ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DISEÑO E IMPLEMENTACIÓN DEL CONTROL DE LA FORMACIÓN DE UN GRUPO DE ROBOTS MÓVILES TERRESTRES DE TAMAÑO REDUCIDO ENFOCADO AL SEGUIMIENTO DE TRAYECTORIA BASADO EN EL SISTEMA OPERATIVO ROBÓTICO (ROS).

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

LUIS JEAMPOOL ARCOS ENRIQUEZ
CRISTIAN ANDRES CALALA AYALA

DIRECTOR: ING. PATRICIO JAVIER CRUZ DÁVALOS, PhD.

QUITO, 2020

AVAL

Certifico que el presente trabajo fue desarrollado por Luis Jeampool Arcos Enríquez y Cristian Andrés Calala Ayala, bajo mi supervisión.

ING. PATRICIO JAVIER CRUZ DÁVALOS, PhD.
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Nosotros, Luis Jeampool Arcos Enríquez, Cristian Andrés Calala Ayala, declaramos bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedemos nuestros derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Luis Jeampool Arcos Enríquez

Cristian Andrés Calala Ayala

DEDICATORIA

El presente trabajo lo dedico principalmente a Dios, por darme salud, dame fuerzas, cuidarme y nunca dejarme caer.

A mis padres Luis y Lucia, por ser un ejemplo de superación en mi vida, por todo el amor y el cariño que me han brindado, los amo.

A mis hermanos Antony y Maylin que, con sus locuras alegran mi vida.

Y a mi hermosa novia Magali, por haberme dado todo el apoyo incondicional y ser mi soporte para levantarme cuando me sentía derrotado, Te amo.

Luis Jeampool Arcos Enríquez

DEDICATORIA

A mis padres Ulpiano y Elena que han sido el mayor regalo que pudo darme la vida y me han apoyado a lo largo de este proyecto y mi vida.

A mis hermanos Nicolas y Leonardo por todo su cariño y confianza.

.

Cristian Andrés Calala Ayala

AGRADECIMIENTO

Agradezco a mi Dios por mantener a mi familia unida, porque en toda esta etapa de mi vida me ha permitido conocer personas maravillosas que con su granito de arena han dado alegría a mi vida.

A mis padres por su amor, trabajo y sacrificio. A mi ma Lucia por mantener el hogar siempre unido con su amor y ternura. A mi pa Luis porque día a día se esfuerza para que nunca nos falte nada, son lo mejor que tengo en mi vida.

A mis hermanos por todas las risas y enojos, los amo muchísimo. Gracias ñaña por endulzar mi vida con tus postres cuando me siento triste. Gracias ñaño por todos los momentos que hemos pasado, espero seguir compartiendo contigo en esta nueva etapa de tu vida, la universidad, tienes todo mi apoyo.

A mi novia Magi, porque día a día me enseña a ser mejor persona, con ella me siento tan amado y feliz. Gracias por todos los momentos que hemos pasado juntos, todos los viajes, todos los problemas, todas las alegrías y todos los momentos que vendrán, no te imaginas cuanto te amo conejita.

A mi pana de tesis, gracias por ser un amigo incondicional, por haberte esforzado y haber dado todo para culminar este proyecto. Caliman espero que tu vida este llena de éxitos y recuerda que cuando nos proponemos algo podemos conseguirlo, siempre puedes contar conmigo.

A mis mejores amigos del cole que son mi segunda familia, Marleins, Jona, Llumi, Henry, Santi y Juaneis porque a pesar de los años seguimos manteniendo esa amistad. Pilas muchachos, amigos hasta viejos, algún día seremos campeones en el Proyección.

A mis amigos del conjunto que son como mis ñaños y los conozco desde la infancia, Eve y Marco los quiero muchísimo, gracias por todo.

A lo largo de esta etapa he vivido momentos maravillosos y he conocido personas muy valiosas, gracias a Angel, Dario, Guirri, Juampa, Diux, Santi, Liz, Jenny, Arita, Paito, Sawerseins, Juani, Abuelito, David, Dupray, Margarita y a muchas personas más que a pesar de los problemas y diferencias, hemos sabido darnos la mano y salir adelante, espero mis amigos colegas seguir contando con su amistad, sus conocimientos y la alegría que los caracteriza a cada uno de ustedes. Suerte y muchos éxitos.

Gracias Dr. Patricio Cruz por su apoyo, por su tiempo y la confianza que nos brindó al realizar este proyecto. Ing. Diego Maldonado gracias por brindarnos su conocimiento que

nos ayudo a arrancar con nuestro trabajo y a Marisol que además de su amistad nos brindó todas las herramientas para poder entender, continuar y ampliar el estudio de su trabajo de titulación.

Luis Jeampool Arcos Enríquez

AGRADECIMIENTO

En primer lugar, quiero agradecer a mis padres quienes siempre estuvieron apoyándome a lo largo de vida, a ellos quienes me han dado un gran ejemplo de esfuerzo y perseverancia. Son los mejores padres del mundo, gracias por todo los quiero mucho.

A mis hermanos Nico y Leo, que siempre están de un modo y otro ayudándome, hasta desestresándome, contando algún dato curioso o jugando juegos conmigo. Saben que pueden contar conmigo siempre.

A mi tía y madrina Ede, que es mi segunda madre, siempre me demostró con su ejemplo el verdadero significado de ser una persona altruista. Gracias, tía por todo tu cariño.

A mi amigo, hermano y medio primo de toda la vida Israel, loco gracias de verdad por acolitarme en todo y siempre demostrarme que no hay que rendirse pese a las dificultades.

A Jeampool que más que un compañero de tesis es un gran amigo, empezamos juntos este proyecto con la meta de aprender algo totalmente nuevo y gracias a ti amigo lo conseguimos.

Gracias Doc. Patricio Cruz por su tiempo y guía durante este proyecto, sus consejos fueron vitales para alcanzar todos los objetivos.

A mis amigas Vane, Jessy, Maggie, Vane que siempre estuvieron cuando más las necesite a lo largo de la carrera. En especial a Jessy, hermana gracias por estar presente en mi vida, aunque estés a miles de kilómetros.

A mi grupo de amigos Cristian, Christian, Darío, David, André, Carmen como siempre decíamos somos un grupo pequeño, pero bien bulliciosos.

Finalmente, a todos mis amigos y amigas de la universidad gracias por todo su apoyo y confianza.

Cristian Andrés Calala Ayala

ÍNDICE DE CONTENIDO

RESUMEN

Este trabajo de titulación presenta el diseño e implementación de un banco de pruebas experimental para el control de formación de un grupo de tres robots móviles de tamaño reducido enfocado al seguimiento de trayectoria. Se propone una arquitectura de control en cascada que consiste en un lazo interno de velocidad en serie con un controlador externo que ayuda a mantener la formación y postura deseadas del grupo de robots en todo momento.

El monitoreo para el seguimiento de la posición de cada robot en el área de trabajo se basa en el seguimiento, mediante visión artificial, de marcadores de referencia ArUco. El sistema de monitoreo junto con los sistemas de comunicación y control están implementados bajo el empleo de ROS (Sistema Operativo Robótico), el cual facilita su interconexión. Además, un HMI desarrollado en QT Creator facilita la modificación de parámetros de los controladores interno y externo, así como en la realización de pruebas experimentales y en la visualización de sus resultados.

Gracias al banco de pruebas implementado, el esquema de control propuesto es sometido a varias pruebas utilizando diferentes trayectorias predefinidas, incluyéndose para algunos casos perturbaciones. Los resultados obtenidos demuestran la efectividad del controlador al mantener la formación del grupo de robots mientras se realiza el seguimiento de una trayectoria deseada.

PALABRAS CLAVE: ROS, banco de pruebas, formación de robots, control de formación, sistema de monitoreo basado en visión, seguimiento de trayectorias.

ABSTRACT

This project presents the design and implementation of an experimental testbed for the control formation of three small-sized mobile robots focused on trajectory tracking. For the control design a cascade architecture is proposed. It consists of an internal PID speed loop in series with an external controller that maintains the desired formation and posture of the group at all times.

The robot pose tracking in the work area is based on artificial vision with the help of the ArUco reference markers. The monitoring system together with the communication and control systems are implemented in ROS (Robotic Operating System). This framework facilitates the interconnection between these modules. In addition, an HMI developed in QT Creator allows to change parameters of the internal and external controllers, to setup different tests and visualize their results.

The proposed control scheme is validated by using different predefined trajectories including disturbances in some cases. The experimental results demonstrate the effectiveness of the designed controller on maintaining the group formation while tracking a desired path.

KEYWORDS: ROS, Multi-robot Testbed, Formation Control, Vision-based Monitoring, Trajectory tracking.

1. INTRODUCCIÓN

El desarrollo de la robótica ha permitido crear en la industria líneas de producción y logística más eficientes; sin embargo, el reto actual es la introducción de sistemas robóticos a la vida cotidiana. Para alcanzar ese objetivo se requiere de robots autónomos, que transporten objetos, que se movilicen en entornos complejos y que trabajen de forma cooperativa. Las aplicaciones de la robótica cooperativa son numerosas y van desde la vigilancia, movimiento de objetos hasta la búsqueda y rescate. Un ejemplo de esto es la carrera robótica que se ha dado entre Amazon y Alibaba con su flota de robots KIVA y QuicktronRobots, respectivamente. Estas empresas utilizan un equipo de robots que transportan y ordenan los productos alrededor de todo un almacén evitando colisiones y trabajando en forma de enjambre. [1]

Por otro lado, la evaluación comparativa es muy importante en el campo de la robótica ya que permite contrastar diferentes soluciones para un determinado problema, ya sea mecánico o enfocado a los algoritmos de control para coordinaciones estratégicas de grupos de robots. En este campo existen trabajos como Multi-Robot Exploration Testbed, que, mediante un diseño experimental con sus respectivos parámetros definidos por el usuario, permiten una simulación realista de los diferentes experimentos que la plataforma basada en ROS (Robotic Operating System) ofrece. Un ejemplo es la exploración autónoma con una flota de robots, la cual utiliza el intercambio de mapas entre ellos como estrategia de coordinación. [2]

En la Escuela Politécnica Nacional se dispone actualmente de un testbed que permite realizar pruebas de experimentación de algoritmos de control con dos robots móviles terrestres de tamaño reducido para el control de posición y trayectoria de forma no cooperativa. Este banco de pruebas esta realizado en Open CV y Visual Studio para el manejo del sistema de posicionamiento, y Matlab para el desarrollo y prueba de los algoritmos de control [3]. Visual Studio y Matlab son programas que manejan licencias y no son de código abierto; además, el sistema tiene una carga computacional grande que limita las aplicaciones que se pueden realizar.

Por tal motivo el presente proyecto mejora este banco de pruebas para que sea más eficiente respecto a su carga computacional, migrándolo a un sistema de código abierto como lo es Linux y ROS [4][5]. El banco de pruebas constará de dos componentes principales hardware y software. El primer componente mejorado será la adhesión de un tercer robot móvil terrestre a los dos ya existentes y la implementación de encoders en cada llanta de cada uno de los robots. El área de trabajo y la cámara web de alta definición

colocada en un pedestal y marcadores de referencia matriciales serán los mismos utilizados en el banco de pruebas a mejorarse, pero su manejo será sobre Linux/ROS. El segundo componente mejorado será la implementación del sistema de monitoreo para el rastreo individual de cada robot móvil y los algoritmos de control, permitiéndoles moverse manteniendo una formación de tres robots móviles mientras siguen una trayectoria preestablecida. Todo el sistema correrá sobre Linux/ROS; así como su respectiva interfaz gráfica. Por lo que este proyecto al ser de código abierto facilitará el desarrollo de algoritmos más complejos con un libre intercambio de información sirviendo de apoyo a futuras investigaciones en el campo de la robótica móvil.

1.1 OBJETIVOS

El objetivo general de este trabajo es:

Diseñar e implementar un control de la formación de un grupo de robots móviles terrestres de tamaño reducido enfocado al seguimiento de trayectoria basado en el sistema operativo robótico (ROS).

Los objetivos específicos de este trabajo son:

- Realizar la revisión bibliográfica del control para seguimiento de trayectoria de un grupo de robots móviles terrestres, así como el manejo y aplicación del sistema operativo robótico ROS en sistemas de control.
- Diseñar e implementar un algoritmo para un control interno de velocidad para cada robot y un control externo de seguimiento de trayectorias que mantenga la formación de tres robots móviles terrestres.
- Migrar el banco de pruebas desarrollado en [3] al sistema operativo Linux/ROS que conste de tres robots móviles, un sistema de monitoreo/posicionamiento y sistema de comunicaciones inalámbricas.
- Diseñar una interfaz gráfica en un software libre que corra sobre Linux que permita al usuario monitorear y tener control sobre los parámetros del sistema.
- Analizar y comparar el desempeño de los controladores diseñados (control interno de velocidad y control de formación y seguimiento de trayectoria), mediante el cálculo de errores en pruebas grupales de la formación de tres robots móviles con al menos dos trayectorias predefinidas.

1.2 ALCANCE

- Se realiza la correspondiente revisión bibliográfica para el manejo de mensajes, nodos, tópicos, publicadores, subscriptores y un análisis relacionado con el manejo de visión artificial, comunicación inalámbrica e interfaces gráficas en la plataforma ROS.

- Se realiza la correspondiente revisión bibliográfica sobre técnicas de seguimiento de trayectoria para equipos de robots móviles, para el diseño e implementación de un algoritmo de control de seguimiento de trayectoria mientras se mantiene la formación del grupo de tres robots.

- Del banco de pruebas existente en la Escuela Politécnica Nacional, se utiliza la cámara web de alta definición, el soporte que permite mantener a la cámara a una altura adecuada para la detección de los robots, los marcadores de referencia matricial ArUco y la plataforma de madera de 120cm x 80 cm que conforma el área de trabajo, a esto se le suma un robot móvil adicional y se integran encoders en los ejes de los motores de los tres robots móviles para implementar un control interno de velocidad, así como un algoritmo para mantener la formación del grupo de tres robots mientras siguen trayectorias predefinidas.

- En cuanto al sistema de monitoreo con visión artificial y de comunicación inalámbrica entre el ordenador y robots móviles del banco de pruebas existente, se lo migrará al sistema operativo Linux-ROS en donde además se diseñará una interfaz gráfica (HMI) desarrollada en un software libre y de código abierto para fácil operabilidad y monitoreo del sistema.

- Se realizan pruebas tanto del sistema de monitoreo, comunicación y control para comprobar el desempeño del banco de pruebas integrado, además de la verificación del funcionamiento de los controladores (control interno de velocidad y control de formación y seguimiento de trayectoria) diseñados e implementados mediante el cálculo de errores en pruebas experimentales. En el caso del control interno de velocidad estos errores son respecto a valores de referencia y para el control de formación son con respecto al seguimiento de al menos dos trayectorias predefinidas.

1.3 MARCO TEÓRICO

En este capítulo se recopila la información teórica relacionada con el desarrollo de este proyecto. Para ello se realiza un breve análisis de robótica móvil, los fundamentos básicos de los sistemas de control implementados, así como fundamentos de visión artificial y el análisis de los sistemas operativos sobre los cuales se desarrolla el proyecto.

Robótica Móvil

La Robótica Móvil se refiere a los dispositivos que tienen un sistema de locomoción capaz de movilizarse sobre diferentes ambientes de trabajo ya sean estructurados o no, con la posibilidad de alcanzar determinados objetivos con cierto grado de autonomía, dependiendo de la fuente energética y la poca o nula intervención de un operador humano. Los diferentes sistemas de control han permitido que la robótica móvil tenga muchas aplicaciones en diferentes campos que van desde el transporte de cargas peligrosas, tareas de exploración, reconocimiento de terrenos, hasta tareas domésticas y de oficina [6].

Clasificación de la Robótica Móvil [7]

Dependiendo de su forma de locomoción, los robots móviles se pueden clasificar en:

- Locomoción con ruedas.
- Robots con patas.
- Robots articulados.

Locomoción con ruedas [8]

Los robots móviles con ruedas están diseñados para desplazarse en terrenos suficientemente estables, con velocidades altas y asegurando estabilidad con un mínimo de 3 ruedas. Dependiendo de las diferentes configuraciones que se tenga en el diseño se puede mejorar su estabilidad, su controlabilidad y su manera de desenvolverse en diferentes superficies. Estas configuraciones se pueden observar en la Figura 1.1.

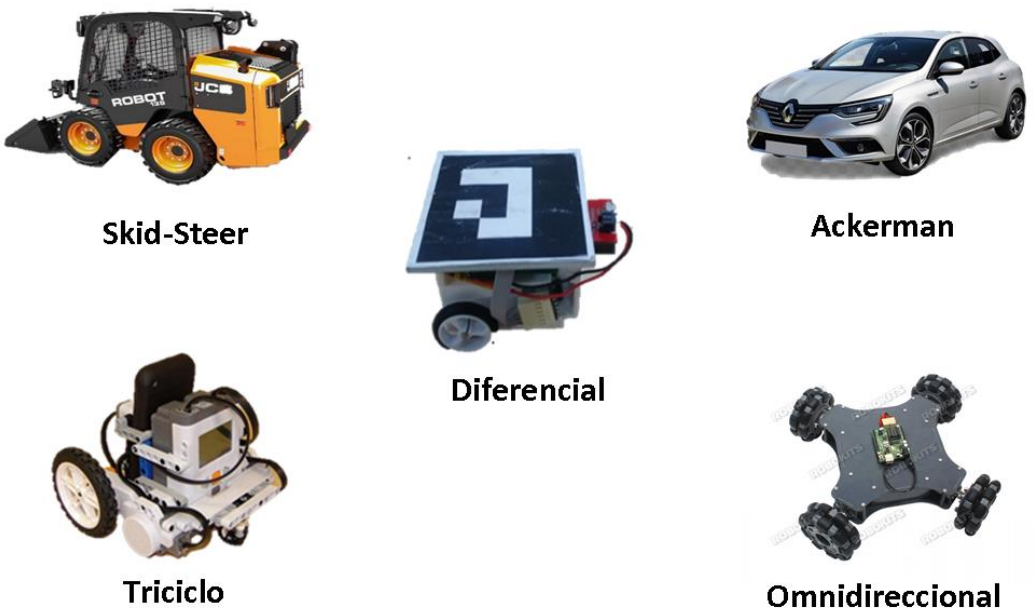


Figura 1.1 Configuraciones Locomoción por ruedas.

Skid-Steer

Esta configuración tiene más de una rueda en cada lado del robot móvil que giran a la misma velocidad por lado. Los giros y el desplazamiento dependen de la combinación de las velocidades de las ruedas de lado derecho e izquierdo. Esta configuración es robusta y con buena maniobrabilidad con la desventaja que durante los cambios de dirección las ruedas producen un deslizamiento lateral la cual causa una mala estimación de la posición del robot móvil.

Ackerman

Esta configuración reduce el deslizamiento con buena estabilidad a altas velocidades, es la utilizada en los vehículos convencionales.

Triciclo

Esta configuración consta de tres ruedas, dos ruedas posteriores fijas de tracción y una rueda delantera motriz que controla la dirección del robot. Estos robots son de fácil implementación, tienen bajo deslizamiento, pero son inestables con maniobrabilidad limitada. Comúnmente se los utiliza para llevar cargas pesadas a baja velocidad.

Omnidireccional

Esta configuración permite desplazar al robot en cualquier dirección sin necesidad de reorientarse.

Diferencial

En esta configuración se tiene dos ruedas fijas perpendiculares en un mismo eje las cuales controlan los giros del móvil dependiendo de la velocidad de cada una, además para generar mayor estabilidad se adicionan una o dos ruedas locas, como se puede ver en la Figura 1.2. Considerando que estos robots son de bajo costo y de fácil implementación, se elige el diseño de robot diferencial como modelo de prototipos del presente banco de pruebas.

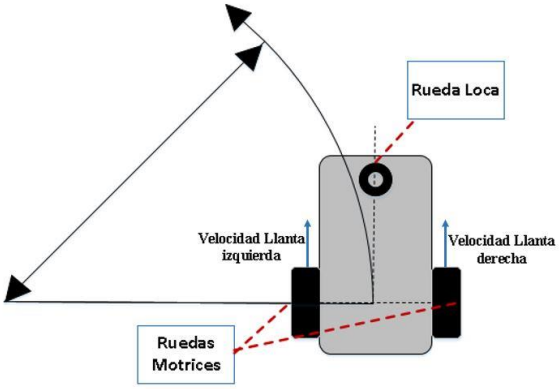


Figura 1.2 Configuración tipo diferencial

Robots con patas

Los robots móviles con patas están diseñados para desplazarse en terrenos irregulares, son capaces de superar obstáculos superiores a su tamaño. Su principal inconveniente es su complejo diseño mecánico y sus sistemas de control. Un ejemplo de esto se puede observar en la Figura 1.3, el "SpaceBok", prototipo de robot con cuatro patas desarrollado por estudiantes del ETH Zurich que está siendo probando por la Agencia Espacial Europea para sustituir a los robots Rover en futuras misiones en Marte o la Luna debido a su versatilidad y su forma de adaptarse a diferentes terrenos. [9]

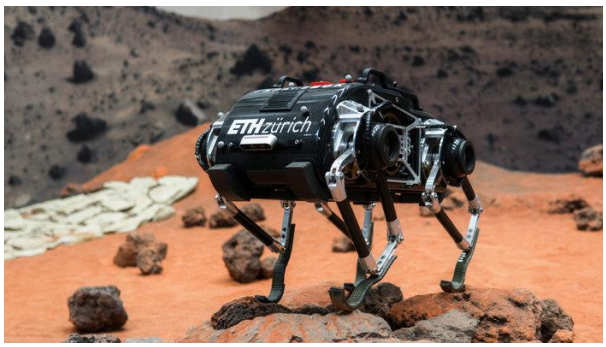


Figura 1.3 Space Bok – El robot saltador para la exploración espacial [9]

Robots articulados

Los robots móviles articulados, están diseñados para desplazarse en terrenos difíciles, estos robots poseen múltiples grados de libertad debido a los varios segmentos que poseen, de manera similar a las serpientes como se observa en la Figura 1.4.

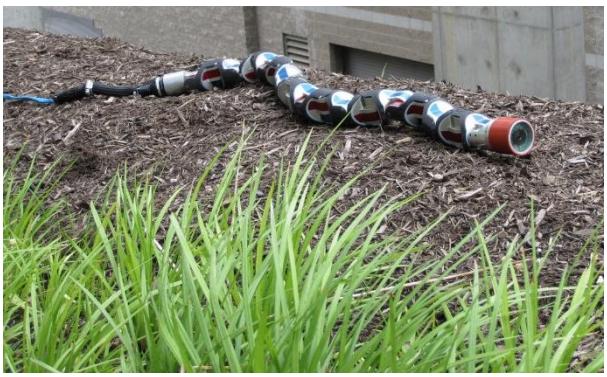


Figura 1.4 Robot Serpiente explorando en los escombros del terremoto de México 2017 [10]

Sistemas Multi-agentes [11]

En los últimos años, la robótica móvil se ha desarrollado aceleradamente, sin embargo, existen problemas que son complejos o imposibles de resolver para un solo robot, así que se ha visto la necesidad de coordinar múltiples robots móviles para poder solucionar dichos problemas.

Trabajar de forma colaborativa permite tener diferentes aplicaciones como, por ejemplo, en la exploración y mapeo de entornos desconocidos, transporte y manipulación de cargas, vigilancia y seguridad, detección de minas, entre otros. Las ventajas que ofrece trabajar con grupos de robots son:

- **Robustez:** Si un robot de la flota sufre un daño otro puede cumplir su función evitando que el sistema se comprometa.
- **Escalabilidad:** Al ser sistemas abiertos se puede ingresar nuevos robots al sistema sin hacer grandes cambios en él.
- **Cobertura:** En aplicaciones como búsqueda y rescate o mapeo de territorios, los robots se pueden distribuir cubriendo mayor área geográfica.
- **Simplicidad:** Cada robot tiene un diseño más simple que los sistemas de un solo robot, además los sistemas multi-agentes son modulares.

Arquitecturas de Software de control

En un enjambre o grupo de robots, para el control y las comunicaciones entre ellos existen diferentes arquitecturas de software las cuales son:

Arquitectura Centralizada

Como se observa en la Figura 1.5, las arquitecturas centralizadas tienen una unidad central que procesa, controla, supervisa y envía las acciones que deben tomar cada uno de los robots en la flota. Esta unidad central resuelve todos los problemas que se relacionan con cada robot y transmite las órdenes directamente a cada unidad; para dicha transmisión de información se debe establecer un sistema de comunicación entre la unidad central y los robots de la flota.

El problema con este sistema es que si la unidad central falla, el sistema completo deja de funcionar.

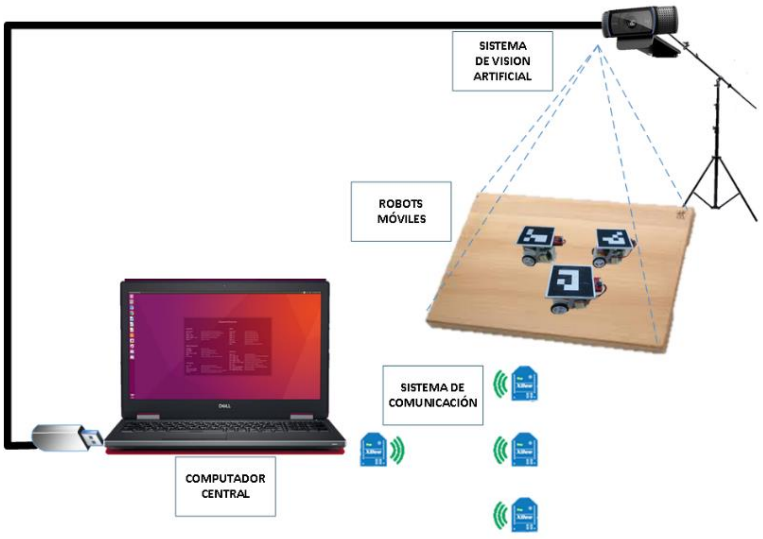


Figura 1.5 Arquitectura del control centralizado

Arquitectura descentralizada

En este tipo de arquitectura los robots no dependen de ninguna unidad central, a través de sensores, sistemas de auto localización, sistemas de autoplanificación integrado y el sistema de comunicación entre los robots de la flota, caracterizan el entorno y cada robot toma sus propias decisiones como parte de la estrategia de equipo para llegar a un objetivo. Un ejemplo de ello se puede observar en la Figura 1.6, donde cada robot toma sus propias decisiones para lograr una anotación por equipo.

La ventaja de esta arquitectura es que, si uno de los robots de la flota falla, los demás seguirán con su objetivo. La desventaja es que el sistema y los algoritmos de control son muy complejos, se debe controlar gran cantidad de variables y se debe tener un robusto sistema de comunicación entre ellos.

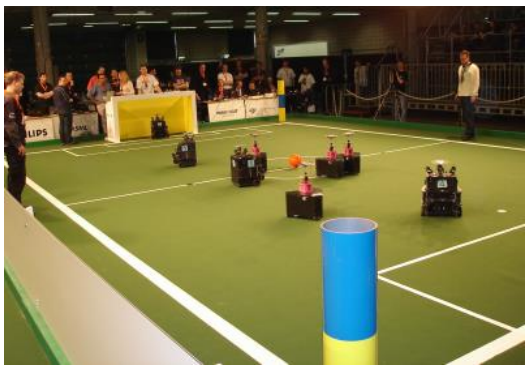


Figura 1.6. Torneo Robocup, Arquitectura distribuida entre los robots de cada equipo. [12]

Formación de un grupo de robots

Trabajar con formaciones de grupos de robots se ha vuelto más popular en los últimos años, desarrollando una serie de aplicaciones como sistemas de carreteras automatizadas, formaciones autónomas de aeronaves por parte de la NASA y por la fuerza aérea en diferentes países; así como, tareas que requieren cooperación como equipos de robots para juegos, exploración, manejo cooperativo de objetos etc.

Para lograr dichas formaciones existen las siguientes técnicas:

- **Estructura Virtual:** Esta técnica trata a toda la formación de robots como una estructura sólida tomando en cuenta que las relaciones geométricas no son impuestas por las fuerzas moleculares que mantienen unido a un objeto sólido sino por un sistema hecho por el hombre. Como el sistema es considerado como una estructura rígida, las referencias de posición de cada robot están referenciadas a un solo punto de la estructura, así se puede asignarle a la estructura una trayectoria y orientación para la movilidad de todo el grupo y, si un robot sufre una avería o perturbación, la formación de la estructura se mantiene [13]. En este trabajo se plantea una estructura virtual formada por tres robots móviles que forman un triángulo como se observa en la Figura 1.7.

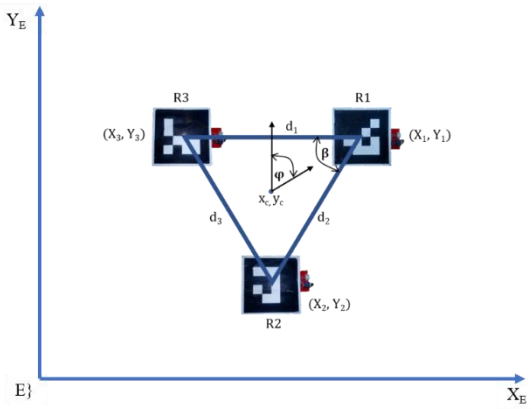


Figura 1.7 Esquema Estructura Virtual

- **Comportamiento Grupal:** En esta estructura se asignan a cada robot varios comportamientos, se realiza una ponderación de todos ellos y se obtiene la acción final de control de cada robot. Esta estructura sirve para evasión de obstáculos, evitar colisiones entre los robots manteniendo la formación como se observa en la Figura 1.8.

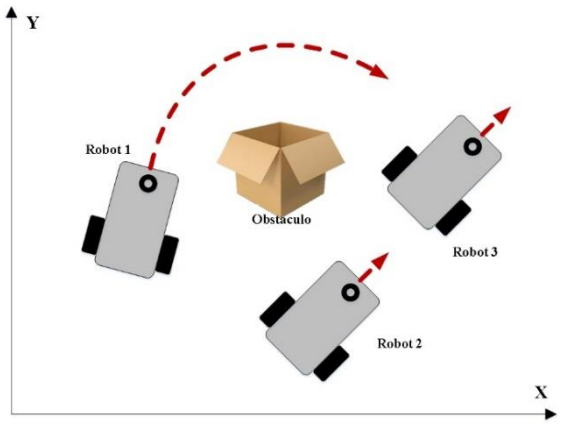


Figura 1.8 Esquema Comportamiento Grupal

- **Líder-Seguidor:** En la Figura 1.9 se puede observar que en esta estructura se designa a uno de los robots como líder y a los demás robots de la flota como seguidores. El comportamiento de los robots seguidores depende del líder el cual tiene una trayectoria propia, y las posiciones deseadas de los seguidores dependen de los estados del líder. Una de las ventajas de esta estructura es que si uno de los robots seguidores sufre una avería o perturbación la formación se mantiene, pero si el líder sufre una perturbación, el cambio se notará en toda la flota. [14]

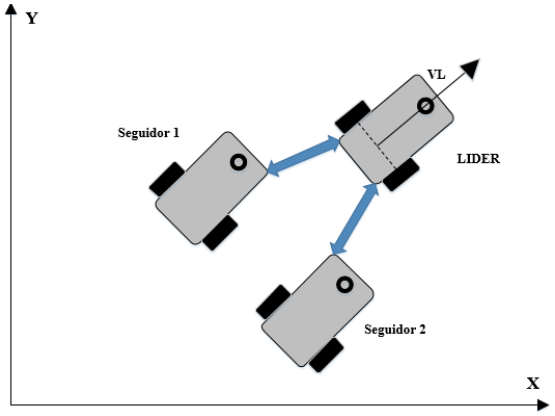


Figura 1.9 Esquema Líder-seguidor

Como ejemplo práctico de la formación de robots, en la Figura 1.10 se puede observar, durante la ceremonia de apertura de los Juegos olímpicos de Invierno PyeongChang 2018, la formación coordinada de 1200 drones de Intel creando complejas figuras en el firmamento controladas por un computador. [15]

Figura 1.10 Formación de 1200 drones Intel, Juegos Olímpicos PyeongChang 2018 [15]

Seguimiento de Trayectoria

El seguimiento de trayectoria busca conseguir que un robot móvil, o por ejemplo el centroide de una formación de robots, siga un camino parametrizado en el tiempo. En el seguimiento de trayectoria no importa la posición inicial del centro de la formación, éste va a intentar alcanzar a llegar a la referencia y seguir la trayectoria hasta su posición final, como se observa en la Figura 1.11. En este proyecto se generan las trayectorias y el seguimiento de estas se lo realiza con la ayuda de un controlador.

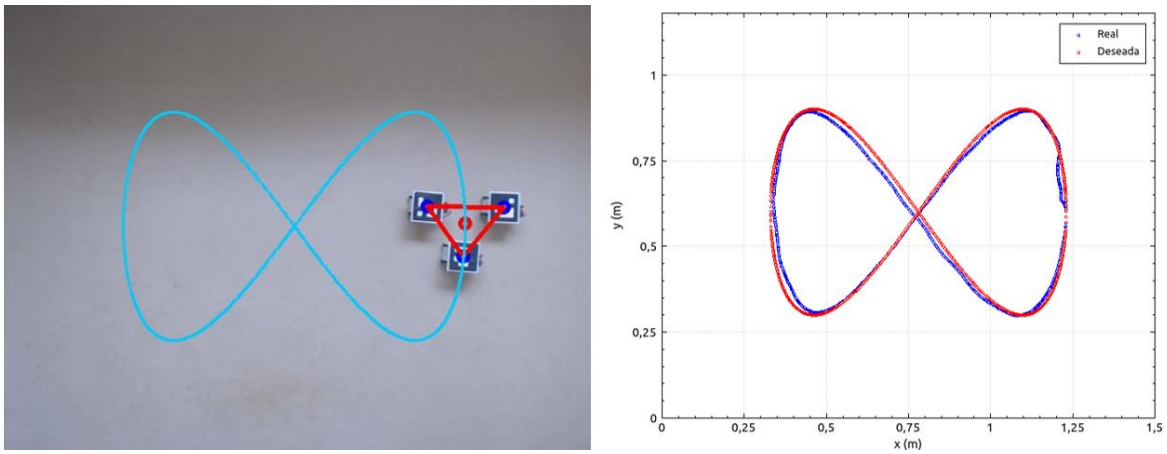


Figura 1.11 Seguimiento de Trayectoria para un grupo de 3 robots móviles

Métodos de Control

Es necesario tener conocimiento de los métodos de control que se utilizarán para que el sistema de robots colaborativos cumpla con el comportamiento deseado; los métodos de control permiten que las variables dentro del sistema tengan el valor deseado.

Control tipo PID [16]

Es uno de los controladores más usados a nivel mundial dentro de muchas industrias debido a que se acopla a todos los sistemas donde se necesita controlar una variable. Se define mediante la Ecuación 1.1.

$$c(t) = Kp\, e(t) + Ki \int e(t)\ dt + Kd\ \frac{\partial e(t)}{\partial t}\ , \tag{1.1}$$

donde:

- Kp . – Constante de proporcionalidad.
- Ki . – Constante de Integración.
- Kd . – Constante de Derivación.
- $e(t)$. – Error en el tiempo.

Esta contiene tres términos los cuales son una parte proporcional, una integral y una derivativa, cada uno de estos términos tienen un efecto en nuestro controlador las cuales se describen a continuación:

Acción Proporcional

Esta acción se aplica de manera proporcional al error, por lo que el error se multiplica con una constante. Esto intenta reducir el error del sistema cuando la acción de control es

grande. El controlador proporcional puede controlar cualquier planta estable, pero bajo ciertos límites y con un error en estado estable.

Acción Integral

Esta acción calcula la integral del error por lo cual se puede decir que esta responde a un error acumulado en el tiempo. Esta acción elimina el error en estado estacionario frente a perturbaciones de carga constante, debido a esto la respuesta se vuelve un poco más lenta conociéndola como acción de controlador lento.

Acción Derivativa

Esta acción de control deriva la señal del error, por lo que, reacciona a que tan rápido cambia la entrada respecto al tiempo, teniendo así un cálculo de la velocidad del error. Esta acción estabiliza una respuesta evitando oscilaciones fuertes lo cual hace que el control sea más rápido, pero teniendo como desventaja la amplificación de señales de ruido.

Control en cascada

La característica principal de este tipo de control es que en su configuración la salida del controlador es también la realimentación para otro controlador, por lo que un sistema en cascada está definido como varios sistemas de control en realimentación. A continuación, se puede observar en la Figura 1.12 la estructura de un control en cascada.

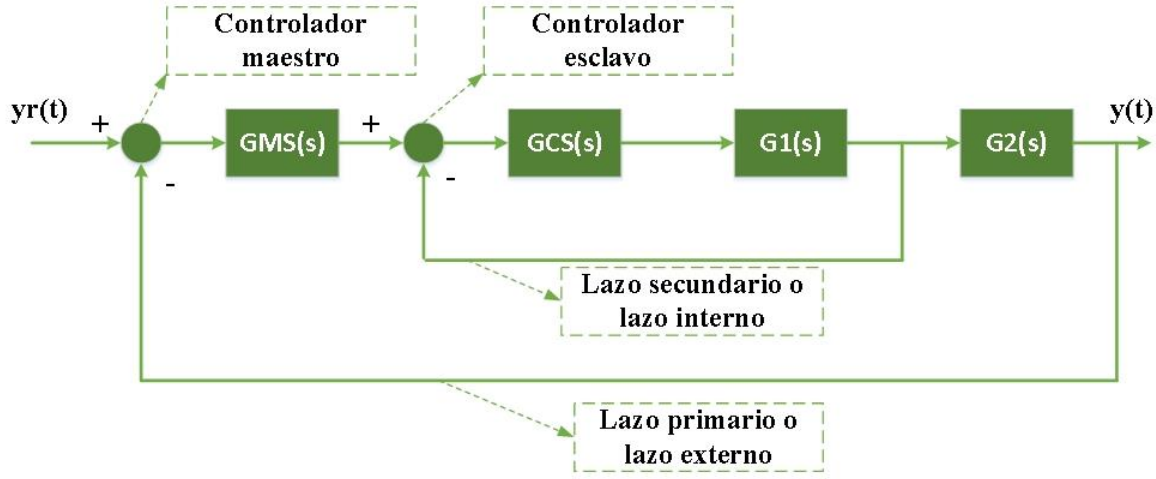


Figura 1.12 Diagrama del control cascada

El objetivo de realizar este tipo de control es que ayuda a eliminar el efecto que producen ciertas perturbaciones haciendo que el sistema sea más estable con una respuesta más rápida, además de mejorar la dinámica del lazo de control.

Una forma de implementar estos sistemas de control es la utilización de algoritmos de programación, los cuales pueden ser desarrollados sobre una plataforma de código abierto.

Lenguaje de Código Abierto

Un lenguaje de código abierto es un término utilizado para todo software que tenga como característica principal que el usuario pueda acceder al código fuente de este. Esto permite al usuario modificar el código dependiendo de sus necesidades.

La principal ventaja de usar software de código abierto es su fácil distribución al no tener que pagar licencias por el mismo, además que esto permite el desarrollo del lenguaje. Adicionalmente, los usuarios tienen la libertad de mejorar y modificar el software. [17]

LINUX

Es un sistema operativo el cual se encontraba en la mayoría de sistema a inicios de la década de 1990. Actualmente se lo puede encontrar en diversas plataformas como computadoras, celulares, drones e incluso automóviles. LINUX tiene una licencia de código abierto, lo cual, entre otras cosas, da libertad de ejecutar programas, estudiarlos, ver cómo funciona, cambiar las líneas de código, distribuir copias de estas versiones, etc. Existen diversas distribuciones de LINUX. Entre las más usadas están:

- LINUX MINT
- DEBIAN
- UBUNTU
- FEDORA

Estas distribuciones se adaptan a los diferentes tipos de usuarios y las necesidades de estos, cada una de estas posee una interfaz propia del escritorio. En general todas las distribuciones de LINUX permiten tener un control de los recursos del sistema [18]. En el presente trabajo se utiliza la distribución UBUNTU debido a que el entorno de trabajo da facilidad al usuario para el desarrollo de diferentes tipos de aplicaciones.

UBUNTU

Es una distribución de LINUX la cual tiene soporte comunitario y profesional. Es popular ya que incorpora un sistema operativo UNIX con una GUI personalizable y es usado en universidades y organizaciones de investigación. UBUNTU contiene diversos paquetes de software que se encuentran bajo la licencia GNU (General Public License). [19]

Sobre UBUNTU se puede instalar otro sistema operativo destinado a la creación de software complejo para el manejo de Sistemas Roboticos, como lo es ROS (Robotic Operating System).

ROS [5]

Este proyecto nace en 2007 con el nombre Switchyard por parte de Morgan Quigley como parte del proyecto de robótica de STANFORD STAIR, aunque el principal colaborador para la creación de ROS es Willow Garage. Por sus siglas en inglés Robotic Operative System, es un sistema operativo para robots, el cual trabaja habitualmente sobre sistemas operativos tipo UNIX. Su funcionalidad radica en tener un framework que permite una fácil escritura sobre el software de un robot ya que posee una gama amplia de librerías y herramientas para la implementación de sistemas robóticos.

Entre sus principales ventajas están:

- **Robustez:** Al ser un sistema robusto, si un robot de la flota sufre un daño otro puede cumplir su función evitando que el sistema se comprometa.
- **Escalabilidad:** Al ser sistemas abiertos se puede ingresar nuevos robots al sistema sin hacer grandes cambios en él.
- **Características de Alta gama:** Dispone sistemas complejos como SLAM (sistema de localización y mapeo) y AMCL (Localizacion Adaptativa Monte Carlo), además de sistemas de path planning, los cuales pueden ser usados directamente sobre los robots.
- **Soporte de sensores y actuadores:** Permite el uso de motores, válvulas, sensores de presión, sensores de presencia, encoders, entre otros.
- **Operatividad entre diversas plataformas:** Es capaz de comunicar diferentes nodos con diferentes tipos de lenguaje de programación entre los cuales puede ser C, C+, java y phyton.
- **Modularidad:** Es una de las principales fortalezas, ya que permite trabajar con nodos de programas de forma separada lo cual lo hace un sistema robusto en caso de que uno de estos nodos falle.
- **Manejo de Recursos:** Al obtener datos de un sensor estos pueden ser usados para diferentes propósitos en diferentes nodos, reduciendo la complejidad computacional.
- **Comunidad Activa:** ROS actualmente tiene una comunidad activa lo cual permite que esta tenga gran soporte por parte de la comunidad y los creadores de ROS.

Estructura de ROS [5]

ROS packages

Es la estructura básica de los programas de ROS, esta contiene los programas a ser ejecutados (nodos), librerías y configuraciones necesarias para ejecutar dentro de ROS. El

paquete dentro de ROS agrupa toda esta información como una unidad como se observa en la Figura 1.13.

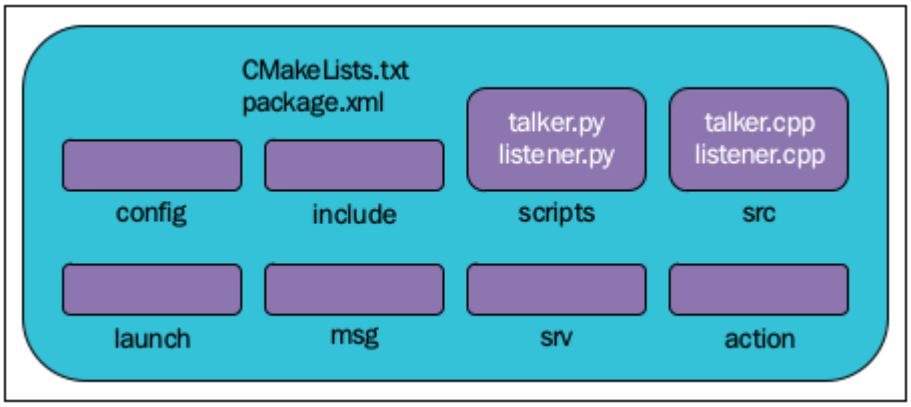


Figura 1.13 Ejemplo de una estructura de un paquete de ROS [5]

ROS messages (msg)

Es el tipo de información utilizada para enviar datos de un proceso a otro, ya que los nodos pueden enviar y recibir información. ROS messages permite describir de manera simple el tipo de formato que se usa en la comunicación. En la Figura 1.14 se observa un ejemplo de esta descripción que se encuentra en la estructura de ROS package, y consta de dos campos, uno para el formato del tipo de dato y otro de la etiqueta que se utiliza dentro de la comunicación entre los nodos.

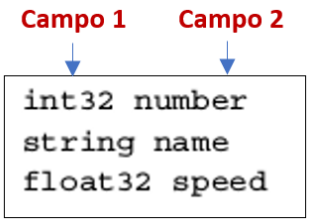


Figura 1.14 Ejemplo de declaración de mensajes de ROS [5]

Cada formato que se usa dentro de la comunicación tiene un tamaño predeterminado el cual se describe en la Tabla 1.1.

Tabla 1.1 Tipos de formato de mensajes de ROS [5]

Tipo	Serialización	C++	Python
bool(1)	unsigned 8-bit int	u int8_t(2)	bool
int8	signed 8-bit int	int8_t	int
uint8	unsigned 8-bit int	uint8_t	int(3)
int16	signed 16-bit int	int16_t	int
uint16	unsigned 16-bit int	uint16_t	int
int32	signed 32-bit int	int32_t	int
uint32	unsigned 32-bit int	uint32_t	int
int64	signed 64-bit int	int64_t	long
uint64	unsigned 64-bit int	uint64_t	long
float32	32-bit ieee float	float	float
float64	64-bit ieee float	double	float
string	ascci string(4)	std::string	string
time	secs/nsecs unsigned 32_bits ints	ros::Time	rospy.Time
duration	secs/nsecs signed 32_bits ints	ros::Duration	rospy.Duration

ROS services (srv)

Permite la comunicación entre nodos con el tipo de comunicación Request/Response, es decir un nodo envía un mensaje y este a su vez espera a que el nodo con el que se comunicó responda.

ROS services dispone de dos campos como ROS messages, pero para el tipo de dato y para la etiqueta, en este se especifica además el formato de datos para Request y para el Response. Un ejemplo de formato de declaración de servicios de ROS se observa en la Figura 1.15.

```
#Request message type
string str
---
#Response message type
string str
```

Figura 1.15 Ejemplo de declaración de servicios de ROS [5]

ROS Nodes

Son los procesos que se ejecutan en base a las librerías de ROS, los nodos son capaces de comunicarse entre sí por medio de mensajes o servicios. Cada nodo tiene una función específica y puede o no abarcar varios procesos del robot a la vez, por lo cual si un nodo falla el sistema aún funciona.

Uno de los objetivos de los nodos ROS es construir procesos simples en lugar de un proceso grande con toda la funcionalidad. Al ser una estructura simple, los nodos ROS también son fáciles de depurar.

ROS Topic

Los mensajes enviados y recibidos entre los nodos utilizan buses de comunicación los cuales se denominan “tópicos”. Cuando un nodo envía un mensaje se llama publicación mientras que cuando un nodo recibe un mensaje a través de un tópico se llama suscripción. Cada “tópico” es único y cada nodo lo puede utilizar para enviar o recibir mensajes a través de este.

ROS Master

Para iniciar ROS es necesario ejecutar primero un máster, ya que toda la comunicación e información se registra en este. El Master es el entorno donde se ejecutan todos los nodos, se tiene la información de los tópicos, mensajes y servicios que se hayan ejecutado. Se lo puede comparar como un servidor DNS.

Se puede ver en la Figura 1.16, como el Master es el entorno en el cual se comunican el nodo que está publicando el mensaje con el que está subscrito.

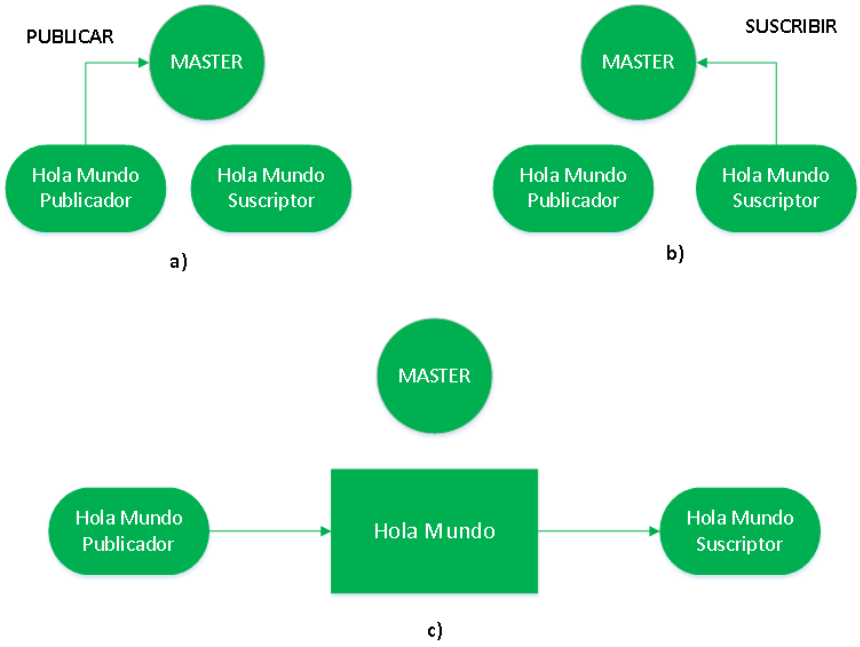


Figura 1.16 Ejemplo de un entorno de trabajo de ROS

Considerando que ROS es un sistema sobre LINUX, de igual manera es necesario manejar un software de código abierto para la creación de una interfaz gráfica que permita al usuario interactuar de forma amigable con las aplicaciones.

Interfaz Gráfica – Qt Creator [20]

Una interfaz gráfica de usuario conocida como GUI (Graphical User Interface), es un software que permite que el usuario y una máquina o sistema automático se comuniquen e interactúen de forma amigable, sin la necesidad de que el usuario disponga de profundos conocimientos de informática.

Para el presente proyecto se utiliza Qt Creator ya que es de código abierto, todas sus herramientas son gratuitas y está permitido usarlas en proyectos comerciales, además corre sobre la plataforma LINUX y su programación es práctica e intuitiva. Qt Creator es un entorno de desarrollo integrado que brinda múltiples herramientas para poder desarrollar aplicaciones e interfaces de usuario una vez y luego desplegarlas en varios sistemas operativos de escritorio, integrados y móviles. Estas herramientas también permiten trabajar desde la creación del proyecto hasta la implementación de dicho proyecto en las plataformas objetivo.

Qt Creator a diferencia de un editor de texto, facilita construir y ejecutar aplicaciones, entiende los lenguajes C++ y QML como código permitiéndole proporcionar funciones útiles, verificación de sintaxis, finalización de código y acciones de refactorización. Adicionalmente, Qt Creator permite crear paquetes de instalación para dispositivos móviles que son adecuados para publicar en tiendas de aplicaciones y otros medios. En la Figura 1.17 se observa el Logo de Qt Creator.

Figura 1.17 Logo de Qt Creator [20]

Qt Designer

Qt Designer es una herramienta de Qt Creator que permite construir interfaces gráficas de usuario con Qt Widgets. Los widgets y formularios creados en Qt Designer se acoplan perfectamente con el código programado utilizando el mecanismo de slots, el cual se utiliza para la comunicación entre objetos en un formulario asignando comportamientos de los elementos gráficos de una forma sencilla.

Luego de revisar los fundamentos teóricos de la plataforma de software sobre el cual se va a implementar el sistema con su controlador, es fundamental disponer un lazo de

realimentación para corregir el error de posición. En el presente trabajo se utiliza la realimentación mediante visión artificial, tema que será detallado en las siguientes secciones.

Visión por Computadora [21]

La visión por computadora es un campo de la inteligencia artificial que permite que las computadoras imiten el sistema de visión humano, obteniendo información necesaria de las imágenes digitales, videos y otras entradas visuales para la toma de decisiones.

El sistema de visión humana puede entrenarse a sí misma para distinguir los objetos, distancias de enfoque, tamaño de los objetos, si están en movimiento o no, entre otros. El objetivo de la visión por computadora es, mediante cámaras, datos y algoritmos en lugar de retinas, nervios ópticos y una corteza visual, entrenar a las máquinas para realizar estas funciones de forma más rápida. En la Figura 1.18 se detalla la similitud entre el funcionamiento del sistema de visión humana y el sistema de visión por computadora.

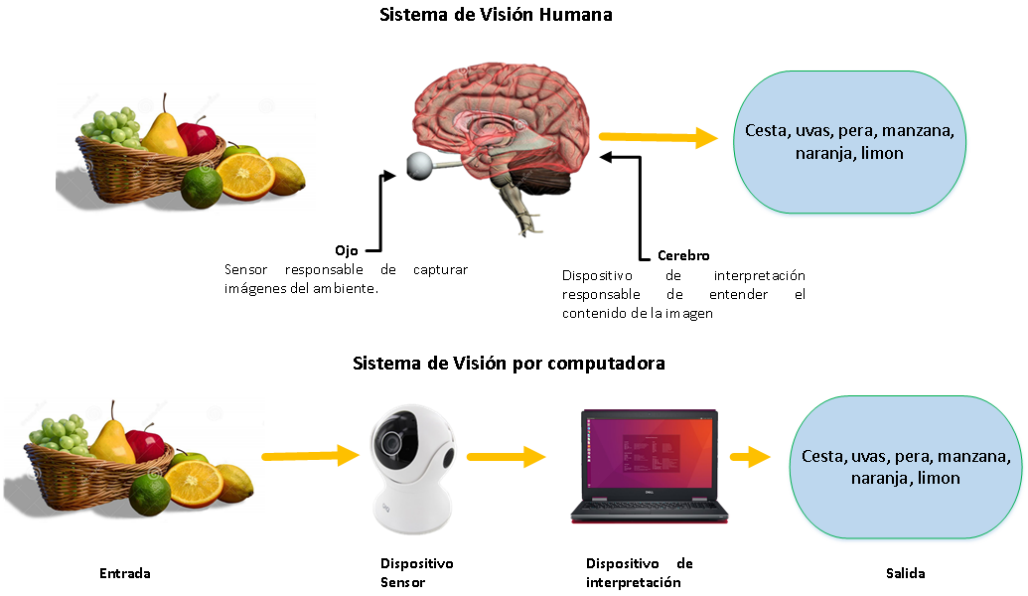


Figura 1.18 Sistema de visión Humana vs Sistema de visión por computadora

Entre las aplicaciones de la visión por computadora se tienen:

- **Clasificación de imágenes:** El sistema de visión artificial es capaz de predecir con precisión que una imagen dada pertenece a una determinada clase.
- **Detección de objetos:** Puede usar la clasificación de imágenes para identificar una determinada clase, luego detectar y tabular su apariencia en una imagen o video. Uno de los ejemplos incluye la detección de daños en líneas de ensamblaje o maquinaria que requiere mantenimiento.

- **Seguimiento de objetos:** El sistema rastrea un objeto una vez que lo detecta, esto se ejecuta, por ejemplo, en alimentaciones de video en tiempo real. Los vehículos autónomos utilizan esta aplicación ya que además de clasificar y detectar los objetos debe rastrearlos y evitar colisiones en las vías.
- **Recuperación de imágenes basada en contenido:** Se puede explorar, buscar y recuperar imágenes de grandes almacenes de datos en función del contenido de las imágenes en lugar de las etiquetas manuales de las imágenes.
- **Estimación de pose:** El sistema de visión artificial permite identificar la posición y orientación de un objeto en relación con su entorno. Esto se puede aplicar en sistemas robóticos donde visualmente se necesita detectar donde están los objetos para evasión de obstáculos.

Para lograr dichas aplicaciones se tienen diferentes técnicas de visión por computadora, como la realidad aumentada, alineación, segmentación, etc. Para un mayor entendimiento sobre estas técnicas se pueden revisar en [3]. Estas técnicas pueden ser desarrolladas en programas como OpenCV con la ayuda de Qt Creator.

OpenCV

OpenCV es un conjunto de librerías y clases desarrolladas por Intel en el año de 1999 que permiten el manejo de visión artificial en diferentes proyectos. Estas librerías están bajo licencia BSD (Berkeley Software Distribution) por lo que son de libre uso comercial. Están escritas en C y C++ con interfaces en C++, Python, Java, MATLAB, entre otros lenguajes; y son compatibles con Linux, Windows, iOs, Android, y OS X, contando con más de 2500 algoritmos para el manejo de visión artificial. Las aplicaciones con estas librerías van desde procesos de control con reconocimiento de objetos, robótica avanzada, sistemas de seguridad, sistemas de realidad aumentada, entre otros. [22]

Aplicación de las librerías OpenCV en Qt Creator y ROS

Qt Creator además de brindar herramientas para el desarrollo de interfaces gráficas, también permite probar diseños en C++ por su capacidad de depuración, verificación de errores, y compilación. Debido a esto se ha vinculado Qt Creator con OpenCV para generar y detectar los marcadores de referencia Matricial colocados en los robots móviles y que posteriormente serán explicados.

Debido a que en el proyecto se va a utilizar el sistema operativo robótico ROS, el tratamiento de la visión artificial también tiene que integrarse a este sistema. El conjunto de librerías de OpenCV se puede integrar al sistema robótico ROS de una manera muy rápida proporcionando herramientas de integración, sin importar la versión de ROS que se

utilice. A continuación, se especifica en que consiste la calibración de la cámara, así como la generación y detección de marcadores, enfocados a la integración con el sistema robótico ROS.

Calibración de cámara [23]

Mediante la calibración de la cámara se pueden obtener sus parámetros externos e internos que dan información acerca del modelo de la misma, la distorsión que introducen en las imágenes que capturan, y la posición y orientación del cuadro de referencia de la cámara con respecto al mundo real.

La matriz del modelo de la cámara está definida por la Ecuación 1.2.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} fx & 0 & cx & 0 \\ 0 & fy & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \left[{}^{R}/_{t}\right] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \tag{1.2}$$

donde:

- $u, v.$ – Coordenadas homogéneas en la imagen de un punto en la escena.

Parámetros internos:

- $fx, fy.$ – Longitudes focales de la lente de la cámara expresadas en pixeles.
- $cx, cy.$ – Posición del centro de proyección en pixeles.

Parámetros externos:

- $X, Y, Z.$ – Coordenadas 3D del espacio en milímetros.
- $\left[{}^{R}/_{t}\right].$ – Matriz de rotación-traslación.

Siendo $\left[{}^{R}/_{t}\right]$ la matriz de Rotación-Traslación que define la posición y la orientación de la cámara con respecto al sistema de coordenadas global, y viene dada por:

$$\left[{}^{R}/_{t}\right] = \begin{bmatrix} r1 & r2 & r3 & tx \\ r4 & r5 & r6 & ty \\ r7 & r8 & r9 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.3}$$

Los parámetros de $r1$ a $r9$ son los factores de la matriz de rotación de la cámara con respecto a cada uno de los ejes y tx, ty y tz son los factores del vector de traslación que representa un cambio en la posición del sistema de coordenadas de la cámara o de un objeto con respecto a ella. Todos son calculados internamente por un nodo en ROS.

Las cámaras debido a su óptica y su geometría introducen distorsión a las imágenes o videos que capturan. Dos distorsiones principales son la radial y la tangencial. En la Figura 1.19, se pueden observar estos tipos de distorsiones.

- **Distorsión radial:** las líneas rectas se visualizan curvadas a medida que se aleja del centro de la imagen.
- **Distorsión tangencial:** se produce cuando la imagen al ser tomada no está alineada de forma paralela al plano de imagen.

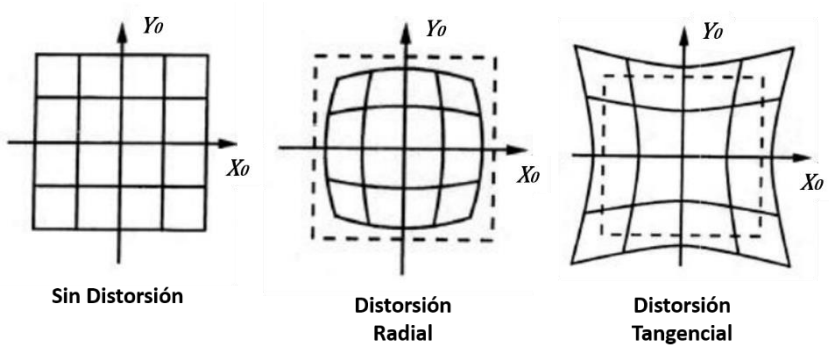


Figura 1.19 Tipos de Distorsión en la imagen.

La matriz de los coeficientes de distorsión que se obtiene al calibrar la cámara está definida por la Ecuación (1.4).

$$coef\ Distorsión = [k1 \quad k2 \quad p1 \quad p2 \quad k3]\,, \tag{1.4}$$

donde:

- $k1, k2, k3.-$ Coeficientes de distorsión radial.
- $p1, p2.-$ Coeficientes de distorsión tangencial.

Para una explicación más detallada de estas matrices se puede revisar la referencia [3], la cual fue tomada como base para este proyecto.

Para la calibración de la cámara a través de OpenCV y ROS se tiene los pasos del método de Zhang 2000 [24], esquematizados en la Figura 1.20.

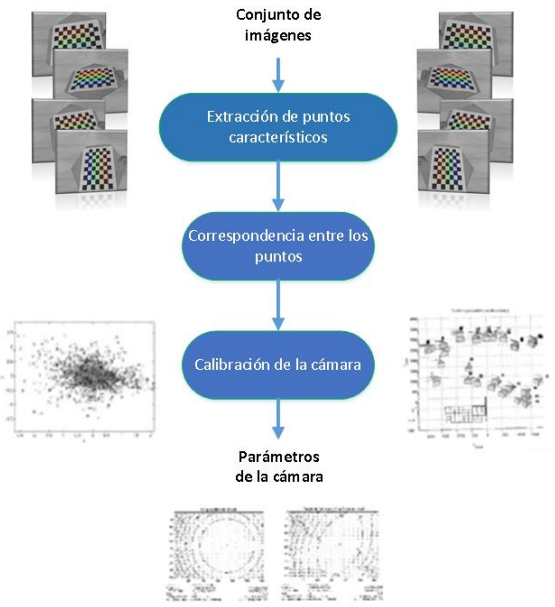


Figura 1.20 Etapas de la calibración de cámaras.

El método de Zhang 2000 consiste en la calibración utilizando patrones en una plantilla plana en 2D. Estos patrones de calibración planos son fáciles de construir y económicos debido a la precisión y exactitud que brindan las impresoras láser al imprimirlas. En este método la cámara y la plantilla pueden ser movidas libremente, se toman varias imágenes detectando el patrón y se hallan las coordenadas de cada esquina basándose en el cálculo de homografía que se realiza de forma transparente por las librerías de Open CV, entre el patrón bidimensional y su proyección en la imagen. En la Figura 1.21 se observa la transformación de homografía que es la transformación de las coordenadas reales a coordenadas de imagen.

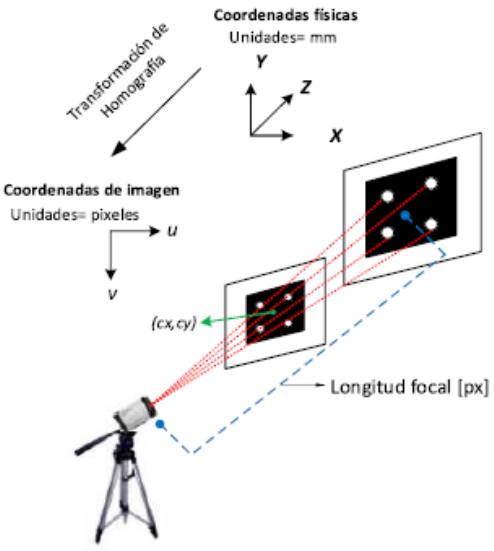


Figura 1.21 Estructura de sistema de coordenadas. [3]

El patrón utilizado se observa en la Figura 1.22, es un tablero de ajedrez de 7x10 cuadros con dimensiones conocidas. Se realiza la calibración con este patrón debido a que es más fácil detectar las uniones del patrón al momento de la calibración.

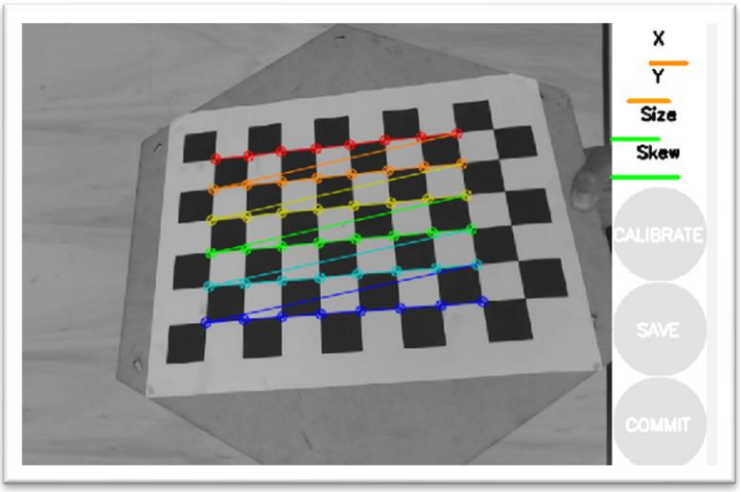


Figura 1.22 Calibración mediante un tablero de ajedrez

Una vez que se calibra la cámara se procede a la generación e identificación de los marcadores de referencia que permiten ubicar a los robots móviles en el entorno de trabajo. A continuación, se explica que son los marcadores de referencia y como se realiza su detección.

Marcadores de Referencia

Los marcadores codificados permiten almacenar distinta información que puede ser leída por un lector laser, una cámara de video, entre otros. Entre las aplicaciones de los marcadores codificados se tienen la identificación de objetos ya sea para el control de inventario y rastreo de un producto, o también en la robótica para analizar la orientación y posición de los objetos en tiempo real.

Estos marcadores, los más básicos, están compuestos por líneas y figuras de color negro con un fondo blanco que, dependiendo de su geometría, como es el caso del espacio entre barras en los códigos de barra, Figura 1.23 (a), o la cantidad de cuadros coloreados en la matriz bidimensional de los códigos QR, Figura 1.23 (b), dan información digitalizada de los productos u objetos. [25]

a) b)

Figura 1.23 (a) Código de Barras[25] (b) Código QR [25]

Para aplicaciones de visión por computadora se han desarrollado diferentes tipos de marcadores [3]. El presente proyecto hace uso de los marcadores de referencia cuadrados con borde, específicamente los marcadores de referencia ArUco para obtener información del entorno de trabajo y de la orientación y posición relativa de los robots móviles.

Marcadores de Referencia matricial ArUco [26]

La librería ArUco que está basado en OpenCV, contiene un numeroso tipo de marcadores cuadrados compuestos por un borde de color negro que facilita la detección rápida de la imagen y en el interior una matriz de codificación binaria que los identifica como se observa en la Figura 1.24. Cada uno de ellos se encuentran agrupados en diccionarios virtuales dependiendo del tamaño de cada marcador que es determinado por su matriz interna.

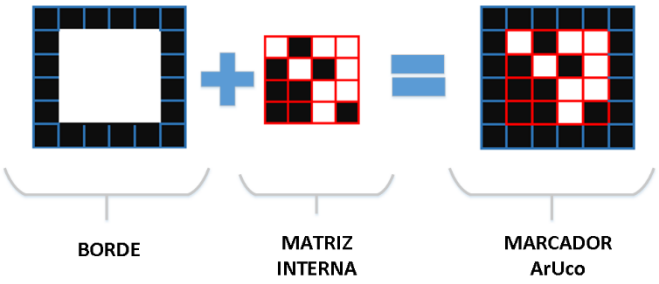


Figura 1.24 Formación Marcador ArUco

La codificación binaria se debe al valor 1 asignado al color blanco y al valor 0 al color negro. Cabe señalar que se puede encontrar un marcador rotado en el entorno, sin embargo, el proceso de detección debe ser capaz de determinar su rotación original, de modo que cada esquina se identifique inequívocamente. Esto también se realiza en base a la codificación binaria.

Generación de marcadores de referencia Aruco [27]

Antes de detectar los marcadores, se deben generarlos, para luego ser impresos y colocarlos en el entorno. Para la generación de marcadores existen diferentes diccionarios virtuales proporcionados por las librerías ArUco de OpenCV. Un diccionario de marcadores es un conjunto de marcadores que contiene la lista de codificaciones binarias de cada uno de sus marcadores.

Las propiedades principales de un diccionario son:

- **Tamaño del diccionario:** es el número de marcadores que componen el diccionario.
- **Tamaño del marcador:** es el tamaño de esos marcadores (el número de bits de la matriz interna).

El módulo ArUco de OpenCV incluye algunos diccionarios predefinidos que cubren una variedad de diferentes tamaños de diccionario y tamaños de marcador. A continuación, en la Tabla 1.2 se muestra algunos diccionarios y sus tamaños.

Tabla 1.2 Diccionarios de Marcadores ArUco

Nombre de Diccionario	Numero de marcadores	Dimensión de matriz interna
DICT_4X4_50	50	4X4
DICT_5X5_100	100	5X5
DICT_6X6_250	250	6X6
DICT_7X7_1000	1000	7X7

Como ejemplo, para la Figura 1.25 se elige uno de los diccionarios predefinidos (DICT_4X4_50). Concretamente, este diccionario está compuesto por 50 marcadores y un tamaño de marcador de 4x4 bits como se observa en la Figura 1.26.

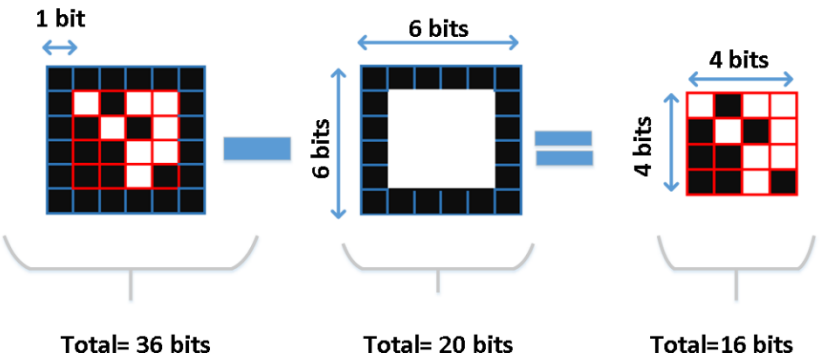


Figura 1.25 Tamaño Marcador ArUco

Se debe tener en cuenta que cada diccionario está compuesto por un número diferente de marcadores. En este caso, los identificadores válidos van de 0 a 49. Cualquier identificación específica fuera del rango válido producirá una excepción.

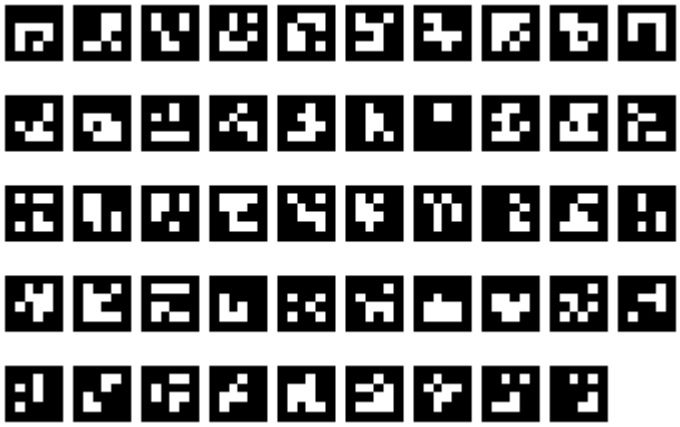


Figura 1.26 Marcadores ArUco generados por el diccionario DICT_4x4_50

Detección de marcadores de referencia Aruco

El proceso de detección debe devolver una lista de marcadores detectados. Cada marcador detectado incluye la posición de sus cuatro esquinas en la imagen (en su orden original), y la identificación del marcador.

En la Figura 1.27 se observa el proceso de detección de marcadores, el cual se explica a continuación.

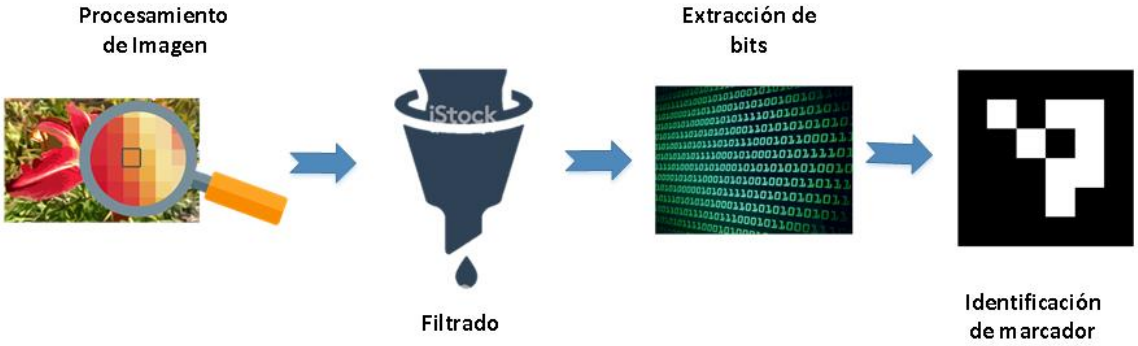


Figura 1.27 Pasos para detección de marcadores

Procesamiento de Imágenes (Detección de marcadores candidatos)

En este paso, se analiza la imagen para encontrar formas cuadradas que sean candidatos para ser marcadores. Uno de los primeros pasos del proceso de detección de marcadores es un umbral adaptativo de la imagen de entrada. Esto se logra convirtiendo la imagen a

una escala de grises y binarizando la imagen. Por ejemplo, en la Figura 1.28 se muestra el proceso para llegar a la imagen umbral para la imagen de muestra.

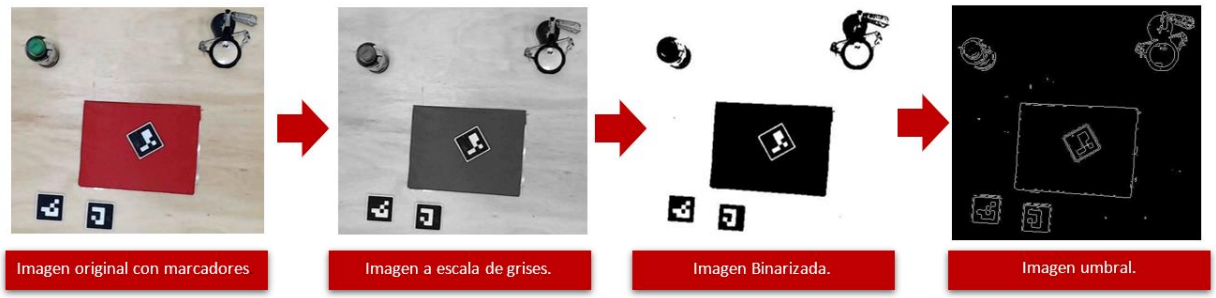


Figura 1.28 Imagen Umbral de una imagen original

Gracias a la binarización de imagen se pueden extraer los contornos de la imagen umbral y se descartan aquellos que no son convexos o que no se aproximan a una forma cuadrada. Si el marcador es demasiado grande o demasiado pequeño el borde no se detectaría reduciendo así el rendimiento.

Filtrado

Luego de detectar los contornos, se aplican ciertos filtros adicionales debido a que no todos los contornos se consideran marcadores candidatos. En este paso se eliminan contornos demasiado pequeños o grandes, se eliminan contornos demasiado cercanos entre sí, etc. Como ejemplo, el resultado del filtrado de la imagen de muestra de la Figura 1.28 se observa en la Figura 1.29. Es mejor descartar contornos incorrectos debido a que al procesar todos los contornos en la siguiente etapa se eleva la carga computacional.

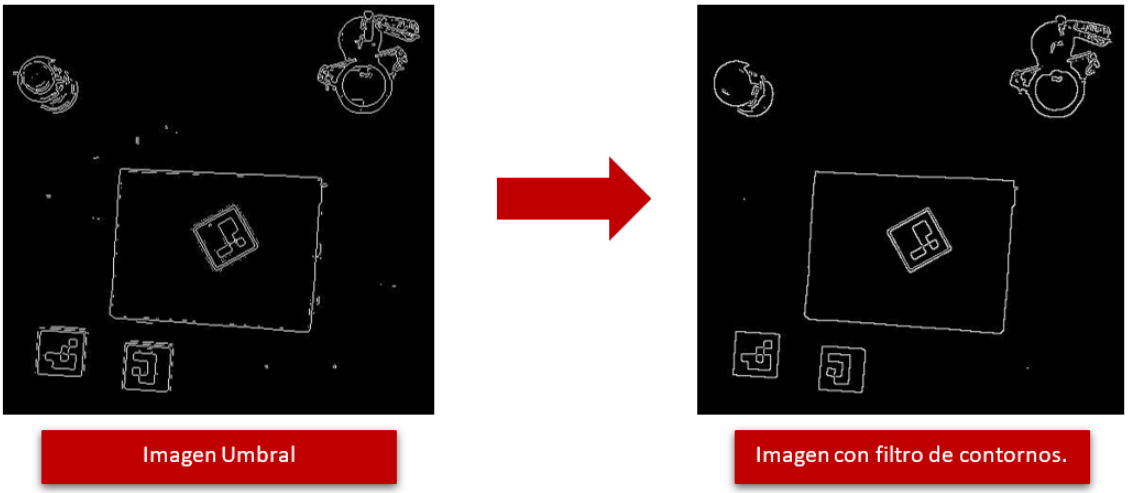


Figura 1.29 Filtrado de la imagen original

Extracción de bits e identificación de marcador

La Figura 1.30 describe los pasos para la extracción de bits de cada marcador. Para hacerlo, primero, se aplica la transformación de perspectiva para obtener el marcador en su forma canónica. Luego, la imagen se divide en una cuadrícula con las mismas celdas que el número de bits en el marcador. En cada celda, se cuenta el número de píxeles en blanco y negro para decidir el bit asignado a la celda.

Finalmente, los bits se analizan para determinar si el marcador pertenece al diccionario específico.

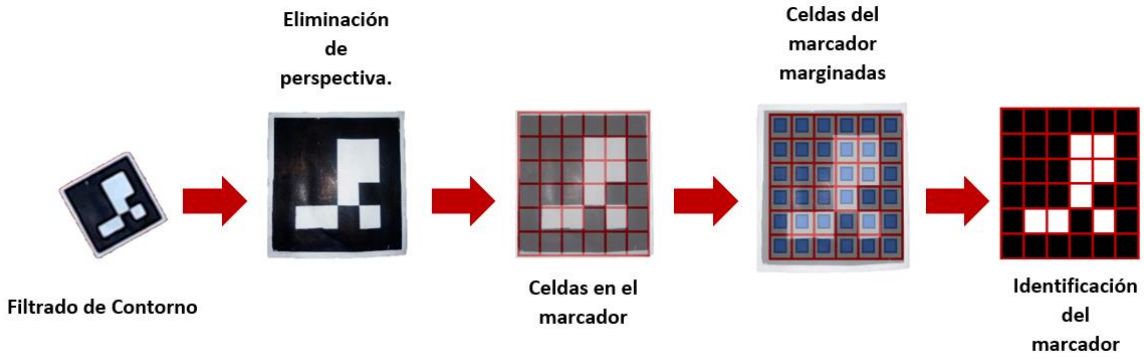


Figura 1.30 Proceso de extracción de bits

Para una información más detallada acerca de los marcadores de referencia ArUco; su generación y detección se recomienda dirigirse al trabajo realizado en [3].

Finalmente, para que el sistema esté totalmente integrado se necesita implementar un sistema de comunicación entre los robots móviles y el ordenador, este sistema es inalámbrico y utiliza módulos Xbee.

Comunicación con módulos Xbee

Xbee es un módulo de radiofrecuencia que permite realizar comunicación inalámbrica entre dispositivos, además de permitir un alto tráfico de datos, con un bajo consumo de energía y baja latencia. Estos utilizan el protocolo de comunicación ZigBee el cual está bajo el estandar IEEE 802.15.4 para crear diferentes topologías de redes. [28]

Tipos de Dispositivos

Coordinador

Es el módulo Xbee encargado de actuar como director de la red ya que este crea la red y posee la información de esta, también permite el enlace con otras redes debido a que hace el papel de router.

Routers

Son los encargados de elegir el camino óptimo entre los dispositivos de la red para la transmisión de paquetes de datos, por lo que estos deben estar conectados a la red.

End-Device

Estos dispositivos son los encargados de enviar o recibir información, los dispositivos End-Device no pueden enviar información directamente entre ellos, siempre tienen que enviarla a través del Router o del Coordinador.

Topología de Redes

Existen algunas topologías para la red inalámbrica con módulos Xbee, entre las principales se encuentra Punto-Punto, Punto- Multipunto o Estrella, Árbol y Malla.

Punto-Punto

Esta topología funciona entre dos dispositivos Xbee, como se puede ver en la Figura 1.31.

Figura 1.31 Topología Punto a Punto

Punto-Multipunto (Estrella)

Esta topología como se observa en la Figura 1.32, tiene un módulo Xbee como Coordinador y varios módulos Xbee como “End Devices”. Tiene como característica principal el ser una topología centralizada ya que cada dispositivo “End Device” si desea comunicarse con otro debe comunicarse primero con el Coordinador el cual dará la información al dispositivo.

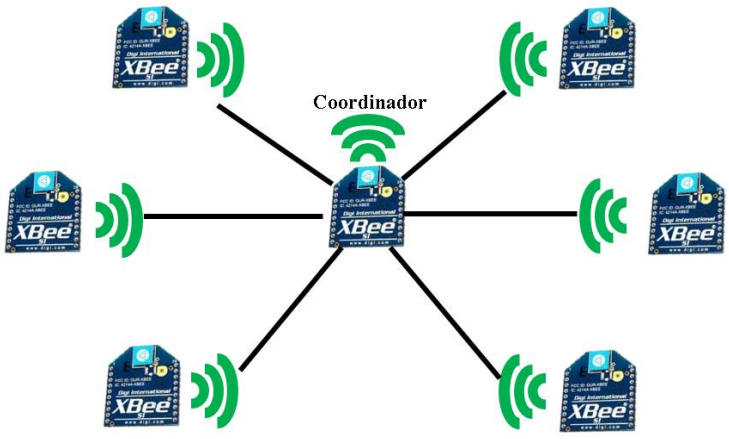


Figura 1.32 Topología Estrella

Árbol

Como se observa en la Figura 1.33, la topología árbol es una mezcla entre las anteriores topologías, ya que el coordinador de la red se comunica con los Routers mediante una topología Punto-Punto mientras que los Router son los encargados de las ramificaciones de los “End Device”, los cuales no participan en el enrutamiento de los mensajes.

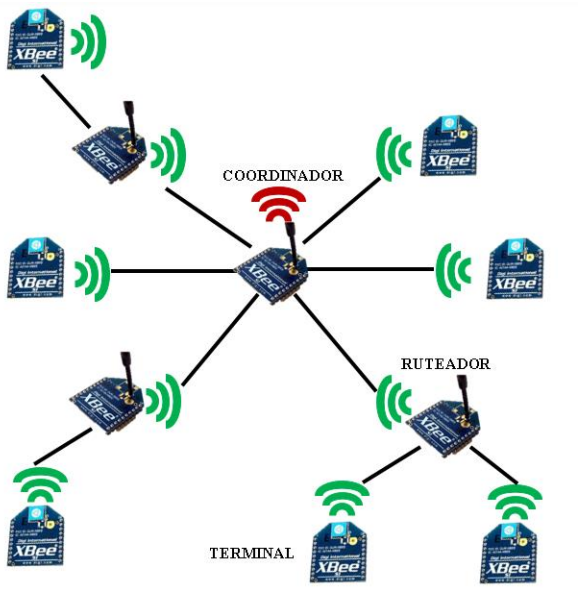


Figura 1.33 Topología Árbol

Malla

La topología malla, Figura 1.34, tiene como característica principal que los routers puede comunicarse entre ellos sin la necesidad de comunicarse a través del Coordinador lo cual lo hace confiable y aumenta el rendimiento ya que pueden existir múltiples trayectorias.

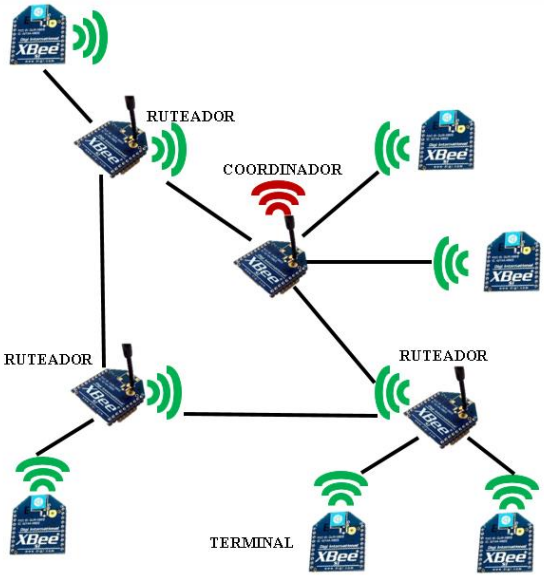


Figura 1.34 Topología Malla

Modos de Operación [29]

Los modulos Xbee pueden ser configurados de acuerdo con las aplicaciones que tendrán estos, por lo cual a continuación se detallan los modos de operación que existen.

Modo Transparente (AT)

El modo de operación Transparente o AT tiene como característica principal que cualquier dato se envía inmediatamente al módulo remoto que contenga la dirección de destino en su memoria. En este no es necesario formación de paquetes ya que se envían como comunicación serial, por lo que la dirección y el destino se mantienen constantes, como se observa en la Figura 1.35. Es la mejor opción cuando se realiza una comunicación entre solo dos módulos Xbee, por lo que es el modo de operación por default.

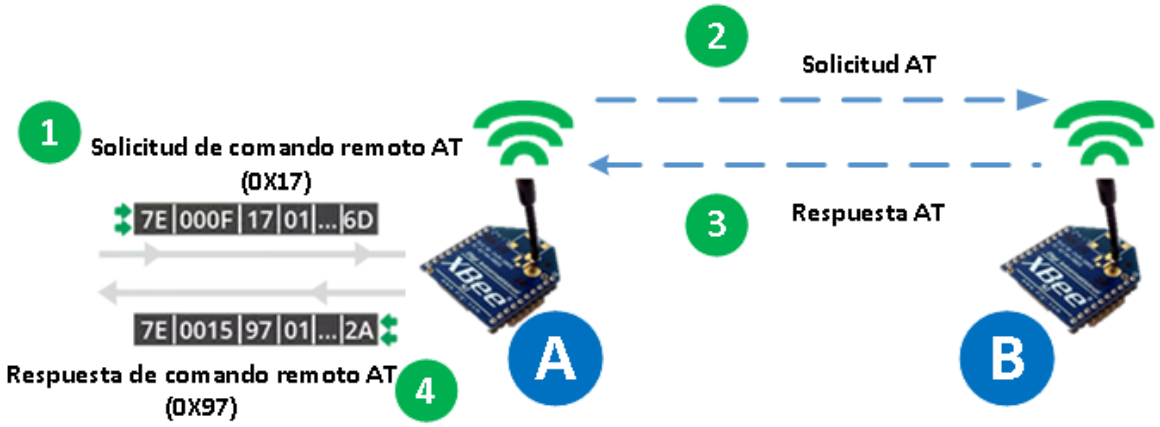


Figura 1.35 Modo Transparente

Modo API

El modo de operación API (Application Programming Interface) tiene como característica que los datos que se envían o reciban en el módulo se encuentra en tramas. Estas tramas contienen el direccionamiento y la carga de información útil. Además, dispone de varias ventajas, como, por ejemplo: permite la transmisión a múltiples destinatarios sin tener que ingresar al modo Command, facilita la verificación de que los paquetes fueron entregados y permite identificar la dirección de origen de cada paquete recibido.

En el modo de operación API las tramas tienen el formato dado en la Figura 1.36.

Figura 1.36 Trama Modo API

Modo API con escape (escaped operating mode)

Este modo de operación es similar al Modo API normal, ya que de igual forma manda los datos en tramas, la diferencia se encuentra en que el modo API normal o sin escape tiene un delimitador de inicio y la longitud de bytes para diferenciar cada trama API.

El modo de operación de escape API se caracteriza por tener un carácter de escape, lo cual está recomendado para sistemas donde pueda existir ruido o interferencia ya que esto mejora la confiabilidad, ya que el carácter de escape permite que los delimitadores que existen entre cada espacio de la trama no sean mal interpretados como los datos recibidos o enviados entre cada Xbee

API con escape garantiza que todos los bytes 0x7E sean delimitador de inicio de una trama API, y que esta no pertenezca a otro campo de la trama como lenght, data o el checksum.

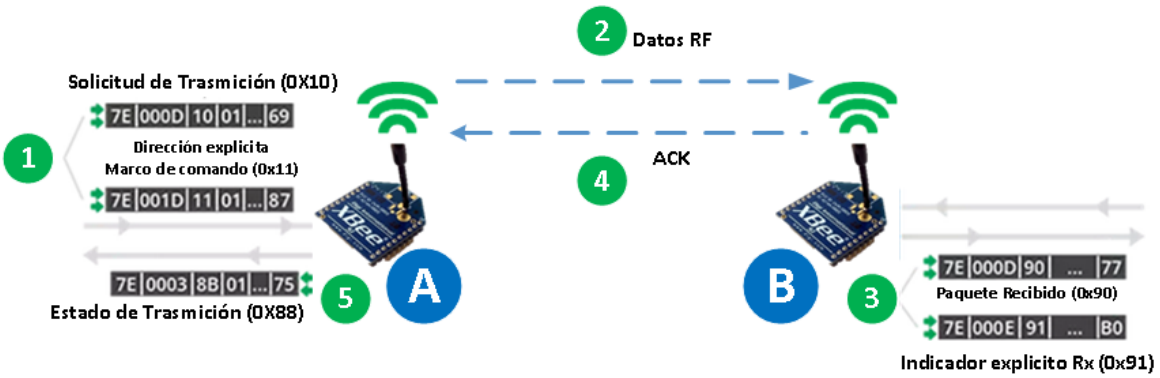


Figura 1.37 Modo API con escape

En este trabajo se va a emplear comunicación modo API con escape, debido a que es un protocolo más seguro y disminuye la perdida de información al garantizar que cada trama sea enviada y recibida en orden.

2. METODOLOGÍA

El presente capítulo se compone de cuatro apartados, el primero presenta los componentes físicos y la implementación de los robots móviles, el segundo y tercero se enfocan en el diseño de los controladores y finalmente el cuarto explica la implementación de todo el sistema.

2.1 ROBOTS MÓVILES DE TAMAÑO REDUCIDO

Con el fin de cumplir los objetivos planteados para este proyecto es necesario tener en cuenta que, para su realización, se utilizó como base los robots móviles diseñados en [3], los cuales se caracterizan por ser de tamaño reducido al poseer un volumen de 280 cm^3 y una masa de 129 gramos.

Estos robots constan de los elementos descritos a continuación.

Elementos electrónicos

Los robots móviles del banco de pruebas tomado como base disponen de los siguientes componentes electrónicos.

- **Mini motores reductores:** Son los actuadores del sistema ya que permiten el movimiento de los robots. Estos operan a un bajo voltaje, pero con un alto torque, además disponen de un eje de salida que tiene forma de D con una longitud de 9.7 mm y un diámetro de 3 mm para la colocación de una rueda. Se los puede observar en la parte (a) de la Figura 2.1.

- **Módulo DRV8835:** Se lo utiliza para manejar el par de motores de cada robot móvil, ya que tiene dos salidas como se observa en la parte (b) de la Figura 2.1. Internamente disponen de una configuración de puente H para cada una de estas salidas. El modo de operación es simple al tener dos pines para el control de la dirección del motor, los diferentes modos de operación se los puede revisar en detalle en [30].

- **Xbee S1:** Es un módulo de comunicación por radiofrecuencia. Esta serie Xbee se caracteriza por ser la más simple y fácil de usar permitiendo realizar redes de comunicación con topologías tipo punto-punto o estrella, siempre y cuando la red tenga la misma serie de tarjeta Xbee [28]. En la parte (c) de la Figura 2.1 se puede observar el módulo Xbee S1.

- **Fio V3-Atmega32U4:** Es una tarjeta embebida Arduino lanzada por SparkFun Electronics, tiene como característica principal la opción de conectar un módulo

Xbee y una batería Lipo en la misma tarjeta como se observa en la parte (d) de la Figura 2.1.

- **Batería de Litio:** La fuente de alimentación para cada uno de los robots del banco de pruebas es una batería de litio, la cual se acopla directamente en la tarjeta Fio V3 como se observa en la parte (e) de la Figura 2.1. La batería tiene capacidad de 500mA y trabaja a 1.9 Wh.

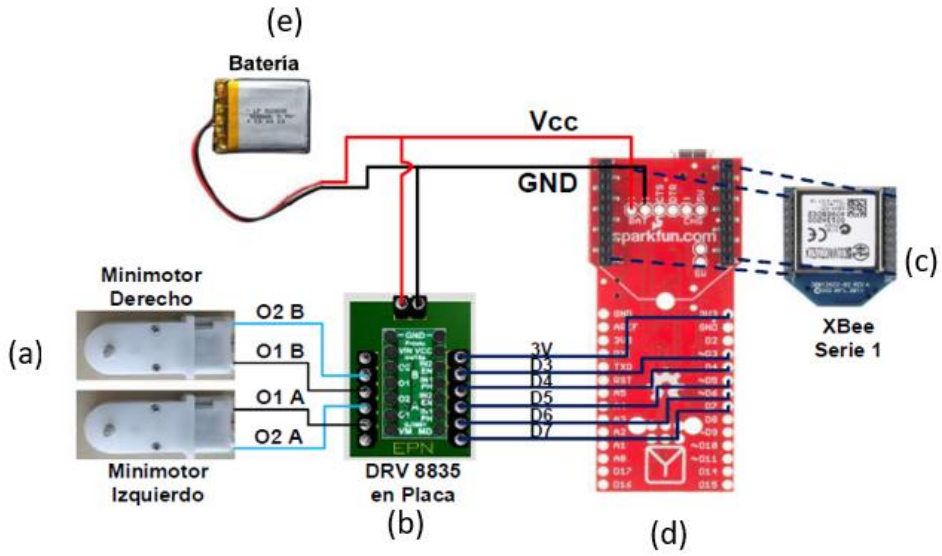


Figura 2.1 (a) Mini motores reductores, (b) DRV8835, (c) Xbee, (d) Fio V3 Atmega32U4, (e) Batería de Litio

Componentes Mecánicos

La estructura mecánica de cada uno de los robots móviles del banco de pruebas está conformada de las siguientes partes:

- **Ruedas**: Los robots móviles poseen dos ruedas marca Pololu las cuales se observan en la parte (a) de la Figura 2.2. Estas pueden acoplarse al eje de salida de los mini motores reductores que tiene forma de D con un diámetro de 3 mm. Las ruedas tienen un diámetro de 32 mm y un ancho de 6.5 mm.
- **Rueda Loca:** Cada robot posee una rueda loca como se observa en la parte (b) de la Figura 2.2, esta permite al robot tener una mayor estabilidad ya que se la acopla en la parte delantera del robot, tiene una dimensión de 3/8 de pulgada.
- **Estructura de polímero:** Para el acoplamiento de todas las partes que conforman el robot móvil se tiene como base una estructura de polímero la cual se la puede observar en la parte (c) de la Figura 2.2. Esta estructura se encuentra conformada por dos partes: en la primera están colocados los componentes electrónicos, las ruedas y la bola loca, y en la segunda se coloca sobre esta el marcador de

referencia. Estas dos partes pueden unirse o separarse fácilmente mediante un pedazo de velcro.

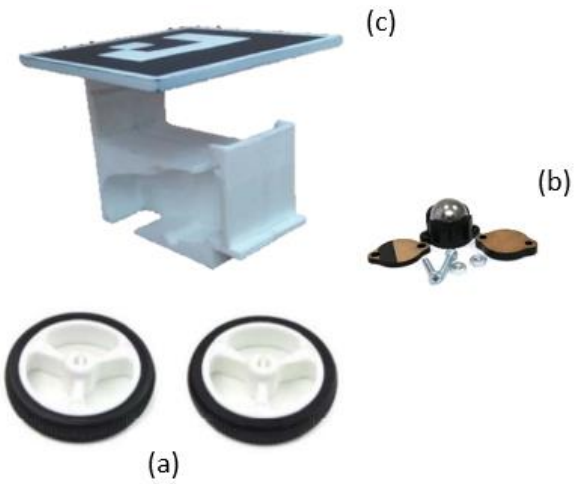


Figura 2.2 (a) Ruedas (b) Rueda Loca (c) Estructura de polímero

Actualización de los robots móviles del banco de pruebas

Para la realización del proyecto fue necesario realizar una actualización a los robots móviles del anterior banco de pruebas. Estas actualizaciones realizadas comprenden principalmente cambios en los componentes electrónicos y serán detallados a continuación.

Mini motor reductor

Los mini motores reductores que utilizaban los robots móviles tenían como desventaja, que, debido a que el modelo empleado no dispone de un eje de motor extendido, no se puede colocar sensores de velocidad. Por tal motivo se cambiaron los motores por una versión que tenga eje de extensión como se observa en la Figura 2.3.

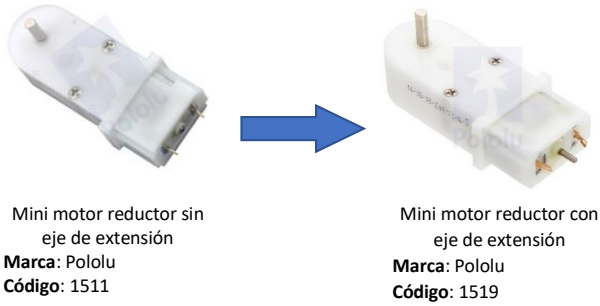


Figura 2.3 Cambio de modelo de mini motores reductores marca Pololu.

Los datos técnicos de estos motores se encuentran en la Tabla 2.1, en ella se observa que las características técnicas de los motores se conservan con la única diferencia que tienen un eje de motor extendido.

Tabla 2.1 Características de Motores Pololu [31]

Mini motor reductor	1511	1519
Voltaje de Operación	4.5 [v]	4.5 [v]
RPM	150 [rpm]	150 [rpm]
Relación	120:1	120:1
Corriente	130 [mA]	130 [mA]
Eje de Motor Extendido	-	3 mm

Encoder Magnético Pololu

Para la medición de velocidad se utilizan encoders magnéticos de cuadratura marca Pololu, los cuales están diseñados para ser colocados en el eje del mini motor reductor. El Encoder de cuadratura dispone de 6 pines, los cuales se observan en la parte (a) de la Figura 2.4.

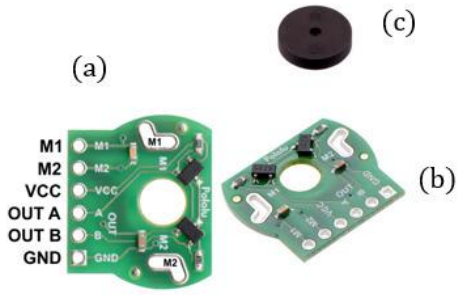


Figura 2.4 (a) Pines Encoder Marca Pololu, (b) Encoder Marca Pololu, (c) imán circular.

Estos pines tienen las siguientes funciones:

M1: Primer terminal del motor.
M2: Segundo terminal del motor.
VCC: Voltaje de alimentación del módulo Encoder Pololu.
Out A: Salida canal A de Encoder de cuadratura.
Out B: Salida canal B de Encoder de cuadratura.
GND: Tierra del módulo Encoder Pololu.

Adicionalmente al módulo del Encoder viene por separado un imán circular que tiene un agujero en el centro para acoplarlo en el eje de los mini motores reductores como se observa en la parte (b) de la Figura 2.4. En la Figura 2.5 se puede observar el módulo del Encoder de cuadratura acoplado a los mini motores reductores.

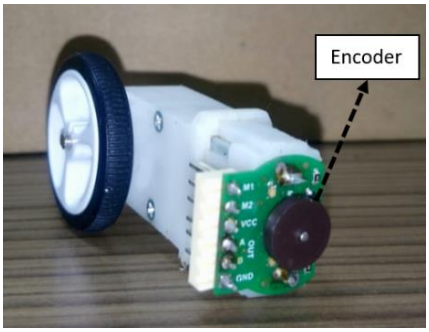


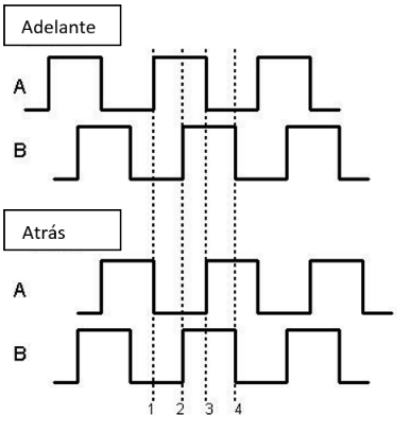
Figura 2.5 Acoplamiento de Encoder Pololu en mini motor reductor.

Los datos técnicos de los Encoders se encuentran resumidos en la Tabla 2.2 donde se observan los niveles de voltaje de salida del sensor en sus canales, así como el voltaje de alimentación.

Tabla 2.2 Características de Encoder Magnético Pololu [32].

Rango de voltaje de operación	2.7 a 18[v]
Canal A	2.7 a 18 [v]
Canal B	2.7 a 18 [v]
Pulsos por Revolución	12

Las señales resultantes que se obtienen en el canal A y canal B son dos ondas cuadradas independientes de igual periodo y frecuencia, las cuales se encuentran desfasadas 90º una con respecto a la otra, como se observa en la Figura 2.6. Esto permite determinar no solo la velocidad a la que están girando los mini motores, sino que además indican si se ha cambiado el sentido de giro. En la lectura de los canales A y B se tiene 4 estados lógicos cuyo orden se repite, de tal manera que se puede saber en qué dirección está girando el mini motor al ver la secuencia de estos estados.



Estado	A	B
1	1	0
2	1	1
3	0	1
4	0	0

Figura 2.6 Señales de salida de Encoder de cuadratura.

Al disponer de estas nuevas señales en los robots móviles se debieron modificar las conexiones del circuito, por lo que fue necesario diseñar y crear una PCB nueva.

PCB tipo shield

Los robots móviles disponían previamente de una PCB para la conexión entre el Arduino Fio V3-Atmega32U4, el driver DRV8835, y los mini motores Pololu. Esta PCB tuvo que ser reemplazada debido a que se añadieron las señales de salida de los Encoders.

La nueva PCB fue diseñada para que sea tipo shield, lo cual permite una fácil conexión y desconexión de la PCB con el Arduino. Su principal ventaja es poder cambiar fácilmente algún componente en caso de existir alguna falla. Esta PCB fue diseñada mediante el software ARES de Proteus. Como resultado final se tiene la PCB que se observa en la Figura 2.7.

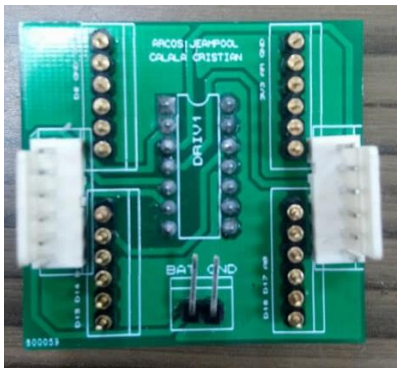


Figura 2.7 PCB tipo shield.

Conectores

Anteriormente los robots móviles tenían sus conexiones mediante cables que estaban soldados directamente de la PCB a la tarjeta Fio y a los motores. Esto tenía una gran desventaja ya que para realizar el reemplazo de algún componente era necesario desoldar todas las partes. Por tal motivo se decidió cambiar la forma de conexión mediante el uso de buses de datos para mayor facilidad al conectar y desconectar, como se observa en la Figura 2.8.

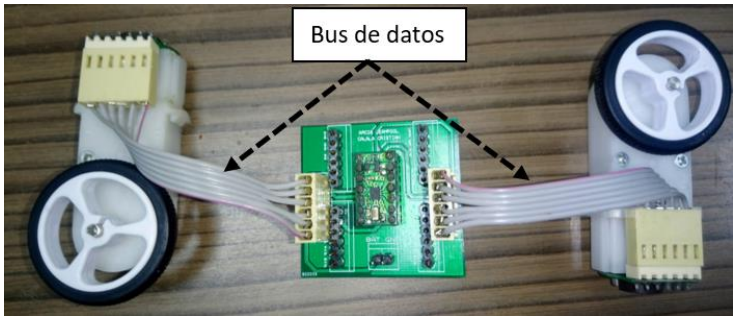


Figura 2.8 Bus de datos conectados a tarjeta.

Con estos cambios se obtiene robots móviles modulares y fáciles de montar en la estructura mecánica. En la Figura 2.9 se observa el robot con todos los cambios realizados.

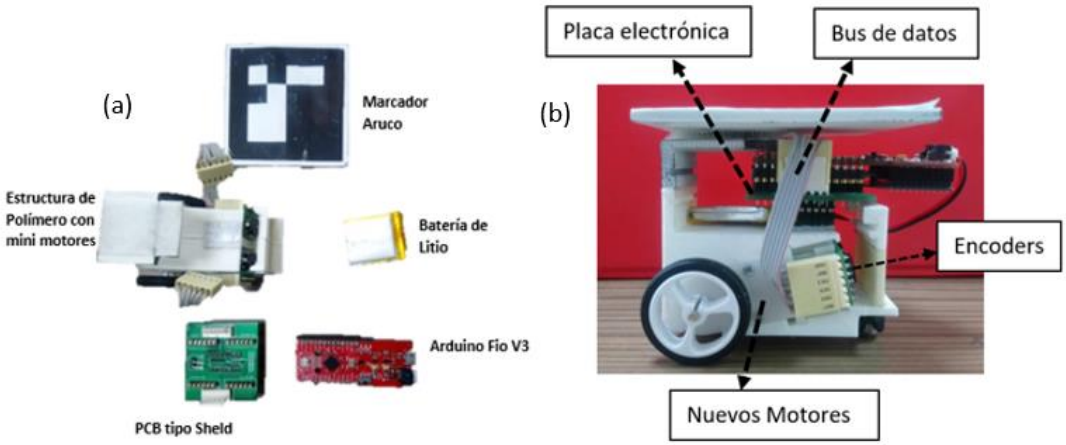


Figura 2.9 (a) Robot desacoplado, (b) Robot actualizado.

Debido a que se añadieron 4 nuevas señales de entrada provenientes de los Encoders ubicados en los dos minimotores, se realizaron cambios en la conexión con los pines del Arduino. En las Figuras 2.10 y 2.11 se esquematizan los diagramas de conexión entre la tarjeta Arduino, el PCB, el enconder, el driver DRV8835, y los mini motores reductores.

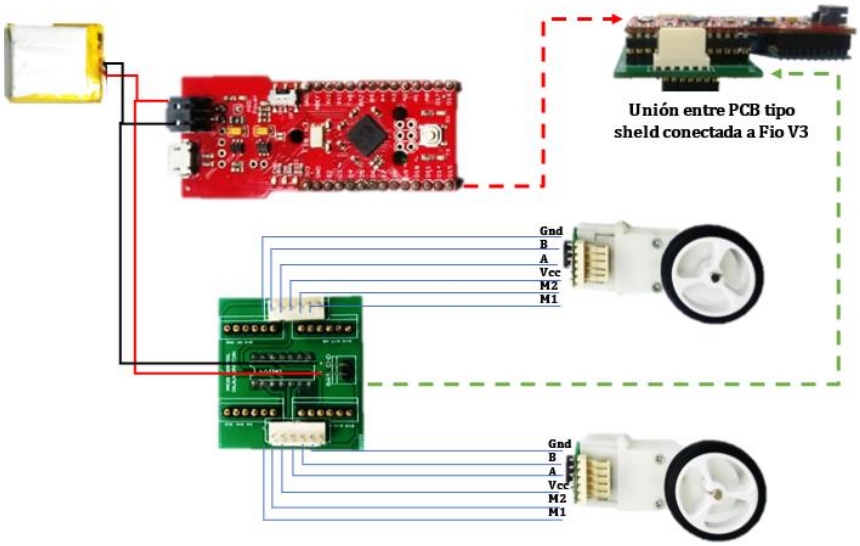


Figura 2.10 Diagrama de conexión de los componentes electrónicos.

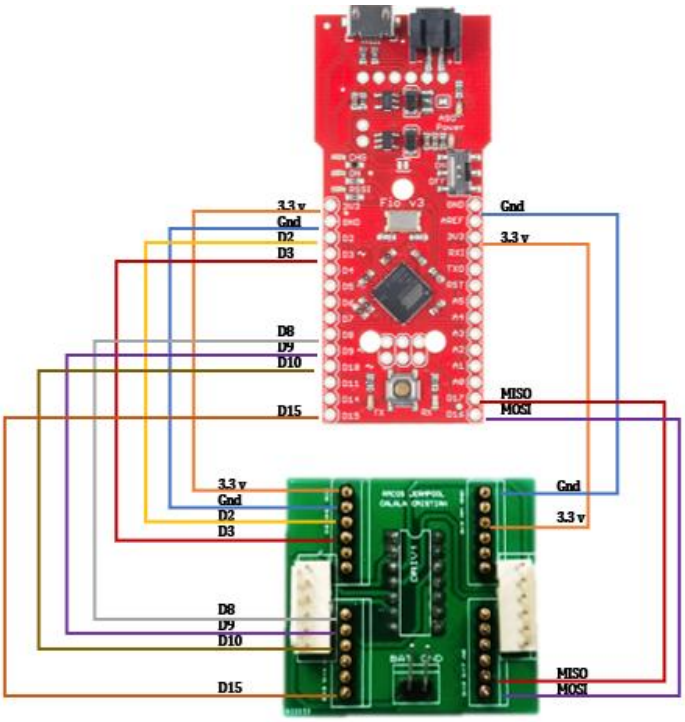


Figura 2.11 Diagrama de pines de entrada y salida entre la tarjeta Fio V3 y PCB.

2.2 MODELAMIENTO DE LA FORMACIÓN DE LOS ROBOTS MÓVILES

Los robots diseñados y construidos en este trabajo tienen cierta similitud constructiva a los robots comerciales Pioneer 3-DX, estos robots tienen dos ruedas fijas perpendiculares en un mismo eje las cuales controlan sus giros dependiendo de la velocidad de cada rueda, y una rueda loca para brindar mayor estabilidad. Las diferencias de las plataformas móviles implementadas en este trabajo con estos robots comerciales son su tamaño reducido y su peso que puede asumirse como despreciable. Por lo tanto, es posible no considerar las características dinámicas del robot y tomar en cuenta únicamente su modelo cinemático.

Modelo Cinemático [33]

Es importante considerar el modelo cinemático de un robot móvil diferencial, ya que contiene restricciones holonómicas que deben ser consideradas. Una restricción holonómica ocurre cuando el robot tiene una restricción en su movimiento, en el caso del robot diferencial no puede realizar movimientos laterales directos, solo movimientos hacia adelante y atrás. En el modelo cinemático se considera la posición del robot que se define por el punto P, con coordenadas (x, y), tomado en el centro de gravedad del robot con respecto al sistema de referencia global E. Este punto está ubicado a una distancia $\boldsymbol{a}$ del centro de la línea base de las ruedas, como se ve en la Figura 2.12.

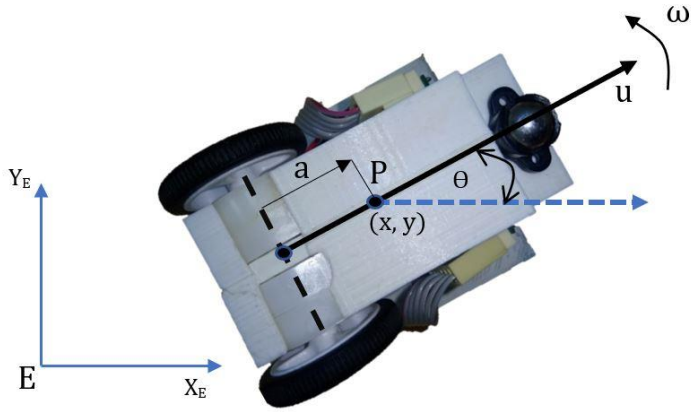


Figura 2.12 Representación del modelo cinemático del robot móvil

Las ecuaciones que definen este modelo cinemático son:

$$\dot{x} = u\,cos\theta - a\omega\,sen\theta, \tag{2.1}$$

$$\dot{y} = u\,sen\theta + a\omega\,cos\theta, \tag{2.2}$$

$$\dot{\theta} = \omega, \tag{2.3}$$

donde:

- $\dot{x}, \dot{y}.$ – Primera derivada de la posición del punto P (centro de gravedad del robot) con respecto al sistema de referencia global E.
- $\dot{\theta}.$ – Primera derivada del ángulo de orientación del robot con respecto al eje x.
- $a.$ – Desplazamiento del centro del eje de las ruedas al punto P.
- $u.$ – Velocidad lineal del robot móvil.
- $\omega.$ – Velocidad angular del robot móvil.

A este sistema de ecuaciones se lo representa en forma matricial de la siguiente manera:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos\theta & -a\ sen\theta \\ sen\theta & a\ cos\theta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ \omega \end{bmatrix} \tag{2.4}$$

Modelo del grupo de robots móviles

Al trabajar con el control de la formación de robots móviles se deben incluir las ecuaciones del modelo cinemático de cada uno de ellos. Generalmente dentro del modelo cinemático no se considera la Ecuación (2.3) ya que se asume que la orientación del robot viene definida por la trayectoria, por lo que la Ecuación (2.4) queda reducida de la siguiente manera:

$$\begin{bmatrix} \dot{x}_i \\ \dot{y}_i \end{bmatrix} = \begin{bmatrix} cos\theta_i & -a\ sen\theta_i \\ sen\theta_i & a\ cos\theta_i \end{bmatrix} \begin{bmatrix} u_i \\ \omega_i \end{bmatrix}, \tag{2.5}$$

El subíndice i se refiere al i –ésimo robot en el sistema. En este trabajo se tiene el control de una formación de un grupo de tres robots móviles así que el modelo cinemático del sistema queda definido como:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{y}_1 \\ \dot{x}_2 \\ \dot{y}_2 \\ \dot{x}_3 \\ \dot{y}_3 \end{bmatrix} = \begin{bmatrix} cos\theta_1 & -a\ sen\theta_1 & 0 & 0 & 0 & 0 \\ sen\theta_1 & a\ cos\theta_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & cos\theta_2 & -a\ sen\theta_2 & 0 & 0 \\ 0 & 0 & sen\theta_2 & a\ cos\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & cos\theta_3 & -a\ sen\theta_3 \\ 0 & 0 & 0 & 0 & sen\theta_3 & a\ cos\theta_3 \end{bmatrix} \begin{bmatrix} u_1 \\ \omega_1 \\ u_2 \\ \omega_2 \\ u_3 \\ \omega_3 \end{bmatrix} \tag{2.6}$$

De forma general la Ecuación (2.6) se la puede representar de la siguiente manera:

$$\dot{X} = P\ V\ , \tag{2.7}$$

donde: $\dot{X} = [\dot{x}_1\ \dot{y}_1 \dots \dot{x}_3\ \dot{y}_3]^T$; $P = \begin{bmatrix} cos\theta_1 & -a\ sen\theta_1 & \dots & 0 & 0 \\ sen\theta_1 & a\ cos\theta_1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & cos\theta_3 & -a\ sen\theta_3 \\ 0 & 0 & \dots & sen\theta_3 & a\ cos\theta_3 \end{bmatrix}$ y

$V = [u_1\ \omega_1 \dots u_3\ \omega_3]^T$.

Formación y Postura del grupo de robots móviles

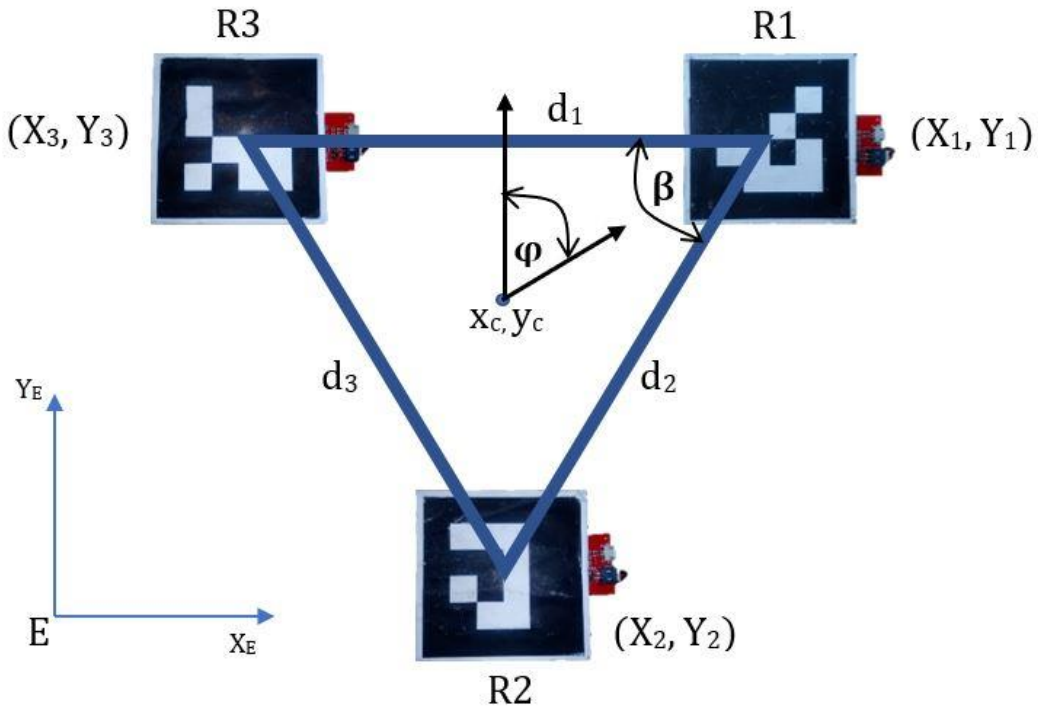


Figura 2.13 Formación y postura de un grupo de robots móviles

En este trabajo se plantea una estructura virtual formada por tres robots móviles donde a cada robot se lo toma como un vértice (x_1, y_1), (x_2, y_2) y (x_3, y_3), formando un triángulo en el sistema de referencia global E, como se observa en la Figura 2.13. Como se conoce un triángulo puede ser definido por la distancia de dos de sus lados d_1 y d_2 y el ángulo interior entre ellos β, así que se toma estos parámetros como variables para el vector de formación, Ecuación (2.8).

$$Q_F = \begin{bmatrix} d_1 \\ d_2 \\ \beta \end{bmatrix} = \begin{bmatrix} \sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2} \\ \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \\ \arccos\left(\frac{{d_1}^2 + {d_2}^2 - {d_3}^2}{2\, d_1\, d_2}\right) \end{bmatrix} \tag{2.8}$$

Para el vector de postura, Ecuación (2.9), se utiliza como parámetros Xc y Yc que son las coordenadas de la posición del centroide del triángulo y el ángulo de orientación φ que indica la orientación del triángulo en el plano. [34]

$$Q_P = \begin{bmatrix} Xc \\ Yc \\ \varphi \end{bmatrix} = \begin{bmatrix} \frac{x_1+x_2+x_3}{3} \\ \frac{y_1+y_2+y_3}{3} \\ \arctan\left(\frac{\frac{2}{3}x_1-\frac{1}{3}x_2-\frac{1}{3}x_3}{\frac{2}{3}y_1-\frac{1}{3}y_2-\frac{1}{3}y_3}\right) \end{bmatrix} \tag{2.9}$$

Reduciendo la Ecuación (2.9) se tiene:

$$Q_P = \begin{bmatrix} Xc \\ Yc \\ \varphi \end{bmatrix} = \begin{bmatrix} \frac{x_1+x_2+x_3}{3} \\ \frac{y_1+y_2+y_3}{3} \\ \arctan\left(\frac{2x_1-x_2-x_3}{2y_1-y_2-y_3}\right) \end{bmatrix} \tag{2.10}$$

Al obtener la matriz del Jacobiano se puede relacionar el vector formación (Q_F) y el vector postura (Q_P) con el vector posición de los tres robots móviles (X), como se indica en la Ecuación 2.11.

$$\dot{Q} = \begin{bmatrix} \dot{d}_1 \\ \dot{d}_2 \\ \dot{\beta} \\ \dot{X}c \\ \dot{Y}c \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} \frac{\partial d_1}{\partial x_1} & \frac{\partial d_1}{\partial y_1} & \frac{\partial d_1}{\partial x_2} & \frac{\partial d_1}{\partial y_2} & \frac{\partial d_1}{\partial x_3} & \frac{\partial d_1}{\partial y_3} \\ \frac{\partial d_2}{\partial x_1} & \frac{\partial d_2}{\partial y_1} & \frac{\partial d_2}{\partial x_2} & \frac{\partial d_2}{\partial y_2} & \frac{\partial d_2}{\partial x_3} & \frac{\partial d_2}{\partial y_3} \\ \frac{\partial \beta}{\partial x_1} & \frac{\partial \beta}{\partial y_1} & \frac{\partial \beta}{\partial x_2} & \frac{\partial \beta}{\partial y_2} & \frac{\partial \beta}{\partial x_3} & \frac{\partial \beta}{\partial y_3} \\ \frac{\partial X_c}{\partial x_1} & \frac{\partial X_c}{\partial y_1} & \frac{\partial X_c}{\partial x_2} & \frac{\partial X_c}{\partial y_2} & \frac{\partial X_c}{\partial x_3} & \frac{\partial X_c}{\partial y_3} \\ \frac{\partial Y_c}{\partial x_1} & \frac{\partial Y_c}{\partial y_1} & \frac{\partial Y_c}{\partial x_2} & \frac{\partial Y_c}{\partial y_2} & \frac{\partial Y_c}{\partial x_3} & \frac{\partial Y_c}{\partial y_3} \\ \frac{\partial \varphi}{\partial x_1} & \frac{\partial \varphi}{\partial y_1} & \frac{\partial \varphi}{\partial x_2} & \frac{\partial \varphi}{\partial y_2} & \frac{\partial \varphi}{\partial x_3} & \frac{\partial \varphi}{\partial y_3} \end{bmatrix} \begin{bmatrix} \dot{x}_1 \\ \dot{y}_1 \\ \dot{x}_2 \\ \dot{y}_2 \\ \dot{x}_3 \\ \dot{y}_3 \end{bmatrix} \tag{2.11}$$

De forma general la Ecuación 2.11 se la puede representar de la siguiente manera:

$$\dot{Q} = J(X)\ \dot{X}, \tag{2.12}$$

donde: $\dot{Q} = \left[\dot{d}_1\ \dot{d}_2 \dots\ \dot{\varphi}\right]$; $J(X)\ es\ el\ Jacobiano$ y $\dot{X} = [\dot{x}_1\ \dot{y}_1 \dots\ \dot{x}_3\ \dot{y}_3]$.

El Jacobiano se lo obtiene mediante el uso del software computacional MATLAB. El código del script utilizado se encuentra en el ANEXO II y las derivadas parciales de la Ecuación 2.11 se muestran en el ANEXO I.

2.3 DESARROLLO DE ALGORITMOS DE CONTROL

Una vez conocido el modelo de la planta, se desarrolla un control interno de velocidad para cada rueda de cada robot móvil y un control externo de la formación y postura para el grupo de robots móviles, en la Figura 2.14 se esquematiza la arquitectura de control completa del sistema.

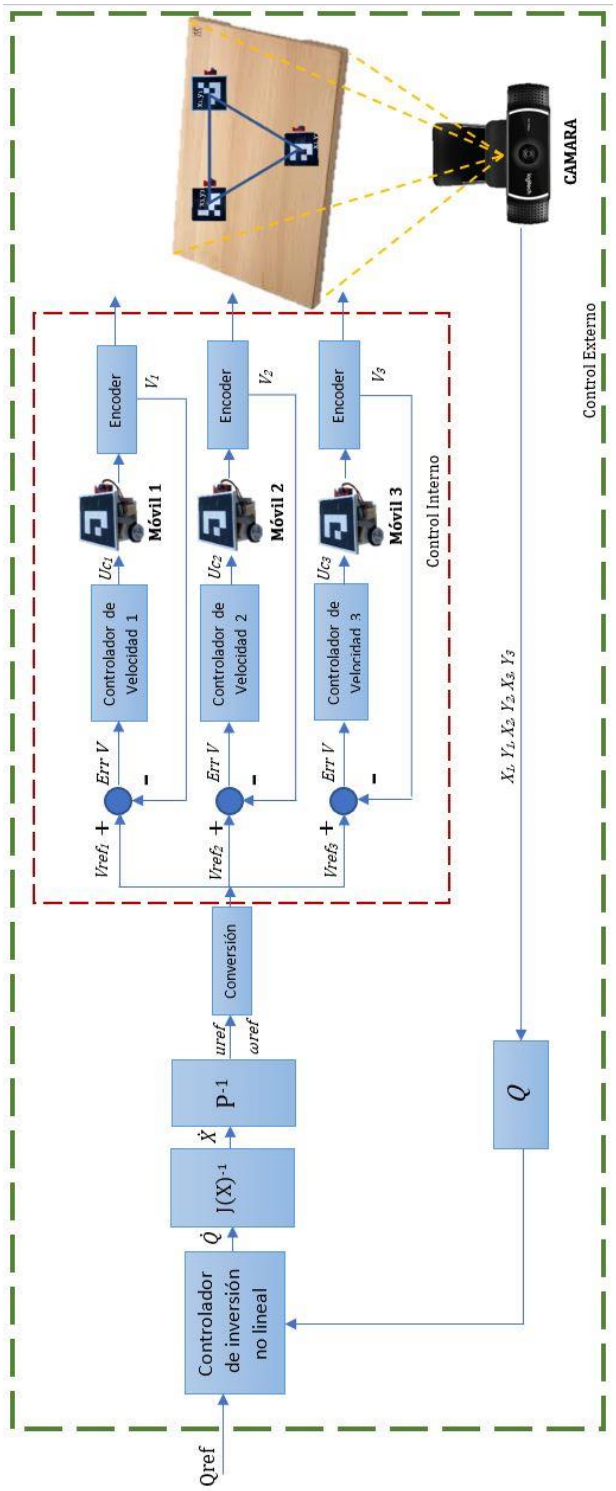


Figura 2.14 Lazo interno y externo de control

Lazo Interno de control

El lazo interno se encarga de realizar el control de velocidad de los robots móviles, en la Figura 2.14 se puede observar que las referencias de entrada son las velocidades de cada rueda (*Vref1, Vref2, Vref3*) y a la salida de la planta se tiene las velocidades reales (V_1, V_2, V_3) obtenidas por los encoders en cada rueda de los robots. Se definen $Vref_i = [Vl\,ref_i \quad Vr\,ref_i]^T$ y $V_i = [Vl_i \quad Vr_i]^T$, donde $i = 1,2,3$, como el vector de velocidades de referencia de la llanta izquierda y derecha y como el vector de velocidades de salida, respectivamente.

De manera más detallada y desglosada en la Figura 2.15 se puede esquematizar el bloque Controlador de Velocidad de la Figura 2.14, que representa los dos controladores internos de cada robot.

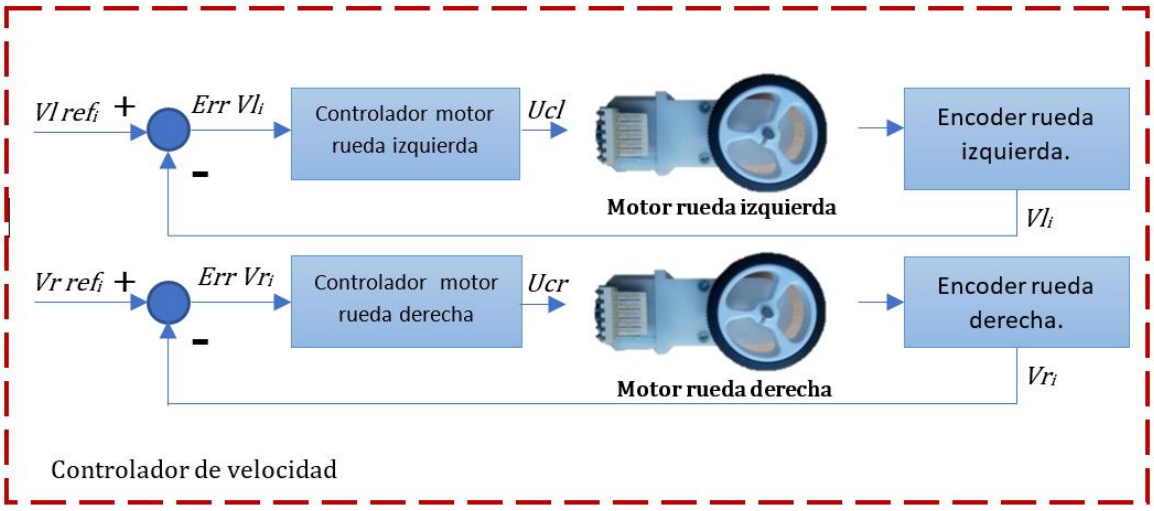


Figura 2.15 Controlador interno para robot móvil i

Este bloque está conformado por dos controladores PID, uno por cada llanta. En cada controlador ingresa el error de velocidad de cada llanta (*Err Vl*$_i$, *Err Vr*$_i$) que es igual a la diferencia entre las velocidades de referencia con las velocidades reales del robot, y como salida envían las señales de control (*Ucl*$_i$, *Ucr*$_i$) a cada motor respectivamente. En base a estas variables las ecuaciones de los PID son:

$$Ucl_i(t) = Kp\ Err\ Vl_i\ (t) + Ki \int Err\ Vl_i\ (t)\ dt + Kd\ \frac{\partial Err\ Vl_i\ (t)}{\partial t}\ , \tag{2.13}$$

$$Ucr_i(t) = Kp\ Err\ Vr_i\ (t) + Ki \int Err\ Vr_i\ (t)\ dt + Kd\ \frac{\partial Err\ Vr_i\ (t)}{\partial t} \tag{2.14}$$

Inversión dinámica no lineal (Lazo Externo)

Como el lazo de control interno requiere como referencias las velocidades de cada rueda de los robots móviles, se usa la Ecuación (2.7) para despejar el vector de velocidades ya que mediante la matriz P se relacionan las velocidades lineal y angular de cada robot móvil con la derivada de su posición.

$$V = P^{-1}\,\dot{X} \tag{2.15}$$

De forma similar se usa la Ecuación (2.12) para obtener el vector de las derivadas de las posiciones.

$$\dot{X} = [J(X)]^{-1}\,\dot{Q} \tag{2.16}$$

Sustituyendo la Ecuación (2.16) en la Ecuación (2.15) se obtiene:

$$V = P^{-1}\,[J(X)]^{-1}\,\dot{Q} \tag{2.17}$$

Para encontrar el vector de las derivadas de formación y postura $\dot{Q}$, se usa la Ecuación (2.18) que representa el control externo basado en inversión dinámica no lineal. [35]

$$\dot{Q} = \dot{Q}ref + K[\tanh(Qref - Q)]\,, \tag{2.18}$$

donde:

- $Qref.-$ Referencia de formación y postura definida por el vector $Qref = [d_{1_d}\ d_{2_d}\ \beta_d\ Xc_d\ Yc_d\ \varphi_d]$.
- $\tanh(Qref - Q).-$ Tangente hiperbólica del error de formación y postura que suaviza y satura la señal de control.
- $K.-$ Matriz diagonal con valores positivos.
- $\dot{Q}ref.-$ Primera derivada del vector de referencia de formación y postura.

Como la Ecuación (2.18) es un sistema de primer orden, se garantiza la estabilidad al definir la matriz K positiva [35]. Finalmente se sustituye la Ecuación (2.18) en la Ecuación (2.17) y se tiene a la salida las referencias de velocidad para el control interno.

$$V = P^{-1}\,[J(X)]^{-1}\left\{\dot{Q}ref + K[\tanh(Qref - Q)]\right\} \tag{2.19}$$

Debido a que las entradas del robot móvil son entradas de accionamiento diferencial, es decir velocidades para las ruedas derecha e izquierda, como se observa en la Figura 2.16,

mientras que la salida del controlador de inversión dinámica no lineal (controlador externo) son velocidades lineal y angular del robot, se debe realizar una conversión.

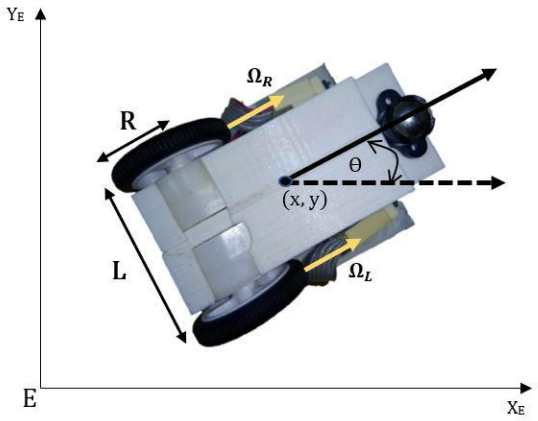


Figura 2.16 Robot Diferencial.

Basándose en el modelo matemático de un robot tipo uniciclo, el cual describe generalmente el movimiento de los robots diferenciales, se obtienen las ecuaciones que relacionan la velocidad lineal y angular aplicadas al robot móvil con las velocidades de sus ruedas. Las ecuaciones obtenidas son:

$$u_m = \frac{R(\Omega_R + \Omega_L)}{2}, \tag{2.20}$$

$$\omega_m = \frac{R(\Omega_R - \Omega_L)}{L}, \tag{2.21}$$

donde:

- u_m . – Velocidad lineal del robot móvil.
- ω_m. – Velocidad angular del robot móvil.
- R. – Radio de las ruedas.
- L. – Distancia entre las dos ruedas (eje).
- Ω_R. – Velocidad angular de la rueda derecha.
- Ω_L. – Velocidad angular de la rueda izquierda.

De las Ecuaciones 2.20 y 2.21, se obtienen las acciones de control sobre cada rueda derecha e izquierda respectivamente.

$$\Omega_R = \frac{(2\,u_m + \omega_m\,L)}{2\,R}, \tag{2.22}$$

$$\Omega_L = \frac{(2\,u_m - \omega_m\,L)}{2\,R} \tag{2.23}$$

Para su fácil manejo, estas acciones de control se expresan en revoluciones por minuto y son las entradas del controlador interno de velocidad.

2.4 IMPLEMENTACIÓN DEL SISTEMA

Implementación en ROS (Robotic Operating System)

El uso de ROS como base principal para la implementación del sistema se debe a que es un framework que permite la comunicación transparente entre nodos, los cuales pueden ser configurados para diferentes propósitos. La instalación y creación de área de trabajo de ROS se encuentra en el ANEXO III.

Para la implementación de todo el sistema primero se crea un paquete en ROS el cual se describe a continuación.

Creación de paquete en ROS

Se crea una carpeta con el nombre “FRM_ROS” (Formación de Robots Móviles en ROS), la cual es el paquete donde se ubican los nodos del sistema. La configuración e información del paquete FRM_ROS se encuentra en el archivo *CMakeList.txt,* que se encuentra en la Figura 2.17. Este archivo gestiona las dependencias, mensajes, servicios y nodos del paquete de ROS.

Para la creación de este paquete es necesario definir las dependencias y las librerías que se usan para el desarrollo del sistema. Las principales librerías usadas por los diferentes nodos son*: cv_bridge, image_transport, roscpp, rospy, sensor_msgs, std_msgs, geometry_msgs, message_generation, visualization_msgs, tf.*

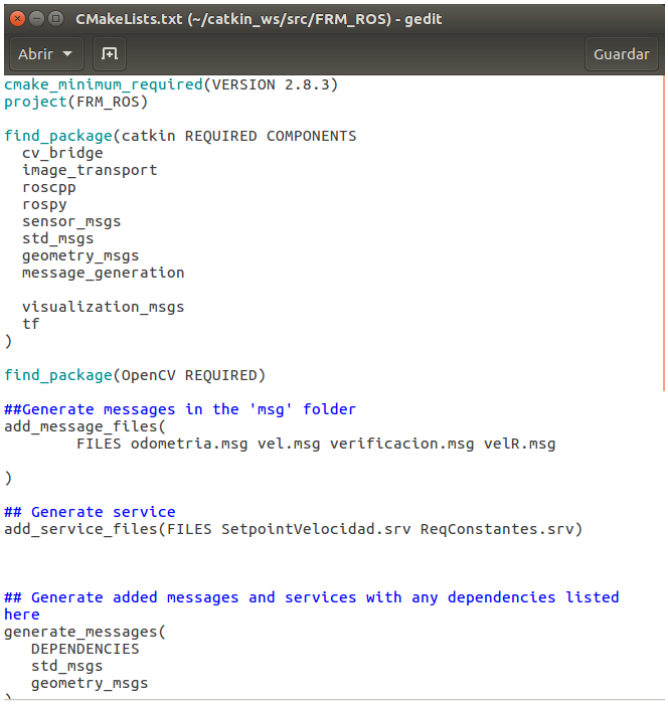


Figura 2.17 *CMakeList.txt.*

Otro archivo que contiene información y es parte de la configuración del paquete es el package.xml. Este archivo, como se observa en la Figura 2.18, contiene los tipos de mensajes que son utilizados en el paquete.

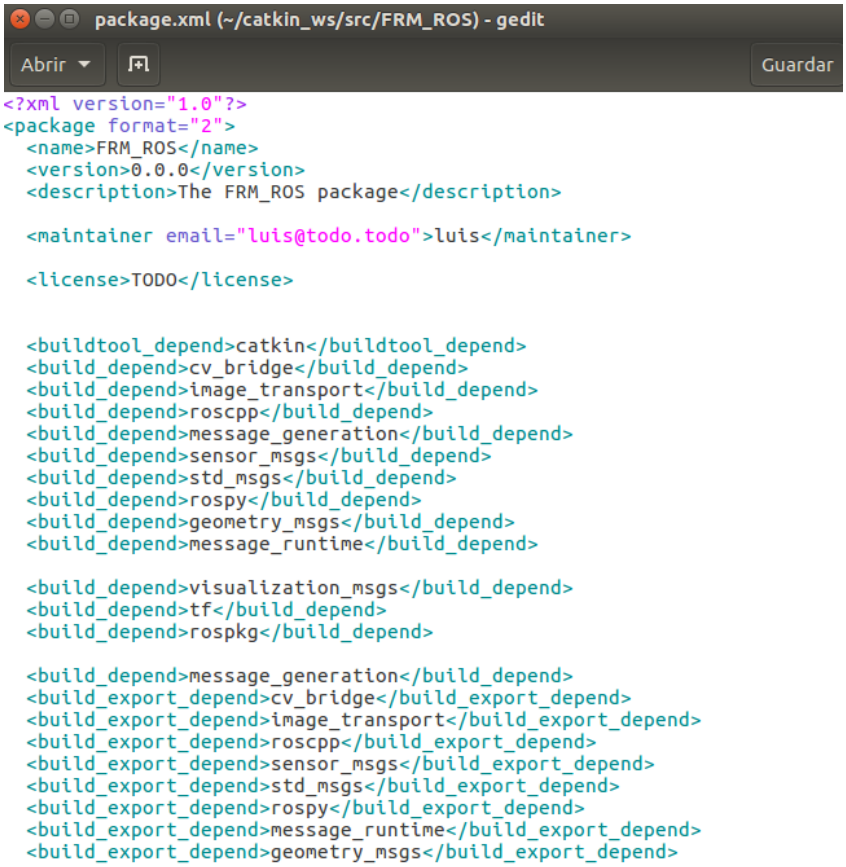


Figura 2.18 package.xml.

En la Figura 2.19 se observa que el paquete FRM_ROS tiene además los siguientes directorios: *include, launch, msg, src, srv*. Cada carpeta contiene diferentes archivos para la ejecución del sistema.

Figura 2.19 Paquete FRM_ROS.

La carpeta *msg* contiene los mensajes creados para la comunicación entre nodos. El paquete FRM_ROS utiliza tres mensajes los cuales son *odometria*, *vel*, *velR*. El uso de estos mensajes se encuentra en la descripción de cada nodo, así como también los servicios empleados, los cuales se encuentran en la carpeta *srv* y estos servicios son *ReqConstantes* y *SetpointVelocidad.*

Los nodos del paquete se encuentran en la carpeta *include* y *src*. La carpeta *include* contiene a los nodos realizados en lenguaje C++ mientras que en el *src* están los nodos realizados en lenguaje Python.

El paquete en total tiene cuatro nodos los cuales son:

- v_artificial.cpp realizado en C++
- unidad_central.py realizado en Python
- Response_robots.py realizado en Python
- Request_robots.py realizado en Python

Cada uno de estos nodos realiza una función específica que se detallará más adelante. La interacción entre estos nodos se puede observar en la Figura 2.20 gracias al comando *rqt_graph*, la cual es una herramienta que dispone ROS para visualizar los nodos, mensajes y tópicos activos del paquete FRM_ROS, los óvalos representan los nodos y los servicios mientras que los rectángulos los tópicos.

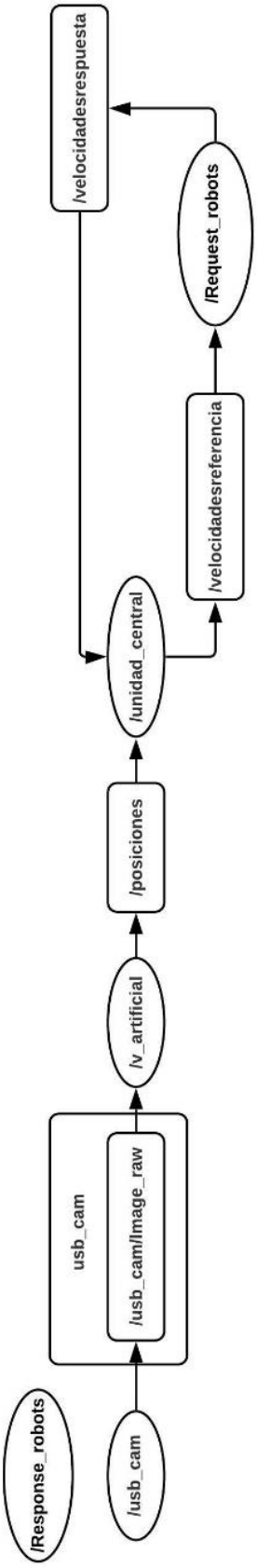


Figura 2.20 Rqt_graph.

Adicionalmente se utiliza un paquete llamado usb_cam, este paquete viene instalado dentro de ROS y permite que cualquier cámara conectada mediante conexión USB envíe datos de forma que cualquier nodo pueda leerlos y hacer uso de ellos. Estos datos son publicados en el tópico */usb_cam/image_raw* como se observa en la Figura 2.20, y a su vez está suscrito al nodo de visión artificial.

La carpeta *launch* contiene el archivo *FRM_ROS_launch.launch* que mediante la herramienta de roslaunch de ROS, se inicia fácilmente múltiples nodos ROS de forma local y remota, así como configura parámetros en el servidor de parámetros.

Nodo visión artificial

El nodo de visión artificial, el cual se observa en la Figura 2.20 tiene el nombre *v_artificial*. Este nodo realizado en lenguaje de programación C++ es el encargado de procesar los datos obtenidos por la cámara y su diagrama de flujo se presenta en la Figura 2.21.

La cámara entrega la información en forma de matriz la cual tiene el nombre de image_convert y con ayuda de las librerías de OpenCv se detectan los marcadores de referencia ArUco y se obtiene los datos necesarios para determinar la posición de los robots móviles en el área de trabajo. Estos valores obtenidos son publicados al nodo *unidad_central* usando el mensaje *odometria.msg* el cual está subscrito mediante el tópico */posiciones*. En este mensaje se encuentra el identificador, la posición en x,y y el ángulo de dirección de cada marcador de referencia. Además, al tener los datos de odometría se puede obtener el punto central de la formación el cual tiene como nombre en el mensaje *xcentralreal* y *ycentralreal* como se observa en la Figura 2.22. Este proceso será detallado más adelante en la sección de implementación del sistema de visión artificial.

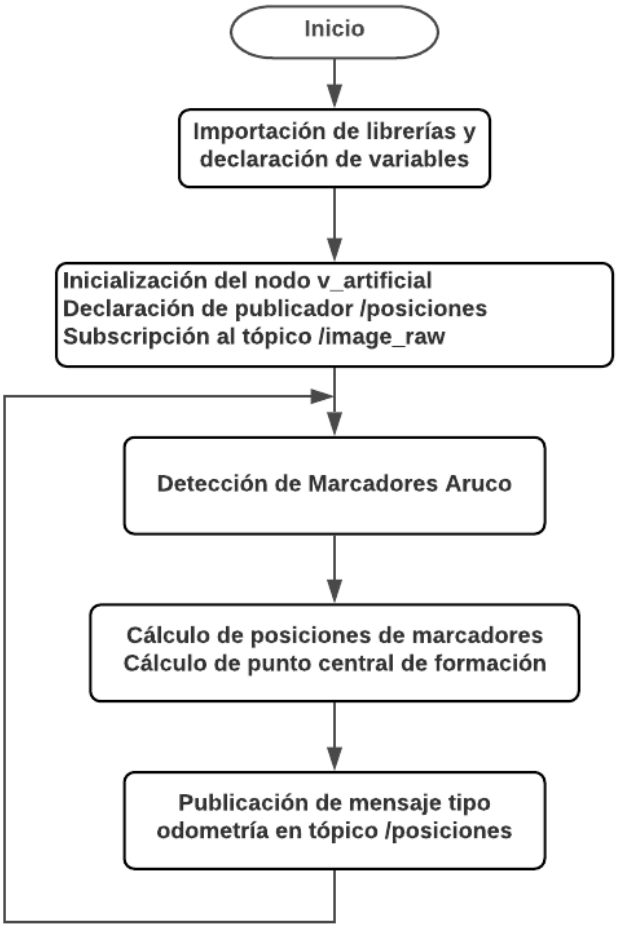


Figura 2.21 Diagrama de flujo nodo *v_artificial*.

odometria.msg (~/catkin_ws/src/FF

Abrir

```
# posiciones.msg

# datos odometria
float64  id0
float64  x0
float64  y0
float64  angular0
float64  id1
float64  x1
float64  y1
float64  angular1
float64  id2
float64  x2
float64  y2
float64  angular2

#puntocentral
float64  xcentralreal
float64  ycentralreal
```

Figura 2.22 *odometria.msg* del paquete FRM_ROS.

Nodo Unidad Central

El nodo *unidad_central* contiene todos los cálculos matemáticos para la implementación del controlador externo de formación y postura, por tal motivo este nodo es programado en lenguaje de programación Python que dispone de librerías para cálculos matriciales.

Los datos enviados mediante el tópico */posiciones* son recibidos por el nodo *unidad_central* el cual está suscrito al nodo *v_artificial*, con estos datos se cierra el lazo de control ya que se obtienen los valores de referencia de la matriz de formación y postura. Estos valores se actualizan en cada instante de tiempo al ser procesos independientes y que funcionan de forma paralela, principal ventaja de usar ROS.

Del controlador externo se obtiene las velocidades de salida las cuales son enviadas al grupo de robots móviles, para lo cual el nodo usa el tópico */velocidadesreferencia* con el mensaje *vel.msg*.

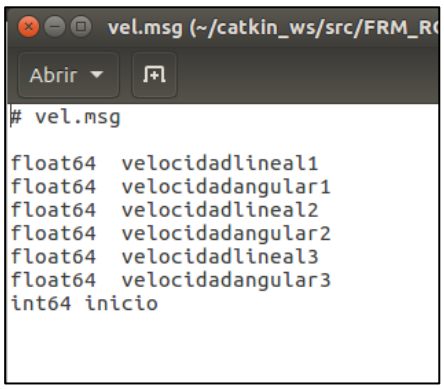


Figura 2.23 *vel.msg* del paquete FRM_ROS

En la Figura 2.23 se observa que el mensaje *vel.msg* tiene las velocidades lineales y angulares del grupo de robots móviles, además se tiene un campo llamado *inicio* el cual permite seleccionar las diferentes opciones enviadas por el nodo *unidad_central* hacia el grupo de robots. En total se pueden enviar cuatro opciones que corresponden a estado de espera, estado continuo, solicitud de estado de los robots, y actualización de constantes PID del controlador interno de velocidad.

El nodo *unidad_central* además de realizar los cálculos correspondientes al controlador externo, también permite la comunicación con la interfaz gráfica mediante archivos de texto JSON, los cuales son archivos ejecutables de texto hecho en un lenguaje de programación para almacenar y transferir datos. Python admite JSON a través de un paquete incorporado del mismo nombre. Para usar esta función, se importa este paquete en el script de Python.

A continuación, se observa en la Figura 2.24 el diagrama de flujo correspondiente al nodo *unidad_central.*

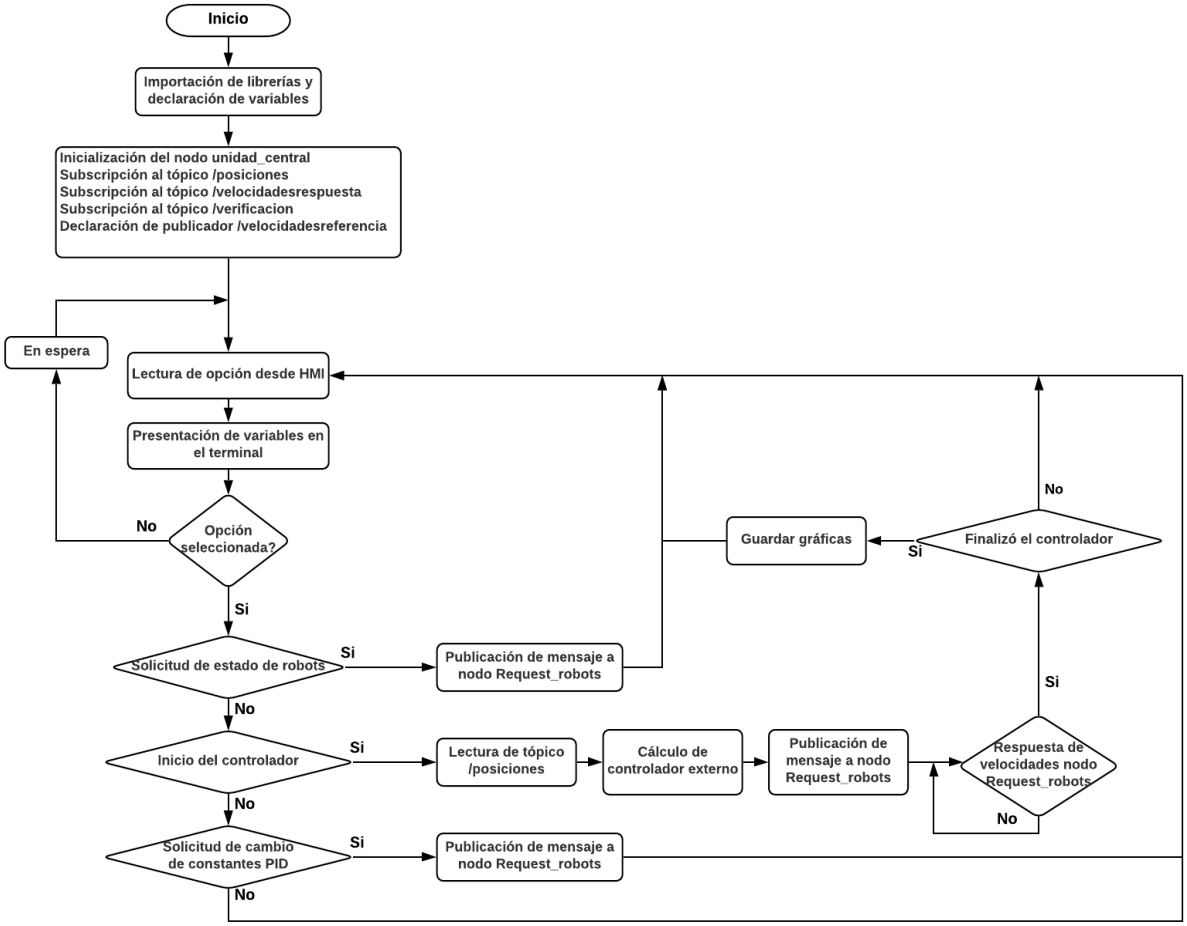


Figura 2.24 Diagrama de flujo nodo *unidad_central.*

Este nodo durante su ejecución presenta en un terminal de Ubuntu (Figura 2.25) información de las variables del controlador externo de formación y postura, parámetros de la trayectoria escogida y el tiempo del seguimiento de trayectoria.

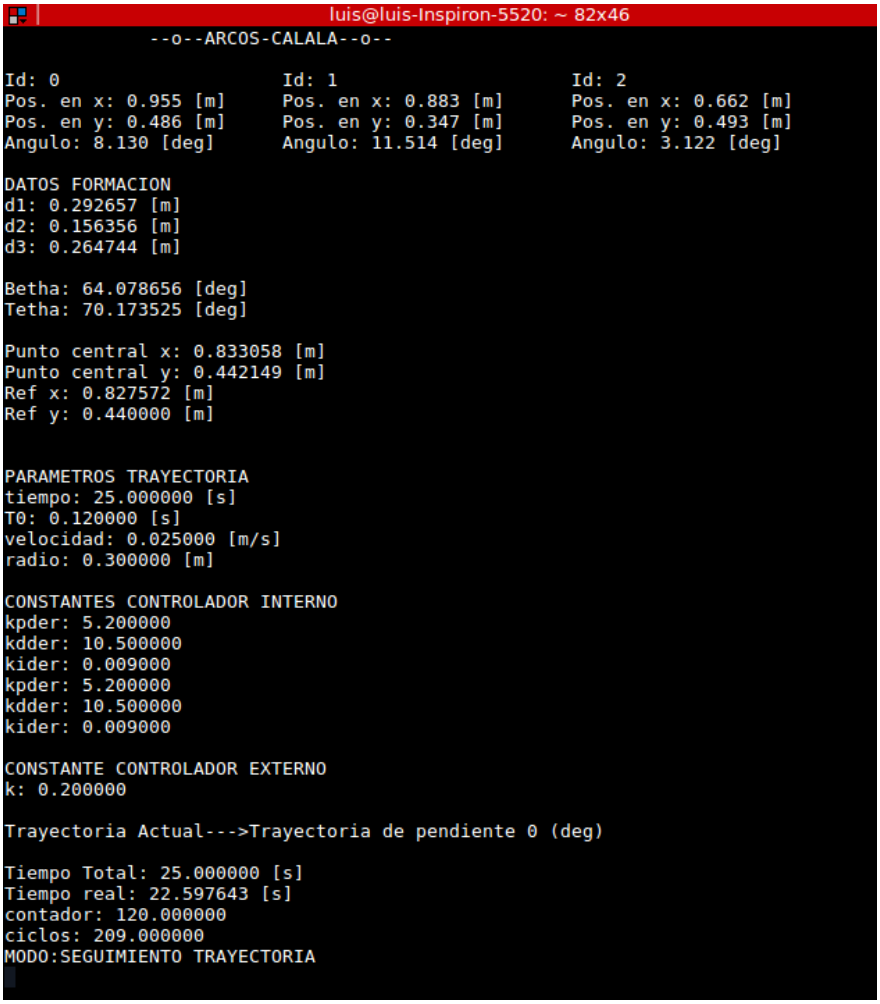


Figura 2.25 Interfaz en el terminal del ordenador, nodo *unidad_central.*

Nodo Request Robots

El nodo *Request_robots* tiene como tarea comunicar las diferentes opciones o solicitudes hacia el nodo *Response_robots*, el sistema de comunicación es de vital importancia ya que debe asegurar que estas solicitudes sean entregadas al grupo de robots móviles.

Para garantizar este proceso se utiliza ROS Service ya que mientras el nodo *Response_robots* no haya enviado una respuesta a la solicitud recibida desde *Request_robots* no permite que otra solicitud sea enviada, evitando que estas se acumulen y no exista una desincronización entre todo el sistema.

Para la comunicación con el nodo *Response_robots* se crearon dos servicios los cuales son *ReqConstantes.srv* y *SetpointVelocidad.srv* como se observan en la Figura 2.26.

a)

ReqConstantes.srv (~/catkin_v

Abrir

```
# Setpoint_velocidad.srv

float64 KpR
float64 KiR
float64 KdR
float64 KpL
float64 KiL
float64 KdL
int64 id1
int64 opc
---
int64 act
```

b)

SetpointVelocidad.srv (~/catki

Abrir

```
# Setpoint_velocidad.srv

float64 setpointR_1
float64 setpointL_1
float64 setpointR_2
float64 setpointL_2
float64 setpointR_3
float64 setpointL_3
int64 opc
---
float64[] velocidad
```

Figura 2.26 a) *ReqConstantes.srv*, b) *SetpointVelocidad.srv.*

La solicitud *ReqConstantes.srv* permite enviar al grupo de robots móviles las constantes PID del controlador interno de velocidad, mientras que el *SetpointVelocidad.srv* permite enviar las señales de control y la solicitud del estado del grupo de robots móviles.

Este nodo a la vez gestiona las respuestas de acuerdo con el requerimiento. En el caso del envió de señales de control, estas son transformadas en velocidad de rueda izquierda y rueda derecha, mientras que las respuestas serán las velocidades actuales de los robots. Esta respuesta es enviada hacia el nodo *unidad_central* por medio del mensaje *velR.msg* (Figura 2.27).

velR.msg (~/catkin_ws/src/FRM_ROS/msg)

Abrir

```
# velR.msg

float64  velocidadlineal1
float64  velocidadangular1
float64  velocidadlineal2
float64  velocidadangular2
float64  velocidadlineal3
float64  velocidadangular3
int64 inicio
int64 bit
```

Figura 2.27 *VelR.msg.*

Este mensaje es publicado mediante el tópico */velocidadesrespuesta* hacia el nodo *unidad_central.* Este mensaje tiene dos funciones, la primera corresponde a entregar las

velocidades al grupo de robots móviles y la segunda corresponde a permitir que el controlador siga ejecutándose ya que este se encuentra en espera de confirmación de recibimiento del mensaje por parte de los robots.

El diagrama de flujo de la Figura 2.28 muestra el funcionamiento del nodo *Request_robots.*

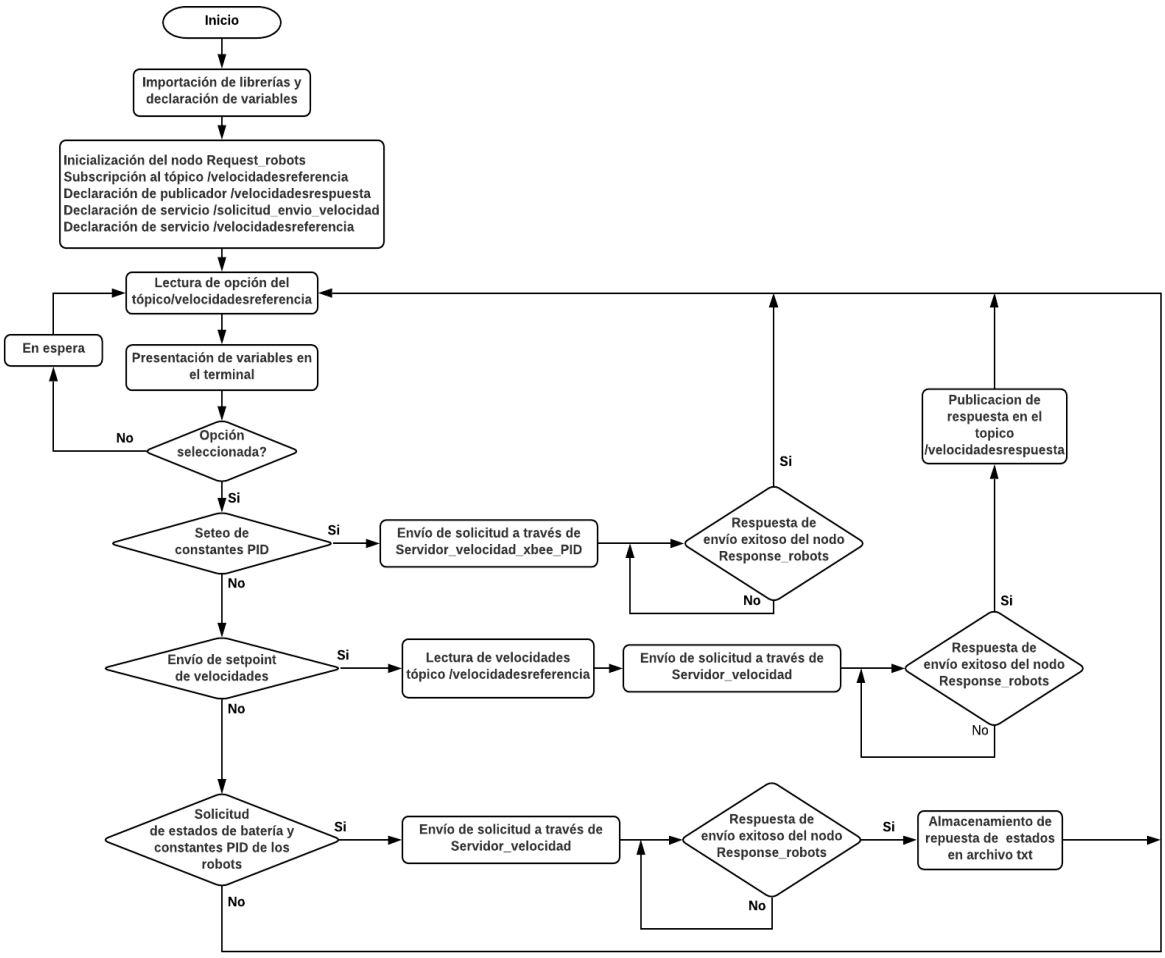


Figura 2.28 Diagrama de flujo nodo *Request_robots.*

El nodo presenta en un terminal (Figura 2.29) las velocidades enviadas del nodo *unidad_central* y las recibidas por el nodo *Response*, además se observan las respuestas enviadas por el grupo de robots móviles.

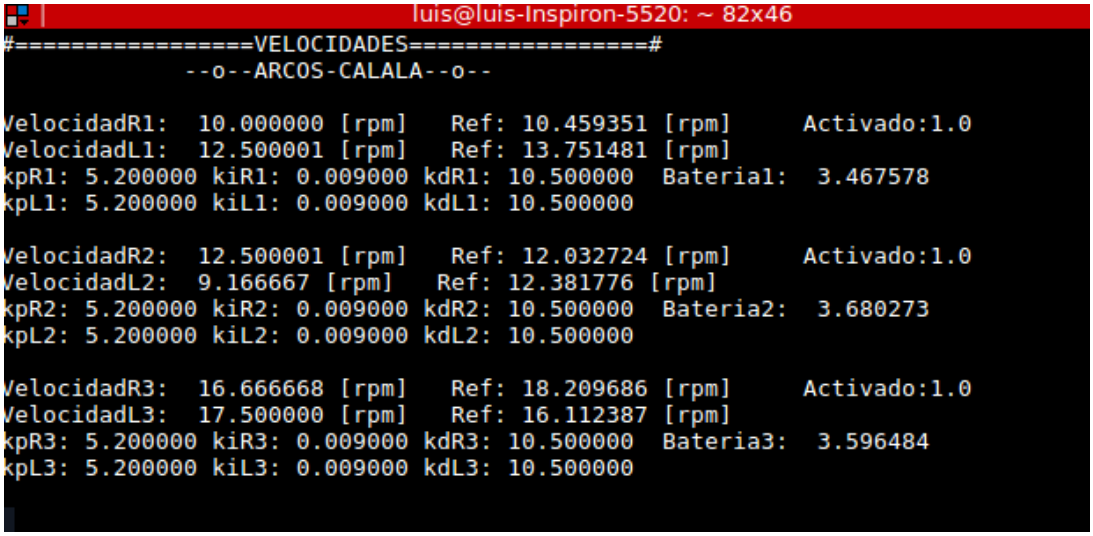


Figura 2.29 Interfaz en el terminal del ordenador, nodo *Request_robots*

Nodo Response Robots

Este nodo se encarga de procesar las solicitudes de ROS Service en mensajes que son enviados hacia los robots mediante el protocolo de comunicación API con escape, descrito en el capítulo anterior. Las variables que se configuran primero son el número de puerto que utiliza la tarjeta Xbee en el ordenador y las direcciones de los diferentes módulos Xbee de los robots móviles.

Para la comunicación se utiliza el paquete *rosserial* que dispone de una librería de Xbee modificada. La modificación consiste en que la comunicación punto-multipunto tenga un tiempo máximo de espera en el envío de información a cada una de las Xbee. Para el empaque y desempaque de los datos enviados y recibidos por las Xbee se utiliza la librería *struct* la cual permite gestionar el formato del dato enviado y recibido.

La descripción del funcionamiento del nodo *Response_robots* se la puede observar en la Figura 2.30.

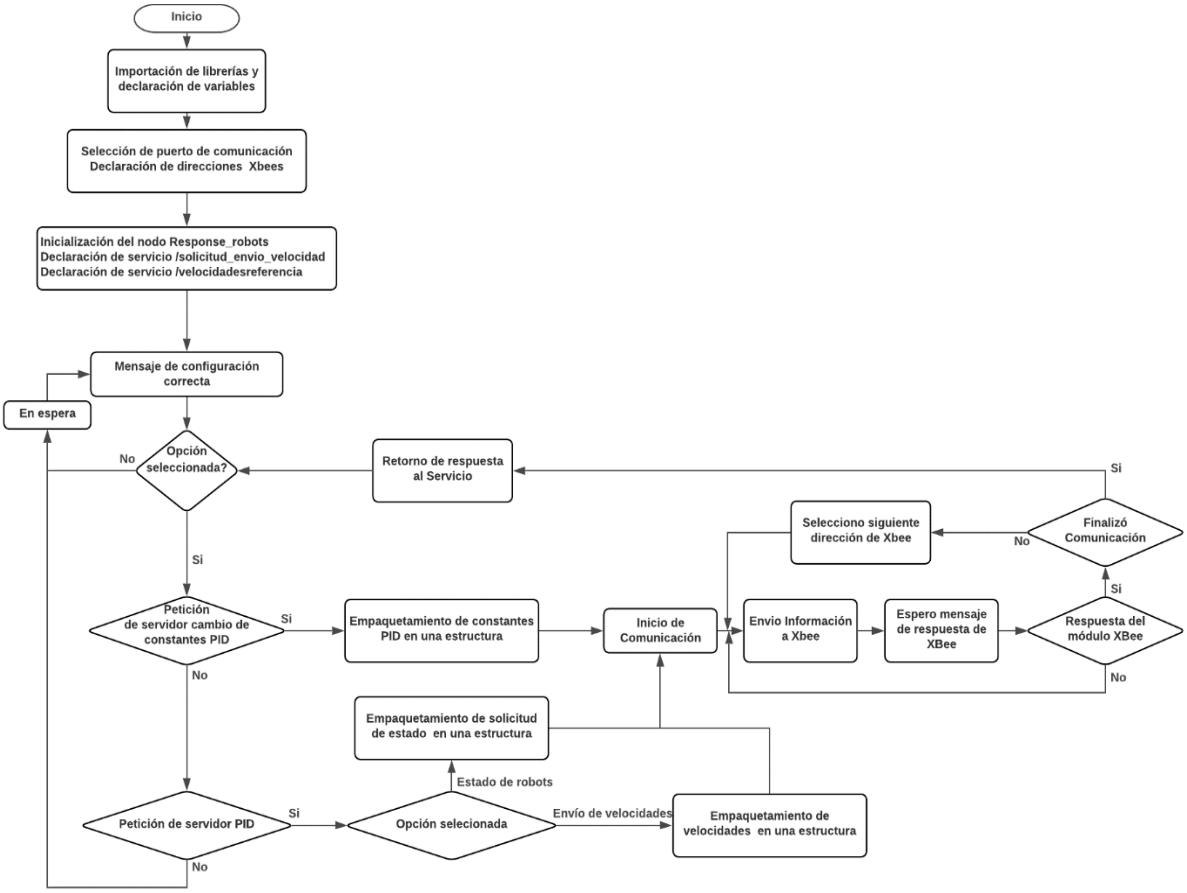


Figura 2.30 Diagrama de flujo nodo *Response_robots*

En cada uno de los cuatro nodos ya mencionados se implementan distintas partes del sistema, a continuación, se detalla la implementación del sistema de monitoreo con visión artificial, el sistema de control y el sistema de comunicación.

Implementación del sistema de visión artificial

El sistema de monitoreo por visión artificial está compuesto por un ordenador, la cámara de video de alta definición colocada sobre un pedestal, el área de trabajo y los marcadores de referencia, como se observa en la Figura 2.31. Todos estos componentes son los mismos utilizados en el banco de pruebas que se toma como base [3], con la diferencia que se migran sus algoritmos de visión artificial al sistema operativo Linux/ROS.

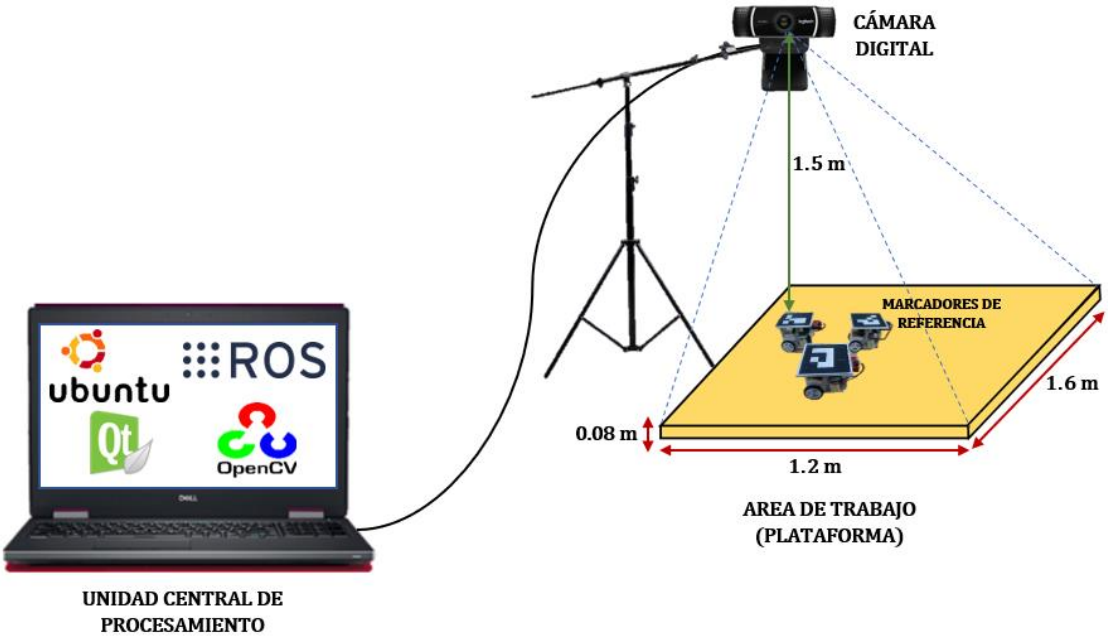


Figura 2.31 Arquitectura del sistema de monitoreo.

A continuación, se resumen las características de los componentes ya existentes y se detallan las librerías y algoritmos para la visión por computadora, pero con base a la migración a Linux/ROS.

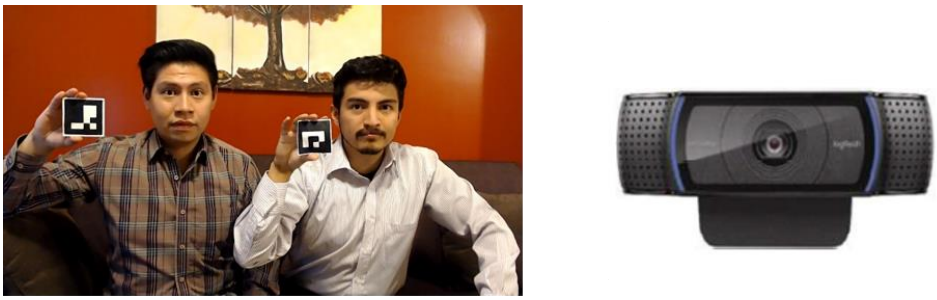
Cámara de video

La cámara de video sirve para obtener información del entorno de trabajo y funciona como sensor de visión para la realimentación de las posiciones de los robots móviles. La cámara de video que se utiliza en el presente proyecto es la cámara Logitech PRO-HD C920, con características presentadas en la Tabla 2.3.

Tabla 2.3 Características cámara Logitech PRO-HD C920 [36]

Características	Logitech PRO-HD C920
Formato de video digital	MP4
Máxima Resolución de video [pixeles]	1080 [HD]
Cuadros por segundo	30 fps
Sensor de Imagen	1000
Tipo de enfoque	Automático
Memoria RAM Requerida del sistema	2 GB RAM
Procesador mínimo requerido del sistema	2.4 GHz Intel Core 2 Duo processor.

Para verificar la calidad de imagen de la cámara se procedió a capturar una imagen del entorno con una definición de 1080p como se observa en la Figura 2.32, la eficiencia del código fuente hace que la velocidad de las aplicaciones Linux sean superiores a las que corren sobre Windows, debido a esto la carga computacional de la cámara no afecta al significativamente al sistema [37]



(a) (b)

Figura 2.32 (a) Imagen capturada con cámara Logitech; (b) Cámara Logitech C920 [36] .

La imagen capturada es de muy buena calidad, así que la cámara es ideal para la detección de los marcadores de referencia ArUco.

Área de Trabajo y campo de visión

Como se observa en la Figura 2.33, la cámara se coloca sobre un pedestal de micrófono con enfoque perpendicular al área de trabajo que mide 1.6 m x 1.2 m x 0.08m. Luego de varias pruebas la cámara se colocó a una altura de 1.5 metros, a esta altura la cámara

reconoce muy bien los marcadores de referencia y el campo de visión abarca toda el área de trabajo.

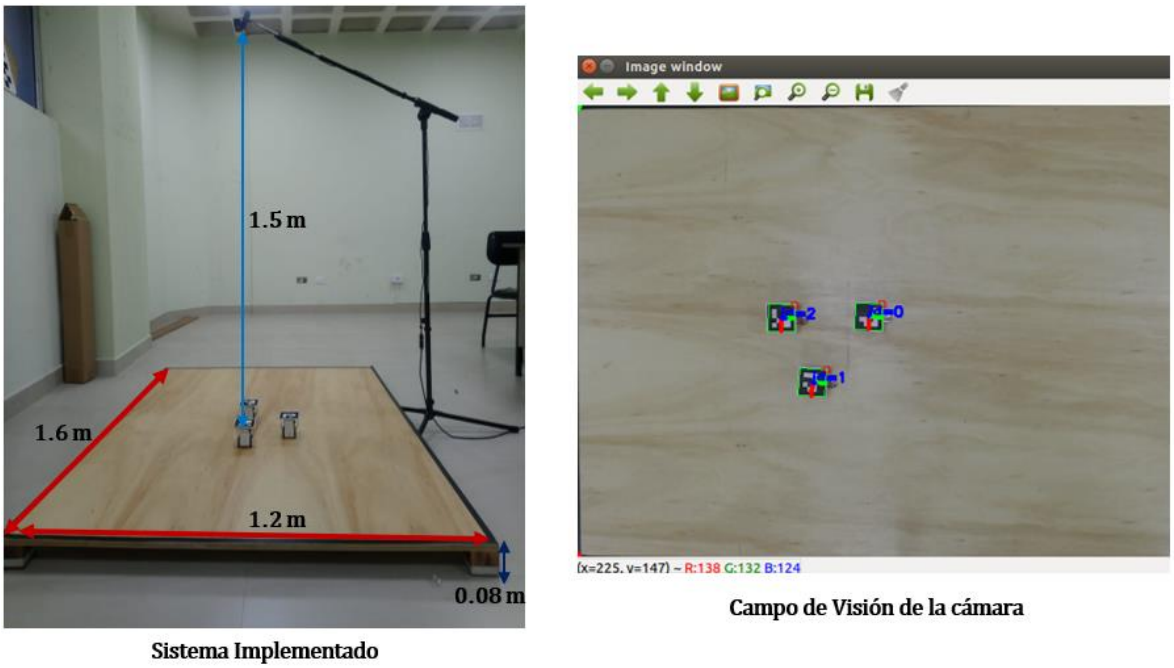


Figura 2.33 Montaje de la cámara sobre el área de trabajo y campo de visión.

Como se especificó en el Capítulo 1 en la Sección de Visión Artificial, para realizar el sistema de visión por computadora es necesario tomar en cuenta tres aspectos:

- Generación de marcadores de referencias. – En esta etapa se elige un diccionario y se generan los tres marcadores de referencia para la identificación de los robots móviles en el entorno.
- Calibración de la cámara. - En esta etapa se extraen los parámetros de la cámara para la detección de marcadores.
- Detección de marcadores ArUco. – En esta etapa la cámara envía fotogramas al ordenador, se procesa la información y se detecta la posición y orientación de los marcadores de referencia.

Generación de marcadores

En el presente proyecto se utiliza el software computacional OpenCV 3.4.4 y Qt Creator 3.5 para generar los marcadores de referencia. Esta versión de OpenCV dispone de varios módulos y librerías que facilitan el manejo de marcadores de referencia ArUco Marker.

Las librerías de OpenCV han sido desarrolladas en C++ y empleadas en Qt Creator. Para hacer uso de estas librerías es necesario realizar una serie de pasos detallados a continuación.

Primero se deben configurar los archivos de proyecto de Qmake, en este caso para un proyecto tipo aplicación [38]. En la Figura 2.34 se observa cómo se configura agregando las variables del sistema en una plantilla de la aplicación en el archivo .pro el cual especifica información sobre la aplicación. Además, se incluye la ubicación de los directorios y de las librerías de OpenCV.

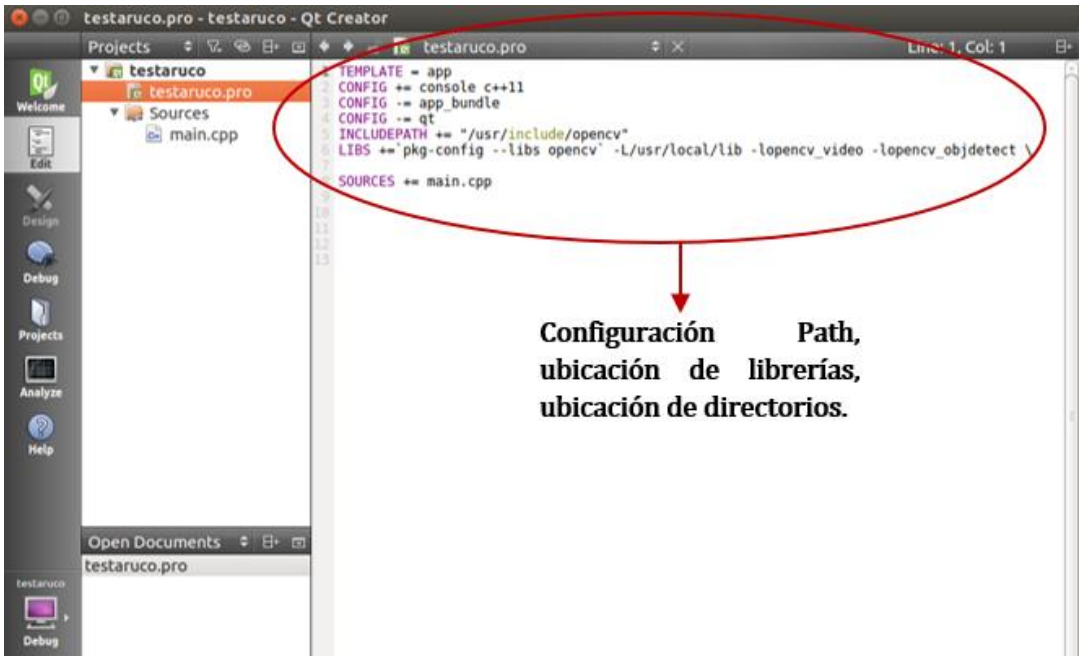


Figura 2.34 Configuración de directorios y librerías en el proyecto.

Entre los comandos usados están:

- *TEMPLATE.* - Indica que el objeto del proyecto es una aplicación o programa; se puede escribir *lib* en lugar de *app* para realizar un proyecto tipo librería.
- *CONFIG.* – Le dice a la configuración base que se desea un programa modo consola.
- *INCLUDEPATH.* – Es usada por el compilador C++ cuando resuelve las sentencias *#include.*
- *LIBS.* - Permite añadir la ubicación de las librerías de OpenCV en el proyecto.

Luego de haber configurado los archivos de proyecto, en el encabezado del programa principal se debe agregar las librerías como se observa en la Figura 2.35; entre ellas están:

opencv2/highgui.hpp, opencv2/aruco.hpp, opencv2/opencv.hpp, opencv2/core.hpp.

OpenCV tiene una estructura modular, lo que significa que un paquete incluye varias bibliotecas compartidas o estáticas. Para la generación de marcadores se utilizan las siguientes bibliotecas [39] :

- *Core.* – Es un módulo compacto que define las estructuras básicas de datos, incluidas las funciones básicas de OpenCV y la matriz multidimensional Mat que sirve para almacenar la imagen de la cámara.
- *Highgui.* – Es un módulo que permite la creación de un interfaz fácil de usar para capturar video, códecs de imagen y video, así como opciones simples de interfaz de usuario.
- *Aruco.* – Este módulo contiene todas las librerías de la biblioteca ArUco.

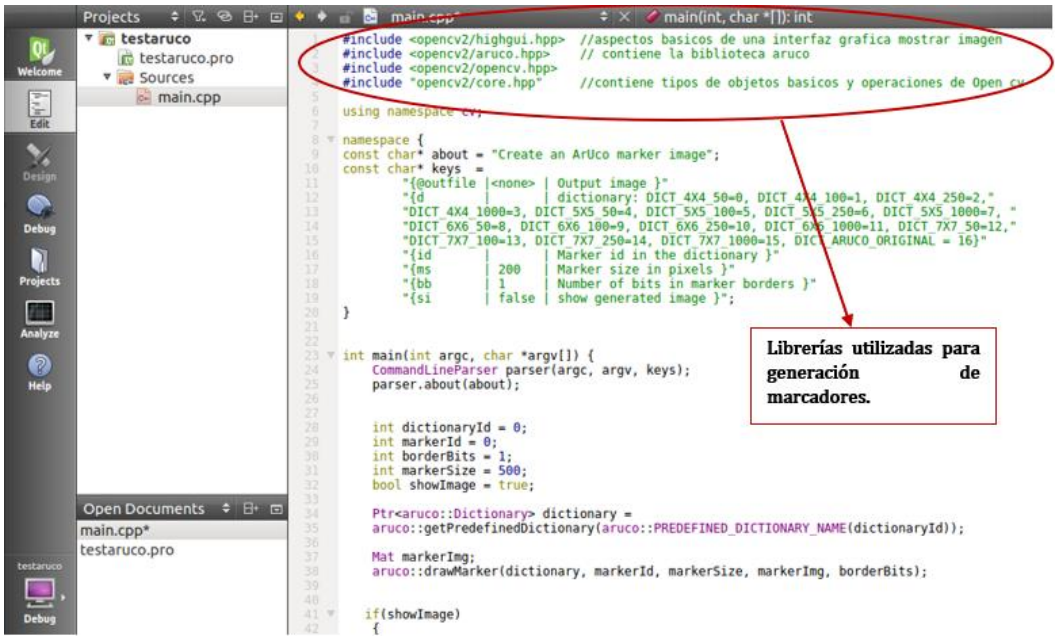


Figura 2.35 Declaración de librerías

Luego de la vinculación de librerías, a continuación, en la Figura 2.36, se esquematiza el proceso que se realizó para la generación de marcadores ArUco. Cabe recalcar que se estableció el mismo diccionario usado en el banco de pruebas anterior, DICT_4X4_50, que contiene 50 marcadores cada uno con una matriz interna de identificación de 4x4.

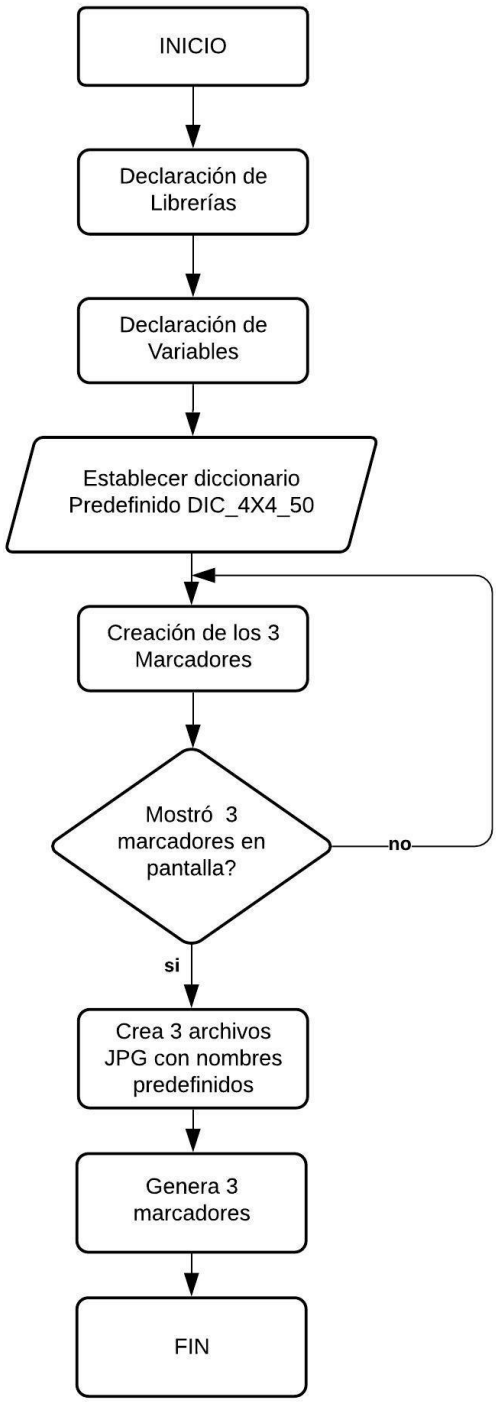


Figura 2.36 Generación de Marcadores de Referencia Matricial ArUco.

Los comandos más importantes usados en el proceso son:

- *getPredefinedDictionary.* - Mediante este comando se obtiene el contenido del diccionario elegido, se eligió DICT_4X4_50.
- *drawMarker.* - Este comando permite dibujar los marcadores con su respectiva codificación binaria individual.

- *Imshow.* - Permite mostrar las imágenes de los marcadores en una interfaz gráfica sencilla.
- *Imwrite.* – Permite crear el archivo de imagen tipo .JPG con el contenido de cada marcador de referencia matricial ArUco.

Para el banco de pruebas se utilizan tres marcadores de este diccionario los cuales se muestran en la Figura 2.37.

Figura 2.37 Marcador 0, Marcador 1 y Marcador 2.

En el caso de que se quiera expandir el número de robots móviles, se puede utilizar el mismo código para generar todos los marcadores de referencia.

Calibración de la cámara de video [40]

Para la calibración de la cámara se utilizaron las herramientas que proporciona ROS, por lo que se usa el nodo *cameracalibrator.py* del paquete *camera_calibration*. Este paquete de ROS permite una fácil calibración de cámaras monoculares o estéreo utilizando un tablero de ajedrez como patrón de calibración. Se utiliza un tablero de ajedrez grande con dimensiones conocidas, de 8 x 6 con cuadrados de 35 mm. En la Figura 2.38 se observa que la calibración utiliza los puntos de vértice interior del tablero de ajedrez, por lo que una placa 9 x 7 utiliza el parámetro de vértices internos 8 x 6.

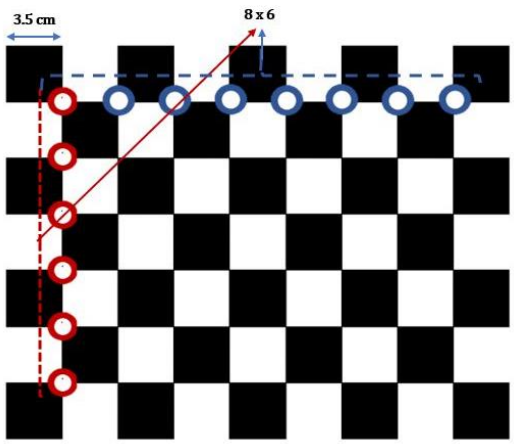


Figura 2.38 Patrón de Calibración provisto en ROS tutorials/ cámara_calibration [40].

A continuación, se detalla el proceso de calibración.

Compilación

Se comienza obteniendo las dependencias y compilando el controlador.

```
$ rosdep install camera_calibration
```

Se debe asegurar que la cámara esté publicando imágenes sobre ROS así que se hace una lista de los tópicos para verificar que las imágenes sean publicadas.

```
$ rostopic list
```

Este comando muestra todos los tópicos publicados así que se verifica que haya el tema *image_raw*. Los temas predeterminados proporcionados por la mayoría de los controladores de cámara ROS son:

/usb_cam/camera_info
/usb_cam/image_raw

Ejecución del nodo de calibración

Para comenzar la calibración, se deberá cargar los tópicos de imagen que se calibrarán.

```
$ rosrun camera_calibration cameracalibrator.py --size 8x6 --square
0.35 image:=/usb_cam/image_raw usb_cam:=/usb_cam
```

Esto abrirá la ventana de calibración que resaltará el tablero de ajedrez como se observa en la Figura 2.39.

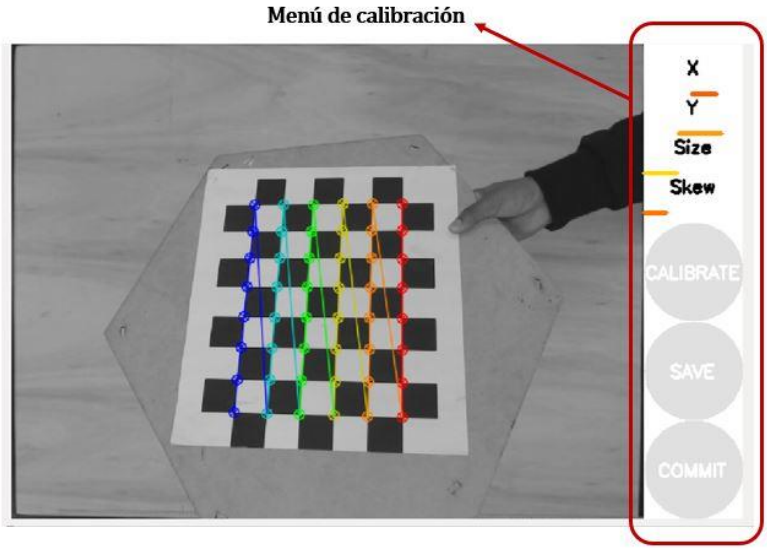


Figura 2.39 Ventana de calibración.

Mover el tablero de ajedrez

Para obtener una buena calibración, se deberá mover el tablero de ajedrez en el marco de la cámara de manera que:

- El tablero de ajedrez se mueva hacia la izquierda, derecha, arriba y abajo del campo de visión de la cámara. El menú lateral de calibración, ubicado en la parte derecha de la ventana como se observa en la Figura 2.38, muestra el tipo de movimiento que se debe realizar para la calibración.
    - Barra X – izquierda / derecha en el campo de visión.
    - Barra Y – arriba / abajo en el campo de visión.
    - Barra de tamaño: hacia / lejos e inclinación de la cámara.
- El tablero de ajedrez llene todo el campo de visión.
- El tablero de ajedrez se mueva de forma inclinada hacia la izquierda, derecha, arriba y abajo.

En la Figura 2.40, se observan las diferentes posiciones en las que se mueve el tablero. En cada paso, se mantiene el tablero de ajedrez estático hasta que la imagen queda resaltada en la ventana de calibración.

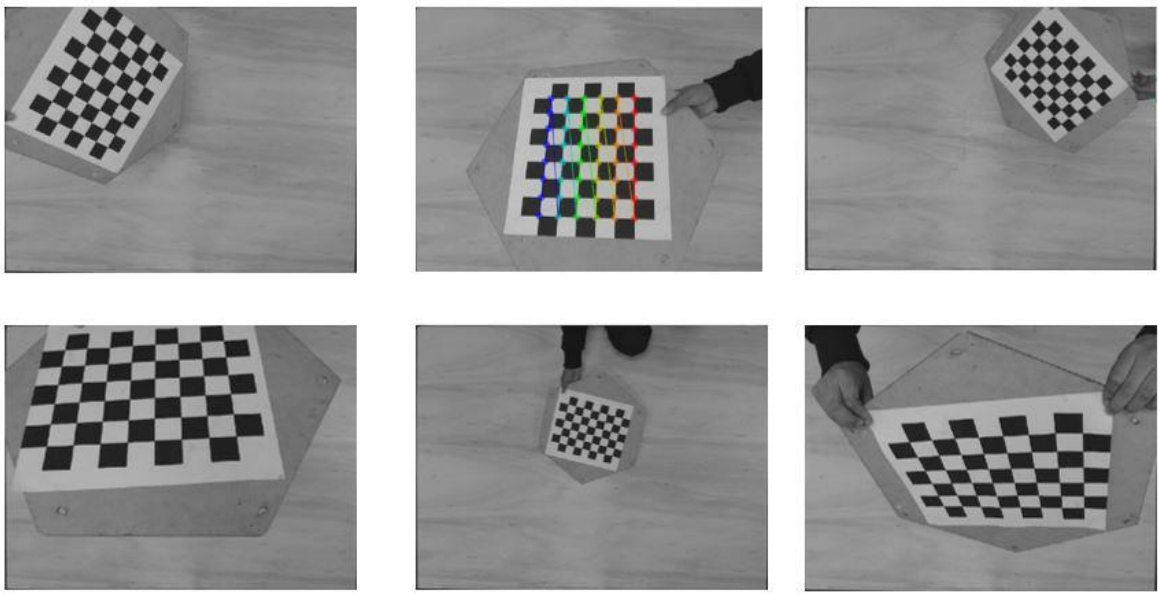


Figura 2.40 Movimientos del tablero en el marco de la cámara.

A medida que se mueve el tablero de ajedrez, se debe observar que las tres barras en el menú lateral de calibración aumentan de longitud. Cuando se ilumina el botón CALIBRAR, como se observa en la Figura 2.41, el programa indica que se tienen suficientes datos para

la calibración y se puede hacer clic en el mismo para ver los resultados. La calibración puede tomar aproximadamente un minuto.

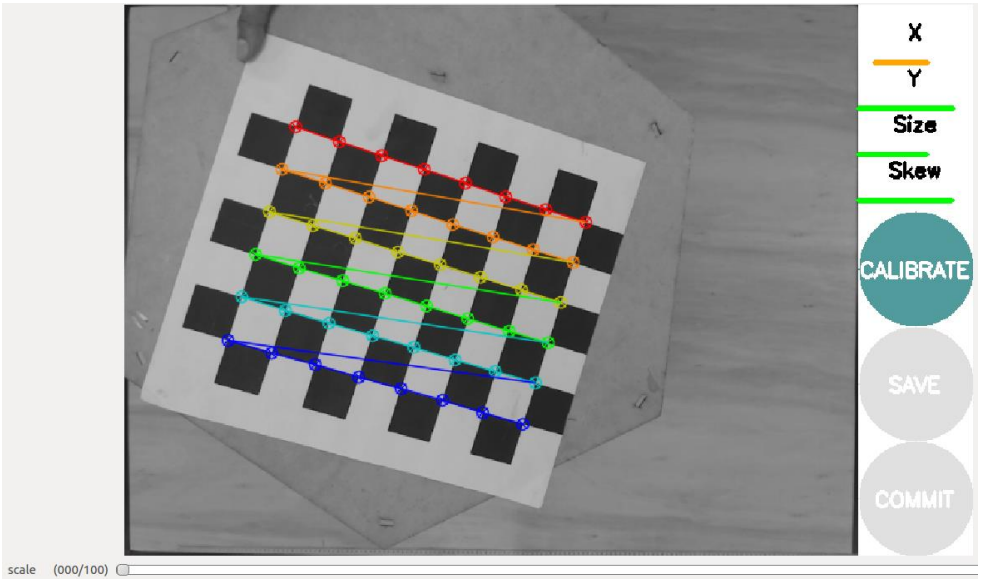


Figura 2.41 Adquisición de datos completa listo para CALIBRAR.

Resultados de la calibración (Modelo de la cámara)

Una calibración exitosa da como resultado, que los bordes rectos del mundo real aparezcan también rectos en la imagen corregida. Completada la calibración, se puede observar los resultados en el terminal del nodo. En la Figura 2.42 se presentan los valores de la matriz de la cámara y el vector de distorsión.

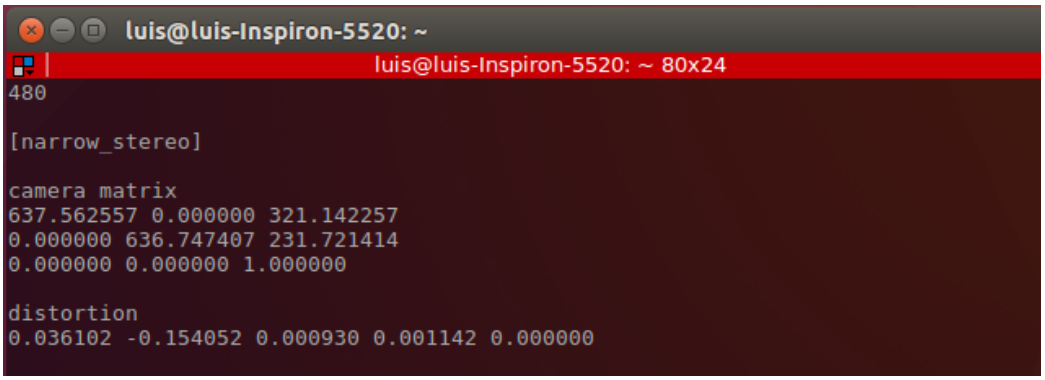


Figura 2.42 Ventana de calibración.

Los valores obtenidos para formar la matriz de calibración, Ecuación (2.18), y el vector de distorsión, Ecuación (2.19), son los siguientes:

$$Matriz\ de\ calibración = \begin{bmatrix} 637.562557 & 0 & 321.142257 \\ 0 & 636.747407 & 231.721414 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.18}$$

$$coef\ Distorsión = [0.036182 \quad -0.154052 \quad 0.000930 \quad 0.001142 \quad 0] \tag{2.19}$$

Creación del archivo yml

El archivo yml almacena los datos del modelo de la cámara, que luego van a ser usados en la detección de los marcadores ArUco. En la Figura 2.43 se observa los datos que se van a necesitar de la calibración de la cámara en el archivo yml.

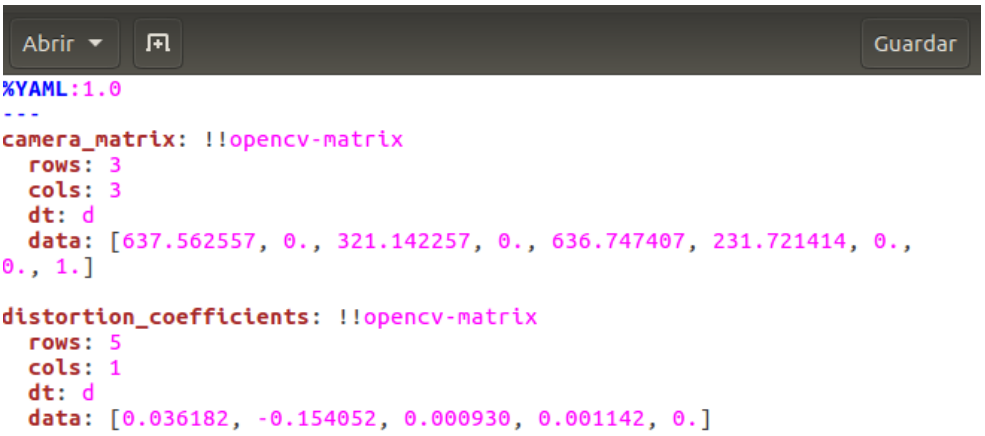


Figura 2.43 Archivo yml

Una vez terminada la calibración es posible realizar el proceso de detección de los marcadores de referencia matricial.

Detección de Marcadores ArUco

De acuerdo con la información detallada en la Sección Detección de Marcadores de Referencia del Capítulo anterior, el proceso de detección tiene que retornar una lista de marcadores detectados con la posición de sus cuatro esquinas y con la identificación del marcador en la imagen.

Con la ayuda de la función *detectMarkers()* se realiza este proceso. En cada etapa este comando utiliza una serie de parámetros, los cuales deben ser configurados previamente. Esta función y dichos parámetros se detallan a continuación.

detectMarkers() [41]

Es una función que detecta únicamente los marcadores incluidos en el diccionario especificado. Para cada marcador detectado, devuelve la posición 2D de sus cuatro esquinas en un vector y despliega solamente la esquina superior derecha y su identificador

correspondiente en la imagen. Se debe tomar en cuenta que esta función no realiza la estimación de pose (posición más orientación) del marcador. La sintaxis de la función es:

Void cv::aruco::detectMarkers (image, dictionary, corners, id, DetectorParameters:: create(), rejected)

Los parámetros que usa esta función son:

- *image:* es la imagen que adquiere la cámara.
- *dictionary:* indica el diccionario de marcadores que se va a utilizar.
- *corners: es el* vector de esquinas de los marcadores detectados. Para cada marcador, se proporcionan sus cuatro esquinas (por ejemplo su estructura es: std :: vector <std :: vector <cv :: Point2f>>). Para los N marcadores detectados, las dimensiones de esta matriz son Nx4. El orden de las esquinas es en sentido horario.
- *ids:* es el vector de los identificadores de los marcadores detectados. Para los N marcadores detectados, el tamaño de este vector tipo int también es N.
- *DetectorParameters::create():* son los parámetros para detección de marcador.
- *rejected:* contiene información de los puntos de los cuadrados no reconocidos en la imagen.

Los parámetros para detección de marcador se los almacena en un archivo yml y para su uso en la función detectMarkers() se utilizó la estructura cv::aruco::DetectorParameters, la cual se detalla a continuación.

DetectorParameters() [42]

Esta estructura permite extraer todos los parámetros para realizar el proceso de detección de los marcadores de referencia ArUco. En cada etapa del proceso detectMarkers utiliza diferentes parámetros los cuales son examinados a continuación.

- **Binarización y dibujo de contornos**
  - *adaptiveThreshWinSizeMin:* tamaño mínimo de ventana para el umbral adaptativo antes de encontrar contornos.
  - *adaptiveThreshWinSizeMax:* tamaño máximo de ventana para el umbral adaptativo antes de encontrar contornos.
  - *adaptiveThreshWinSizeStep:* pasos desde el mínimo valor al máximo valor del umbral adaptativo.
  - *adaptiveThreshConstant:* constante para el umbral adaptativo antes de encontrar contornos.

Si el valor de intensidad de un pixel es mayor al valor de umbral se le asigna el color blanco, caso contrario se le asigna el color negro como se observa en la Figura 2.44.

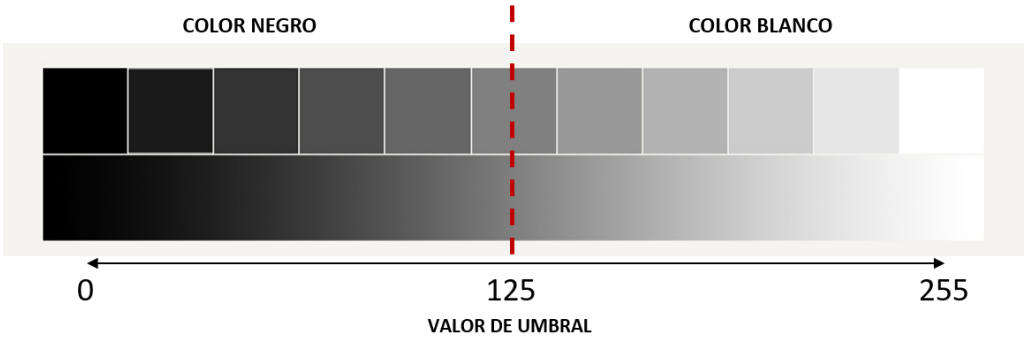


Figura 2.44 Umbral adaptativo

- **Detección de contornos**
  - *minMarkerPerimeterRate:* determina el perímetro mínimo para que se detecte el contorno del marcador. Esto se define como una tasa con respecto a la dimensión máxima de la imagen de entrada.
  - *maxMarkerPerimeterRate:* determina el perímetro máximo para que se detecte el contorno del marcador. Esto se define como una tasa con respecto a la dimensión máxima de la imagen de entrada.
- **Filtrado de contornos**
  - *polygonalApproxAccuracyRate:* precisión mínima durante el proceso de aproximación poligonal para determinar qué contornos son cuadrados.
- **Extracción de bits e identificación de marcador**
  - *minCornerDistanceRate:* distancia mínima entre las esquinas para los marcadores detectados en relación con su perímetro.
  - *minDistanceToBorder:* distancia mínima de cualquier esquina al borde de la imagen para los marcadores detectados.
  - *minMarkerDistanceRate:* distancia media mínima entre dos esquinas del marcador para que se considere similar, de modo que se elimine la más pequeña. La tasa es relativa al perímetro más pequeño de los dos marcadores.
  - *markerBorderBits*: número de bits del borde del marcador, es decir, ancho del borde del marcador.
  - *perspectiveRemovePixelPerCell*: número de bits (por dimensión) para cada celda del marcador al eliminar la perspectiva.
  - *perspectiveRemoveIgnoredMarginPerCell:* ancho del margen de píxeles en cada celda no considerado para la determinación del bit de celda. Representa la tasa con respecto al tamaño total de la celda.

- *maxErroneousBitsInBorderRate:* número máximo de bits erróneos aceptados en el borde (es decir, número de bits blancos permitidos en el borde). Representado como una tasa con respecto al número total de bits por marcador.
- *minOtsuStdDev:* desviación estándar mínima en valores de píxeles durante el paso de decodificación para aplicar el umbral de Otsu (calcula de forma automática un valor de umbral desde el histograma de la imagen bimodal [43]) de lo contrario, todos los bits se establecen en 0 o 1 dependiendo de la media superior a 128 o no.
- errorCorrectionRate: tasa de corrección de errores con respecto a la capacidad máxima de corrección de errores para cada diccionario.

Todos estos parámetros utilizan valores default los cuales se encuentran disponibles en el ANEXO IV. Luego de que el marcador es identificado, se procede a analizar la posición y orientación del marcador sobre el área de trabajo. Debido a que detectMarkers() no realiza estimación de pose, se utiliza la siguiente función.

estimatePoseSingleMarkers() [41]

La estimación de pose es muy importante ya que permite encontrar correspondencias entre puntos en el entorno real y su proyección de imagen 2D. Para realizar la estimación de la postura de la cámara se utilizan los parámetros de calibración generados anteriormente (matriz de la cámara y coeficientes de distorsión) y como resultado se tiene las matrices de rotación y traslación. La sintaxis de la función es:

aruco::estimatePoseSingleMarkers(corners, markerLength, camMatrix, distCoeffs, rvecs, tvecs);

Los parámetros que usa esta función son:

- *corners: es el* vector de esquinas de los marcadores detectados.
- *markerLength:* es el vector con la longitud del lado de los marcadores.
- *camMatrix*: es la matriz de la cámara.
- *distCoeffs:* es el vector de los coeficientes de distorsión.
- *rvecs:* es la matriz de salida de vectores de rotación, cada elemento del rvecs corresponde a cada marcador identificado.
- *tvecs:* es la matriz de salida de vectores de traslación, cada elemento del tvecs corresponde a cada marcador identificado.

Esta función retorna los puntos de cada sistema de coordenadas del marcador en el sistema de coordenadas de la cámara. El sistema de coordenadas del marcador está ubicado en el centro del marcador, con el eje Z perpendicular al plano del marcador.

Como se estableció una resolución de cámara de 640 x 480 pixeles, las coordenadas del centro de los marcadores también están en pixeles y con un sistema de referencia de cámara con origen en la parte superior como se observa en la Figura 2.45.

Figura 2.45 Límites del sistema de coordenadas de la cámara en pixeles

Para el presente proyecto se realizan ciertas trasformaciones para obtener las posiciones de los marcadores en medidas reales (metros). Primero se invierte el eje Y dispuesto por OpenCV, como se indica en la Figura 2.46 y mediante la estimación de pose se transforma la ubicación del marcador sobre el área de trabajo de coordenadas 2D expresadas en pixeles en coordenadas de homografía expresada en metros.

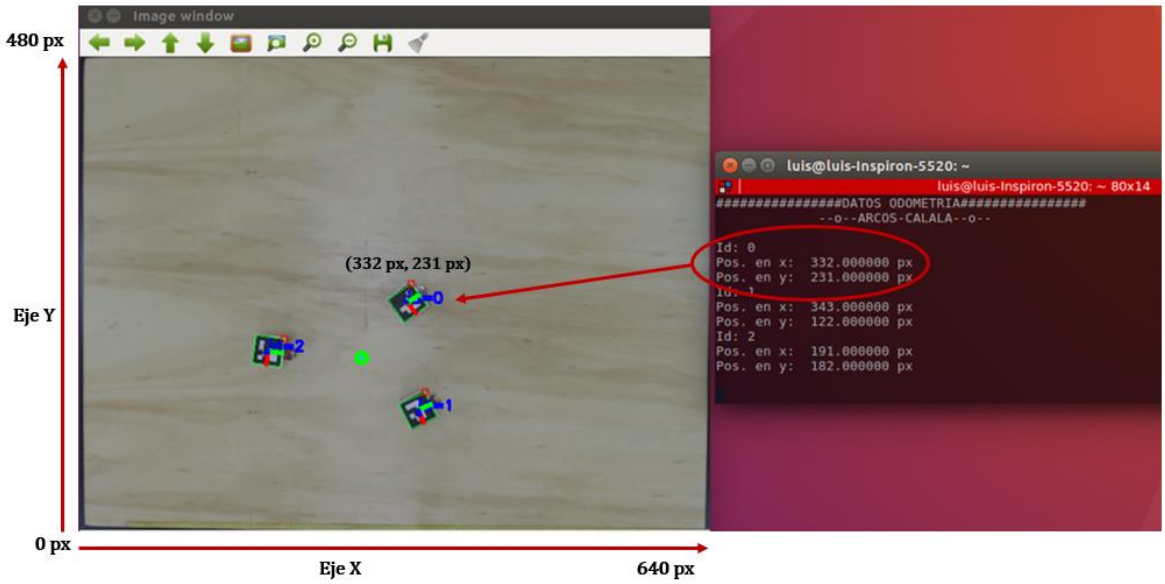


Figura 2.46 Ubicación de marcadores de referencia en coordenadas 2D

Para establecer esta equivalencia se coloca sobre el área de trabajo un objeto con dimensiones conocidas, en este caso se usó una regla de 30 cm y se la colocó sobre dos marcadores de referencia con sus esquinas en el centro de cada marcador. Luego, se toman los valores en pixeles de cada esquina de la regla como se observa en la Figura 2.47, se restan los valores de las posiciones en el eje X obteniendo 127 pixeles y mediante una regla de tres simple se obtiene la relación de un píxel en metros.

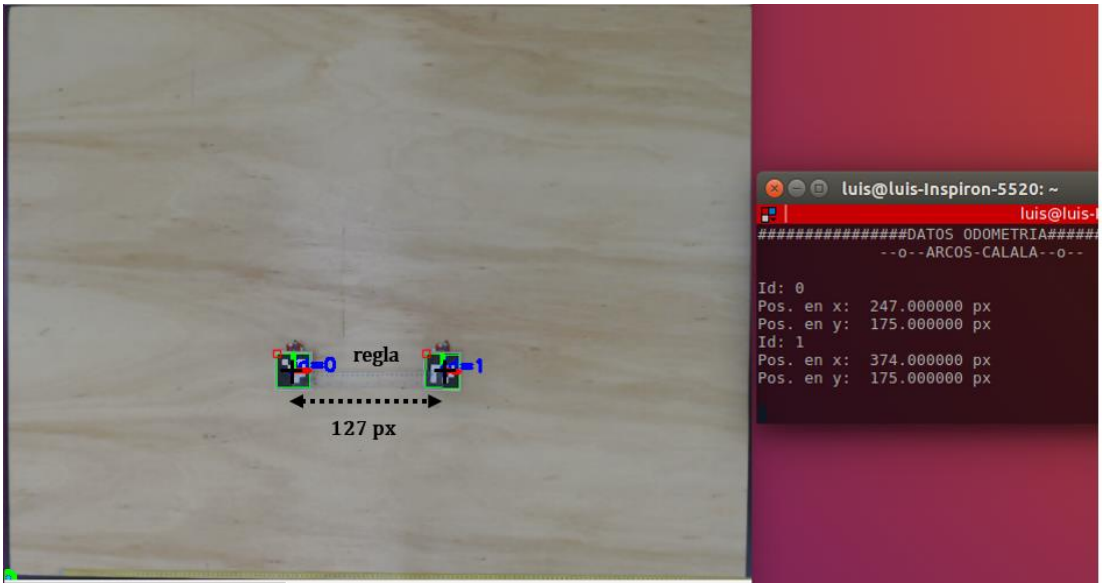


Figura 2.47 Equivalencia de medida de la regla en pixeles.

Como se muestra en la Ecuación (2.22), X representa la equivalencia de 1 pixel en metros.

$$X = \frac{0.3[m]*1[px]}{127\ [px]} = 0.00236\ [m] \tag{2.22}$$

En la Figura 2.49 se puede observar las posiciones de los tres marcadores de referencia en metros y además el ángulo de dirección de cada marcador. El ángulo de dirección del marcador de referencia se lo obtiene mediante la función arco tangente (ATan2) de la librería math.h de C++, la cual devuelve el ángulo θ entre la recta r que une el origen de coordenadas con un punto (x,y) y el eje positivo x, limitado al rango [-π, π], como se observa en la Figura 2.48.

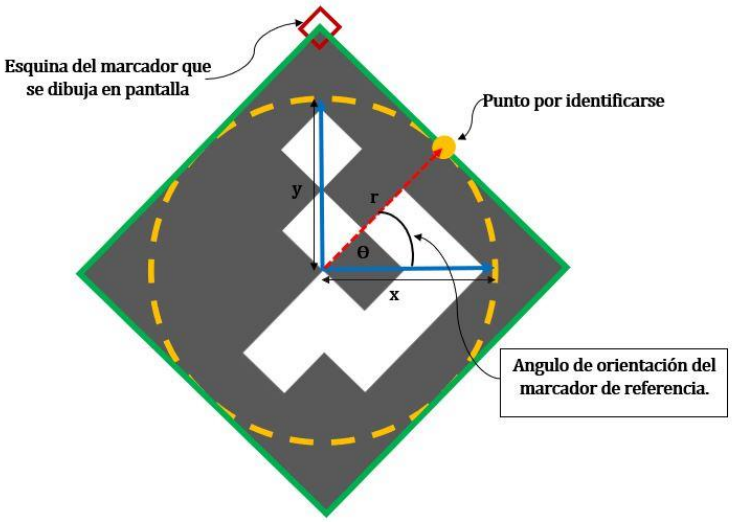


Figura 2.48 Obtención del ángulo de dirección del marcador de referencia.

Además, la función *estimatePoseSingleMarkers* permite dibujar sobre los marcadores los ejes x, y, z y su respectivo identificador. Estos datos junto con el ángulo de orientación son utilizados por el controlador.

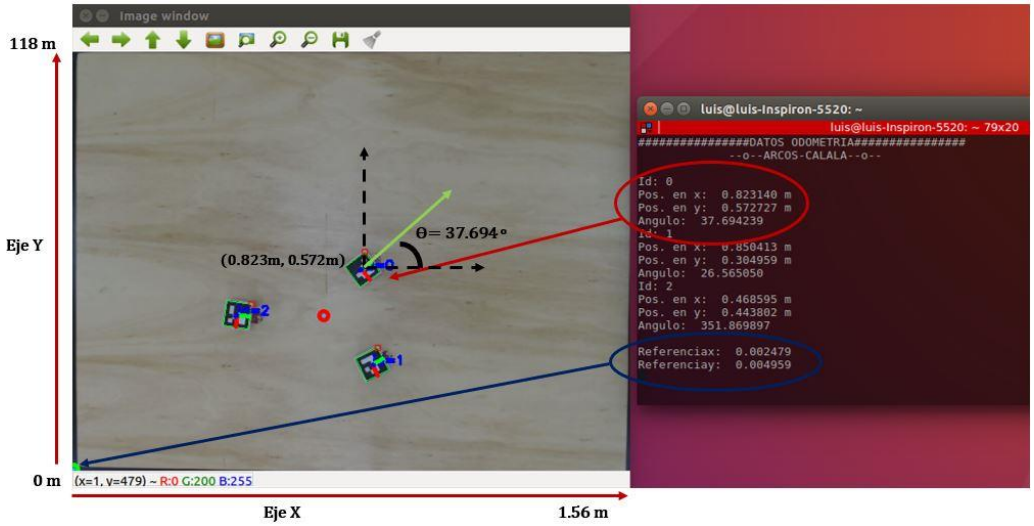


Figura 2.49 Aproximación de pixeles a metros en el área de trabajo.

El proceso de detección de marcadores explicado anteriormente se resume en el diagrama de flujo esquematizado en la Figura 2.50.

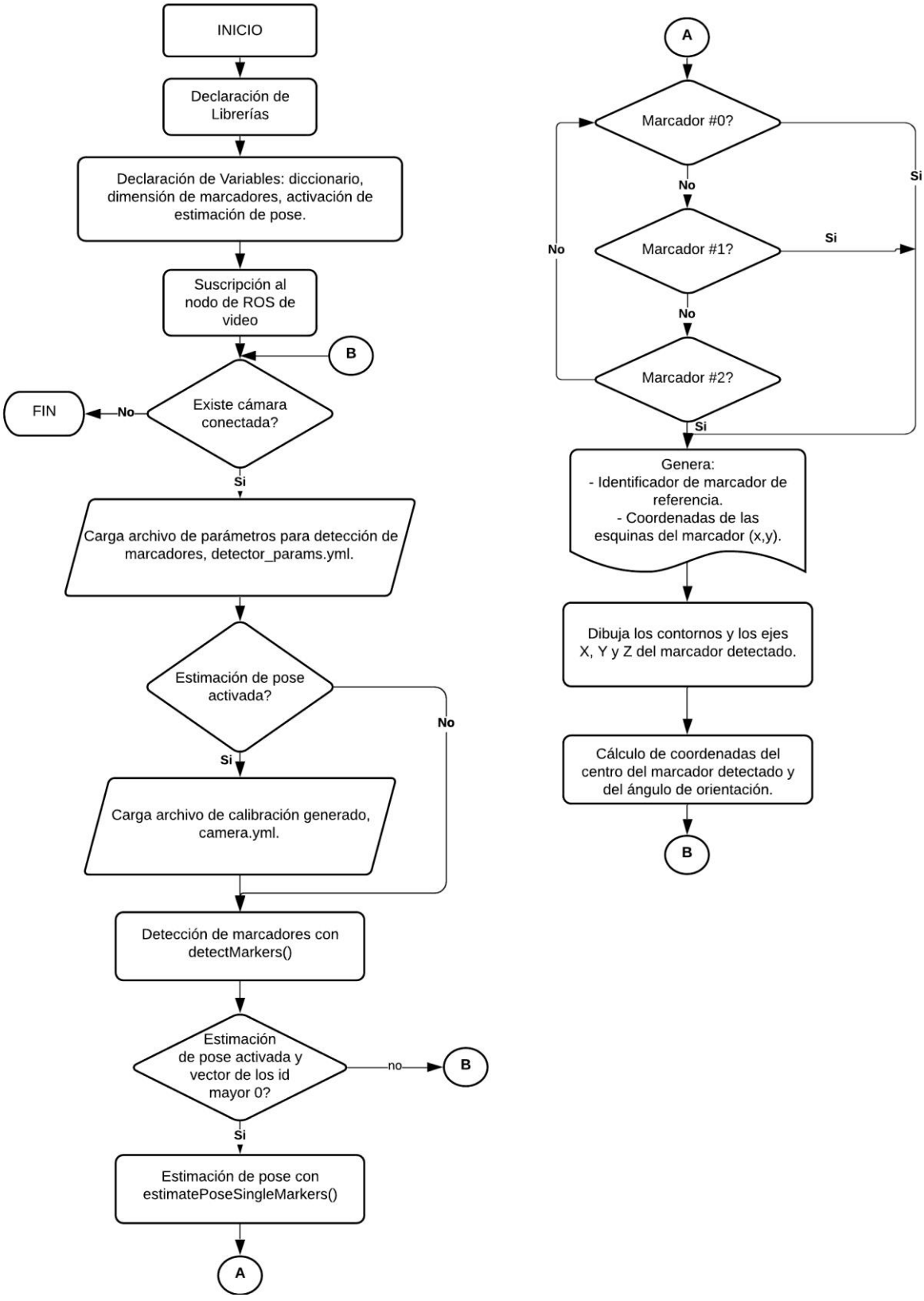


Figura 2.50 Diagrama de flujo de detección de marcadores.

Implementación de algoritmos de control

En la sección de desarrollo de algoritmos de control se obtuvieron las ecuaciones tanto para el controlador interno de velocidad como para el controlador externo de formación y postura. En el presente proyecto el controlador interno es implementado en la tarjeta Fio V3 Arduino de cada robot móvil y el controlador externo es implementado en el ordenador sobre el entorno de ROS en lenguaje de programación Python.

Implementación del controlador interno

El controlador interno tiene como finalidad realizar el control de velocidad de las dos ruedas de los robots móviles. Las velocidades de referencia de los motores de los robots móviles son recibidas por medio de un módulo Xbee.

Implementación del sensor de velocidad

La medición de velocidad se hace a través del enconder de cuadratura descrito en la sección anterior, para su uso se crea una librería que contiene las declaraciones de las funciones y variables necesarias. La lectura de datos del enconder se lo puede observar en el diagrama de flujo esquematizado en la Figura 2.51.

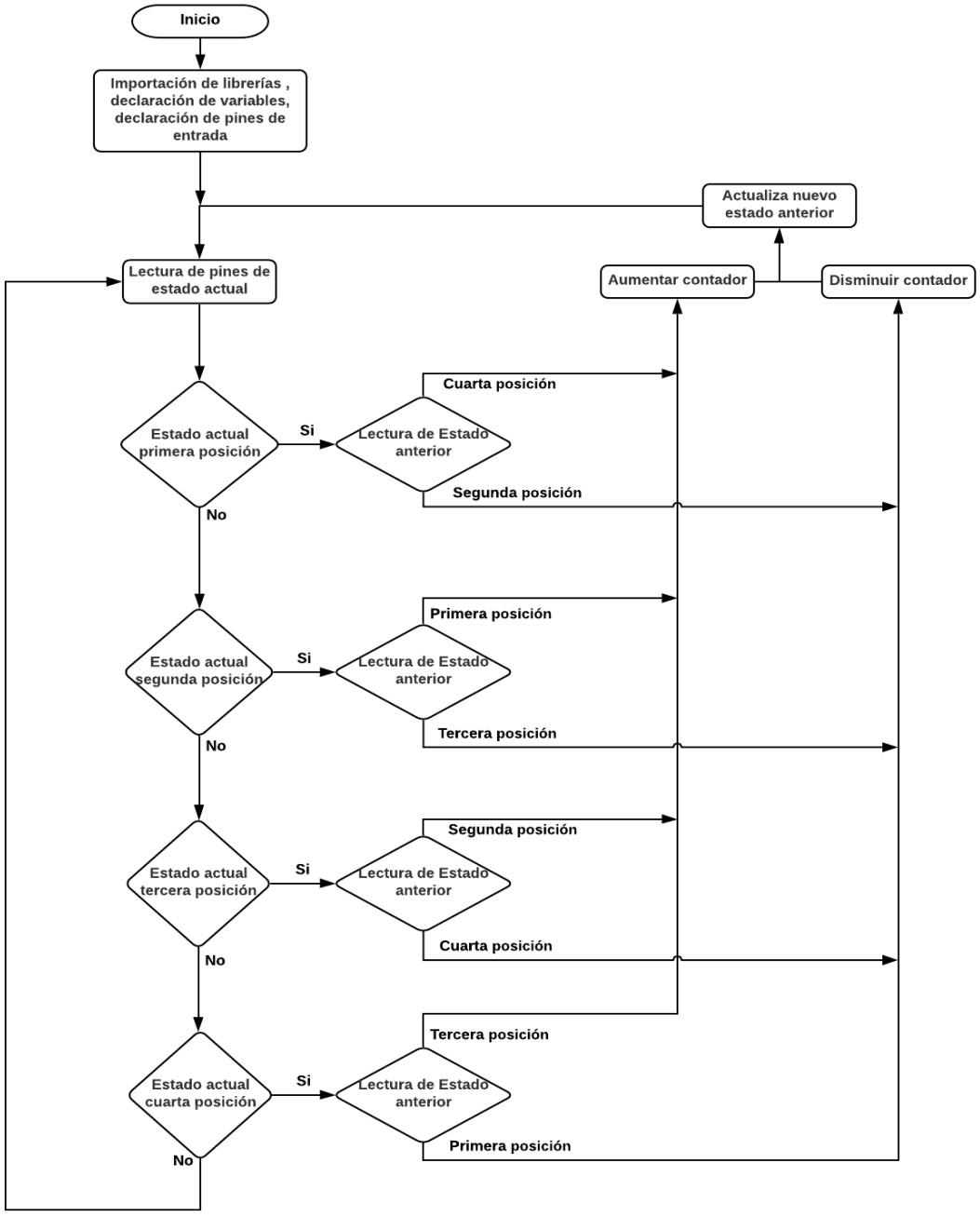


Figura 2.51 Diagrama de flujo del funcionamiento de librería del encoder.

Con esta librería se cuenta el número de pulsos en base a los cuatro estados que tiene el encoder de cuadratura. La resolución final se lo calcula en base al número de pulsos entregados por el encoder y la relación de la caja reductora de los motores como indica la Ecuación 2.23.

$$\text{Resolución } = 12(\text{PPR encoder}) \text{ x } 120(120{:}\,1 \text{ Caja Reductora}) \qquad (2.23)$$

$$\text{Resolución} = 1440\,\frac{pulsos}{rev}$$

Este es el número de pulsos por cada revolución de una rueda del robot y con el cual se puede realizar la medición de velocidad. La acumulación o disminución del número de pulsos se da cada vez que se detecta una interrupción por parte de los pines del Arduino conectados al encoder. Además, la medición de velocidad emplea un Timer de Arduino el cual se configura para generar una interrupción cada $\frac{1}{20}$ de segundo. En esta interrupción se contabiliza el número de pulsos contados por el encoder durante este periodo de tiempo y se transforma a revoluciones por minuto.

Esta transformación se la describe mediante la Ecuación 2.24.

$$\#\text{Rev}_{1/20} = \#\text{pulsos} \times \frac{1[\text{ rev}]}{1440[pulsos]} \tag{2.24}$$

$$\text{RPS} = \#\text{Rev}_{\frac{1}{20}} \times 20$$

$$\text{RPM} = \text{RPS} \times 60$$

donde:

- $\#\text{Rev}_{1/20}.-$ Numero de revoluciones en $\frac{1}{20}$ de segundo.
- $\text{RPS}.-$ Revoluciones por segundo.
- $\text{RPM}.-$ Revoluciones por minuto.

La operación determina el número de revoluciones en la fracción de 1/20 de segundo en base a la relación de numero de pulsos por cada revolución. Luego de esto se multiplica por 20 para determinar el número de revoluciones por segundo para posteriormente multiplicar por 60 para determinar el número de revoluciones por minuto. Por tanto, la actualización del valor de velocidad se da cada 0.05 segundos en el lazo de control.

Implementación controlador PID de velocidad

El control de velocidad implementado tiene una estructura PID, donde se tiene como referencia las velocidades en rpm de la llanta izquierda y derecha, ésta es recibida por la tarjeta Xbee.

El controlador se ejecuta de manera continua una vez recibida la orden por parte del Xbee coordinador. La acción de control obtenida del controlador se relaciona con la modulación de ancho de pulso o PWM la cual permite el movimiento de los motores.

El funcionamiento general del controlador de velocidad se lo puede observar en el diagrama de flujo de la Figura 2.52.

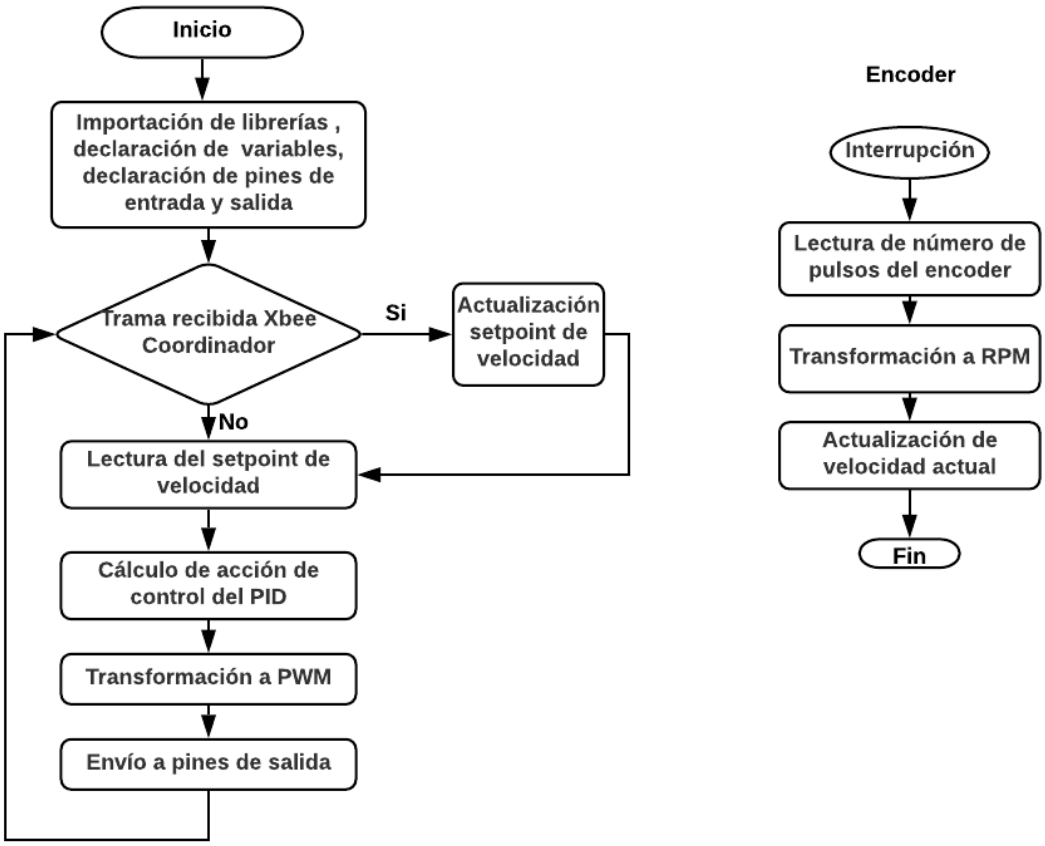


Figura 2.52 Diagrama de flujo de funcionamiento del controlador de velocidad.

En base a pruebas se determina los límites de velocidad de respuesta del controlador de velocidad las cuales serán detalladas en el siguiente capítulo.

Implementación del controlador externo

El controlador externo de formación y postura es implementado en el nodo *unidad_central* de acuerdo con la Ecuación 2.19 y su funcionamiento está descrito en el diagrama de flujo de la Figura 2.53.

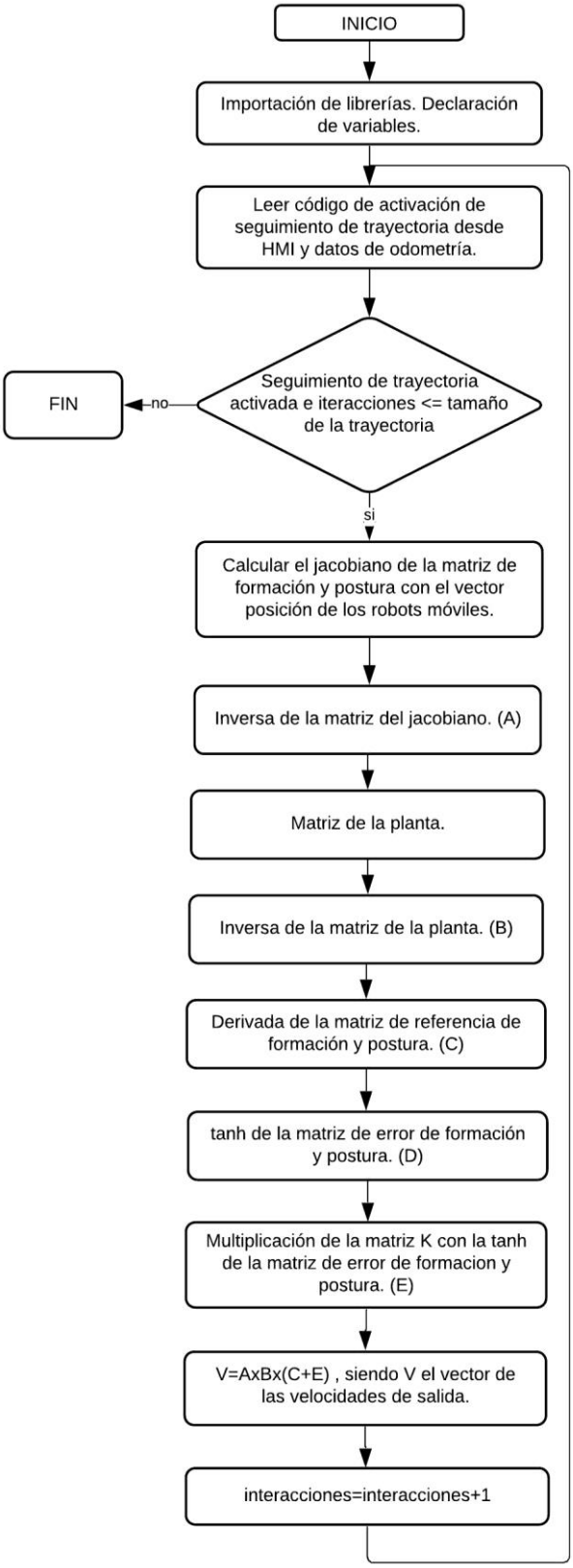


Figura 2.53 Diagrama de flujo del funcionamiento del controlador de inversión no lineal.

Implementación del sistema de comunicación

Como se explicó en el primer capítulo el modo de funcionamiento de las tarjetas Xbee es API con escape. Para la configuración de las tarjetas Xbee se utiliza el software de uso libre XCTU el cual permite detectar y configurar las tarjetas Xbee. Para esto se utiliza el módulo Xbee Explorer de Sparkfun [44], el cual se observa en la Figura 2.54 que permite acoplar las tarjetas XBEE.

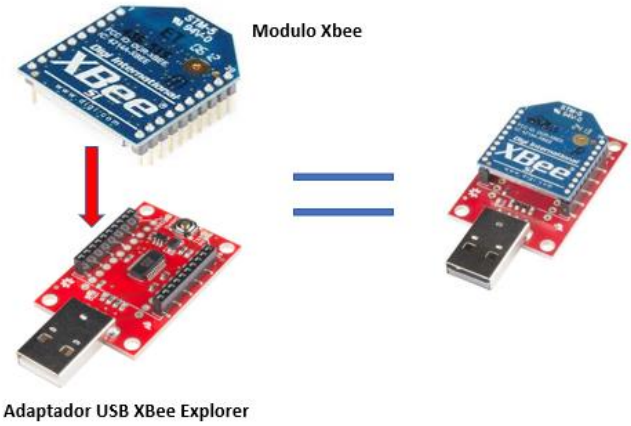


Figura 2.54 Adaptador USB Xbee Xplorer.

Una vez conectado el adaptador en el ordenador, se procede a su detección mediante el programa XCTU, el cual muestra el puerto activo como se observa en la Figura 2.55.

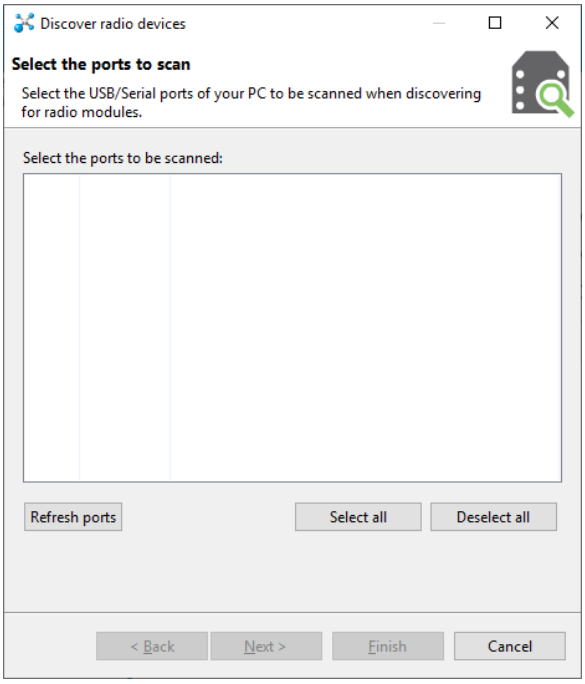


Figura 2.55 Ventana de búsqueda de puertos activos.

En el sistema implementado se tiene dos configuraciones base, la configuración Xbee Coordinador la cual está conectada al ordenador y maneja el envío de las tramas del nodo *Response_robots* y la configuración de las Xbee End Device que son las Xbees conectadas a los robots.

La topología empleada en el sistema es punto-multipunto, ya que se tiene un coordinador y tres módulos End device. Para el funcionamiento es necesario configurar los siguientes parámetros.

- PAN ID. - Identificador de red.
- CH. –Canal de comunicación.
- MY. –Dirección de origen.
- DL y DH. -Dirección de Destino.

El *PAN ID* y *CH* deben tener los mismos valores para crear la red Xbee, mientras que la de dirección de origen MY es la dirección de cada Xbee, en el caso del coordinador esta debe tener el valor de 0.

DL Y DH son las direcciones de destino siendo de 64 o 16 bits. Para el sistema empleado se ha utilizado el direccionamiento de 16bits por lo que solo se emplea DL. La red punto-multipunto se la esquematiza en la Figura 2.56.

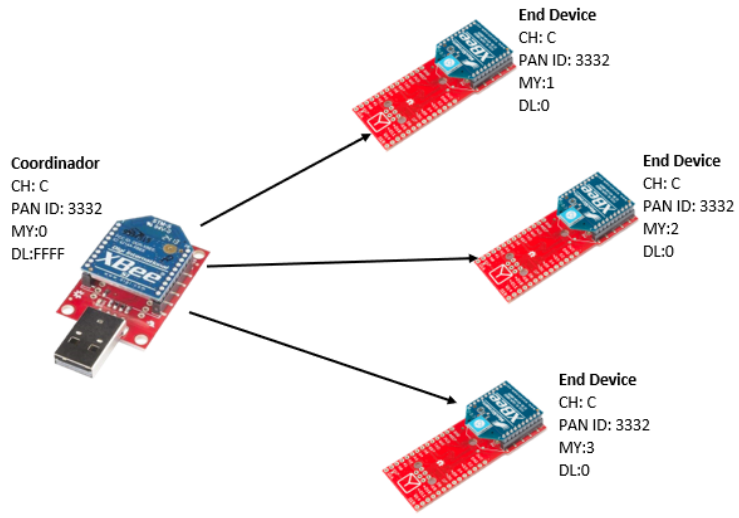


Figura 2.56 Esquema de configuración red Punto-Multipunto.

En la Figura 2.57 y Figura 2.58 se observa los parámetros que se puede modificar en los módulos Xbee mediante el software XCTU. Es importante seleccionar el protocolo de comunicación API.

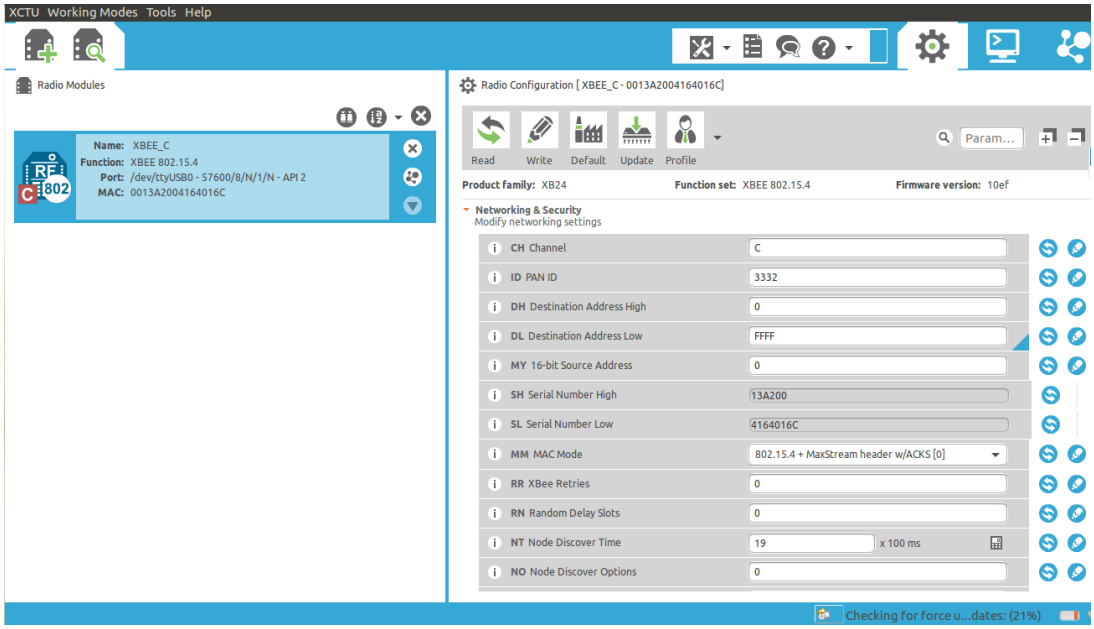


Figura 2.57 Configuración de Xbee en XCTU.

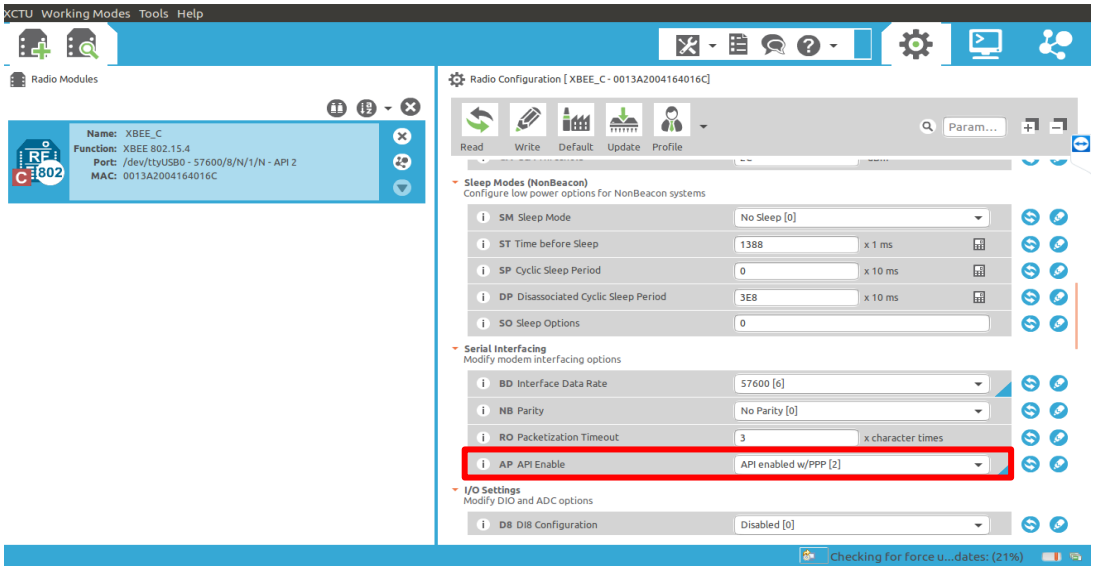


Figura 2.58 Configuración de protocolo API en XCTU

Además, en esta sección se selecciona la velocidad de comunicación a 57600 baudios, la cual está configurada tanto en el nodo *Response_robots* como en el programa de Arduino.

El parámetro CE determina si una Xbee es coordinador o End device. En caso de ser coordinador se coloca el valor de 1 caso contrario 0. En la Figura 2.59 se encuentra marcada la configuración de este parámetro.

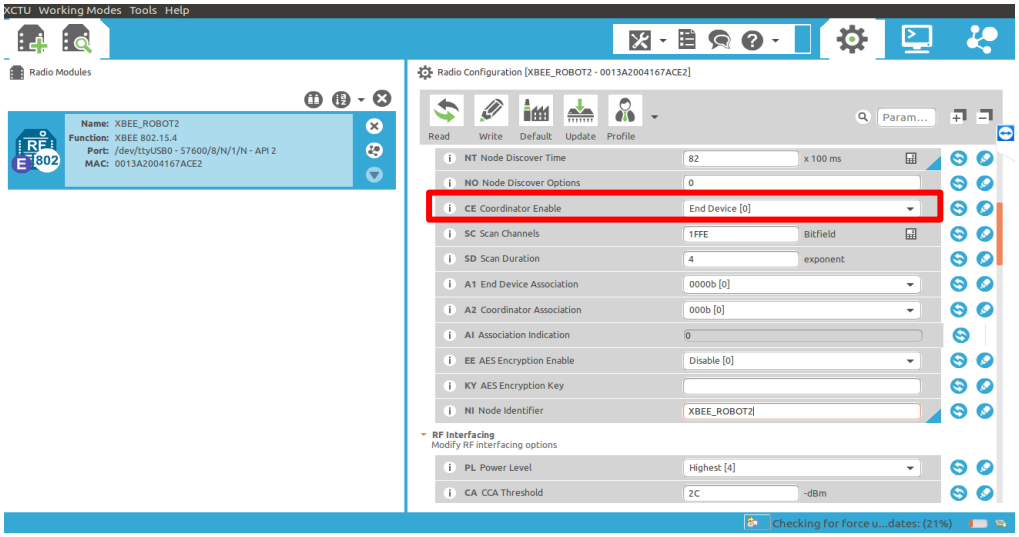


Figura 2.59 Configuración CE en XCTU

Estas son las configuraciones principales realizadas en las tarjetas Xbee para que las librerias usadas para el manejo de las Xbee tanto del nodo *Response_robots* y el programa de Arduino, el cual se detalla a continuacion

Comunicación con tarjeta Arduino

El sistema de comunicación de los robots es a través de una tarjeta Xbee la cual se encuentra conectada a la tarjeta de Arduino. Para la utilización de esta tarjeta en modo API con escape se utilizó la librería Xbee de Arduino, con la cual se puede verificar si existe una trama enviada por el coordinador.

Dependiendo de la trama recibida el robot puede realizar tres acciones diferentes, estas son:

- Cambio de constantes del PID de velocidad
- Inicio del controlador de velocidad y envió de velocidades actuales.
- Envió del estado actual del robot

El estado actual del robot corresponde a las constantes PID y el nivel de la batería del robot. El voltaje de la batería puede ser leído mediante el pin analógico A0.

El controlador de velocidad funciona continuamente una vez recibida la solicitud por parte de la Xbee coordinador, al recibir la trama esta contiene el setpoint de velocidad. Una vez leído y actualizado el setpoint, se envía el valor de velocidades actuales del robot al Xbee Coordinador.

A continuación, en la Figura 2.60 se presenta el diagrama de flujo de las acciones del robot de acuerdo con las tramas leídas por el módulo Xbee.

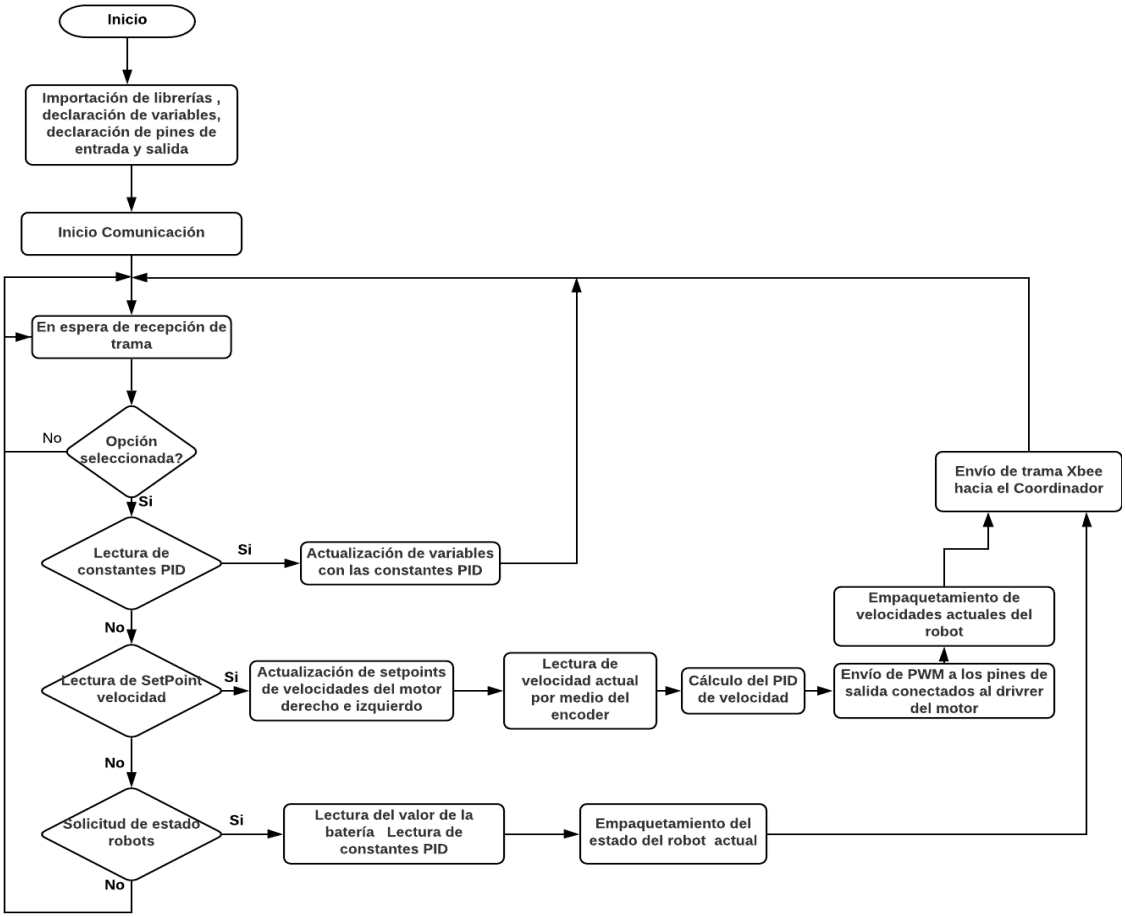


Figura 2.60 Diagrama de flujo de funcionalidades del robot.

Es necesario en este punto la realización de una interfaz gráfica que permita a los usuarios acceder a todas estas opciones desde un solo un punto. Por lo cual a continuación se detalla la implementación de interfaz gráfica en Qt Creator.

Implementación Interfaz Gráfica (Qt Creator)

Para la visualización y monitoreo del sistema se diseñó una interfaz gráfica desarrollada en GUIs (interfaces gráficas de usuario) en entorno de Qt Creator. Estas interfaces se componen de un archivo source (donde se realiza la programación de todos los objetos del GUI), header (donde se inicializan las variables globales y se declaran funciones) y un form (donde se diseña gráficamente las ventanas). En la Tabla 2.4 se detallan las ventanas diseñadas en el sistema.

Tabla 2.4 Archivos generados para la interfaz gráfica del proyecto.

Forms	Función	Source	Headers
inicio.ui	Ventana inicial.	inicio.cpp	inicio.h
tutorial.ui	Ventana en donde se observa un pequeño tutorial sobre el manejo del sistema.	tutorial.cpp	tutorial.h
mainwindow.ui	Ventana principal de la interfaz.	mainwindow.cpp	mainwindow.h
informacion.ui	Ventana en donde se observa el estado de batería y las constantes de los controladores internos de los robots móviles.	informacion.cpp	informacion.h
graficos.ui	Ventana en donde se observa el seguimiento de trayectoria, las velocidades y la posición de la formación.	gráficos.cpp	gráficos.h
		qcustomplot.cpp	qcustomplot.cpp
graficas_errores.ui	Ventana que despliega gráficas de posiciones en el tiempo del seguimiento de la trayectoria.	graficas_errores.cpp	graficas_errores.h
		qcustomplot.cpp	qcustomplot.cpp
graficas_velocidad.ui	Ventana que despliega gráficas de velocidad del robot móvil 1.	graficas_velocidad.cpp	graficas_velocidad.h
		qcustomplot.cpp	qcustomplot.cpp

graficas_velocidad_1.ui	Ventana que despliega gráficas de velocidad del robot móvil 2.	graficas_velocidad_1.cpp	graficas_velocidad_1.h
		qcustomplot.cpp	qcustomplot.cpp
graficas_velocidad_2.ui	Ventana que despliega gráficas de velocidad del robot móvil 3.	graficas_velocidad_2.cpp	graficas_velocidad_2.h
		qcustomplot.cpp	qcustomplot.cpp
acerca.ui	Ventana que despliega información de los autores del proyecto.	acerca.cpp	acerca.h

Ventana inicial (inicio.ui)

Al iniciar la aplicación aparece una ventana de inicio presentada en la Figura 2.61. En la cual se presentan dos botones:

- **Información**. – Detalla un resumen acerca del proyecto y un breve tutorial del manejo de la aplicación.
- **Continuar. –** Permite ingresar y navegar en las demás ventanas.

Figura 2.61 Ventana de presentación de Interfaz Gráfica (inicio.ui)

Una vez presionado el botón “Información” se despliega la ventana de la Figura 2.62 y para regresar a la pantalla de inicio se presiona el botón “Cerrar”.

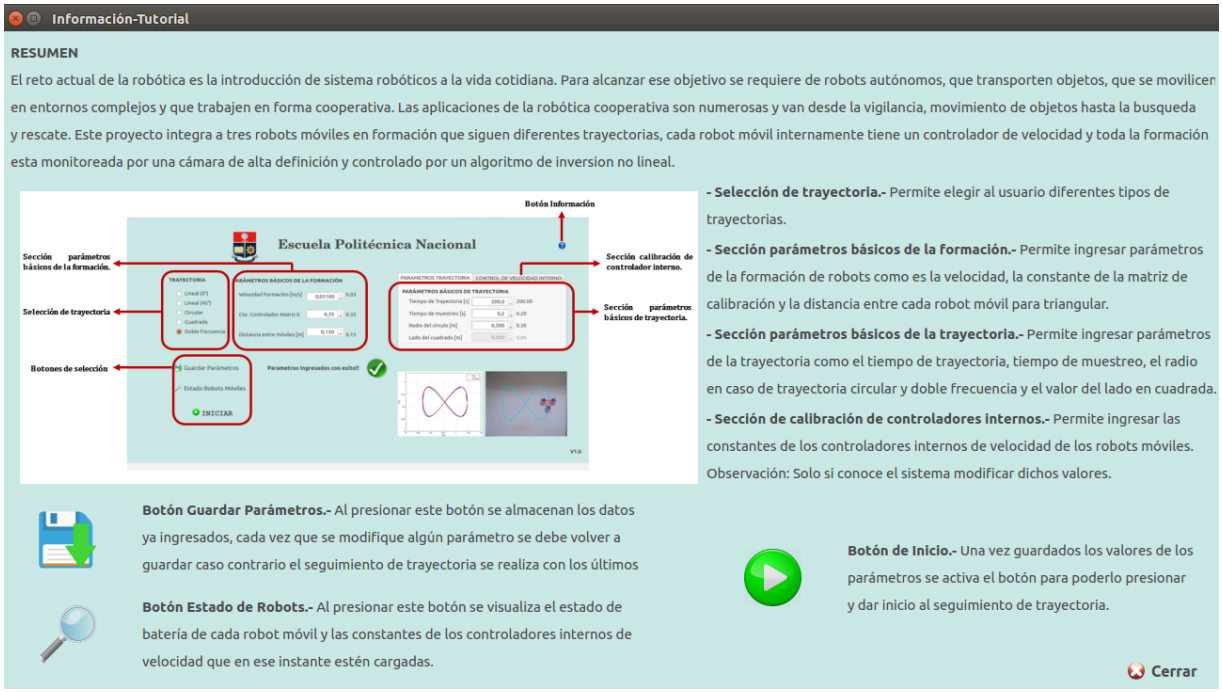


Figura 2.62 Ventana de resumen de proyecto y tutorial de manejo(tutorial.ui)

De vuelta en la pantalla de inicio, al presionar el botón “Continuar” se despliega la ventana principal.

Ventana principal (mainWindow.ui)

Como se observa en la Figura 2.63, en la ventana principal se encuentran todas las opciones de configuración para el funcionamiento del sistema.

Figura 2.63 Ventana principal de la interfaz gráfica (mainwindow.ui)

En la ventana se encuentran las siguientes partes:

- **Selección de trayectoria**. – Permite elegir al usuario diferentes tipos de trayectorias, entre ellas están trayectorias lineales, lineales con pendiente de 45º, circular, cuadrada y doble frecuencia.
- **Sección parámetros básicos de la formación. -** Permite ingresar parámetros de la formación de robots como la velocidad, la constante de la matriz de calibración y la distancia entre cada robot móvil.
- **Sección parámetros básicos de la trayectoria. –** Permite ingresar parámetros de la trayectoria como el tiempo de trayectoria, tiempo de muestreo, el radio en caso de trayectoria circular y doble frecuencia y el valor del lado en trayectoria cuadrada.
- **Sección de calibración de controladores internos. –** Permite ingresar las constantes de los controladores internos de velocidad de los robots móviles.
- **Botón de Guardar Parámetros. –** Al presionar este botón se almacenan los datos ya ingresados, cada vez que se modifique algún parámetro se debe volver a guardar caso contrario el seguimiento de trayectoria se realiza con los últimos valores guardados.
- **Botón de Estado de Robots. –** En la Figura 2.64 se observa la ventana de estado de robots móviles (informacion.ui), donde el usuario visualiza constantes de los controladores internos que en ese instante estén cargadas y el estado de batería de cada robot móvil. Existen diferentes niveles de batería.
    - Batería parcialmente llena. - Valor de batería mayor o igual a 3.7 voltios.
    - Batería media. - Valor de batería entre 3.4 y 3.57 voltios.
    - Batería baja. - Valor de batería menor a 3.4 voltios.

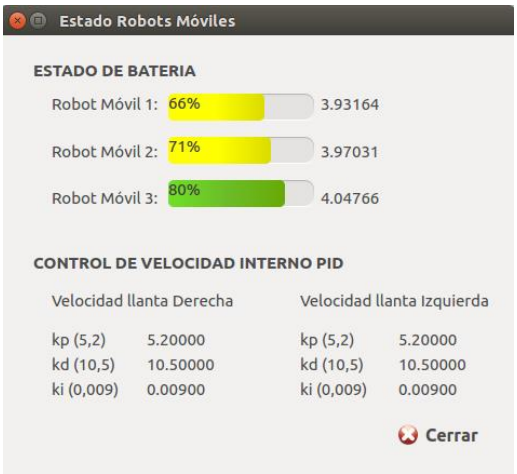


Figura 2.64 Ventana Estado de los robots (informacion.ui)

- **Botón de Inicio. –** Una vez guardados los valores de los parámetros se activa el botón de Inicio, el cual abre la ventana de seguimiento de trayectoria.
- **Barra de menú.** – En esta barra se encuentra las opciones ayuda e información.

El funcionamiento de esta ventana esta descrito por el diagrama de flujo de la Figura 2.65.

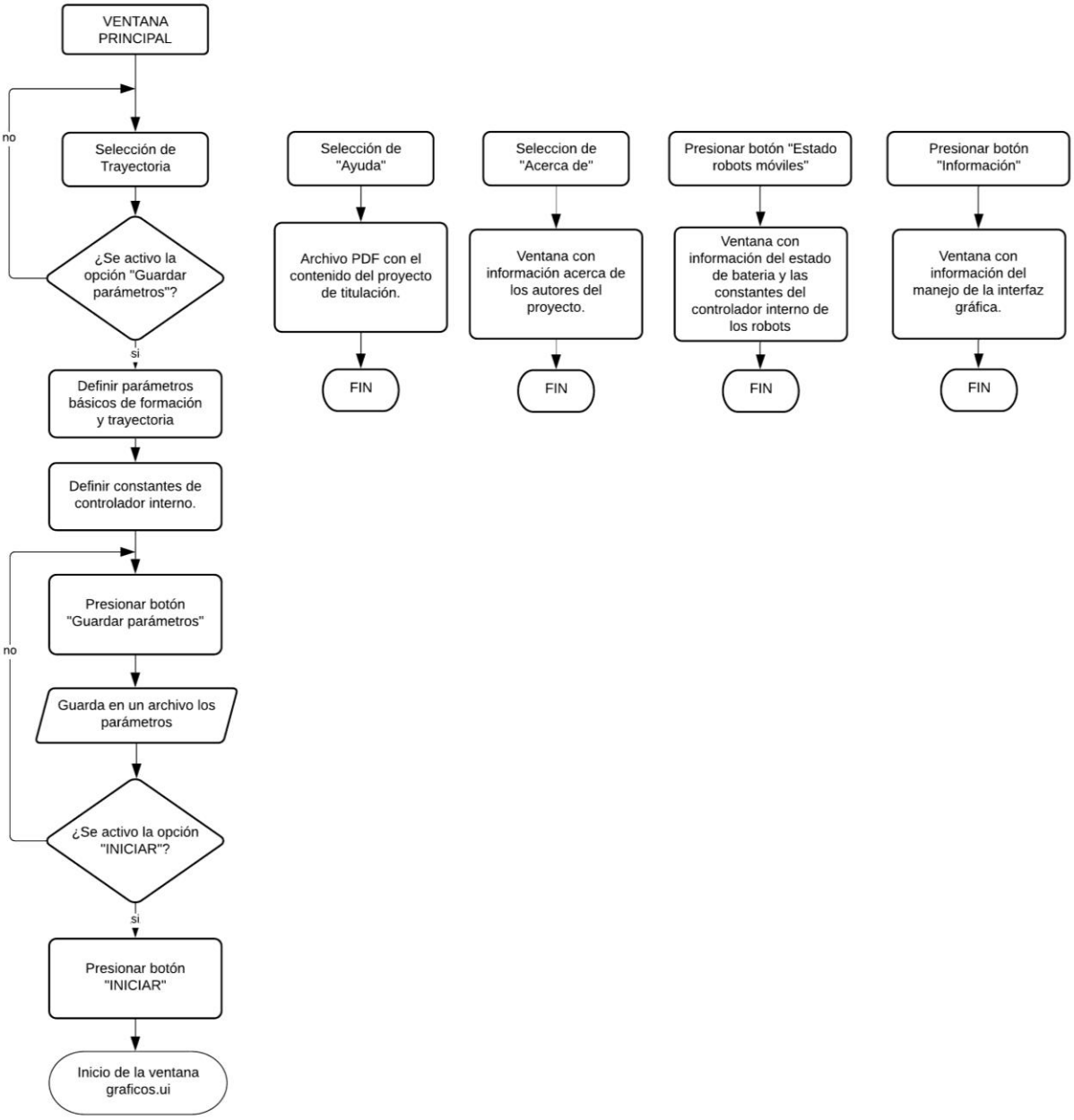


Figura 2.65 Diagrama de flujo de la ventana principal

Ventana de seguimiento de trayectoria (gráficos.ui)

Al iniciar el seguimiento de trayectoria, aparece una ventana de visualización como se observa en la Figura 2.66. En esta ventana se puede visualizar en una gráfica el movimiento de la formación de robots, la información de las velocidades de los robots móviles y la posición del centro de la formación. Su funcionamiento se describe en el diagrama de flujo de la Figura 2.67.

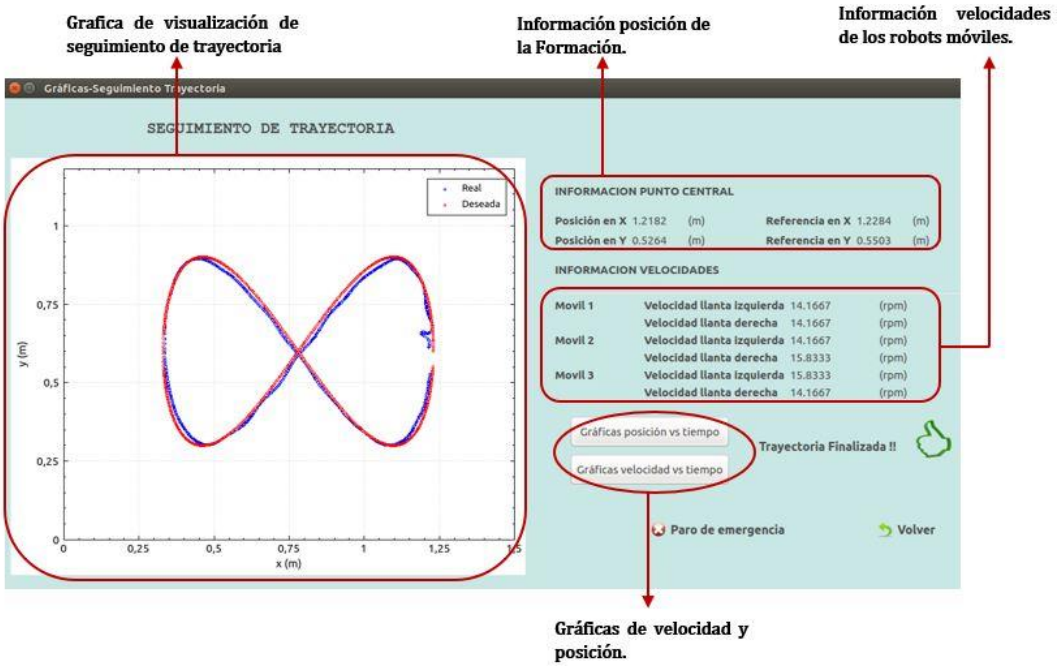


Figura 2.66 Ventana de Seguimiento de Trayectoria

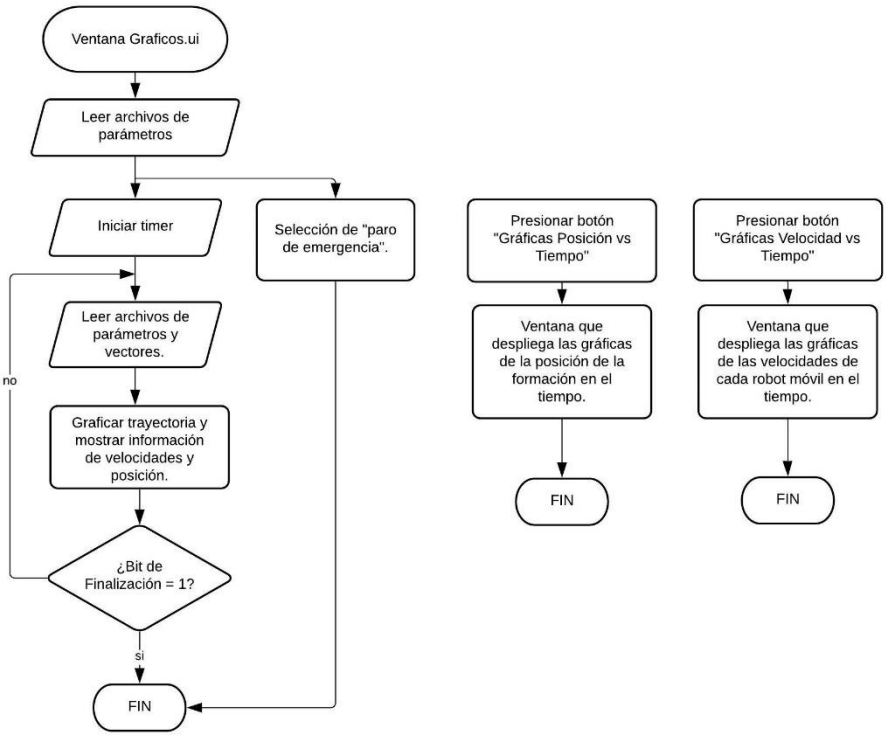


Figura 2.67 Diagrama de flujo de la ventana de seguimiento de trayectoria

Ventanas de Análisis de Gráficos

Una vez finalizado el seguimiento de trayectoria se pueden observar las gráficas de los resultados de las posiciones del centro de la formación y las velocidades de los robots en el tiempo, con sus respectivas referencias para su interpretación y análisis. El caso mostrado en la Figura 2.68, es el de visualización de posición de la formación de robots. En esta ventana se despliegan dos gráficas con sus respectivas leyendas de valores actuales y de referencia tanto para posición en X y posición en Y.

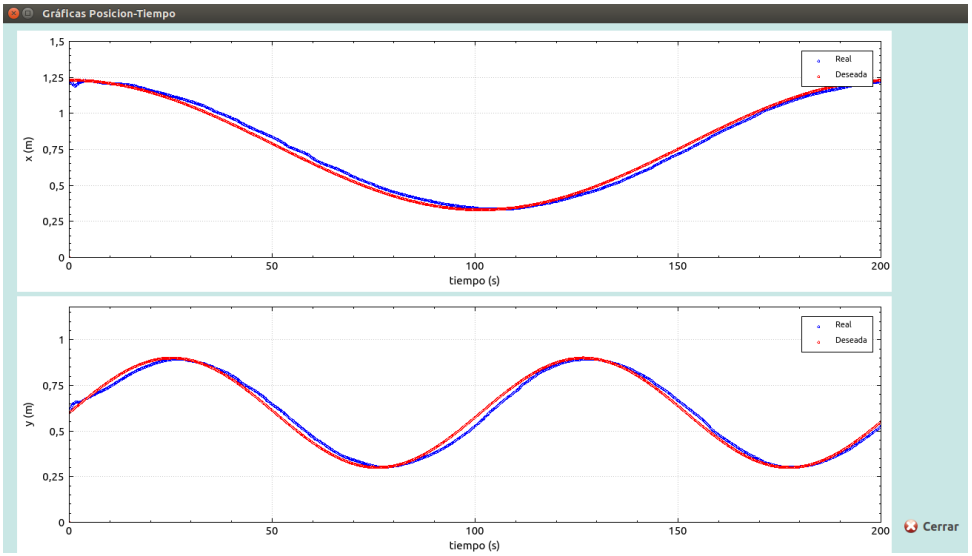


Figura 2.68 Ventana de Gráficas Posición – Tiempo

De igual manera como se observa en la Figura 2.69, en otra ventana se despliegan las gráficas de velocidad de cada rueda de los robots con sus respectivas referencias. Con el botón “Siguiente” se puede navegar entre las gráficas de cada robot móvil.

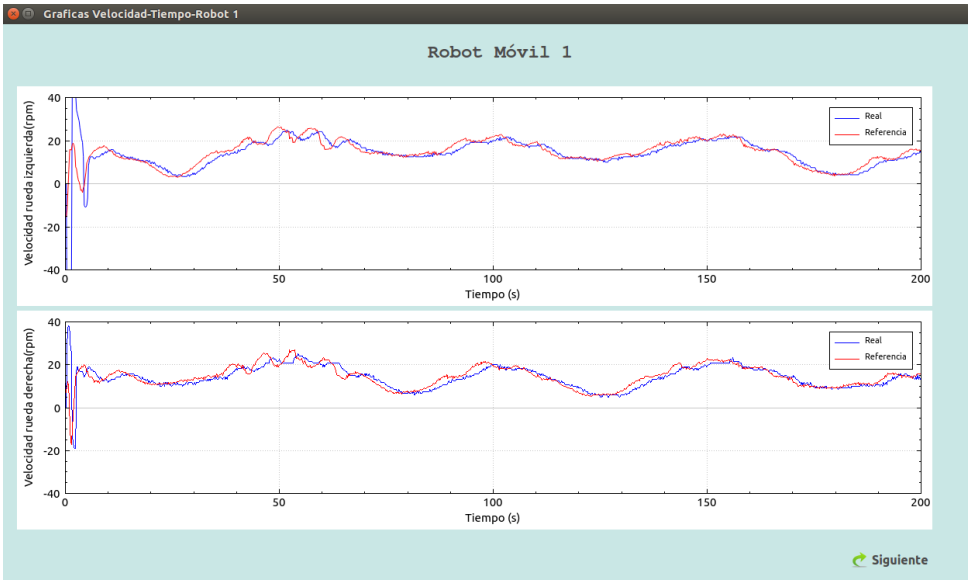


Figura 2.69 Ventana de Gráficas Velocidad – Tiempo (Robot móvil 1)

3. RESULTADOS Y DISCUSIÓN

En esta sección se presentan las diferentes pruebas realizadas para comprobar el desempeño del sistema, y estas se enfocan en lo siguiente:

- Pruebas de comunicación entre los robots y el ordenador mediante los módulos Xbee.
- Calibración del tiempo de latencia.
- Calibración y pruebas del control interno de velocidad.
- Seguimiento de trayectoria circular con diferentes ganancias de control a una velocidad predeterminada de 0.09 (m/s).
- Seguimiento de trayectorias circular, cuadrada y doble frecuencia a diferentes velocidades.
- Respuesta a perturbaciones en el seguimiento de trayectoria.

Para los casos de validación del desempeño del controlador en cascada (interno más externo) se efectúan pruebas en base al seguimiento de trayectorias del centroide de la formación (x_c, y_c). Para lo cual se consideran las siguientes tres trayectorias: 1) circular, 2) cuadrada y 3) doble frecuencia, definidas por funciones en el tiempo que se encuentran definidas en la Tabla 3.1.

Tabla 3.1 Ecuaciones para el seguimiento de trayectoria.

Trayectorias	$\boldsymbol{x_c}$	$\boldsymbol{y_c}$
Circular	$r\cos\left(\frac{vel}{r}\ t\right) + 0.78$	$r\sin\left(\frac{vel}{r}\ t\right) + 0.59$
Cuadrada	L_1: $(vel\ t) + 0.47$ L_2: x' L_3: $x' - (vel\ t)$ L_4: 0.47	0.44 $(vel\ t) + 0.44$ y' $0.44 - (vel\ t)$
Doble Frecuencia	$1.5\ r\cos(vel\ t) + 0.78$	$r\sin(2\ vel\ t) + 0.59$

Donde los parámetros de las diferentes trayectorias son elegidos desde la interfaz gráfica, siendo el punto (0.47, 0.44) m el inicio de la trayectoria cuadrada, y el punto (0.78, 0.59) m el centro de la trayectoria circular y doble frecuencia. En la trayectoria cuadrada x', y' representan las posiciones luego de haber recorrido la distancia de un lado del cuadrado.

En la Figura 3.1 se muestra un gráfico explicativo de las trayectorias cuadrada y circular generadas mediante Open CV en el nodo *visión_artificial*.

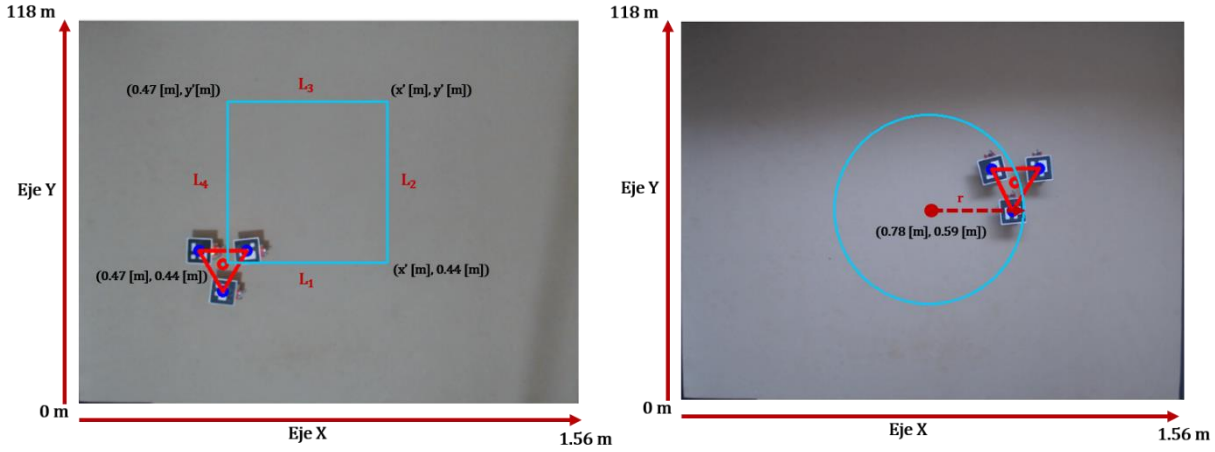


Figura 3.1 Trayectoria circular y cuadrada.

Adicionalmente, para el desempeño del controlador externo se utiliza como base la Ecuación 3.1 que corresponde a la norma del error.

$$e = \|q_{ref} - q\|$$

$$= \sqrt{\left(d_{1_{ref}} - d_1\right)^2 + \left(d_{2_{ref}} - d_2\right)^2 + \left(\beta_{ref} - \beta\right)^2 + \left(X_{c_{ref}} - X_{c_{act}}\right)^2 + \left(Y_{c_{ref}} - Y_{c_{act}}\right)^2 + \left(\varphi_{ref} - \varphi\right)^2} \qquad (3.1)$$

3.1 PRUEBAS DEL SISTEMA DE COMUNICACIÓN

Esta prueba tiene como objetivo validar el sistema de comunicación entre la Xbee Coordinador de la computadora y las Xbee End Device de los robots móviles. El modo de comunicación empleado es API con escape, ya que se tiene una topología punto-multipunto. En este esquema, el Coordinador envía una trama a unos de los robots y se queda en espera de la respuesta de este. Cada robot está programado para responder al Coordinador.

La trama enviada desde el Coordinador contiene la dirección de destino y los datos empaquetados dentro de una estructura que dependerá de la solicitud requerida. Para el empaquetado se utiliza la librería *struct* dentro de Python, la cual crea estructuras de datos que, tienen los formatos mostrados en la Figura 3.2.

Envió de constantes PID

Opción	Kp Derecha	Ki Derecha	Kd Derecha	Kp Izquierda	Ki Izquierda	Kd Izquierda
0x0100	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000

Solicitud de velocidades actuales

Opción	Velocidad Derecha	Velocidad Izquierda
0x0200	0x0000	0x0000

Solicitud de estado actual

Opción
0x0300

Figura 3.2 Estructura de solicitudes creadas en Python.

Estas estructuras son enviadas por el coordinador de manera que los robots dependiendo de la opción seleccionada, responderán con la estructura mostrada en la Figura 3.3.

Respuesta del robot

Datos	RSSI	Dirección de Origen	Identificador	Opción
0x0000	0x0000	0x0000	0x0000	0x0000

Figura 3.3 Estructura de respuestas del robot.

Los datos de esta estructura contendrán los valores de velocidades actuales o del estado actual del robot; esta se desempaqueta obteniendo los valores requeridos. A continuación, se muestra la validación de las diferentes solicitudes mediante el uso del nodo *Response_robots*.

Prueba de solicitud estado de los robots

En esta prueba se envía como solicitud desde el nodo *Response_robots* el estado actual de los robots para lo cual se empaqueta la opción 3 de la Figura 3.2 y se la envía hacia el robot. El robot envía como respuesta las constantes PID del controlador interno de velocidad de cada rueda y el nivel de batería actual. A continuación, en la Figura 3.4, se muestra un ejemplo.

a)

```
[INFO] [1592608296.606046]: Estructura del dato Robot1: 0300
Tiempo inicio: 0.000000 [s]
Tiempo maximo de espera: 0.150000 [s]
Enviando a robot 1
En espera de respuesta robot 1
```

b)

```
Respuesta recibida!
Estructura de datos recibida:  {'rf_data': '\x00\x00 Bo\x12\x03<\x00\x00pA\x00\x00
 Bo\x12\x03<\x00\x00pA\x00\xa0{@', 'rssi': ':', 'source_addr': '\x00\x01', 'id': '
rx', 'options': '\x00'}
Direccion del origen de datos: Robot  1
Datos desempaquetados:  (40.0, 0.00800000037997961, 15.0, 40.0, 0.0080000003799796
1, 15.0, 3.931640625)
```

Figura 3.4 a) Trama enviada desde el nodo *Response_robots*, b) Respuesta del estado actual del robot.

Prueba de solicitud de velocidad actual

En esta prueba se verificó la solicitud de velocidad actual del robot enviada desde el nodo *Response_robots* para lo cual se empaqueta las referencias del controlador de velocidad empleando la opción 2 de la Figura 3.2. El robot respectivo envía como respuesta la velocidad actual de la rueda izquierda y derecha, un ejemplo se lo puede observar en la Figura 3.5.

a)

```
[INFO] [1592609006.734687]: Velocidades:[14.830309, 14.357369]
Empaquetar Velocidades
[INFO] [1592609006.735041]: Estructura del dato Robot1: 0200cb059b05
Tiempo inicio: 0.000000 [s]
Tiempo maximo de espera: 0.150000 [s]
Enviando a robot 1
En espera de respuesta robot 1
```

b)

```
Estructura de datos recibida:  {'rf_data': '\xab\xaabA\xab\xaabA', 'rssi': '>', 's
ource_addr': '\x00\x01', 'id': 'rx', 'options': '\x00'}
Direccion del origen de datos: Robot  1
Datos desempaquetados:  (14.166666984558105, 14.166666984558105)
```

Figura 3.5 a) Trama enviada desde el nodo *Response_robots,* b) Respuesta de velocidades actuales del robot.

Prueba de solicitud de cambio de constantes PID

La prueba consiste en enviar a través del nodo *Response_robots* empaquetadas las constantes PID hacia los robots, véase la opción 1 de la Figura 3.2. Para demostrar el cambio de estas ganancias, se envía como solicitud previa el estado actual para observar el cambio de las constantes. En la Figura 3.6, se observa un ejemplo del cambio de las constantes PID.

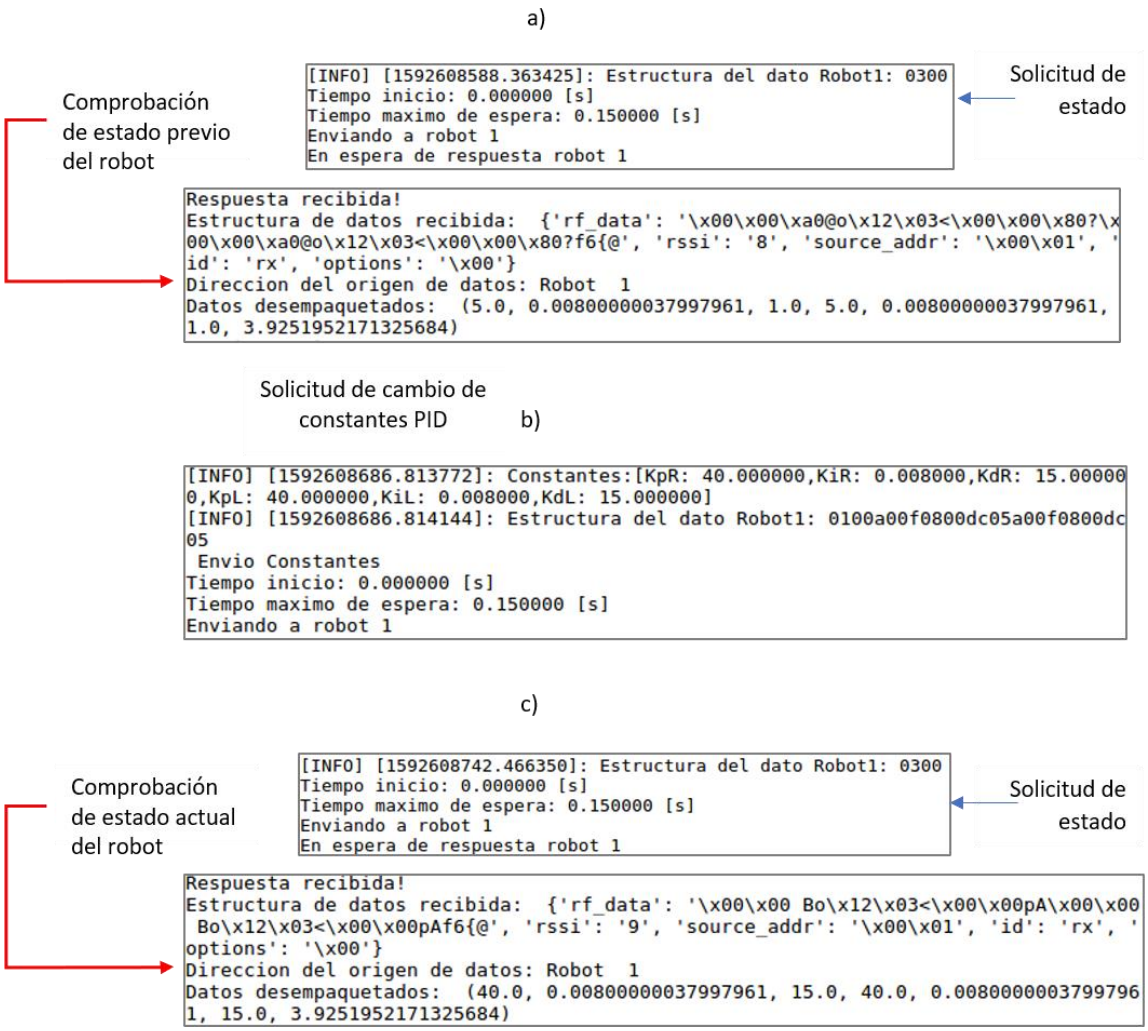


Figura 3.6 a) Estado previo del robot b) Trama enviada desde el nodo *Response_robots* c) Estado actual del robot.

Una vez comprobada la comunicación entre las Xbee del sistema es necesario determinar los tiempos de procesamiento y el retardo por la comunicación con el objetivo de definir el tiempo de latencia del sistema.

3.2 CALIBRACIÓN DEL TIEMPO DE LATENCIA

La calibración del tiempo de latencia se la realizó en base al tiempo que toma la comunicación y el procesamiento de las imágenes, lo cual está relacionado con el nodo encargado de la visión artificial.

Pruebas del tiempo de procesamiento

Para la determinación del tiempo de procesamiento se recopilaron 20 muestras diferentes del tiempo de lazo o bucle para el nodo de visión artificial, véase en la Tabla 3.2.

Tabla 3.2 Tiempos de lazo del nodo visión artificial

i	Tiempo [s]	i	Tiempo [s]
1	0.009863	11	0.013905
2	0.011019	12	0.017996
3	0.010609	13	0.018412
4	0.017556	14	0.010848
5	0.010636	15	0.010989
6	0.010472	16	0.009815
7	0.010119	17	0.010566
8	0.00995	18	0.011818
9	0.012567	19	0.010632
10	0.008344	20	0.010046

A partir de la Tabla 3.2 se utiliza la Ecuación 3.2 para obtener el promedio del tiempo de procesamiento del nodo visión artificial.

$$\bar{T}_{Lazo} = \frac{\sum_{i=1}^{n} Tiempo_i}{n} \tag{3.2}$$

El valor obtenido del tiempo de ejecución promedio del lazo en el nodo de visión artificial es 0.0118 segundos.

Pruebas del tiempo de comunicación

El tiempo de comunicación se basa en el tiempo que se demora en completar el envío de velocidades a todo el grupo de robots móviles. Este se determina por el tiempo de comunicación entre las Xbee Coordinador en el nodo *Response_robots* y la Xbee End Device en cada robot.

El inicio de cada interacción empieza cuando el nodo *Response_robots* envía la trama que contiene las velocidades del robot móvil r_1, y el robot envía la velocidad actual como respuesta al nodo. De esta manera pasa al robot móvil r_2 repitiendo el proceso, la interacción finaliza una vez que el nodo reciba las velocidades actuales del robot móvil r_3.

A continuación, se muestra la Tabla 3.3 que contiene el tiempo de duración de 20 interacciones diferentes:

Tabla 3.3 Tiempo de interacción de comunicación XBee

Interacción	Tiempo [s]	Interacción	Tiempo [s]
1	0.10308	11	0.10299
2	0.10327	12	0.09279
3	0.10360	13	0.09488
4	0.10278	14	0.09488
5	0.10354	15	0.10364
6	0.10334	16	0.10253
7	0.10323	17	0.10823
8	0.10359	18	0.10254
9	0.09379	19	0.10934
10	0.10585	20	0.10123

A partir de la Tabla 3.3 se utiliza la Ecuación 3.3 para obtener el promedio del tiempo de comunicación del grupo de robots móviles.

$$\bar{t}_{comunicación} = \frac{\sum_{i=1}^{n} \text{Tiempo}_i}{n} \tag{3.3}$$

El valor promedio obtenido es 0.101956 segundos. Por lo cual, el tiempo de latencia de todo el sistema es T_L es 0.1137 segundos obtenido al sumar el tiempo de procesamiento y el tiempo de comunicación, este valor es empleado para la generación de trayectorias y se encuentra por defecto en el HMI con el valor de 0.11 segundos.

En la determinación del tiempo de latencia no se considera el tiempo de envió de información entre los nodos de ROS ya que este retraso es despreciable.

Una vez determinado el tiempo que toma la adquisición de datos se procede a realizar las pruebas de los controladores.

3.3 PRUEBAS DEL CONTROL INTERNO DE VELOCIDAD

La determinación de las constantes del PID interno de los robots móviles para el control de la velocidad se la realizó en base a los resultados obtenidos en pruebas de seguimiento de velocidad para rampas de aceleración y cambios de referencia.

Las constantes empleadas en estas pruebas se muestran en la Tabla 3.4, donde cada grupo de constantes tendrá valores para k_p, k_d y k_i. A estos grupos de constantes se los define como sets.

Tabla 3.4 Constantes para pruebas del controlador interno de velocidad

	k_p	k_d	k_i
Set 1	2	5	0.008
Set 2	40	15	0.008
Set 3	5.2	10.5	0.009

Pruebas en rampas de aceleración y desaceleración

Esta prueba se realizó para observar el seguimiento del controlador interno de velocidad con distintas rampas de aceleración y desaceleración, para la cual durante la prueba se obtienen la velocidad actual y la de referencia. Los resultados obtenidos se los observa en la Figura 3.7.

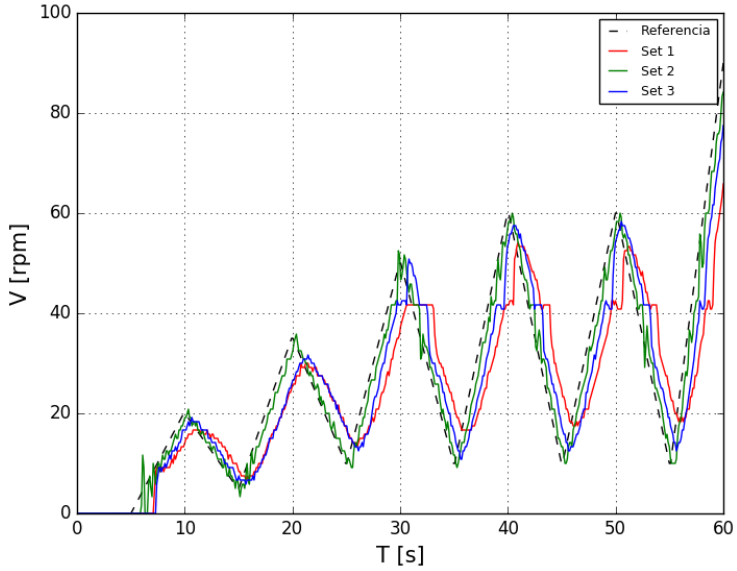


Figura 3.7 Seguimiento del controlador interno de velocidad ante rampas de aceleración y desaceleración con diferentes constantes.

Se observa en la Figura 3.7 que las constantes de los Sets 1 y 3 no logran seguir de manera adecuada a las rampas de aceleración presentando un error notable a los 30 segundos de la prueba, esto frente a los resultados obtenidos con las constantes del Set 2.

Pruebas a cambios de referencia.

La finalidad de esta prueba es determinar el set de constantes que reacciona de mejor manera ante cambios de referencia, ya que para el seguimiento de trayectoria es necesario que el tiempo de reacción sea lo más rápido posible. Para esta prueba se sometió al sistema a diferentes cambios de referencia de velocidad en un periodo de 60 segundos, la prueba se realizó para los Sets 1,2 y 3 descritas en la Tabla 3.4.

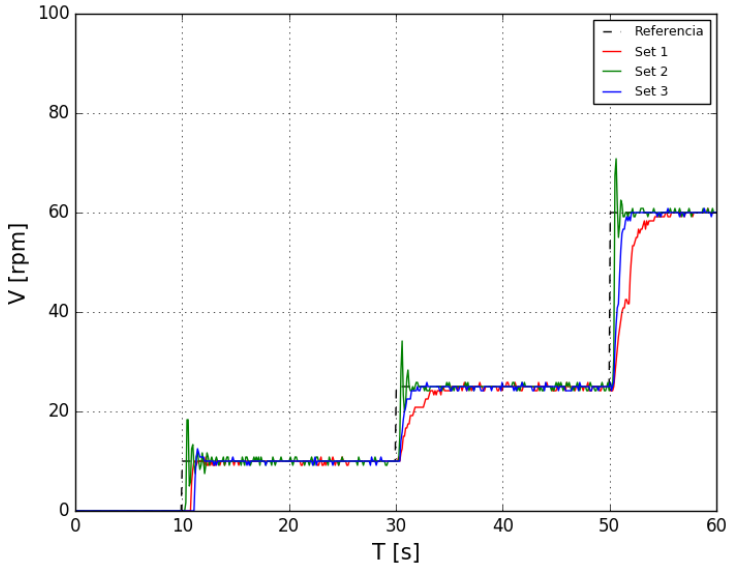


Figura 3.8 Seguimiento del controlador interno de velocidad ante cambios de referencia con diferentes constantes.

En la Figura 3.8 se observan los resultados obtenidos para los diferentes Sets. Los cambios de referencia se dan a los 10 ,30 y 50 segundos. Se observa que los Sets 1 y 3 no presentan sobre impulso, pero tienen un tiempo de estabilización mayor que para el Set 2, aunque este último presenta un sobre impulso.

Resumiendo lo anterior, se escoge las constantes del Set 2: $k_p = 40$, $k_d = 15$ y $k_i = 0.008$ para las pruebas de seguimiento de trayectoria del grupo de robots móviles debido a que con estas constantes los resultados en las dos pruebas realizadas para el controlador interno de velocidad presentan un menor error en el seguimiento de referencia y menor tiempo de establecimiento.

3.4 PRUEBAS DE SEGUIMIENTO DE TRAYECTORIA CIRCULAR PARA DIFERENTES GANANCIAS PARA EL CONTROLADOR EXTERNO.

La ganancia del controlador externo está definida por los valores de la diagonal principal de la $Matriz\ K$ de la Ecuación 2.19. Para estas pruebas se utiliza 3 valores diferentes en la $Matriz\ K$ que son $K = 0.04$, $K = 0.3$ y $K = 0.7$ como se observa en la Ecuación 3.4.

$$Matriz\ K = \begin{bmatrix} K & 0 & 0 & 0 & 0 & 0 \\ 0 & K & 0 & 0 & 0 & 0 \\ 0 & 0 & K & 0 & 0 & 0 \\ 0 & 0 & 0 & K & 0 & 0 \\ 0 & 0 & 0 & 0 & K & 0 \\ 0 & 0 & 0 & 0 & 0 & K \end{bmatrix} \tag{3.4}$$

Para el seguimiento se utiliza una trayectoria circular que tiene un radio de 0.3 metros, con centro en (0.78, 0.59) m. Inicialmente el centroide de la formación se ubica en la posición (1.08, 0.39) m y cada robot móvil tiene una orientación inicial de 90º, por lo que, la formación deberá alcanzar la posición inicial de la trayectoria (1.08, 0.59) m (banderín de color rojo en la Figura 3.9) y posteriormente seguirla. En la Tabla 3.5 se muestran tanto los parámetros de formación y postura iniciales como los deseados. Las pruebas fueron realizadas para una velocidad de referencia de 0.09 m/s.

Tabla 3.5 Variables de forma y postura.

Trayectorias	Valores Iniciales	Valores deseados
d_1	$0.17\ [m]$	$0.17\ [m]$
d_2	$0.17\ [m]$	$0.17\ [m]$
β	$\frac{\pi}{3}\ [rad]$	$\frac{\pi}{3}\ [rad]$
Xc	$1.08\ [m]$	$x_c\ circular(t)$
Yc	$0.39\ [m]$	$y_c\ circular(t)$
φ	$\frac{\pi}{3}\ [rad]$	$\frac{\pi}{3}\ [rad]$

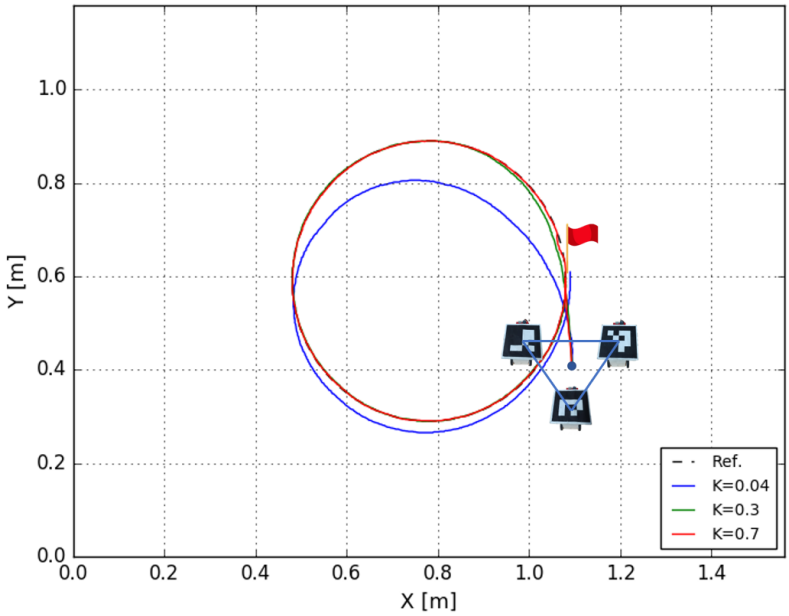


Figura 3.9 Seguimiento de trayectoria circular ante diferentes constantes para el controlador externo.

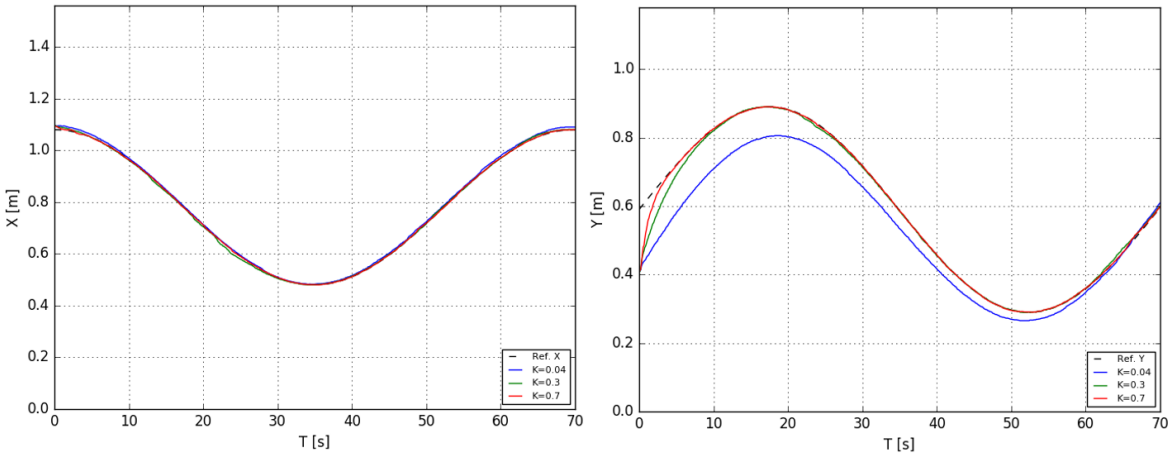


Figura 3.10 Trayectoria en *X* y *Y* de la formación.

En la Figura 3.10 se observa que solo en el eje Y existe una diferencia en el seguimiento de trayectoria, teniendo como mejor resultado las pruebas con $K = 0.3$ y $K = 0.7$ ya que las gráficas están prácticamente superpuestas a la referencia. En la gráfica de errores de trayectoria presentadas en la Figura 3.11 se puede comprobar que para un valor de $K = 0.04$ el error se acerca a cero casi al final de la trayectoria, a $K = 0.3$ llega a cero aproximadamente a los 8 s y con $K = 0.7$ el error se acerca a cero a los 5 s.

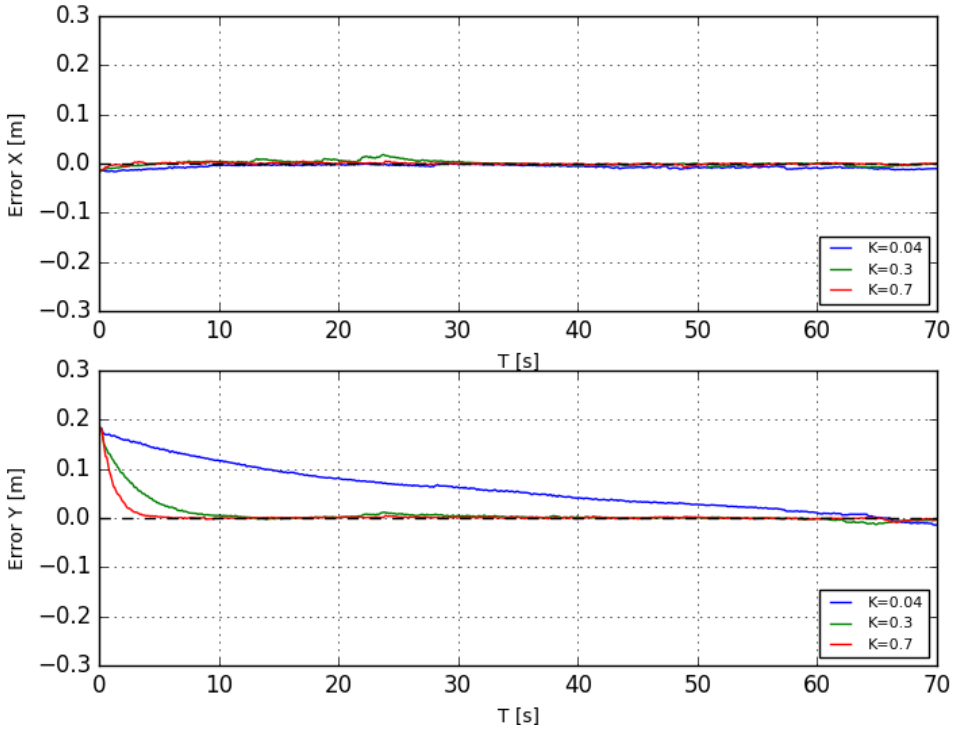


Figura 3.11 Errores de trayectoria en *X* y *Y* de la formación.

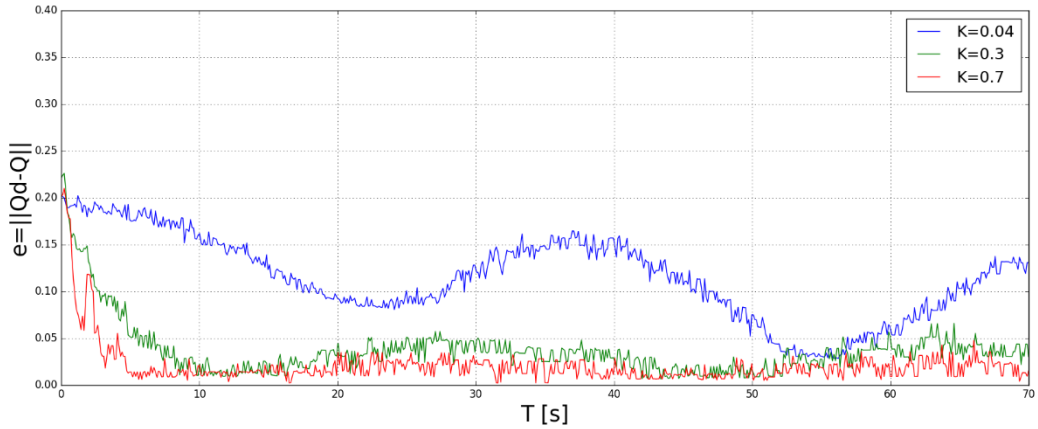


Figura 3.12 Norma del error $e = |Q_d - Q|$ ante diferentes constantes de control externo.

En la Figura 3.12 se muestra la comparación de la norma del error con las tres constantes para el control externo y se puede observar que el menor error promedio obtenido fue con $K = 0.7$ llegando a 0.03. A pesar de esto se escogió como mejor constante $K = 0.3$ debido a que las señales de control de las ruedas del robot móvil r_1, véase Figura 3.13, no presentan sobrepicos elevados que exigen un esfuerzo excesivo a los motores y descargan las baterías de forma más rápida. Para este sistema se consideró señales de control menores a 90 rpm.

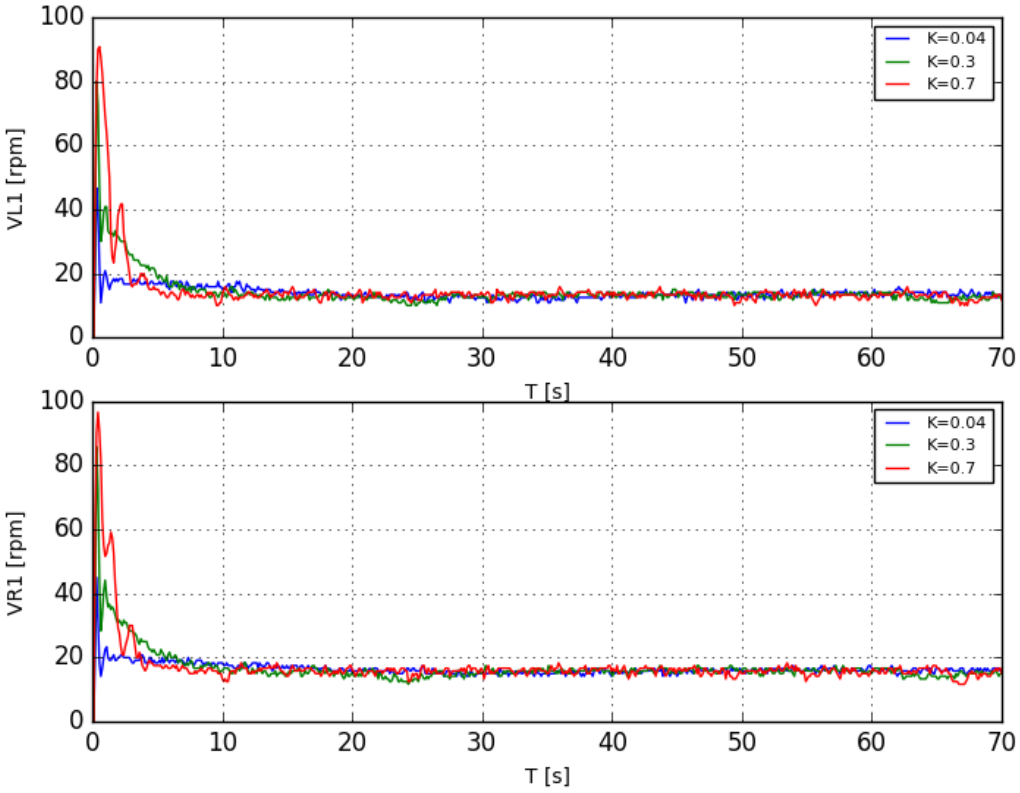


Figura 3.13 Señales de control de las velocidades de rueda izquierda y derecha del robot móvil r_1.

Para los robots r_2 y r_3 se tienen señales similares y se presentan en el Anexo V junto con las gráficas de la norma del error para cada uno de los parámetros de formación y postura.

Finalmente, con el valor de $K = 0.3$, la $Matriz\ K$ queda definida como:

$$Matriz\ K = \begin{bmatrix} 0.3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.3 \end{bmatrix}$$

3.5 PRUEBAS DE SEGUIMIENTO DE TRAYECTORIAS ANTE DIFERENTES VALORES DE VELOCIDAD.

Una vez definida la $Matriz\ K$ del controlador externo de formación y postura, se procede a realizar pruebas para el seguimiento de trayectorias a diferentes valores de velocidad, de esta manera se establecen rangos de velocidad de referencia para cada una de las trayectorias en las cuales el sistema puede ser probado.

Trayectoria circular

Los valores iniciales y de referencias de forma y postura son los mismos utilizados para la prueba anterior. Según varias pruebas se determinó que para velocidades de referencia superiores a 0.2 m/s, el sistema de control comienza a trabajar de forma inadecuada y no llega a la referencia de la trayectoria.

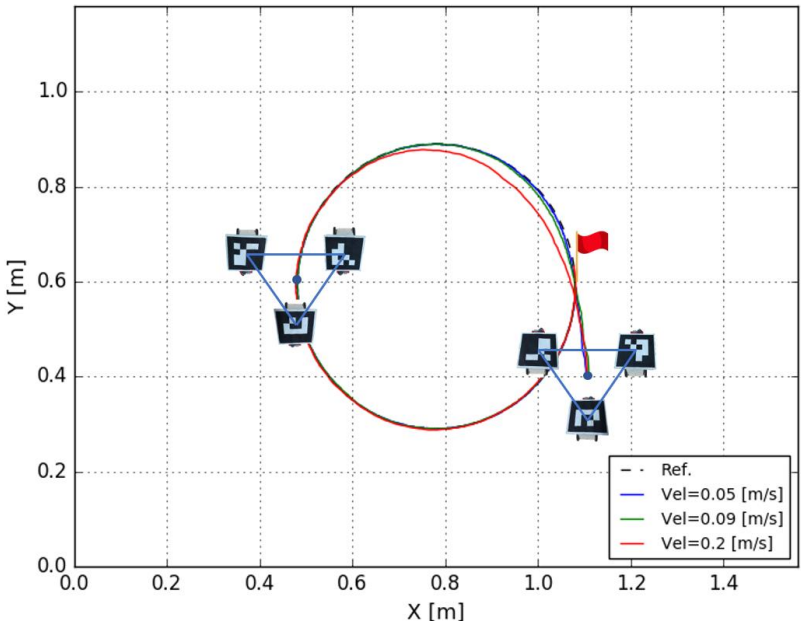


Figura 3.14 Seguimiento de trayectoria circular ante diferentes velocidades.

En la Figura 3.14 se indica el seguimiento de trayectoria con 3 valores de velocidad de referencia:

- Caso A: Seguimiento de trayectoria a 0.05 m/s.
- Caso B: Seguimiento de trayectoria a 0.09 m/s.
- Caso C: Seguimiento de trayectoria a 0.2 m/s.

Para los Casos A y B la trayectoria que describe la formación de robots es suave y sigue la trayectoria de referencia; esto, a diferencia del Caso C donde la formación demora un mayor tiempo en alcanzar la trayectoria de referencia.

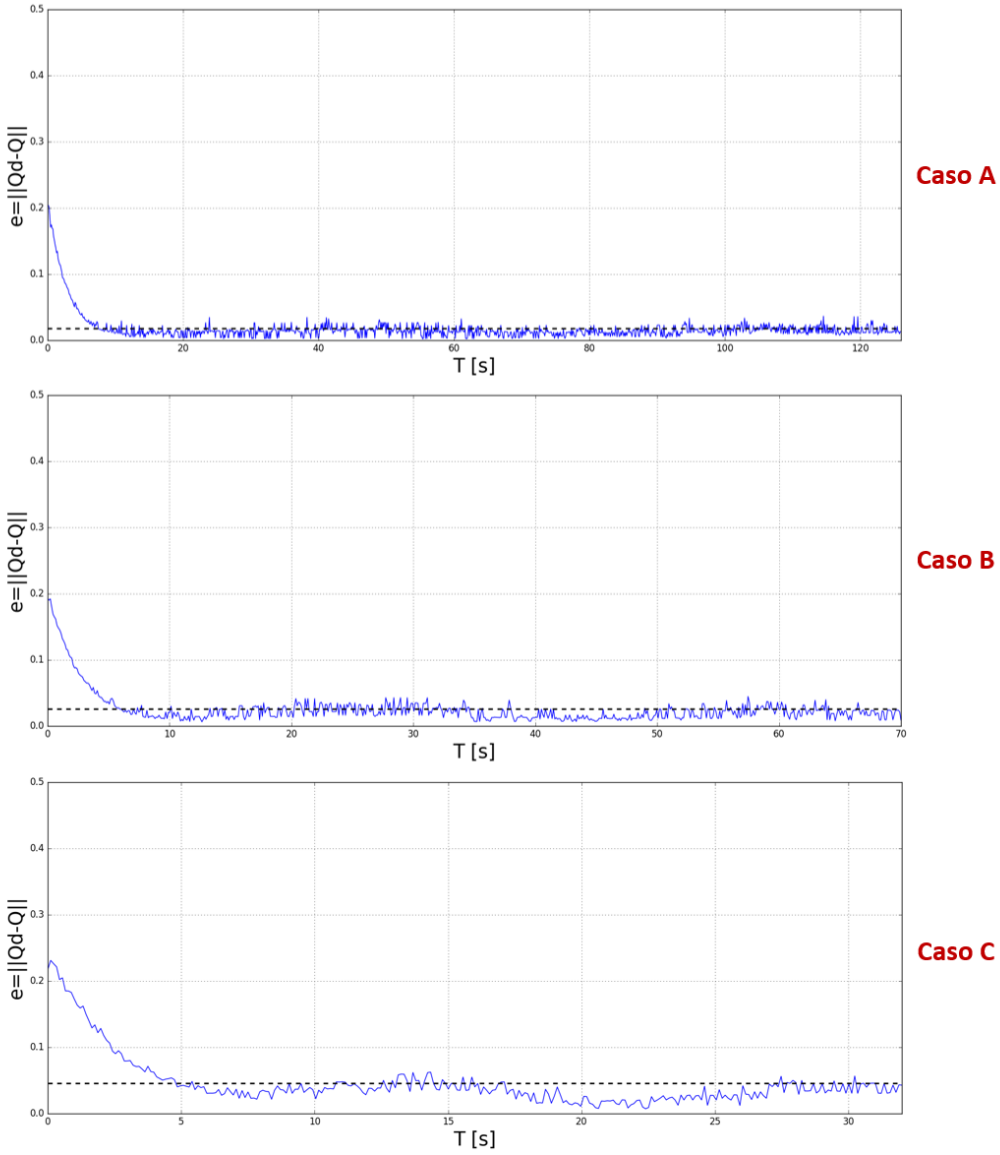


Figura 3.15 Norma del error $e = |Q_d - Q|$ y error promedio ante diferentes velocidades.

En la Figura 3.15 se muestra la comparación de la norma del error y el error promedio entre los tres casos. Debido a que la comparación se hace en una trayectoria completa y esta se recorre en un menor tiempo conforme aumenta la velocidad, el rango del eje tiempo no es el mismo para todos los casos. Se puede observar que el menor error promedio obtenido fue en el Caso A con un error de 0.017, en el Caso B el error llega a los 0.025 mientras que en el Caso C llega a un valor de 0.046. Según los resultados a velocidades bajas se tiene un menor error promedio. Adicionalmente, en el Anexo VI.I, se muestran las señales de error de cada uno de los parámetros de formación y postura para cada caso.

En las Figuras 3.16 y 3.17 se puede observar que para el Caso A las señales de control de la velocidad de la rueda izquierda y derecha del robot móvil r_1 se estabilizan en los valores 6 rpm y 8 rpm respectivamente. Para el Caso B las señales de control alcanzan los 13 y 15

rpm. Finalmente, para el Caso C las señales de control son de 30 y 35 rpm con ciertas oscilaciones al inicio hasta estabilizarse. En el Anexo VI.II se presentan las señales de control de los robots móviles r_2 y r_3.

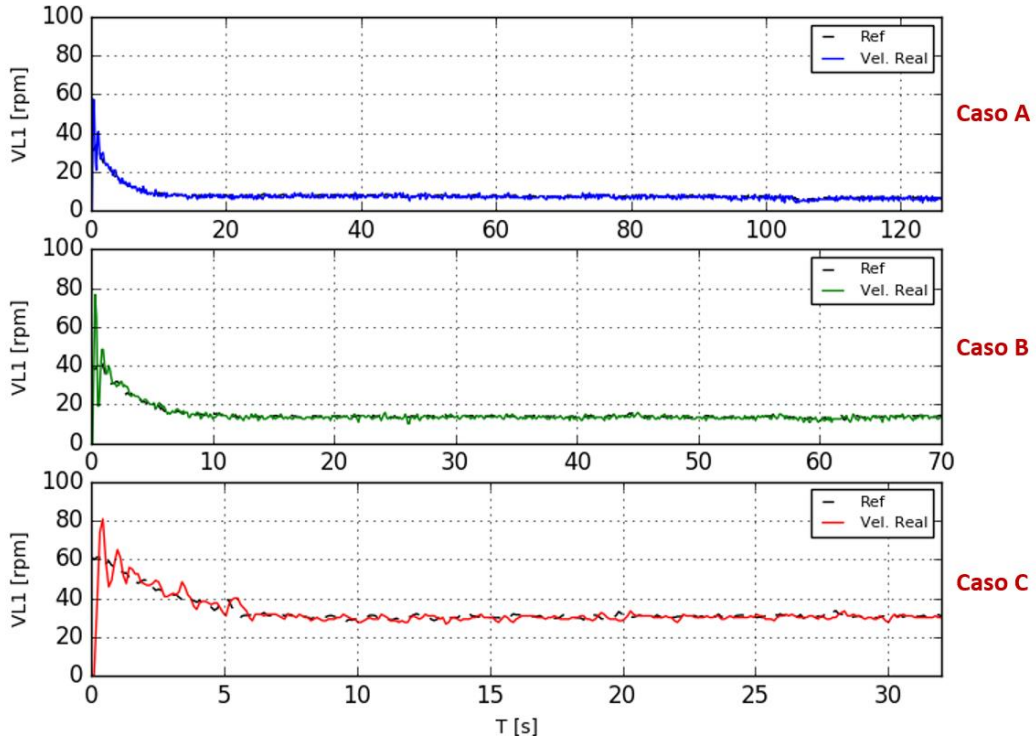


Figura 3.16 Señales de control de velocidad de la rueda izquierda del robot móvil r_1.

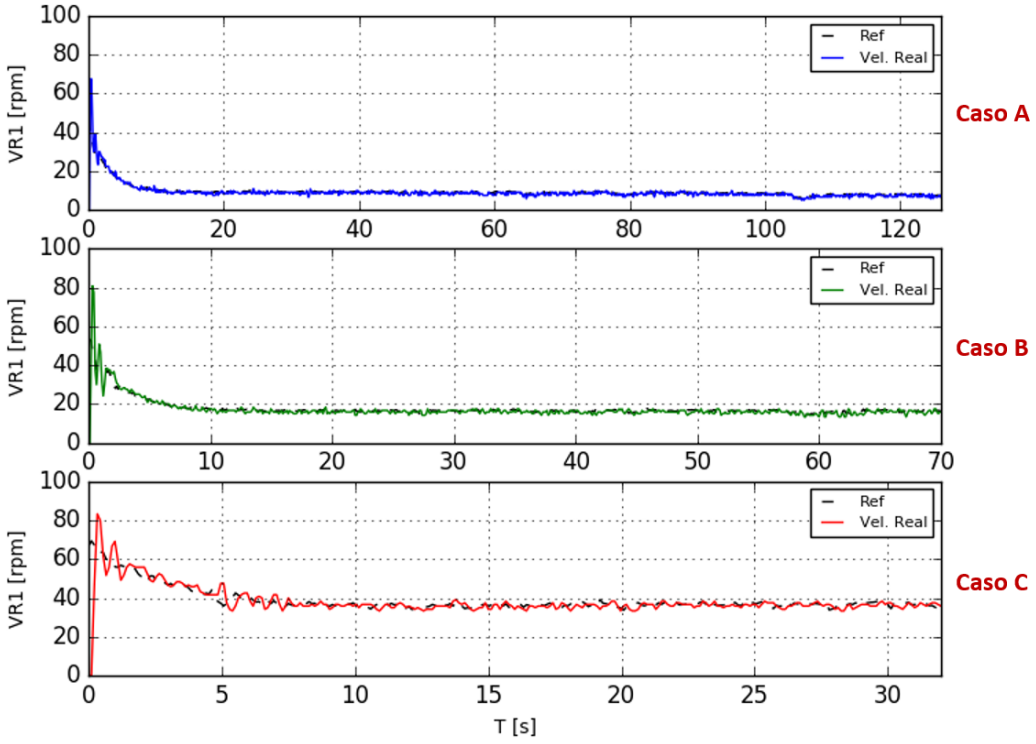


Figura 3.17 Señales de control de velocidad de la rueda derecha robot móvil r_1.

Trayectoria cuadrada

Para esta prueba se utiliza una trayectoria cuadrada que tiene un lado de 0.5 metros. Inicialmente el centroide de la formación se ubica en la posición (0.25, 0.44) m y cada robot móvil se coloca con una orientación de 0º. La formación deberá alcanzar la posición inicial de la trayectoria (0.47, 0.44) m (banderín de color rojo en la Figura 3.18) y posteriormente seguirla. En la Tabla 3.6 se muestran los valores iniciales y referencias de los parámetros de formación y postura para esta trayectoria.

Tabla 3.6 Variables de forma y postura.

Trayectorias	Valores Iniciales	Valores deseados
d_1	$0.17\ [m]$	$0.17\ [m]$
d_2	$0.17\ [m]$	$0.17\ [m]$
β	$\frac{\pi}{3}\ [rad]$	$\frac{\pi}{3}\ [rad]$
Xc	$0.25\ [m]$	$x_c\ cuadrada\ (t)$
Yc	$0.44\ [m]$	$y_c\ cuadrada\ (t)$
φ	$\frac{\pi}{3}\ [rad]$	$\frac{\pi}{3}\ [rad]$

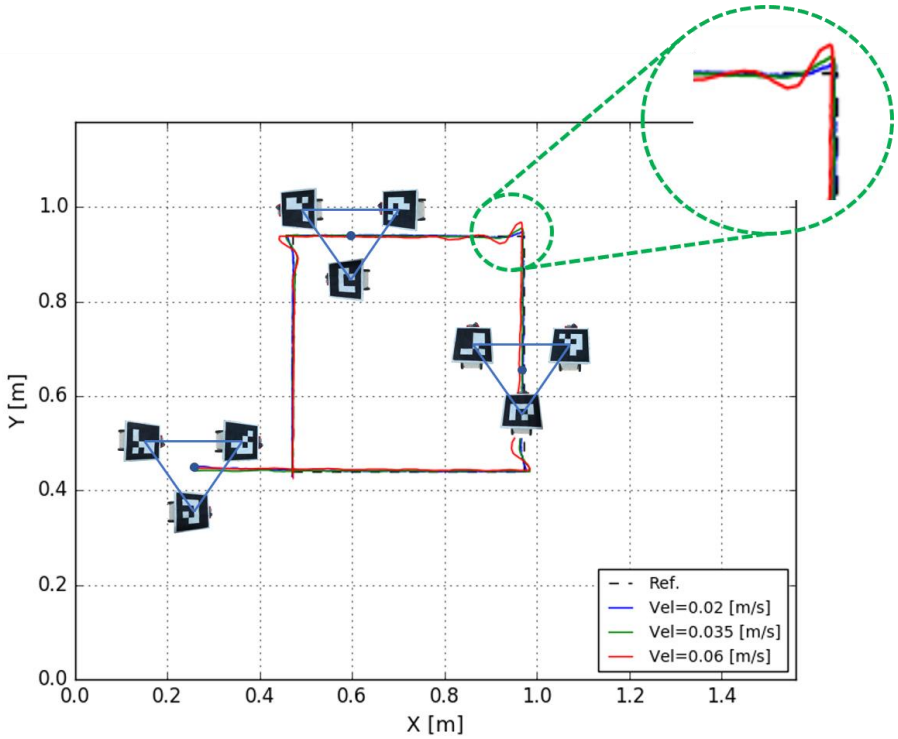


Figura 3.18 Seguimiento de trayectoria cuadrada ante diferentes velocidades.

Los casos para las pruebas con esta trayectoria son:

- Caso A: Seguimiento de trayectoria a 0.02 m/s.
- Caso B: Seguimiento de trayectoria a 0.035 m/s.
- Caso C: Seguimiento de trayectoria a 0.06 m/s.

En el Caso A se describe una trayectoria casi igual a la referencia, en el Caso B existe un pequeño error cuando la formación pasa por las esquinas de la trayectoria cuadrada, mientras que en el Caso C en los cambios bruscos de trayectoria presenta oscilaciones hasta alcanzar nuevamente la trayectoria.

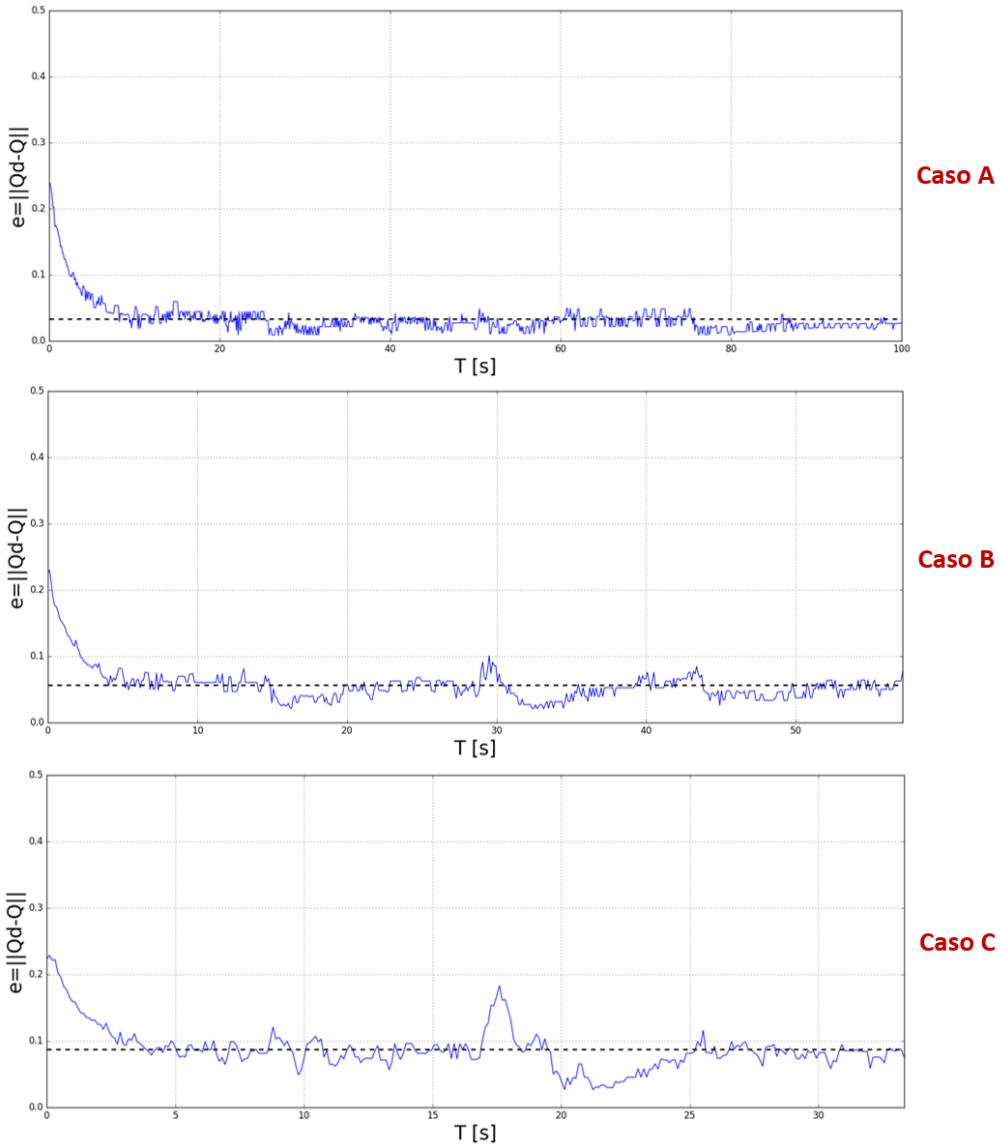


Figura 3.19 Norma del error $e = |Q_d - Q|$ y error promedio a diferentes velocidades.

El error de posicionamiento inicial fue de 0.22 metros alejado del punto objetivo. En la Figura 3.19 se muestra la comparación de la norma del error y error promedio en los tres casos de velocidad consideradas. En el Caso A se puede observar que aparecen breves picos en $t = 25, 50$ y 75 s, para el Caso B en $t = 14.2, 28.57$ y $42{,}85$ s y para el Caso C en $t = 8.5, 17$ y 25.5 s. Para todos los casos, estos son los instantes en los cuales el centroide alcanza las esquinas de la trayectoria cuadrada. Para el caso A el error promedio es de 0.034, para el caso B es 0.056 y para el caso C es 0.087. En el Anexo VII.I se muestran las señales de error de cada uno de los parámetros de formación y postura con cada caso de velocidad.

Las señales de control para cada caso se indican en las Figuras 3.20 y 3.21. En el Caso A las señales de control de la velocidad de la rueda izquierda del robot móvil r_1 son de 10 rpm con un salto a -20 rpm cuando la formación pasa por las esquinas de la trayectoria cuadrada. De manera similar, para el Caso B las señales de control son de 20 rpm con un salto a -30 rpm y para el Caso C las señales de control son de 38 rpm con un salto a - 60 rpm. Las señales de control de la velocidad de la rueda derecha del robot móvil r_1 para el Caso A son de 10 rpm con picos hasta 20 rpm, para el Caso B son de 20 rpm con picos hasta 30 rpm y para el Caso C son de 38 rpm con ciertas oscilaciones cuando la formación pasa por las esquinas de la trayectoria cuadrada.

En el Anexo VII.II se muestran las señales de control de los robots móviles r_2 y r_3.

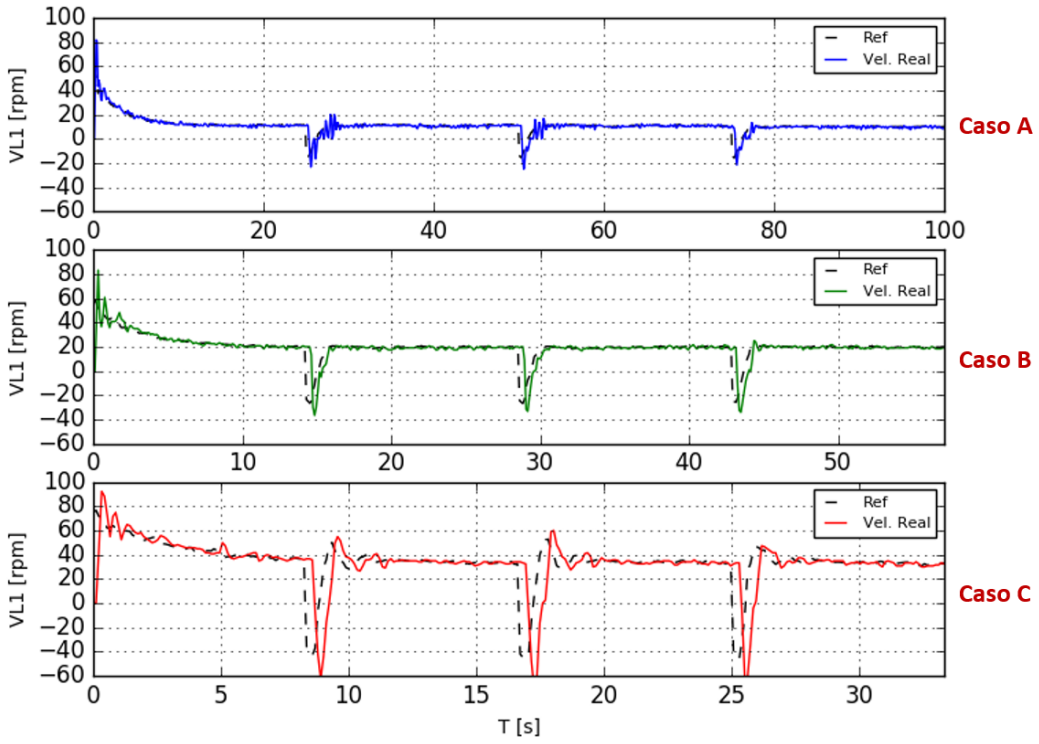


Figura 3.20 Señales de control de velocidad de la rueda izquierda del robot móvil r_1.

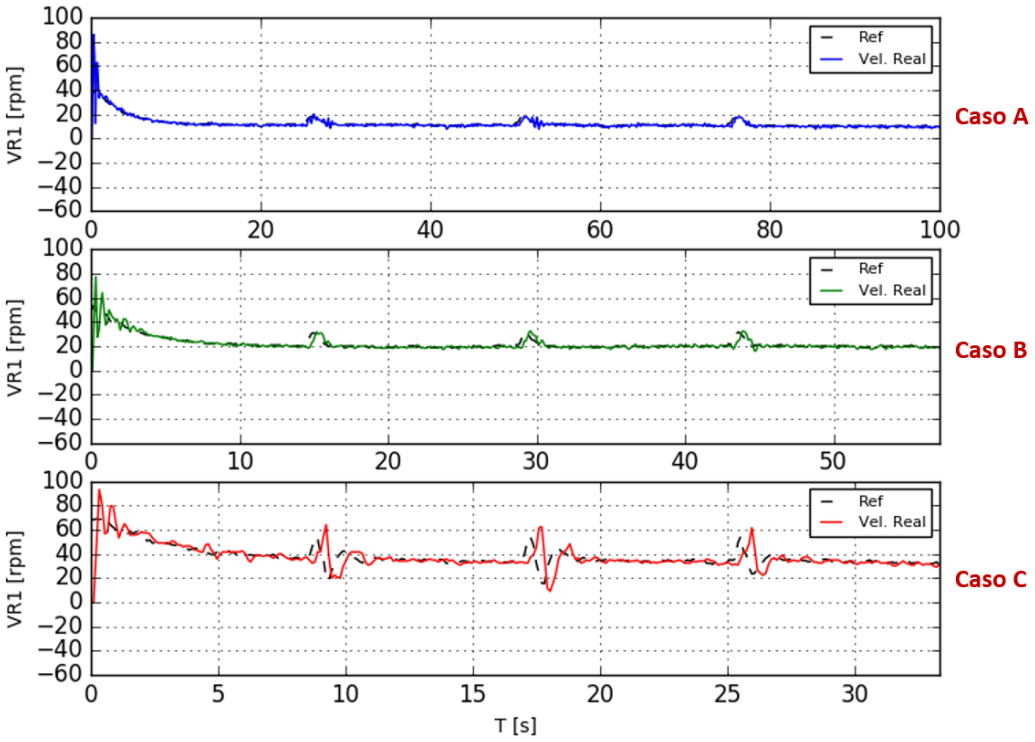


Figura 3.21 Señales de control de velocidad de la rueda derecha del robot móvil r_1.

Trayectoria doble frecuencia

En esta trayectoria se consideró un radio de 0.3 metros. Inicialmente el centroide de la formación se ubica en la posición (1.23, 0.39) m y cada robot móvil se coloca con una orientación de 90º, por lo que la formación deberá alcanzar la posición inicial de la trayectoria (1.23, 0.59) m (banderín de color rojo en la Figura 3.22) y posteriormente seguirla. En la Tabla 3.7 se muestran los valores iniciales y referencias de los parámetros de formación y postura para esta trayectoria.

Tabla 3.7 Variables de forma y postura.

Trayectorias	Valores Iniciales	Valores deseados
d_1	$0.17\ [m]$	$0.17\ [m]$
d_2	$0.17\ [m]$	$0.17\ [m]$
β	$\frac{\pi}{3}\ [rad]$	$\frac{\pi}{3}\ [rad]$
Xc	$1.23\ [m]$	$x_c\ doble\ frecuencia\ (t)$
Yc	$0.39\ [m]$	$y_c\ doble\ frecuencia\ (t)$
φ	$\frac{\pi}{3}\ [rad]$	$\frac{\pi}{3}\ [rad]$

Los valores de velocidad de referencia para los casos con esta trayectoria son:

- Caso A: Seguimiento de trayectoria a 0.03 m/s.
- Caso B: Seguimiento de trayectoria a 0.05 m/s.
- Caso C: Seguimiento de trayectoria a 0.1 m/s.

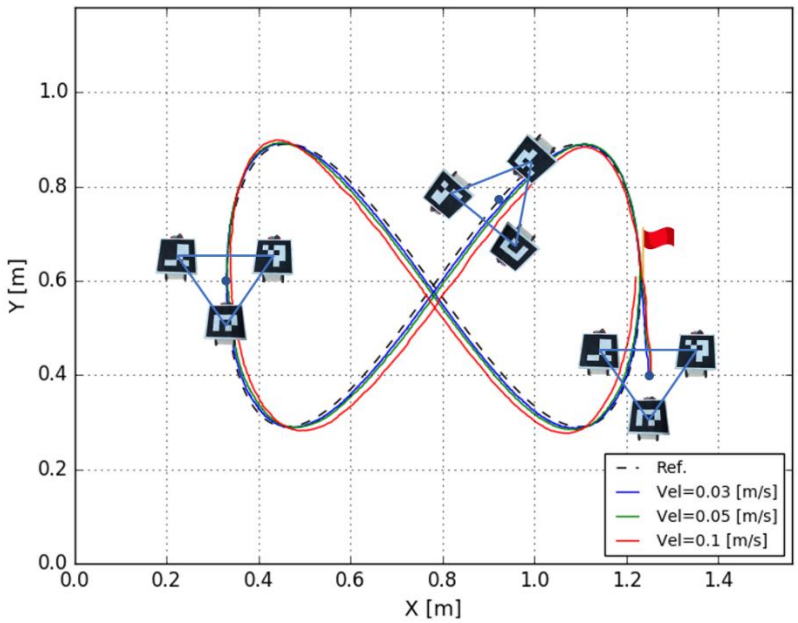


Figura 3.22 Seguimiento de trayectoria doble frecuencia ante diferentes velocidades.

En la Figura 3.22 es posible observar que para el caso A y B la trayectoria descrita por la formación es muy cercana a la referencia, mientras que para el Caso C se tiene un mayor error en ciertas partes de la trayectoria.

El error inicial es de 0.2 respecto al punto objetivo. En la Figura 3.23 se muestra la comparación de la norma del error y el error promedio para los tres casos de velocidad considerados. En el Caso A se puede observar que el error promedio es 0.027, para el Caso B es 0.037 y para el Caso C es 0.065 m. En el Anexo VIII.I se muestran las señales de error de cada uno de los parámetros de formación y postura para cada caso de velocidad.

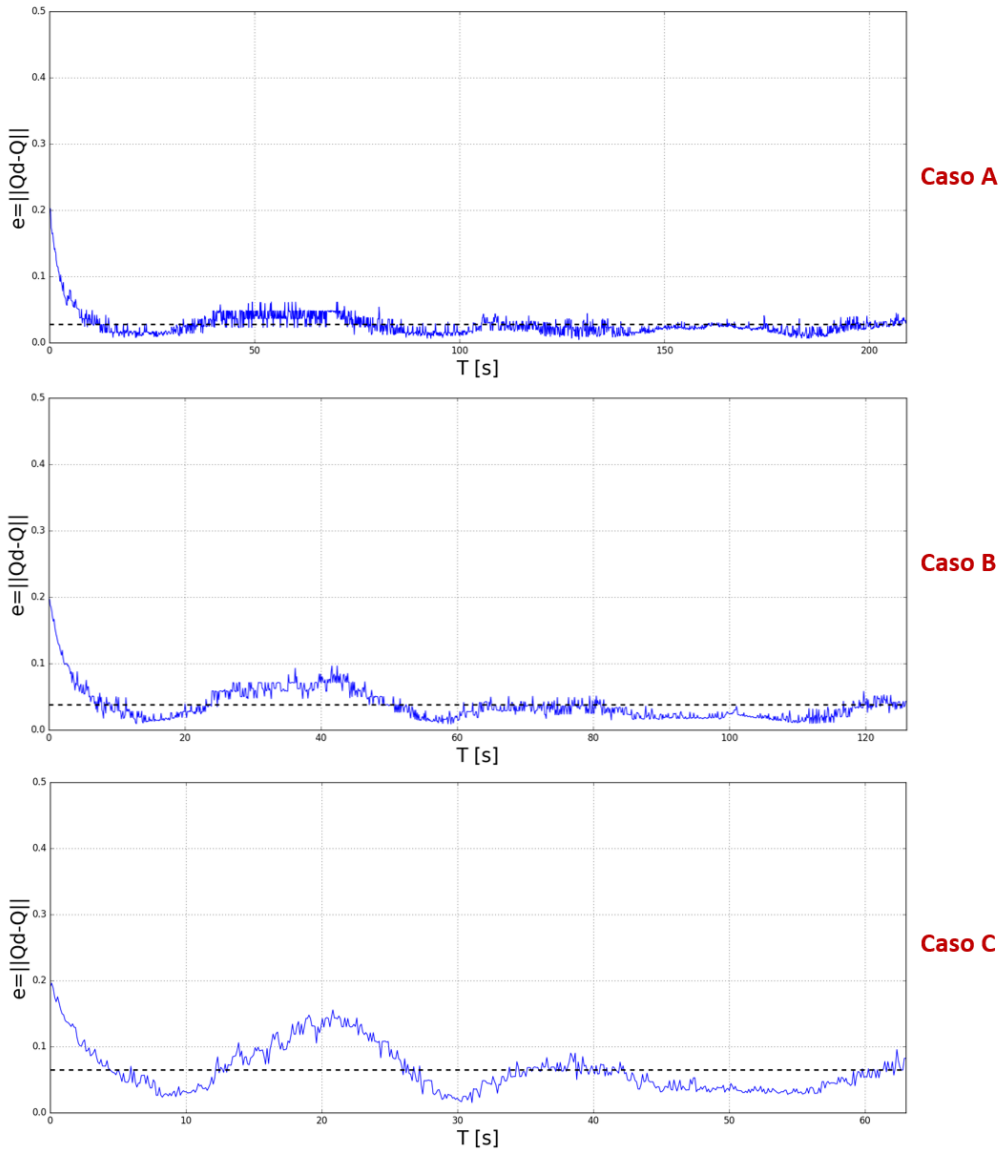


Figura 3.23 Norma del error $e = |Q_d - Q|$ y error promedio a diferentes velocidades.

En las Figuras 3.24 y 3.25 se puede observar que las señales de control disminuyen cuando el centroide de la formación pasa por las curvas de los lóbulos de la trayectoria como por ejemplo para el Caso A de 50 a 75 s baja a 5 rpm aproximadamente, y al salir de las curvas las señales empiezan a aumentar llegando a 15 rpm a los 100 s. Para esta trayectoria, a velocidades lineales superiores a 0.1 m/s el controlador no trabaja de forma adecuada y se ve afectada la señal con ciertas oscilaciones hasta alcanzar la trayectoria como se observa en el Caso C. En el Anexo VIII.II se muestran las señales de control de los robots móviles r_2 y r_3.

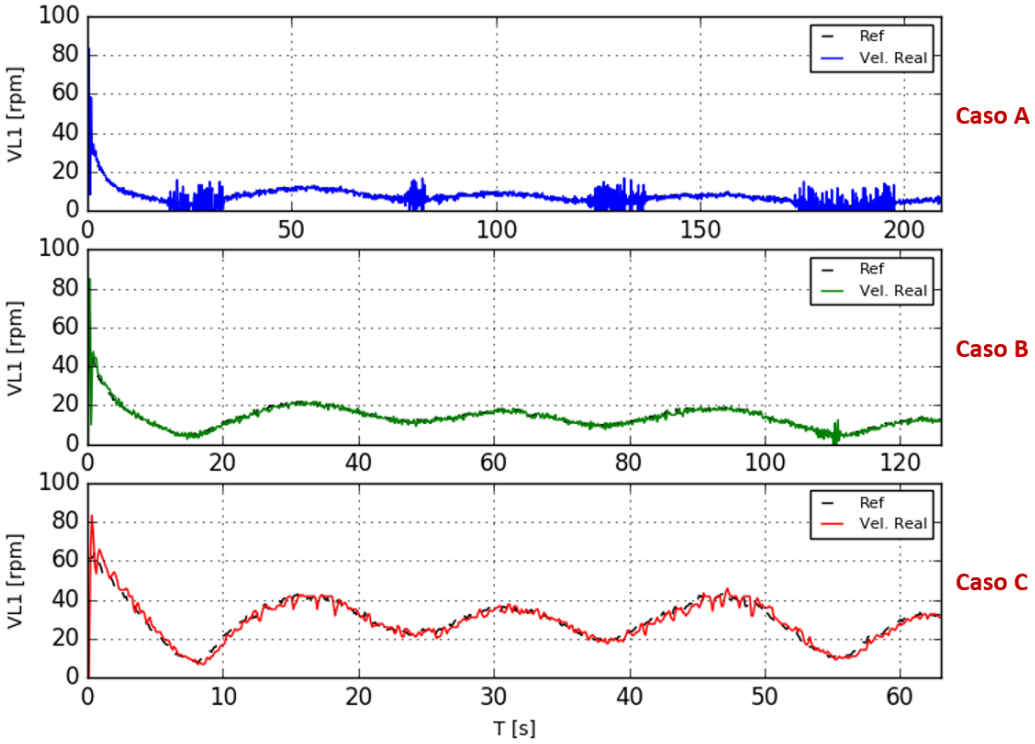


Figura 3.24 Señales de control de velocidad de la rueda izquierda del robot móvil r_1.

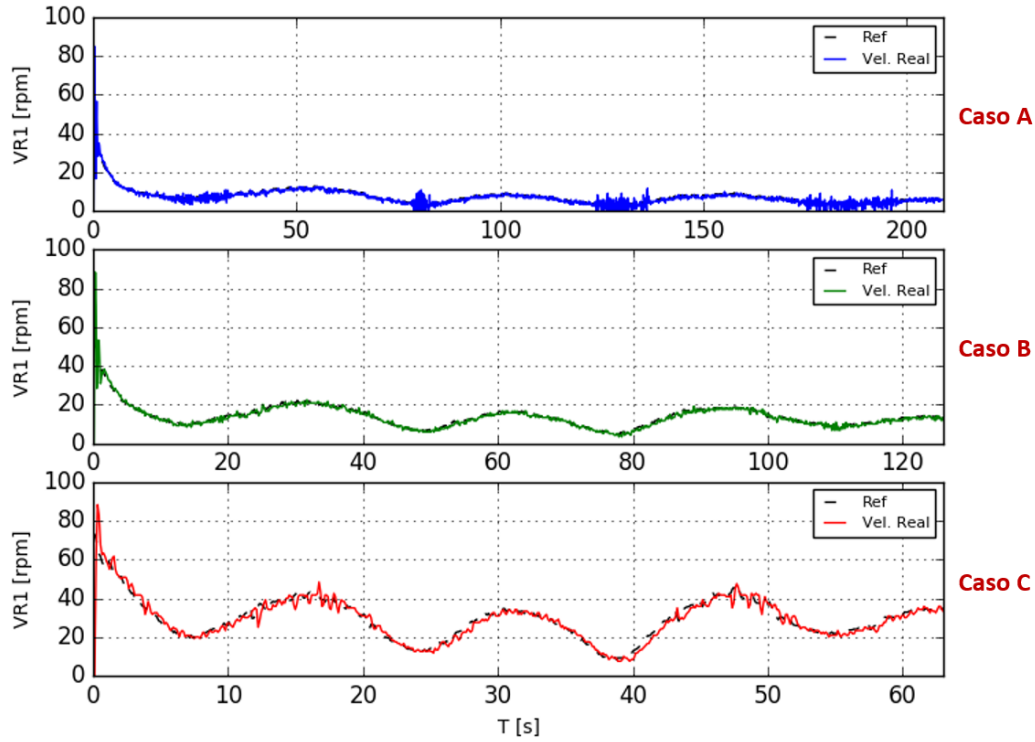


Figura 3.25 Señales de control de velocidad de la rueda derecha del robot móvil r_1.

3.6 PERTURBACIONES.

En la Figura 3.26 se indica el comportamiento de la formación de robots para la trayectoria de doble frecuencia ante perturbaciones, que ocurren a los 20, 60 y 90 segundos. Estas perturbaciones consisten en alejar aproximadamente 0.20 m al robot móvil r_2 de la formación. Para esta prueba inicialmente el centroide de la formación se ubica en la posición (1.23, 0,39) m y cada robot móvil inicia con una orientación de 90º. El centroide alcanza la posición (1.23, 0.59) m corrigiendo los parámetros de forma ya que se tiene un error inicial de 0.13 m en las distancias d_1 y d_2, para luego seguir la trayectoria de doble frecuencia, la cual es generada en base a un radio de 0.3 m y una velocidad de referencia de 0.05 m/s. En la Tabla 3.8 se muestran los valores iniciales y referencias de los parámetros de formación y postura para esta prueba.

Tabla 3.8 Variables de forma y postura.

Trayectorias	Valores Iniciales	Valores deseados
d_1	$0.30\ [m]$	$0.17\ [m]$
d_2	$0.30\ [m]$	$0.17\ [m]$
β	$\frac{\pi}{3}\ [rad]$	$\frac{\pi}{3}\ [rad]$
Xc	$1.23\ [m]$	$x_c\ doble\ frecuencia\ (t)$
Yc	$0.39\ [m]$	$y_c\ doble\ frecuencia\ (t)$
φ	$\frac{\pi}{3}\ [rad]$	$\frac{\pi}{3}\ [rad]$

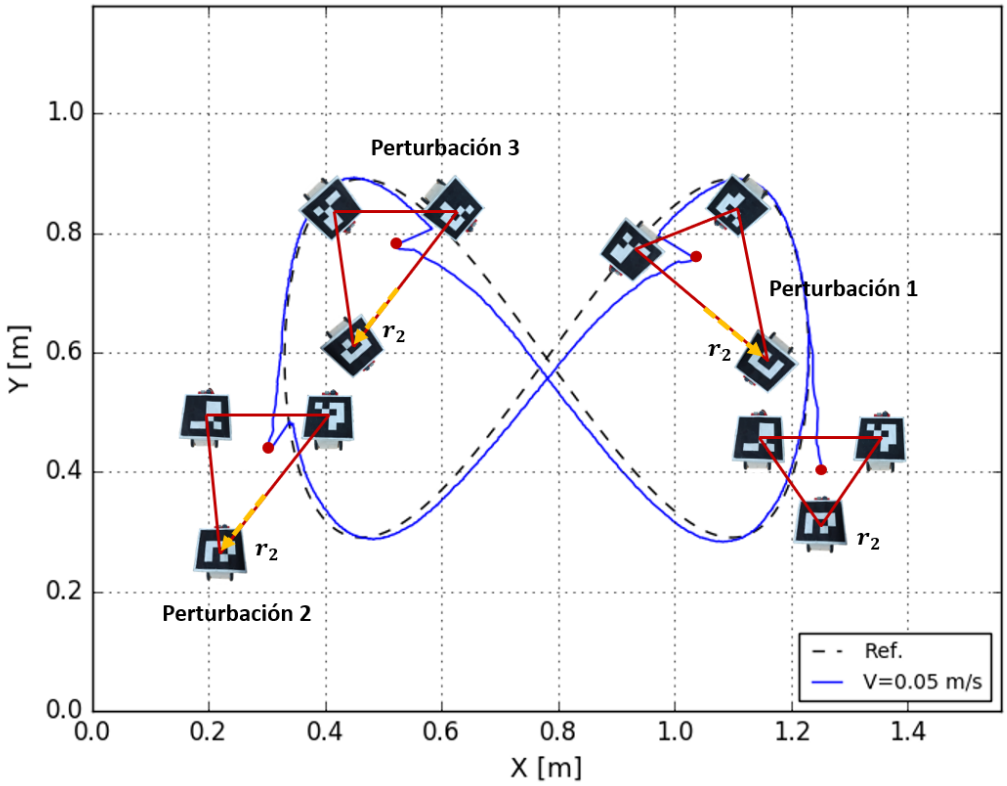


Figura 3.26 Seguimiento de trayectoria doble frecuencia con perturbaciones. El robot móvil r_2, al que se le aplica la perturbación, es etiquetado en esta figura.

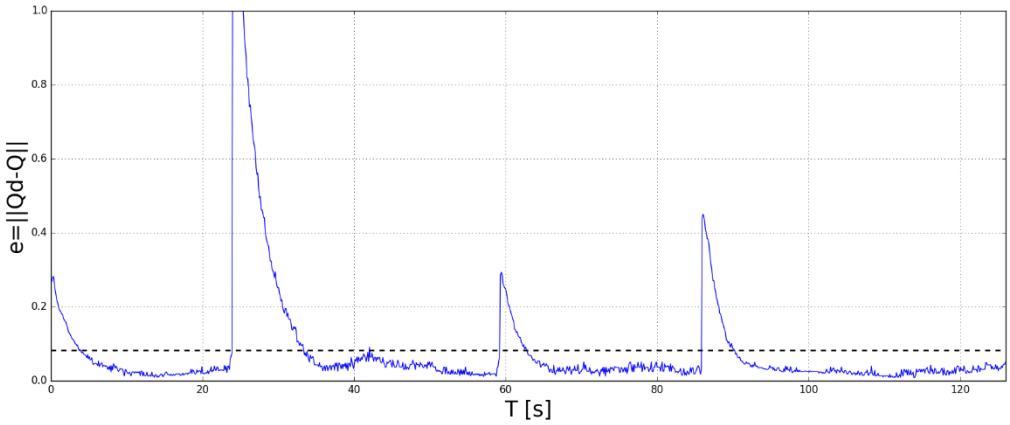


Figura 3.27 Norma del error $e = |Q_d - Q|$ a diferentes velocidades.

El error de posicionamiento inicial fue de 0.2 m. En la Figura 3.27 se puede observar que en el momento que ocurren las perturbaciones el error incrementa y luego vuelve a reducir a casi cero teniendo un error promedio de 0.081.

En la Figura 3.28 se puede observar que las señales de control de velocidad del robot móvil r_2 incrementan a 80 rpm para la rueda izquierda y 60 rpm para la rueda derecha en la primera perturbación llegando a estabilizarse a los 8 segundos después de ocurrida la

perturbación. Como en las perturbaciones se afectó al robot móvil r_2,en el Anexo IX se incluyen las gráficas de las velocidades de los robots móviles r_1 y r_3.

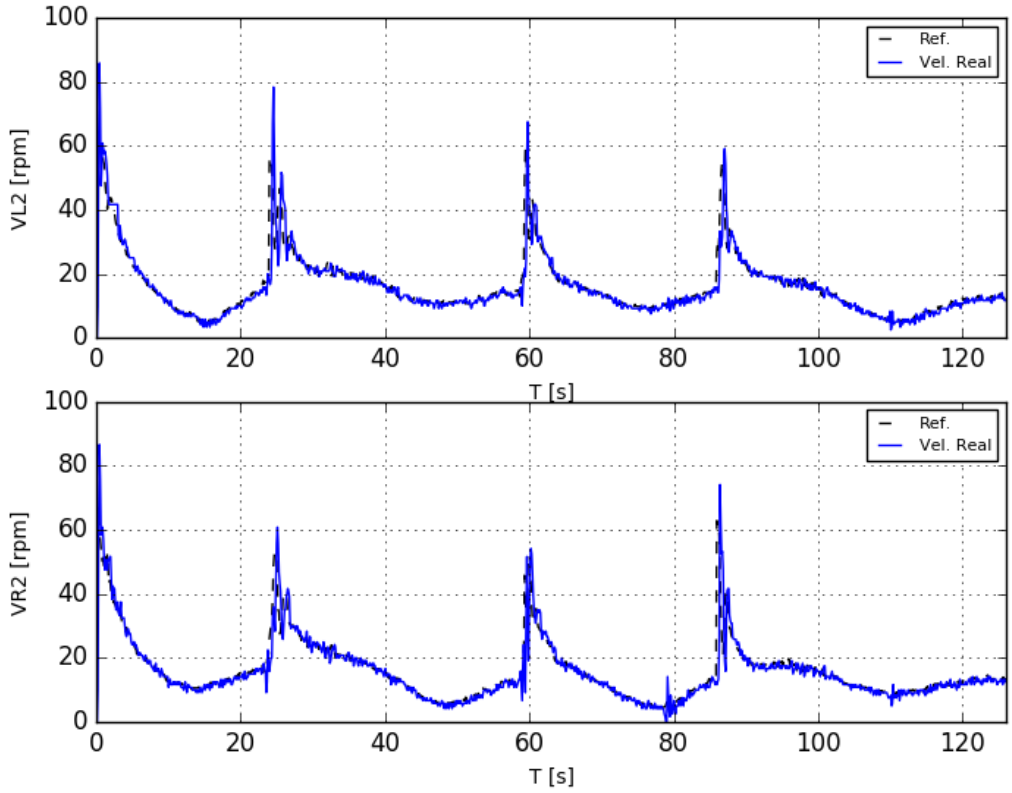


Figura 3.28 Señales de control de las velocidades de rueda izquierda y derecha del robot móvil r_2.

4. CONCLUSIONES

- En base a la revisión bibliográfica se determinó que la arquitectura de control más adecuada para la formación de robots para este trabajo es la arquitectura centralizada con la técnica de estructura virtual ya que el controlador y el sistema de monitoreo se encuentra en el ordenador.

- Las principales mejoras en lo referente al hardware realizadas al banco de pruebas presentado en [3] se basaron en la implementación de un controlador de formación para un grupo de tres robots móviles de tamaño reducido. El control de velocidad se implementó en una tarjeta Arduino Fio V3 la cual fue acoplada sobre una PCB shield para cada robot móvil debido a que el acoplamiento de encoders a los motores de cada robot requirió de una actualización de la interconexión entre todos los componentes electrónicos que son parte de los robots, estos cambios aumentaron su modularidad y facilitó su mantenimiento correctivo en casos de malfuncionamiento de sus elementos.

- Con respecto al software del sistema, el tiempo de procesamiento del sistema de monitoreo basado en visión artificial es mejorado al enlazar las librerías de Open CV con ROS. Además, el desarrollo de los sistemas de comunicación y control junto con la integración de la GUI en QT Creator sobre ROS facilita el manejo del sistema por parte del usuario.

- Entre las grandes ventajas que ofrece ROS destacan la modularidad, ya que los sistemas de la plataforma, estos son monitoreo, visión artificial, control y comunicaciones, interactúan fácilmente entre si mediante la utilización de mensajes, esto hace posible la programación en diferentes lenguajes; en particular Python y C++.

- En este trabajo se utilizaron dos lenguajes de programación, C++ y Python; este último tiene varias herramientas y librerías similares a MATLAB por lo que resulta la mejor opción para la programación del controlador externo.

- Para el monitoreo y adquisición de los datos de posición y orientación de cada uno de los robots móviles en el área de trabajo, se hizo uso del seguimiento de marcadores de referencia ArUco con ayuda de las bibliotecas de OpenCV en C++ sobre ROS.

- Respecto al sistema de comunicación mediante los módulos Xbee, el modo API con escape ayudó a implementar una mejor coordinación entre el controlador de

formación centralizado y el grupo de robots móviles. Este permite tener una topología punto-multipunto donde el coordinador se encuentra en el ordenador y los End Device en los robots y sus mensajes son en base a estructuras, que son creadas en Python y Arduino.

- La arquitectura de control de formación propuesta (control interno de velocidad en cascada con el controlador externo de forma y postura) demostró excelentes resultados en el seguimiento de todas las trayectorias predefinidas: circular, cuadrada y doble frecuencia. Además, el controlador responde de forma correcta frente al error inicial de los parámetros de formación y postura, ya que el grupo de robots alcanza la trayectoria manteniendo la forma triangular deseada.
- La corrección de forma y postura del grupo de robots móviles depende de la $Matriz\ K$ del controlador externo. Si los valores de su diagonal están más cerca de 0.01, entonces el controlador responde de mejor manera a las perturbaciones externas, pero el error de seguimiento de trayectoria aumenta. Mientras tanto, si los valores son superiores a 0.2, el error de seguimiento de trayectoria es cercano a cero, pero las señales de control de velocidad se vuelven más bruscas. A pesar de que el esquema de control no fue diseñado para posibles perturbaciones, el sistema es capaz de identificar el retraso del robot móvil y compensarlo.
- El tiempo de latencia se determinó principalmente en base al tiempo promedio de comunicación y procesamiento de las imágenes, de lo cual se puede concluir que la parte más lenta del sistema es la comunicación.
- Trabajos futuros pueden enfocarse en ampliar el uso del banco de pruebas con la adición de un mayor número de agentes. Además, se está analizando la posibilidad de incluir cuadricópteros de tamaño reducido como parte del grupo de robots. También se están desarrollando aplicaciones de transporte basadas en la plataforma experimental multi-robot presentada, así como la implementación de algoritmos basados en consenso.

Recomendaciones

- Antes de usar la plataforma se recomienda revisar el manual de usuario del sistema y entender su funcionamiento, en caso de instalar en un ordenador diferente se recomienda considerar las indicaciones proporcionadas en el Anexo V.

- Cada robot móvil puede llegar a funcionar hasta con un voltaje de 3.5 voltios, pero se recomienda utilizar las baterías a un voltaje mayor para evitar que no se altere la operación del controlador en el momento del seguimiento de trayectoria. En la interfaz gráfica se puede verificar el estado de la batería de cada uno de los robots móviles.
- Se recomienda verificar que en el puerto USB se encuentre conectado el módulo Xbee Coordinador, caso contrario al correr el nodo *Response_Robots* se presentará un mensaje de error debido a que no se podrá ejecutar la comunicación entre los robots y el ordenador.
- No se recomienda variar los parámetros del PID interno, debido a que ya han sido previamente calibrados y comprobados, en caso de variarlos se debe comprobar el comportamiento del controlador tanto de la rueda izquierda como de la derecha.
- En caso de que el sistema presente fallas, la interfaz gráfica cuenta con un botón de paro de emergencia que permite al usuario detener la ejecución del experimento en cualquier momento.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] A. Raya, “Tus pedidos del 11-11 lo llevan este enjambre de robots,” Sep. 2018.

[2] Z. Yan, L. Fabresse, J. Laval, and N. Bouraqadi, “Building a ROS-Based Testbed for Realistic Multi-Robot Simulation: Taking the Exploration as an Example,” *Robotics*, vol. 6, no. 3, p. 21, Sep. 2017, doi: 10.3390/robotics6030021.

[3] G. Israel and M. Sierra, “Diseño e implementación de un banco de pruebas de experimentación con robots móviles terrestres de tamaño reducido para el control de posición y seguimiento de trayectoria,” 2019.

[4] M. Quigley, B. Gerkey, and W. D. Smart, *Programming Robots with ROS*, 1st ed. United States of America.: O’Reilly Media, 2015.

[5] L. Joseph and J. Cacace, *Mastering ROS for robotics programming : design, build, and simulate complex robots using Robot Operating System*, 1st ed. 2015.

[6] I. Bambino, “Una Introducción a los Robots Móviles,” 2008. Accessed: Nov. 19, 2019. [Online].

[7] B. Chu, K. Jung, C. S. Han, and D. Hong, “A survey of climbing robots: Locomotion and adhesion,” *International Journal of Precision Engineering and Manufacturing*, vol. 11, no. 4, pp. 633–647, Aug. 2010, doi: 10.1007/s12541-010-0075-3.

[8] M. A. Molina Villa, E. L. Rodriguez Vasquez, and I. en Mecatrónica, “Flotilla de robots para trabajos en robótica cooperativa,” Facultad de Ingeniería, Aug. 2014. Accessed: Sep. 02, 2020. [Online]. Available: http://repository.unimilitar.edu.co/handle/10654/12129.

[9] B. de Vera, “Un robot con patas diseñado para saltar podría sustituir a los Rover en futuras visitas a Marte o la Luna noticias de ciencia,” *N+1, ciencia que suma*, Nov. 2018.

[10] R. Pérez, “Estas serpientes robot acudieron tras el terremoto de México para aprender a localizar víctimas,” *Xataka*, 2017.

[11] J. Alberto Jiménez Builes, D. Arturo Ovalle Carranza, and J. Fredy Ochoa Gómez, “SMART: SISTEMAS MULTI-AGENTE ROBÓTICO SMART: MULTI-AGENT ROBOTIC SYSTEM,” *Año*, vol. 75, pp. 179–186, 2008, Accessed: Nov. 21, 2019. [Online].

[12] “RoboCup Federation.” https://www.robocup.org/photos (accessed Nov. 21, 2019).

[13] M. A. Lewis and K. H. Tan, “High Precision Formation Control of Mobile Robots Using Virtual Structures,” *Autonomous Robots*, vol. 4, no. 4, pp. 387–403, 1997, doi: 10.1023/A:1008814708459.

[14] J. P. Desai, J. P. Ostrowski, and V. Kumar, “Modeling and control of formations of nonholonomic mobile robots,” *IEEE Transactions on Robotics and Automation*, vol. 17, no. 6, pp. 905–908, Dec. 2001, doi: 10.1109/70.976023.

[15] J. A. Pascual, “Más de mil drones de Intel deslumbran en los Juegos Olímpicos de Invierno,” *Computer Hoy*, Feb. 11, 2018. https://computerhoy.com/noticias/life/mas-mil-drones-intel-deslumbran-juegos-olimpicos-invierno-75769 (accessed Nov. 24, 2019).

[16] V. Mazzone and R. Centrífugo De Watt, “Controladores PID,” 2002. Accessed: Dec. 09, 2019. [Online]. Available: http://iaci.unq.edu.ar/caut1.

[17] “Código abierto - EcuRed.” https://www.ecured.cu/C%C3%B3digo_abierto (accessed Dec. 01, 2019).

[18] “What is Linux? - Linux.com.” https://www.linux.com/what-is-linux/ (accessed Dec. 01, 2019).

[19] “What is Ubuntu? - Definition from Techopedia.” https://www.techopedia.com/definition/3307/ubuntu (accessed Dec. 01, 2019).

[20] “Tutorials | Qt Creator Manual,” *The QT Company*. https://doc.qt.io/qtcreator/creator-tutorials.html (accessed Nov. 24, 2019).

[21] “Computer Vision | IBM.” https://www.ibm.com/topics/computer-vision (accessed Nov. 26, 2019).

[22] “OpenCV: OpenCV modules.” https://docs.opencv.org/master/index.html (accessed Nov. 25, 2019).

[23] A. de la Escalera, J. María Armingol, J. Luis Pech, and J. Julián Gómez, “Detección Automática de un Patrón para la Calibración de Cámaras,” *Revista Iberoamericana de Automática e Informática Industrial RIAI*, 2010, doi: 10.1016/s1697-7912(10)70063-7.

[24] Z. Zhang, “A flexible new technique for camera calibration,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, Nov. 2000, doi: 10.1109/34.888718.

[25] M. A. Szövetség and L. Várallyai, “From barcode to QR code applications,” 2012. Accessed: Dec. 17, 2019. [Online]. Available: http://www.magisz.org/.

[26] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, “Automatic generation and detection of highly reliable fiducial markers under occlusion,” *Pattern Recognition*, vol. 47, no. 6, pp. 2280–2292, Jun. 2014, doi: 10.1016/j.patcog.2014.01.005.

[27] doxygen, “OpenCV: Detection of ArUco Markers,” *OpenCV*, 2015. https://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_detection.html (accessed Jan. 07, 2020).

[28] P. A. G. Monsalve and C. M. D. Acevedo, “ADQUISICIÓN DE DATOS DE UNA MATRIZ DE SENSORES DE GASES (E-NOSE), MEDIANTE MÓDULOS DE COMUNICACIÓN XBEE,” *REVISTA COLOMBIANA DE TECNOLOGIAS DE AVANZADA (RCTA)*, vol. 2, no. 26, May 2017, doi: 10.24054/16927257.v26.n26.2015.2383.

[29] Z. Dannelly, “AT vs API (What, Why, How) | Mbed,” *armMBED*, 2015. https://os.mbed.com/users/dannellyz/notebook/at-vs-api-when-why-how/ (accessed Dec. 09, 2019).

[30] “Pololu - DRV8835 Dual Motor Driver Carrier.” https://www.pololu.com/product/2135 (accessed Mar. 16, 2020).

[31] "Pololu - Mini Plastic Gearmotors." https://www.pololu.com/category/158/mini-plastic-gearmotors (accessed Mar. 16, 2020).

[32] "Pololu - Magnetic Encoder Pair Kit for Mini Plastic Gearmotors, 12 CPR, 2.7-18V." https://www.pololu.com/product/1523 (accessed Mar. 08, 2020).

[33] A. Rosales, G. Scaglia, V. Mut, and F. di Sciascio, "Formation control and trajectory tracking of mobile robotic systems - A Linear Algebra approach," *Robotica*, vol. 29, no. 3, pp. 335–349, May 2011, doi: 10.1017/S0263574710000068.

[34] I. Mas, O. Petrovic, and C. Kitts, "Cluster space specification and control of a 3-robot mobile system," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2008, pp. 3763–3768, doi: 10.1109/ROBOT.2008.4543788.

[35] B. Arévalo, "Diseño e implementación de un controlador SMC-Delay para seguimiento de trayectoria de una formación de robots móviles con retardo de entrada," 2018.

[36] "Logitech C920 HD Pro Webcam para Windows, Mac y Chrome OS." https://www.logitech.com/es-roam/product/hd-pro-webcam-c920#specification-tabular (accessed Feb. 24, 2020).

[37] A.-R. Hadeel Tariq, "Studying Main Differences Between Linux & Windows Operating Systems," 2012. Accessed: Mar. 16, 2020. [Online].

[38] "Building Common Project Types | qmake Manual." https://doc.qt.io/qt-5/qmake-common-projects.html (accessed Mar. 02, 2020).

[39] "Introduction — OpenCV 2.4.13.7 documentation." https://docs.opencv.org/2.4/modules/core/doc/intro.html (accessed Mar. 02, 2020).

[40] "camera_calibration/Tutorials/MonocularCalibration - ROS Wiki." http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration (accessed Mar. 05, 2020).

[41] "OpenCV: ArUco Marker Detection." https://docs.opencv.org/master/d9/d6a/group__aruco.html#gab9159aa69250d8d3642593e508cb6baa (accessed Mar. 07, 2020).

[42] "OpenCV: cv::aruco::DetectorParameters Struct Reference." https://docs.opencv.org/trunk/d1/dcd/structcv_1_1aruco_1_1DetectorParameters.html#details (accessed Mar. 07, 2020).

[43] E. Molina, J. Diaz, H. Hidalgo-Silva, and E. Chávez, "Algoritmos de Binarización Robusta de Imágenes con Iluminación No Uniforme," doi: 10.4995/riai.2017.8847.

[44] "SparkFun XBee Explorer Dongle - WRL-11697 - SparkFun Electronics." https://www.sparkfun.com/products/11697 (accessed Apr. 29, 2020).

6. ANEXOS

ANEXO I

DERIVADAS PARCIALES DE FORMACIÓN Y POSTURA

$$\frac{\partial d_1}{\partial x_1} = \frac{\partial \sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2}}{\partial x_1} = \frac{x_1 - x_3}{\sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2}}$$

Ecuación I.1. Derivada parcial 1.

$$\frac{\partial d_1}{\partial y_1} = \frac{\partial \sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2}}{\partial y_1} = -\frac{y_1 - y_3}{\sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2}}$$

Ecuación I.2. Derivada parcial 2.

$$\frac{\partial d_1}{\partial x_2} = \frac{\partial \sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2}}{\partial x_2} = 0$$

Ecuación I.3. Derivada parcial 3.

$$\frac{\partial d_2}{\partial x_1} = \frac{\partial \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}}{\partial x_1} = \frac{x_1 - x_2}{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}}$$

Ecuación I.4. Derivada parcial 4.

$$\frac{\partial d_2}{\partial y_1} = \frac{\partial \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}}{\partial y_1} = \frac{y_1 - y_2}{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}}$$

Ecuación I.5. Derivada parcial 5.

$$\frac{\partial d_2}{\partial x_2} = \frac{\partial \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}}{\partial x_2} = -\frac{x_1 - x_2}{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}}$$

Ecuación I.6. Derivada parcial 6.

$$\frac{\partial x_c}{\partial x_1} = \frac{\partial \frac{x_1 + x_2 + x_3}{3}}{\partial x_1} = \frac{1}{3}$$

Ecuación I.7. Derivada parcial 7.

$$\frac{\partial x_c}{\partial y_1} = \frac{\partial \frac{x_1 + x_2 + x_3}{3}}{\partial y_1} = 0$$

Ecuación I.8. Derivada parcial 8.

$$\frac{\partial x_c}{\partial x_2} = \frac{\partial \frac{x_1 + x_2 + x_3}{3}}{\partial x_2} = \frac{1}{3}$$

Ecuación I.9. Derivada parcial 9.

$$\frac{\partial y_c}{\partial x_1} = \frac{\partial \frac{y_1 + y_2 + y_3}{3}}{\partial x_1} = 0$$

Ecuación I.10. Derivada parcial 10.

$$\frac{\partial y_c}{\partial y_1} = \frac{\partial \frac{y_1 + y_2 + y_3}{3}}{\partial y_1} = \frac{1}{3}$$

Ecuación I.11. Derivada parcial 11.

$$\frac{\partial y_c}{\partial x_2} = \frac{\partial \frac{y_1 + y_2 + y_3}{3}}{\partial x_2} = 0$$

Ecuación I.12. Derivada parcial 12.

$$\frac{\partial \varphi}{\partial x_1} = \frac{\partial \operatorname{atan}\left(\frac{\frac{2}{3}x_1 - \frac{1}{3}x_2 - \frac{1}{3}x_3}{\frac{2}{3}y_1 - \frac{1}{3}y_2 - \frac{1}{3}y_3}\right)}{\partial x_1} = -\frac{3}{2\left(\frac{\left(\frac{x_2}{3} - \frac{3\,x_1}{2} + \frac{x_3}{3}\right)^2}{\left(\frac{y_2}{3} - \frac{3\,y_1}{2} + \frac{y_3}{3}\right)^2} + 1\right)\left(\frac{y_2}{3} - \frac{3\,y_1}{2} + \frac{y_3}{3}\right)}$$

Ecuación I.13. Derivada parcial 13

$$\frac{\partial \varphi}{\partial y_1} = \frac{\partial \operatorname{atan}\left(\frac{\frac{2}{3}x_1 - \frac{1}{3}x_2 - \frac{1}{3}x_3}{\frac{2}{3}y_1 - \frac{1}{3}y_2 - \frac{1}{3}y_3}\right)}{\partial y_1} = \frac{3\left(\frac{x_2}{3} - \frac{3\,x_1}{2} + \frac{x_3}{3}\right)}{2\left(\frac{\left(\frac{x_2}{3} - \frac{3\,x_1}{2} + \frac{x_3}{3}\right)^2}{\left(\frac{y_2}{3} - \frac{3\,y_1}{2} + \frac{y_3}{3}\right)^2} + 1\right)\left(\frac{y_2}{3} - \frac{3\,y_1}{2} + \frac{y_3}{3}\right)^2}$$

Ecuación I.14. Derivada parcial 14

$$\frac{\partial \varphi}{\partial x_2} = \frac{\partial \operatorname{atan}\left(\frac{\frac{2}{3}x_1 - \frac{1}{3}x_2 - \frac{1}{3}x_3}{\frac{2}{3}y_1 - \frac{1}{3}y_2 - \frac{1}{3}y_3}\right)}{\partial x_2} = \frac{1}{3\left(\frac{\left(\frac{x_2}{3} - \frac{3\,x_1}{2} + \frac{x_3}{3}\right)^2}{\left(\frac{y_2}{3} - \frac{3\,y_1}{2} + \frac{y_3}{3}\right)^2} + 1\right)\left(\frac{y_2}{3} - \frac{3\,y_1}{2} + \frac{y_3}{3}\right)}$$

Ecuación I.15. Derivada parcial 15

$$\frac{\partial \varphi}{\partial y_2} = \frac{\partial \operatorname{atan}\left(\frac{\frac{2}{3}x_1 - \frac{1}{3}x_2 - \frac{1}{3}x_3}{\frac{2}{3}y_1 - \frac{1}{3}y_2 - \frac{1}{3}y_3}\right)}{\partial y_2} = \frac{\frac{x_2}{3} - \frac{3\,x_1}{2} + \frac{x_3}{3}}{3\left(\frac{\left(\frac{x_2}{3} - \frac{3\,x_1}{2} + \frac{x_3}{3}\right)^2}{\left(\frac{y_2}{3} - \frac{3\,y_1}{2} + \frac{y_3}{3}\right)^2} + 1\right)\left(\frac{y_2}{3} - \frac{3\,y_1}{2} + \frac{y_3}{3}\right)^2}$$

Ecuación I.16. Derivada parcial 16

Debido a que las derivadas faltantes de la Ecuación 2.11 son expresiones muy largas, se incluye en el Anexo II el código del script de Matlab que permite obtenerlas.

ANEXO II

SCRIPT EN MATLAB PARA DETERMINAR DERIVADAS PARCIALES

```
clc;
clear all;
syms x1 y1 x2 y2 x3 y3

%Ecuaciones
d1=sqrt((x1-x3)^2+(y1-y3)^2);
d2=sqrt((x1-x2)^2+(y1-y2)^2);
d3=sqrt((x2-x3)^2+(y2-y3)^2);
fact1=(d1^2)+(d2^2)-(d3^2);
fact2=2*d1*d2;
beta=acos(fact1/fact2);
xc=(x1+x2+x3)/3;
yc=(y1+y2+y3)/3;
aux1=(2/3)*x1-(1/3)*x2-(1/3)*x3;
aux2=(2/3)*y1-(1/3)*y2-(1/3)*y3;
phi=atan(aux1/aux2);

%% Derivadas variables de forma
jf11=diff(d1,x1)
dd1x1=simplify(jf11)
jf12=diff(d1,y1)
dd1y1=simplify(jf12)
jf13=diff(d1,x2)
dd1x2=simplify(jf13)
jf14=diff(d1,y2)
dd1y2=simplify(jf14)
jf15=diff(d1,x3)
dd1x3=simplify(jf15)
jf16=diff(d1,y3)
dd1y3=simplify(jf16)

jf21=diff(d2,x1)
dd2x1=simplify(jf21)
jf22=diff(d2,y1)
dd2y1=simplify(jf22)
jf23=diff(d2,x2)
dd2x2=simplify(jf23)
jf24=diff(d2,y2)
dd2y2=simplify(jf24)
jf25=diff(d2,x3)
dd2x3=simplify(jf25)
jf26=diff(d2,y3)
dd2y3=simplify(jf26)

jf31=diff(beta,x1)
jf32=diff(beta,y1)
jf33=diff(beta,x2)
jf34=diff(beta,y2)
jf35=diff(beta,x3)
jf36=diff(beta,y3)
```

```
%Derivadas de variables de postura
jp11=diff(xc,x1)
dxcx1=simplify(jp11)
jp12=diff(xc,y1)
dxcy1=simplify(jp12)
jp13=diff(xc,x2)
dxcx2=simplify(jp13)
jp14=diff(xc,y2)
dxcy2=simplify(jp14)
jp15=diff(xc,x3)
dxcx3=simplify(jp15)
jp16=diff(xc,y3)
dxcy3=simplify(jp16)

jp21=diff(yc,x1)
dycx1=simplify(jp21)
jp22=diff(yc,y1)
dycy1=simplify(jp22)
jp23=diff(yc,x2)
dycx2=simplify(jp23)
jp24=diff(yc,y2)
dycy2=simplify(jp24)
jp25=diff(yc,x3)
dycx3=simplify(jp25)
jp26=diff(yc,y3)
dycy3=simplify(jp26)

jp31=diff(phi,x1)
jp32=diff(phi,y1)
jp33=diff(phi,x2)
jp34=diff(phi,y2)
jp35=diff(phi,x3)
jp36=diff(phi,y3)
```

ANEXO III
INSTALACIÓN, CREACIÓN DE ÁREA DE TRABAJO Y COMANDOS DE ROS

III.1. Instalación de ROS

Para la instalación y configuración de ROS en la distribución de UBUNTU de Linux se realiza los siguientes pasos, los cuales podemos encontrar detalladamente en la página oficial de ROS [ros wiki].

El primer paso es ingresar a la consola de Ubuntu e ingresar el comando:

- *$ sudo apt-get install ros-kinetic-desktop*

III.2. Creación de área de Trabajo

Luego se crea un entorno o directorio donde crearemos e importaremos los paquetes necesarios para el sistema, el directorio por defecto se lo crea con los comandos:

- *$ echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc*
- *$ source ~/.bashrc*

Para crear un área de trabajo en un directorio personal usamos el siguiente comando

- *$ mkdir -p ~/catkin_ws/src*
- *$ cd ~/catkin_ws/src*
- *$ catkin_init_workspace*

De esta manera el área de trabajo tiene el nombre de catkin_ws, en esta se crearán los archivos necesarios para el funcionamiento de los diferentes paquetes y nodos.

III.3. Comandos de ROS

A continuación, se da una lista de los comandos más comúnmente usados en ROS.

- *$ roscore* comando para iniciar el nodo master de ROS en Ubuntu, este debe estar siempre activado para permitir la comunicación entre los nodos.
- *$ rosrun* permite ejecutar los scripts o programas de diferentes paquetes de ROS sin cambiar el directorio.
- *$ roscd* cambia el directorio actual del terminal hacia la carpeta del área de trabajo de ROS.
- *$ roslaunch* permite ejecutar el nodo máster y varios nodos simultáneamente sin la necesidad de ejecutar antes el comando *roscore.*

- *$ rosnode list* muestra la lista de todos los nodos ejecutándose actualmente.
- *$ rosnode kill <nodo>* termina el proceso del nodo seleccionado.
- *$ rostopic echo* muestra el mensaje publicado a través del tópico seleccionado.
- *$ rostopic pub* publica un mensaje a través del tópico seleccionado.
- *$ rqt_graph* se abre una ventana con un diagrama de flujo de la interacción entre nodos.
- *$ killall -9 rosmaster* termina todo el proceso del núcleo de ROS.
- *$ catkin_create_pkg <nombre> [dependencia1] [dependencia2]* permite crear un paquete de ros con las dependencias que seleccionemos.
- *$ catkin_make* compila los paquetes y cambios realizados en estos.
- *$ catkin_make install* compila los mensajes de los paquetes de ROS.
- `$ rosrun rviz rviz` ejecución de modulo rviz de ROS.

ANEXO IV
VALORES DEFAULT DE PARÁMETROS PARA DETECCIÓN DE MARCADORES

En la Figura IV.1 se muestran el archivo *detector_params.yml* utilizado por la librería *detectMarkers()* donde se encuentra los valores por default de parámetros para la detección marcadores.

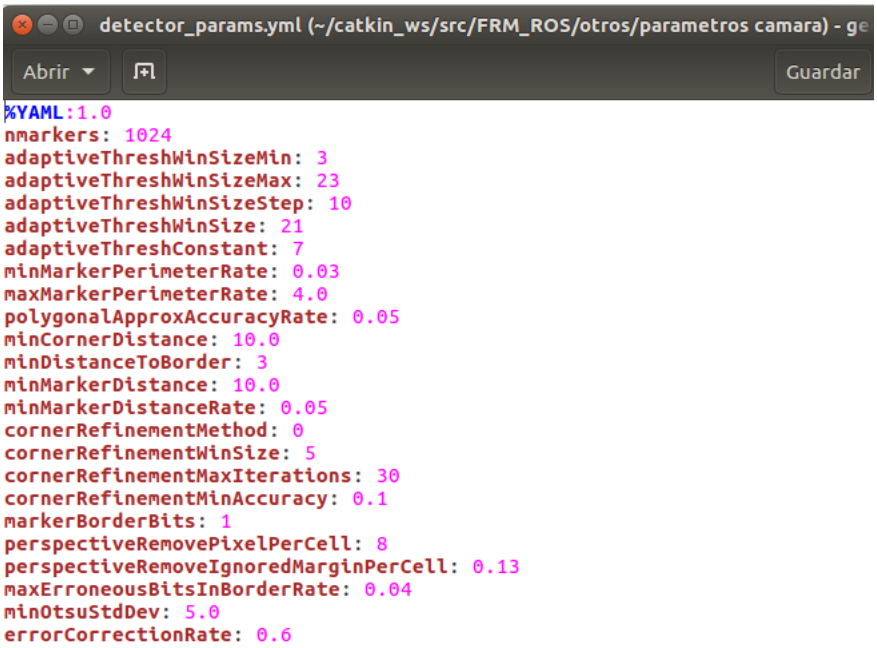


Figura IV.1. Archivo *detector_params.yml*.

ANEXO V
RESULTADOS TRAYECTORIA CIRCULAR CON DIFERENTES GANACIAS DE CONTROL EXTERNO

V.1. Señales de error

K= 0.04

En la Figura V.1 se muestran los errores de formación y postura, puesto que para Yc, la referencia es 0.59 m y la posición inicial del centroide es 0.39 m se tiene un error de 0.2 m, llegando a cero al finalizar la trayectoria. El ángulo β y φ presentan un error considerable debido a que el controlador con $K = 0.04$ no ofrece un buen control tanto para el seguimiento de trayectoria como para la corrección de parámetros de formación y postura.

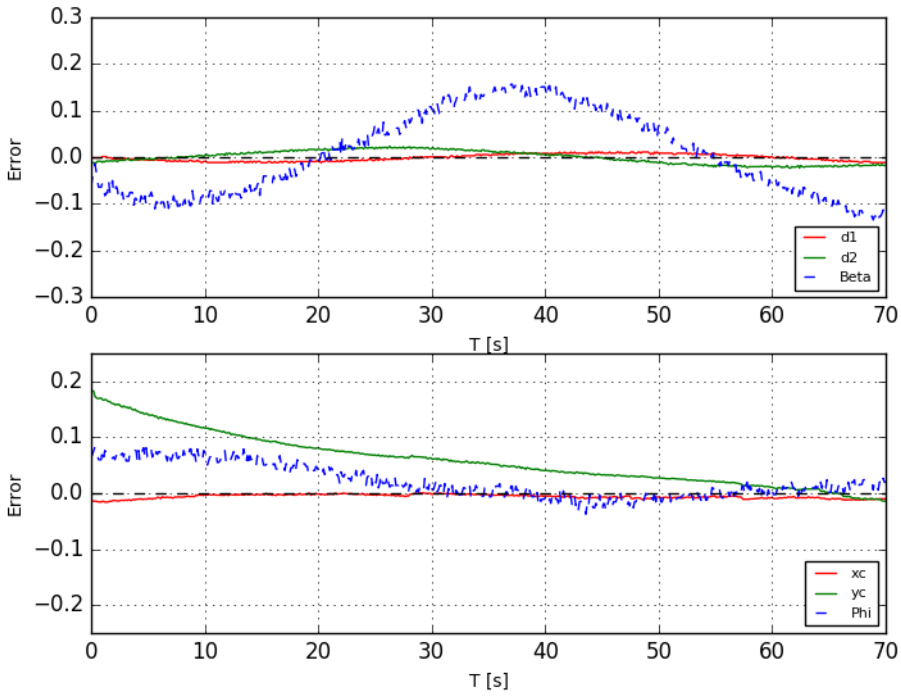


Figura V.1. Errores de forma y postura – K=0.04

K= 0.3

En la Figura V.2 se observa que el error dado por los ángulos β y φ se redujo y el error de posición en Yc se corrige más rápido llegando a cero en 8 s.

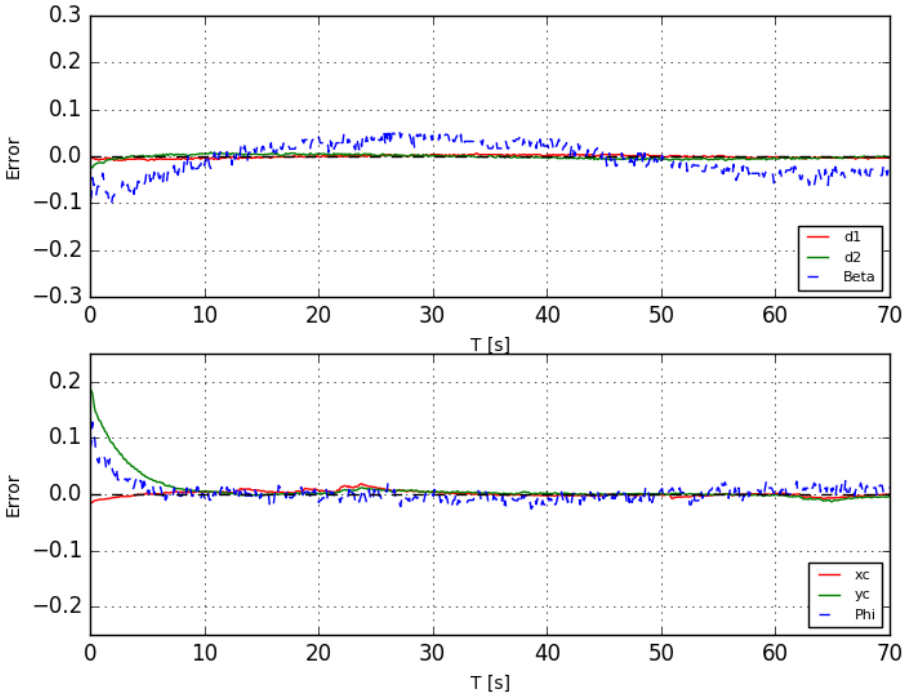


Figura V.2. Errores de forma y postura - K=0.3

K= 0.7

En la Figura V.3 se observa que el error dado por los ángulos β y φ se mantiene en cero y el error de posición Yc llega a cero en 5 s.

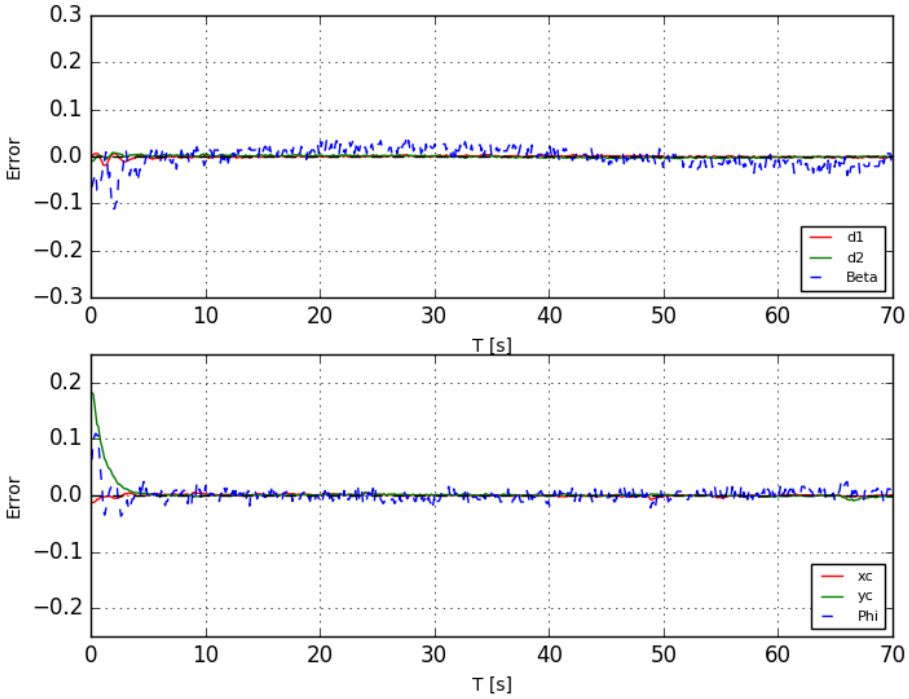


Figura V.3. Errores de forma y postura - K=0.7

V.2. Señales de control

En la Figura V.4 y Figura V.5 se muestran las señales de control de las velocidades izquierda y derecha para los robots móviles r_2 y r_3, respectivamente. Estos resultados son considerando los diferentes valores de controlador K. En las gráficas se puede observar que para valores más elevados de K las señales de control tienen picos más elevados y son más oscilatorias hasta estabilizarse.

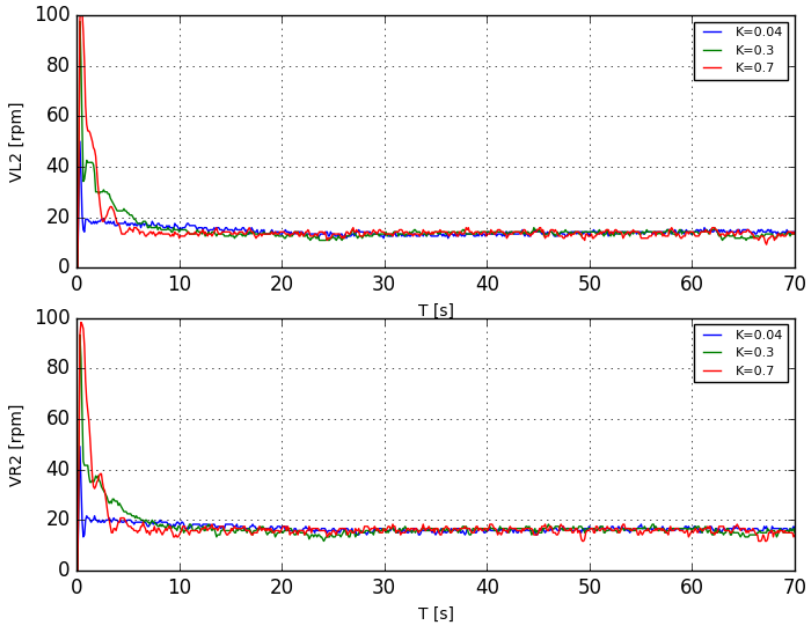


Figura V.4. Señales de control de las velocidades de rueda izquierda y derecha del robot móvil r_2.

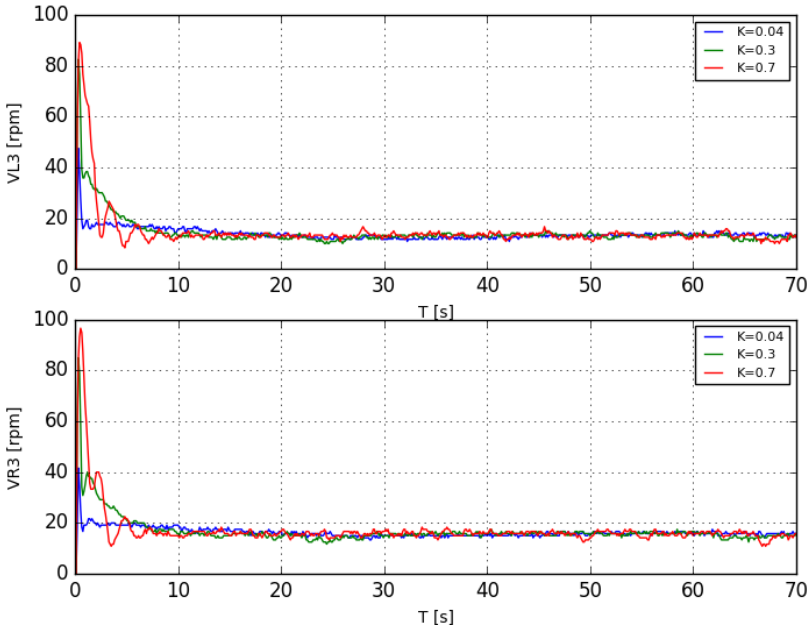


Figura V.5. Señales de control de las velocidades de rueda izquierda y derecha del robot móvil r_3.

ANEXO VI

RESULTADOS TRAYECTORIA CIRCULAR ANTE DIFERENTES VELOCIDADES DE REFERENCIA

VI.1. Señales de error

V = 0.05 m/s

En la Figura VI.1 se muestran los errores de formación y postura para una velocidad de referencia de 0.05 m/s, puesto que a bajas velocidades el controlador funciona de mejor manera, el error dado por el ángulo β y φ se mantiene en cero. El error de la posición inicial se corrige en 8 segundos aproximadamente.

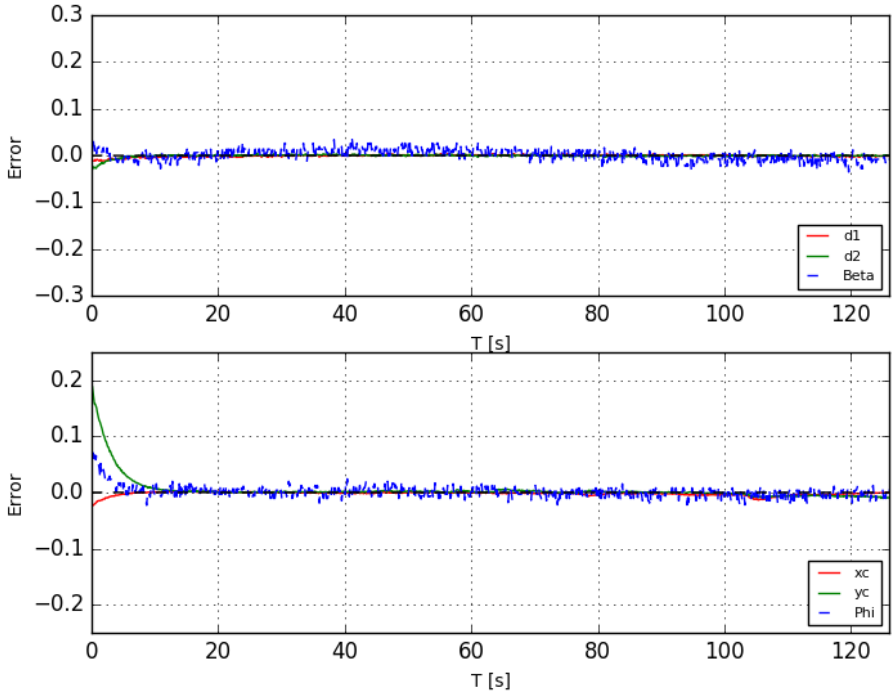


Figura VI.1. Errores de forma y postura - V=0.05 m/s

V = 0.09 m/s

En la Figura VI.2 se muestran los errores de formación y postura para una velocidad de referencia de 0.09 m/s. El error dado por el ángulo β y φ se mantiene en cero y el error de posición inicial se vuelve cero a los 10 s.

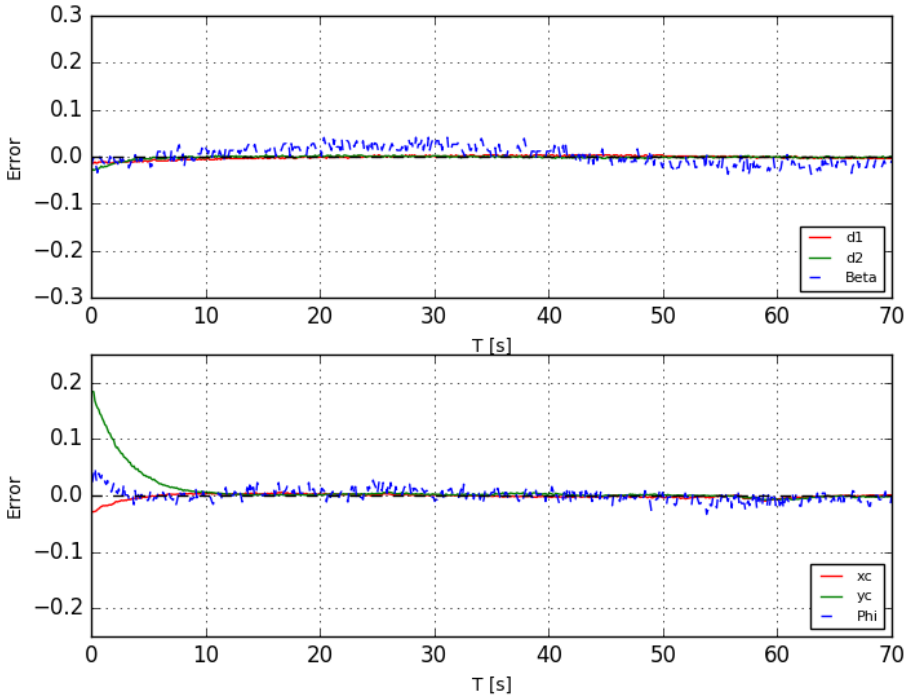


Figura VI.2. Errores de forma y postura - V=0.09 m/s

V = 0.2 m/s

En la Figura VI.3 se muestran los errores de formación y postura para este caso. Debido a que para velocidades de referencia iguales o superiores a 0.2 m/s para la trayectoria circular, el controlador no actúa correctamente, por lo que los ángulos β y φ presentan oscilaciones y el error de posición inicial se estabiliza a cero luego de 10 segundos.

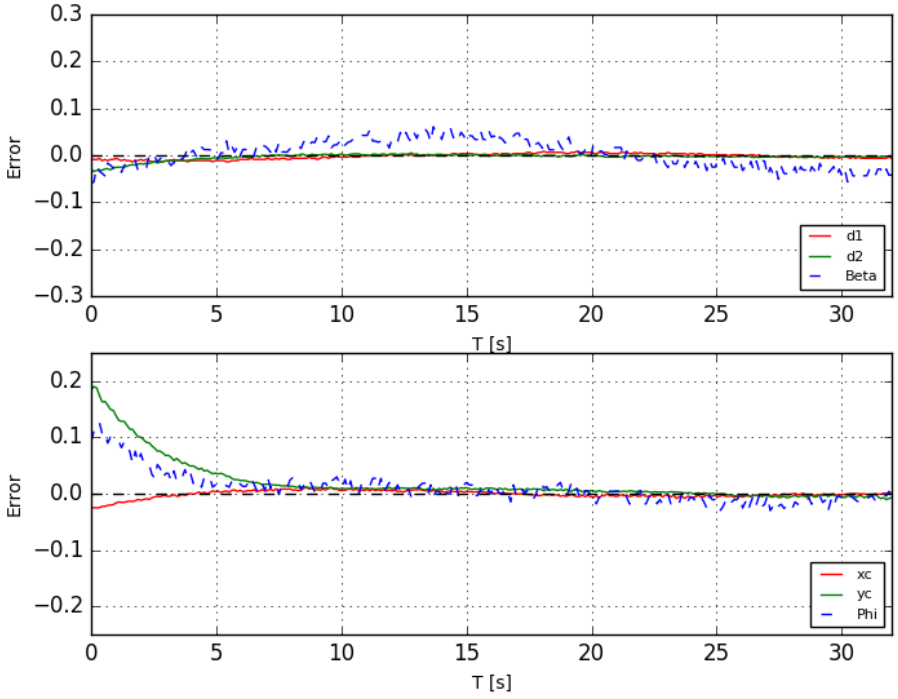


Figura VI.3. Errores de forma y postura - V=0.2 m/s

VI.2. Señales de control

En la Figura VI.4 y Figura VI.5 se muestran las señales de control de las velocidades izquierda y derecha del robot móvil r_2 ante los diferentes valores de velocidad de referencia de la formación 0.05 m/s, 0.09 m/s y 0.2 m/s.

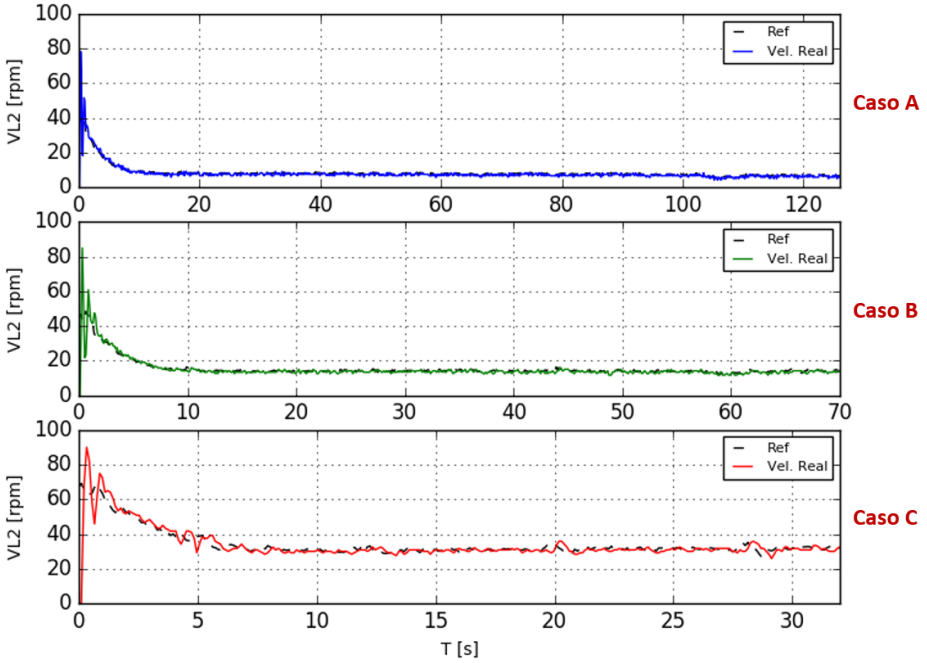


Figura VI.4. Señales de control de velocidad de la rueda izquierda del robot móvil r_2.

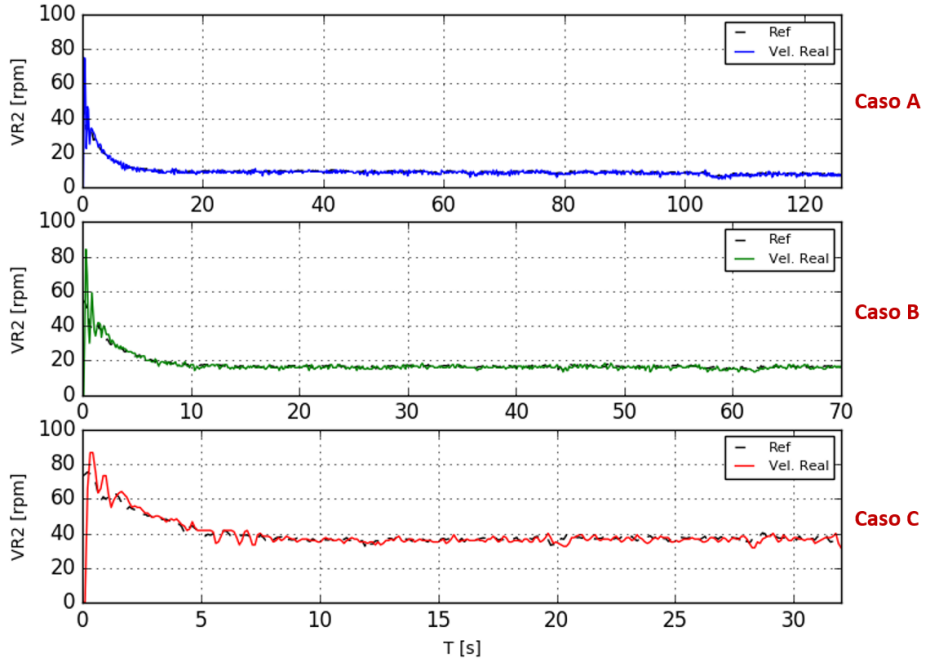


Figura VI.5. Señales de control de velocidad de la rueda derecha del robot móvil r_2.

En la Figura VI.6 y Figura VI.7 se muestran las señales de control de las velocidades izquierda y derecha del robot móvil r_3.

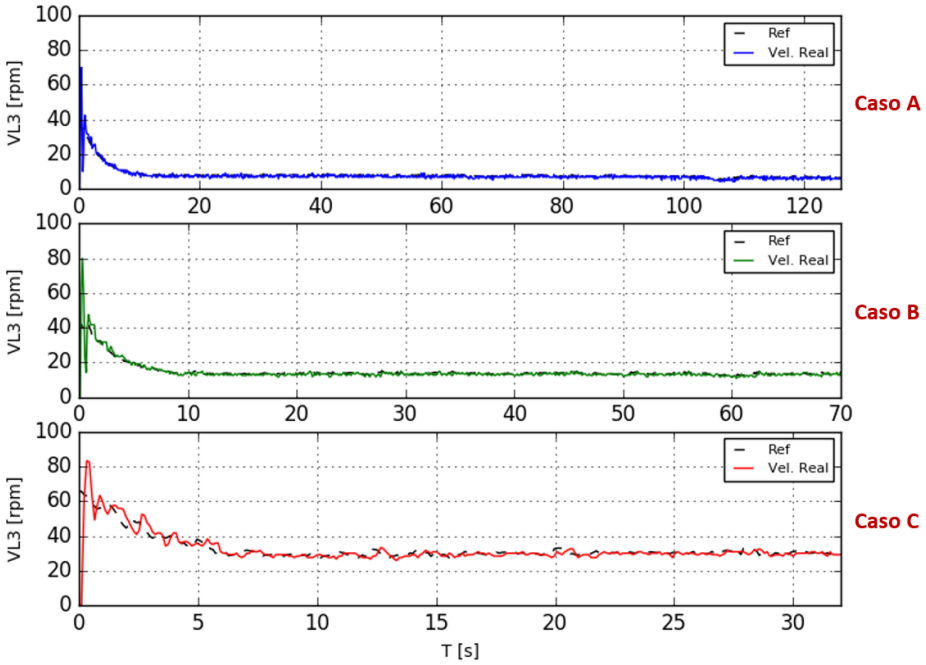


Figura VI.6. Señales de control de velocidad de la rueda izquierda del robot móvil r_3.

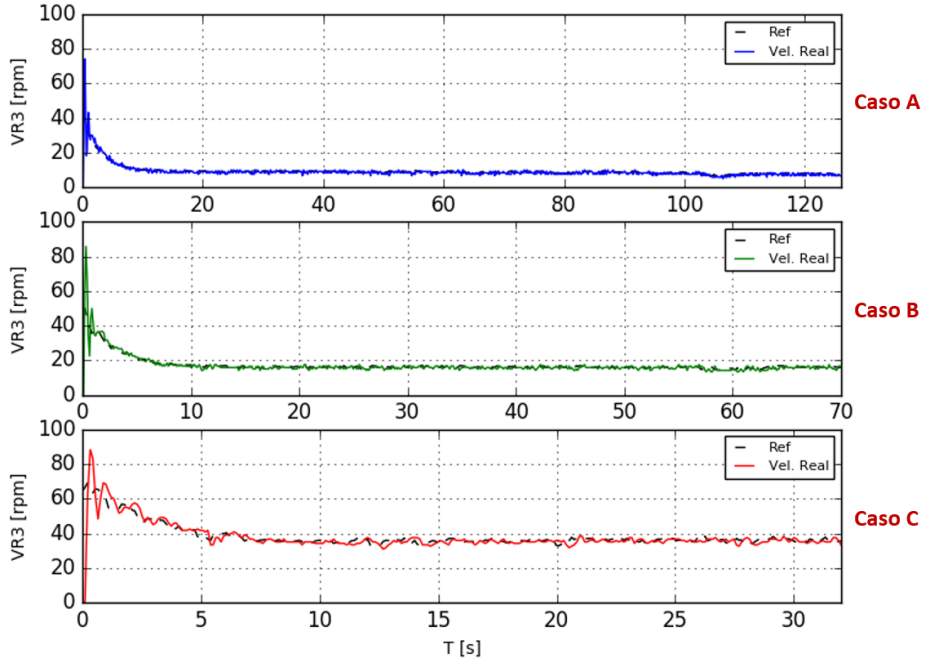


Figura VI.7. Señales de control de velocidad de la rueda derecha del robot móvil r_3.

ANEXO VII

RESULTADOS TRAYECTORIA CUADRADA ANTE DIFERENTES VELOCIDADES DE REFERENCIA

VII.1. Señales de error

V = 0.02 m/s

En la Figura VII.1 se muestran los errores de formación y postura para una velocidad de referencia de 0.02 m/s, el ángulo β presenta ciertas oscilaciones mientras que el ángulo φ se mantiene en cero.

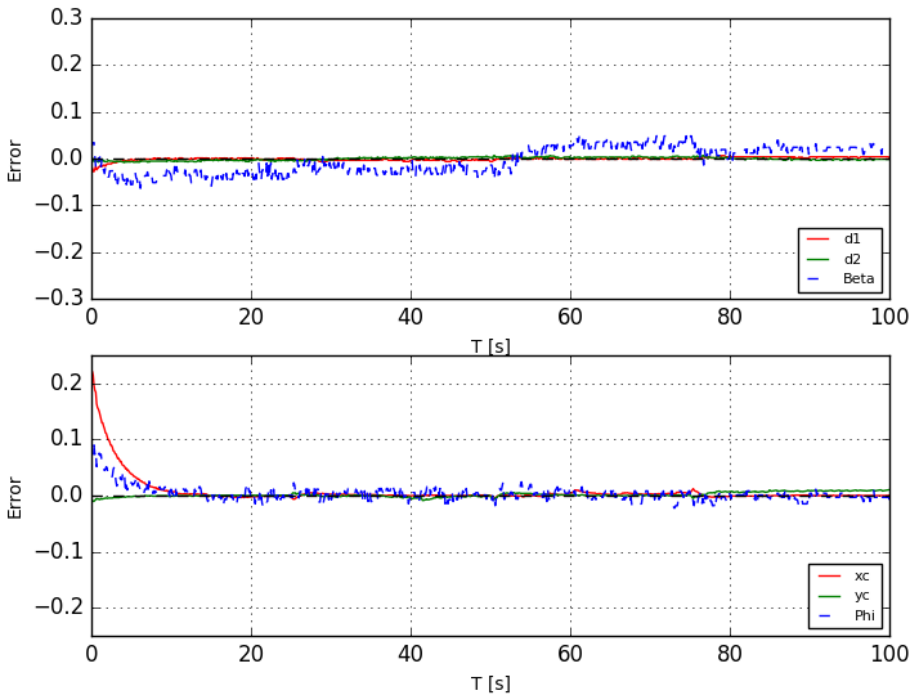


Figura VII.1. Errores de forma y postura – V=0.02 m/s

V = 0.035 m/s

En la Figura VII.2 se muestran los errores de formación y postura para una velocidad lineal de 0.035 m/s, el ángulo β y φ presentan aumento de error al pasar por las esquinas de la trayectoria cuadrada y luego vuelven a disminuir.

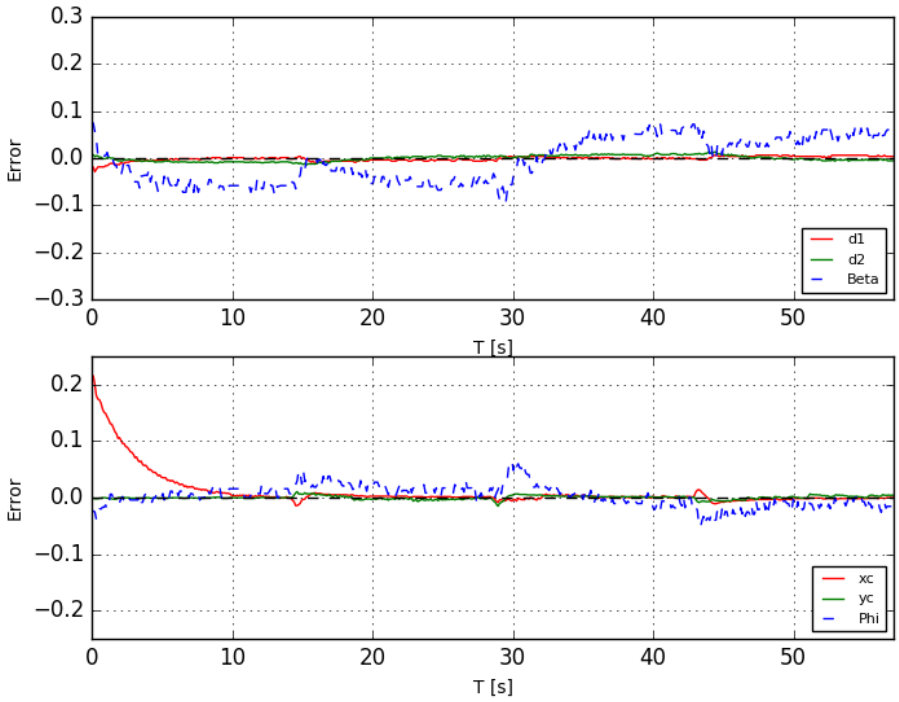


Figura VII.2. Errores de forma y postura - V=0.035 m/s

V = 0.06 m/s

En la Figura VII.3 se observa que a velocidades superiores a 0.06 m/s en la trayectoria cuadrada el controlador no actúa correctamente, todas las señales presentan oscilación siendo las más afectadas los ángulos β y φ.

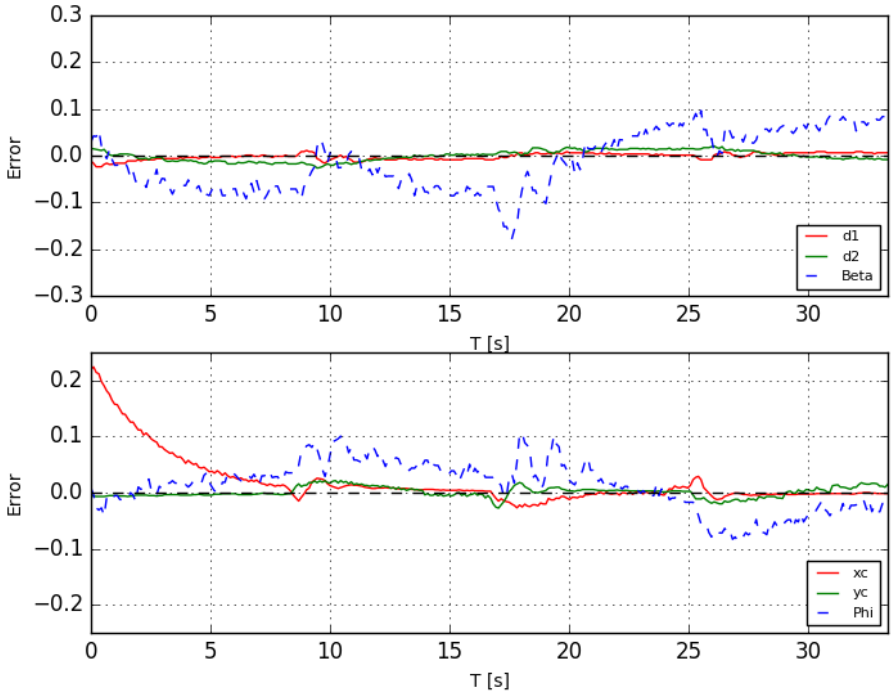


Figura VII.3. Errores de forma y postura - V=0.06 m/s

VII.2. Señales de control

En la Figura VII.4 y Figura VII.5 se muestran las señales de control de las velocidades izquierda y derecha del robot móvil r_2 ante los diferentes valores de velocidad lineal de la formación 0.02 m/s, 0.035 m/s y 0.06 m/s. En las gráficas se puede observar que a velocidades más elevadas las señales de control son más oscilatorias y no llegan a estabilizarse.

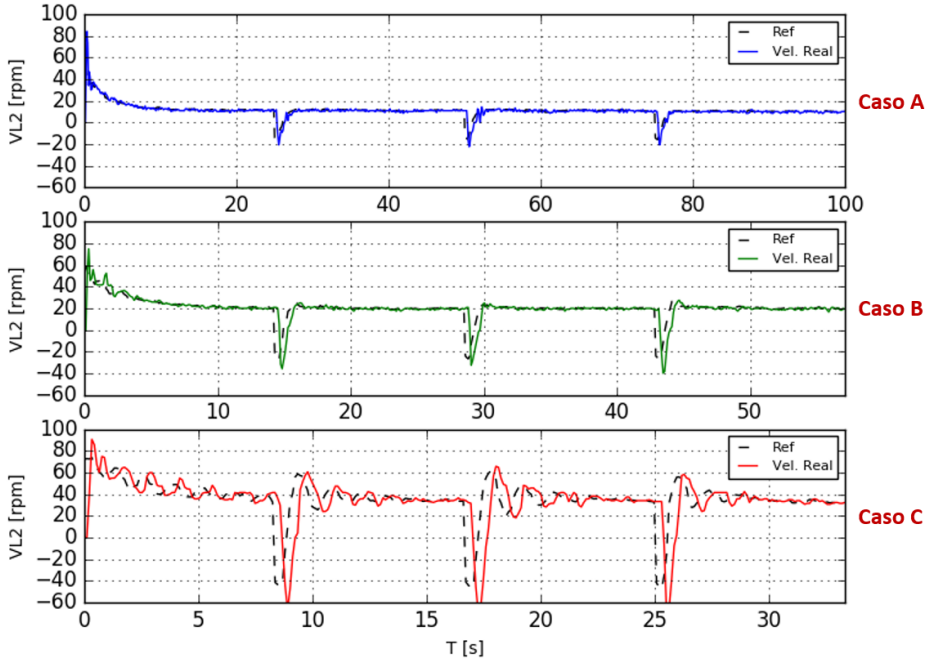


Figura VII.4. Señales de control de velocidad de la rueda izquierda del robot móvil r_2.

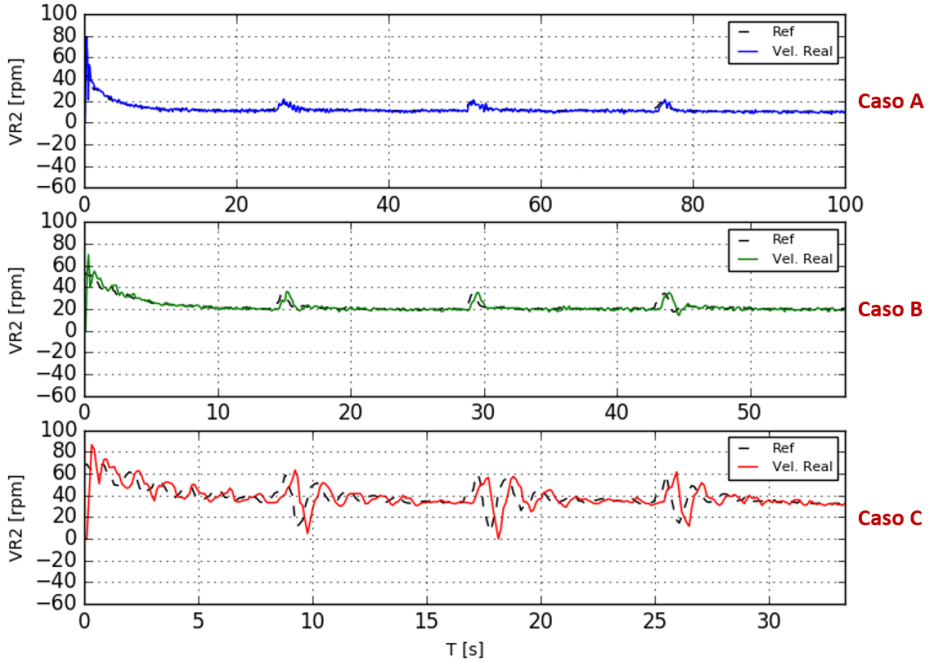


Figura VII.5. Señales de control de velocidad de la rueda derecha del robot móvil r_2.

En la Figura VII.6 y Figura VII.7 se muestran las señales de control de las velocidades izquierda y derecha del robot móvil r_3.

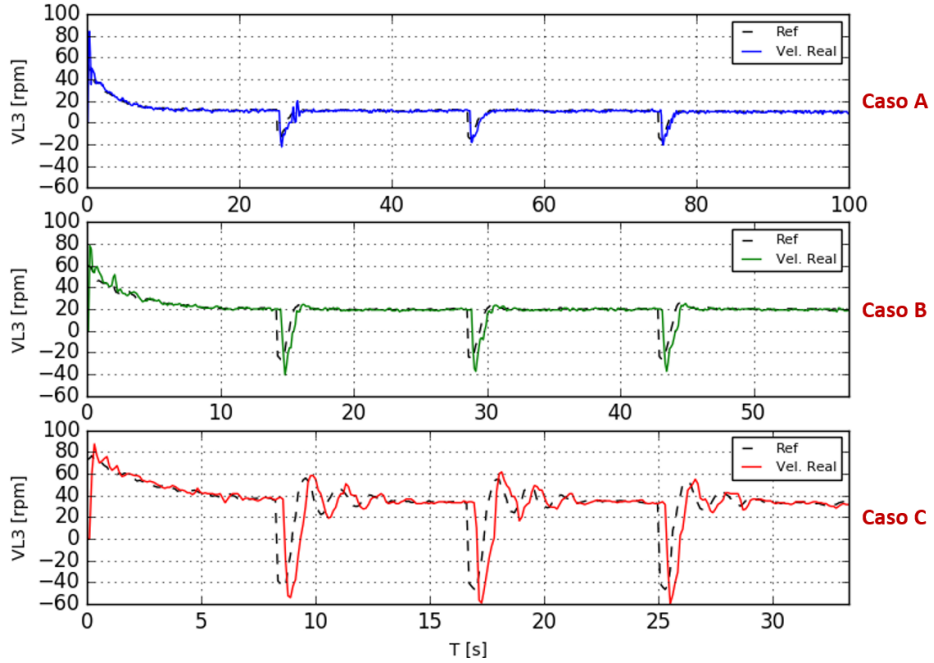


Figura VII.6. Señales de control de velocidad de la rueda izquierda del robot móvil r_3.

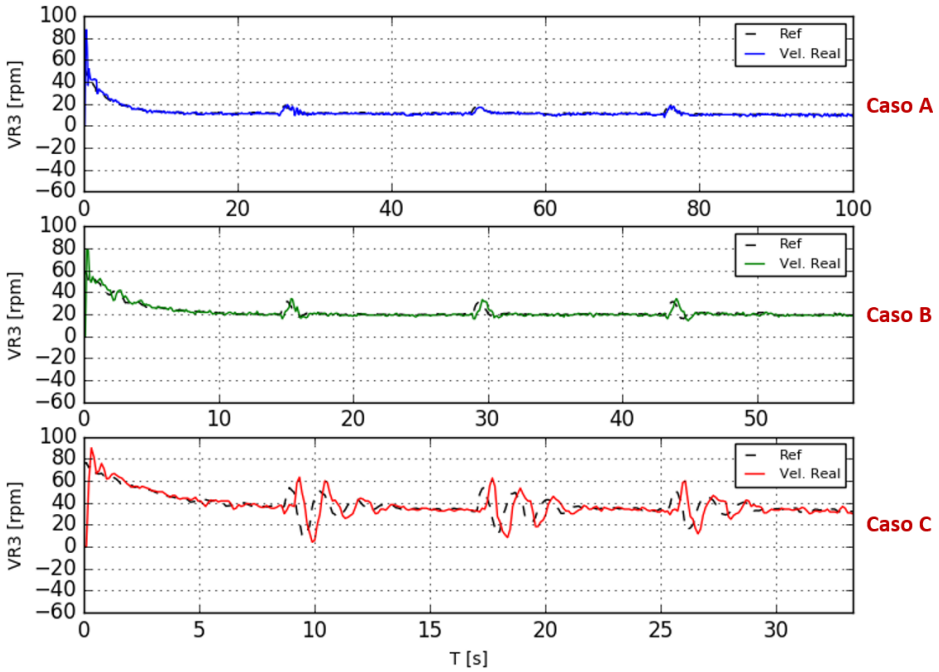


Figura VII.7. Señales de control de velocidad de la rueda derecha del robot móvil r_3.

ANEXO VIII

RESULTADOS TRAYECTORIA DOBLE FRECUENCIA ANTE DIFERENTES VELOCIDADES DE REFERENCIA

VIII.1. Señales de error

V = 0.03 m/s

En la Figura VIII.1 se muestran los errores de formación y postura para una velocidad de referencia de 0.03 m/s, todas las señales se mantienen a cero debido a que bajas velocidades el controlador funciona correctamente.

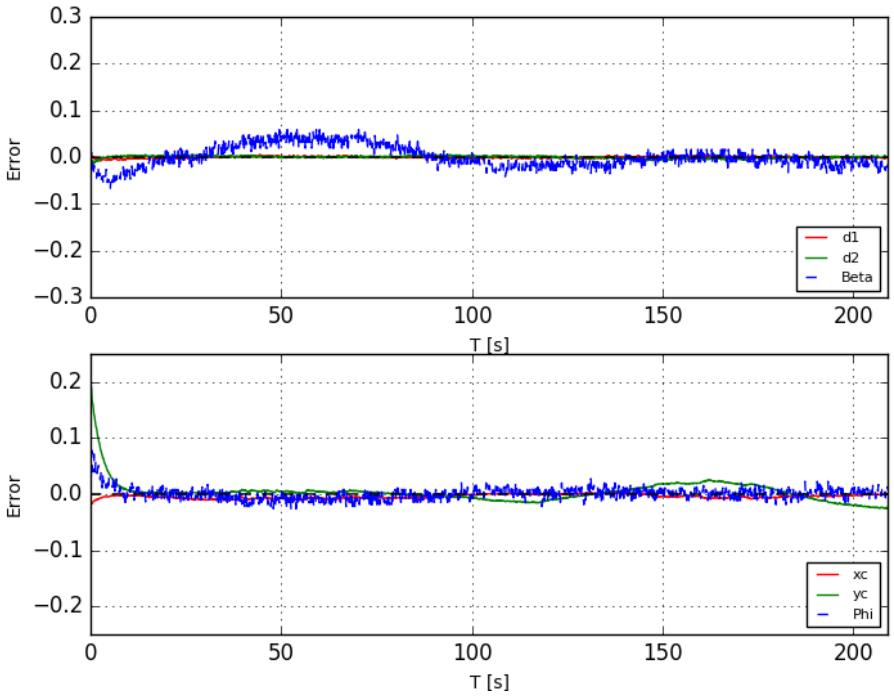


Figura VIII.1. Errores de forma y postura – V=0.03 m/s

V = 0.05 m/s

En la Figura VIII.2 se muestran los errores de formación y postura para una velocidad de referencia de 0.05 m/s, el ángulo β presenta un aumento de error mientras que el ángulo φ se mantiene en cero.

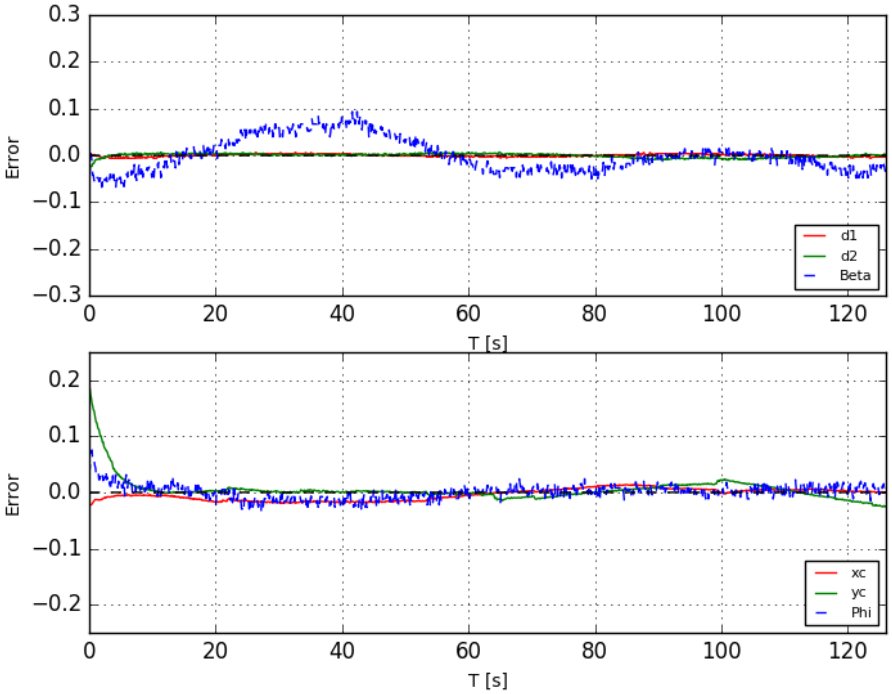


Figura VIII.2. Errores de forma y postura – V=0.05 m/s

V = 0.1 m/s

En la Figura VIII.3 se observa que a velocidades superiores a 0.1 m/s todas las señales de los parámetros de formación y postura se ven afectados teniendo oscilaciones en los ángulos β y φ en toda la trayectoria.

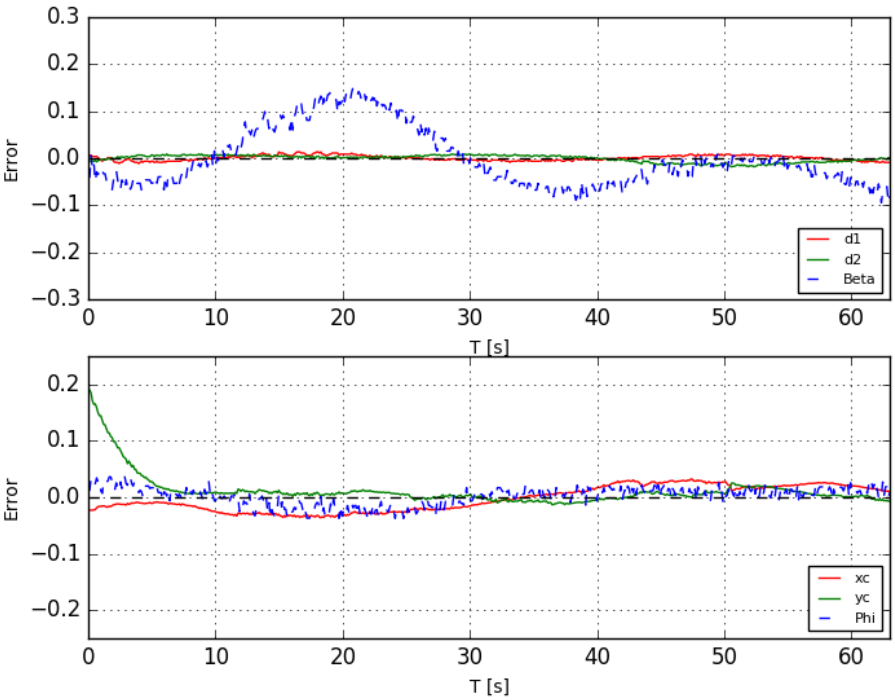


Figura VIII.3. Errores de forma y postura – V=0.1 m/s

VIII.2. Señales de control

En la Figura VIII.4 y Figura VIII.5 se muestran las señales de control de las velocidades izquierda y derecha del robot móvil r_2 ante los diferentes valores de velocidad de referencia de la formación 0.03 m/s, 0.05 m/s y 0.1 m/s.

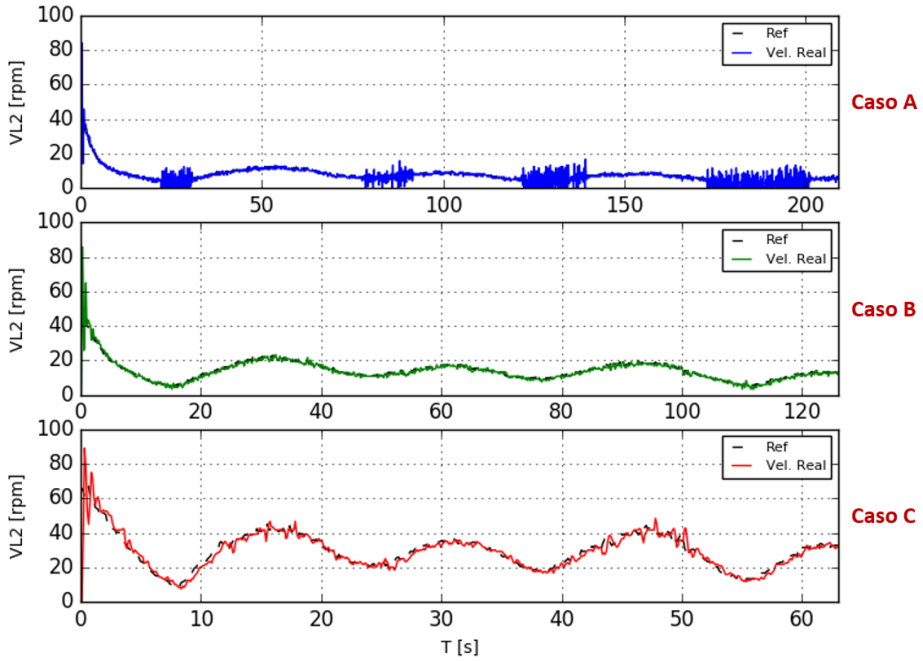


Figura VIII.4. Señales de control de velocidad de la rueda izquierda del robot móvil r_2.

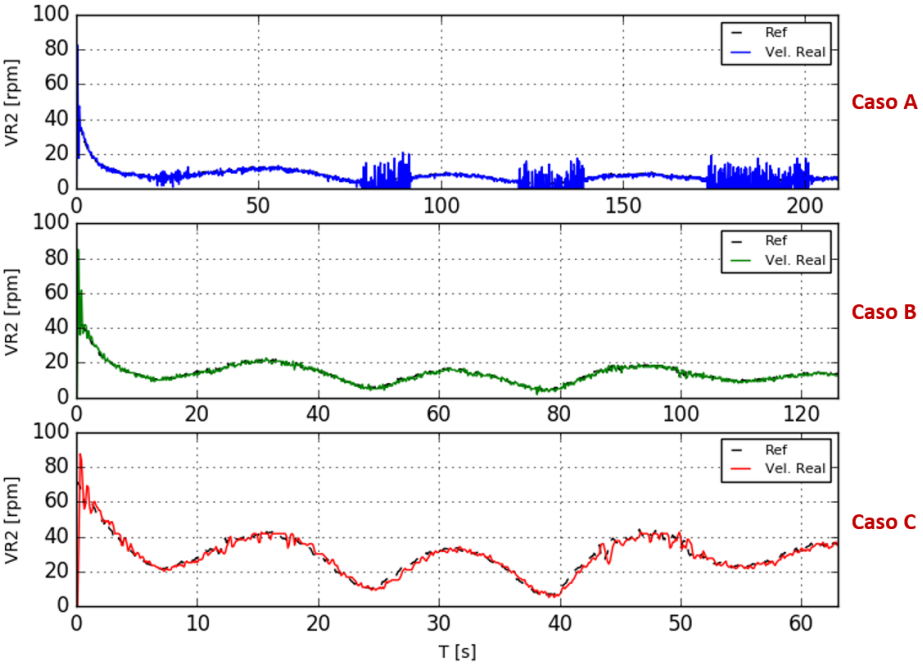


Figura VIII.5. Señales de control de velocidad de la rueda derecha del robot móvil r_2.

En la Figura VIII.6 y Figura VIII.7 se muestran las señales de control de las velocidades izquierda y derecha del robot móvil r_3.

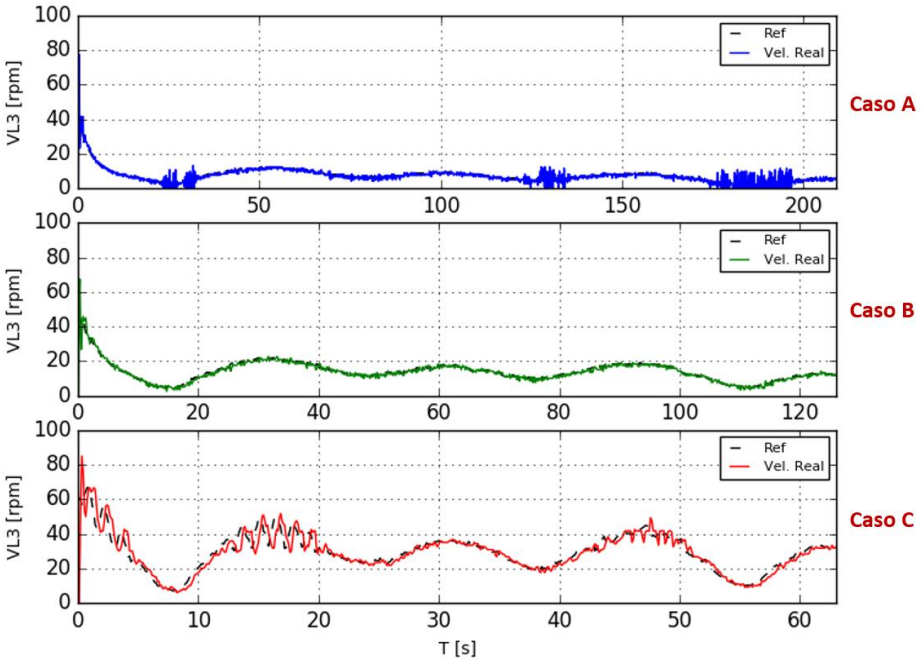


Figura VIII.6. Señales de control de velocidad de la rueda izquierda del robot móvil r_3.

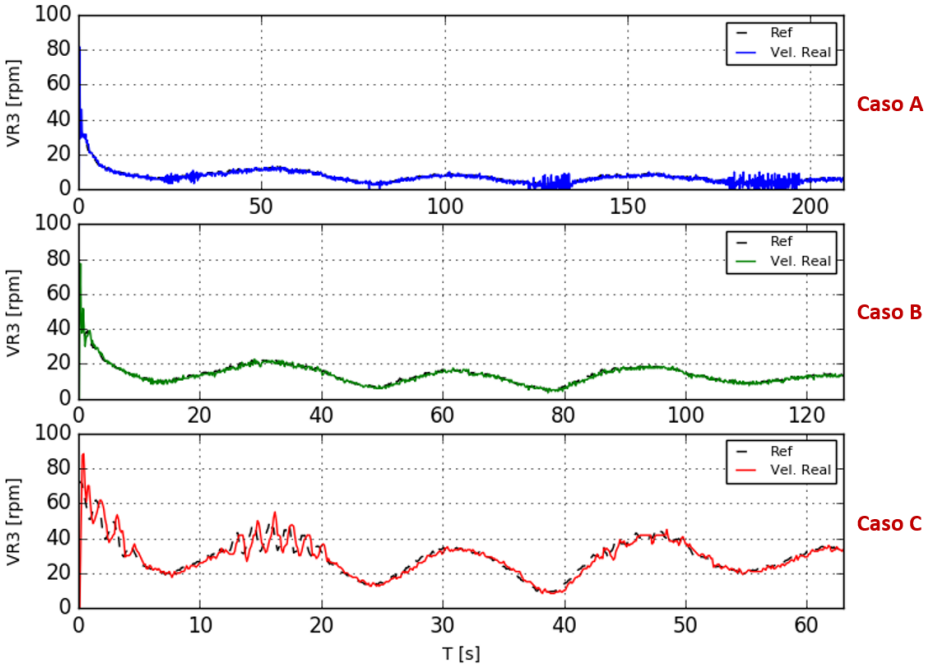


Figura VIII.7. Señales de control de velocidad de la rueda derecha del robot móvil r_3.

ANEXO IX

RESULTADOS TRAYECTORIA DOBLE FRECUENCIA ANTE PERTURBACIONES

IX.1. Señales de error

En la Figura IX.1 se muestran los errores de formación y postura para una velocidad de referencia de trayectoria de 0.05 m/s ante perturbaciones en 20, 60 y 90 segundos de la trayectoria. El error inicial en las distancias d_1 y d_2 se corrigen en 8 segundos y todas las señales aumentan en error cuando se producen las perturbaciones volviendo a cero en aproximadamente 8 segundos.

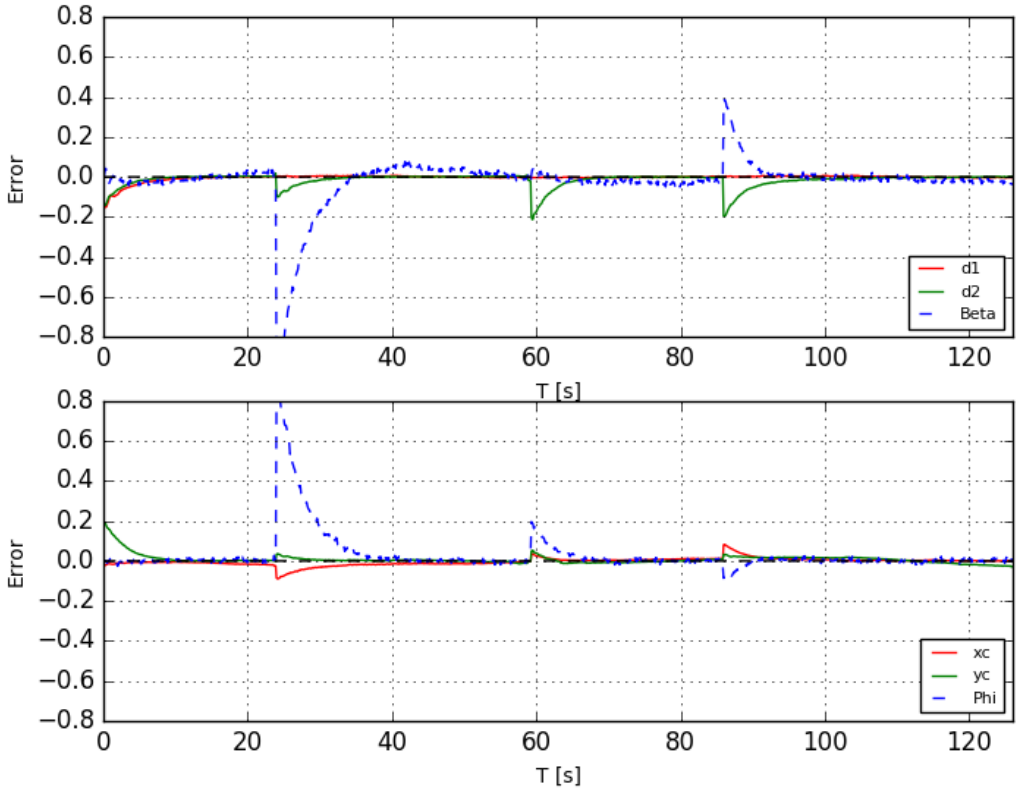


Figura IX.1. Errores de forma y postura ante perturbaciones

IX.2. Señales de control

En la Figura IX.2 y IX.3 se observan las señales de control de velocidad de los dos robots móviles r_1 y r_3.

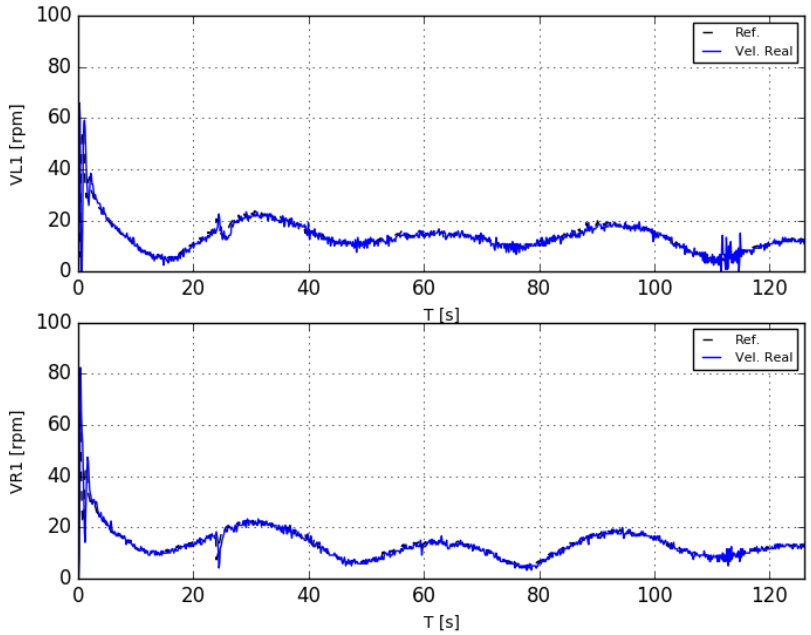


Figura IX.2 Señales de control de las velocidades de rueda izquierda y derecha del robot móvil r_1.

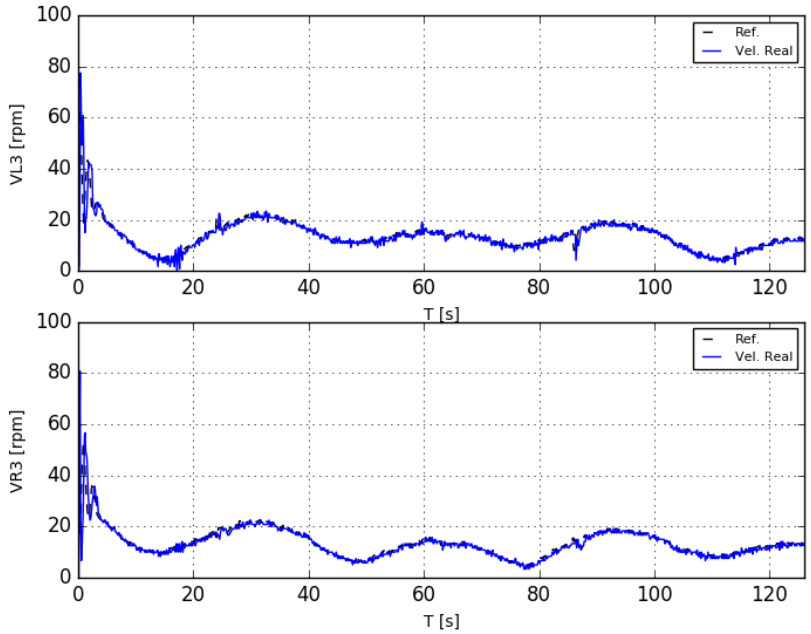


Figura IX.3 Señales de control de las velocidades de rueda izquierda y derecha del robot móvil r_3.

ANEXO X

MANUAL DE USUARIO

X.1. REQUISITOS DEL SISTEMA

A continuación, se describe los pasos para utilizar la plataforma de manera adecuada y los requisitos para su uso.

Software	Versión
Sistema Operativo	Linux Ubuntu 16.04.6 LTS
ROS Version	Kinetic Kame
Qt Creator	3.5.1

En caso de no tener instalado algunos de estos componentes, se los puede obtener a través de sus páginas oficiales, enlistadas a continuación:

- *https://ubuntu.com/*
- *http://wiki.ros.org/*
- *https://www.qt.io/download*

Para comprobar que ROS ha creado el entorno de trabajo utilizamos el comando.

- `$ roscd`

Como se observa a continuación en la Figura X.1.

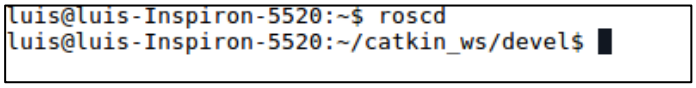


Figura X.1 Comprobación del entorno de trabajo de ROS.

X.1.1 Verificación o copia de paquete de ROS

Para verificar si están los archivos necesarios se accede al directorio:

/home/USER/catkin_ws/src/

Donde *USER* es el nombre de usuario que tiene el ordenador.

En este directorio se verifica si existe la carpeta o paquete *FRM_ROS* si no es así se copia el paquete de los archivos de respaldo.

Una vez terminado esto se abre la carpeta *FRM_ROS* que debe contener los directorios *include, launch, msg, src, srv*, como se observa en la Figura X.2.

Figura X.2 Paquete FRM_ROS.

Si se va a iniciar el sistema por primera vez se recomienda compilar el paquete FRM_ROS utilizando los comandos en un terminal de Ubuntu como se observa en la Figura X.3.

- *$ roscd*
- *$ cd ..*
- *$ catkin_make*
- *$ catkin_make install*

Figura X.3 a) `Catkin_make` b) `Catkin_make install`.

X.1.2 Activación de los robots

Cada robot del banco de pruebas dispone de una batería de Lipo la cual debe ser previamente cargada empleando el módulo de carga SparkFun USB mostrado en la Figura

X.4. Se conecta la batería al módulo, y en caso de que se encuentre descargada completamente se encenderá un led rojo.

Figura X.4 Modulo de carga Sparkfun USB LiPoly Charger.

Una vez cargadas las baterías Lipo se procede a conectarlas a cada uno de los robots como se observa en la Figura X.5.

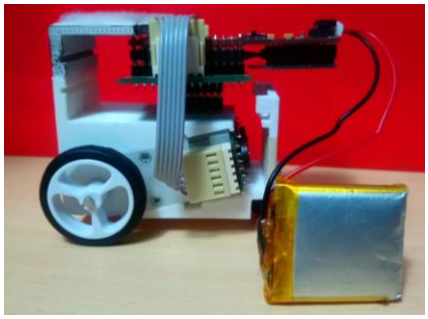
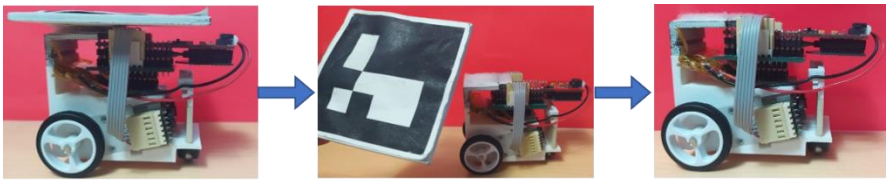
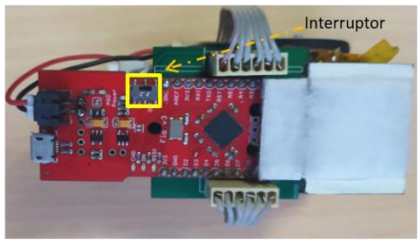


Figura X.5 Conexión de batería Lipo a tarjeta Fio V3.

Para la activación de los robots se debe en primer lugar retirar el marcador que se encuentra sobre el robot, como se indica en la Figura X.6 (a). Esto permite visualizar un interruptor que viene incorporado a la tarjeta Fio V3, como se observa en la Figura X.6 (b) el cual se lo coloca en la posición ON.



a)



b)

Figura X.6 a) Retiro del marcador b) Encendido del robot.

X.2. PROCEDIMIENTO DE EJECUCIÓN

Una vez instalados todos los programas y configurado el espacio de trabajo en ROS con las carpetas correspondientes se procede a ejecutar el sistema, para esto se ingresa en el terminal del ordenador el siguiente comando:

- *$ roscore*

Teniendo un resultado como el de la Figura X.7.

```
SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.14

NODES

auto-starting new master
process[master]: started with pid [14097]
ROS_MASTER_URI=http://luis-Inspiron-5520:11311/

setting /run_id to 704d5250-b661-11ea-8b9d-84a6c8aff0fa
WARNING: Package name "FRM_ROS" does not follow the naming conventions. It should
start with a lower case letter and only contain lower case letters, digits, unders
cores, and dashes.
process[rosout-1]: started with pid [14110]
started core service [/rosout]
```

Figura X.7 Roscore.

Al ejecutarse el roscore permite que los nodos de ROS se comuniquen entre sí, así que se prosigue a ejecutar los nodos en base al siguiente orden.

X.2.1 Arranque Nodo *usb_cam*

El nodo *usb_cam* permite interactuar con la cámara USB Logitech, para ejecutarlo se ingresa en el terminal del ordenador el siguiente comando:

- *$ rosrun usb_cam usb_cam_node*

En caso de que la cámara no se encuentre conectada al ordenador aparece un error como el de la Figura X.8, caso contrario se muestra como en la Figura X.9

```
luis@luis-Inspiron-5520:~$ rosrun usb_cam usb_cam_node
[ INFO] [1593034134.429344213]: using default calibration URL
[ INFO] [1593034134.429515056]: camera calibration URL: file:///home/luis/.ros/cam
era_info/head_camera.yaml
[ INFO] [1593034134.431241842]: Starting 'head_camera' (/dev/video1) at 640x480 vi
a mmap (yuyv) at 30 FPS
[ERROR] [1593034134.431368157]: Cannot identify '/dev/video1': 2, No such file or
directory
```

Figura X.8 Error arranque Nodo *usb_cam*.

```
luis@luis-Inspiron-5520:~$ rosrun usb_cam usb_cam_node
[ INFO] [1593034200.253806831]: using default calibration URL
[ INFO] [1593034200.253971028]: camera calibration URL: file:///home/luis/.ros/cam
era_info/head_camera.yaml
[ INFO] [1593034200.255577215]: Starting 'head_camera' (/dev/video1) at 640x480 vi
a mmap (yuyv) at 30 FPS
```

Figura X.9 Arranque Nodo *usb_cam*.

X.2.2 Arranque Nodo *v_artificial*

El nodo *v_artificial* permite activar el sistema de monitoreo desarrollado en C++ con la ayuda de las librerías de Open CV. Para ejecutarlo se ingresa en el terminal del ordenador el siguiente comando:

- *$ rosrun FRM_ROS v_artificial_node*

Teniendo como resultado el despliegue de dos ventanas como se muestra en la Figura X.10. y en el terminal se indica la ubicación del cursor verde, Figura X.11.

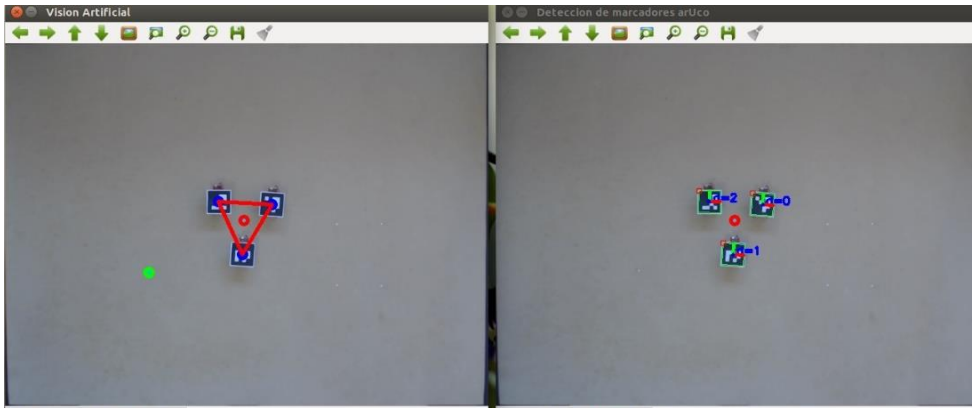


Figura X.10 Arranque Nodo *v_artificial*.

```
luis@luis-Inspiron-5520:~$ rosrun FRM_ROS v_artificial_node
[0.471074, 0.438843]
```

Figura X.11 Arranque Nodo *v_artificial* en el terminal.

X.2.3 Arranque Nodo *Response_robots*

Los nodos *Response_robots* y *Request_robots* permiten la comunicación entre los robots móviles y el ordenador, para ejecutar el nodo *Response_robots* se ingresa en el terminal del ordenador el siguiente comando:

- *$ rosrun FRM_ROS Response_robots.py*

Es necesario conectar el módulo Xbee Coordinador a la computadora, como se indica en la Figura X.12, en caso de no hacerlo aparece un error en el terminal como en el de la Figura X.13, caso contrario se muestra como en la Figura X.14.

Figura X.12 Conexión de módulo Xbee Coordinador.

```
luis@luis-Inspiron-5520:~$ rosrun FRM_ROS Response_robots.py
Traceback (most recent call last):
  File "/home/luis/catkin_ws/src/FRM_ROS/src/Response_robots.py", line 29, in <mod
ule>
    ser = serial.Serial(DEVICE, 57600)
  File "/usr/lib/python2.7/dist-packages/serial/serialutil.py", line 180, in __ini
t__
    self.open()
  File "/usr/lib/python2.7/dist-packages/serial/serialposix.py", line 294, in open
    raise SerialException(msg.errno, "could not open port %s: %s" % (self._port, m
sg))
serial.serialutil.SerialException: [Errno 2] could not open port /dev/ttyUSB0: [Er
rno 2] No such file or directory: '/dev/ttyUSB0'
```

Figura X.13 Error arranque Nodo *Response_robots.*

```
luis@luis-Inspiron-5520:~$ rosrun FRM_ROS Response_robots.py
Ready.
```

Figura X.14 Arranque Nodo *Response_robots*.

X.2.4 Arranque del Nodo *Request_robots*

Para ejecutar el nodo *Request_robots* se ingresa en el terminal del ordenador el siguiente comando:

- *$ rosrun FRM_ROS Request_robots.py*

Obteniendo un resultado como el de la Figura X.15.

```
#================VELOCIDADES================#
             --o--ARCOS-CALALA--o--

VelocidadR1:  0.000000 [rpm]   Ref: 0.000000 [rpm]     Activado:0.0
VelocidadL1:  0.000000 [rpm]   Ref: 0.000000 [rpm]
kpR1: 0.000000 kiR1: 0.000000 kdR1: 0.000000  Bateria1:  0.000000
kpL1: 0.000000 kiL1: 0.000000 kdL1: 0.000000

VelocidadR2:  0.000000 [rpm]   Ref: 0.000000 [rpm]     Activado:0.0
VelocidadL2:  0.000000 [rpm]   Ref: 0.000000 [rpm]
kpR2: 0.000000 kiR2: 0.000000 kdR2: 0.000000  Bateria2:  0.000000
kpL2: 0.000000 kiL2: 0.000000 kdL2: 0.000000

VelocidadR3:  0.000000 [rpm]   Ref: 0.000000 [rpm]     Activado:0.0
VelocidadL3:  0.000000 [rpm]   Ref: 0.000000 [rpm]
kpR3: 0.000000 kiR3: 0.000000 kdR3: 0.000000  Bateria3:  0.000000
kpL3: 0.000000 kiL3: 0.000000 kdL3: 0.000000
```

Figura X.15 Arranque Nodo *Request_robots*.

X.2.5 Arranque del Nodo *unidad_central*

Para el arranque del nodo *unidad_central* se ingresa en el terminal del ordenador el siguiente comando:

- *$ rosrun FRM_ROS unidad_central.py*

Teniendo un resultado como se observa en la Figura X.16.

```
################DATOS ODOMETRIA################
             --o--ARCOS-CALALA--o--

Id: 0                    Id: 1                    Id: 2
Pos. en x: 0.875 [m]     Pos. en x: 0.779 [m]     Pos. en x: 0.699 [m]
Pos. en y: 0.662 [m]     Pos. en y: 0.498 [m]     Pos. en y: 0.672 [m]
Angulo: 85.914 [deg]     Angulo: 85.914 [deg]     Angulo: 88.995 [deg]

DATOS FORMACION
d1: 0.176312 [m]
d2: 0.190070 [m]
d3: 0.190829 [m]

Betha: 62.645296 [deg]
Tetha: 60.592811 [deg]

Punto central x: 0.784298 [m]
Punto central y: 0.610744 [m]
MODO:EN ESPERA
```

Figura X.16 Arranque Nodo *unidad_central*.

X.2.6 Arranque de la Interfaz

Para el arranque de la interfaz se debe dirigir al directorio:

/home/USER/catkin_ws/src/FRM_ROS/

Allí se ejecuta el archivo HMI.pro en Qt Creator y se inicia la interfaz dando clic en la flecha verde tal como se ve en la Figura X.17.

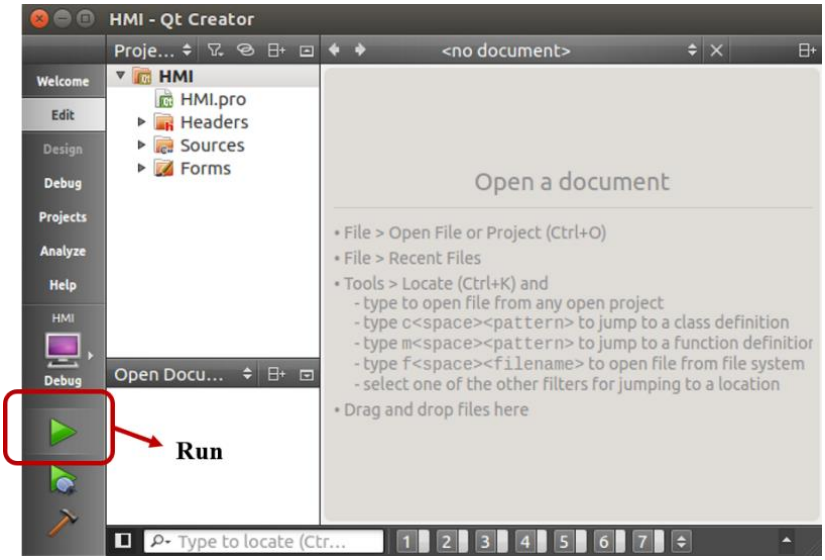


Figura X.17 Arranque Interfaz.

Una vez ejecutado el archivo se debería abrir una ventana de inicio como se ve en la Figura X.18.

Figura X.18 Ventana de presentación de Interfaz Gráfica

X.3. CONFIGURACIÓN DE PARÁMETROS

Al presionar el botón “Continuar” de la ventana de presentación se despliega la ventana principal en la cual el usuario debe seleccionar la trayectoria, y para cada una aparecerán valores predefinidos de parámetros de formación y trayectoria como se observa en las Figura X.19. El usuario puede variar dichos valores y se almacenan al presionar el botón “Guardar Parámetros”, cada vez que se modifique algún parámetro se debe volver a guardar caso contrario el seguimiento de trayectoria se realiza con los últimos valores guardados.

Figura X.19 Interfaz de usuario

Cada vez que se modifique algún parámetro para la generación de trayectoria, en el nodo *v_artificial* se grafica la trayectoria que va a recorrer la formación como se observa en la Figura X.20

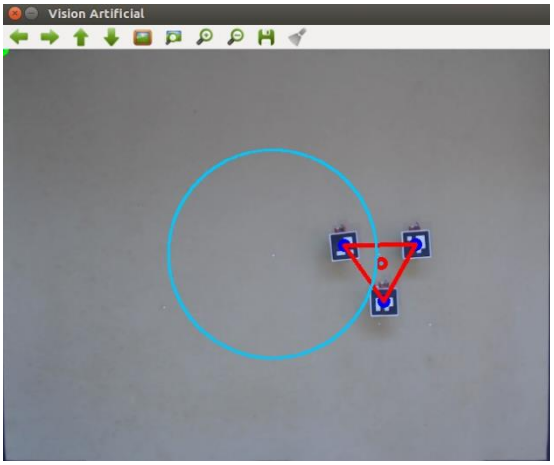


Figura X.20 Nodo *v_artificial* trayectoria circular

Al aplastar el botón “Estado de robots móviles” se despliega la ventana que se observa en la Figura X.21, donde el usuario visualiza las constantes de los controladores internos que en ese instante estén cargadas y el estado de la batería de cada robot móvil.

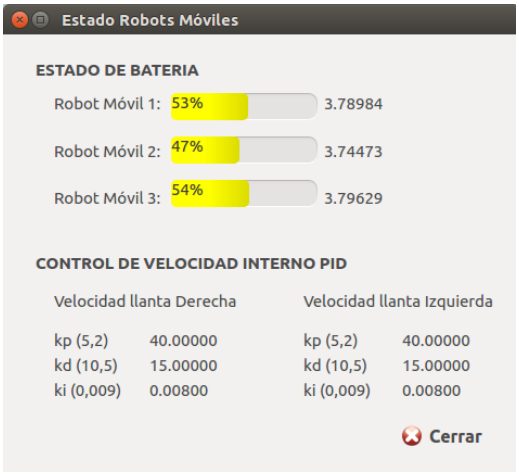


Figura X.21 Ventana de Estado de Robots móviles

X.4. EJECUCIÓN DEL CONTROLADOR

Una vez guardados los valores de los parámetros se activa el botón de Inicio, el cual abre la ventana de seguimiento de trayectoria, como se observa en la Figura X.22; donde se puede visualizar en una gráfica el movimiento de la formación de robots, la información de las velocidades de los robots móviles y la posición del centro de la formación.

Mientras se realiza el seguimiento de trayectoria, en el nodo *unidad_central* se reflejan los parámetros ingresados y los datos del monitoreo del seguimiento de la formación, Figura X.23.

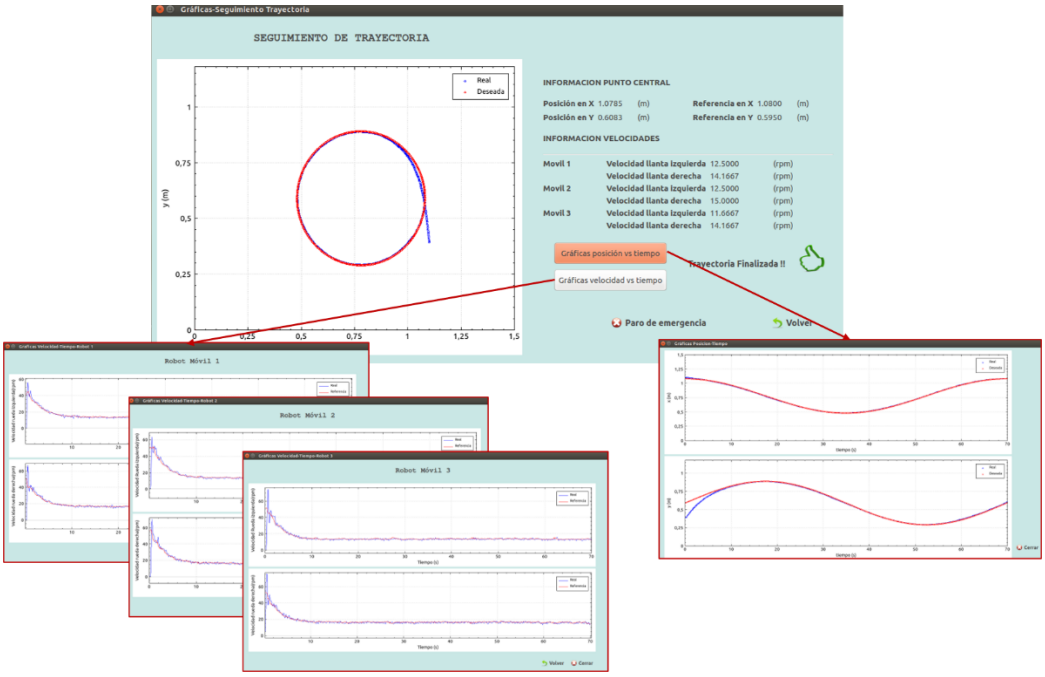


Figura X.22 Ventana de Visualización del Seguimiento de trayectoria.

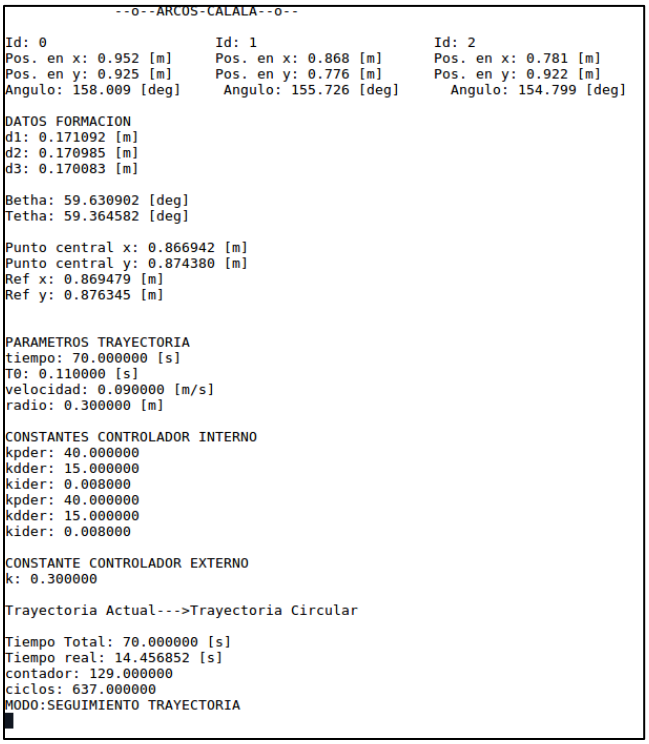


Figura X.23 Nodo *v_artificial* trayectoria circular

X.5. FINALIZACION DEL PROCESO

Al final del experimento el usuario tiene la oportunidad de analizar los resultados en las respectivas ventanas tal como se ve en la Figura X.22.

Una vez cerrados los gráficos se puede volver a la ventana principal para repetir otro experimento. Si se desea apagar el sistema se debe cerrar el HMI y todos los terminales donde se encuentran los nodos ejecutándose, finalizando así el sistema.

ORDEN DE EMPASTADO