 = 10$$

Se selecciona el nodo con menor mérito. Dado que $f(J) < f(H) < f(E)$, el siguiente nodo a analizar será el nodo J. Dado que J es el nodo meta, se detiene la ejecución y se presenta la solución encontrada en la Figura 1.9:

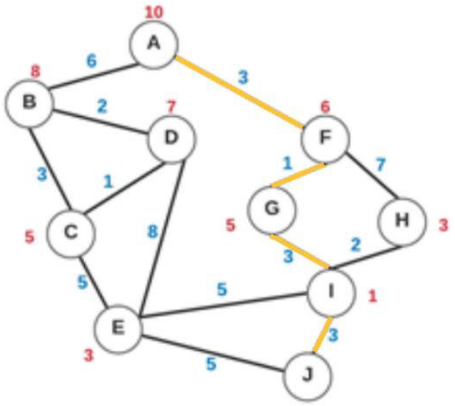



**Figura 1.9** Solución - Ejemplo de A*

Tras completar el algoritmo se obtiene la solución:

$$Ruta: A - F - G - I - J$$

Las mejores instancias donde este algoritmo puede ser una solución viable para hallar una ruta son aquellas en que se conoce el entorno, y se pueden determinar nodos que se interconecten entre sí de manera que el camino entre ellos pueda cubrir la mayor distancia posible. En casos en que el entorno no esté discretizado, este algoritmo puede ser aplicado solo luego de discretizarlo. En un entorno continuo, la discretización se puede realizar dividiendo al espacio en una malla, y considerando a cada cajón como un nodo [23]. Un ejemplo de esta discretización es mostrado en la Figura 1.10, donde los cajones en negro son obstáculos y los cajones blancos son espacio libre. El nodo A es el punto de partida y el nodo B es el punto de llegada.

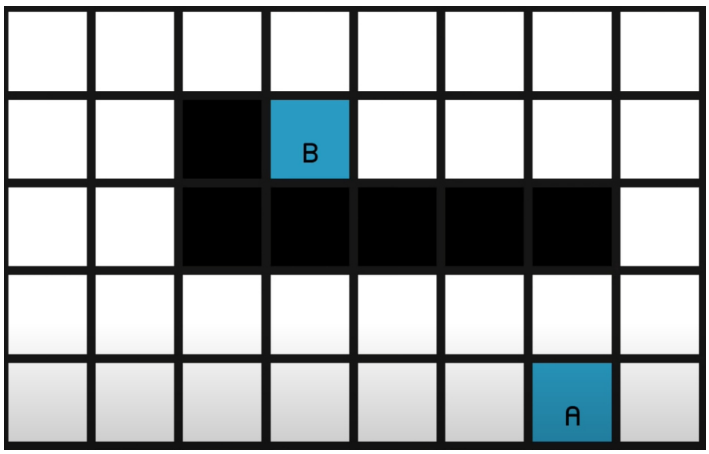



**Figura 1.10** A* aplicado en entornos discretizados en malla.

Una desventaja de este método, es que la precisión con la que se puede seleccionar un punto específico del espacio continuo depende del nivel de subdivisiones que se realicen en la malla. Si la malla consta de muy pocos cajones, un punto específico no podrá ser seleccionado como meta y el área cubierta por el cajón será una aproximación al punto meta deseado.

La solución obvia al mencionado problema es subdividir la malla en secciones extremadamente pequeñas, pero a cambio, se creará una cantidad exuberante de nodos, por lo que el tiempo computacional para encontrar una solución con A* será excesivamente alto.

*Mapas de rutas probabilísticas (PRM)*

PRM es un algoritmo basado en el muestreo aleatorio del entorno. Fue propuesto en [6] en 1998, donde se lo describe como un algoritmo que consta de una etapa de aprendizaje y una etapa de consulta. Este algoritmo plantea tomar muestras del espacio de trabajo y determinar si estas están en el espacio libre o en los obstáculos. Cuando se hayan creado suficientes nodos en el espacio de trabajo, estos se conectan entre aquellos que no se encuentren obstaculizados. Una de las principales ventajas de este algoritmo es que una vez creados todos los nodos, el entorno quedara descrito de manera que se puedan interconectar dos puntos cualesquiera del área muestreada a través de los nodos creados.

Dado que PRM genera una malla de nodos interconectados, suele ser acompañado por un algoritmo de menor nivel que analice los nodos creados para generar una ruta a través de estos. Algoritmos como A* o el algoritmo de Dijkstra [24] suelen ser usados con este fin.

Una consecuencia de crear nodos de manera aleatoria en todo el entorno, es que se genera un mapa de conexiones que puede ser consultado de manera consecutiva para hallar una ruta entre dos puntos. La Figura 1.11 [32] muestra el resultado de la aplicación de este algoritmo en un entorno con obstáculos. PRM es conocido como un algoritmo de consulta múltiple ya que es posible cambiar los puntos de partida y llegada y obtener una solución sin tener que generar la malla de nodos nuevamente.

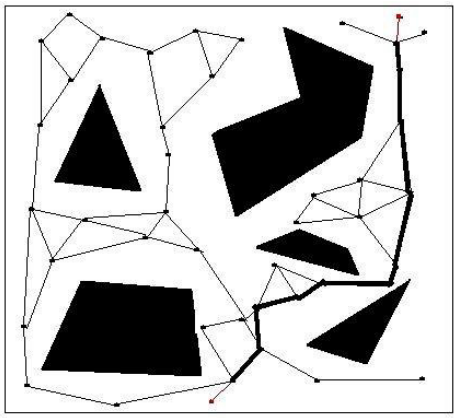


**Figura 1.11** Resultado del algoritmo PRM

Una de las desventajas del uso de este algoritmo es que, por ser de naturaleza aleatoria, dependiendo del entorno, se pueden llegar a requerir un alto número de nodos antes de que se pueda establecer la existencia de una ruta viable desde el punto de inicio hasta la meta. Además, existen ciertos tipos de entornos que pueden presentar retos muy complejos para este algoritmo, específicamente aquellos mapas que presenten corredores angostos o paso a través de aperturas pequeñas. Un entorno complejo para PRM se muestra en la Figura 1.12, donde no se ha logrado conectar todas las áreas del mapa dejando inconexos los puntos de partida y llegada.

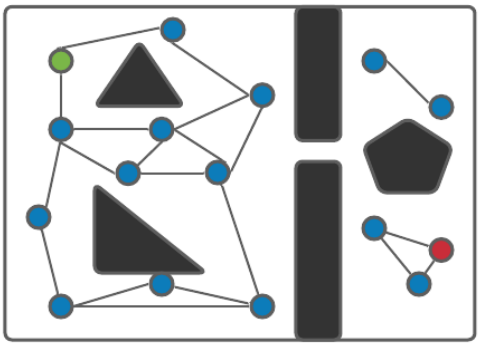


**Figura 1.12** Entornos complejos para algoritmo PRM

Para aliviar en cierta medida las desventajas de este método, se puede realizar un proceso mixto, donde la etapa de aprendizaje, donde se crean los nodos, y la etapa de consulta, donde se busca una ruta que conecte los nodos desde el inicio a la meta, se realizan de manera alternada. Con este método se pueden mapear y encontrar soluciones de manera parcial hasta llegar a la meta de un entorno extenso y complejo.

*Árboles aleatorios de exploración rápida (RRT*)*

Como parte de los algoritmos basados en muestreo, RRT* se caracteriza por generar estructuras en forma de árbol o ramas, las cuales se extienden a través del entorno.

Fue propuesto en [9] como variación al ya propuesto RRT [8]. Existen muchas alternativas propuestas basadas en RRT*, las cuales apuntan a solucionar algunas de las deficiencias que tiene el algoritmo. Algunas variaciones conocidas son RRT-Quick[12], RRT_Smart [10] e Informed-RRT [13].

La idea general de RRT es muestrear aleatoriamente el entorno, creando nuevos nodos y conectándolos al nodo más cercano. Los nodos creados se conectan en forma de ramas al nodo inicial y estas se extienden en el entorno. El entorno donde se aplique RRT no necesariamente debe estar discretizado, por lo que puede ser fácilmente aplicado en un espacio continuo.

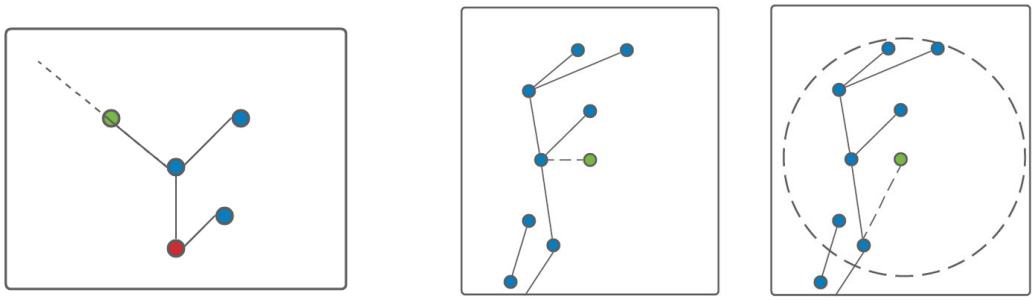
La Figura 1.13 (a) muestra el proceso de creación de los nuevos nodos. Cuando se agrega un nuevo nodo, se conecta al nodo más cercano solo si no existe un obstáculo entre ellos; en caso de estar obstaculizado, el nodo será ignorado y no se creará.

RRT* implementa dos pasos adicionales a RRT que permiten producir resultados muy distintos y caminos más directos. La primera diferencia, ejemplificada en la Figura 1.13 (b), es que se toma en cuenta el costo, o distancia entre cada nodo generado hasta el nodo padre, al cual está conectado. Una vez hallado el nodo más cercano al nuevo nodo aleatorio, se analiza la vecindad del nuevo nodo para hallar el nodo con menor costo. Si existe un nodo con menor costo que el nodo más cercado, este será asignado como el padre del nuevo nodo.

La segunda diferencia en RRT* se muestra en la Figura 1.13 (c). Consiste en la adición de una rutina de reconexión del árbol. Luego de que el nuevo nodo haya sido conectado al nodo padre (que presentó el menor costo), los nodos vecinos son nuevamente analizados para revisar si se pueden reconectar al nuevo nodo y reducir su costo. En caso de que, reconectándolos al nuevo nodo, su costo disminuya, se eliminaran las antiguas conexiones y se reconfigurara esa sección del árbol para mantener un camino más suave y directo.

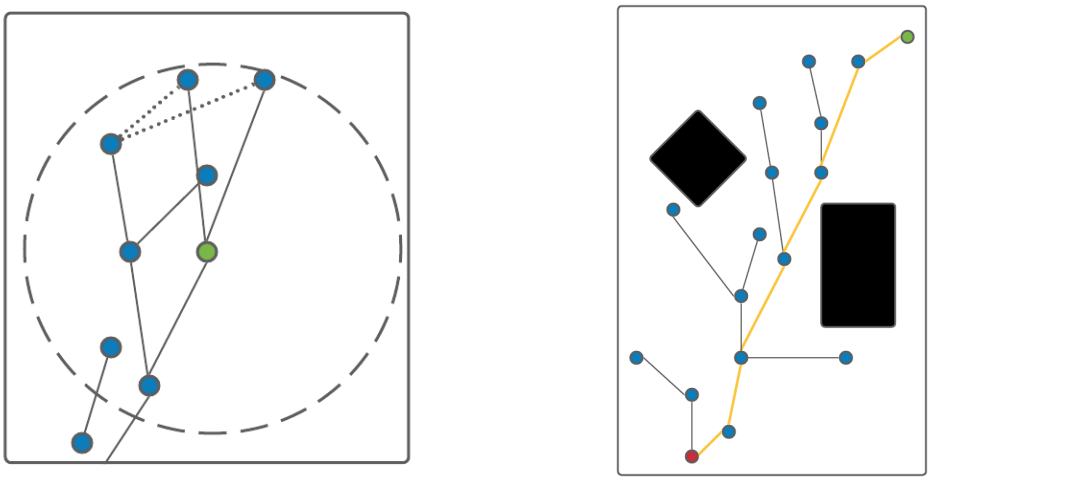
Una vez encontrado un nodo a una distancia aceptable del punto meta, la solución será el conjunto de nodos que lleven lleguen el nodo inicial hasta el nodo final con el menor costo. La solución ejemplificada se muestra en la Figura 1.13 (d)

Una consecuencia de los pasos agregados en RRT* es el aumento de tiempo computacional invertido en analizar la vecindad de nodos y reconfigurar el árbol. Adicionalmente, se debe revisar que ningún obstáculo se encuentre entre cada nuevo nodo creado y el nodo padre al que será conectado [9].



**(a)** Proceso de creación de un nuevo nodo en RRT*

**(b)** Conexión del nuevo nodo al nodo vecino que presente menor costo o mérito



**(c)** Reconexión de los nodos vecinos para mantener el menor costo.

**(d)** Resolución de la secuencia de nodos que presente el menor costo.

**Figura 1.13** Pasos más importantes en algoritmo RRT*

Ya que el árbol crece a partir del nodo inicial, el algoritmo puede proporcionar una solución a casi cualquier ubicación en el mapa seleccionada como meta. Esta posibilidad se muestra en la Figura 1.14. RRT y RRT* no son considerados como algoritmos de consulta múltiple ya que en caso de que la posición inicial cambie, el árbol generado no será de utilidad.

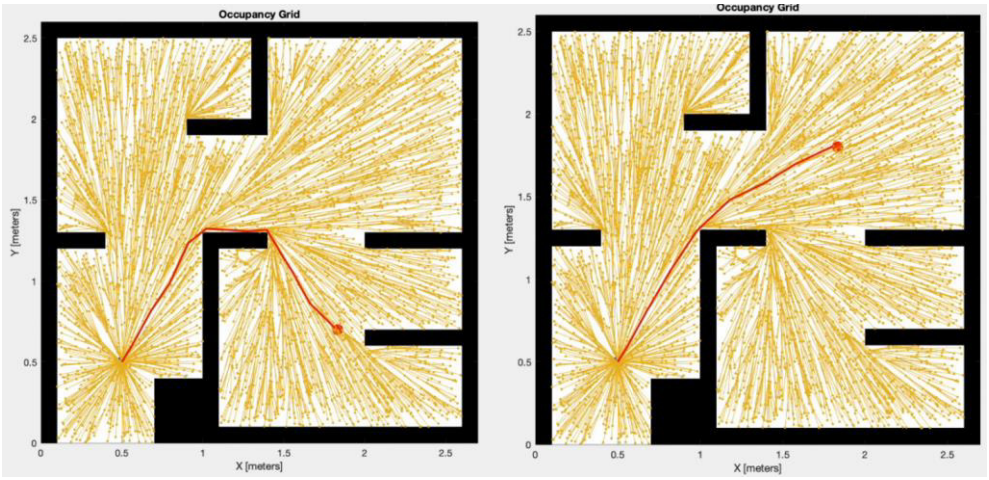



**Figura 1.14** Resultado de RRT* para diferentes puntos meta en el mismo entorno

Si el número de nodos generados se aproxima al infinito, el algoritmo proporcionará una solución asintóticamente estable. De manera práctica, los requerimientos de la aplicación y la capacidad computacional del sistema serán los limitantes para el número de nodos que se puedan generar para buscar una solución. [9]

El algoritmo es capaz de encontrar una solución asintóticamente óptima con el número de nodos dispuestos. Como se mencionó, un mayor número de nodos siempre mejorará la solución, haciéndola más corta y suave, como se observa en la Figura 1.15.

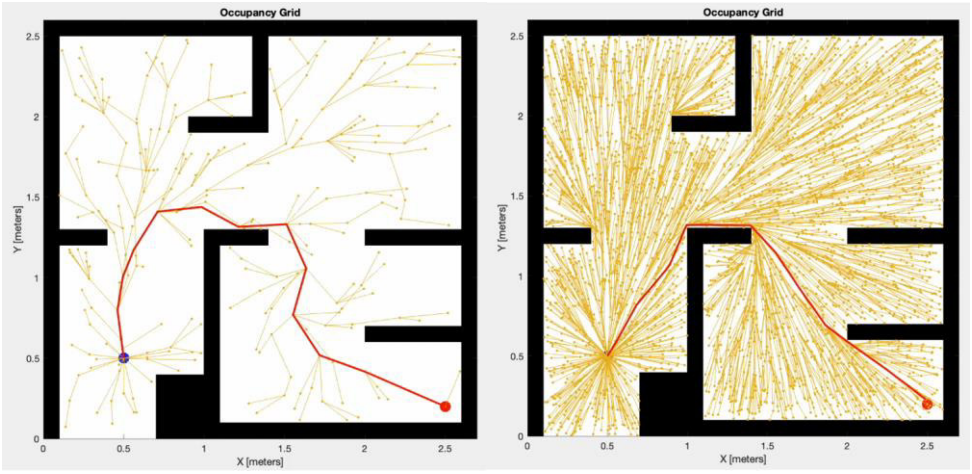


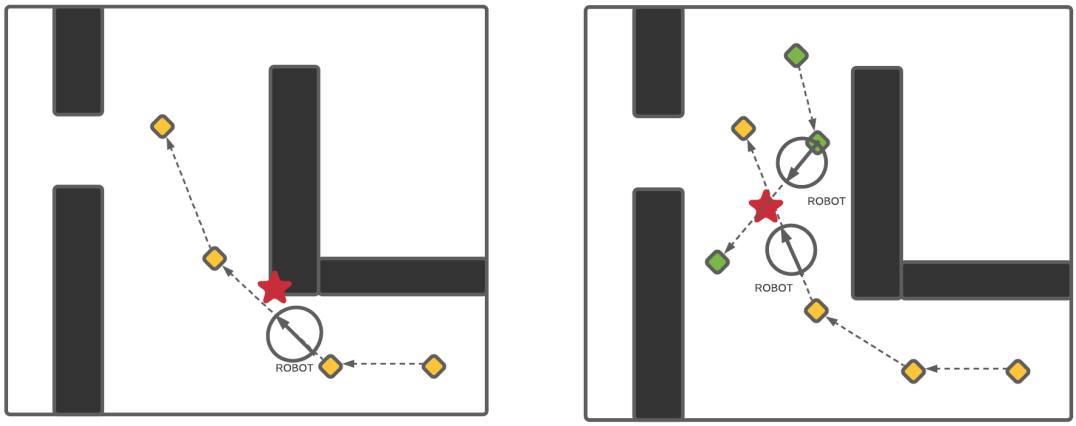

**Figura 1.15** Resultados de RRT* con distinto número de nodos.

**Evasión de obstáculos en la planificación de rutas**

Como parte de la planificación de rutas, la evasión de los obstáculos presentes y detectados en el proceso de mapeo es esencial para producir soluciones que no contemplen colisiones entre agentes y obstáculos.

Para que un algoritmo de planificación de rutas cumpla su función de manera exitosa, dicha planificación debe realizarse contemplando los obstáculos presentes en el entorno. La ruta generada como solución debe siempre evitar colisiones entre los robots y los obstáculos. En la Figura 1.16 (a) se muestra un ejemplo de colisión entre el robot y un obstáculo como resultado de la generación de una ruta que no contempla la geometría del robot.

Cuando se utiliza un algoritmo de planificación de rutas, dicha evasión de obstáculos es parte de los pasos del algoritmo para determinar una ruta viable. Existen otros casos, en los que la evasión de obstáculos debe realizarse de manera dinámica y en línea, principalmente cuando se consideran entornos dinámicos con obstáculos móviles. En algunas aplicaciones, es necesario también implementar algún tipo de algoritmo o reglas que prevengas colisiones entre agentes, dado que cada uno debe ser considerado como un obstáculo para los demás. En la Figura 1.16 (b) se muestra un caso de colisión entre dos robots moviéndose en un entorno estático.




(a) Colisión de ruta con obstáculo.

(b) Colisión entre rutas de robots.

**Figura 1.16** Posibles colisiones en un entorno con obstáculos

### 1.3.4 Interfaces de usuario sobre Linux

El desarrollo de toda aplicación requiere de alguna interfaz de usuario que permita la modificación de parámetros, el comando del sistema y el monitoreo de este. En la mayoría de los casos, las aplicaciones que son desarrolladas sobre ROS pueden ser monitoreadas desde una consola del computador. A pesar de ser un método funcional, el monitoreo y comando a través de consolas no es siempre práctico. Con el propósito de facilitar el uso de la aplicación, y hacer al sistema más amigable y organizado para los usuarios, existen herramientas que facilitan la creación de interfaces de usuario. La Figura 1.17 muestra la diferencia entre la interfaz de usuario diseñada en el presente trabajo y la consola utilizada en caso de no contar con una interfaz gráfica.


**Figura 1.17** métodos de interacción del usuario con el sistema.

**Qt Designer**

Qt Designer es una aplicación de código abierto con herramientas gratuitas e intuitivas. Se basa en la creación de interfaces a través del posicionamiento de elementos de manera gráfica sobre un lienzo. Provee de varios widgets que integran funcionalidades comunes como botones pulsantes, etiquetas de texto, menús de selección, barras de edición de texto y más. Los archivos generados con Qt Designer son de extensión .ui, por lo que su uso puede ser adaptado a través de otras herramientas, convirtiendo el archivo a programación en distintos lenguajes de programación [33].

**PyQt5**

La herramienta Pyqt5 ofrece varias funcionalidades orientadas a la creación de interfaces de usuario basadas en Python. Es un conjunto de librerías que implementa múltiples funciones de una interfaz de usuario. Además, a través de su herramienta Pyuic5, permite convertir archivos con extensión .ui a archivos escritos en lenguaje Python, permitiendo así usar las interfaces creadas en Qt Designer en programas basados en Python.

Pyqt5 no requiere de archivos creados en Qt Designer para implementar interfaces. Se pueden escribir en Python interfaces de usuario sin ningún archivo previamente generado [34].

En el presente trabajo, se utilizará Qt Designer para generar un archivo .ui, posteriormente se usará la herramienta Pyuic5 para convertir y generar un archivo de extensión .py. La implementación de las interfaces de usuario se realizará sobre cada nodo ROS referenciando como una clase externa Python a las interfaces generadas previamente.

# 2 METODOLOGÍA

El presente trabajo se ha realizado mediante investigación descriptiva y experimentación práctica como métodos de exposición y validación. Se han utilizado diversas fuentes bibliográficas, entre las que se encuentran artículos científicos, trabajos de titulación y material académico relevante. La metodología del presente trabajo se ha dividido en cuatro partes:

A. *Fase teórica*: Se presenta una revisión bibliográfica acerca de los algoritmos de planificación de rutas, así como de temas importantes para la comprensión e implementación de las funciones establecidas en los objetivos del proyecto. Adicionalmente, se presenta una recopilación de información acerca de las herramientas utilizadas.

B. *Fase de diseño*: Consiste en la conceptualización del sistema propuesto para cumplir con los objetivos de reconocimiento, mapeo y planificación de rutas, así como el diseño de las interfaces de usuario requeridas para el control del sistema. Incluye además, el diseño de todos los módulos necesarios para el funcionamiento de la plataforma de pruebas.

C. *Fase de implementación*: Consiste en la elaboración y programación de los módulos propuestos, además de la configuración del sistema inalámbrico de comunicación. Esta etapa incluye la adición de un robot móvil al grupo de robots disponibles en la plataforma. Finalmente, se agrega la integración total del sistema, incluyendo las interfaces de usuario.

D. *Fase de análisis de resultados*: Mediante la ejecución de pruebas de funcionamiento, se evalúa el estado final del sistema y el desempeño de los nodos implementados. Se realiza también un análisis de los resultados de manera cuantitativa y cualitativa, respecto a la operación de los robots en la plataforma de pruebas.

## 2.1 ADICIÓN DE UN ROBOT MÓVIL

Como se indicó en la introducción, el continuo crecimiento de la plataforma de pruebas es parte de los objetivos del presente proyecto. Uno de los aspectos básicos de la plataforma es el número de robots móviles de tracción diferencial disponibles para realizar pruebas, implementar algoritmos y probar nuevas funcionalidades. Como parte de esta nueva iteración del proyecto, se ha agregado un robot adicional, lo que lleva a cuatro el total de robots disponibles.

Un mayor número de robots en la plataforma de pruebas abre las puertas a la posibilidad de implementación de algoritmos más complejos y a la incursión en nuevos ámbitos de investigación y estudio. Existen múltiples aplicaciones que requieren y/o aprovechan la disponibilidad de múltiples agentes para realizar una tarea. Sea esto permitiendo dar redundancia al sistema, cubriendo áreas de trabajo más extensas o realizando tareas que requieran un mayor número de agentes.

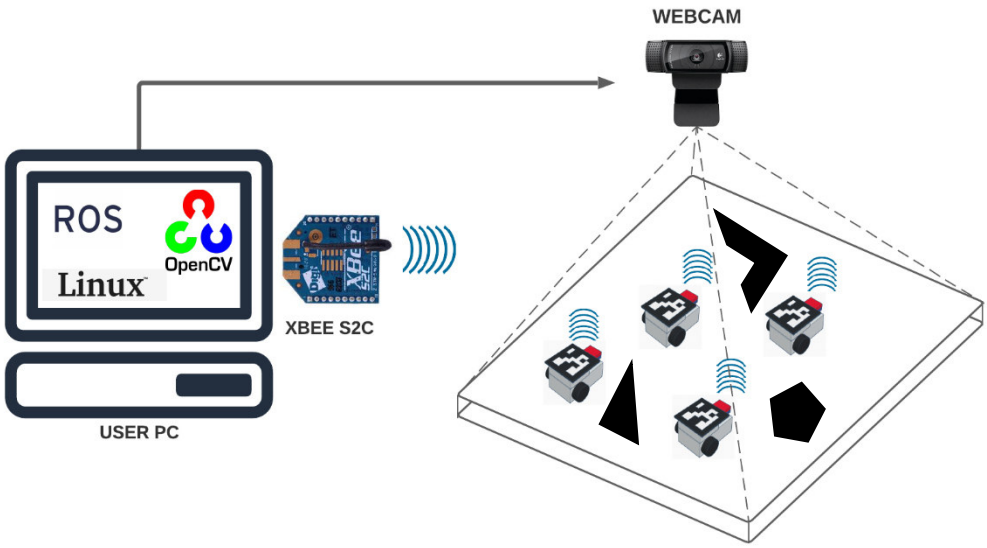



**Figura 2.1** Arquitectura completa del sistema con cuatro robots móviles y con obstáculos presentes en el área de trabajo.

Las actualizaciones realizadas sobre el diseño de los robots en [2], además de permitir la implementación del controlador interno de velocidad en cada robot, fueron suficientes para poder mantener el diseño y componentes de los robots en la actual iteración de la plataforma de pruebas.

El cuarto robot agregado al grupo robótico se compone de los mismos elementos que los otros tres ya existentes, manteniendo consistencia y equidad en los requerimientos del controlador respecto a cada robot. No se ha requerido modificar la forma de controlarlos y al contar con igualdad de características en todos ellos ha permitido diseñar programas de control que funcionen de manera uniforme para todos los robots, simplificando la implementación de nuevas funcionalidades en la plataforma. La arquitectura del sistema, con la adición del cuarto robot móvil se puede observar en la Figura 2.1.

El detalle de los componentes individuales, así como de la estructura física de los robots móviles puede ser encontrado en el trabajo realizado en [2], específicamente en la Sección 2.1 de dicha referencia.

## 2.2 SELECCIÓN DEL ALGORITMO DE PLANIFICACIÓN DE RUTAS

De los algoritmos revisados en el Capítulo 1 se ha seleccionado el algoritmo RRT* como el mejor candidato para ser implementado como primer paso hacia aplicaciones que requieran esta funcionalidad. Una de las principales razones por las que se seleccionó RRT* es su adaptabilidad, a través de modificaciones menores en el código, a necesidades específicas de aplicaciones futuras. Como se señaló en la introducción, existen varios algoritmos basados en RRT* [11] que mejoran aspectos puntuales del mismo, los cuales podrán ser implementados en base al código escrito para el presente trabajo.

### 2.2.1 Algoritmo RRT*

A continuación se presenta pseudocódigo del algoritmo RRT* aplicado en el presente trabajo.

| | $\boldsymbol{Algoritmo\ RRT*}$ | | |
|---|---|---|---|
| 1 | $V \leftarrow \{x_{init}\}$ | ► | El primer vértice se denomina $x_{init}$. |
| 2 | $E \leftarrow 0$ | | |
| 3 | $\boldsymbol{for}\ i \leftarrow 1 \ldots N$: | ► | $x_{rand}$ corresponde al punto seleccionado de manera aleatoria. |
| 4 | $x_{rand} \leftarrow SampleFree(i)$ | | |
| 5 | $x_{nearest} \leftarrow Nearest((V,E), x_{rand})$ | | |
| 6 | $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$ | ► | $x_{new}$ corresponde al nuevo nodo creado. |
| 7 | $\boldsymbol{if}\ ObstacleFree(x_{nearest}, x_{new})$: | | |
| 8 | $X_{near} \leftarrow Near((V,E), x_{new}, r_n)$ | ► | La función $Near$ determina el nodo más cercano al nuevo nodo. |
| 9 | $V \leftarrow V \cup \{x_{new}\}$ | | |
| 10 | $x_{min} \leftarrow x_{nearest}$ | | |
| 11 | $c_{min} \leftarrow Cost(x_{nearest}) + c(Line(x_{nearest}, x_{new}))$ | | |
| 12 | $\boldsymbol{for\ each}\ x_{near} \in X_{near}$: | | |
| 13 | $\boldsymbol{if}\ CollisionFree(x_{near}, x_{new})$: | | |
| 14 | $\boldsymbol{if}\ Cost(x_{near}) + c(Line(x_{near}, x_{new})) < c_{min}$: | ► | La función de costo corresponde a la |
| | $x_{min} \leftarrow x_{near}$ | | |

| | | | |
|---|---|---|---|
| 15 | $c_{min} \leftarrow Cost(x_{near}) + c\big(Line(x_{near}, x_{new})\big)$ | | distancia entre todos los nodos de la ruta. |
| 16 | $E \leftarrow E \cup \{(x_{min}, x_{new})\}$ | | |
| 17 | $\boldsymbol{for\ each}\ x_{near} \in X_{near}$: | | |
| 18 | $\boldsymbol{if}\ CollisionFree(x_{new}, x_{near})$: | | |
| 19 | $t \leftarrow Cost(x_{new}) + c\big(Line(x_{new}, x_{near})\big)$ | ► | |
| 20 | $\boldsymbol{if}\ t < Cost(x_{near})$: | | En caso de hallar un nodo vecino con menor costo, este se convierte en el nodo padre. |
| 21 | $x_{parent} \leftarrow Parent(x_{near})$ | | |
| 22 | $E \leftarrow (E \backslash \{(x_{parent}, x_{near})\}) \cup \{(x_{new}, x_{near})\}$ | | |
| 23 | $\boldsymbol{return}\ (V, E)$ | | |

En la línea 4 del algoritmo, la función *SampleFree* retorna un set de coordenadas dentro del espacio libre de manera aleatoria, mientras que la función *Nearest* (línea 5) retorna el nodo más cercano al nodo creado aleatoriamente.

La función *Steer* (línea 6) determina la posición del nuevo nodo, de manera que si $x_{rand}$ se encuentra a una distancia mayor a un valor predeterminado épsilon, el nuevo nodo se crea a dicha distancia de $x_{nearest}$. De lo contrario $x_{rand}$ será $x_{new}$.

En la línea 7, *ObstacleFree* retorna un valor booleano si existe o no obstáculos entre $x_{nearest}$ $y$ $x_{new}$. *Near* retorna el conjunto de nodos dentro un radio $r_n$ alrededor del nodo $x_{new}$. La función *Cost* retorna la distancia entre un nodo y el punto inicial del árbol.

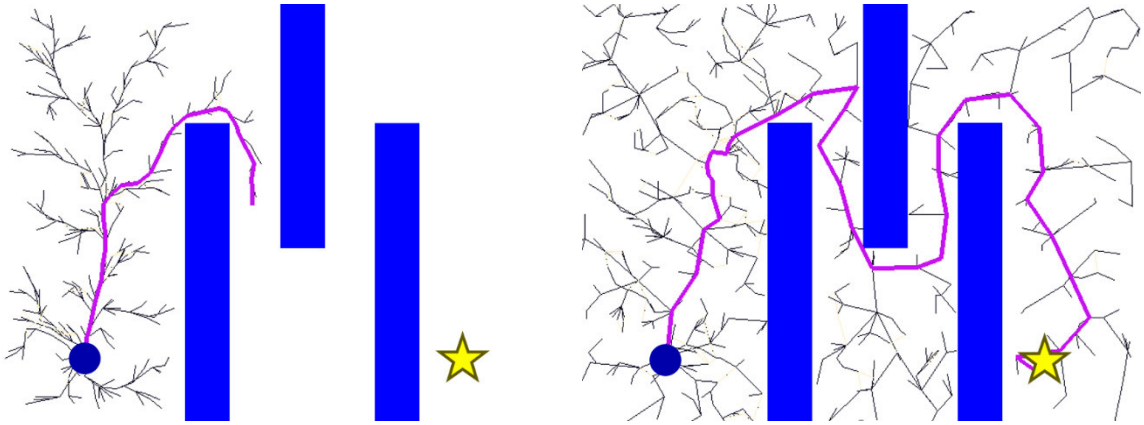
La función *c,* utilizada en la línea 11, retorna el costo o distancia entre dos nodos conectados con una arista, mientras que *Line* retorna una arista recta conectando dos nodos. *CollisionFree* (línea 13) retorna un valor booleano si existe colisión entre dos nodos. Finalmente, *Parent* (línea 21) asigna a un nodo la propiedad de contar con una conexión o arista a un nodo previamente existente.

El resultado del algoritmo es el conjunto de nodos que conectan el camino más corto entre el punto de partida y el punto de llegada. Dichos nodos están representados por aristas $(E)$ y vértices $(V)$, los vértices siendo las coordenadas del nodo y las aristas la conexión entre nodos.

Como se mencionó en el detalle del algoritmo, existen tres parámetros que afectan el desempeño del algoritmo y que deben ser ajustados dependiendo de las condiciones del entorno: distancia máxima de conexión $(\varepsilon)$, radio de búsqueda $(R)$ y número de nodos totales $(N)$. La distancia máxima de conexión determina que tan extensas serán

las aristas que conecten un par de nodos. El radio de búsqueda determina el área analizada para realizar la conexión y reconfiguración del árbol. El número de nodos totales determina cuantas iteraciones se realizarán en el algoritmo, en cada iteración se creará un nodo.

En consecuencia, el comportamiento del árbol generado y la forma de la solución final dependerá de los parámetros mencionados anteriormente. Un valor grande de ε, en relación con el entorno, resultará en caminos con tramos más extensos, lo que puede resultar en una solución con menos puntos intermedios. Por otra parte, cuando ε es configurado con un valor muy pequeño, se generan tramos más cortos, resultando en un mayor número de nodos o iteraciones requeridas para alcanzar el punto meta.



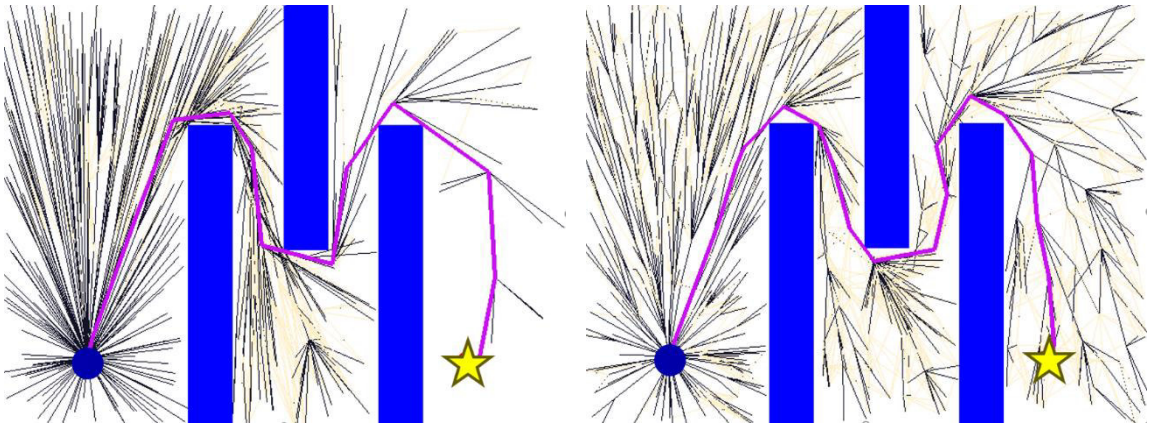
(a) Resultado obtenido con ε=10 y $R$=20 (b) Resultado obtenido con ε=50 y $R$=20

**Figura 2.2** Comparación de resultados de RRT* con distintos valores de ε

La Figura 2.2 muestra una comparación del resultado obtenido con dos valores de ε, manteniendo el mismo valor de $R$ y de $N$. Se puede observar que cuando ε es muy pequeño, el árbol no se logra extender hacia toda el área del entorno. Adicionalmente, con un ε mayor se observa el efecto de tener tramos más cortos y menos puntos intermedios en la ruta final.

Los efectos del radio de búsqueda ($R$) se muestran en la conformación y estructura del árbol resultante. Cuando se configura un valor de $R$ muy alto, las ramas del árbol adyacentes al nodo inicial estarán siempre conectadas al mismo, produciendo una estructura que desaprovecha la naturaleza de exploración del árbol a través de sus ramas. Por lo que el árbol generado consistirá en amplias aristas rectas que conecten distintos puntos del entorno pero no se contarán con ramas que curven a través de los obstáculos presentes. Por otro lado, un valor pequeño de $R$ permite al árbol crecer envolviendo a los obstáculos de manera más eficiente, creando ramas que llegan a

explorar más áreas del entorno. La Figura 2.3 muestra los resultados obtenidos con distintos valores de $R$.



(a) Resultado obtenido con $R$ =160 y ε=80 (b) Resultado obtenido con $R$ =80 ε=80

**Figura 2.3** Comparación de resultados de RRT* con distintos valores de $R$

Adicionalmente, existe un una relación entre los valores de ε y $R$. Esta interacción se puede observar en la Figura 2.2(b), en la cual un valor pequeño de $R$, respecto a ε, resulta en una estructura similar al resultado obtenido con el algoritmo RRT, en lugar del RRT*. En sí, se pierde la capacidad de reconfiguración del árbol ya que el radio de búsqueda es muy pequeño y no alcanza a incluir los nodos cercanos al nodo generado aleatoriamente.

Comprender la incidencia del valor de $N$ es más sencillo puesto que de manera intuitiva se puede inferir que un mayor número de nodos generados resultará en la creación de un árbol más completo y una solución más cercana a una respuesta óptima. A pesar de esto, como se mencionó anteriormente, el valor de $N$ también debe mantener coherencia con el valor configurado de ε. Finalmente, se debe tomar en cuenta que un mayor número de nodos totales significa un mayor tiempo computacional de ejecución del algoritmo. Por esto, dependiendo de la aplicación, en algunos casos puede ser más deseable un menor tiempo computacional versus una solución óptima.

## 2.3 SISTEMA DE CONTROL Y MONITOREO

Como se ha mencionado, toda la funcionalidad de la plataforma, respecto al sistema de control y monitoreo, se basa en ROS. La Figura 2.4 muestra un diagrama de bloques que incluye las tareas necesarias para mantener la funcionalidad propuesta en el presente proyecto.

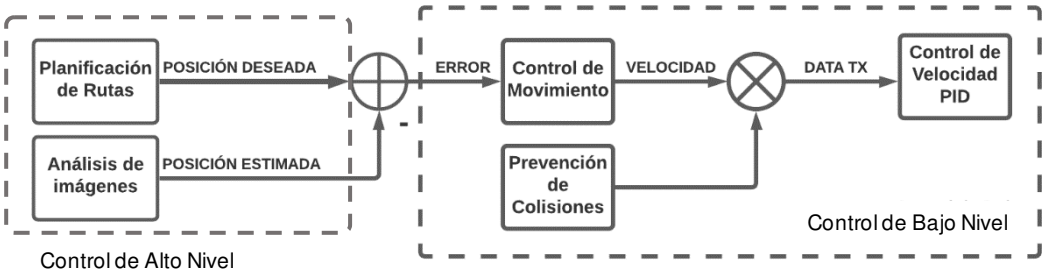



**Figura 2.4** Esquema de control de la plataforma de pruebas.

Como se puede observar, la plataforma cuenta con dos bloques: Planificación de rutas y Análisis de Imágenes, que actúan como un controlador de alto nivel. En esencia, ambos trabajan de manera conjunta en la planificación de rutas y monitoreo del sistema. La salida de dichos bloques se convierte en la entrada para los bloques que controlan el movimiento de los robots. Estos bloques consisten en un Controlador de Movimiento y un sistema de Prevención de Colisiones que se encarga de evitar posibles choques entre robots. Una vez obtenidos los datos de velocidad lineal y angular para cada robot, se convierte dichos datos a valores de velocidad en revoluciones por minuto (rpm) de cada llanta para cada agente. Estos últimos datos se envían a través de la red de comunicación Xbee hacia cada robot, donde el control PID interno de velocidad se encarga del accionamiento de los motores de acuerdo con los datos de velocidad.

Para la implementación práctica de cada bloque, varios nodos ROS fueron programados, cada uno cumpliendo funciones esenciales, de manera modular para permitir la modificación de las funcionalidades del sistema sin afectar toda la plataforma.

### 2.3.1 Topología e Interacción de los Nodos ROS

Un esquema de la topología e interacción de los nodos ROS se encuentra ilustrado en la Figura 2.5.

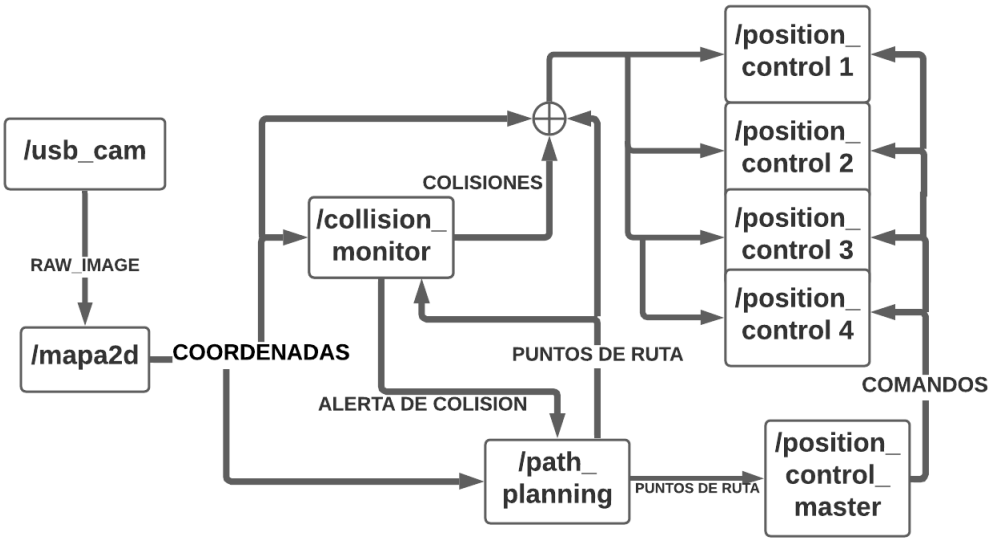



**Figura 2.5** Topología e interacción de los nodos ROS.

**Nodo de adquisición de datos**

El nodo /usb_cam forma parte de la librería integrada de ROS [36] dedicada a la interconexión entre ROS y cámaras de tipo USB. Este nodo permite adquirir las imágenes capturadas por la cámara USB, además de ofrecer varios parámetros de configuración como exposición, brillo, contraste, saturación, resolución y número de cuadros por segundo.

Los parámetros configurados en el sistema para el presente proyecto son:

- Resolución: 1280x720p
- Formato: mjpeg
- Enfoque: 0 (Enfoque a distancia infinita)
- Brillo: 130
- Saturación: 120
- Contraste: 120
- Nitidez: 120
- Auto exposición: True
- Cuadros por segundo: 30

Finalmente, el nodo /usb_cam publica un mensaje ROS tipo “Image” al tópico “image_raw”, al cual está suscrito el nodo de mapeo.

**Nodo de mapeo**

Lleva el nombre de /mapa2d y se encarga del reconocimiento y análisis de las imágenes publicadas por el nodo /usb_cam. Específicamente, realiza tres tareas relacionadas con el mapeo del entorno de trabajo de manera bidimensional:

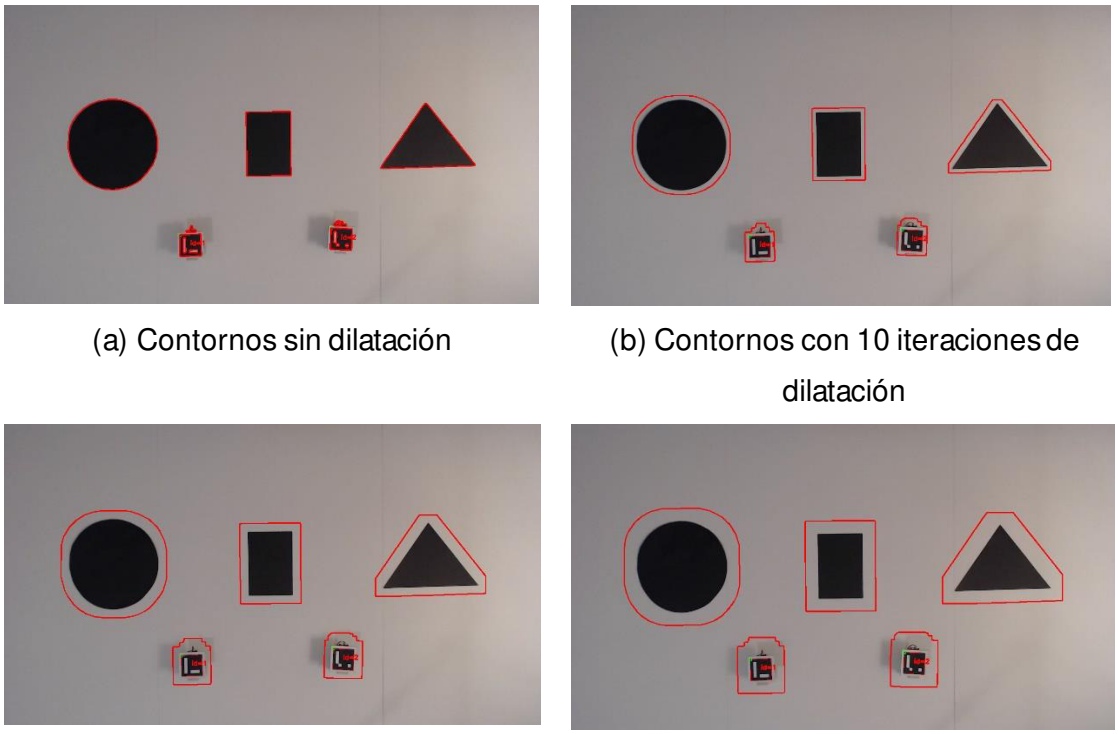
- Reconocimiento de marcadores ArUco.
- Identificación de las coordenadas de posición (además de su orientación) de cada marcador.
- Identificación de las coordenadas de los vértices de cualquier objeto que sea colocado sobre el entorno, el cual será reconocido como un obstáculo.

Dado que el análisis de imágenes se realiza en base a las formas geométricas y colores presentes en el entorno de trabajo, para mejorar el desempeño de este nodo, se utilizan figuras geométricas de colores oscuros sobre un fondo claro. A mayor diferencia de contraste entre el espacio libre y los obstáculos, más fácil será realizar el mapeo en distintas condiciones lumínicas.

Las coordenadas de los marcadores ArUco, así como las coordenadas de los obstáculos presentes se guardan en un archivo JSON para ser utilizados de forma asíncrona por los demás nodos del sistema. Por otro lado, el análisis de imágenes se realiza en línea con el uso de la librería OpenCV, la cual es utilizada para realizar la detección de marcadores ArUco y la detección de contornos de los obstáculos presentes. En este nodo se calcula también la orientación en radianes y grados de cada robot.

Como se mencionó en el primer capítulo, el mapeo del entorno debe tomar en cuenta las características físicas (área o volumen) de los robots para que el algoritmo de planificación de rutas no genere caminos que produzcan colisiones con los obstáculos al recorrer cerca de ellos. Por esto, como parte del análisis de imágenes y reconocimiento de contornos, se realiza un proceso de dilatación de las figuras consideradas como obstáculos. La dilatación de las formas geométricas asegura que el área inmediatamente adyacente a estas no sea considerada como espacio libre, garantizando que ninguna ruta que produzca colisiones sea generada.

La Figura 2.6 muestra un ejemplo de distintos valores de dilatación sobre las figuras identificadas por OpenCV.



(a) Contornos sin dilatación

(b) Contornos con 10 iteraciones de dilatación

(c) Contornos con 20 iteraciones de dilatación

(d) Contornos con 30 iteraciones de dilatación

**Figura 2.6** Dilatación de contornos con OpenCV.

Finalmente, el nodo de mapeo publica dos mensajes ROS que son utilizados por los demás nodos. Primeramente, un mensaje tipo “String”, indicando la actualización de los archivos JSON que contienen las coordenadas de marcadores y obstáculos y un mensaje tipo “CompressedImage” que publica una imagen sobre la cual se dibujan recuadros que indican los obstáculos reconocidos y los marcadores identificados. El mensaje “CompressedImage” corresponde a los datos de las imágenes procesadas por OpenCV que se trasmiten entre nodos ROS para hacer uso de las mismas en diferentes nodos, específicamente los nodos que ejecutan las interfaces de usuario, donde las imágenes procesadas necesitan mostrarse al usuario.

**Nodo de Planificación de Rutas**

Este es uno de los nodos más importantes en el sistema, lleva el nombre de /path_planning y se encarga de calcular las rutas para cada robot, tomando en cuenta los datos proporcionados por el nodo de mapeo.

Sobre este nodo está implementado el algoritmo descrito en la Sección 2.2.1 basado en el código propuesto por [15]. Respecto a dicho código se realizaron varios cambios necesarios para integrar el algoritmo a las necesidades de la plataforma de pruebas, teniendo en cuenta los requerimientos de los demás nodos.

Justamente para facilitar la observación de la evolución del algoritmo, se utilizó la librería gráfica Pygame [37] que permite dibujar los nodos y aristas del árbol creado sobre el entorno adquirido por el nodo de mapeo. La Figura 2.7 muestra la ventana generada por la librería durante el cálculo de las rutas RRT*.

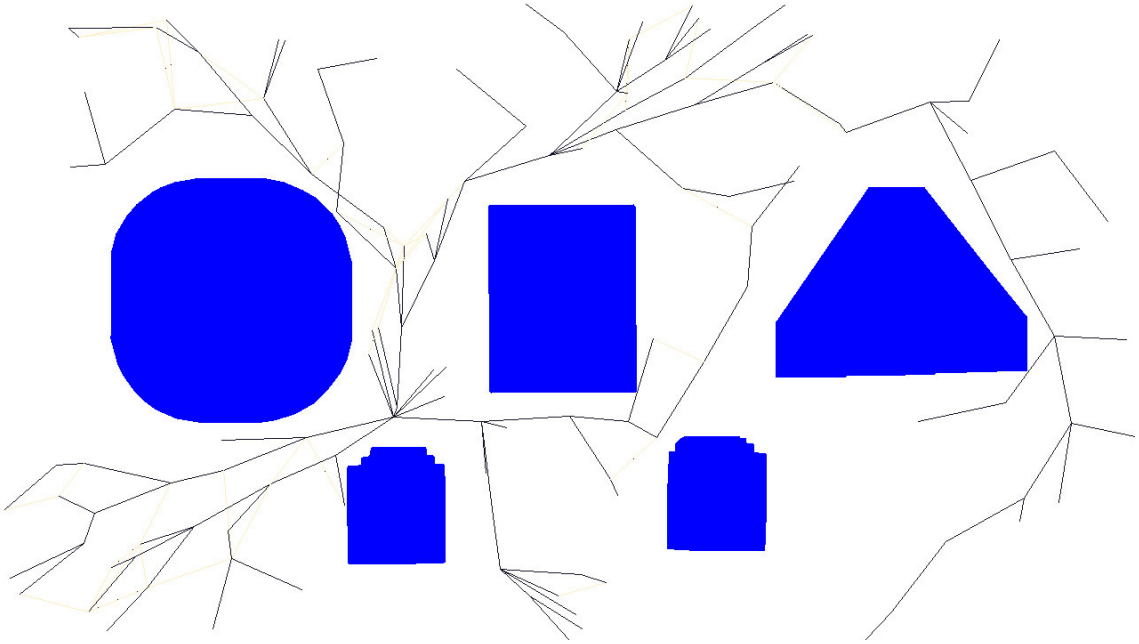
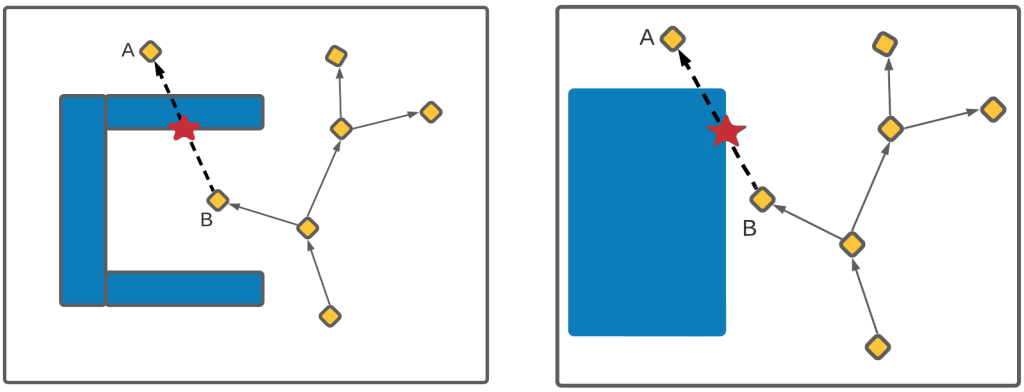


**Figura 2.7** Ventana de Pygame generada para visualizar la evolución del algoritmo RRT*.

Puesto que del nodo de mapeo se adquieren las coordenadas de los vértices de los obstáculos, y teniendo en cuenta que estos pueden ser de cualquier forma geométrica, la determinación de la existencia de obstáculos entre dos nodos se puede llegar a complicar al contar con obstáculos cóncavos. Justamente para facilitar este paso, se implementó una rutina de análisis gráfico, comparando el color de los pixeles que conforman la arista entre dos nodos y los pixeles que conforman los obstáculos.




(a) Ejemplo de obstáculos cóncavos entre dos nodos

(b) Ejemplo de obstáculos convexos entre dos nodos

**Figura 2.8** Análisis de obstáculos entre dos nodos.

La Figura 2.8 ejemplifica los casos en que se requiere analizar los puntos entre los nodos A y B con obstáculos cóncavos y convexos. Cuando se consideran únicamente figuras convexas, el análisis de la existencia de obstáculos entre dos nodos se facilita utilizando funciones que evalúan si una recta interseca un polígono definido por las coordenadas de sus vértices. En el caso de contar con figuras cóncavas, dichas funciones dejan de ser útiles puesto que la definición a través de los vértices no es suficiente para determinar la forma real de la figura. La Figura 2.9muestra un ejemplo en que un polígono definido por sus vértices puede ser interpretado de distintas formas, caso en el que resulta complejo utilizar definiciones matemáticas para realizar la determinación de obstáculos entre dos puntos.

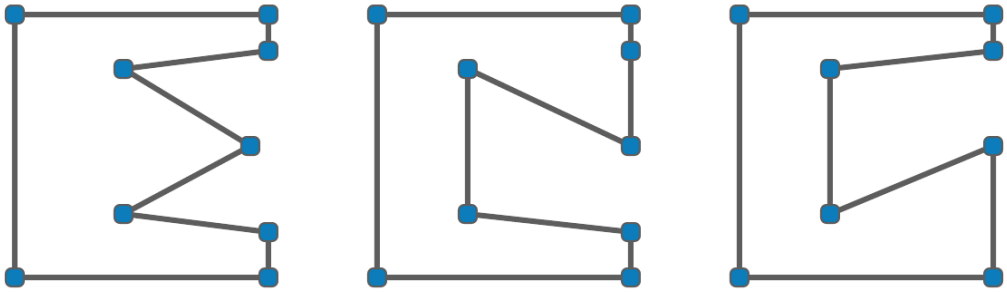


Figura 2.9 Ejemplo de interpretaciones de obstáculos definidos por sus vértices.

La rutina de análisis gráfico mencionada anteriormente evalúa cada pixel de la recta AB y lo compara, respecto a su color, con los pixeles que conforman los obstáculos, que siempre se identifican con color azul. En caso de que un pixel de la recta AB sea de color azul, se habrá encontrado un obstáculo entre A y B, como se muestra en la Figura 2.10.

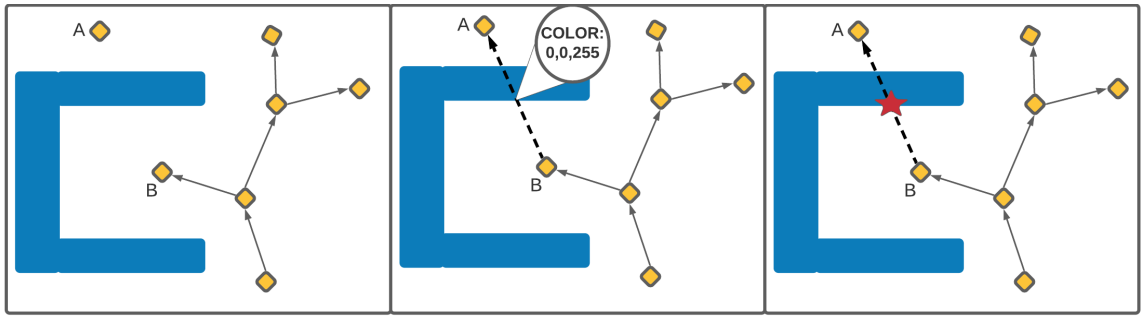



Figura 2.10 Rutina de análisis grafico para detección de obstáculos entre dos nodos.

Adicionalmente, el tipo de datos utilizados en el algoritmo fue transformado a valores de tipo coordenadas, equivalentes a los pixeles de la imagen recogida por el nodo de mapeo. Este cambio facilitó el uso directo de los datos de mapeo sin necesidad de realizar escalamientos para ser usados en el cálculo de generación de rutas.

Dado que se requiere calcular una ruta por cada robot presente, el algoritmo requiere ser ejecutado de manera simultánea múltiples veces. Para esto se implementó un método de ejecución en modo multiprocesamiento (multiprocessing). Python incluye este módulo que permite ejecutar procesos de manera paralela, con espacios independientes de memoria para cada proceso. De esta manera, el cálculo de las rutas para cada robot se puede ejecutar sin necesidad de esperar el término de la anterior.

Finalmente, tras encontrar una ruta para cada robot, el nodo las almacena, dentro de un archivo JSON, en forma de coordenadas ordenadas. Se publica también un mensaje ROS de tipo “String” que indica que un nuevo set de soluciones ha sido publicado.

Como parte de la topología de los nodos implementados, se pude observar que este nodo también recibe mensajes provenientes del nodo de prevención de colisiones. Dicho nodo analiza las soluciones tras cada actualización y envía un mensaje de alerta en caso de que se detecten cruces entre las rutas.

**Nodo de Prevención y Monitoreo de colisiones.**

Lleva el nombre de /collision_monitor y se encarga principalmente de monitorear el estado del sistema respecto a la distancia existente entre robots. Se ejecuta de manera paralela a los nodos de control y recibe los datos publicados por el nodo de mapeo.

La tarea de monitorear colisiones se realiza a través de la creación de dos círculos virtuales que engloban a cada robot a manera de una burbuja. Ambos círculos son utilizados como áreas que indican si cualquier par de robots se encuentran demasiado cerca y necesitan detenerse. En caso de que una colisión haya sido detectada, el nodo publica un mensaje ROS con los números de identificación de los robots que necesitan ser detenidos, este mensaje será utilizado por el nodo de Control de Posición para enviar la instrucción de detener al robot que sea requerido.

Finalmente, como se mencionó en la descripción del nodo anterior, este nodo también cumple la función de analizar y detectar cruces entre las rutas generadas por el nodo de planificación. En caso de que un cruce haya sido detectado, se publicará un mensaje ROS con los números de identificación de los robots que presentan una potencial colisión.

Se identificaron tres escenarios en los que un par de robots podrían evitar una colisión; así, se implementaron un conjunto de reglas en base a los círculos virtuales, las cuales establecen el comportamiento adecuado en base a las condiciones de ambos agentes.

**Colisión frontal-perpendicular**

Se produce cuando ambos robots se dirigen en rutas perpendiculares y de manera frontal. Este tipo de colisiones pueden ser evitadas deteniendo a uno solo de los robots para permitir el paso del otro. En el nodo de Prevención de colisiones se ha establecido una regla de jerarquía en base al número identificador de cada marcador ArUco. Así, el robot con menor número de identificación tendrá el derecho de paso. La Figura 2.11 ilustra la mencionada colisión entre un par de robots y el comportamiento programado.


**Figura 2.11** Colisión de tipo frontal-perpendicular.

La detección de este tipo de colisión se realiza monitoreando la intersección de los dos círculos frontales de los robots. Resumiendo lo indicado, si solamente los círculos frontales de un par de robots se intersecan, aquel con menor jerarquía se detendrá.

**Colisión posterior**

Se ocasiona cuando dos robots transitan uno detrás de otro y en la misma dirección. Cuando el robot que transita por detrás mantiene una velocidad mayor a la del robot que transita por delante, una colisión posterior es inminente.

Justamente para evitar esta colisión se requiere detener a aquel robot que transite por detrás del robot que mantiene una velocidad menor. En este caso, ninguna jerarquía requiere ser aplicada. La Figura 2.12 muestra este tipo de colisión y el comportamiento programado.

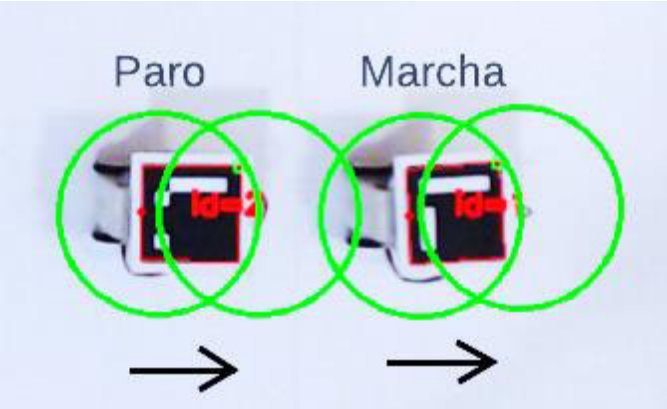



**Figura 2.12** Colisión posterior

La detección de este tipo de colisiones se realiza monitoreando la intersección del círculo posterior y frontal de cada robot. En caso de que exista esta intersección, aquel robot cuyo círculo frontal se encuentre involucrado se detendrá.

**Colisión lateral**

Se produce cuando un robot se acerca por el costado a otro robot. Por la naturaleza de este tipo de colisiones, existen múltiples ocasiones en las que una colisión puede o no ser evitada, dependiendo de las condiciones y de las rutas que se encuentren siguiendo los robots. Específicamente, la evitabilidad de la colisión depende del ángulo de incidencia con la que un robot se acerca al otro. En caso de ser un ángulo cercano a 90 grados, es posible evitar la colisión con la regla especificada para colisiones frontales perpendiculares, de lo contrario, cuando el ángulo de incidencia disminuye, la posibilidad de prevenir un choque y reanudar la operación aumenta en complejidad. Dada esta dificultad de predicción, en estos casos se ha determinado que ambos robots involucrados se detengan, dejando a criterio del usuario la continuación o no de la ruta de uno u otro agente. La Figura 2.13 muestra una colisión lateral y el comportamiento programado.

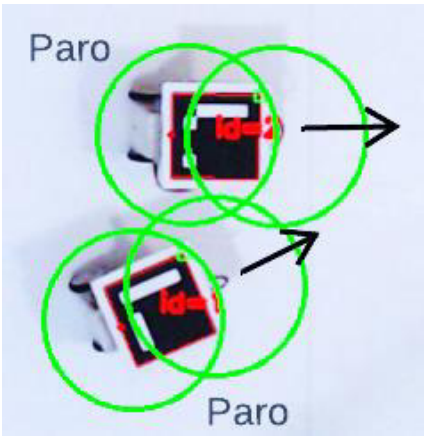



**Figura 2.13** Colisión lateral

Este tipo de colisión es detectada en caso de que uno de los círculos de un robot interseque con ambos círculos de cualquier otro robot. En ese caso, ambos agentes se detendrán.

**Nodo Master de Control**

Este nodo cumple principalmente la función de ejecutar la interfaz gráfica que permite al usuario comandar a los robots de manera individual. El detalle de la funcionalidad de dicha interfaz se explica en la siguiente Sección 2.3.2.

De forma simultánea junto a este nodo, se ejecutan 4 nodos ROS encargados del control de los robots. El nodo master y los 4 nodos de control se ejecutan al mismo tiempo a través de un archivo ROS tipo "launch".

**Nodos de control de posición**

La ejecución de estos nodos es transparente para el usuario, puesto que se ejecutan sin la necesidad de mostrarse en un terminal o consola. Se ejecutan al mismo tiempo que el nodo master a través de un archivo "launch" que los inicializa con distintos parámetros para que cada nodo solo comande a un solo robot.

Se crearon dos nodos de control, uno para cada uno de los tipos de controladores considerados en este proyecto: un control PID actuando sobre la orientación de cada robot y un control de Cinemática Inversa que actúa sobre la distancia entre el robot y su posición deseada.

**Control PID**

El primer control utilizado consiste en un control PID que actúa sobre la orientación de cada robot, reduciendo el error de ángulo que existe entre la orientación del robot y cada una de las coordenadas que requiere seguir para completar la ruta calculada. Una vez que se ha reducido el error de orientación a un valor aceptable, que incluye cierto margen de tolerancia, se comanda al robot transitar en línea recta con una rapidez constante. Cuando el robot haya llegado a una coordenada de su ruta, la nueva orientación hacia una nueva coordenada requerida será considerada como referencia y el control PID entrará en acción nuevamente. Para su implementación en el nodo de control se utilizó un módulo Python [39] que incluye su forma discretizada, así como una rutina anti-windup.

$$u_i = K_p\, e_i + K_i\, (I_{t-1} + e * Tiempo_{iteracion}) + K_d \frac{e - e_{t-1}}{Tiempo_{iteracion}} \qquad (11)$$

La Ecuación (11) describe el controlador PID en su forma clásica. Donde $u_i$ representa la acción de control de cada robot y $e_i$ corresponde al error de orientación para cada

robot. La acción de control del PID corresponde a un valor de velocidad angular. El termino $I_{t-1}$ representa el valor del término integral en la iteración anterior del controlador. Las ganancias $K_p, K_d\, y\, K_i$ del controlador fueron ajustadas de manera heurística tras realizar varias pruebas de funcionamiento. Estos resultados serán presentados más adelante en el Capítulo 3.

Como parte del control PID, la rutina anti-windup evita que el término integral aumente de forma descontrolada, estableciendo un valor máximo y mínimo que este puede tomar durante la ejecución del controlador. Así, en caso de que el término integral rebase dicho limite, su valor será limitado en cada iteración.

Adicionalmente, $Tiempo_{iteracion}$ corresponde al tiempo de muestreo utilizado en la rutina del controlador. En la implementación de la plataforma de pruebas, el controlador se actualiza en intervalos regulares de tiempo que están determinados por el tiempo computacional que toma al sistema enviar los datos de velocidad a los robots y calcular sus orientaciones instantáneas. Específicamente, el cálculo de $Tiempo_{iteracion}$ se realiza en cada iteración del controlador, midiendo el tiempo transcurrido entre la última iteración y el inicio de la nueva iteración. Más adelante se detallarán los tiempos transcurridos a través de pruebas de latencia, para conocer los intervalos de tiempo que existen en el nodo ROS de control.

Tras el cómputo de la señal de control requerida, el valor de rapidez angular y lineal es convertido a valores de Revoluciones por minuto (RPM) correspondientes a cada llanta de cada robot, para esto se utilizan las ecuaciones (7) y (8).

**Control basado en Cinemática Inversa**

Se basa en hallar la velocidad lineal y angular necesaria, para que cada robot llegue a un punto determinado, a partir de las ecuaciones cinemáticas que describen al robot. La implementación de este controlador aprovecha la simplificación del modelo cinemático a través del desplazamiento del punto de control detallado en la ecuación (9). En base a la misma, conociendo que la matriz Jacobiana del sistema es:

$$J = \begin{bmatrix} \cos(\phi) & -a\sin(\phi) \\ \sin(\phi) & a\cos(\phi) \end{bmatrix} \tag{12}$$

Se puede definir una matriz de error:

$$E = \begin{bmatrix} x_r - x \\ y_r - y \end{bmatrix} \tag{13}$$

donde $(x_r, y_r)$ es la coordenada de la posición deseada, y la coordenada $(x, y)$ es la posición actual del robot.

Como parámetros de sintonización, se incluye un término de ganancia que permite modificar el comportamiento del controlador, el cual viene dado por:

$$K = \begin{bmatrix} K_g & 0 \\ 0 & K_g \end{bmatrix} \tag{14}$$

donde $K_g$ es un valor positivo.

La ecuación 15 describe la ley de control aplicada:

$$\begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} \cos(\phi) & -a\sin(\phi) \\ \sin(\phi) & a\cos(\phi) \end{bmatrix}^{-1} \begin{bmatrix} K_g & 0 \\ 0 & K_g \end{bmatrix} \begin{bmatrix} x_r - x \\ y_r - y \end{bmatrix} \tag{15}$$

De esta manera se obtiene directamente los valores de $v$ y $w$ , los cuales corresponden a las rapideces lineal y angular respectivamente. Nuevamente, igual que en el caso anterior, estos valores se transforman a velocidad de rotación en Revoluciones por Minuto (RPM) para ser enviados a los robots.

### 2.3.2 Interfaz Gráfica de Usuario

A pesar de que todas las aplicaciones implementadas sobre ROS pueden ser controladas y monitoreadas desde un terminal, no siempre es la forma más práctica al diseñar una aplicación compleja que requiere formas más intuitivas de monitoreo y control. Justamente para facilitar el uso de la aplicación diseñada para la plataforma de pruebas, se han diseñado dos interfaces gráficas que permiten al usuario comandar y monitorear el sistema.

**Interfaz gráfica de Usuario (GUI) de planificación de rutas**

Se encuentra implementada sobre el nodo de Planificación de Rutas, el cual además de ejecutar el algoritmo RRT*, corre todas las funciones de la interfaz.

Esta permite al usuario modificar los parámetros con los que se ejecutará el algoritmo y determinar a qué robot será asignada la solución encontrada. Los parámetros que puede modificar el usuario son el número de nodos, el radio de búsqueda y la distancia máxima de conexión entre nodos. La selección del punto de salida y llegada se pueden seleccionar con el cursor, marcando el punto de interés sobre el espacio libre en el entorno.

Una vez finalizado el proceso de cálculo de la ruta, el usuario podrá consultar el tiempo de ejecución y la distancia que existe entre el punto final de la solución encontrada y el punto meta originalmente seleccionado por el usuario. La interfaz descrita se observa en la Figura 2.14.

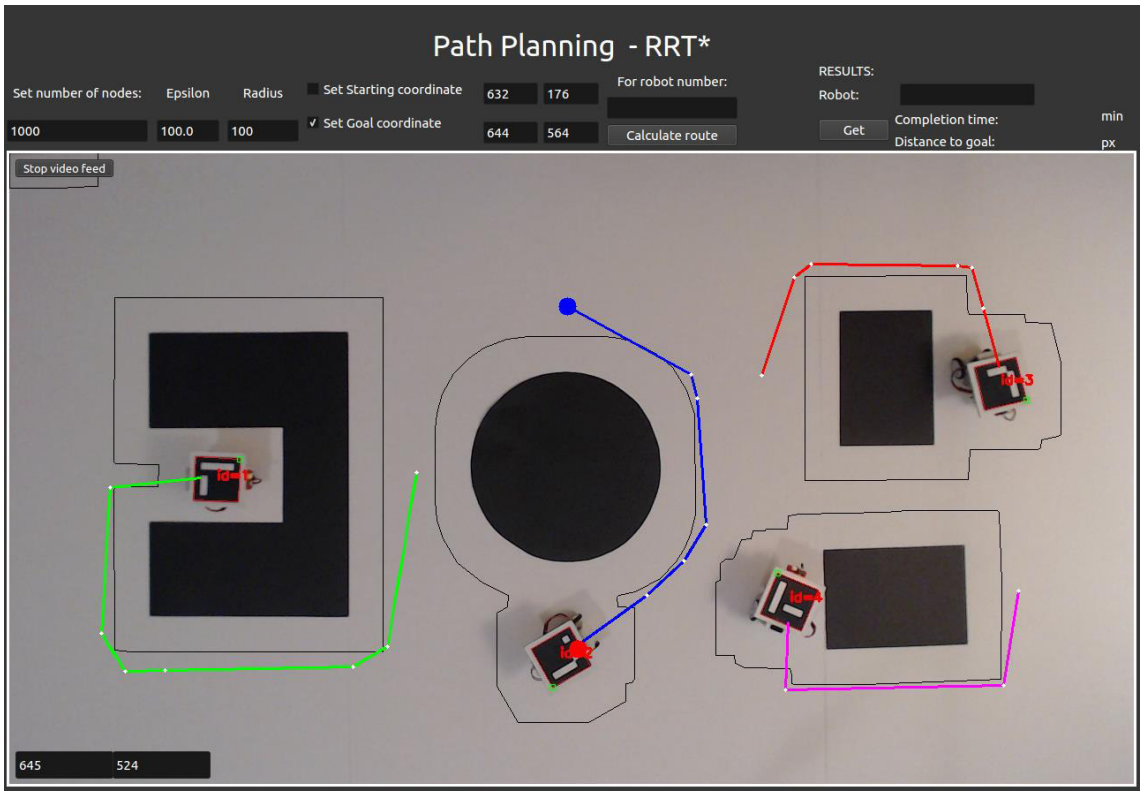



**Figura 2.14** Interfaz de usuario para el algoritmo de navegación RRT*.

**Interfaz gráfica de Usuario (GUI) de Control de Posición**

Se encuentra implementada sobre el nodo master de control, el cual principalmente se encarga de ejecutar la misma. Una vez calculadas las rutas para cada robot, se requieren controles sencillos para comandar a cada uno de los agentes. De esta manera, la interfaz cuenta con botones de inicio y paro para cada robot, además de un botón de anular un comando de paro enviado por el nodo de prevención de colisiones.

Como función adicional, se cuenta con control manual, en caso de requerir mover a los robots sin necesidad de calcular una ruta previamente. Este control manual se ejecuta a través de comandos ingresados por teclado, con las teclas “A,W,S,D” y “Shift”, con las cuales se determina el movimiento lineal y rotacional del robot, así como el paro del mismo. Una vez terminado el movimiento de un robot se despliegan las gráficas de resultados, donde el usuario podrá revisar a detalle el desempeño del controlador utilizado y el recorrido del robot. La interfaz descrita se aprecia en la Figura 2.15.

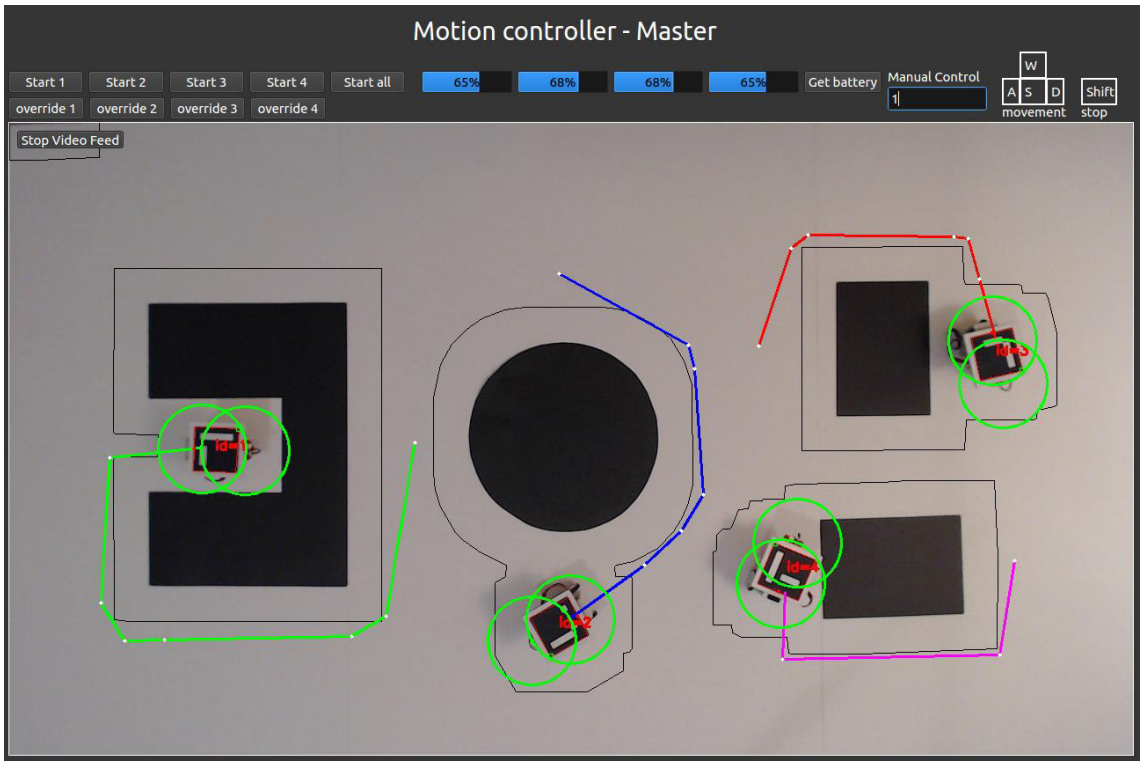



**Figura 2.15** Interfaz de usuario para el control de movimiento de los robots móviles.

Un manual sobre la ejecución de la aplicación presentada el este trabajo y el uso de las interfaces diseñadas se puede encontrar en el Anexo A.

## 2.4 RED DE COMUNICACIÓN

La plataforma de pruebas funciona de manera centralizada, con un computador que realiza todo el procesamiento de monitoreo y control. Particularmente, el sistema de comunicación inalámbrica se ha implementado con una topología tipo estrella, con un módulo central al que se conectan todos los robots presentes en el área de trabajo. Como se indicó anteriormente, el sistema de comunicación está basado en módulos Xbee S2C [41], de manera que cada robot cuenta con un módulo directamente conectado a la tarjeta Fio v3, mientras que el módulo central se conecta al computador a través de un adaptador serial USB.

### 2.4.1 Configuración de módulos

Cada módulo se ha configurado utilizando el software XCTU [42] provisto de manera gratuita y pública por el fabricante Digi International, sobre el cual se ha utilizado el firmware Zigbee TH REG para todos los módulos.

Adicionalmente, existen configuraciones comunes a todos los módulos del sistema, como el número de identificación de red y la velocidad de transmisión serial. Esta última configuración se muestra en la Figura 2.16.

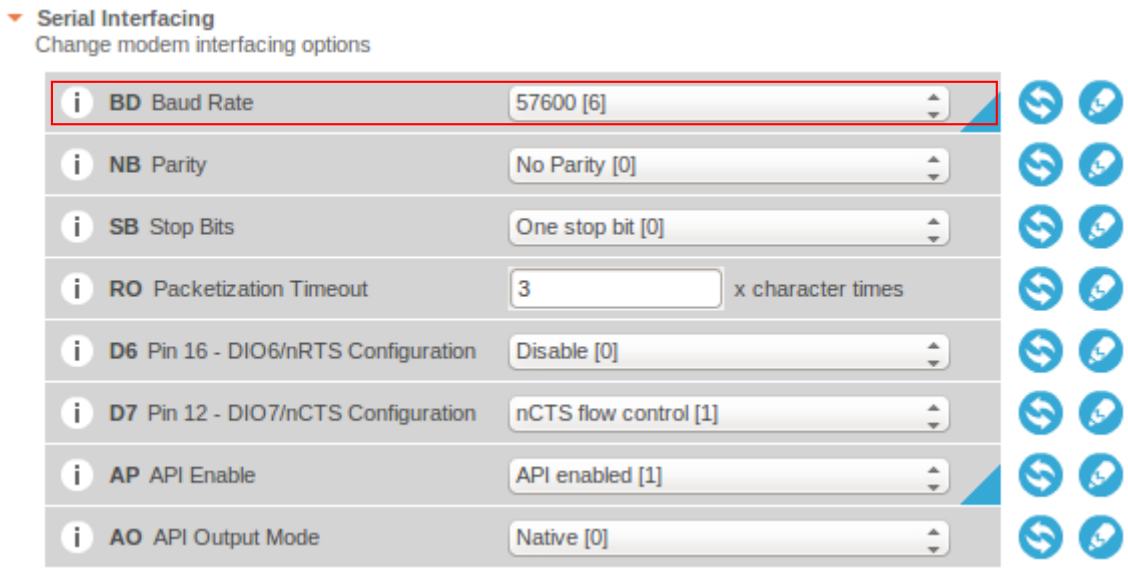



**Figura 2.16** Configuración de comunicación serial para todos los módulos del sistema.

### Módulos periféricos

Los módulos Xbee S2C que se encuentran en cada robot, se han configurado como Routers, lo que permite conectarlos en una sola red identificada por un número único conocido como PAN ID. Justamente, para la plataforma de pruebas, el valor de PAN ID se ha establecido en 1234, valor que debe constar en el campo ID de todos los módulos del sistema. La Figura 2.17 muestra el campo ID de uno de los Routers, así como el campo CE que debe permanecer con el valor de 0.

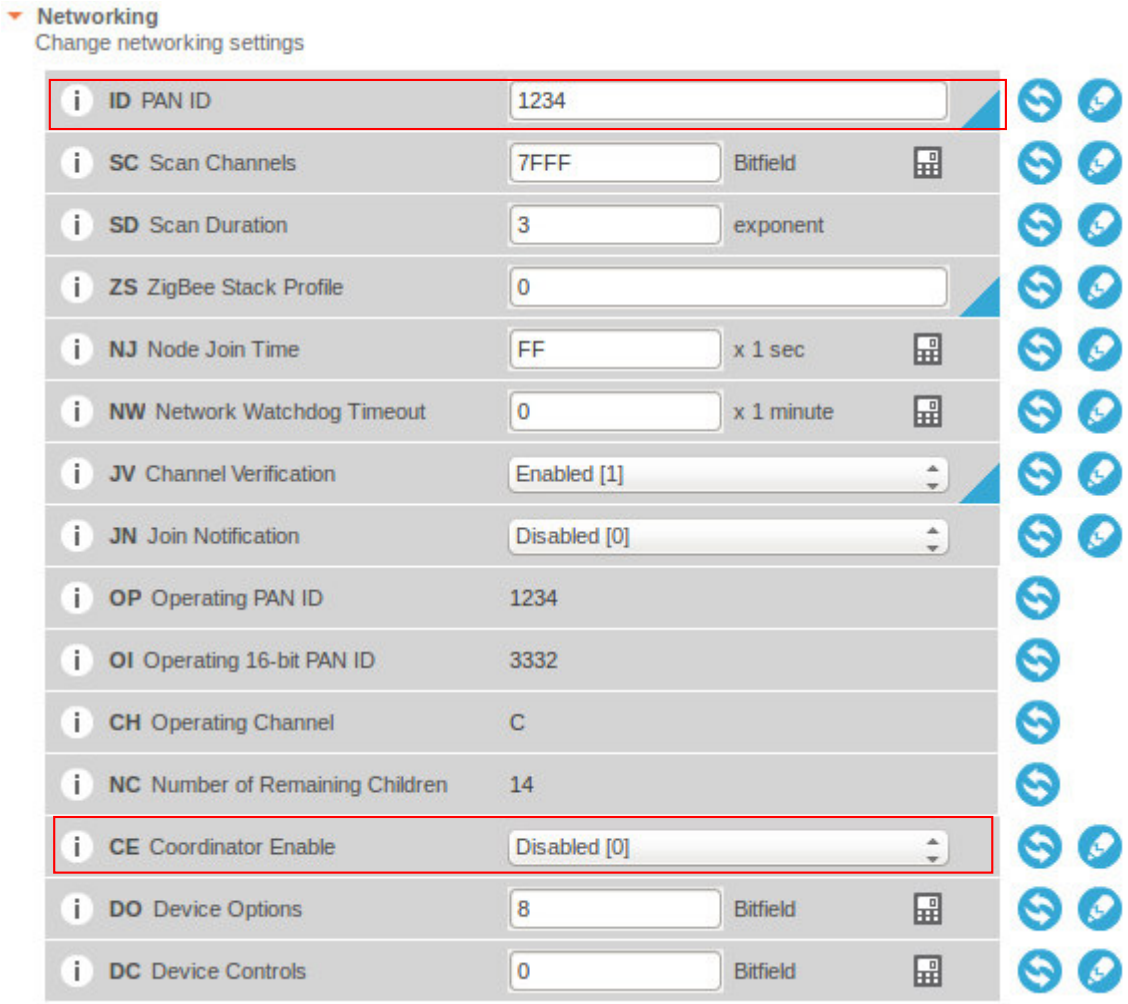



**Figura 2.17** Configuración de red para los módulos periféricos de cada robot móvil.

Cada Router es identificado por dos códigos únicos, que son utilizados por el módulo coordinador al comunicarse de manera individual con cada uno de ellos. Los dos códigos únicos son la dirección MAC del Router y una dirección de red de 16 bits, asignada una vez que se haya integrado un Router a la red local. Este último número puede cambiar en caso de que el módulo sea desvinculado de la red.

Adicionalmente, para asegurar que los módulos Router funcionen adecuadamente, se los configura sin un tiempo de inactividad, el cual debe ser configurado como se muestra en la Figura 2.18.

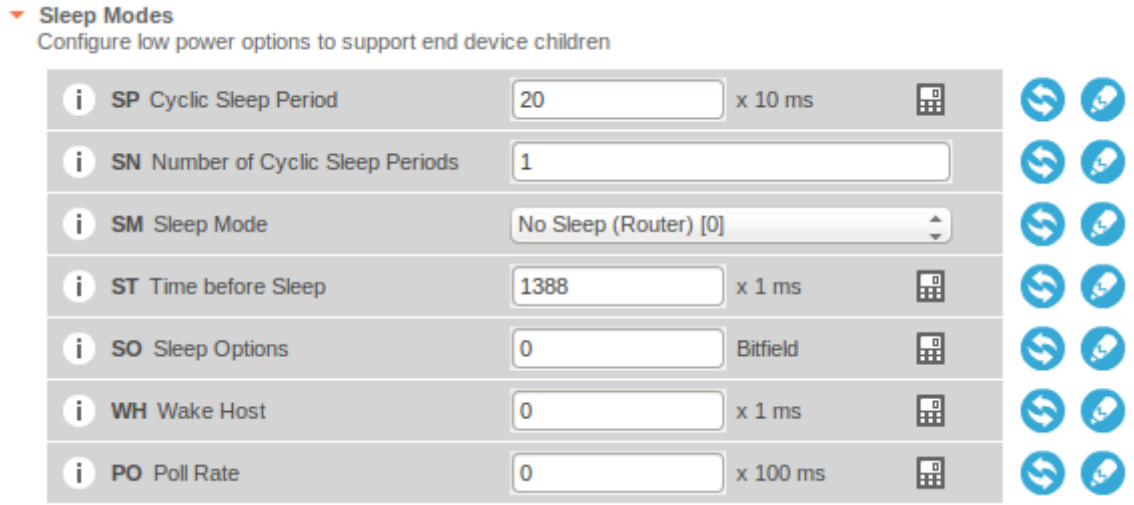



**Figura 2.18** Configuración adicional para módulos periféricos.

### Módulo coordinador

Este módulo se encuentra directamente conectado, a través de un adaptador serial, al computador donde se realiza la operación del sistema. La configuración del módulo coordinador es muy similar a la de los módulos Router, con la diferencia que en el campo CE de la página de configuración en XCTU, se debe especificar que el módulo funcionará como Coordinador. La Figura 2.19 muestra el mencionado cambio.

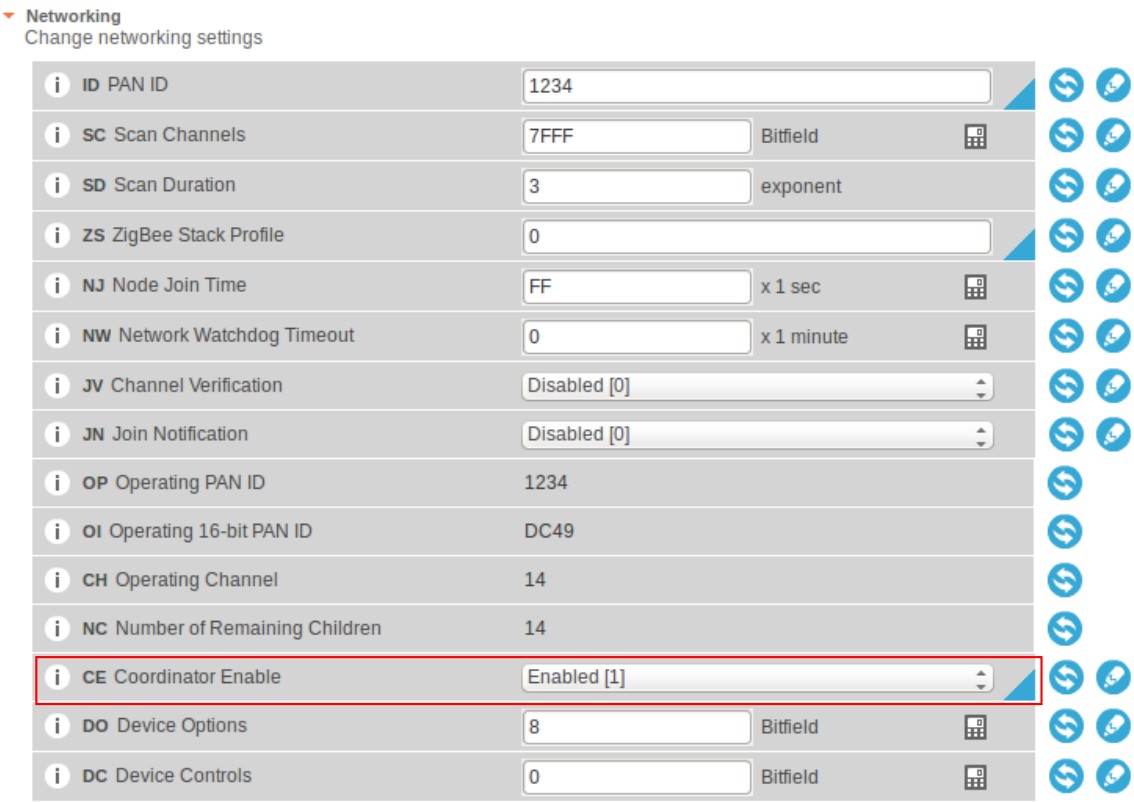



**Figura 2.19** Configuración general para modulo coordinador.

El direccionamiento de red también cambia en el caso del coordinador. La dirección de red de 16 bits queda configurada con un valor de 0, el cual puede ser utilizado por cualquier módulo Router para enviar información al módulo coordinador. La Figura 2.20 muestra el campo MY del módulo coordinador, donde se especifica su dirección de red.

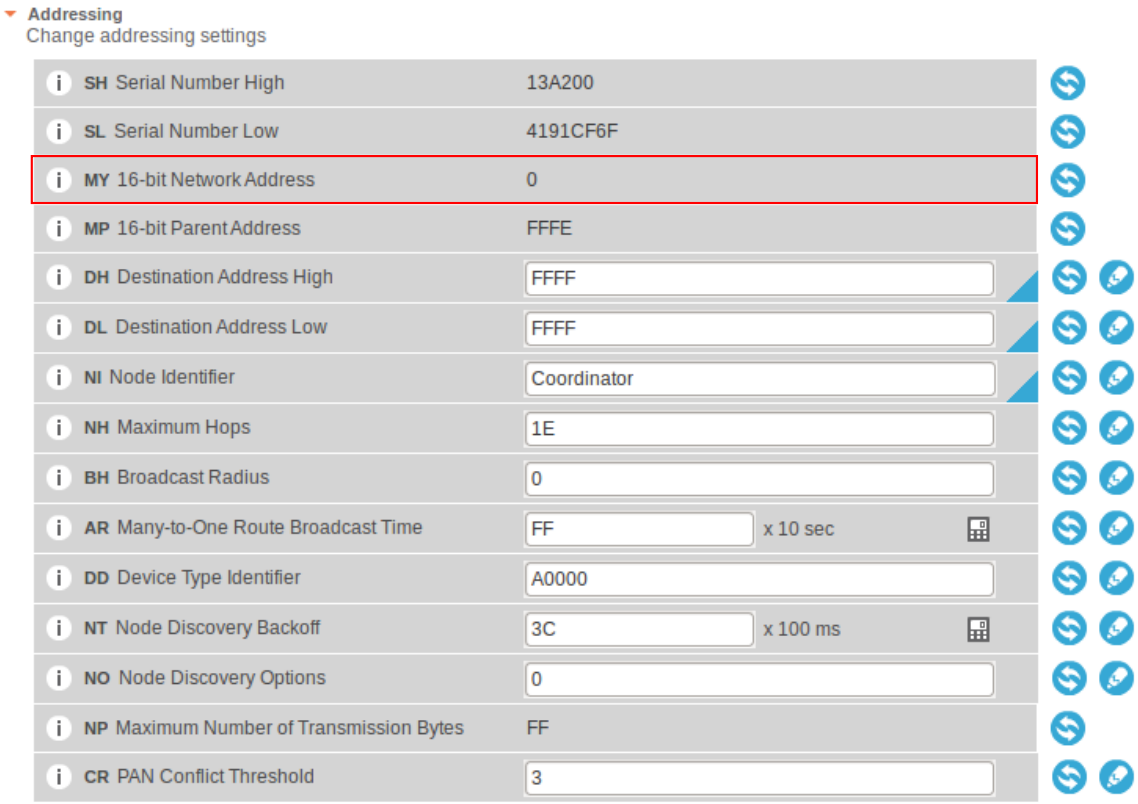



**Figura 2.20** Configuración de direccionamiento para modulo coordinador.

# 3 RESULTADOS Y DISCUSIÓN

## 3.1 PRUEBA DE DETECCIÓN DE OBSTÁCULOS

Como parte del proceso de análisis de imágenes es imperativo determinar las limitaciones del nodo ROS dedicado al mapeo del entorno. Si bien se ha utilizado una cámara con suficiente resolución, siempre existirán limitaciones respecto al tamaño de los obstáculos presentes. Como se mencionó en el capítulo anterior, se utiliza la librería OpenCV para realizar el análisis de imágenes provenientes de la cámara, específicamente la instrucción de detección de contornos "findContours()". Para este fin, se realizó una prueba para determinar el tamaño mínimo de los objetos que la instrucción es capaz de detectar. Se colocaron rectángulos en tamaños descendentes, hasta observar fallas en la detección del objeto más pequeño.

Si bien no existe un límite de tamaño máximo de los obstáculos, se ha logrado identificar que un obstáculo menor a 0.4 x 0.3 cm no es detectable. Este resultado asegura que la gran mayoría de figuras y formas geométricas que se puedan requerir en el entorno serán fácilmente detectadas y analizadas. Justamente, la Figura 3.1 muestra el resultado de la prueba mencionada, donde se observan encerrados en rectángulos negros, los obstáculos detectados y dibujados por OpenCV. Mientras que encerrados en un rectángulo rojo se observa el obstáculo de menor tamaño que no es posible detectar con la librería de Python.

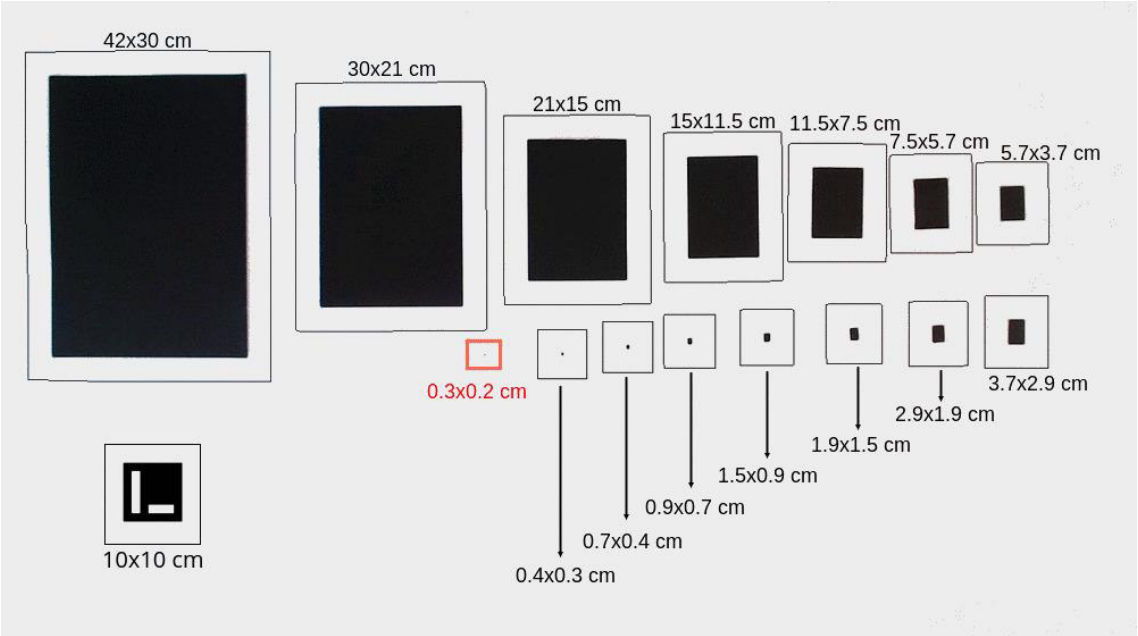



**Figura 3.1** Determinación del tamaño mínimo de obstáculos en el entorno, en la esquina inferior izquierda se presenta un marcador ArUco para poder comparar su tamaño.

## 3.2 PRUEBAS DE MAPEO

Para validar las capacidades del nodo ROS de mapeo, se realizaron distintas pruebas donde se evidencia la detección de obstáculos, de cualquier forma geométrica, así como de los marcadores ArUco a bordo de los robots móviles.

La Figura 3.2 muestra encerrados en color negro los objetos detectados en el entorno, de los cuales, los obstáculos son identificados con su primera coordenada escrita en rojo, y los marcadores de referencia son identificados con una de sus coordenadas escrita en azul. En el caso de los robots, además de ser identificados por su marcador, se consideran también obstáculos en el entorno, puesto que de mantenerse estáticos no deberá generarse una ruta sobre ellos.

Como se puede observar y se explicó en el capítulo anterior, cada obstáculo detectado cuenta con un área adicional que lo encierra, la cual es utilizada como guarda o área virtual para evitar que los robots móviles colisionen con el contorno físico de los obstáculos. Dicha área fue determinada a partir de la geometría y medida física de los robots. De esta manera, se estableció un área de guarda de aproximadamente la mitad del tamaño de los robots móviles, permitiendo que incluso si una ruta generada es adyacente al área de guarda, los robots no tocarán los obstáculos.

Adicionalmente, la detección de los marcadores ArUco viene acompañada del dibujo rectangular, del límite del marcador en rojo, así como del identificador numérico o ID de cada uno.

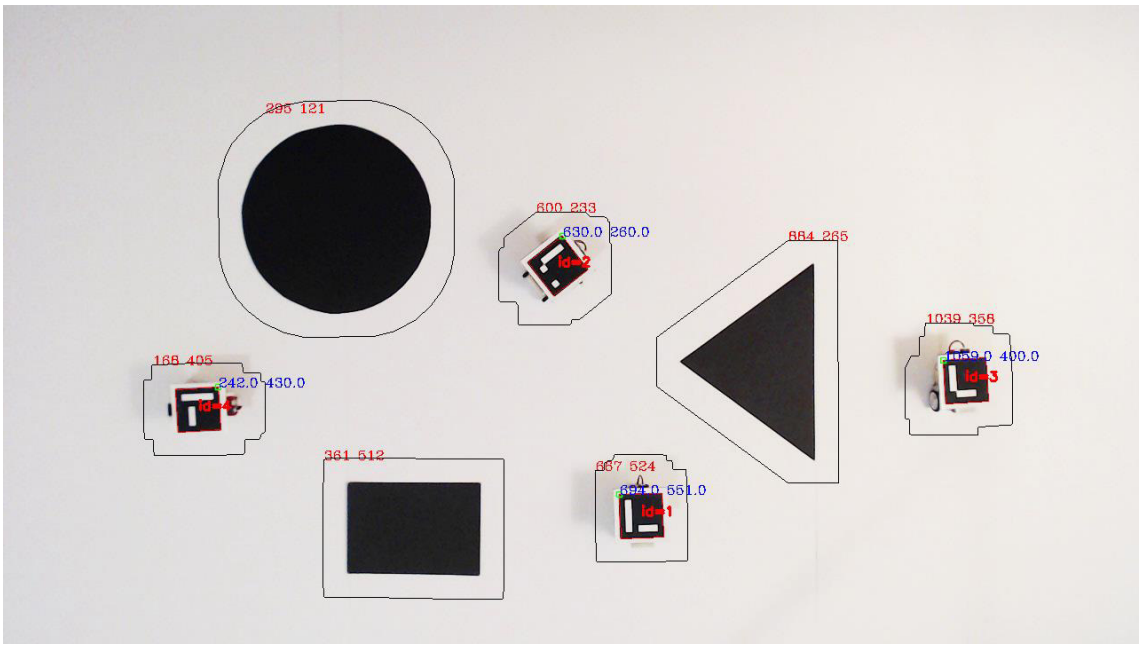



**Figura 3.2** Detección de obstáculos y marcadores de referencia ArUco, incluyendo sus coordenadas.

Varios entornos fueron configurados para demostrar que no existe diferencia en el rendimiento del mapeo respecto a la configuración del espacio de trabajo. Cualquier entorno puede ser mapeado mientras los obstáculos presentes no excedan el límite mínimo de tamaño indicado en la prueba explicada en la Sección 3.1.

La Figura 3.3 muestra cuatro entornos mapeados de forma que se observen, en rojo, las coordenadas de los vértices de cada obstáculo detectado. Se puede apreciar que en cada vértice de las figuras detectadas se dibujan las coordenadas correspondientes. En el caso de figuras circulares, estas son aproximadas a polígonos que presentan numerosos vértices, por lo que se aprecian varias coordenadas que lo describen.

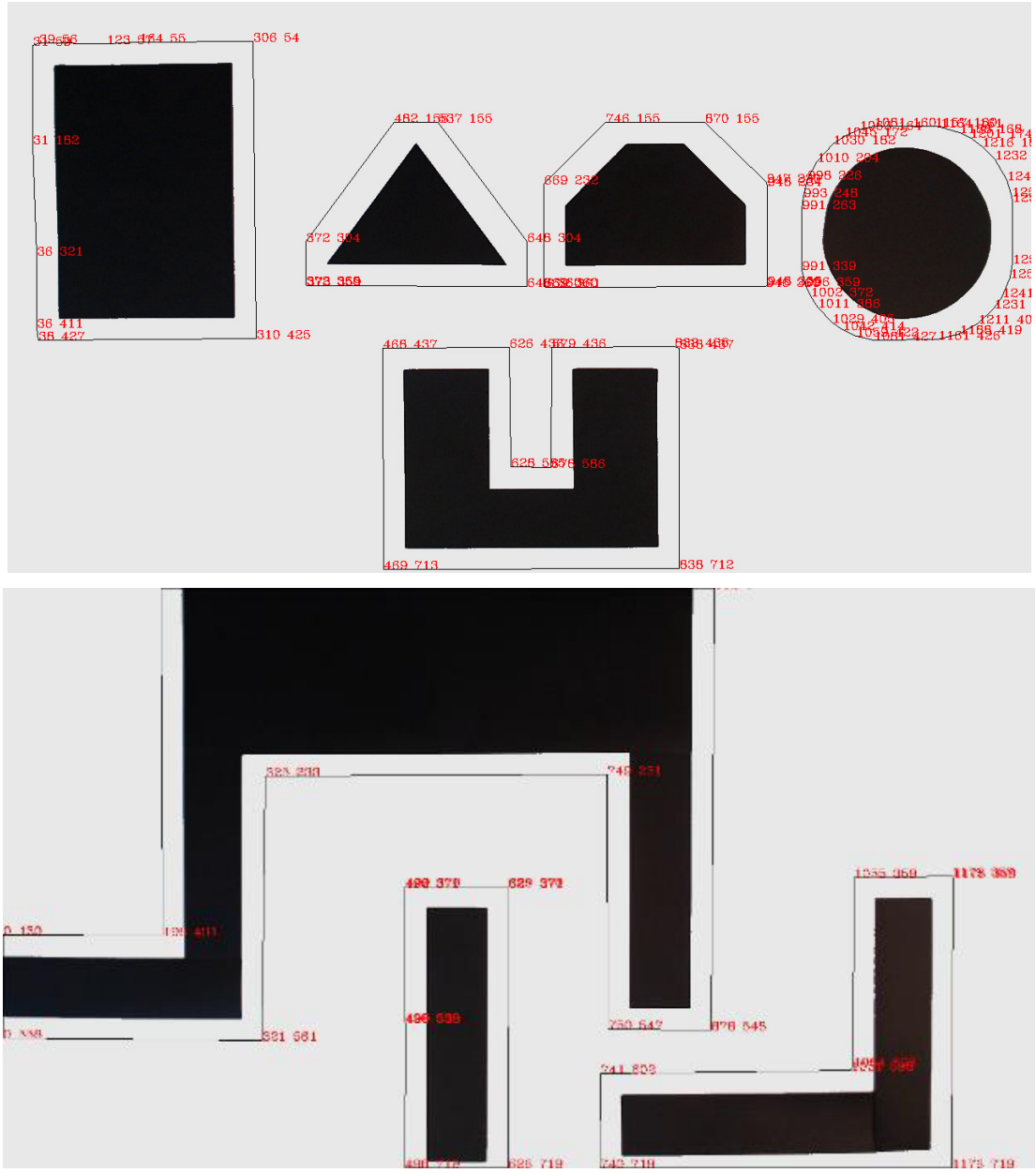


**Figura 3.3** Distintos escenarios a prueba en el nodo de mapeo.

### 3.2.1 Estimación de orientación

Como parte del proceso de mapeo del entorno, se requiere calcular la orientación de los robots móviles. La Figura 3.4 muestra la manera en que se identifican los ángulos de orientación de cada robot. El mismo proceso es utilizado dentro del nodo de control para el cálculo del ángulo de cada punto meta respecto al robot móvil.

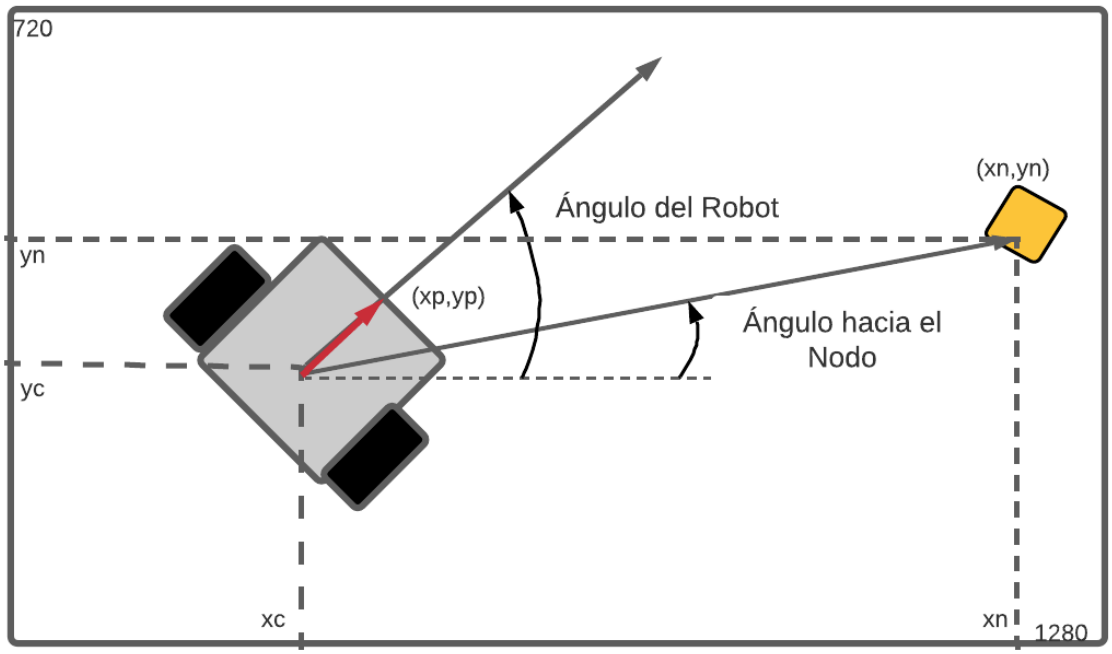



**Figura 3.4** Identificación y cálculo de ángulos de orientación.

El cálculo del ángulo se realiza mediante la función atan2(), la cual retorna el ángulo, entre -π y π, de un par de coordenadas respecto al plano. En el caso del área de trabajo, es un plano de 1280 pixeles de largo por 720 pixeles de ancho.

El cálculo del ángulo hacia el nodo se realiza aplicando la ecuación (16).

$$\varphi_{nodo} = atan2((xn - xc), (yn - yc)) \quad (16)$$

Mientras que el cálculo del ángulo del robot se realiza aplicando la ecuación (17).

$$\varphi_{robot} = atan2((xp - xc), (yp - yc)) \quad (17)$$

Cuando se utiliza el control de orientación, el ángulo hacia el nodo $\varphi_{nodo}$ toma el nombre de Setpoint.

A continuación se muestra un ejemplo del cálculo de los ángulos considerando valores numéricos. La Figura 3.5 muestra el ejemplo propuesto y los resultados al aplicar las ecuaciones (18) y (19).

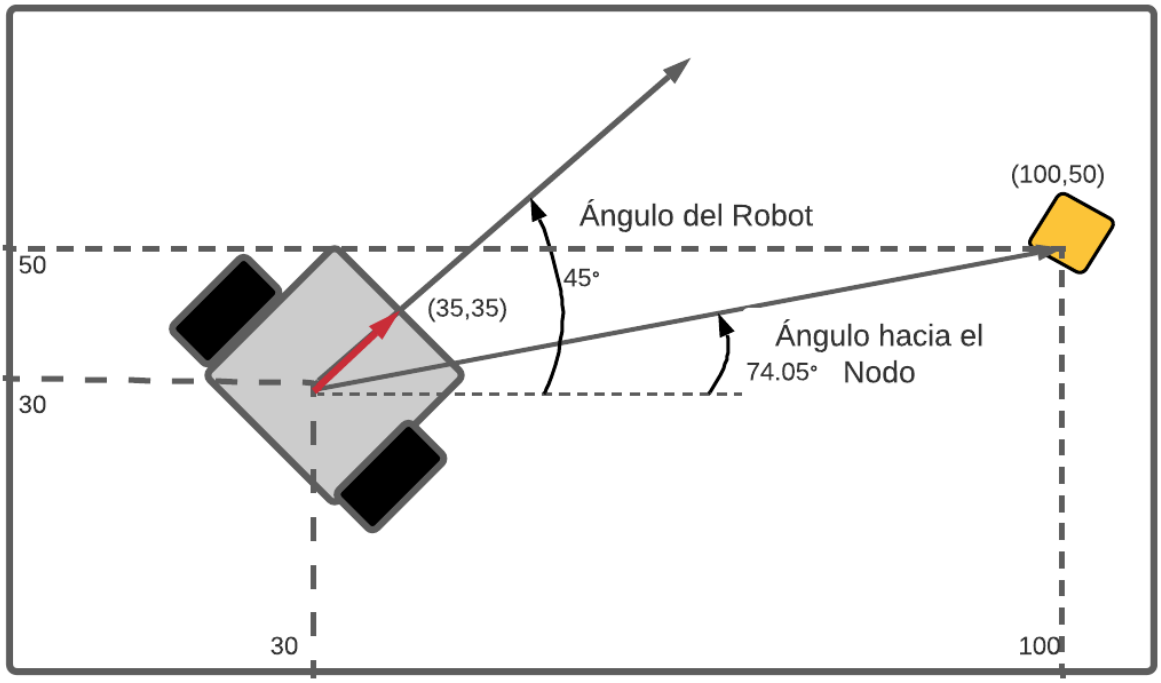



**Figura 3.5** Ejemplo numérico del cálculo de ángulos.

$$\varphi_{nodo} = atan2\big((100-30),(50-30)\big) = 74.05° \quad (18)$$

$$\varphi_{robot} = atan2\big((35-30),(35-30)\big) = 45° \quad (19)$$

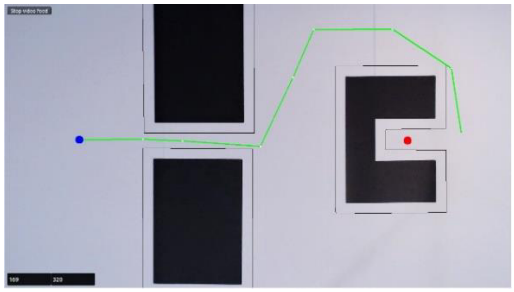
## 3.3 PRUEBAS DE DESEMPEÑO DEL ALGORITMO RRT*

La implementación del algoritmo de navegación requiere también de pruebas para conocer su desempeño en cuanto a tiempo computacional y calidad de las rutas generadas en relación con los parámetros configurados. La calidad de las rutas puede ser comprendida como su eficiencia en conectar el punto inicial y meta, mediante caminos que presenten el menor costo o distancia posible.

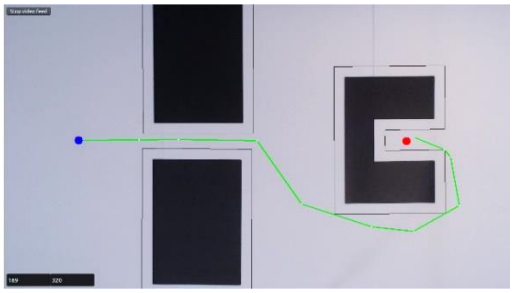
### 3.3.1 Efectos del número de Nodos en el tiempo computacional

Como se mencionó en Capítulo 2, los resultados del algoritmo RRT* son asintóticamente óptimos dependiendo del número de nodos que se generen en el espacio de trabajo. Así, un número mayor de nodos resultará en una mejor solución con caminos más cortos a cambio de un mayor tiempo computacional. Ya que existen otros parámetros que afectan el desempeño del algoritmo e influyen en el tiempo computacional, para realizar una comparación justa se han mantenido todos los demás parámetros en valores fijos para todas las pruebas. Los valores de épsilon (ε) y radio de búsqueda ($R$) fueron constantes con valores de 100 y 200 respectivamente.

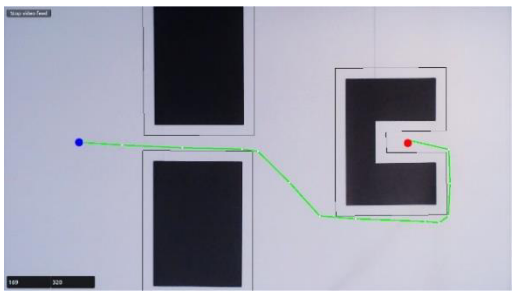
Se realizaron cinco pruebas, con valores incrementales del número de nodos generados, desde 500 hasta 2500; dichos resultados se presentan en la Figura 3.6, donde el punto de partida se muestra en color azul, mientras que el punto meta se observa de color rojo.



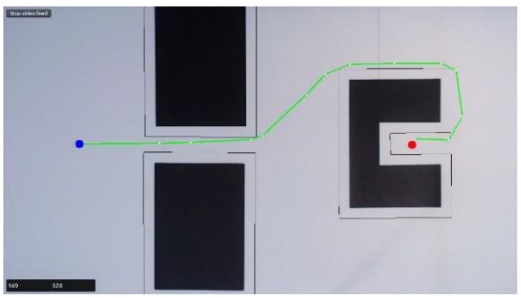
(a) Resultado con 500 nodos



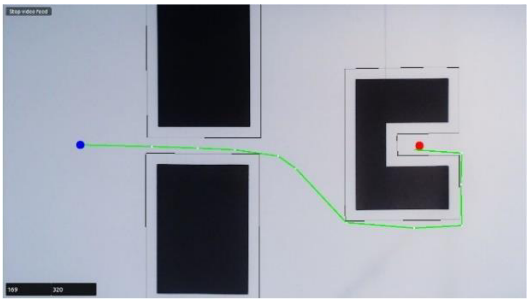
(b) Resultado con 1000 nodos



(c) Resultado con 1500 nodos



(d) Resultado con 2000 nodos



(e) Resultado con 2500 nodos

**Figura 3.6** Resultados obtenidos con un número incremental de nodos.

La Figura 3.6 (a) muestra que, para el escenario propuesto, un número de 500 nodos es claramente insuficiente para conectar de manera satisfactoria los puntos designados, dejando el nodo final de la ruta a una distancia considerable del punto meta. A partir de los 1000 nodos, se observa que se puede generar un camino que acerca al robot móvil al punto meta con una distancia máxima de 24 pixeles, que corresponden a 2 cm; sin embargo, la ruta no rodea correctamente los obstáculos y mantiene trayectos que pueden ser optimizados.

A partir de La Figura 3.6 (c) se observa que la ruta generada es cada vez más eficiente, conectando los puntos de interés con rutas más cortas, rodeando los

obstáculos y evitando trayectos innecesarios. Finalmente, las rutas obtenidas con 2000 y 2500 nodos son notablemente superiores a las anteriores, con la desventaja que toman mucho mayor tiempo computacional en generarse.

Los resultados del tiempo computacional se observan en la Tabla 3.1, así como la distancia entre el punto meta seleccionado por el usuario y el nodo final generado por el algoritmo.

**Tabla 3.1** Tiempo computacional con distintos valores de N en RRT*

| Número de nodos N | Tiempo computacional [min] | Distancia entre meta y nodo final [px] |
|---|---|---|
| 500 | 0.14 | 136 |
| 1000 | 1.39 | 24 |
| 1500 | 5.46 | 8 |
| 2000 | 10.49 | 19 |
| 2500 | 14.67 | 16 |

Como se puede apreciar, en todos los casos, un incremento en el número de nodos representa un incremento en el tiempo computacional para obtener la solución final. Cabe destacar que gracias a la naturaleza aleatoria en la creación de nodos RRT*, no existe una relación lineal entre el número de nodos y el tiempo invertido.

La solución resaltada en la Tabla 3.1 es un intermedio entre calidad de la ruta obtenida y el tiempo computacional invertido. Para la aplicación del presente trabajo de titulación, la ruta mostrada en la Figura 3.6 (c) representa una solución totalmente adecuada, manteniendo un tiempo de computación relativamente corto y conectando los puntos de interés con trayectos rectos que rodean los obstáculos presentes.

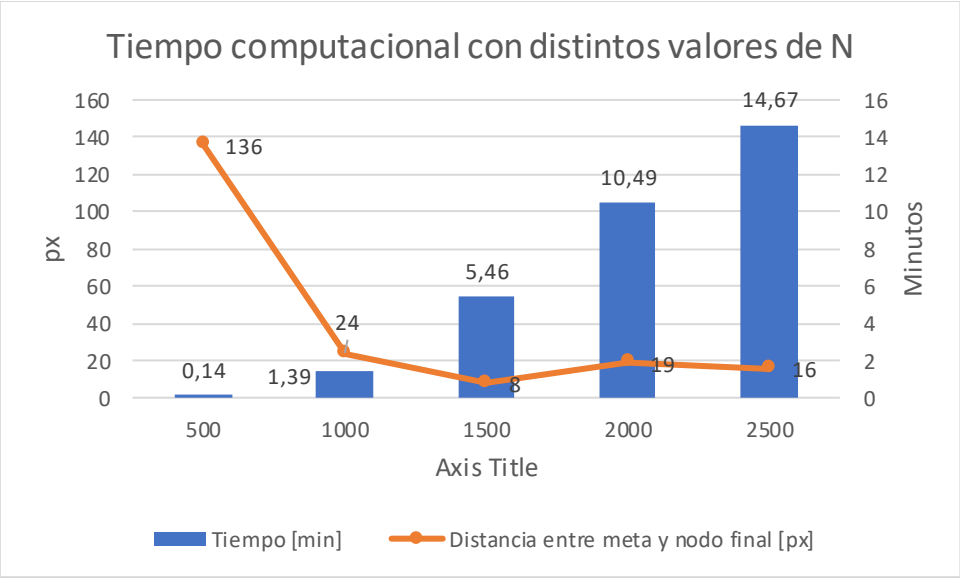



**Figura 3.7** Resultados de tiempo computacional en relación con la distancia final.

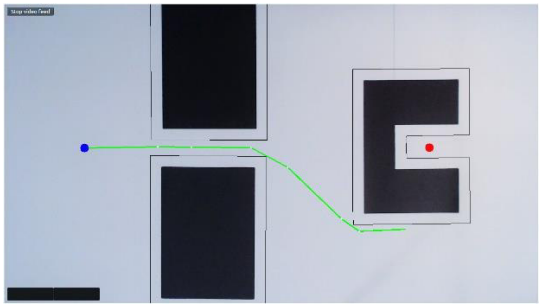
Una métrica útil al determinar el éxito de una solución encontrada con RRT* es la distancia entre el nodo final y el punto meta seleccionado por el usuario. La Figura 3.7 muestra que a partir de los 1500 nodos, dicha distancia no se reduce significativamente, e incluso aumenta dependiendo de los nodos generados aleatoriamente cerca del punto meta. Si bien no se puede garantizar que el aumento de nodos resulte en una reducción de la distancia mencionada, se puede esperar que a partir de un número determinado de nodos, esta distancia se mantenga en valores pequeños y varíe solo en función de la naturaleza aleatoria del algoritmo.

En el caso de las pruebas realizadas, una distancia de 24 pixeles (2 cm) o menos es totalmente aceptable puesto que corresponde a una distancia menor que la dimensión de uno de los lados de los robots. Finalmente, dependiendo de la aplicación, un tiempo computacional más corto puede ser de mayor relevancia que obtener soluciones como las observadas en la Figura 3.6 (d-e), que requieren extensos tiempos para generarse.

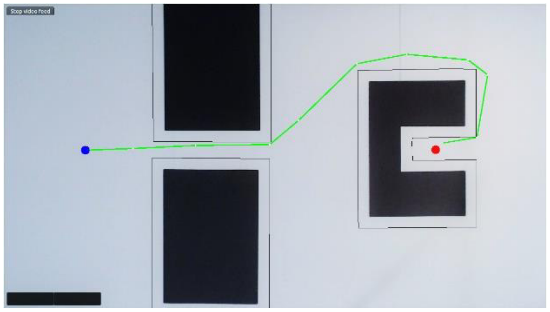
### 3.3.2 Efectos del valor de épsilon en la solución final

Aparte del número de nodos, existen dos parámetros que afectan de forma directa la solución que se encuentre con RRT*. Uno de estos parámetros es épsilon (ε), cuyo valor corresponde a la distancia máxima a la que se crearán los nuevos nodos en cada iteración del algoritmo de navegación. Su sintonización cobra mayor importancia cuando no se puede generar una gran cantidad de nodos puesto que si el valor de épsilon es grande (respecto al entorno), un reducido número de nodos podrá alcanzar y explorar mayores distancias que al configurar un valor de épsilon muy pequeño.

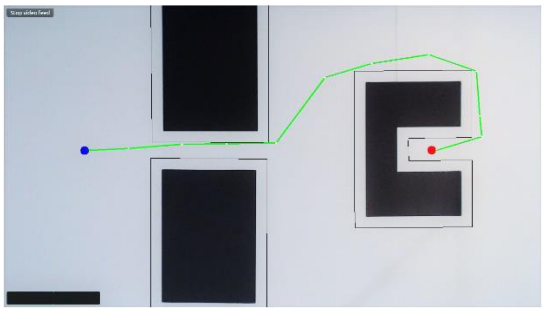
La Figura 3.8 muestra los resultados con 3 valores distintos de épsilon: 25, 200 y 400; manteniendo constante el número de nodos generados (N=1500) y el radio de búsqueda (R=200).



(a) Resultado con épsilon=25



(b) Resultado con épsilon=200



(c) Resultado con épsilon=400

**Figura 3.8** Resultados con distintos valores de épsilon.

La Figura 3.8 (a) muestra claramente que cuando se configura un valor muy pequeño, el árbol no llegara a extenderse de la misma manera que con un valor apropiado de épsilon. De la misma forma que un valor muy pequeño resulta en una solución inadecuada, un valor muy alto también puede generar problemas dependiendo del entorno. Si bien con un valor de épsilon=400 se puede obtener una solución aceptable, dada la naturaleza aleatoria en la generación de nodos, existen casos en que por ser un valor muy alto, y un entorno con espacios reducidos, el árbol puede no llegar a explorar dichos espacios; este resultado se puede observar en la Figura 3.9.

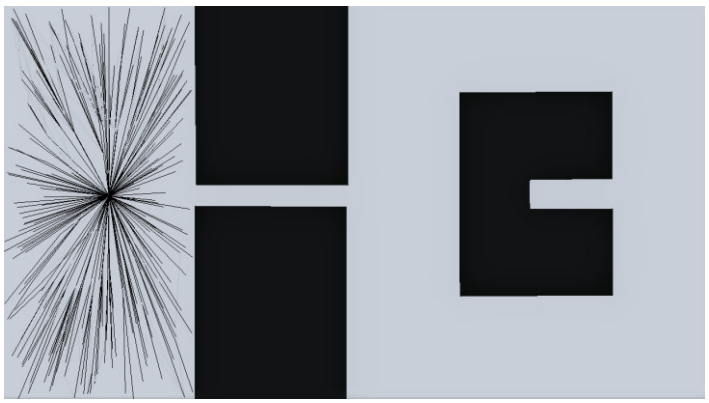


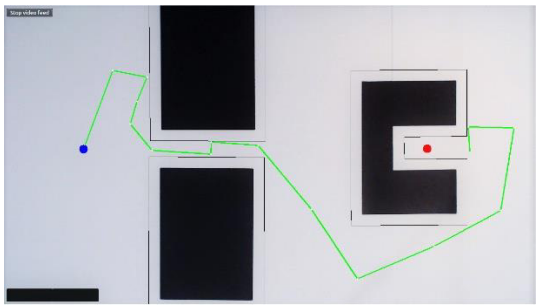
**Figura 3.9** Efectos negativos de un valor de épsilon muy grande respecto al entorno.

### 3.3.3 Efectos del valor del Radio de búsqueda en la solución final

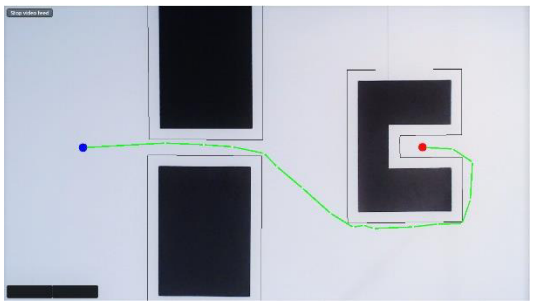
Otro valor que tiene injerencia en la solución final, tanto en su forma como en el tiempo computacional invertido es el Radio de búsqueda ($R$). Este valor corresponde al radio en el que se buscara conectar los nuevos nodos para reducir el costo, así como el radio en el que se reconectarán los nodos vecinos para reconfigurar el árbol.

Justamente, cuando se configura un valor de $R$ muy pequeño, no se logrará mantener caminos con el menor costo posible, lo que resultará en caminos que no reduzcan el costo a medida que se agreguen los nodos creados. Por el otro lado, un valor extremadamente grande, significará que una mayor área tendrá que ser analizada, incluyendo todos sus nodos, para reconfigurarlos en el caso de que se halle un camino con menor costo.

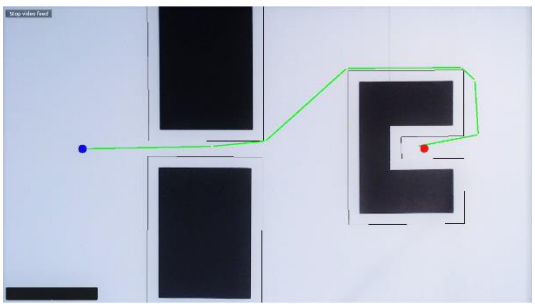
La Figura 3.10 muestra los resultados con 3 valores distintos de $R$: 25, 100 y 400; manteniendo constante el número de nodos generados (N=1500) y épsilon (ε=200).



(a) Resultado con $R$ =25



(b) Resultado con $R$ =100



(c) Resultado con $R$ =400

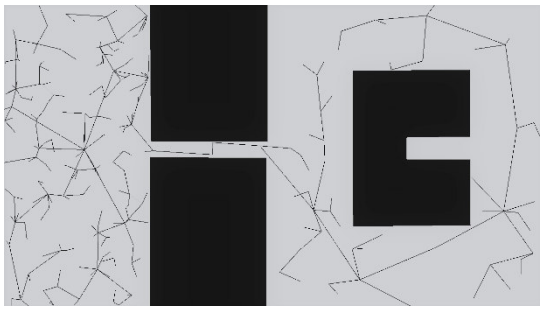
**Figura 3.10** Resultados con distintos valores de R.

Los resultados de la Figura 3.10 (a) muestran que al utilizar valores muy pequeños de $R$, los caminos encontrados no se acercan a una solución óptima, lo que resulta en rutas ineficientes comparado con las obtenidas para los otros casos.

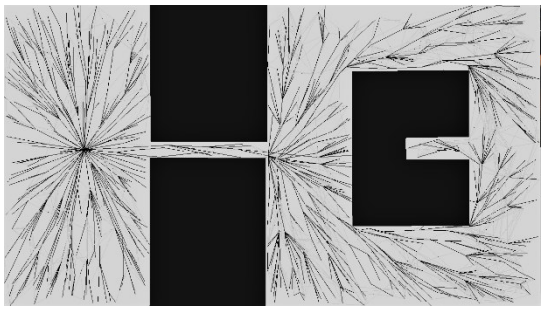
La principal diferencia entre la Figura 3.10 (a) y las Figura 3.10 (b-c) es que, en la primera, el algoritmo no logra eliminar trayectos innecesarios puesto que el análisis de la vecindad de los nodos creados se realiza en un área demasiado pequeña. De esta

manera, valores muy pequeños de $R$ resultan en soluciones similares a las que se obtendría al utilizar el algoritmo RRT, descartando todos los beneficios de RRT*.

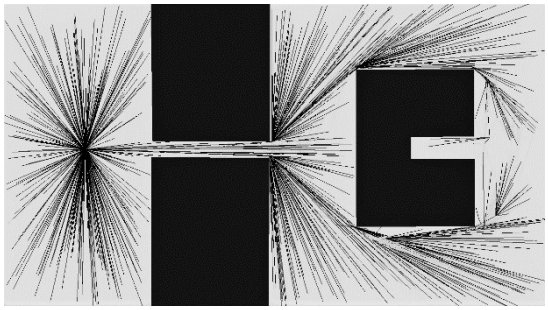
Una comparación entre la topología del árbol generado con cada valor de $R$ se muestra en la Figura 3.11.



(a) Árbol generado con $R$ =25



(b) Árbol generado con $R$ =100



(c) Árbol generado con $R$ =400

**Figura 3.11** Topología del árbol generado por RRT* con distintos valores de R.

Los efectos de un valor más alto de $R$ se pueden apreciar en la Tabla 3.2, donde se aprecia el tiempo computacional con distintos valores de $R$. Si bien se obtiene una solución mucho mejor al usar valores altos del Radio de búsqueda, el tiempo computacional aumenta como se había esperado. A pesar de este aumento en el tiempo de espera y la aparente diferencia en la topología del árbol observado en la Figura 3.11, la diferencia entre las soluciones finales, mostradas en las Figura 3.10 (b) y la Figura 3.10 (c) no es altamente apreciable, resultando en rutas muy similares, con tramos rectos que minimizan la distancia de recorrido.

Finalmente, un valor alto de $R$ es siempre deseable, siempre y cuando no se incurra en valores extremos que incrementen el tiempo computacional sin brindar apreciables mejoras en las rutas generadas.

**Tabla 3.2** Tiempo computacional con distintos valores de R.

| Valor de $R$ | Tiempo computacional [min] |
|---|---|
| 25 | 3.64 |
| 100 | 4.82 |
| 400 | 5.96 |

### 3.3.4 Resultados en tres escenarios distintos

Como parte de las pruebas de desempeño del algoritmo RRT*, se realizaron pruebas en tres escenarios distintos, con valores apropiados de N, ε y $R$ para obtener soluciones adecuadas. Estos tres escenarios son: Laberinto, Obstáculos geométricos y Cuartos cóncavos.

**Escenario 1: Laberinto**

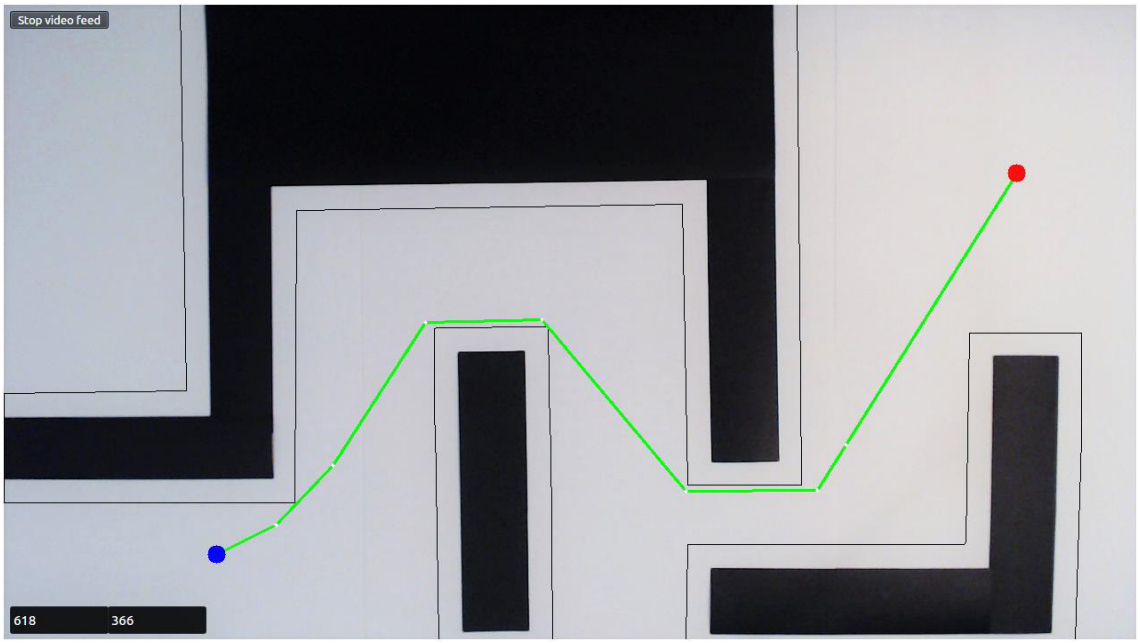



**Figura 3.12** Resultado en escenario tipo laberinto.

La primera prueba fue realizada en un escenario de tipo laberinto, con valores de N=3000, ε=100 y $R$=400. El tiempo computacional invertido fue de 5.66 minutos y la distancia desde el punto meta hasta el nodo final de 9 pixeles (0.75 cm). En el resultado mostrado en la Figura 3.12 se puede apreciar el efecto de un valor de $R$ elevado que permite obtener trayectos más rectos y extensos.

**Escenario 2: Obstáculos geométricos**

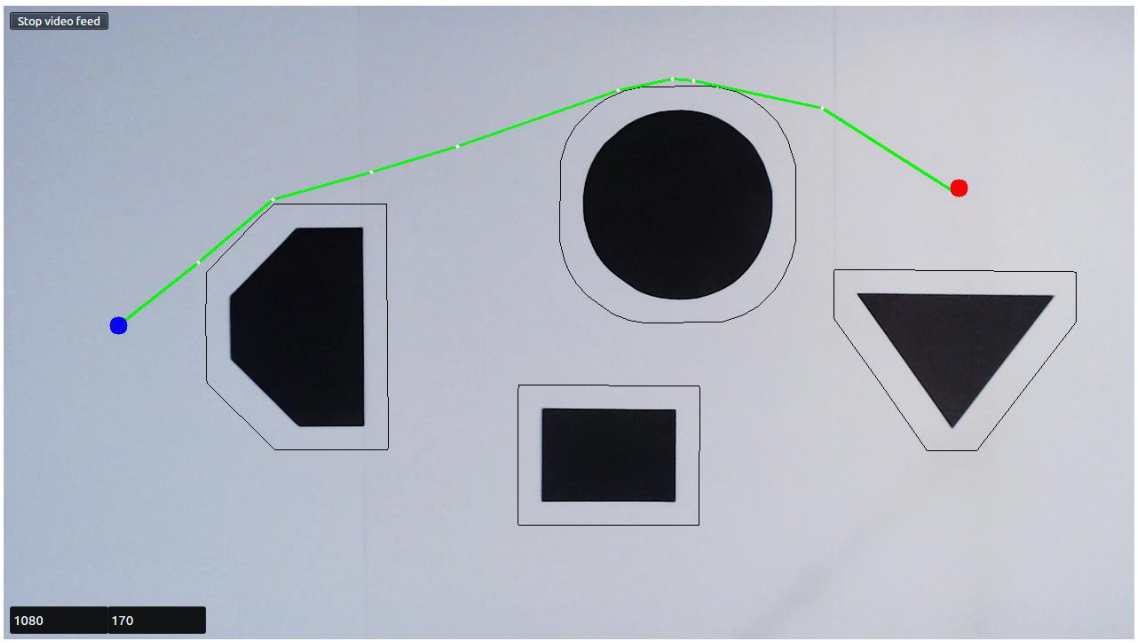



**Figura 3.13** Resultado en escenario de múltiples caminos posibles.

Esta prueba fue realizada en un escenario con múltiples obstáculos geométricos y con valores de N=2000, ε=200 y $R$=200. El tiempo computacional invertido fue de 10.34 minutos y la distancia desde el punto meta hasta el nodo final de 7 pixeles. La Figura 3.13 ilustra la capacidad del algoritmo RRT* de hallar el camino más corto a pesar de tener varias posibilidades que puedan satisfacer el requerimiento de unir el punto inicial con el punto meta.

La Figura 3.14 muestra otras posibles soluciones que el algoritmo podría haber encontrado como parte del proceso iterativo, sin embargo, la solución final (en verde), es aquella que presenta menor distancia o costo en su ejecución. Si bien el algoritmo es incapaz de encontrar la mejor solución global al problema, garantiza que con el número de nodos generados, la solución hallada será siempre la mejor de todas las posibilidades encontradas.

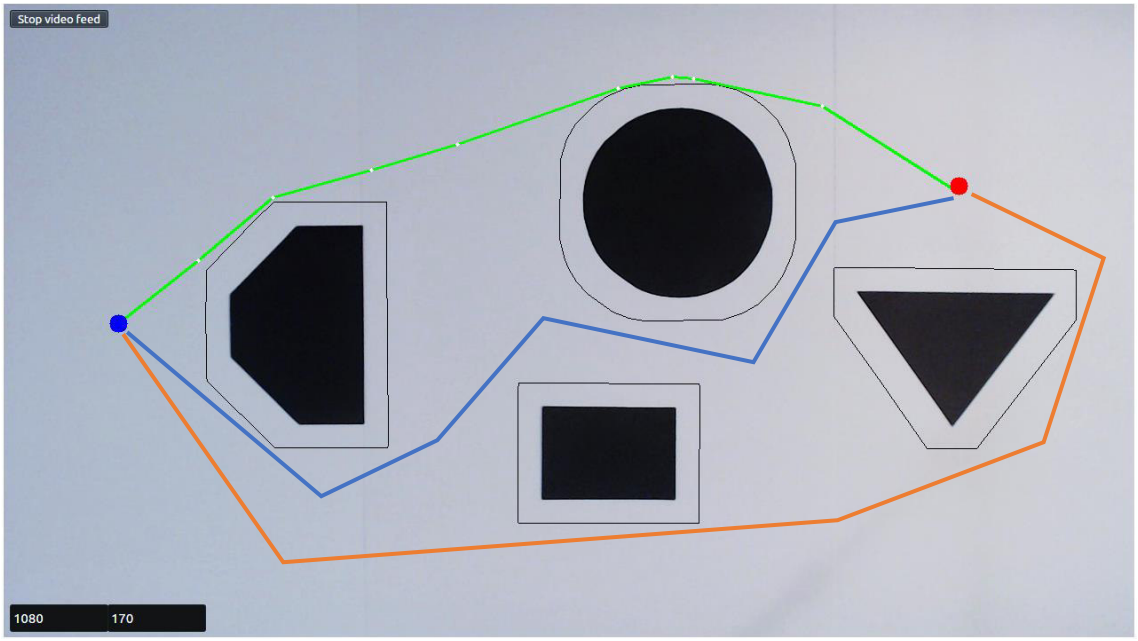



**Figura 3.14** Distintas soluciones posibles con RRT*.

**Escenario 3: Cuartos cóncavos**

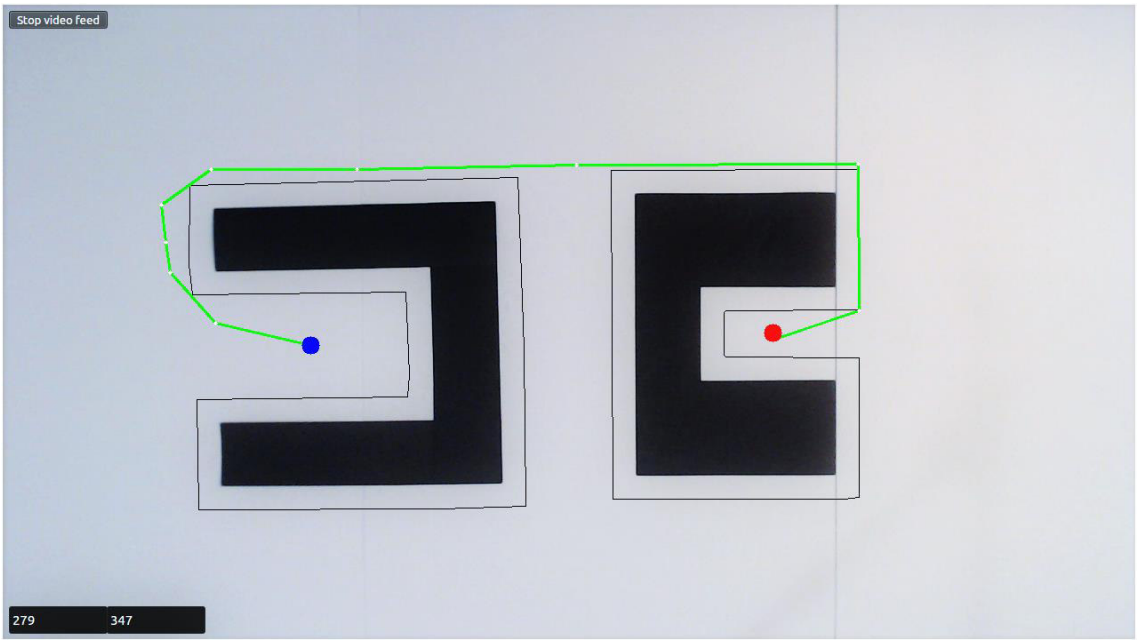



**Figura 3.15** Resultado en escenario con cuartos cóncavos.

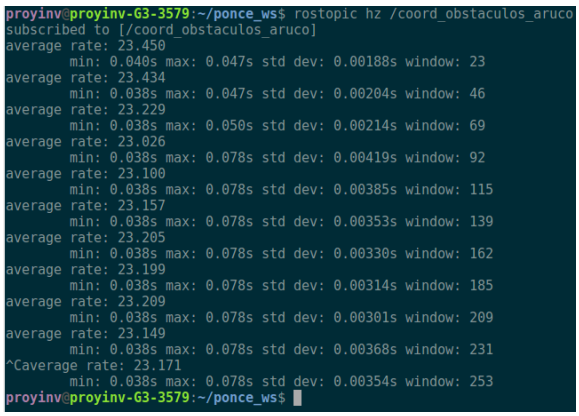
La tercera prueba fue realizada en un escenario de cuartos cóncavos, donde en su interior se encuentran los puntos de inicio y meta para el algoritmo. La prueba se realizó con valores de N=3000, ε=100 y $R$=400. El tiempo computacional invertido fue de 24.22 minutos y la distancia desde el punto meta hasta el nodo final de 7 pixeles (0.75 cm).

En este caso es útil también un valor alto de $R$ para asegurar que en los trayectos extensos se consiga un camino recto con el menor costo posible. El valor relativamente pequeño de épsilon permite generar nodos que rodeen el perímetro de los obstáculos presentes. La Figura 3.15 muestra la solución final hallada para este escenario, demostrando la capacidad de extensión y exploración del árbol RRT* incluso en entornos complejos.

## 3.4 PRUEBAS DE LATENCIA

La latencia de un sistema está limitada por el proceso que tome más tiempo y en el caso de la plataforma de pruebas, es la velocidad de la cámara utilizada. Ya que la máxima velocidad de captura de la cámara Logitech es de 30 cuadros por segundo (fps), esta es la máxima frecuencia con la que se podrían realizar todos los cálculos de estimación de posición y orientación. Dado que dichos cálculos requieren tiempo, la latencia real del sistema de adquisición de datos es un poco menor a 30 Hz. La Figura 3.16 muestra la frecuencia de publicación de datos en los dos tópicos más importantes del sistema, los cuales corresponden a las imágenes procesadas por el nodo de mapeo y los datos de posición de los robots y los obstáculos. Se puede apreciar que dicha frecuencia se encuentra alrededor de los 23Hz, la misma que puede variar dependiendo de la carga de procesamiento que tenga el computador.


(a) Frecuencia de publicación de imágenes procesadas




(b) Frecuencia de publicación de coordenadas de los robots y obstáculos

**Figura 3.16** Frecuencia de publicación de mensajes ROS en el sistema.

Una fuente adicional de latencia, aparte de la actualización de datos, es el tiempo que se requiere para ejecutar el lazo de control. El promedio del tiempo de ejecución del lazo de control depende principalmente de los tiempo necesarios para la comunicación entre los robots móviles y el módulo central del computador. Una de las recomendaciones del fabricante de los módulos de comunicación es permitir un intervalo de 10[ms] entre cada transmisión para asegurar un correcto funcionamiento.

Adicionalmente, se debe considerar que, para el funcionamiento del controlador, es indispensable conocer el tiempo que existe entre cada iteración, ya que se requiere de un tiempo de discretización para la implementación del PID. Como parte de las pruebas de funcionamiento, se ha realizado la recolección de datos respecto al tiempo que toma al lazo de control ejecutarse consecutivamente. Ya que el tiempo del cálculo de valores y el procesamiento de datos no es siempre el mismo, se asume cierta variación entre cada ejecución del lazo de control. En la Figura 3.17 se muestran los datos obtenidos durante el funcionamiento del sistema para 474 iteraciones, donde cada iteración corresponde a una ejecución del lazo de control.

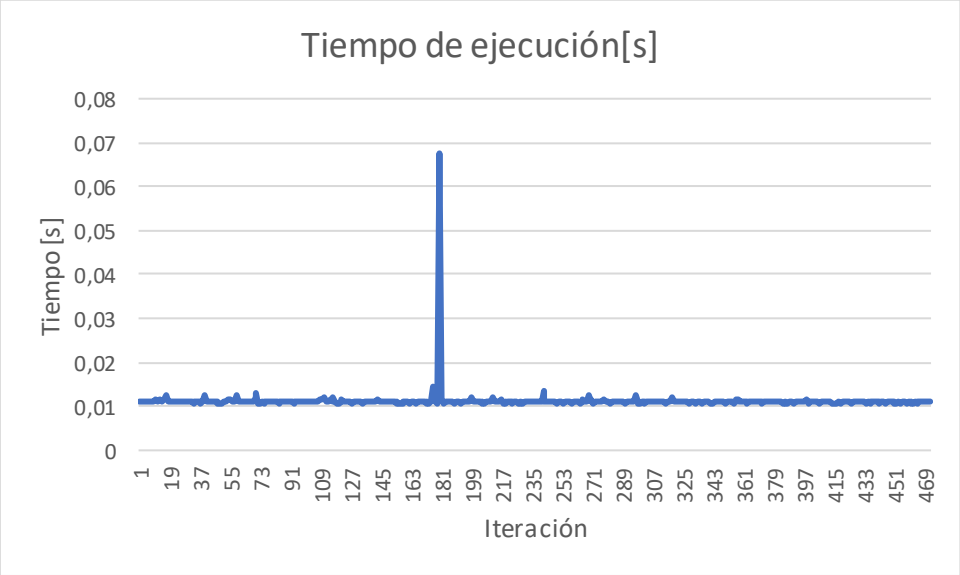



**Figura 3.17** Tiempos de latencia en el lazo de control.

El promedio del tiempo de latencia es de 0.0122 segundos y se puede apreciar que existe un solo caso en que el tiempo aumenta respecto al promedio, pero no afecta de forma significativa el funcionamiento del sistema. Esta variación de latencia puede tener distintos orígenes, entre ellos el hecho de que el computador ejecuta de manera paralela diversas otras tareas que en ciertas ocasiones pueden interferir con el funcionamiento de los nodos ROS. Si bien estos se ejecutan de manera continua, no se puede garantizar que en todos los casos corran en tiempo real con tiempos fijos y exactos.

Por estas razones, la ejecución del control PID se utiliza un tiempo de discretización variable, el cual se calcula de manera inmediata en cada iteración del lazo de control. El cálculo del tiempo transcurrido se realiza utilizando las funciones de la librería “time”

de Python. De esta forma, se permite ejecutar el lazo de control a la mayor velocidad posible de acuerdo con las capacidades del computador.

## 3.5 PRUEBAS DEL CONTROLADOR PID

Una vez obtenidos los resultados de los nodos de mapeo y navegación, es importante realizar pruebas de sintonización y rendimiento de los controladores de movimiento, encargados de encontrar y enviar las órdenes a los robots móviles.

El primer controlador evaluado es el controlador PID que actúa sobre la orientación de los robots. La primera prueba requerida es la sintonización de las ganancias PID para determinar los valores de Kp, Kd y Ki.

### 3.5.1 Prueba de sintonización

Para determinar las ganancias del controlador PID, se realizaron tres pruebas con valores distintos de ganancia. La primera prueba se realizó con un valor de Kp=0.15, Ki=0 y Kd=0, la cual arrojó los resultados mostrados en la Figura 3.18.

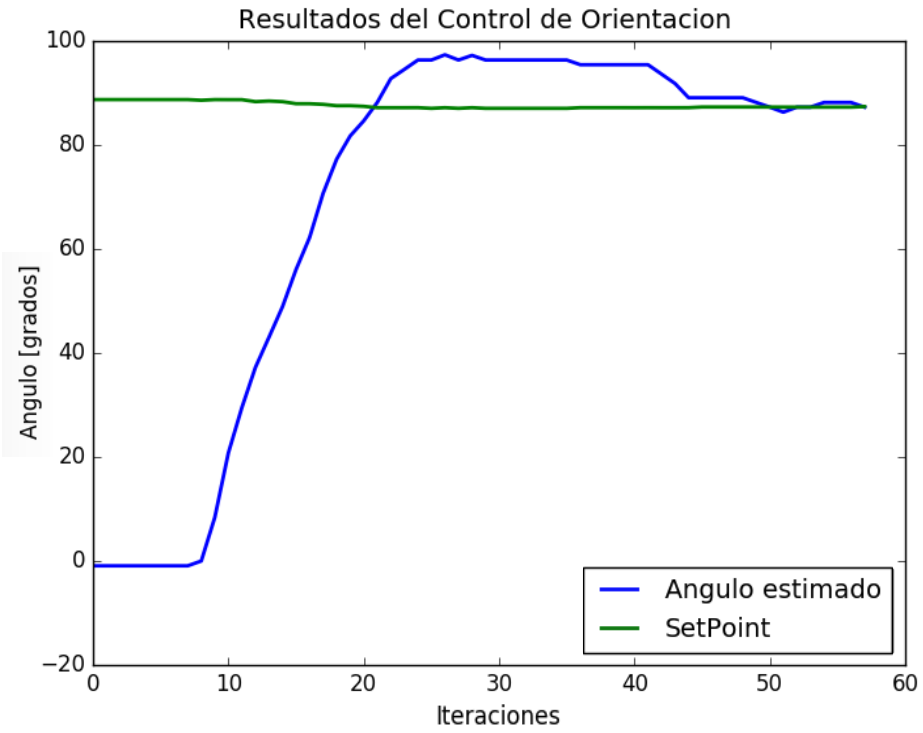



**Figura 3.18** Respuesta paso del control de orientación PID con Kp=0.15

Tras la primera prueba, se evidenció la necesidad de utilizar una ganancia menor para reducir el sobrepico existente. La respuesta deseada respecto al movimiento de los robots móviles corresponde a un desplazamiento sin sobrepicos durante la corrección del ángulo de orientación, para posteriormente realizar un movimiento frontal en

dirección al punto de ruta correspondiente. Por esto, tras disminuir la ganancia Kp a un valor de 0.13 (manteniendo las demás ganancias en 0) se registró el resultado ilustrado en la Figura 3.19.

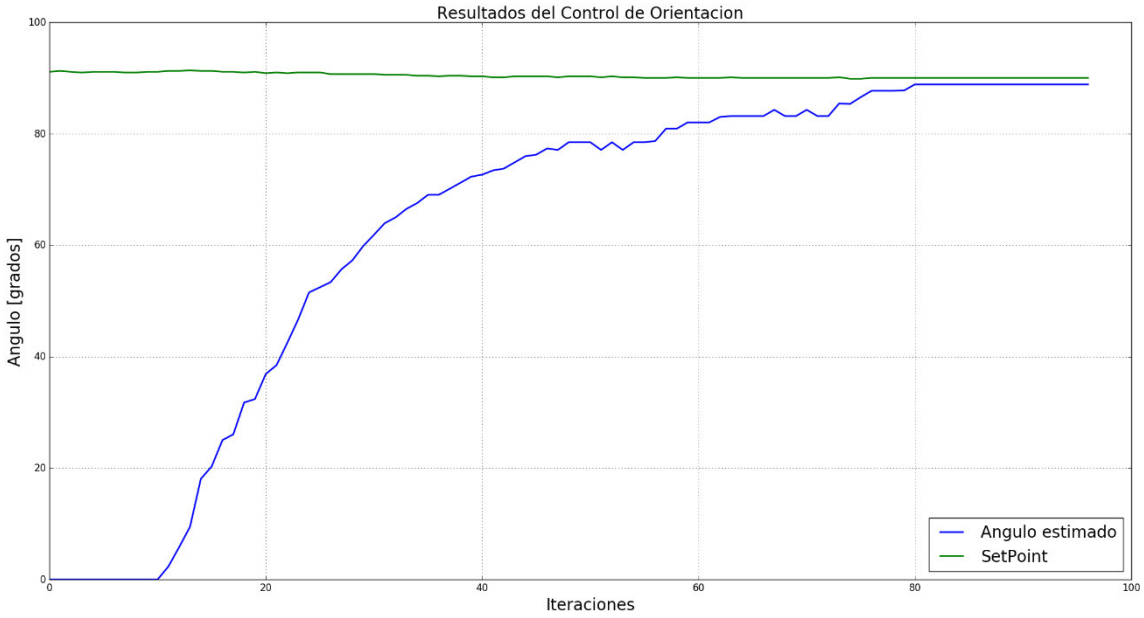



**Figura 3.19** Respuesta paso del control de orientación PID con Kp=0.13

El resultado del controlador PID solamente con ganancia Kp (Ki=0 y Kd=0), demuestra un comportamiento deseable respecto a la eliminación del sobrepico, dejando un error de posición de solamente un grado. Las pequeñas variaciones registradas en la orientación del robot móvil se deben a las pequeñas fluctuaciones en la detección de la posición de los marcadores por parte de OpenCV en el nodo de mapeo. Con las ganancias mencionadas, se obtiene un tiempo de establecimiento aproximado de 1 segundo (80 iteraciones), el cual es adecuado para la aplicación del presente proyecto. Adicionalmente, se puede observar un retardo de 100[ms] antes del movimiento del robot. Este retardo inicial se puede explicar por el tiempo requerido para realizar la adquisición inicia de datos de posición y orientación del robot.

La Figura 3.20 muestra el error obtenido en la corrección del ángulo. En el recuadro interno se aprecia el acercamiento a la etapa de establecimiento, desde la iteración 75 en adelante, donde se puede apreciar que el error de posición es menor a 1 grado. Por esto, se podría considerar un controlador apropiado para la presente aplicación.

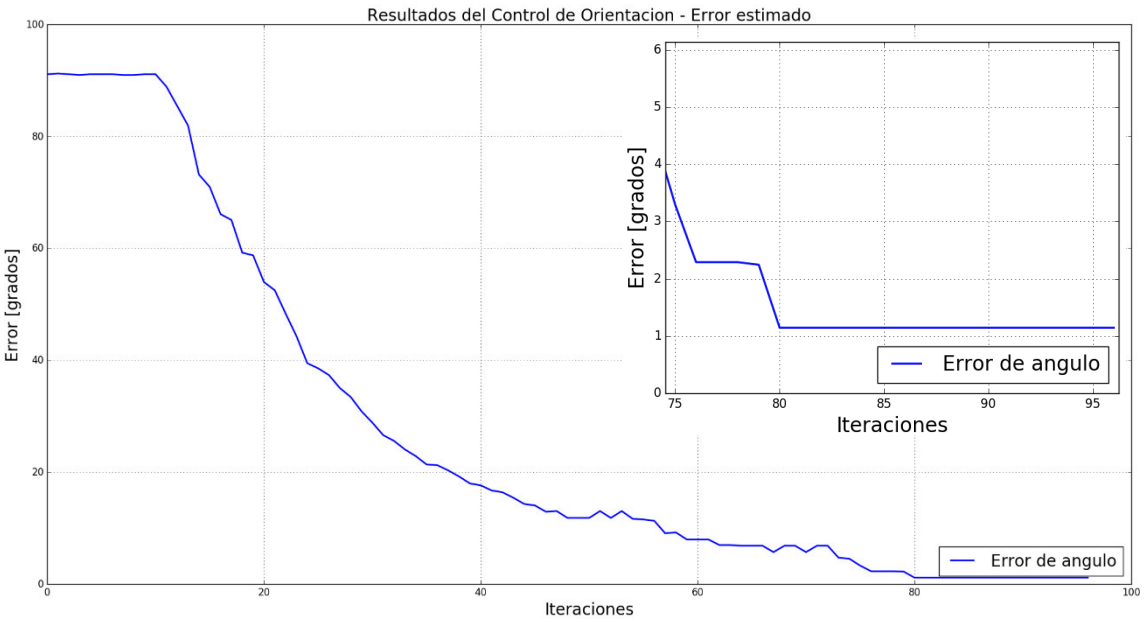



**Figura 3.20** Error estimado con control PID con Kp=0.13

Finalmente, se realizó una prueba con un control PI para intentar eliminar el error de posición. Sin embargo, dada la necesidad de contar con un valor mínimo de tolerancia en la corrección de orientación, el error de posición permitido fue configurado en 1 grado. Los resultados obtenidos con las ganancias de Kp=0.13, Ki=0.001 y Kd=0 se muestran en la Figura 3.21.

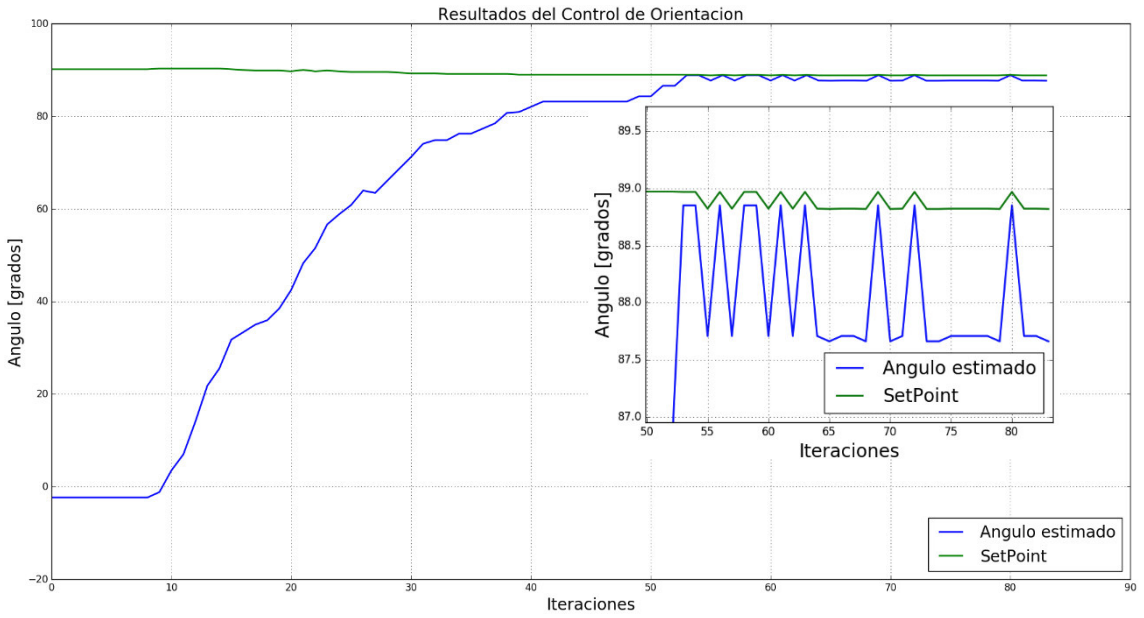



**Figura 3.21** Respuesta paso del control PID con Kp=0.13 y Ki=0.001

Se puede observar que, si bien el error de posición no es completamente eliminado, en parte por los errores de estimación, existe una clara mejora respecto al tiempo de establecimiento. Si bien con el control proporcional se lograba controlar el ángulo, el

tiempo de establecimiento era de alrededor de 80 iteraciones (aproximadamente 1s), mientras con el PI, este se reduce a cerca de 50 iteraciones (aproximadamente 0.6s). Adicionalmente, se aprecian pequeñas variaciones en el valor de Setpoint, que se explica por los pequeños errores en la estimación de posición del robot, los mismos que aparecen por ruido en las imágenes analizadas. Como se ha mencionado anteriormente, el Setpoint se calcula a partir de la posición actual del robot.

La Figura 3.22 muestra el error de posición obtenido con el controlador PI, donde se aprecia que el máximo error luego del establecimiento de la señal es de 1 grado. En el recuadro de acercamiento, se puede observar las fluctuaciones en el error de orientación del robot.

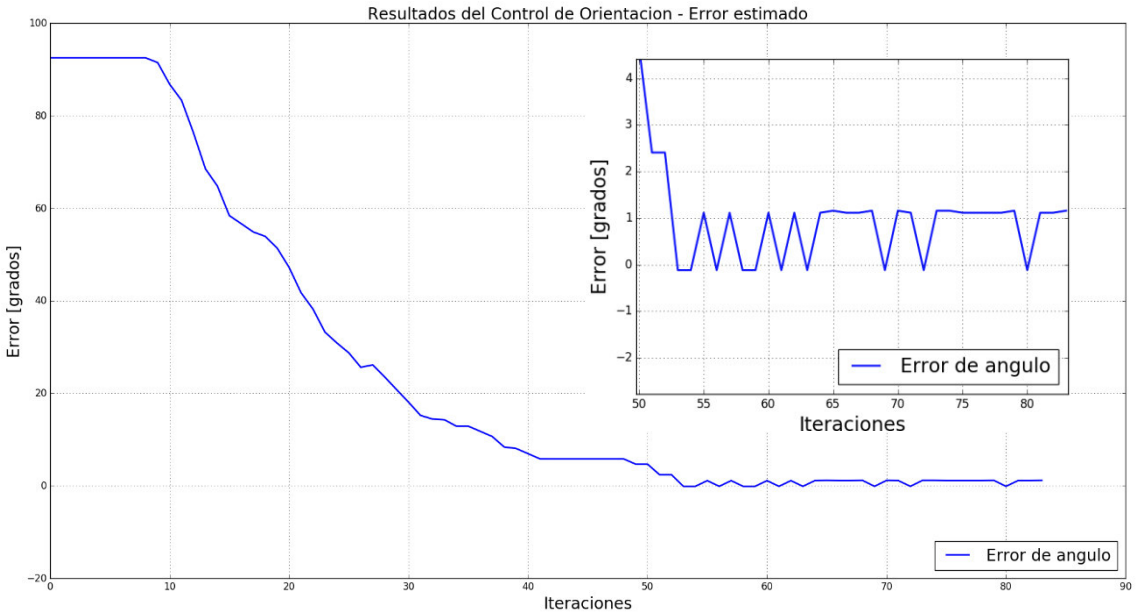



**Figura 3.22** Error estimado con PI Kp=0.13 y Ki-0.001

A pesar de que en las pruebas de sintonización, el control PI demostró resultados mejores que el controlador proporcional, en las pruebas de funcionamiento se observó que el término integral introdujo efectos negativos en algunos casos. Específicamente, en algunas ocasiones se observó sobrepicos en el posicionamiento de los robots. Por esto, se utilizará un control proporcional con ganancia Kp=0.13 para realizar las pruebas de seguimiento de rutas.

La Figura 3.23 muestra una comparación durante el seguimiento de una ruta con el control Proporcional y Proporcional-Integral respecto al error de orientación. En la imagen se observa que, con el control Proporcional-Integral, la corrección de orientación adelanta al controlador Proporcional gracias al menor tiempo de establecimiento observado previamente. Sin embargo, en las partes señaladas, se

observa también que puede generar sobrepicos de error, por lo que finalmente conviene utilizar el control Proporcional que reduce dichos sobrepicos.

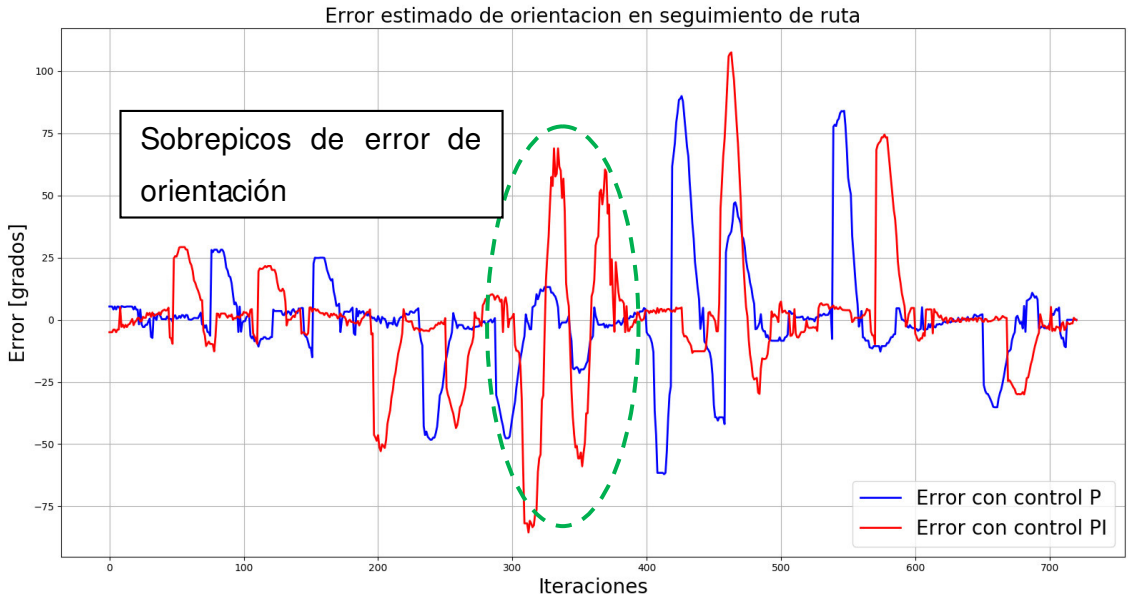



**Figura 3.23** Comparación entre control P y PI durante seguimiento de ruta.

**Análisis de índices de desempeño**

Para determinar de manera cuantitativa el desempeño de un controlador, es pertinente observar índices como la Variación total de Control (TVu) y la Integral del Cuadrado del Error (ISE), generados durante su ejecución.

Las pruebas de sintonización realizadas con Kp=0.13 y Kp=0.13 más Ki=0.001 arrojaron los resultados observados en la Figura 3.24 y la Figura 3.25.

Se puede apreciar que el control PI demuestra mejores características en ambos indicadores. El ISE es menor, así como el índice de TVu, lo que corrobora lo observado en las curvas de reacción, donde se observó un menor tiempo de establecimiento al aplicar el termino integral.

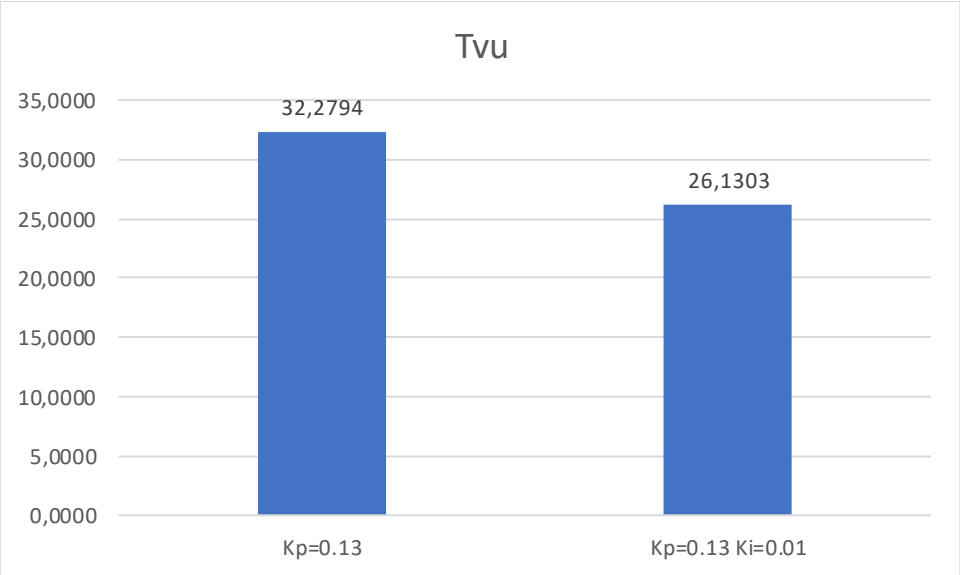



**Figura 3.24** Comparación de TVu entre P y PI.

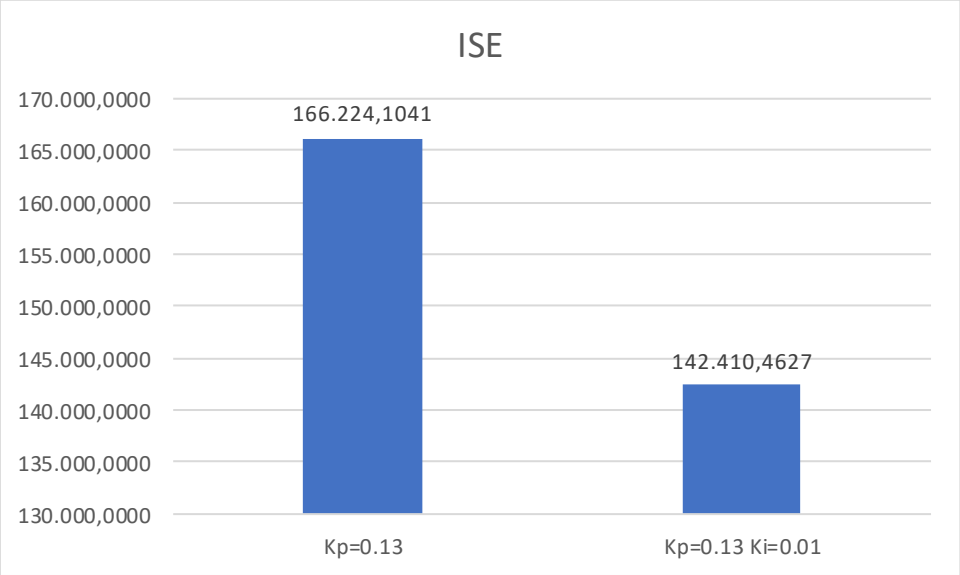



**Figura 3.25** Comparación de ISE entre P y PI.

Como se mencionó anteriormente, a pesar de que el control PI ofrece resultados mejores en pruebas de respuesta paso, durante la operación mostró comportamientos no deseables, por lo que el control utilizado en las siguientes pruebas es de tipo proporcional.

### 3.5.2 Seguimiento de ruta

Tras determinar las ganancias del controlador para la etapa de corrección de ángulo, se presentarán los resultados obtenidos tras utilizar el controlador PID para realizar el seguimiento de una ruta generada por el nodo de planificación. La Figura 3.26 muestra la interfaz gráfica de planificación de rutas una vez realizado el cálculo de la ruta para el robot con ID número 1, así como los resultados de esta.

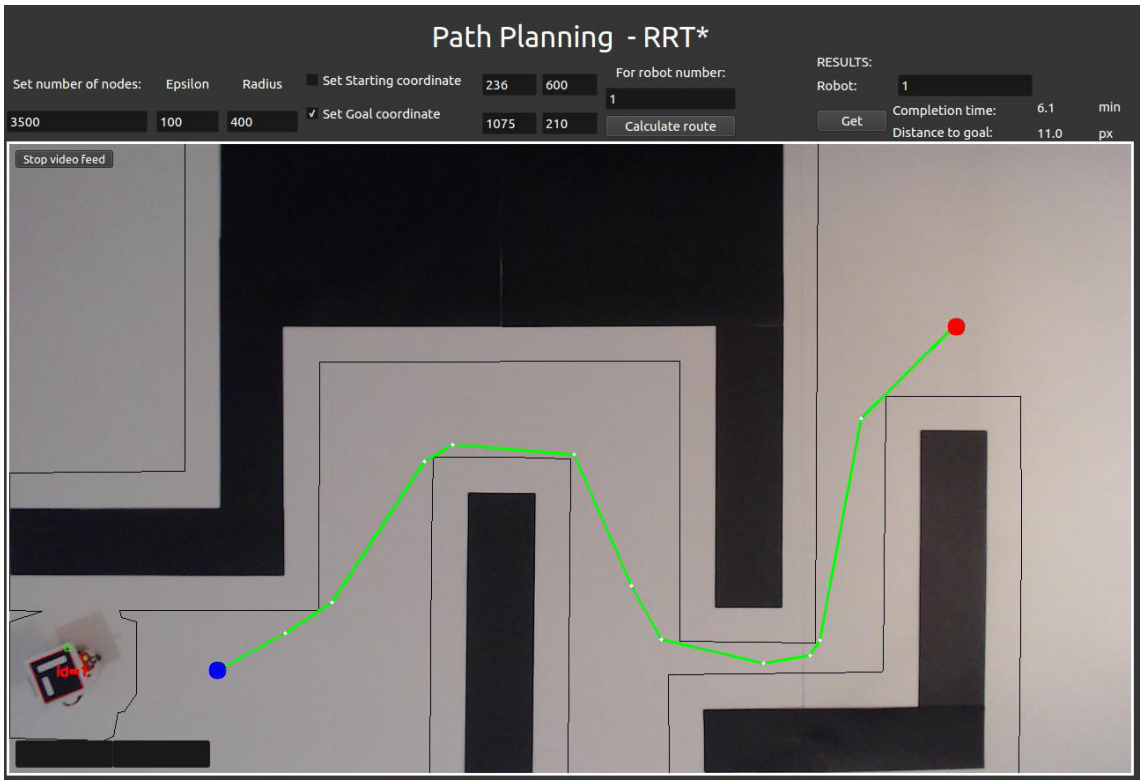



**Figura 3.26** Planificación de ruta en entorno tipo laberinto.

Como se puede observar, el tiempo de cálculo fue de 6.1 minutos y la distancia desde el nodo final hasta el punto meta de 11 pixeles (0.9 cm). La ruta mostrada se obtuvo con 3500 nodos generados, un valor de ε=100 y $R$=400. Los valores de ε y $R$ fueron seleccionados para obtener un árbol RRT* que produzca tramos rectos extensos y curvas que minimicen el trayecto desde el punto inicial al punto final.

La Figura 3.27 muestra la topología del árbol generado por el algoritmo, en la cual se aprecia la extensión del árbol RRT* en el entorno libre del espacio de trabajo. Se puede observar que el valor de $R$ permite eliminar nodos intermedios en tramos que no requieren cambios de dirección del robot. De la misma manera, el valor de épsilon permite generar nodos que cubran una distancia apropiada para extenderse en el entorno cubriendo esquinas sin desperdiciar nodos creados a distancias pequeñas que no aporten a la extensión rápida del árbol.

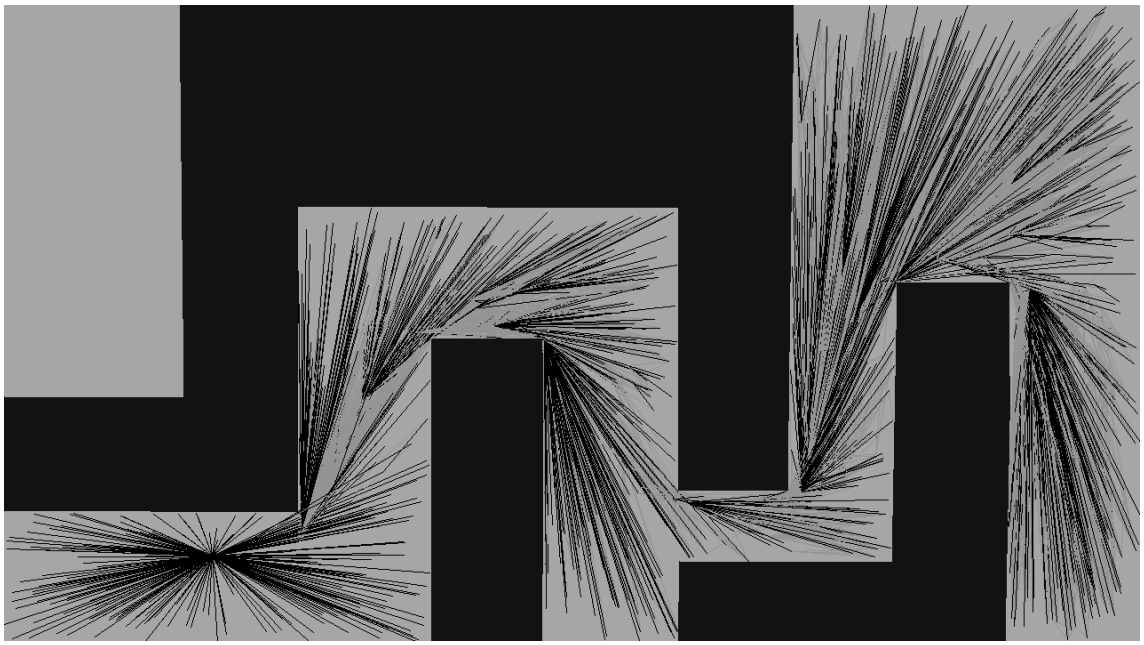


**Figura 3.27** Árbol generado con RRT* en entorno tipo laberinto.

El seguimiento de la ruta es ejecutado desde el nodo de control master, cuya interfaz se muestra en la Figura 3.28. Se observa el robot número 1 cerca al punto inicial de la ruta calculada, así como los botones de comando de este en la parte superior de la interfaz.

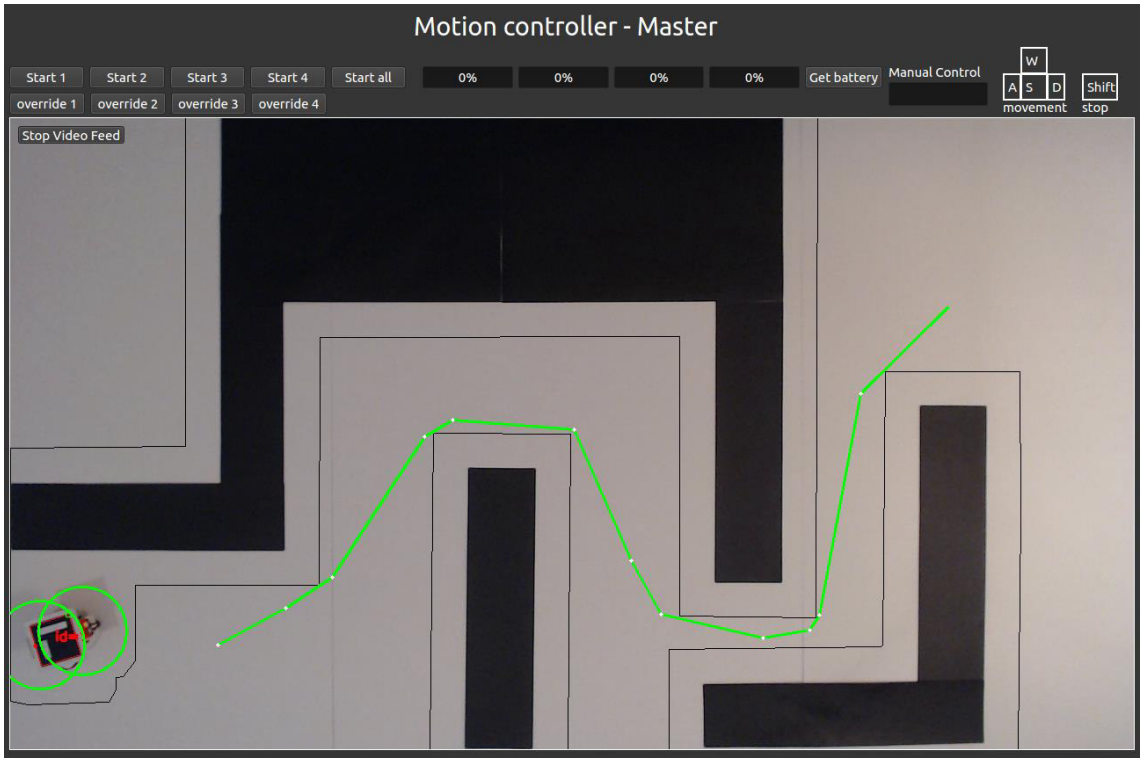



**Figura 3.28** Control de movimiento para seguimiento de ruta en entorno tipo laberinto.

Un video del proceso de seguimiento de ruta se encuentra en el link detallado en el Anexo C, ubicado al final del presente documento. La presente prueba fue realizada con un valor de rapidez lineal configurado de 8 [mm/s].

La Figura 3.29 fue generada tras la finalización del seguimiento de ruta, tras haber almacenado e ilustrado los datos de posición del robot durante la ejecución del control. Se puede observar que el control logra efectivamente llevar al robot desde el punto inicial al punto final sin generar desviaciones significativas. La posición final de robot puede ser identificada por la silueta del mismo que se dibuja en el nodo final de la ruta. Dicha silueta corresponde al reconocimiento del robot como parte de los objetos presentes en el área, ya que este debe ser considerado como un obstáculo para los demás robots.

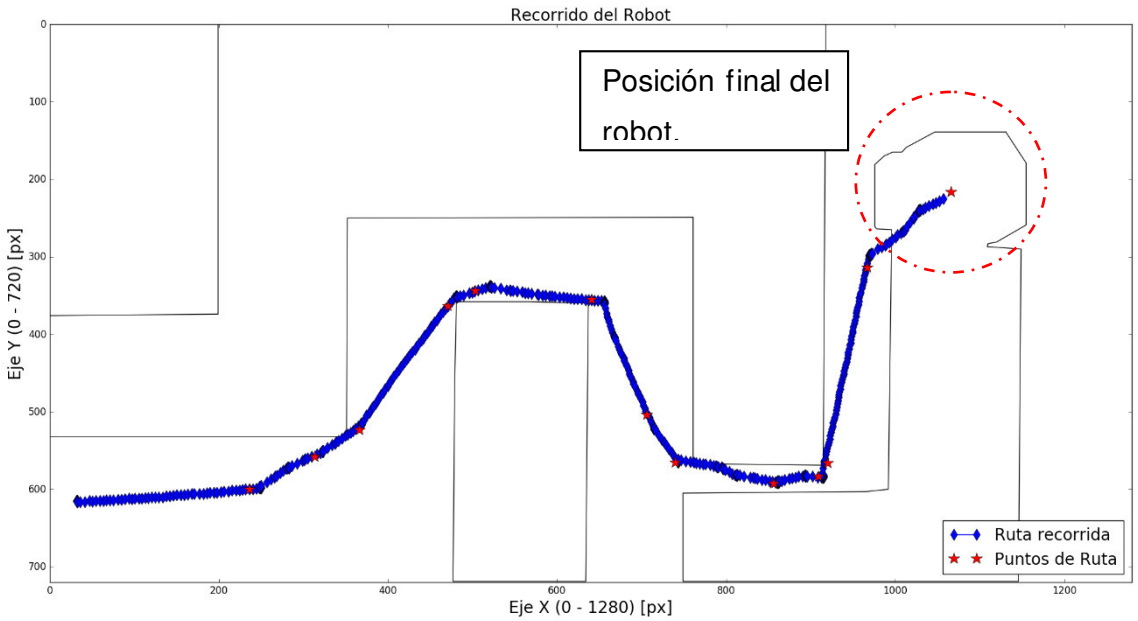



**Figura 3.29** Recorrido registrado del robot móvil en entorno tipo laberinto.

Si bien el resultado final del seguimiento de ruta puede ser descrito como exitoso, es también de interés analizar a detalle el desempeño del controlador a través de las señales ilustradas en la Figura 3.30, donde se muestra el resultado del control de orientación PID, en la cual se grafican los datos de orientación del robot en cada iteración de control así como el ángulo de referencia o Setpoint.

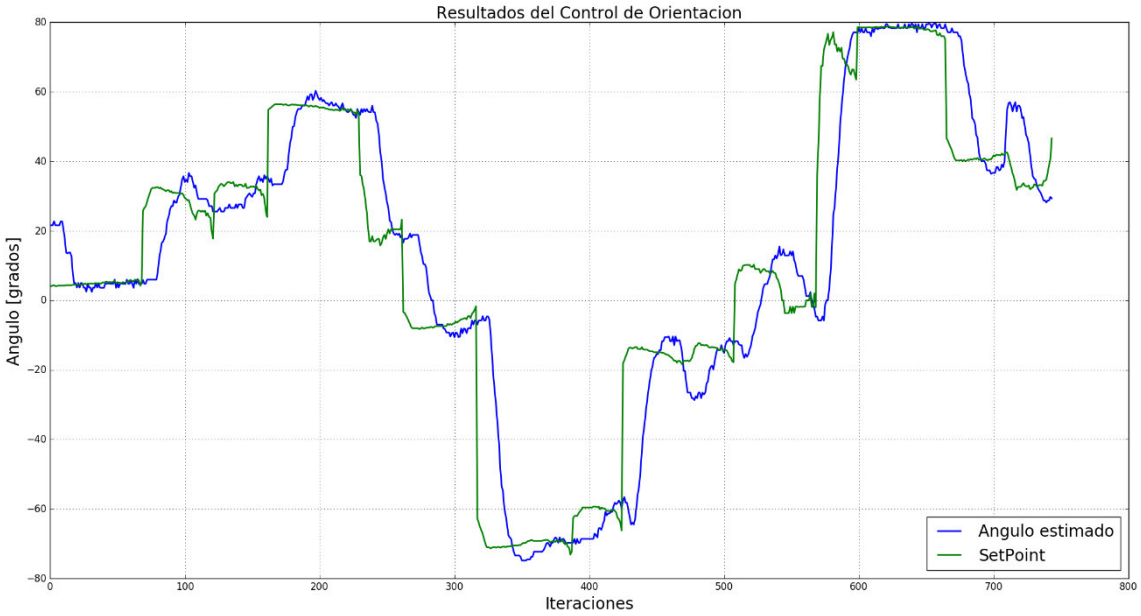



**Figura 3.30** Resultados del controlador de orientación en entorno tipo laberinto.

Es necesario precisar que el ángulo de referencia varía de acuerdo con la posición del robot respecto al eje de coordenadas. Puesto que, durante todo el movimiento del robot, su posición cambia constantemente, el Setpoint varia de manera acorde. Justamente, la Figura 3.30 muestra que el robot móvil reacciona a los cambios de referencia que se producen cada que el robot llega a uno de los nodos que conforman la ruta. Además, durante los trayectos intermedios, la orientación del robot también se ajusta de manera constante para mantener el menor error posible en su orientación.

En los puntos de cambio de referencia se puede apreciar un pequeño retardo en la reacción del robot, el cual existe durante un corto intervalo de tiempo. Dicho retardo existe como parte del procedimiento de cambio entre la etapa de corrección de orientación sin velocidad lineal y la etapa de desplazamiento lineal del robot. Justamente, la Figura 3.31 muestra la diferencia entre la etapa de corrección de orientación y la etapa de desplazamiento en referencia a la rapidez angular del robot durante el seguimiento de una ruta. Se puede observar que durante la etapa de corrección de orientación la rapidez angular es elevada y se reduce de acorde al movimiento del robot; mientras que, durante la etapa de desplazamiento, esta se mantiene cercana a cero, mientras el robot se desplaza de manera frontal hacia la coordenada correspondiente.

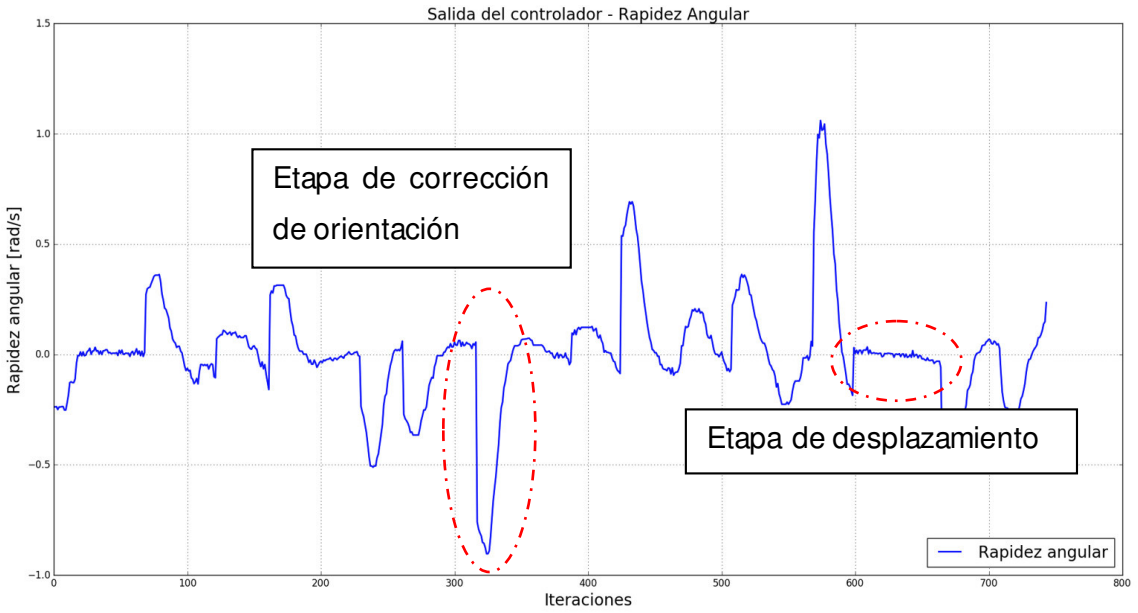



**Figura 3.31** Señal del controlador de rapidez angular en entorno tipo laberinto

La Figura 3.31 ilustra la salida del control respecto a la rapidez angular del robot. Se puede observar que solo existen cambios en la rapidez angular durante la etapa de corrección, que es cuando el robot móvil tiene rapidez lineal de 0[mm/s].

Una métrica útil para evaluar el desempeño del controlador es la gráfica del error. La Figura 3.32 muestra la curva del error de orientación.

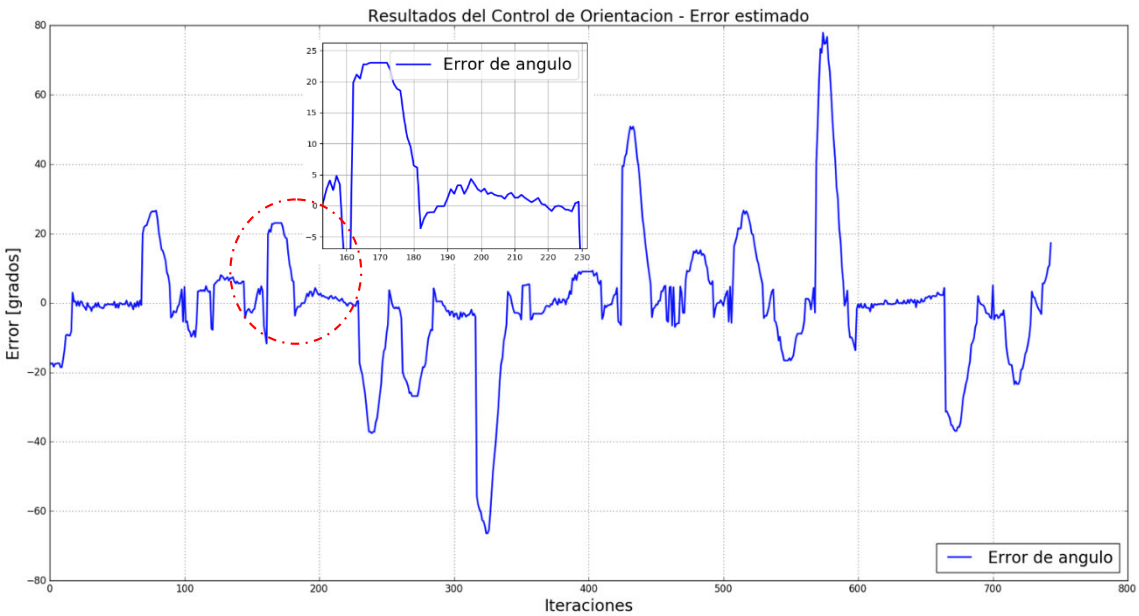



**Figura 3.32** Error de ángulo de orientación en entorno tipo laberinto.

Es posible apreciar que durante los trayectos en que el robot se desplaza, el error de orientación se mantiene en valores muy cercanos a 0. Dicha variación de error alrededor del 0 se debe a un rango de tolerancia programado (5 grados) para evitar

que el robot constantemente se detenga mientras ajusta su orientación. Adicionalmente, este margen de tolerancia en la orientación del robot es indispensable ya que la orientación de cada robot es estimada a partir de las imágenes recogidas por la cámara, dicha estimación inherentemente conlleva errores, los cuales se evidencian en la variación registrada incluso cuando el robot se mantiene estático. Esta característica se puede apreciar en el recuadro de acercamiento de la **Figura 3.32**, donde se observa que la variación del error no supera los 5 grados configurados como margen de tolerancia.

Adicionalmente, la Figura 3.33 muestra la evolución de los valores de error de posición. Ya que el control de orientación no corrige directamente el error de posición del robot, dicha figura no está directamente relacionada con el desempeño del controlador, sin embargo, es de interés observar el comportamiento del robot respecto a su ubicación en el entorno durante la ejecución del control de orientación.

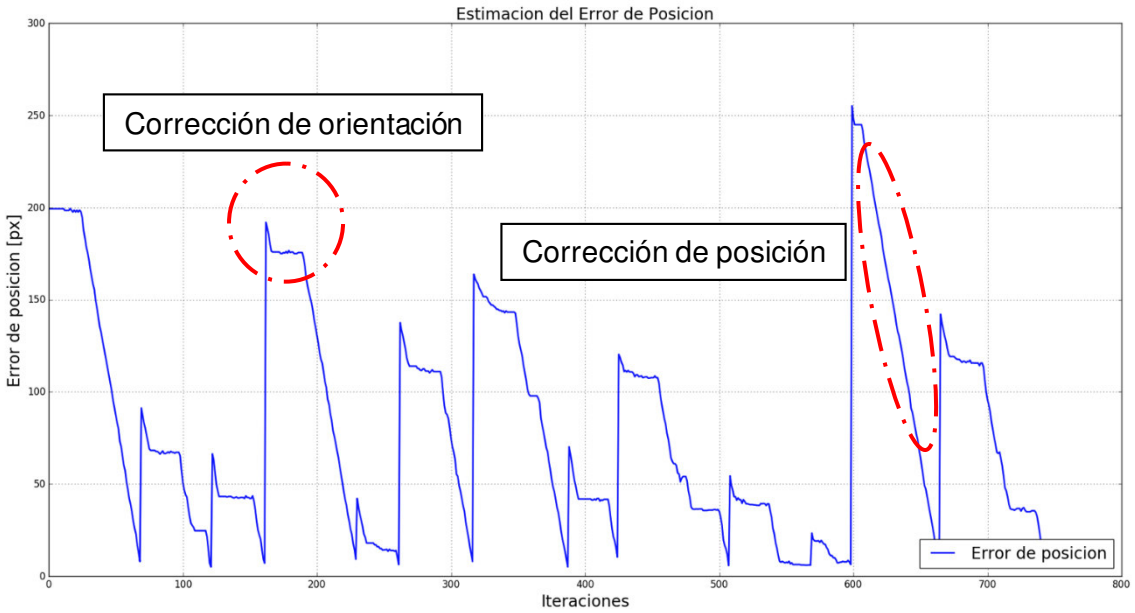



**Figura 3.33** Error de posición durante seguimiento de ruta con control de orientación.

Es evidente que el error de posición no disminuye significativamente durante la etapa de corrección de la orientación, solo lo hace durante la etapa de desplazamiento del robot. Se puede apreciar que dicha diminución ocurre con una pendiente constante ya que el desplazamiento del robot es con una velocidad lineal fija de 8[mm/s], la cual se mantiene en 0 [mm/s] durante la corrección del ángulo.

**Pruebas con distintos valores de rapidez lineal.**

Adicionalmente, se realizaron dos pruebas con distintos valores de rapidez lineal, con el objetivo de conocer la respuesta y desempeño del controlador en diferentes situaciones.

Con el objetivo de poder observar la diferencia de desempeño del controlador sintonizado y los valores de ganancias finales, se realizaron experimentos con 3 valores de velocidad lineal: 4mm/s, 8mm/s y 16mm/s. La prueba examinada anteriormente se realizó con una velocidad de 8mm/s. La Figura 3.34 y

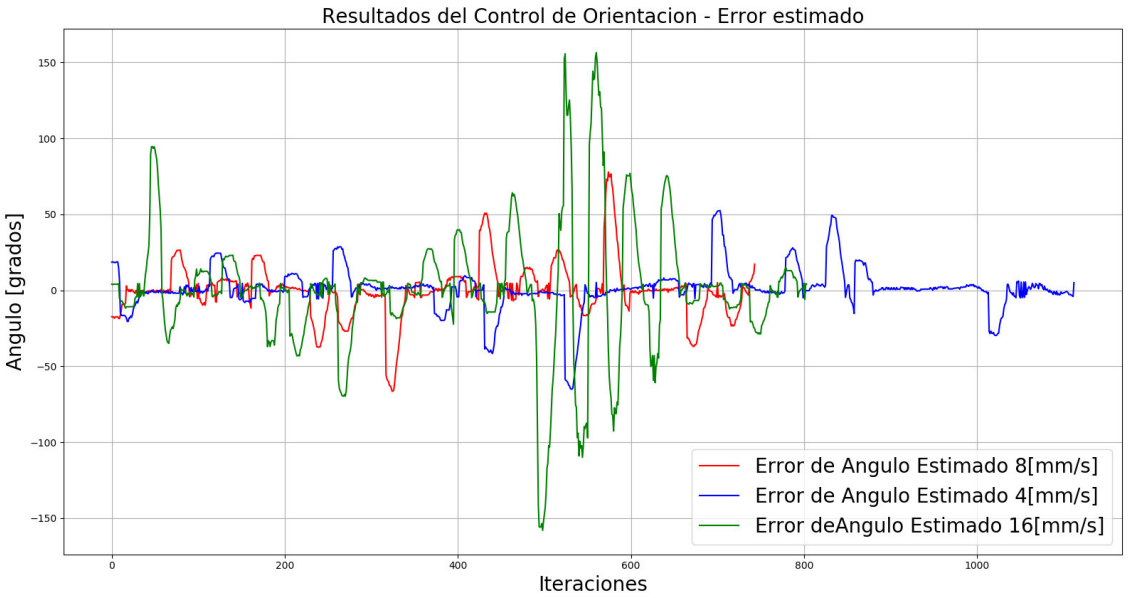



Figura 3.35 muestran una comparación de los resultados obtenidos con las tres velocidades mencionadas.

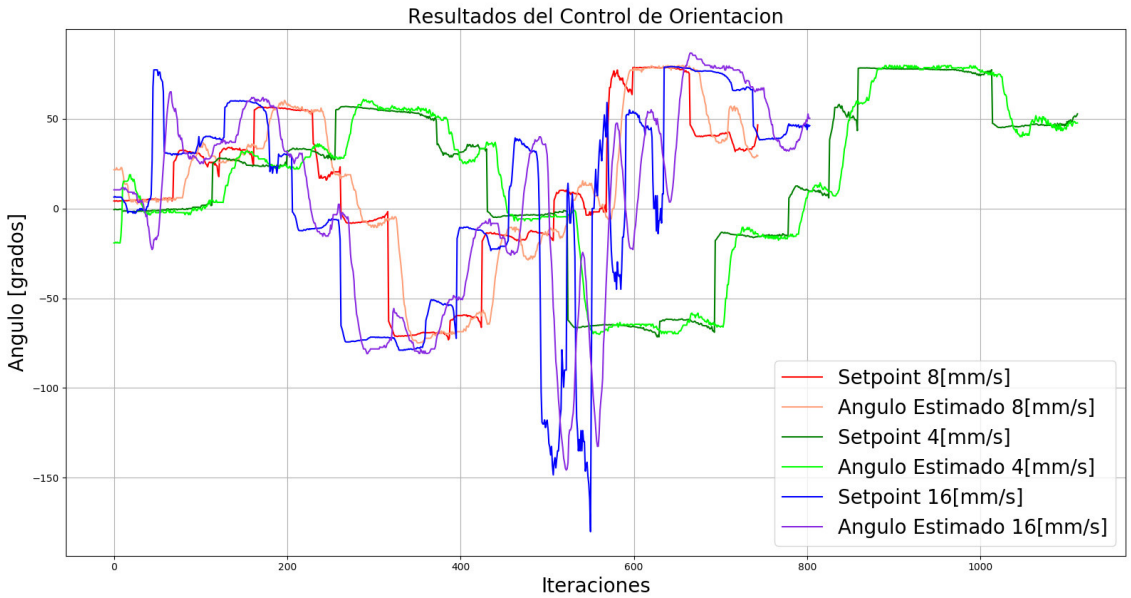



**Figura 3.34** Resultados de control de orientación con distintos valores de rapidez lineal.

Gracias a la mayor velocidad lineal, los resultados obtenidos con una rapidez de 16 [mm/s] muestran que la ruta fue completada en un tiempo menor, evidenciado por el menor número de iteraciones utilizadas por el robot en desplazarse desde el punto inicial a la meta. Se observa también que al utilizar una rapidez lineal de 4 [mm/s] se obtiene un seguimiento de la referencia mucho más exacto, a cambio de completar la tarea de seguimiento de ruta en un tiempo mucho mayor.

La

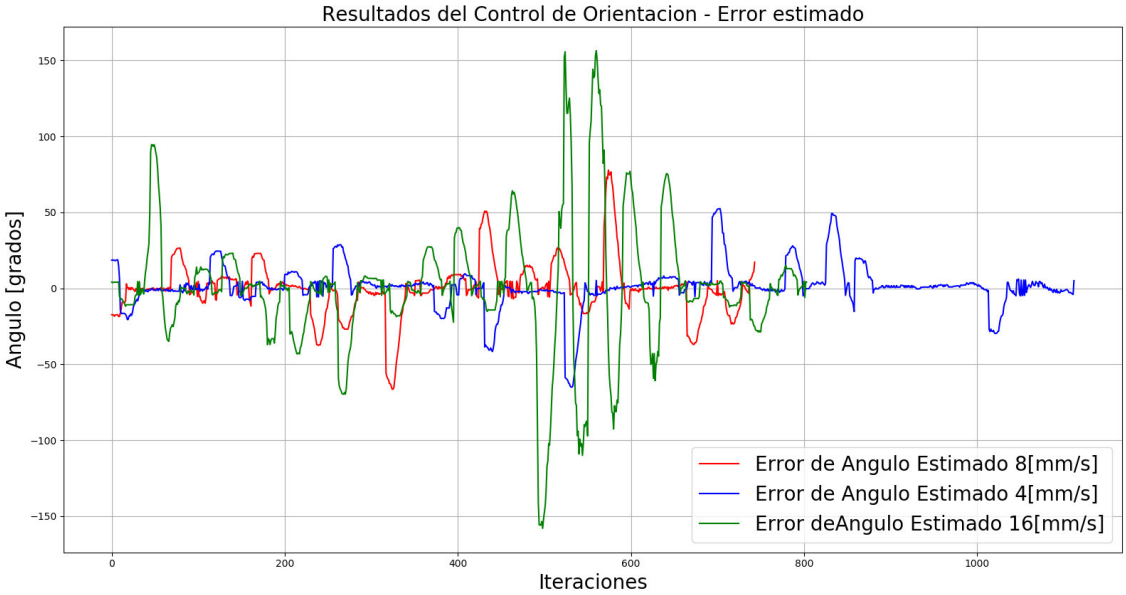



Figura 3.35 compara la señal de error de orientación entre los tres casos mencionados. Se puede observar que durante los trayectos en que el robot se desplaza de manera lineal, el error de orientación con velocidad de 4 [mm/s], no sufre variaciones mayores; mientras que cuando el robot se mueve con 16 [mm/s] el error varía en mayor amplitud. Se observa también que existe una relación entre la rapidez lineal y el máximo valor de error de orientación; cuando se utilizan velocidades altas, el error máximo aumenta; mientras que, con una rapidez lineal baja, el error máximo es menor.

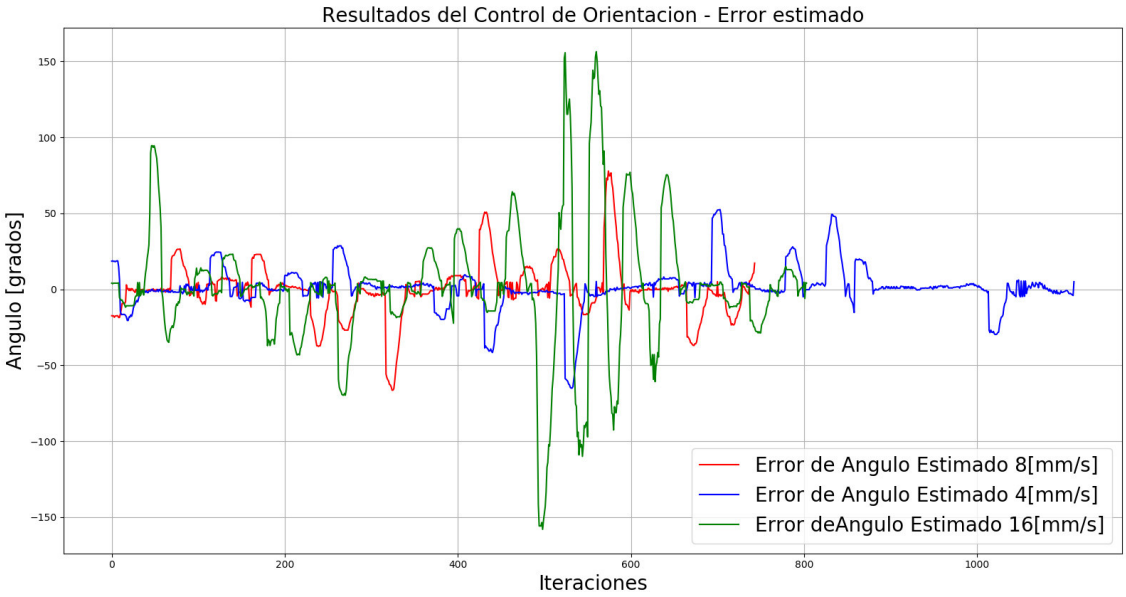



Figura 3.35 Error estimado de orientación con distintos valores de rapidez lineal.

La Tabla 3.3 muestra una comparación de los resultados con los tres valores de rapidez lineal analizados. Se puede apreciar que el índice ISE aumenta al utilizar valores más altos de rapidez dada la imprecisión del control bajo estas condiciones.

Es de interés también mencionar que a pesar de que al aplicar una rapidez lineal de 16 [mm/s] se esperaba que la tarea se termine en menor tiempo que con velocidades inferiores, el tiempo requerido para completar la tarea fue un poco mayor que con 8 [mm/s]. Este fenómeno se explica por el tiempo desperdiciado en corregir los altos errores de orientación producidos durante el recorrido del robot.

**Tabla 3.3** Comparación de resultados para tres valores de rapidez lineal con control PID.

| | Iteraciones de control para completar la ruta | Error máximo [grados] | ISE |
|---|---|---|---|
| PID 4 [mm/s] | 1113 | 52.532 | 213238.584 |
| PID 8 [mm/s] | 744 | 77.857 | 244513.587 |
| PID 16 [mm/s] | 804 | 156.421 | 1454414.828 |

## 3.6 RESULTADOS DE CONTROLDOR DE CINEMATICA INVERSA (CCI)

El segundo controlador utilizado para ejecutar el movimiento de los robots móviles sobre las rutas calculadas fue de Cinemática Inversa, el cual se basa en el cálculo directo de la velocidad lineal y angular a partir de la inversa de la matriz Jacobiana y el error de posición, como se detalló en la Sección 2.3. Justamente, la matriz de ganancia, definida en la ecuación (14), regula la acción de control, permitiendo aumentar o disminuir la agresividad del mismo. Con valores mayores de ganancia, los robots móviles se desplazan a velocidades mayores, pero siempre llegarán con rapidez 0 a cada punto de la ruta, evitando sobrepicos en su posición respecto a la referencia. El objetivo del control es siempre reducir el error de posición a un valor de 0.

Se debe aclarar que la presente prueba se realiza con la misma ruta utilizada para la evaluación del controlador PID, mostrada en la Figura 3.26.

### 3.6.1 Seguimiento de ruta

Se utilizaron 3 valores de ganancia para obtener los resultados presentados en esta sección. Una ganancia de $K = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix}$ fue seleccionada como un valor adecuado que satisface los requerimientos del sistema respecto al desempeño de los robots, su velocidad y trayectoria que estos siguen durante la operación. Otros dos experimentos, con $K = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix}$ y $K = \begin{bmatrix} 7 & 0 \\ 0 & 7 \end{bmatrix}$ también fueron ejecutados.

**Resultados con $\boldsymbol{K = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix}}$**

La Figura 3.36 muestra la trayectoria recorrida por el robot en un entorno provisto de obstáculos dispersos. Este resultado muestra que el controlador es efectivo en completar la ruta generada, llevando al robot hasta el nodo final con trayectos que conectan todos los nodos intermedios. Respecto a los resultados presentados anteriormente, se puede observar que el recorrido del robot es más curvilíneo, conectando cada nodo con movimientos más continuos.

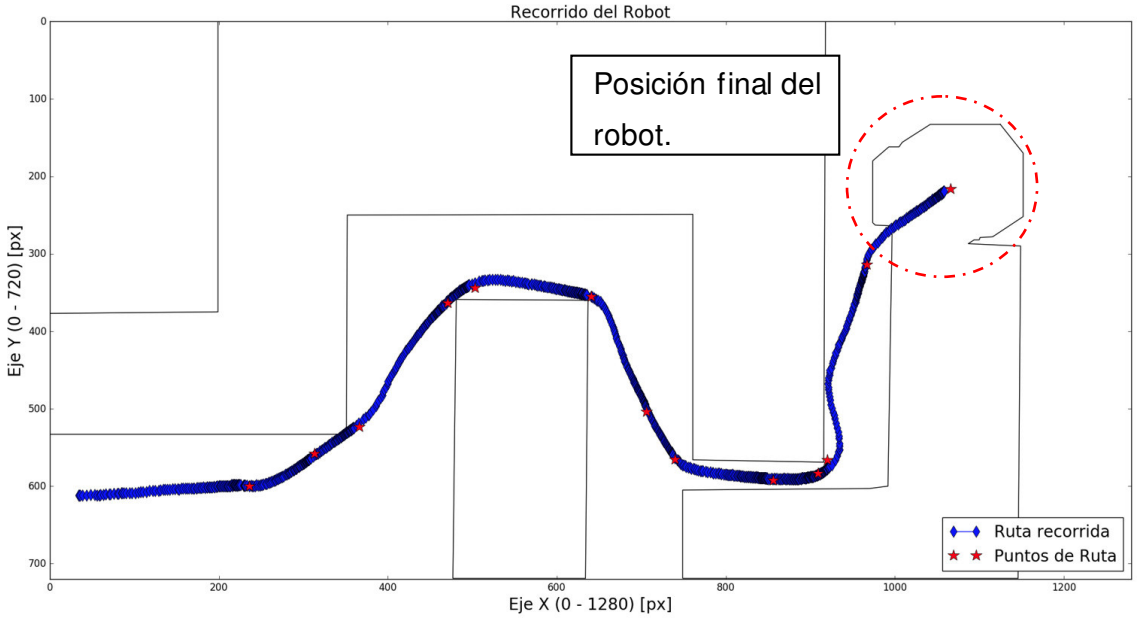



**Figura 3.36** Recorrido registrado del robot con K=3.

La Figura 3.37 y Figura 3.38 muestran los resultados de la orientación del robot. Estas imágenes son presentadas con propósitos de comparación del comportamiento del controlador respecto al controlador de la sección 3.5.2. Evidentemente, en el caso del controlador de Cinemática Inversa, el error de orientación no se reduce a 0 ni el ángulo del robot coincide con la orientación hacia cada nodo intermedio. Sin embargo, es de interés observar que completar la ruta calculada no requiere necesariamente de corregir la orientación del robot constantemente para llevarlo a su punto final de manera eficiente.

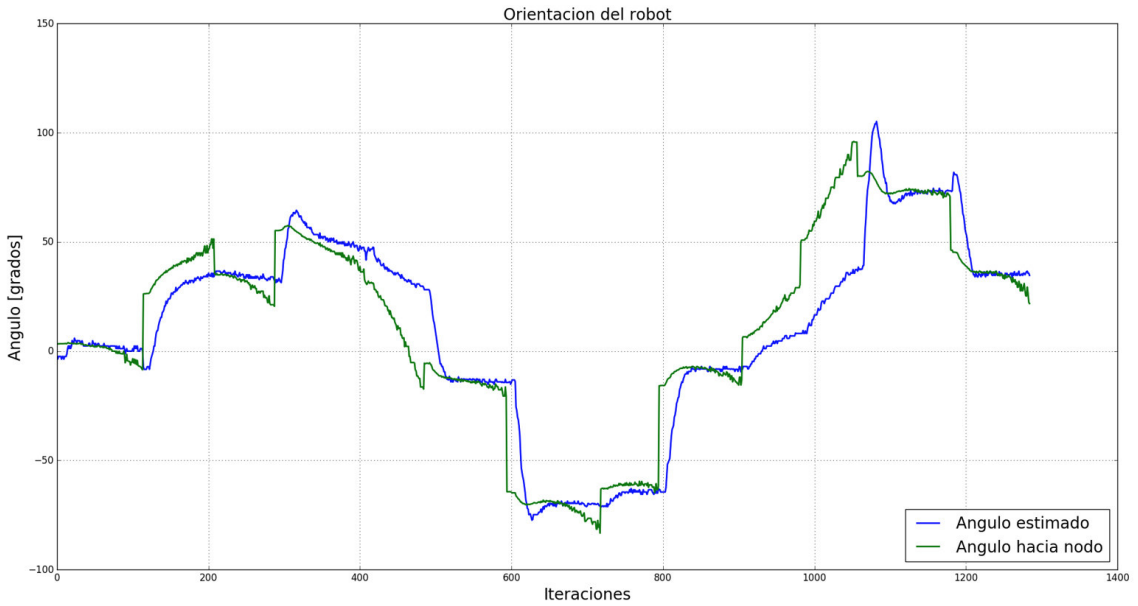



Figura 3.37 Respuesta de Orientación del robot con CCI

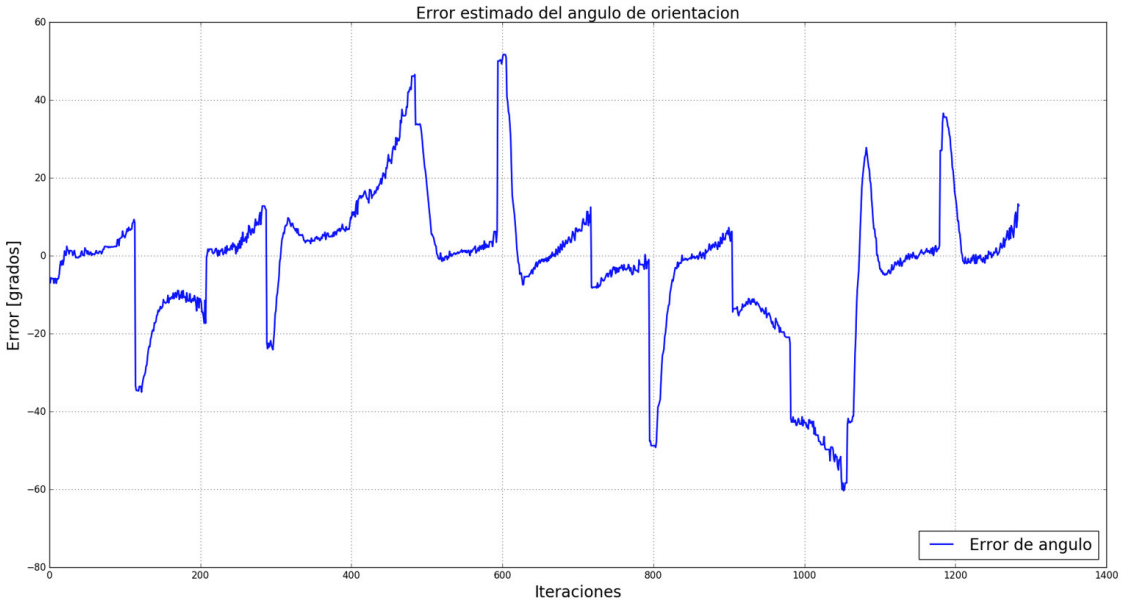



**Figura 3.38** Error de ángulo de orientación con CCI.

Como en el caso anterior, analizar la salida del controlador es importante para determinar su comportamiento sobre los actuadores. En el caso del CCI existen dos salidas: rapidez lineal y rapidez angular. Ambos valores son convertidos a RPM para ser enviados a cada robot.

La Figura 3.39 y Figura 3.40 muestran dichos valores de rapidez angular y lineal. Se observa que el máximo valor de rapidez angular no llega a 1.25[rad/s], lo cual permite una operación suave de los robots móviles, sin cambios bruscos de orientación ni movimientos erráticos. En la Figura 3.39 existen valores positivos y negativos de rotación, donde el signo de la rapidez angular determina la dirección de rotación.

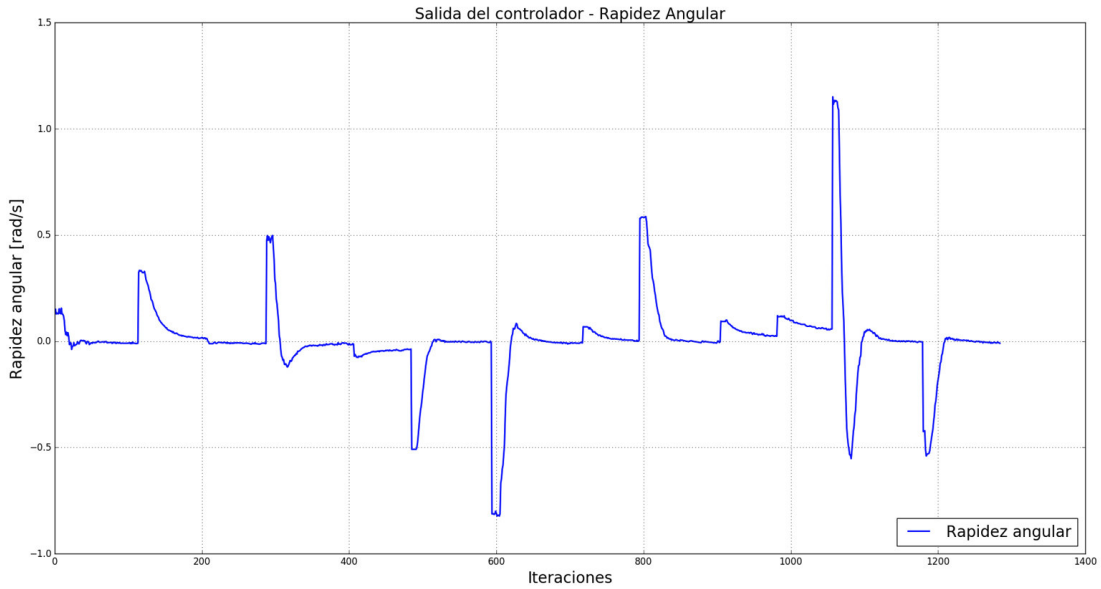



**Figura 3.39** Salida del CCI respecto a rapidez angular.

La Figura 3.40 muestra la salida del controlador respecto a rapidez lineal. Es evidente que la rapidez del robot es mayor cuando se encuentra alejado de cada nodo. Una vez que la distancia robot-nodo se reduce, la rapidez lineal disminuye de manera acorde. Dado que la matriz $K$ actúa como una constante proporcional, el valor de ganancia determinará la máxima rapidez a la que el robot se desplazará.

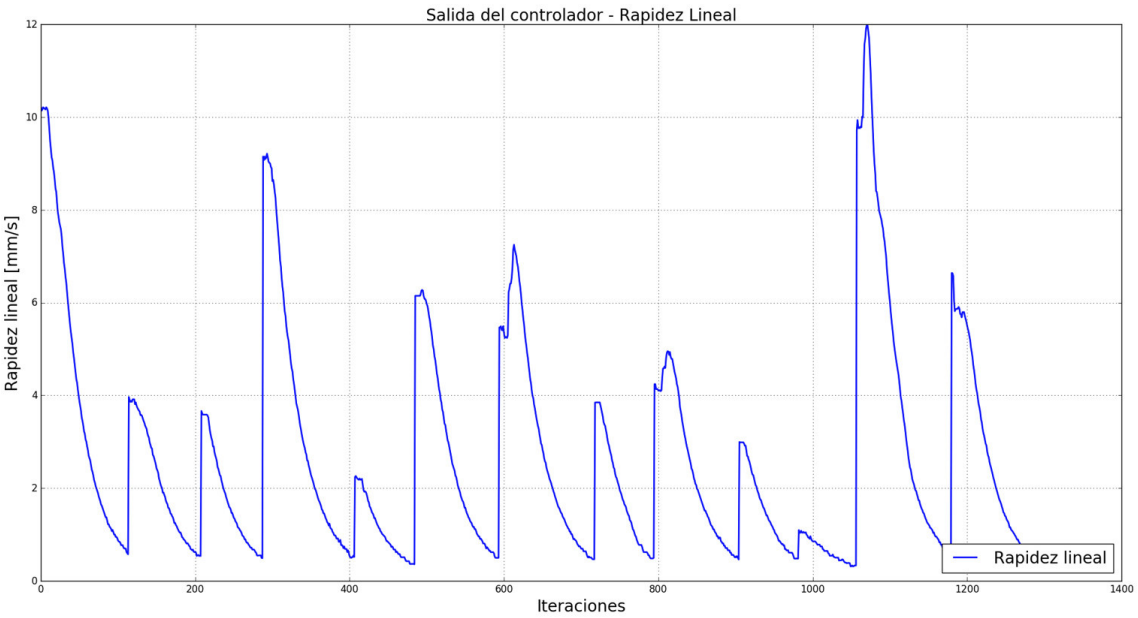



**Figura 3.40** Salida del CCI respecto a rapidez lineal.

Como se ha mencionado, para comandar los robots se requiere enviar los datos de velocidad de cada llanta de manera individual. Por esto es también necesario analizar la salida del controlador una vez convertida a valores RPM para el robot. La Figura 3.41 muestra que en el escenario propuesto, y con la ganancia configurada, la mayor rapidez rotacional fue de poco más que 50 RPM. Si bien los actuadores, o motores, no corren riesgo de avería por ejecutar las velocidades requeridas por el controlador, es necesario tener en cuenta que según las especificaciones del fabricante, los motores tienen un límite de 150 RPM al ser aplicado un voltaje de 4.5 V. Ya que en la aplicación del presente trabajo, el voltaje aplicado es de máximo 4.2V, se conoce de antemano que no se ejercerán voltajes excesivos sobre los motores.

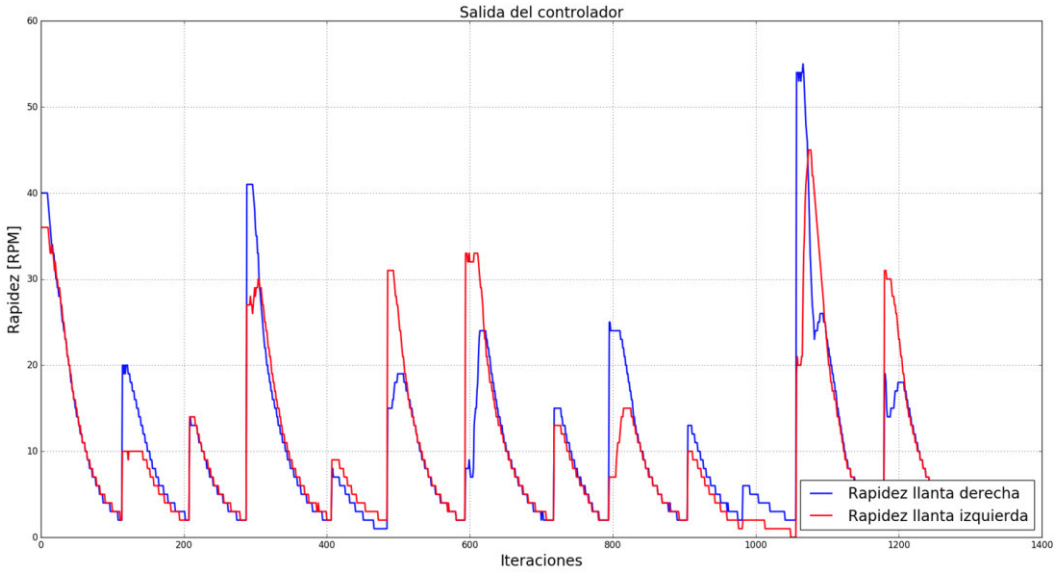



**Figura 3.41** Salida del CCI en RPM para rapidez de llanta.

Finalmente, los valores de error de posición se presentan en la Figura 3.42. Se puede apreciar que la disminución del error de posición se produce casi de manera instantánea, tras la llegada del robot a cada nodo de la ruta. Adicionalmente, es importante precisar que la disminución del error no se produce de manera lineal, pues es el resultado de la combinación de velocidad lineal y angular del robot.

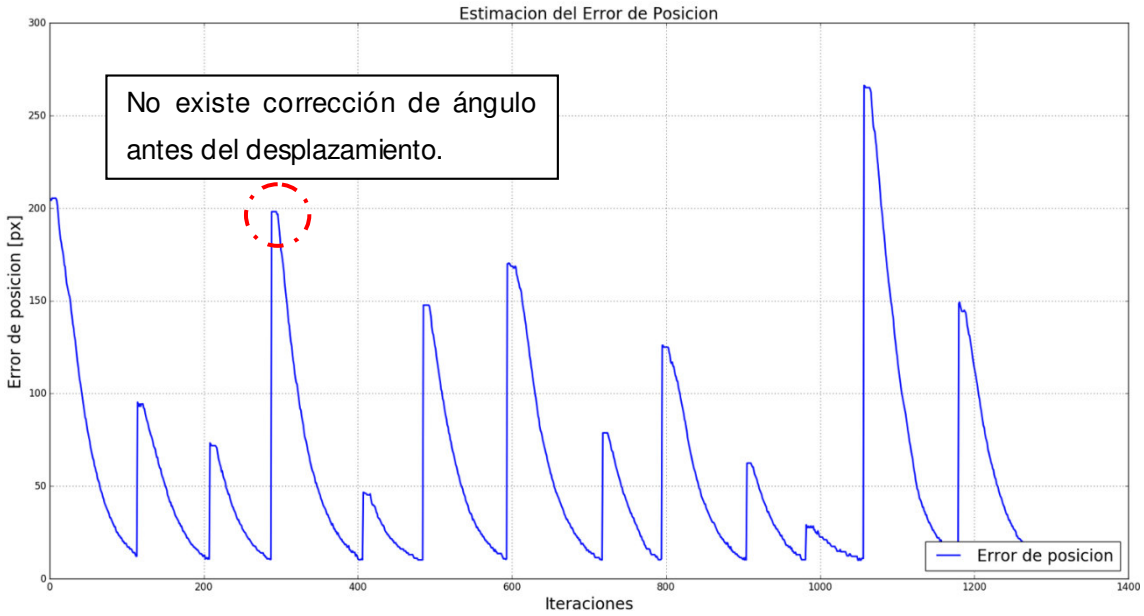



**Figura 3.42** Error de posición con CCI.

**Resultados con $K = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix}$ y $K = \begin{bmatrix} 7 & 0 \\ 0 & 7 \end{bmatrix}$**

Las pruebas realizadas con distintos valores de la matriz de ganancia permiten conocer la respuesta del sistema y comparar los resultados. Como se ha mencionado,

el controlador actúa sobre el error de posición entre el robot y cada nodo de la ruta correspondiente, por lo que valores superiores de ganancia resultan en una mayor velocidad lineal y angular del robot mientras se desplaza. A pesar de que uno de los beneficios de una alta ganancia es la disminución del tiempo de ejecución, se determinó que una ganancia mayor a $K \geq 5$ resulta en movimientos exagerados, que en algunos casos resultan en colisiones innecesarias con los obstáculos del entorno o con robots móviles adyacentes.

Por el otro lado, una matriz de ganancia con valores inferiores resulta en movimientos menos bruscos y tiempos de ejecución de ruta mayores por la menor velocidad del robot. Una de las ventajas de utilizar ganancias bajas es que se minimizan desplazamientos no óptimos para completar el trayecto entre nodos de la ruta.

La Figura 3.43 muestra las posiciones del robot a lo largo del seguimiento de ruta con distintos valores de ganancia. Es apreciable la diferencia entre los dos casos, donde la ganancia más alta resulta en una trayectoria que incurre en una colisión con uno de los obstáculos. Adicionalmente se aprecia que la ganancia de K=7 presenta un recorrido rápido, conectando cada nodo con trayectos menos directos.

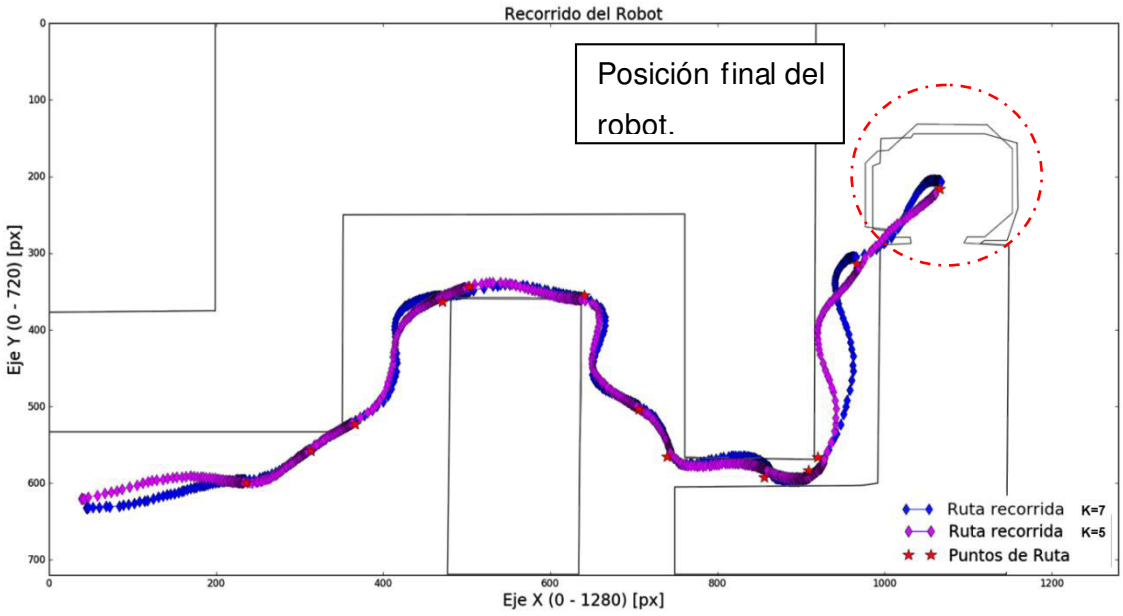



**Figura 3.43** Recorrido del robot con distintas matrices de ganancia.

La Figura 3.44 compara el error estimado en los dos casos utilizando distintas ganancias. Al utilizar una ganancia alta, la velocidad lineal y angular del robot aumentan, resultando en una señal de error que disminuye de manera más rápida, completando la ruta correspondiente en un menor número de iteraciones de control. Si bien el controlador logra en todos los casos llevar al robot hasta cada nodo de su ruta,

dada la alta ganancia del controlador, los trayectos recorridos no son siempre los más adecuados.

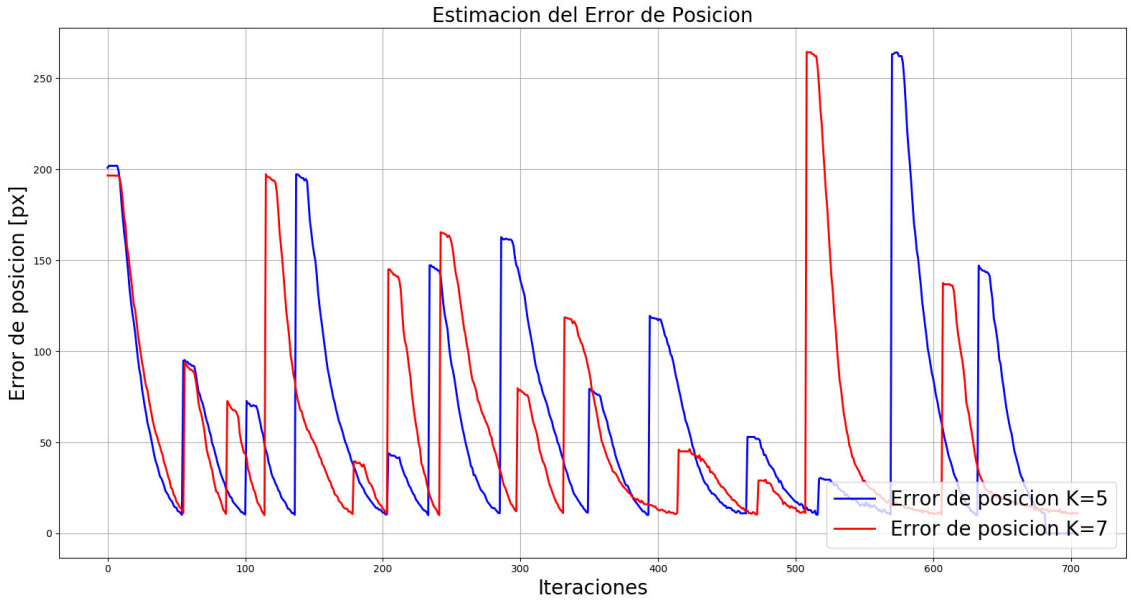



**Figura 3.44** Error de posición con distintos valores de ganancia.

La Figura 3.45 muestra las señales de salida del controlador respecto a la rapidez lineal requerida por el controlador. Se puede observar que con la ganancia más alta, la velocidad lineal llega cerca de los 30 [mm/s], mientras que con un K=5, el valor máximo es de cerca de 20 [mm/s]. De esta manera, valores altos de rapidez lineal resultan en movimientos iniciales del robot que describen curvas pronunciadas ente los nodos de la ruta, que son justamente trayectos no eficientes.

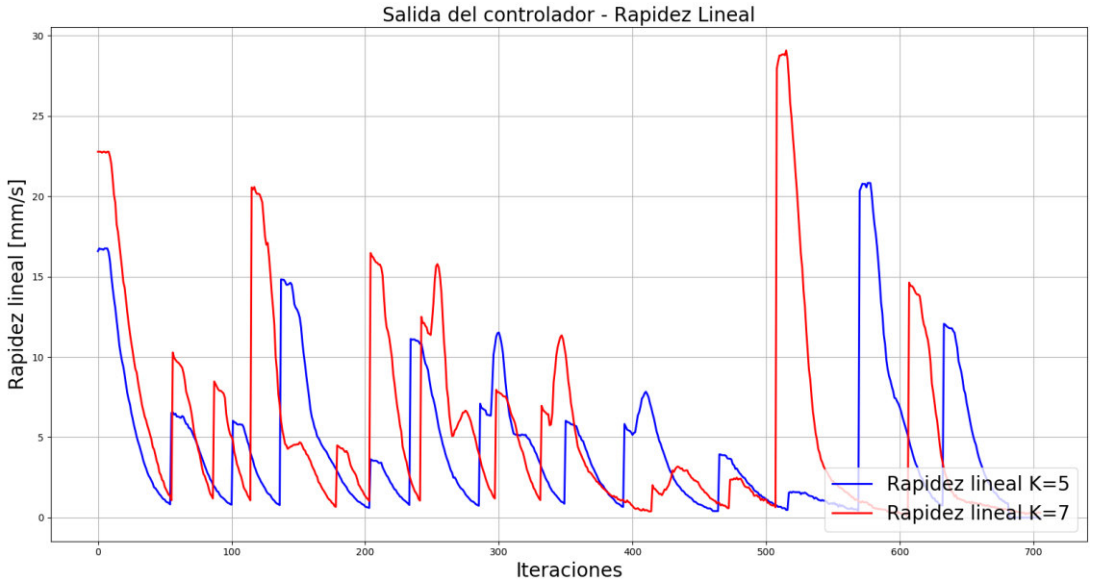



**Figura 3.45** Salida del controlador – Rapidez lineal con distintos valores de ganancia.

La Figura 3.46 ilustra la diferencia entre los dos casos, mostrando valores muy altos de rapidez angular cuando se utiliza una matriz de ganancia muy alta. se aprecia que cuando K=7, la rapidez angular supera los 1.5[rad/s], mientras que con K=5 el máximo valor es cercano a 1[rad/s]. Los mayores picos en la gráfica de la rapidez angular representan movimientos más bruscos del robot móvil, los mismos que no son usualmente deseables puesto que la ruta recorrida por el robot se hace más errática.

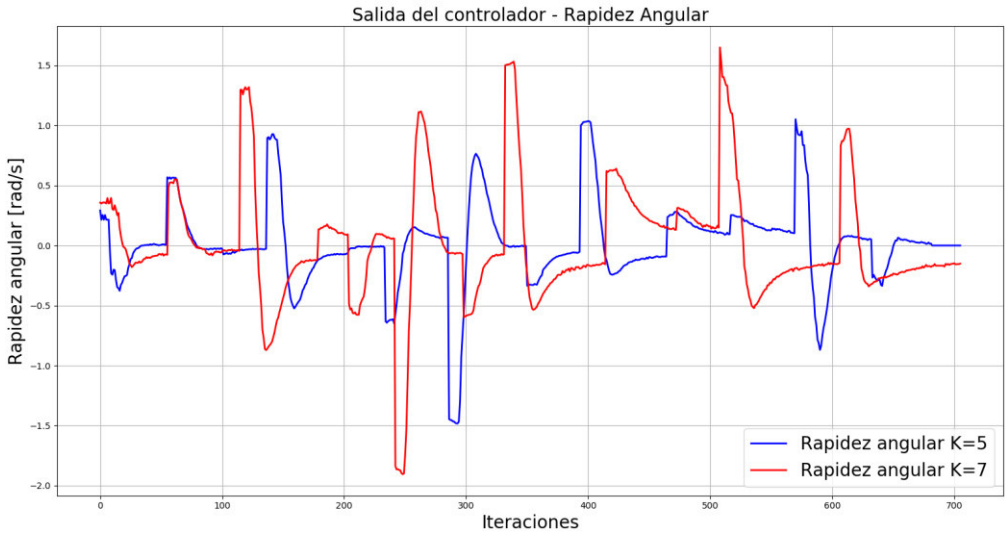



**Figura 3.46** Salida del controlador - Rapidez Angular con distintos valores de ganancia.

### 3.6.2 Prevención de sobre accionamiento en trayectos extensos

Como se pudo observar en los resultados anteriores, ya que el Control de Cinemática Inversa actúa sobre el error de posición; en trayectos extensos se puede observar un sobre accionamiento del controlador dado el gran error que existe.

Para evitar la situación descrita, se ha implementado un método que reduce el error en trayectos extensos dividiéndolos en secciones de menor distancia. De esta manera, el control se realiza de manera mucho más precisa y se evitan potenciales colisiones con los obstáculos durante la ejecución del control de movimiento.

Una comparación de una ruta con y sin el seccionamiento mencionado se observa en la Figura 3.47.

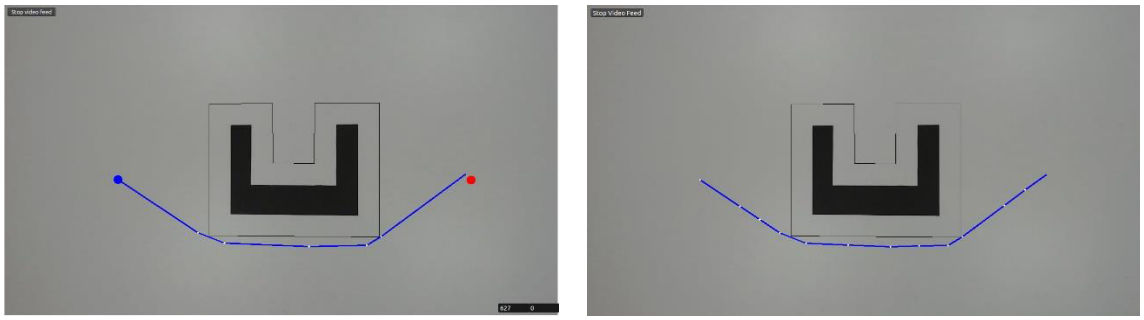


**Figura 3.47** Comparación entre la ruta original y la ruta modificada con puntos intermedios.

La Figura 3.48 muestra en detalle los puntos intermedios generados e incluidos en la ruta original para ser seguidos por el robot.

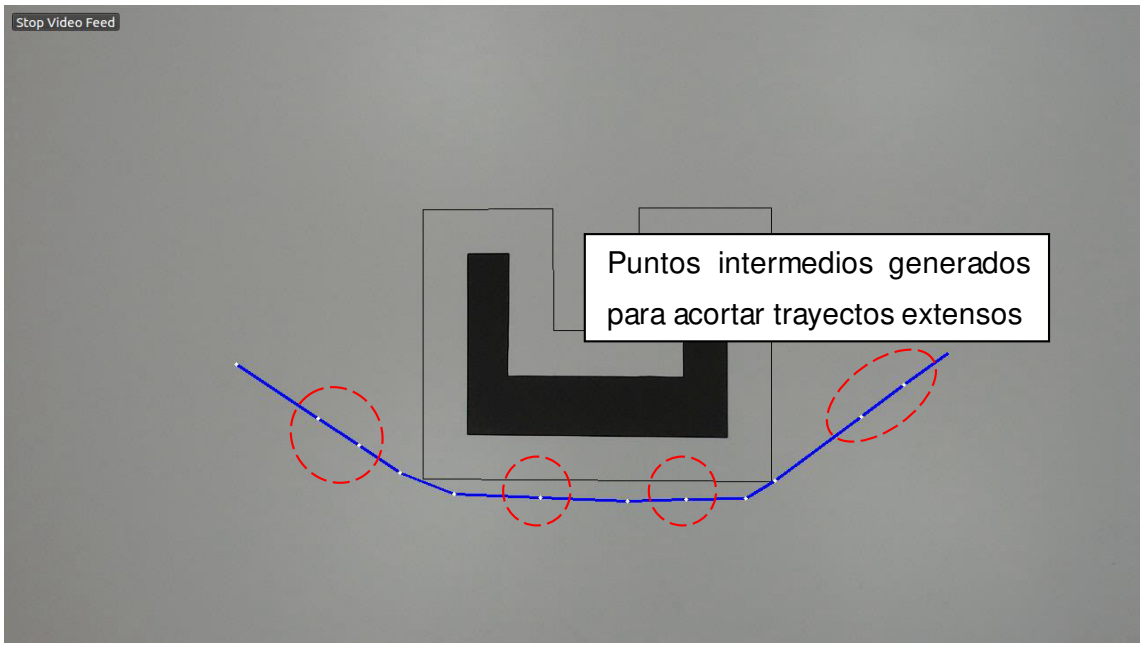



**Figura 3.48** Detalle de los puntos intermedios en la ruta generada.

Tras seguir la ruta modificada, se puede apreciar la diferencia en la ejecución del controlador. La Figura 3.49 muestra una comparación entre la ejecución de la ruta original y la ruta modificada con puntos intermedios.

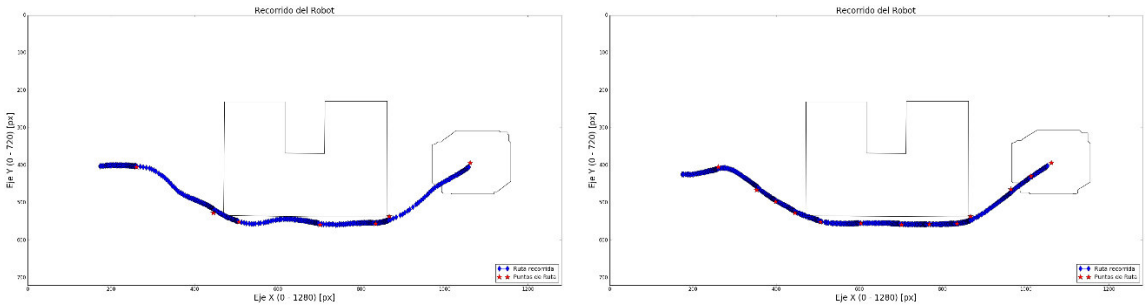



**Figura 3.49** Comparación entre seguimiento de ruta original y ruta modificada.

Como se observa, en el caso de la ruta modificada, el seguimiento de los puntos es mucho más preciso, con desviaciones mucho menores que en el caso del seguimiento de la ruta original.

## 3.7 ANÁLISIS DE ÍNDICES DE RENDIMIENTO

Para comparar ambos controladores es de interés conocer los índices de rendimiento de ambos, respecto al esfuerzo del controlador y a la señal de error, se realiza una comparación de los índices TVu e ISE.

Es importante mencionar que los resultados de los índices TVu e ISE se ven afectados por el tiempo requerido para completar la ruta, puesto que cuando se requiere mayor tiempo para completar la tarea, se acumula un mayor número de datos, esto acorde al mayor número de iteraciones de control necesarias. Por esta razón, la comparación de índices se ha realizado para las pruebas de seguimiento de ruta con el control Proporcional (Kp=0.13) y con el Control de Cinemática Inversa (K=5). En ambos casos se realizó el seguimiento de la ruta mostrada en la Figura 3.26 y el tiempo requerido para completar la tarea fue de alrededor de 1:15 min.

Los controladores Proporcional y CCI arrojaron los resultados observados en la Figura 3.50 respecto al índice TVu. Cabe especificar que la salida directa de ambos controladores no puede ser directamente comparada puesto que el control PID actúa sobre el error de orientación mientras que el CCI actúa respecto al error de posición. Para poder realizar una comparación justa entre ambos controladores, se ha calculado el índice TVu respecto al esfuerzo final de los actuadores izquierdo y derecho para uno de los robots.

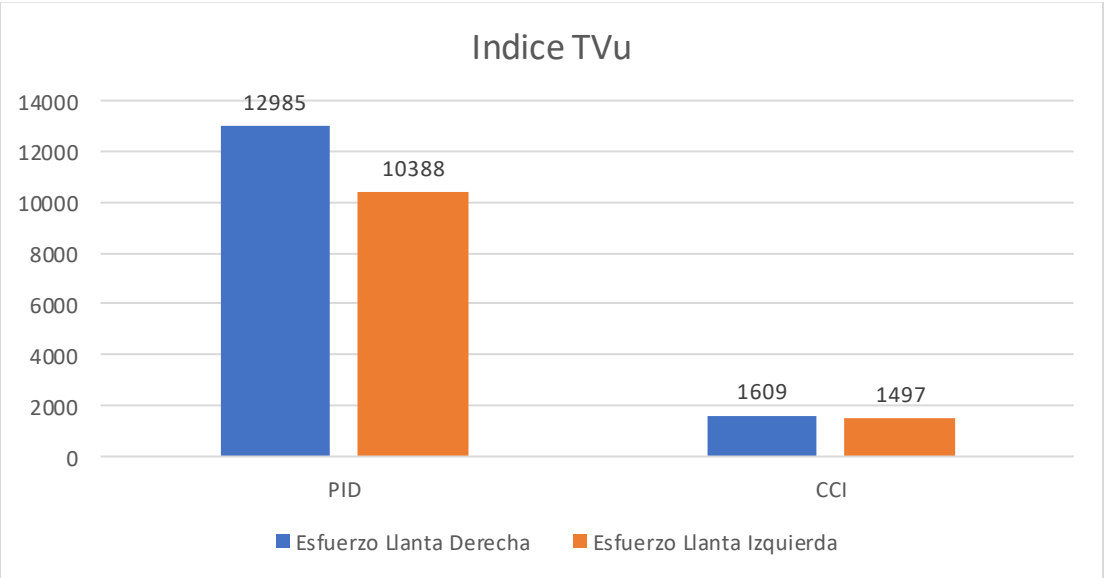
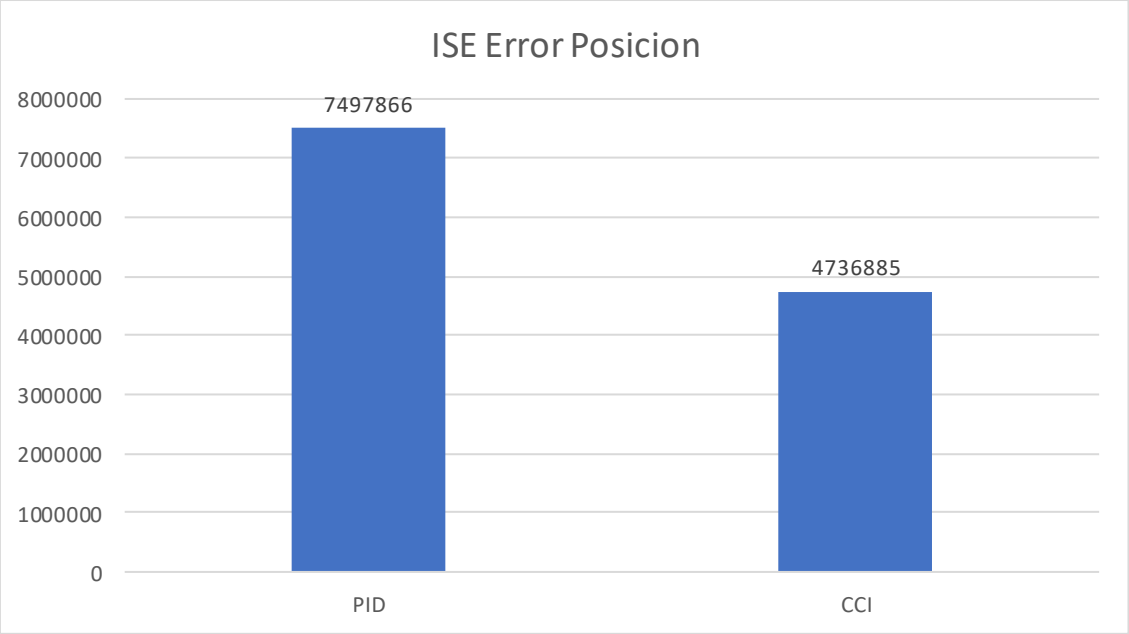



**Figura 3.50** Comparación de índices TVu de ambos controladores.

Se puede apreciar que el esfuerzo realizado por el controlador PID es mucho mayor ya que la rapidez lineal se mantiene constante durante los trayectos de desplazamiento del robot, mientras que con el CCI la rapidez lineal decrece de acuerdo con su cercanía de cada punto de la ruta. El resultado de esta diferencia es que el CCI utilizará menos energía para completar la ruta correspondiente.

La Figura 3.51 muestra una comparación del ISE entre ambos controladores. Respecto a este índice, se requiere precisar que, en el caso del control proporcional, no ha sido calculado respecto al error de orientación, puesto que, si bien el ángulo del robot es la variable controlada, es la posición de este la que finalmente se modifica para completar la ruta asignada. Por lo tanto, con el propósito de realizar una comparación equitativa entre ambos controladores, el ISE fue calculado con los datos de error de posición en ambos casos.




(b) Índice ISE de ambos controladores

**Figura 3.51** Comparación de ISE entre ambos controladores.

Como se puede observar en la Figura 3.51, el control Proporcional arroja un ISE mucho menor en comparación al ISE del CCI. Ya que el Control de Cinemática Inversa actúa directamente sobre el error de posición, reduce dicho error agresivamente cuando el robot se encuentra alejado de la coordenada objetivo.

## 3.8 Prueba de funcionamiento con 4 robots

Para demostrar la funcionalidad del sistema y la capacidad de este de ejecutar el control de manera simultánea se ha realizado también una prueba de seguimiento de rutas con los 4 robots en el área de trabajo. Las rutas calculadas se pueden apreciar

en la Figura 3.52, donde se muestra el entorno con distintos obstáculos y una ruta para cada robot.

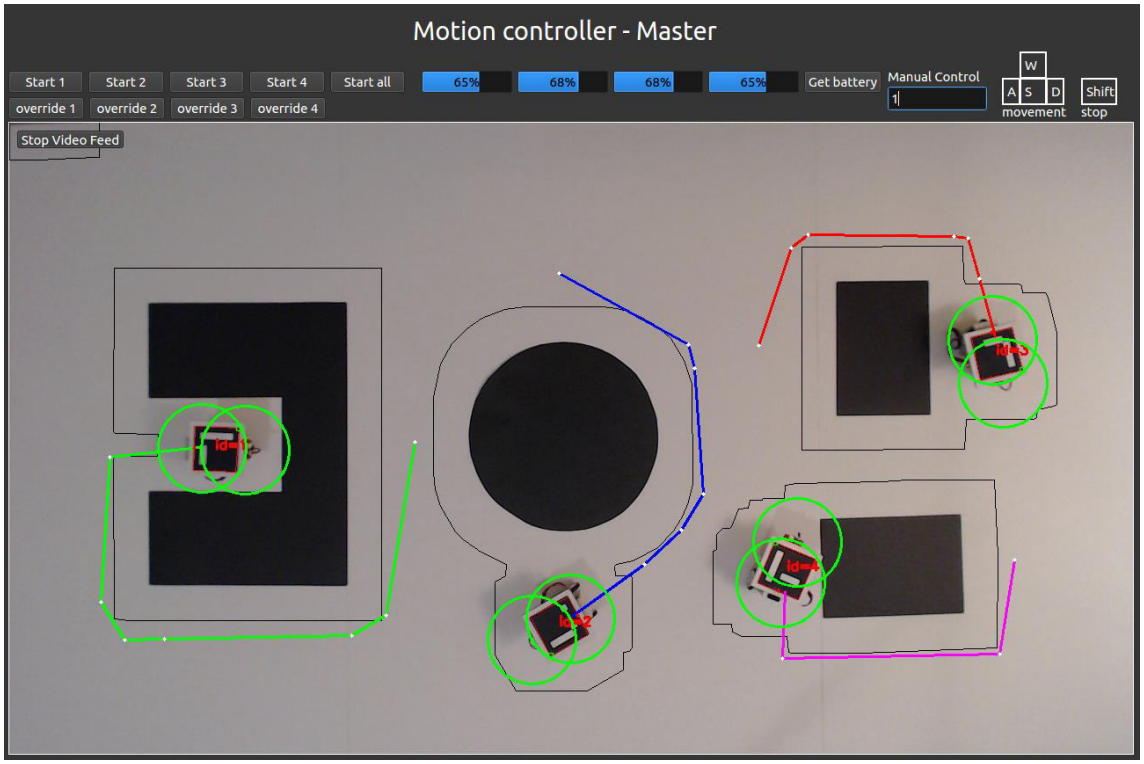



**Figura 3.52** Rutas en prueba se seguimiento con 4 robots.

La Figura 3.53 muestra las rutas recorridas por cada robot luego de la ejecucion del control simultaneo de los 4 robots. Se puede apreciar que para todos los casos se obtuvo un seguimiento correcto de las rutas. Un video de la presente prueba se puede encontrar en el Anexo C.

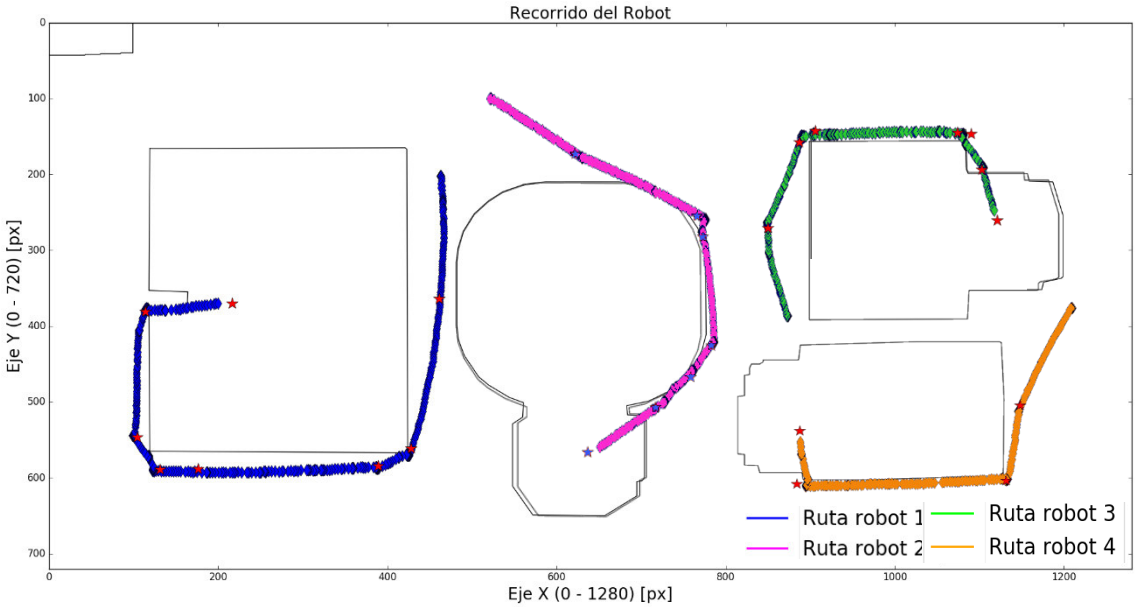
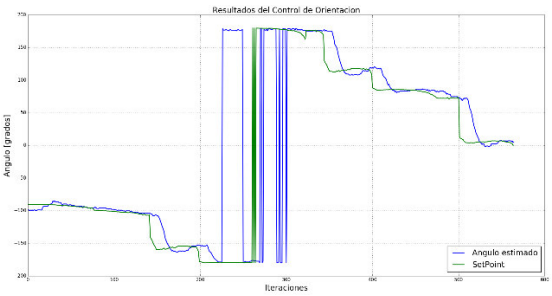
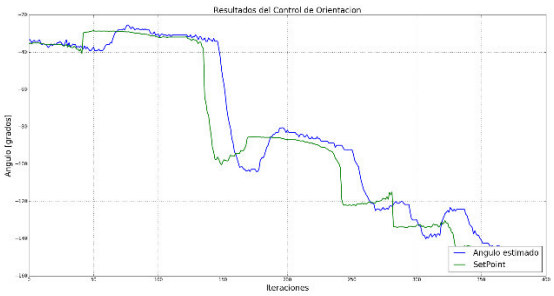



**Figura 3.53** Rutas recorridas por los 4 Robots

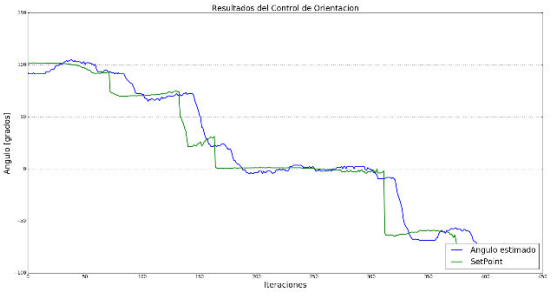
Finalmente, la Figura 3.54 presenta los resultados del control de orientación respecto al setpoint para cada robot, donde se puede apreciar el desempeño del control en cada caso.



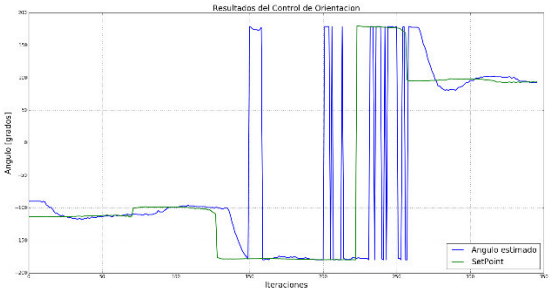

(a) Resultado del control de orientación – Robot 1




(b) Resultado del control de orientación – Robot 2




(c) Resultado del control de orientación – Robot 3




(d) Resultado del control de orientación – Robot 4

**Figura 3.54** Resultados del control de orientación para los 4 robots.

En las cuatro figuras presentadas se observa que el controlador sigue de manera adecuada al setpoint de orientación. Cabe mencionar que las altas variaciones del

ángulo de orientación observado en las figuras a y d, se deben al método de estimación del ángulo, puesto que este se encuentra en un rango de -180 grados a +180 grados. Cuando la orientación de uno de los robots cruza ese límite, se producen las variaciones observadas en las graficas mencionadas.

Figuras complementarias a todas las pruebas encontradas en el presente capitulo, se pueden encontrar en el Anexo B.

# 4 CONCLUSIONES

- Con base en la revisión bibliográfica realizada acerca de los métodos de mapeo y planificación de rutas para aplicaciones robóticas, se determinó que el algoritmo RRT* era la mejor opción para implementar un sistema de seguimiento de rutas de manera exitosa. Dadas las extensas posibilidades de mejoramiento, optimización y experimentación, RRT* fue seleccionado como una excelente manera de permitir la expansión continua del sistema a partir de los avances realizados en el presente trabajo.
- Tras la identificación de los requerimientos del sistema para incluir las nuevas funcionalidades necesarias, se modificó el sistema de visión artificial previamente utilizado. La principal adición fue el reconocimiento de obstáculos en el área de trabajo, lo que permitió reconocer diferentes figuras ubicadas sobre el plano de trabajo.
- Adicionalmente, se incrementó la frecuencia de publicación de mensajes ROS mediante el aumento de la tasa de adquisición de imágenes, así como el aumento de resolución útil para el sistema. Estos cambios permiten realizar la adquisición de datos de manera más confiable y realizar el control a una mayor velocidad.
- Como parte de la expansión continua del sistema, se incrementó un robot móvil con las mismas características que los tres ya existentes. Esta adición es parte del objetivo de contar con un mayor número de agentes para hacer real la posibilidad futura de implementar aplicaciones más complejas en la plataforma.
- Otro cambio de hardware fue la migración del sistema de comunicación a una nueva versión de los módulos Xbee. Si bien la versión utilizada en la iteración anterior del proyecto fue suficiente para cumplir con los objetivos, la nueva versión Xbee S2C abre las posibilidades de implementar y experimentar con esquemas más complejos de comunicación, así como de incluir distintos escenarios que se aproximen a la realidad.

- Como parte de la arquitectura de diseño de los nodos ROS, se utilizaron programas de ejecución simultánea, permitiendo una fácil expansión del sistema sin necesidad de escribir nodos ROS adicionales para el control de los robots que se agreguen. Mediante la ejecución de nodos ROS de manera paralela se facilita el control y escritura de código sin importar el número de robots presentes en la plataforma.
- Como parte de la implementación de la funcionalidad de planificación y seguimiento de rutas, fue necesaria la inclusión de reglas que prevengan las colisiones entre robots. Para esto se escribió un nodo ROS que monitoree la posición y distancia entre robots, permitiendo así detenerlos en caso de ser necesario para evitar una colisión.
- Como parte del desarrollo de la aplicación, se crearon dos interfaces gráficas de usuario para permitir el control y monitoreo del sistema. La interfaz de usuario correspondiente a la planificación de rutas permite visualizar el espacio de trabajo y seleccionar los puntos de partida y final para cada robot. Una vez calculada la ruta mediante el algoritmo RRT*, se dibuja la misma sobre el espacio de trabajo. La interfaz de control de movimiento permite visualizar el área de trabajo y comandar los robots para iniciar el seguimiento de rutas. Adicionalmente habilita un control manual para cada robot en caso de requerir posicionarlos previamente.
- Para el seguimiento de las rutas calculadas, se emplearon y compararon dos controladores. El control PID de orientación es de sintonización flexible y permite cambiar las características en caso de ser requerido a través de la configuración de las ganancias. El Control de Cinemática Inversa actúa para reducir la distancia entre el robot y las coordenadas de su ruta, permitiendo alterar su desempeño a través de la matriz de ganancia.
- Las pruebas realizadas muestran el desempeño del sistema mediante el uso de los dos controladores. Cada uno presenta características particulares, lo cual los hace útiles dependiendo de las condiciones deseadas de operación. Adicionalmente, una comparación cualitativa y cuantitativa entre los dos controladores permite conocer su desempeño durante la operación del sistema.
- La latencia del sistema fue determinada a través de la medición del tiempo que toma la ejecución del lazo de control, el cual esta principalmente influenciado por los tiempos requeridos para la comunicación entre el módulo Xbee del computador y los módulos de cada robot. Adicionalmente, se determinó la latencia del sistema ROS respecto a la adquisición y procesamiento de

imágenes. Dicha latencia se encuentra condicionada por la velocidad de captura de la capara utilizada, por lo que la latencia final fue menor a 30Hz.

## 4.1 RECOMENDACIONES

- Para asegurar el correcto funcionamiento de la red de comunicaciones inalámbricas, es recomendable establecer tiempos de espera adecuados que permitan la compleción de envío y recepción de datos entre los módulos Xbee.
- En caso de requerir ampliar el número de robots móviles, se recomienda utilizar el formato de ejecución paralela de nodos ROS, el cual permite utilizar un mismo programa para el control de cualquier número de robots.
- De ser necesario probar distintos controladores para el control de movimiento de los robots, se recomienda modificar el nodo ROS reemplazando únicamente el código del lazo de control. Así se asegura que el sistema siga funcionando sin necesidad de modificaciones extensas.
- Dado que la adquisición de datos se realiza a través de una cámara web, y que la distinción de obstáculos e identificación de marcadores ArUco se realiza mediante OpenCV, se recomienda utiliza las mejores condiciones lumínicas al utilizar el sistema. De la misma manera, se recomienda utilizar obstáculos que contrasten de manera clara con el fondo del área de trabajo.
- En caso de requerir reposicionar los robots durante el uso de la plataforma, se recomienda utilizar el control manual de los mismos para evitar interrumpir la ejecución del experimento o interferir con la configuración del área de trabajo.

# 5 REFERENCIAS BIBLIOGRÁFICAS


[1] G. Jácome, M. Sierra and P. J. Cruz, "A Mini-sized Agent Testbed for Applications in Mobile Robotics," 2019 IEEE 4th Colombian Conference on Automatic Control (CCAC), 2019, pp. 1-6, doi: 10.1109/CCAC.2019.8921109.

[2] L. Arcos, C. Calala, D. Maldonado, and P. J. Cruz, “ROS based Experimental Testbed for Multi-Robot Formation Control,” presented at the 2020 IEEE ANDESCON, oct. 2020.

[3] J. Yu, S. D. Han, W. N. Tang, and D. Rus, “A portable, 3D-printing enabled multi-vehicle platform for robotics research and education,” presented at the 2017 IEEE International Conference on Robotics and Automation (ICRA), May 2017.

[4] D. Pickem et al., “The Robotarium: A remotely accessible swarm robotics research testbed,” presented at the 2017 IEEE International Conference on Robotics and Automation (ICRA), May 2017.

[5] P. Hart, N. Nilsson, and B. Raphael, “A Formal Basis for the Heuristic Determination of Minimum Cost Paths,” IEEE Trans. Syst. Sci. Cyber., vol. 4, no. 2, pp. 100–107, 1968

[6] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” IEEE Trans. Robot. Automat., vol. 12, no. 4, pp. 566–580, 1996

[7] L. Janson, E. Schmerling, A. Clark, and M. Pavone, “Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions,” The International Journal of Robotics Research, vol. 34, no. 7, pp. 883–921, May 2015

[8] LaValle, S. M., ``Rapidly-Exploring Random Trees: A New Tool for Path Planning'' Report No. TR 98-11, Computer Science Department, Iowa State University, 1998.

[9] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” The International Journal of Robotics Research, vol. 30, no. 7, pp. 846–894, jun. 2011.

[10] J. Nasir et al., “RRT*-SMART: A Rapid Convergence Implementation of RRT*,” International Journal of Advanced Robotic Systems, vol. 10, no. 7, p. 299, Jan. 2013.

[11] Noreen, I., Khan, A., and Habib, Z. . 'A comparison of rrt, rrt* and rrt*-smart path planning algorithms' International Journal of Computer Science and

Network Security, vol 16, no. 10, oct. 2016.

[12] I.-B. Jeong, S.-J. Lee, and J.-H. Kim, “RRT*-Quick: A Motion Planning Algorithm with Faster Convergence Rate,” in Advances in Intelligent Systems and Computing, Springer International Publishing, 2015

[13] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, “Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic,” presented at the 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014), Sep. 2014.

[14] D. Hutabarat, M. Rivai, D. Purwanto, and H. Hutomo, “Lidar-based Obstacle Avoidance for the Autonomous Mobile Robot,” presented at the 2019 12th International Conference on Information and Communication Technology and System (ICTS), jul. 2019.

[15] Md Mahbubur Rahman, Leonardo Bobadilla, and Brian Rapp. ‘Sampling-based planning algorithms for multi-objective missions’ Automation Science and Engineering (CASE), 2016 IEEE International Conference on. IEEE, 2016., Motion Planning Algorithm RRT star( RRT* ) Python Code Implementation, with Obstacle.” Unpublished, 2018.

[16] J. Aguilar, A. Ríos, F. Hidrobo, and M. Cerrada, “Sistemas multiagentes y sus aplicaciones en automatización industrial,” 01 2013.

[17] J. Ota, “Multi-agent robot systems as distributed autonomous systems,” Advanced Engineering Informatics, vol. 20, pp. 59–70, 01 2006.

[18] J.Albus, “Control architecture for cooperative intelligent robots.” Proceedings Robots Biological Systems Session,Tuscany,IT,1989-06-301989.[Online].Available:https://tsapps.nist.gov/publication/getpdf.cfm?pubid= 820141

[19] R. C. Arkin, “Cooperation without communication: Multiagent schema-based robot navigation,” vol. 9, no. 3.Journal of Robotic Systems,1992.

[20] S. Buettrich and A. E. Pascual, “Topología e infraestructura estructura básica de redes inalámbricas.” TRICALCAR, 2007.

[21] Documentation - ROS Wiki", Wiki.ros.org, 2021. [Online]. Available: http://wiki.ros.org/Documentation. [Accessed: 16- Jul- 2021]

[22] C. A. Rosen and N. J. Nilsson, “Application of intelligent automata to reconnaissance, technical report,” vol. Project 5953 Interim Report 1.Stanford Research Institute, 1966

[23] X. Liu and D. Gong, “A comparative study of a-star algorithms for search and rescue in perfect maze,” 04 2011

[24] E. W. Dijkstra, «A note on two problems in connexion with graphs», Numer. Math., vol. 1, n.º 1, pp. 269-271, dic. 1959, doi: 10.1007/bf01386390.

[25] David Filliat, Jean-Arcady Meyer. Map-based navigation in mobile robots - I. A review of localisation strategies. Journal of Cognitive Systems Research, 2003, 4 (4), pp.243–282. ff10.1016/S1389- 0417(03)00008-1ff. ffhal-00655473f

[26] Perera, D. Barnes, and D. Zelinsky, Exploration: Simultaneous Localization and Mapping (SLAM).Boston, MA: Springer US, 2014,pp. 268–275. [Online]. Available: https://doi.org/10.1007/978-0-387-31439-6280

[27] B. Alhafni, S. F. Guedes, L. C. Ribeiro, J. Park, and J. Lee, "Mapping areas using computer vision algorithms and drones,"ArXiv, vol.abs/1901.00211, 2019

[28] "About - OpenCV", OpenCV, 2021. [Online]. Available: https://opencv.org/about/. [Accessed: 16- Jul- 2021]

[29] A. R. Carter, G. M. King, T. A. Ulrich, W. Halsey, D. Alchenberger, y T. T. Perkins, «Stabilization of an optical microscope to 01 nm in three dimensions», Appl. Opt., vol. 46, n.º 3, p. 421, ene. 2007, doi: 10.1364/ao.46.000421.

[30] S. Garrido-Jurado, R. Mu˜noz-Salinas, F. Madrid-Cuevas, and M. Mar´ın-Jim´enez, "Automatic generation and detection of highly reliable fiducialmarkers under occlusion,"Pattern Recognition, vol. 47, p. 2280–2292,06 2014

[31] H. Miao and Y. Tian, "Robot path planning in dynamic environments using a simulated annealing based approach," 2008 10th International Conference on Control, Automation, Robotics and Vision, 2008, pp. 1253-1258, doi: 10.1109/ICARCV.2008.4795701

[32] E. Masehian and D. Sedighizadeh, "Multi-objective robot motion plan-ning using a particle swarm optimization model,"Journal of ZhejiangUniversity - Science C, vol. 11, pp. 607–619, 08 2010

[33] "Qt Designer Manual", Doc.qt.io, 2021. [Online]. Available: https://doc.qt.io/qt-5/qtdesigner-manual.html. [Accessed: 16- Jul- 2021]

[34] "PyQt5", PyPI, 2021. [Online]. Available: https://pypi.org/project/PyQt5/. [Accessed: 16- Jul- 2021]

[35] "API mode in detail", Digi.com, 2021. [Online]. Available: https://www.digi.com/resources/documentation/Digidocs/90001942-13/concepts/c_api_mode_detailed.htm?TocPath=XBee%20API%20mode%7

C_____1. [Accessed: 16- Jul- 2021]

[36] "Usb_cam - ROS Wiki", Wiki.ros.org, 2021. [Online]. Available: http://wiki.ros.org/usb_cam. [Accessed: 17- Jul- 2021]

[37] "About — wiki", Pygame.org, 2021. [Online]. Available: https://www.pygame.org/wiki/about. [Accessed: 19- Jul- 2021]

[38] "ARUCO markers: basics — Scientific Python: a collection of science oriented python examples documentation", Mecaruco2.readthedocs.io, 2021. [Online]. Available: https://mecaruco2.readthedocs.io/en/latest/notebooks_rst/Aruco/aruco_basics.html. [Accessed: 19- Jul- 2021]

[39] C. Durmusoglu, "ivmech/ivPID", GitHub.com, 2021. [Online]. Available: https://github.com/ivmech/ivPID. [Accessed: 20- Jul- 2021]

[40] "Markings on Mars Perseverance Rover", Mars.nasa.gov, 2021. [Online]. Available: https://mars.nasa.gov/mars2020/spacecraft/rover/markings/. [Accessed: 14- Aug- 2021]

[41] "Digi XBee S2C 802.15.4 RF Modules Datasheet", Digi.com, 2021. [Online]. Available: https://www.digi.com/resources/library/data-sheets/ds_xbee-s2c-802-15-4. [Accessed: 18- Aug- 2021]

[42] "XCTU", Digi.com, 2021. [Online]. Available: https://www.digi.com/products/embedded-systems/digi-xbee/digi-xbee-tools/xctu. [Accessed: 18- Aug- 2021]

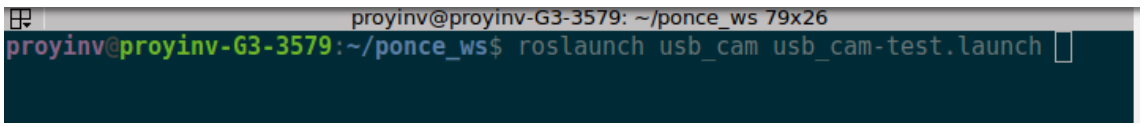
# ANEXOS

# ANEXO A

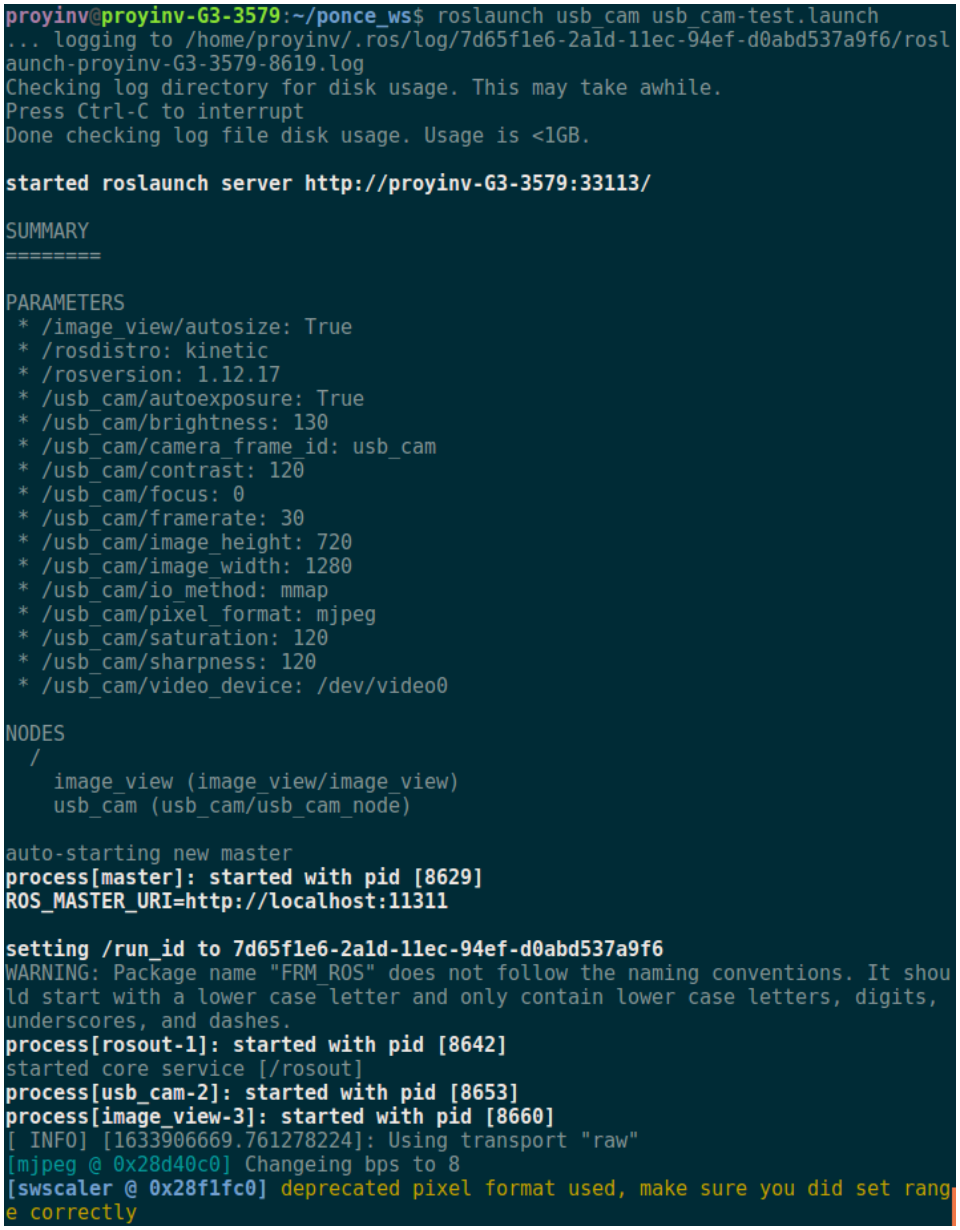
### Manual de Usuario Interfaz Gráfica de Planificación de Rutas

## INICIALIZACIÓN

1. Abrir un terminal y ejecutar la instrucción: `roslunch usb_cam usb_cam-test.launch`

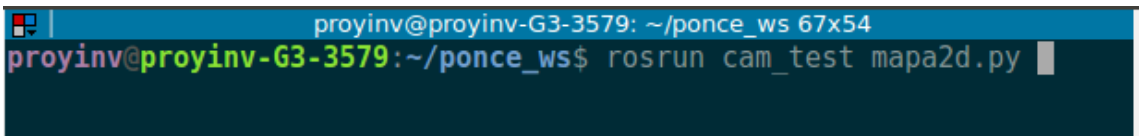



Se desplegará la información de los parámetros utilizados en el nodo ROS



2. Ejecutar el nodo ROS mapa2D para iniciar el análisis de imágenes en tiempo real.

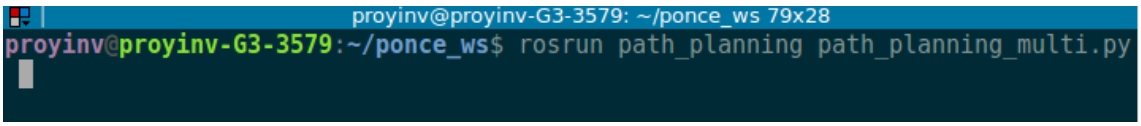
Abrir un terminal y ejecutar la instrucción : `rosrun cam_test mapa2d.py`




Esta función es trasparente durante la operación por lo que no se requiere más interacción a través del terminal.

3. Ejecutar el nodo ROS de Planificación de Rutas

Abrir un terminal y ejecutar la instrucción: `rosrun path_planning path_planning_multi.py`

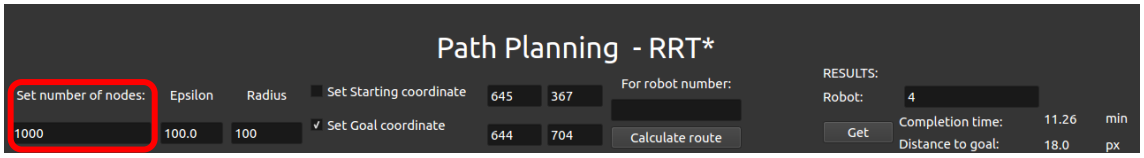



Se desplegará la interfaz de usuario:

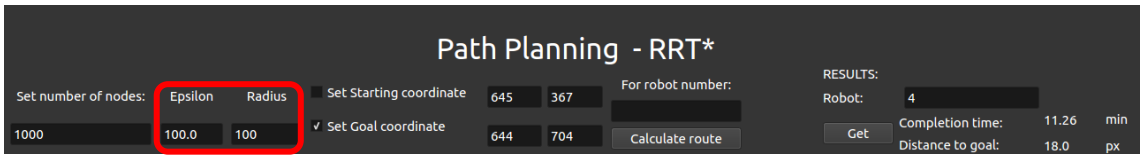
## CONFIGURACIÓN DE PARÁMETROS

Los parámetros que se pueden configurar son el número de nodos, épsilon y el radio de búsqueda para el algoritmo RRT*.

El número de nodos o iteraciones RRT* corresponde a un valor entero mayor a 1. El valor recomendado es de 1000 nodos, pero el usuario podrá modificar este valor antes de ejecutar el algoritmo.



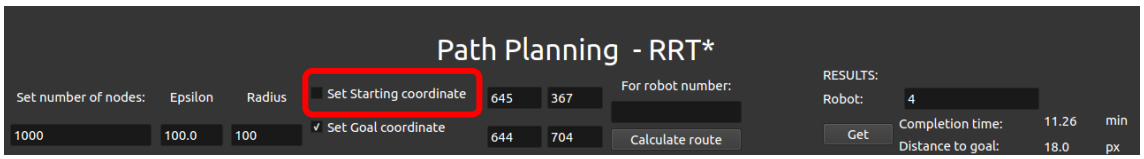

Los valores de épsilon y radio de búsqueda son valores que dependen del entorno que se requiera analizar, sin embargo, los valores recomendados son de 100 unidades para ambos parámetros.



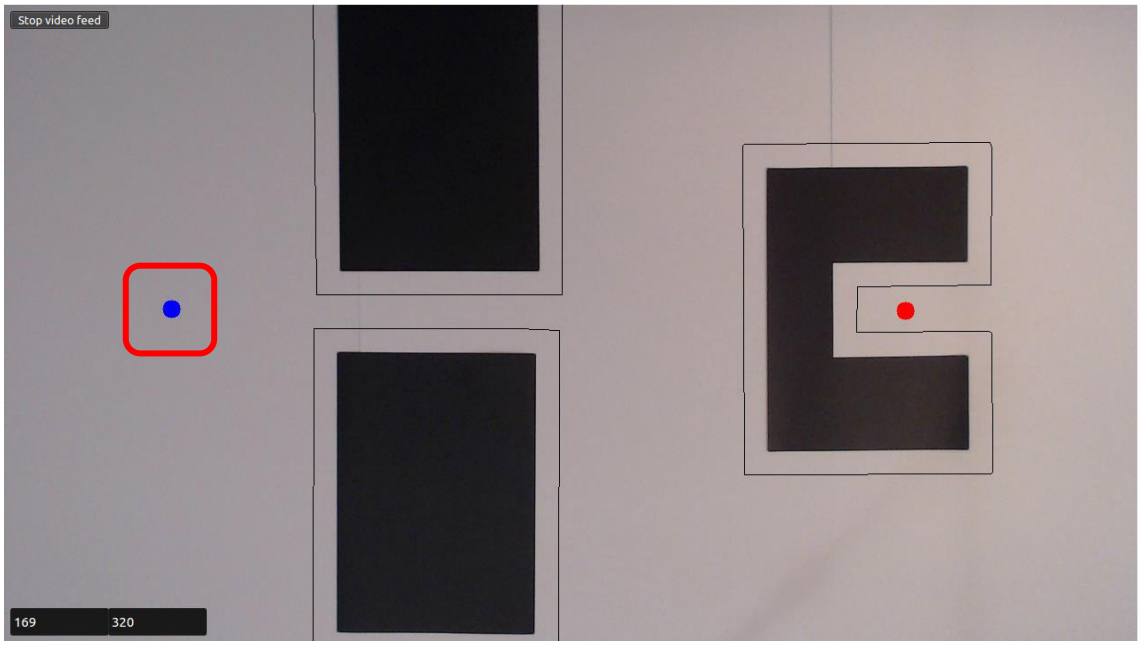

## SELECCIÓN DE COORDENADAS DE INTERÉS

Una vez configurados los parámetros anteriores, el usuario puede elegir con el cursor los puntos de inicio y final de la ruta a ser calculada.

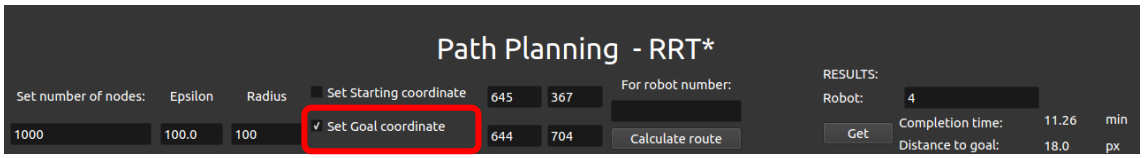
Selección del punto de inicio: Aplicar click en la caja `Set Starting Coordinate`.



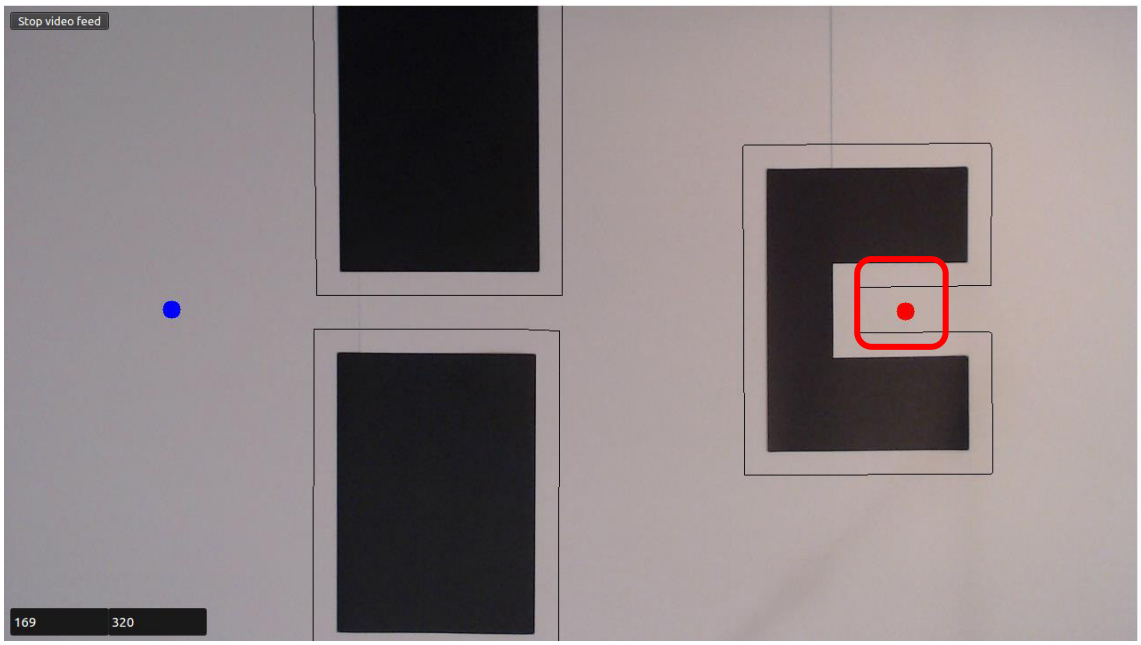

Una vez marcada la caja, seleccionar en el espacio de trabajo el punto o coordenada deseada. El punto de inicio se marcará con un círculo solido azul



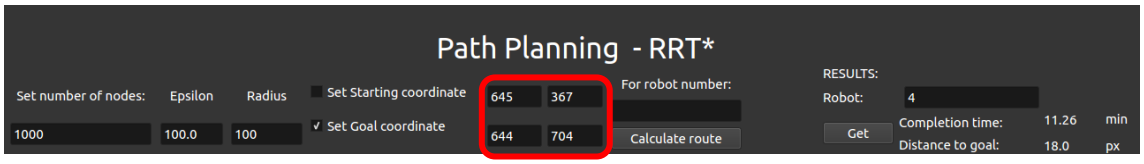

Selección del punto de inicio: Aplicar click en la caja `Set Goal Coordinate`.




Una vez marcada la caja, seleccionar en el espacio de trabajo el punto o coordenada deseada. El punto de inicio se marcará con un círculo solido rojo.

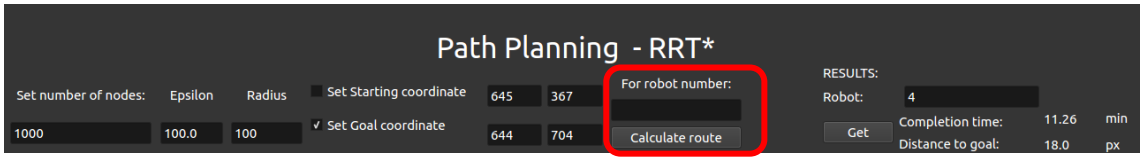



En ambos casos se mostrarán las coordenadas exactas seleccionadas, las mismas que pueden ser modificadas para tener mayor precisión en la selección.

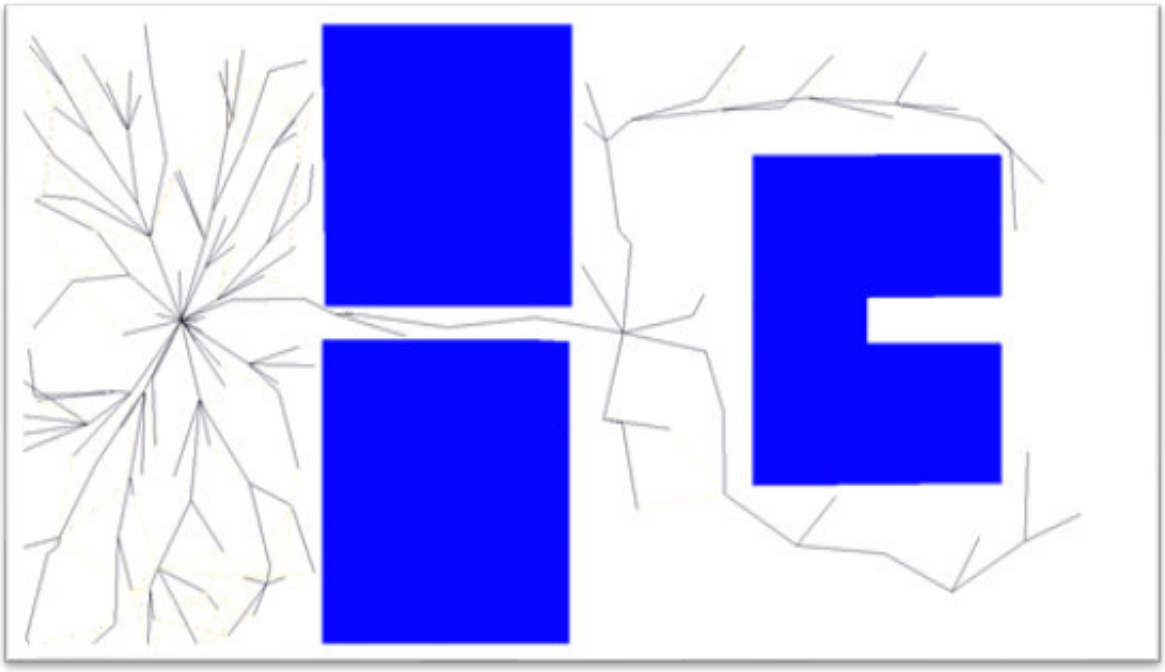



## EJECUCIÓN DEL ALGORITMO

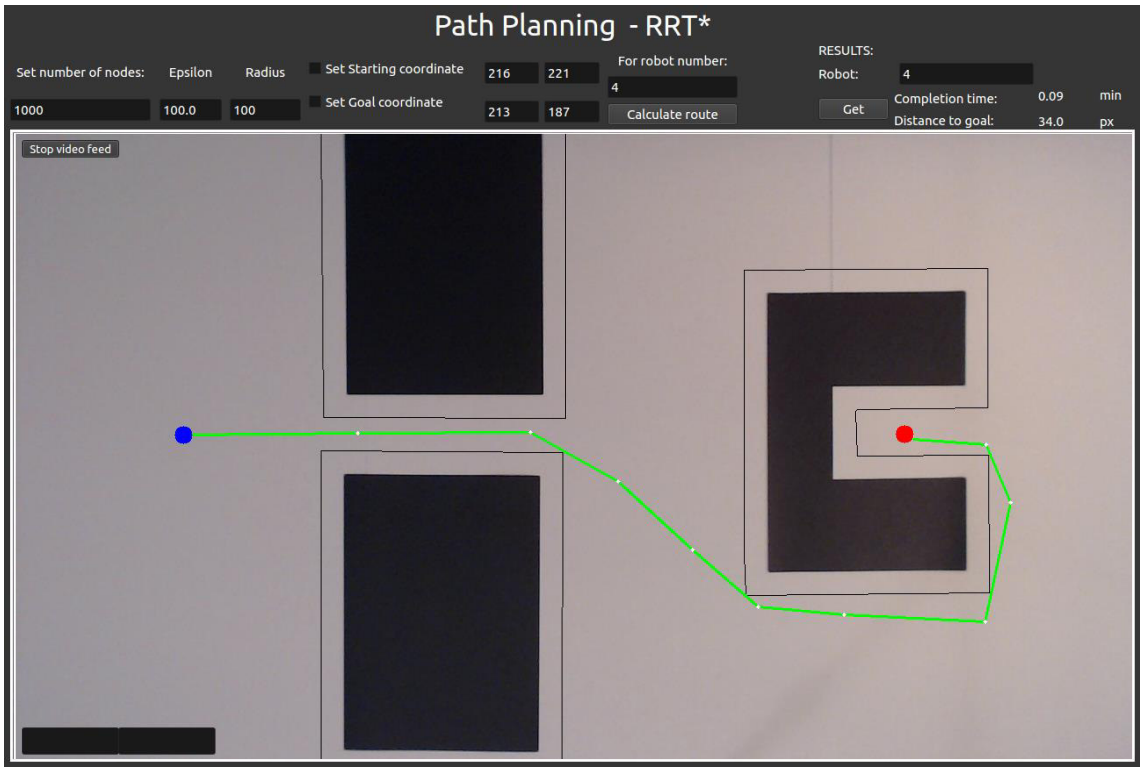
Para iniciar la ejecución del algoritmo, se requiere especificar a qué robot se asignará la ruta resultante. En el presente trabajo, el número máximo será el robot con ID=4, pero en futuras expansiones del proyecto, este número dependerá del número de robots disponibles.




Una vez ingresado el número de ID del robot, realizar click en `Calculate route`. Se desplegará una ventana nueva donde se podrá observar el proceso del algoritmo y su avance de acuerdo con el número de nodos seleccionados.

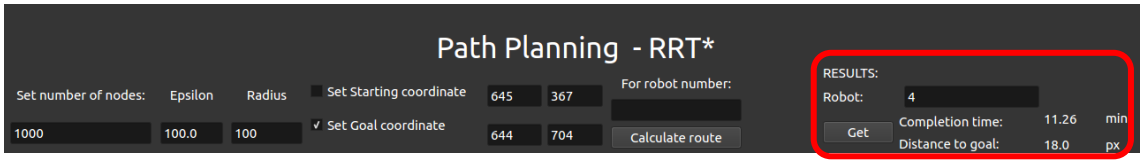


Una vez terminado el proceso de cálculo de la ruta, la ventana emergente desaparecerá y se dibujará la solución encontrada sobre la interfaz.




## OBTENCIÓN DE RESULTADOS

Para conocer las estadísticas de duración de ejecución y distancia al punto meta, se puede consultar a través del botón GET, especificando el número de ID del robot al cual fue asignada la ruta.

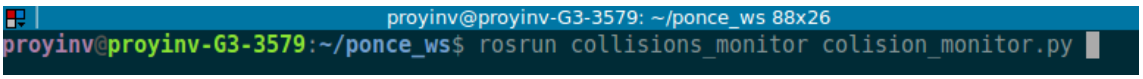


**Manual de Usuario Interfaz Gráfica Control de Movimiento**

## INICIALIZACIÓN

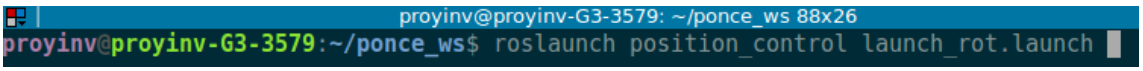
Antes iniciar la interfaz de usuario de Control de Movimiento, se requiere inicializar el nodo ROS de prevención de colisiones.

1. Ejecutar el nodo ROS: `rosrun collisions_monitor colisión_monitor.py`

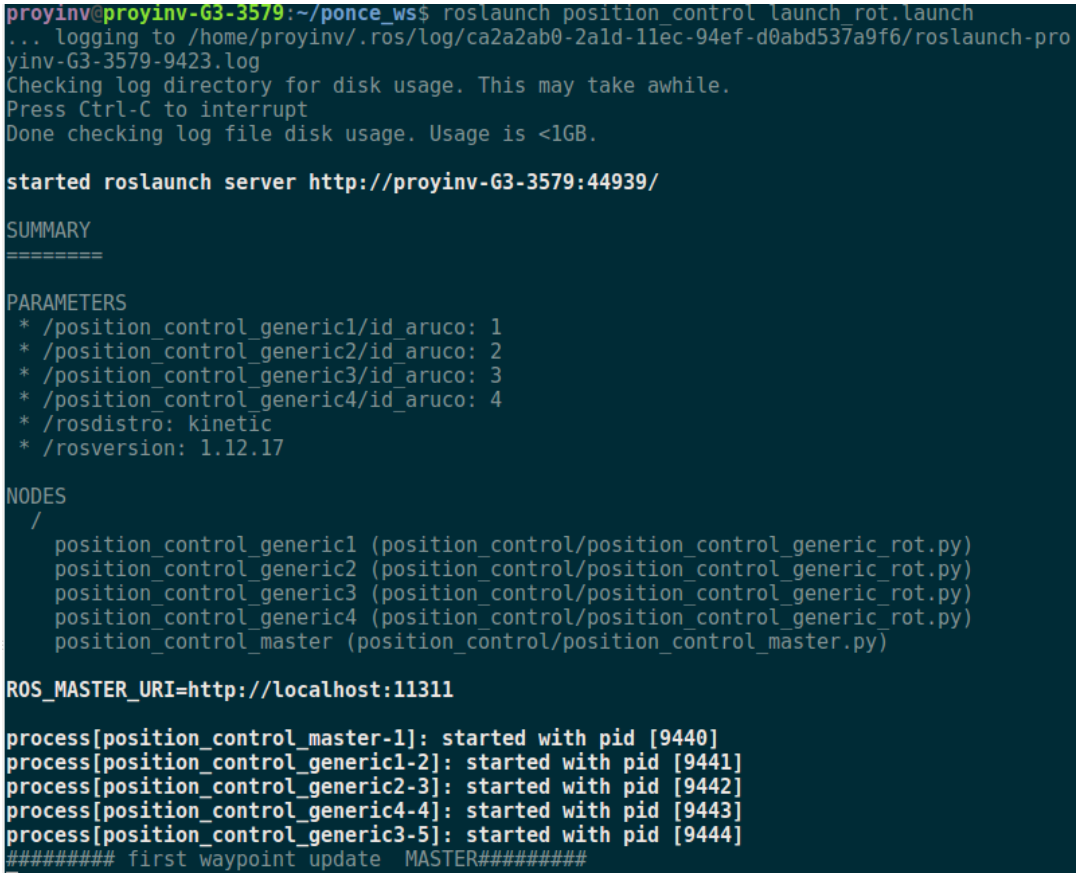



Esta función es trasparente durante la operación por lo que no se requiere más interacción a través del terminal.

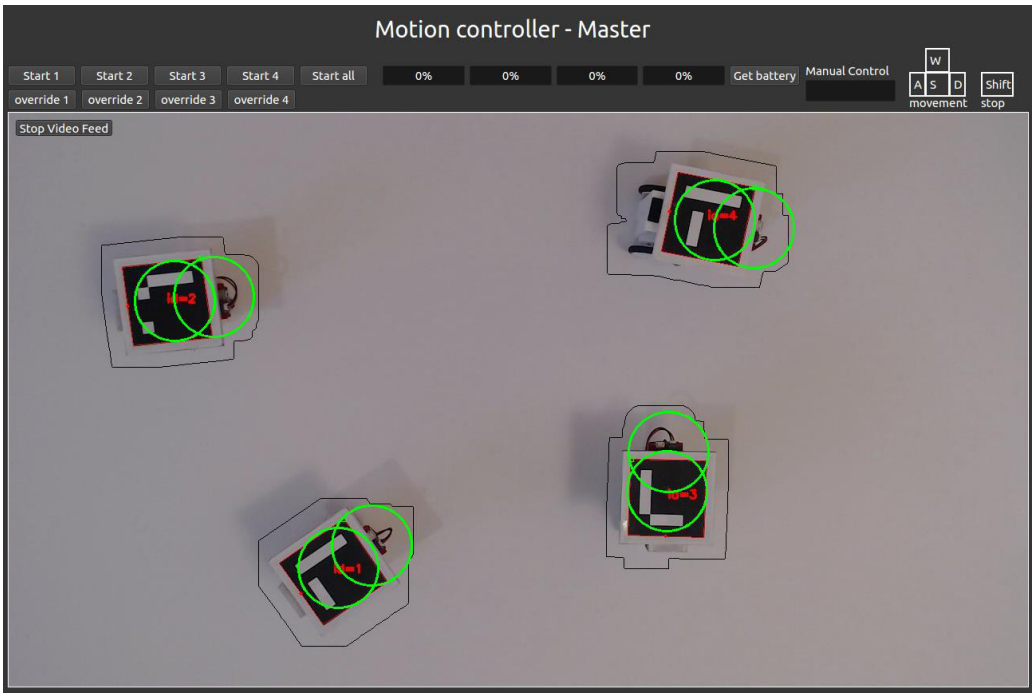
2. Para iniciar la interfaz de usuario, ejecutar el nodo ROS de control de movimiento: `roslaunch position_control launch_rot.launch.`




En el terminal se mostrará la información de los nodos a ejecutar.

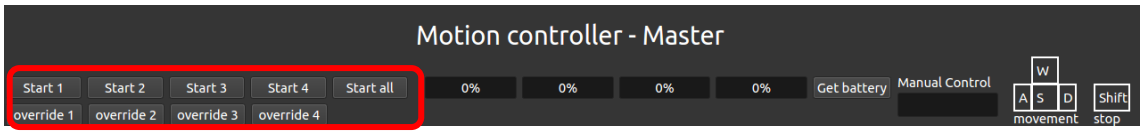



Se desplegará la interfaz de Control de Movimiento.

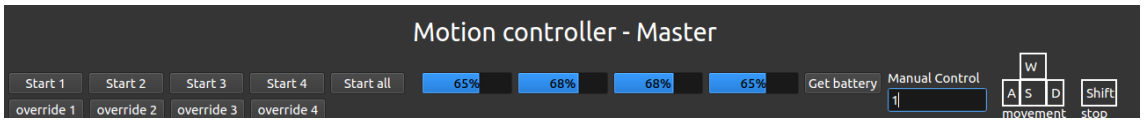



## EJECUCIÓN DEL CONTROL

La ejecución del control se inicia y detiene con los botones ubicados en la parte superior izquierda. Por existir disponibles 4 robots, existen solo 4 botones para su control individual. En caso de ser requerido, los bonotes de OVERRIDE están disponibles para resumir el movimiento de los robots cuando se hayan detenido para evitar colisiones.

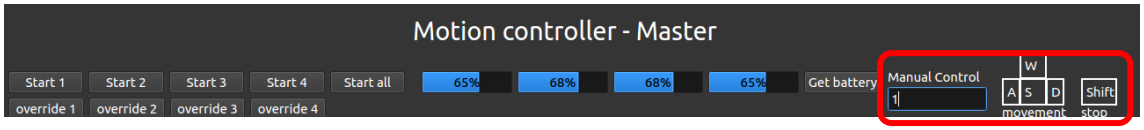



Adicionalmente se puede utilizar el botón GET BATTERY para conocer el estado de batería de los robots presentes en el área de trabajo. Se mostrará el resultado de batería en porcentaje del rango de voltaje entre 3.6 [V] y 4.2 [V].



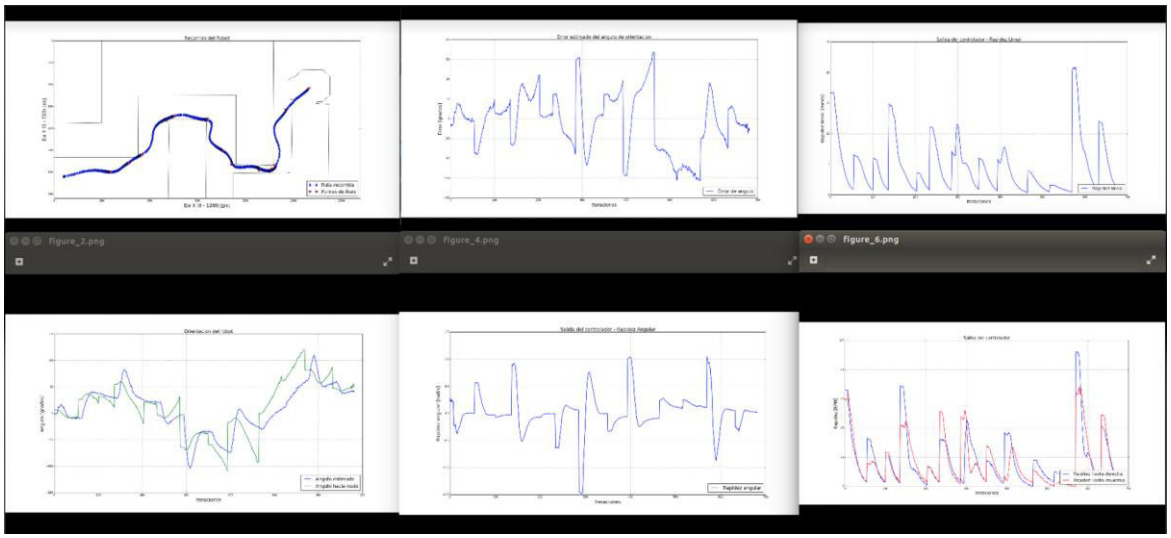

Como herramienta al usuario, se dispone del control manual. Se requiere escribir el ID del robot a controlar. Una vez ingresado el valor, presionar ENTER.

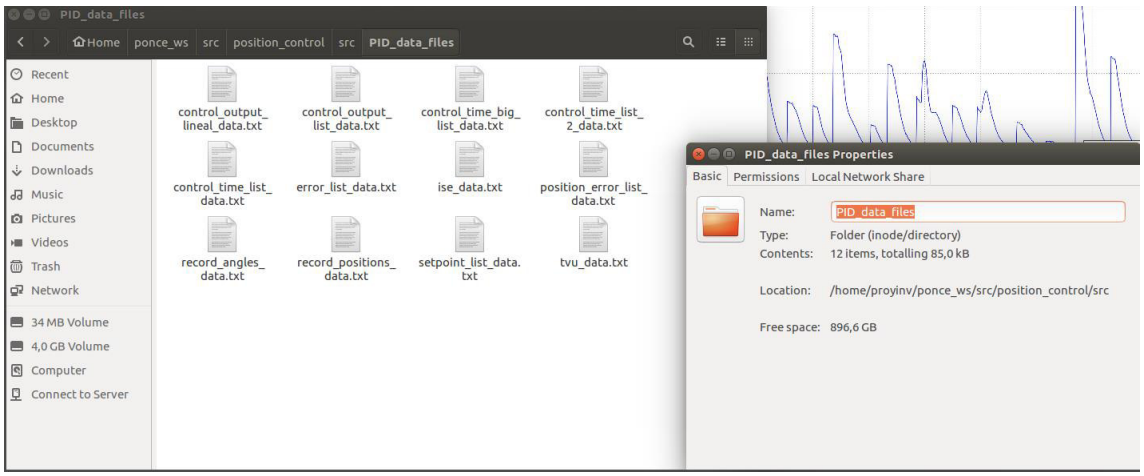
Una vez presionado ENTER se puede controlar al robot seleccionado con las teclas WASD para su movimiento y SHIFT para detenerlo.



## OBTENCIÓN DE RESULTADOS

Las imágenes de los resultados se desplegarán de manera automática una vez completada la ruta de cada robot. Adicionalmente, los datos recolectados durante la ejecución del control, se guardan en un directorio local del computador donde se ejecuta la aplicación.

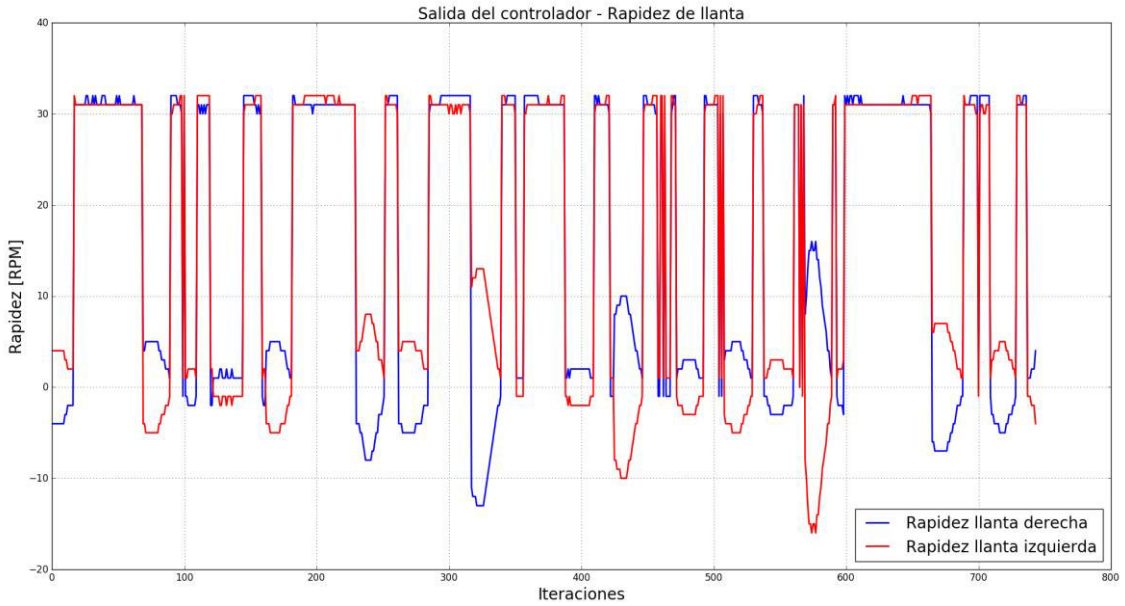


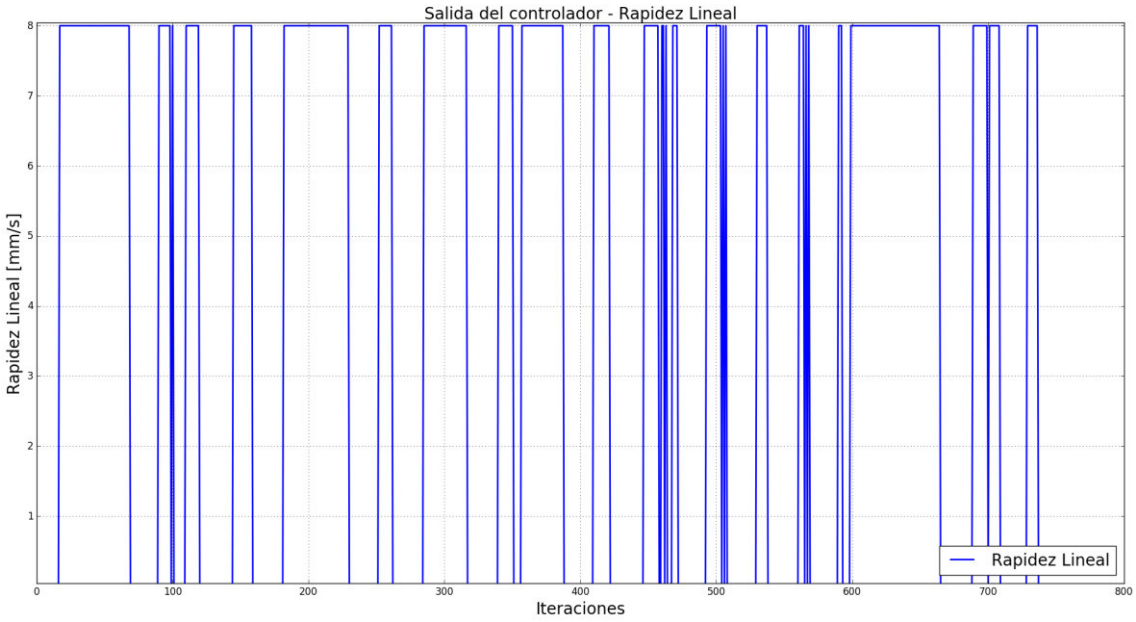



# ANEXO B

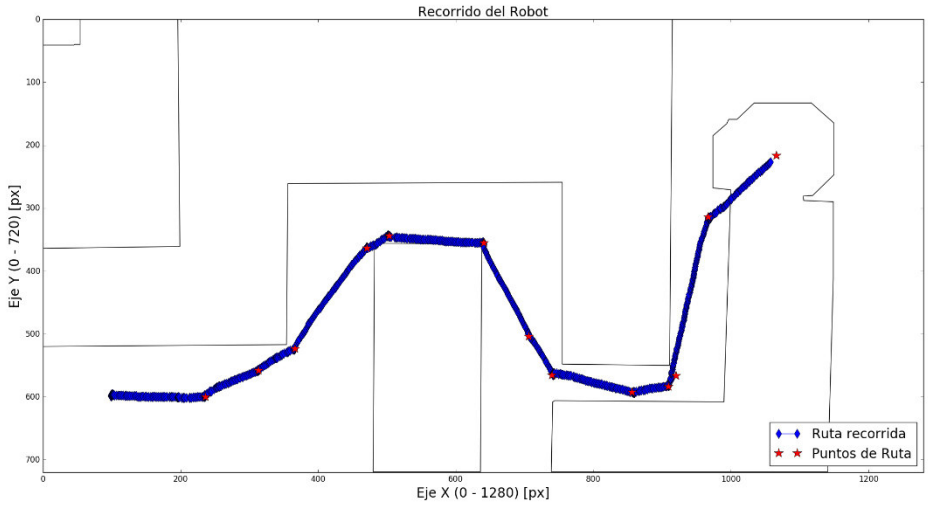
El Anexo B corresponde a las imágenes complementarias de los resultados mostrados en el Capítulo 3.

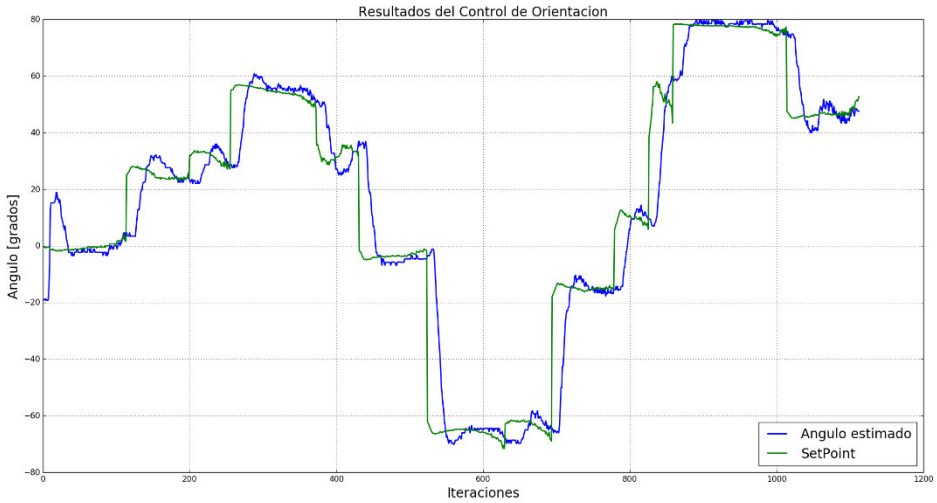
### Seguimiento de Ruta con Control PID 8mm/s

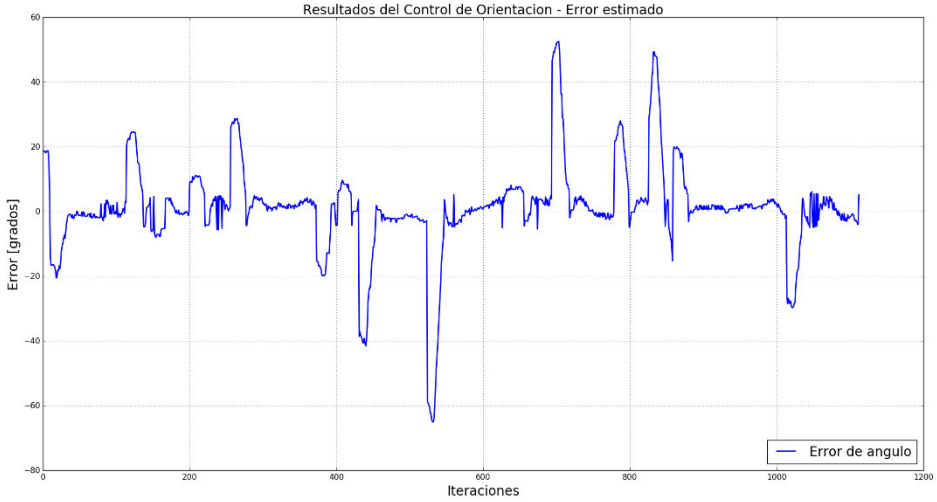


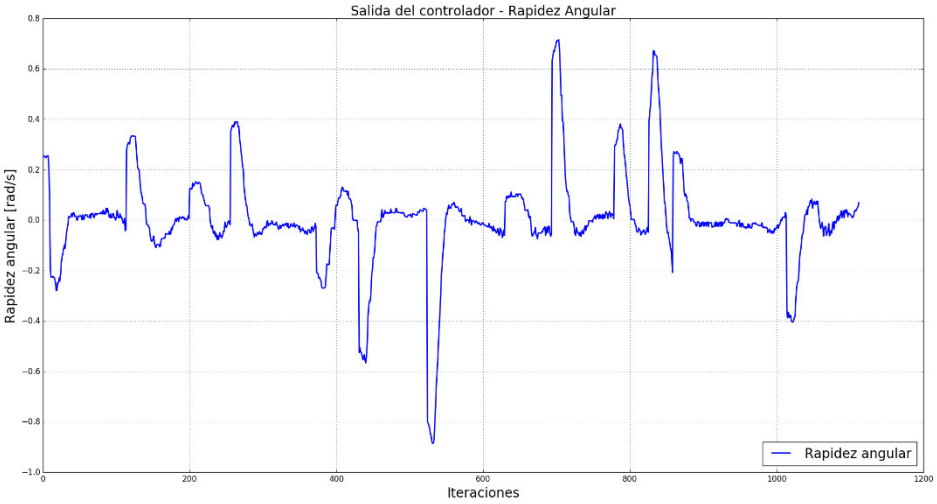



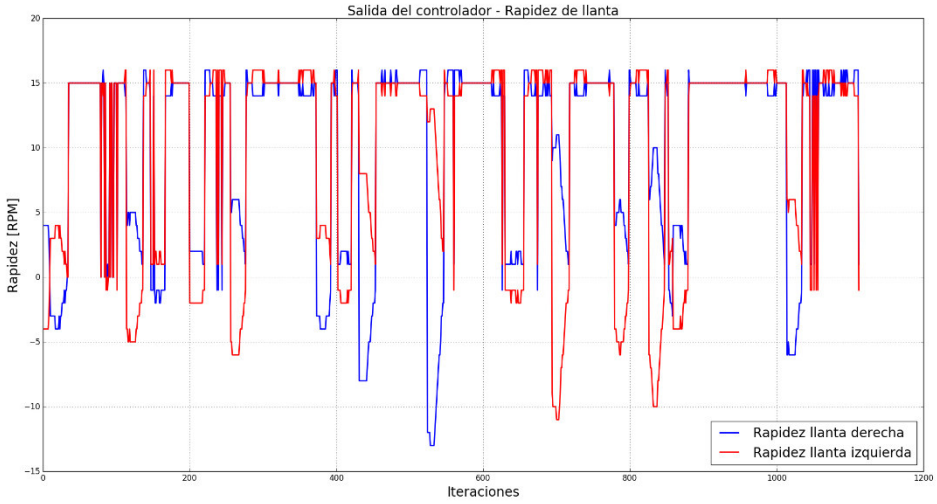
## Seguimiento de Ruta control PID 4mm/s

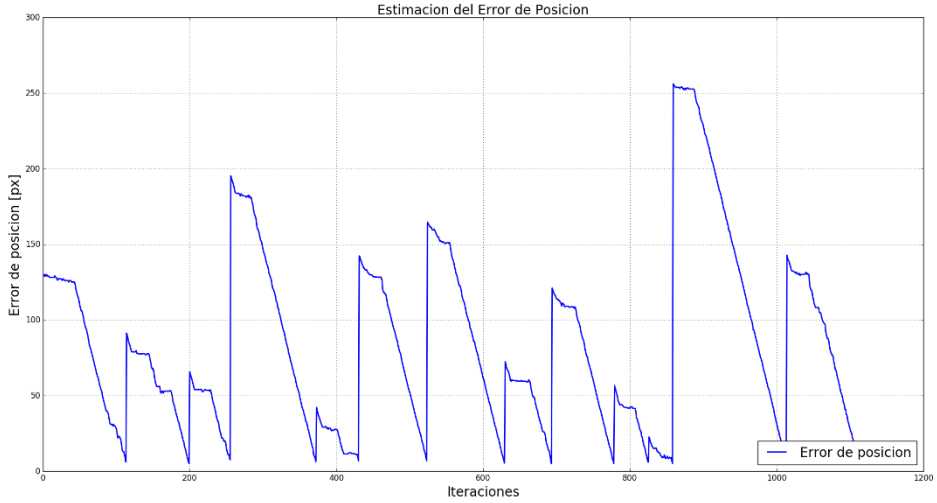


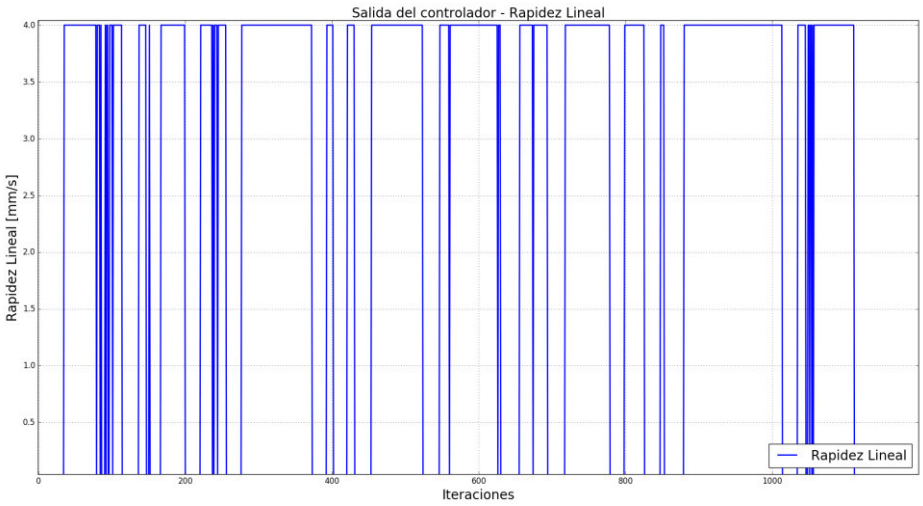



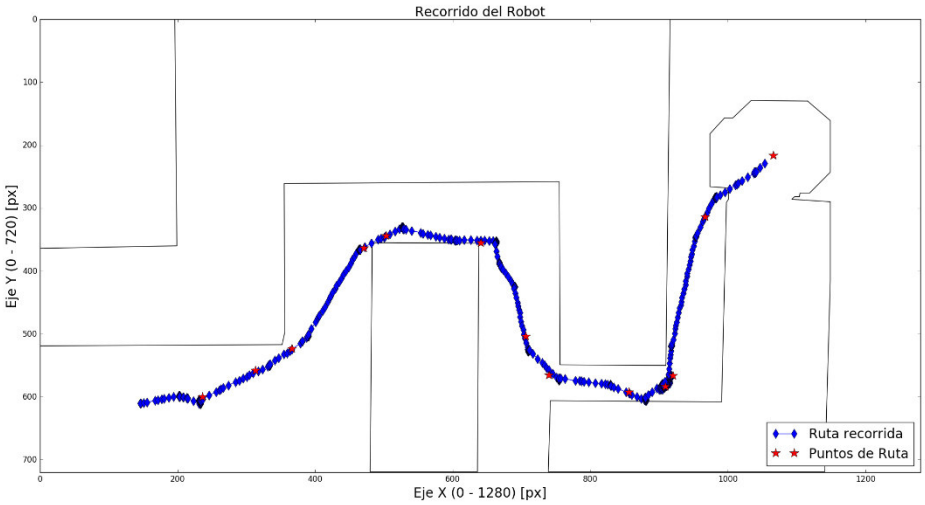

Salida del controlador - Rapidez Angular
Rapidez angular [rad/s]
Iteraciones
Rapidez angular

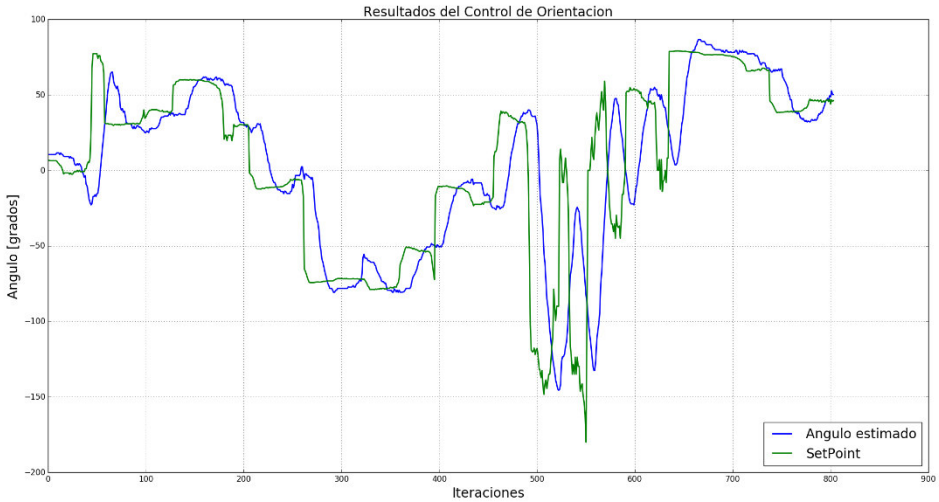

Salida del controlador - Rapidez de llanta
Rapidez [RPM]
Iteraciones
Rapidez llanta derecha
Rapidez llanta izquierda

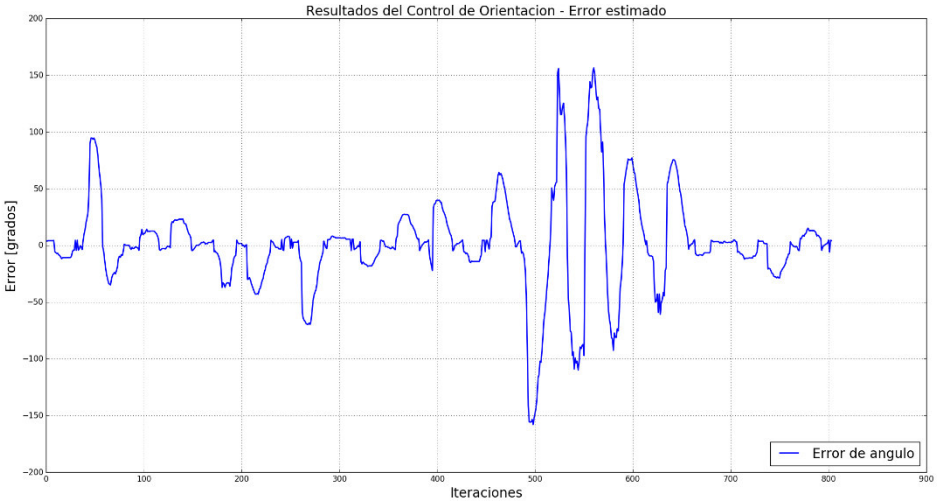

Estimacion del Error de Posicion
Error de posicion [px]
Iteraciones
Error de posicion

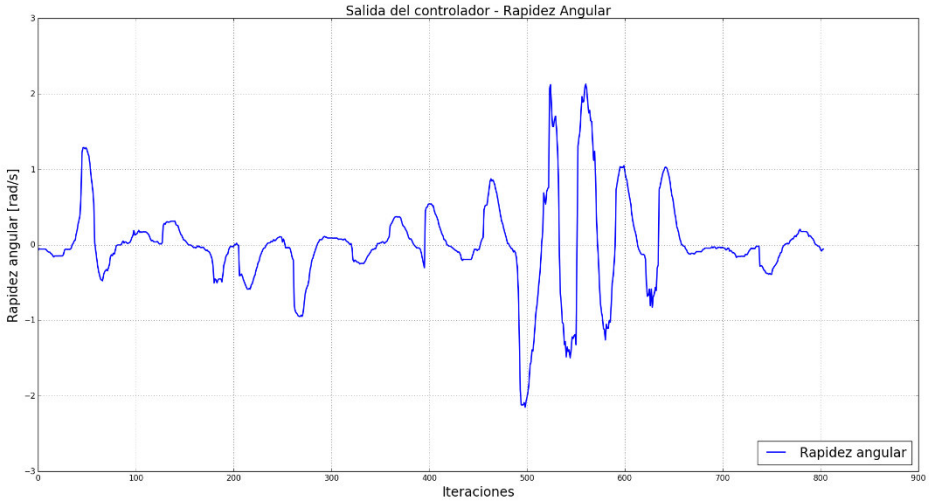


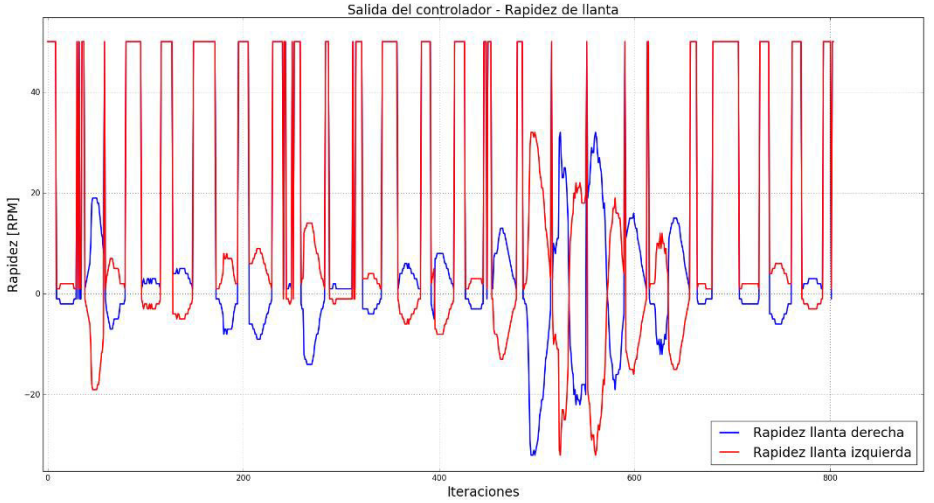

## Seguimiento de Ruta control PID 16mm/s

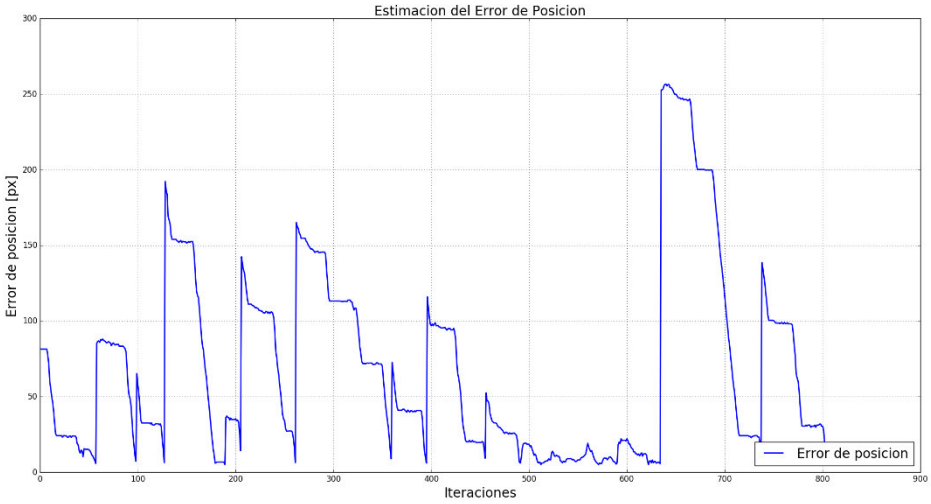


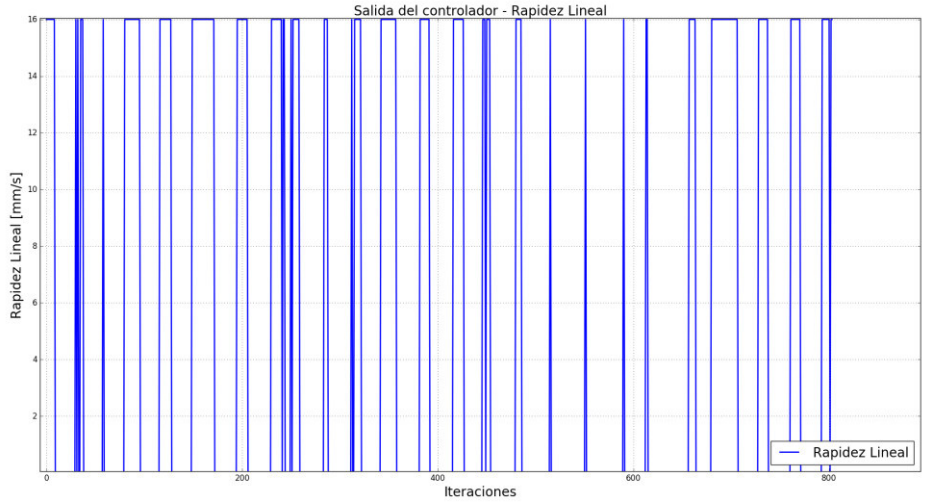




Resultados del Control de Orientacion - Error estimado
Error [grados]
Iteraciones
Error de angulo

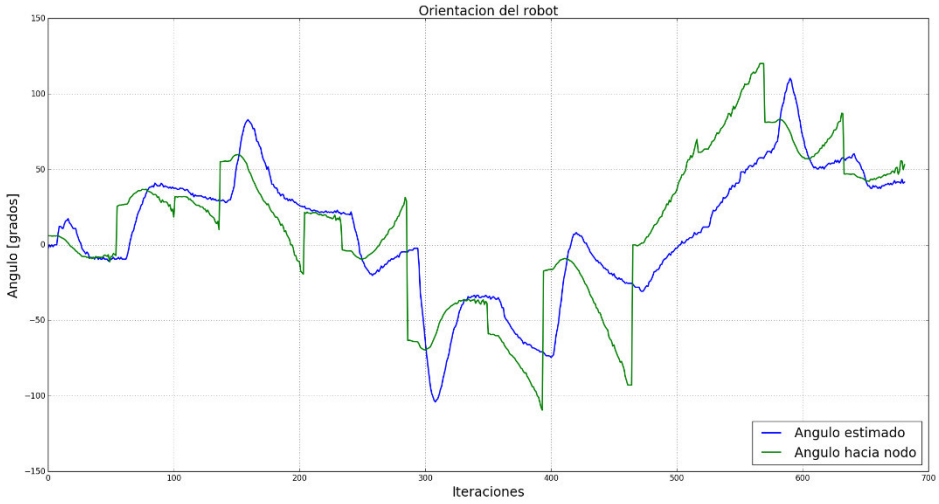

Salida del controlador - Rapidez Angular
Rapidez angular [rad/s]
Iteraciones
Rapidez angular

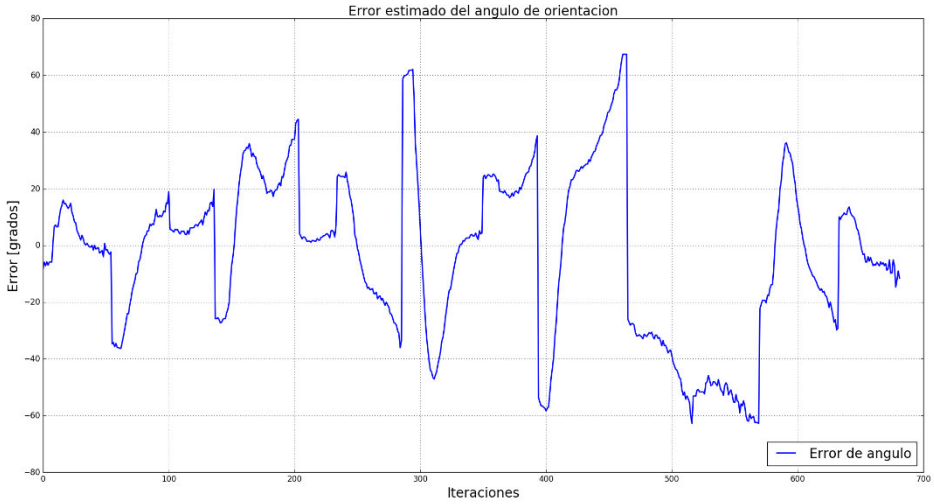

Salida del controlador - Rapidez de llanta
Rapidez [RPM]
Iteraciones
Rapidez llanta derecha
Rapidez llanta izquierda

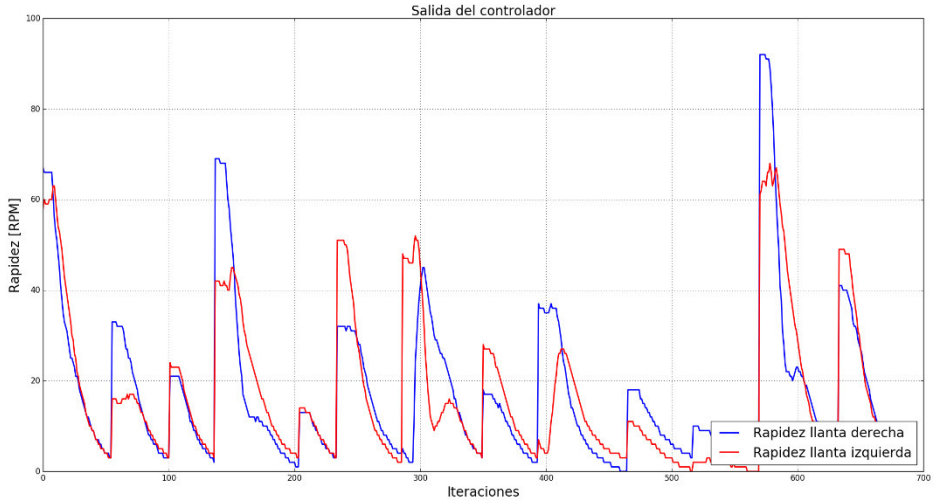


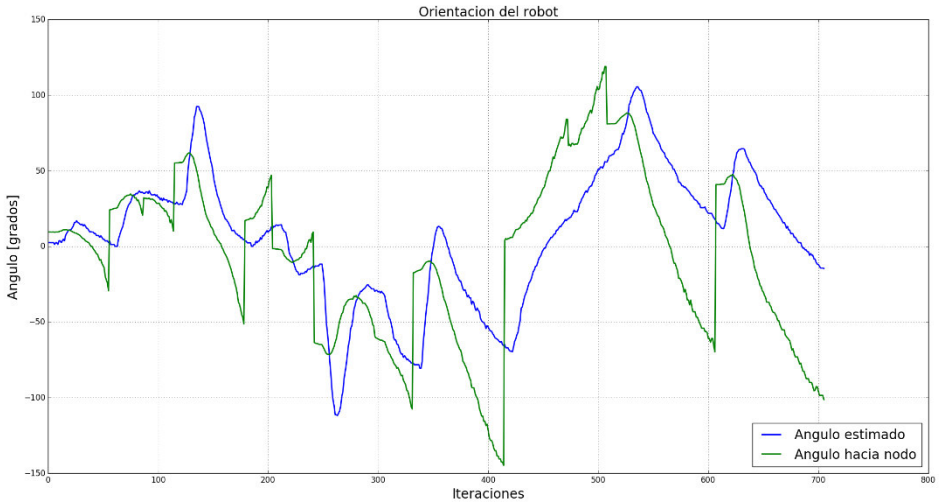

**Seguimiento de ruta control CCI** $K = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix}$

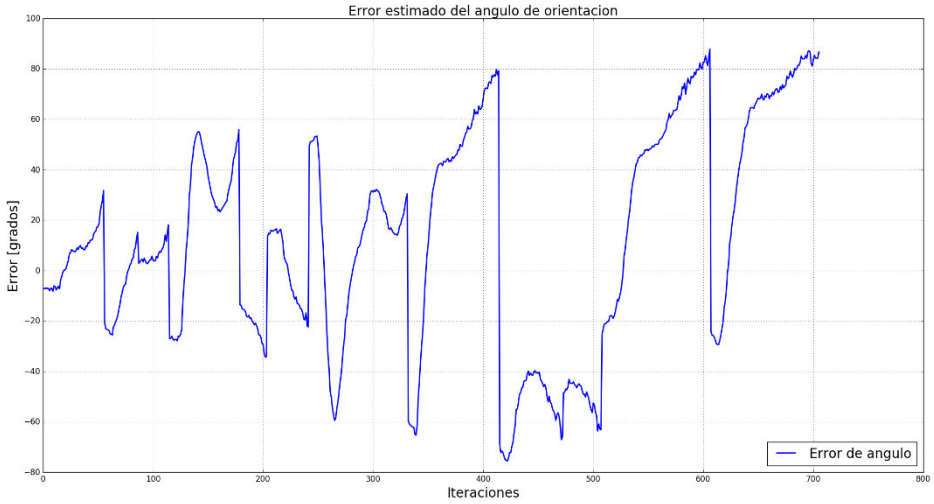


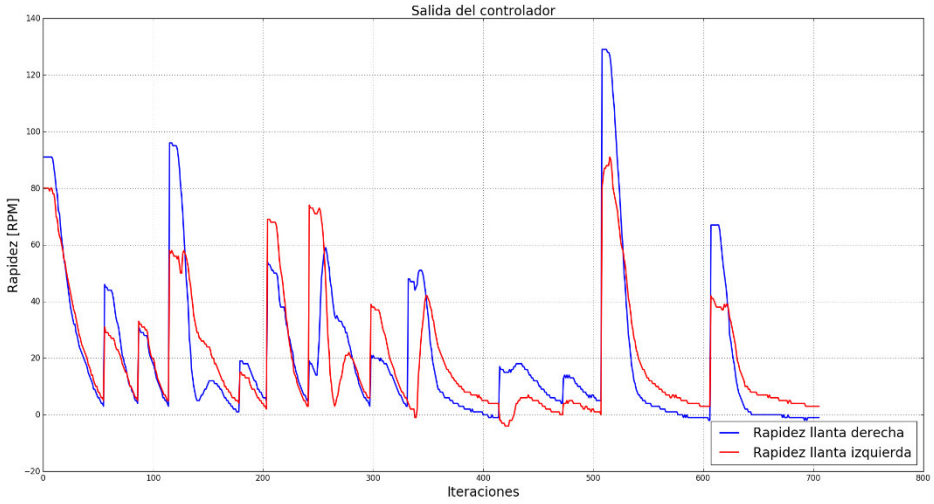



**Seguimiento de ruta control CCI** $K = \begin{bmatrix} 7 & 0 \\ 0 & 7 \end{bmatrix}$

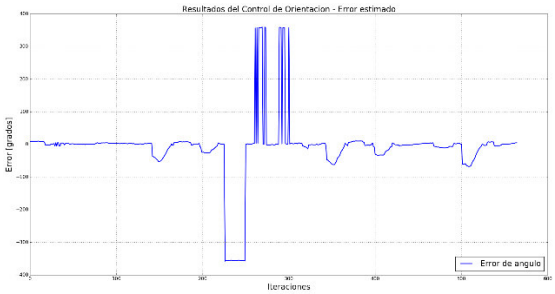




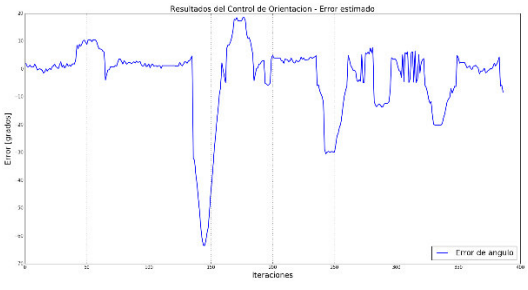




## Seguimiento de ruta con 4 robots simultáneos

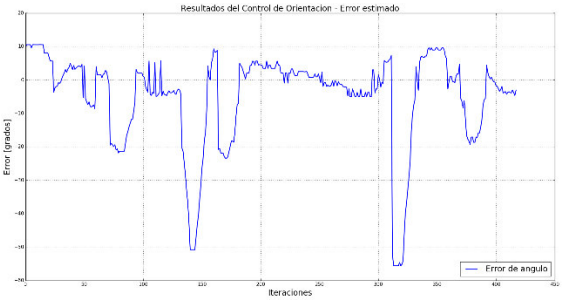
### Error Estimado



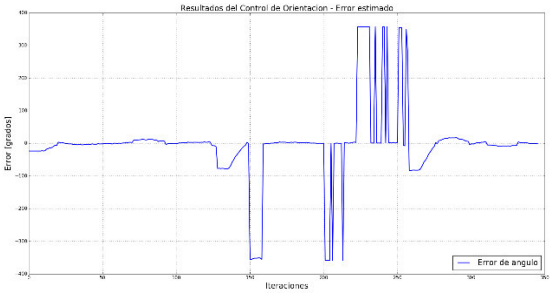

Robot 1




Robot 2

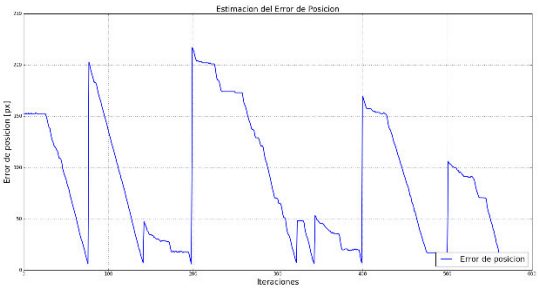



Robot 3

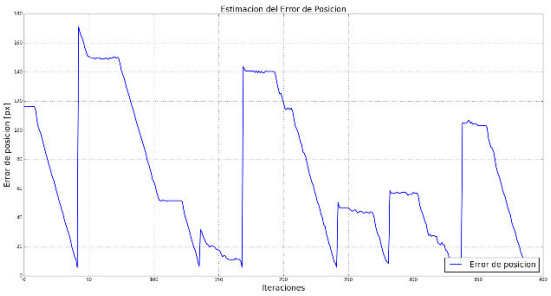



Robot 4

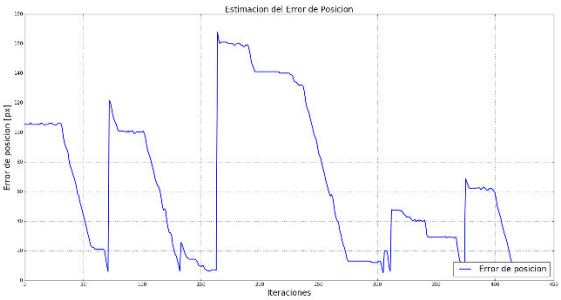
**Error de posición**



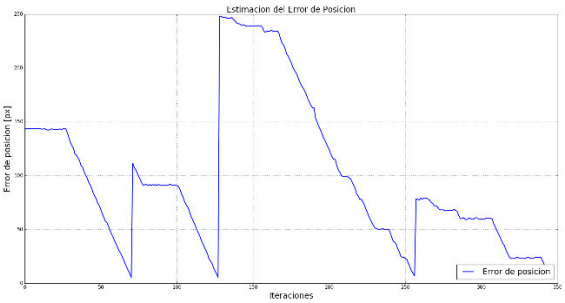

Robot 1




Robot 2

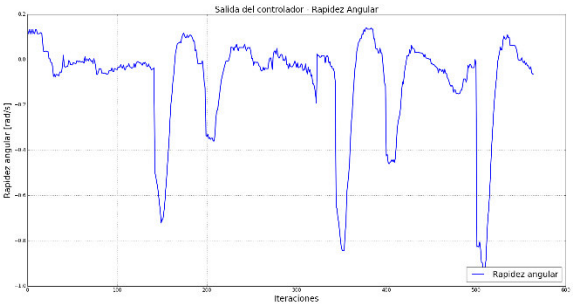



Robot 3

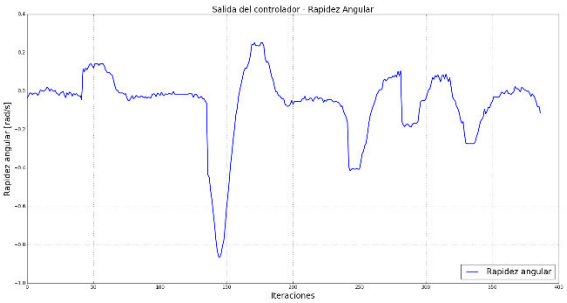



Robot 4

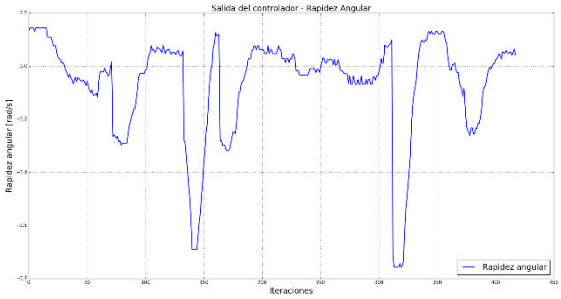
**Rapidez Angular**



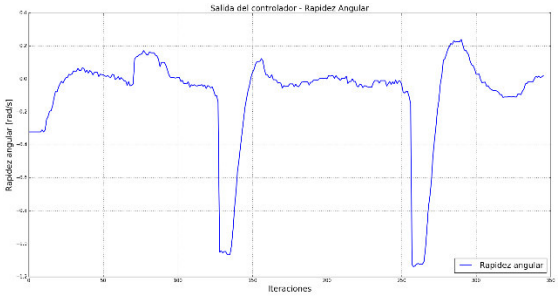

Robot 1




Robot 2




Robot 3




Robot 4

# ANEXO C

Para posibilitar la visualización de los resultados, se creó un canal de YouTube donde se han publicado los videos de las pruebas realizadas.

| | |
|---|---|
| https://www.youtube.com/channel/UCAQIby87xY8kOBrfx7xtjFQ | Canal de YouTube Microbot Arena |

Tabla de enlaces a los videos correspondientes a los resultados mostrados en el Capítulo 3.

| Link de YouTube | Descripción |
|---|---|
| https://www.youtube.com/watch?v=ayElr-l0AiI | Seguimiento de ruta con PID 8mm/s |
| https://youtu.be/BHRD78Ry4Hc | Seguimiento de ruta con PID 4mm/s |
| https://youtu.be/M2S6y3obnP0 | Seguimiento de ruta con PID 16mm/s |
| https://youtu.be/qQQDym6tGi0 | Seguimiento de ruta con CCI K=3 |
| https://youtu.be/RUvGFBQu3yM | Seguimiento de ruta con CCI K=5 |
| https://youtu.be/s5LROefdtLY | Seguimiento de ruta con CCI K=7 |
| https://youtu.be/NSeU2QPNRBU | Seguimiento de ruta con PID – 4 robots simultáneos |
| https://youtu.be/4bQ_teJW7EY | Seguimiento de ruta con CCI – 4 robots simultáneos |

# ORDEN DE EMPASTADO