"El saber de mis hijos hará mi grandeza"

UNIVERSIDAD DE SONORA

Facultad Interdisciplinaria de Ciencias Exactas y Naturales

Ingeniería en Tecnología Electrónica

Título de la tesis:

Implementación de Sistema de Navegación para Robots Móviles Rodantes

Para obtener el título de:

Ingeniero en Tecnología en Electrónica

Presenta:

Gilberto Ramos Valenzuela

Directores de tesis: Dr. Luis Arturo García Delgado

Dr. Ricardo Ramón Pérez Alcocer

Hermosillo, Sonora, México, junio del 2024

SINODALES

Dr. Luis Arturo García Delgado
Universidad de Sonora, Hermosillo, Son., México

Dr. Ricardo Ramón Pérez Alcocer
Universidad de Sonora, Hermosillo, Son., México

Dra. Alicia Vera Marquina
Universidad de Sonora, Hermosillo, Son., México

Dr. Alejandro García Juárez
Universidad de Sonora, Hermosillo, Son., México

Agradecimientos

A mis padres ***Gilberto Ramos Torres*** y ***Blanca Llanet Valenzuela Ruiz***, gracias por inculcarme el valor del estudio, por apoyarme incluso cuando mi desinterés hacia la escuela era total, y por las veces que me obligaron a estudiar aunque lloré y pataleé por mi falta de conciencia. Les agradeceré hasta el día que me vaya de este mundo terrenal. Ni mis aciertos ni mis fracasos son su responsabilidad, pero sus enseñanzas, ejemplos y consejos son factores que influirán en mis éxitos, y aunque en mucha menor medida, en mis fracasos. Por estos y mil motivos más, les estaré agradecido siempre.

A mis hermanos ***Pedro***, ***Ana***, ***Lucía***, ***Lucas***, ***Saúlo***, ***Nadia*** y ***Pedro***, les agradezco por acompañarme, apoyarme, pelearse, contentarse o enojarse conmigo. Crecimos juntos, no nos elegimos los unos a los otros, pero su apoyo, consejos y en menor medida su carilla son factores que me han hecho la persona que soy hoy en día. Les agradezco profundamente por todo lo que me han apoyado y convivido conmigo.

A mis profesores de la Universidad, especialmente a mis profesores de ***Ingeniería en Tecnología Electrónica***, les agradezco por enseñarme lo que no sabía, no solo de las materias de la carrera sino también por sus consejos, evaluaciones, recomendaciones, por aguantar mis dudas, mis desaciertos y los circuitos quemados.

A mis compañeros y amigos de ***ITE***, hemos peleado mil y una batallas: las desveladas, las salidas tarde, los exámenes, los proyectos de final de semestre, las ayudas de estudio. Agradezco todos los momentos buenos y malos, y llevaré esos recuerdos siempre conmigo. Adonde sea que la vida los lleve, marchen siempre hacia adelante y hacia mejor.

A la ***Universidad de Sonora***, desde el personal administrativo hasta el personal de apoyo y limpieza, les estaré agradecido por siempre. Sin ustedes, una universidad no puede funcionar.

A la sociedad mexicana, mis agradecimientos más profundos por el apoyo económico y solidario prestado, con el cual yo y muchos de mis compañeros universitarios fuimos o somos capaces de estudiar una carrera universitaria. Agradezco infinitamente el haber nacido, crecido y vivido en un país donde la educación universitaria es subsidiada por el estado y específicamente por miles de personas que con su trabajo apoyan al desarrollo económico de ***México***.

Un agradecimiento especial al ***Dr. Luis Arturo*** y al ***Dr. Ricardo Alcocer*** por aguantarme los dos años de trabajo que nos llevaron a la finalización y redacción de esta

tesis basada en ***carritos***, y por la paciencia que tuvieron conmigo. Sin ella, no hubiera aprendido ni logrado lo mucho o poco que logré.

Desde lo más profundo de mi alma, agradezco a ***Dios***, a mis ***padres***, ***hermanos***, ***tíos***, ***tías***, ***abuelos***, ***abuelas***, ***profesores*** y ***profesoras***, además de a todos aquellos que hayan influido de manera directa o indirecta en mi formación personal y académica, y en la culminación de este trabajo. Por estos y mil motivos más, aunque estos agradecimientos no estén redactados de la manera más elegante y correcta posible, reitero aquí mis más sinceros y profundos agradecimientos desde lo más profundo de mi alma.

Índice general

Índice de figuras

Capítulo 1

Introducción

En la actualidad se han popularizado los vehículos autónomos, los cuales se emplean en muchos ámbitos, algunos ejemplos son en el transporte personal [3], la logística y traslado de mercancías [4], la minería [5], la exploración espacial [6], entre otros. Para que un vehículo sea considerado autónomo, debe ser capaz de desplazarse en diferentes lugares con poca o nula intervención humana. Para lograr esto, debe tener la capacidad de decidir la ruta a tomar hacia la posición o trayectoria deseada, al mismo tiempo que evita colisionar con obstáculos [7]. Este tipo de vehículos constan de periféricos, tales como sensores de ultrasonido, cámaras de video, radares, lidares y unidades de posicionamiento global (GPS) para obtener información del ambiente en el cual estén trabajando, así como también de computadoras tales como microcontroladores y unidades de procesamiento de alto rendimiento. En conjunto estos componentes son capaces de captar información del mundo real, procesarla y tomar decisiones sobre hacia dónde orientarse y dirigirse, teniendo en cuenta factores como obstáculos, rutas óptimas o perturbaciones ambientales.

La motivación de este trabajo es que en la Universidad de Sonora se cuente con un sistema en el cual los estudiantes puedan aplicar lo aprendido en las clases relacionadas con teoría de control.

La principal contribución al campo es la integración y actualización del software con el cual se pueden realizar pruebas en robots móviles y mas específicamente en el robot móvil rodante (WMR) Amigobot.

1.1. Antecedentes

Los robots móviles pueden ser clasificados considerando distintas características, una de estas clasificaciones se hacen según su lugar de operación. En dicha clasificación existen tres tipos, a saber, vehículos terrestres o superficiales, acuáticos y aéreos.

Robots móviles superficiales

Como su nombre lo indica este tipo de robots operan en superficies como carreteras, caminos, terrenos irregulares, en interiores como almacenes y fábricas o en la superficie fuera del planeta, como en la Luna o Marte. Éstos a su vez se pueden clasificar según su forma de desplazarse.

- Robots móviles rodantes.

  Son los robots que se mueven en tierra mediante el uso de ruedas. Estos diseños se prefieren debido a que son mucho más simples que los diseños con patas, consumen menos energía y desde el punto de vista del control se requiere menos esfuerzo debido a su mecánica simple y a que el problema de estabilidad se simplifica. Un ejemplo de este tipo de robots se observa en la Figura 1.1a.

- Robots con patas

  Este tipo de robots están diseñados para moverse utilizando patas, imitando el movimiento de animales cuadrúpedos o el movimiento propio del ser humano. Su diseño y funcionalidad pueden variar considerablemente dependiendo del propósito específico y el movimiento a imitar. Un ejemplo de este tipo de robots se muestra en la Figura 1.1b.

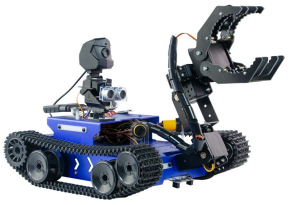
- Robots tipo oruga

  Los robot de este tipo se inspiran en el movimiento de los insectos, reptiles o vehículos militares, entre otros. La similitud con los insectos viene dada por una serie de placas o ruedas dentadas interconectadas por una banda flexible, las cuales permiten que el robot se mueva de manera eficiente sobre una variedad de terrenos, incluyendo superficies rugosas, terrenos irregulares, arena, barro, nieve, escombros o incluso otras superficies fuera del planeta. Un ejemplo de este tipo de robots se observa en la Figura 1.1c.

Robots acuáticos

(a) Robot móvil rodante

(b) Robot cuadrúpedo



(c) Robot tipo oruga

Figura 1.1: Robots móviles superficiales

Los robots acuáticos incluyen aquellos que navegan sobre el agua y los submarinos sumergibles popularizados desde la primera guerra mundial, los cuales son navíos capaces de navegar bajo la superficie del agua. Existen distintos tipos de robots submarinos según su tipo de operación.

- Vehículos Operados Remotamente (*ROVs Remote Operated Vehicles*)
  Son vehículos operados de forma remota controlados por una persona fuera del agua. Estos son equipados con cámaras, manipuladores, sensores y herramientas especializadas para llevar acabo distintas tareas en ambientes bajo el agua. Un ejemplo de este tipo de robots se presenta en la Figura 1.2a.

- Vehículos Submarinos Autónomos (AUVs *Autonomous Underwater Vehicles*)
  Son robots submarinos autónomos, los cuales están diseñados para operar de forma independiente, sin necesidad de control humano en tiempo real. Al igual que los ROVs cuentan con múltiples sensores que le permiten determinar su comportamiento dependiendo de las condiciones que se presentan en el ambiente de trabajo y la tarea que ha sido asignada. Un ejemplo de este tipo de robots se presenta en la Figura 1.2b.

- *Gliders* (Planeadores submarinos) Son robots submarinos los cuales se mueven de manera autónoma mediante cambios en su flotabilidad y alerones. Se emplean en misiones de larga duración, recopilando datos de las condiciones del agua, como puede ser la temperatura, salinidad y turbidez en extensas áreas del océano. Un ejemplo de este tipo de robots se presenta en la Figura 1.2c.

Robots aéreos

Los robots aéreos o mejor conocidos como drones, son robots aeronaves que vuelan sin tripulantes abordo. De manera similar a los robots acuáticos mencionados anteriormente

(a) Submarino remotamente operado ROV

(b) Submarino autónomo AUV

(c) Planeador submarino o *glider*

Figura 1.2: Robots acuáticos

los aéreos pueden ser controlados de forma remota por un operador, o de manera autónoma. Se emplean en una variedad de aplicaciones, que van desde el entretenimiento y la fotografía aérea hasta fines militares, industriales, agrícolas, de búsqueda y rescate, y de vigilancia, entre otras. Según su tipo de hélices o rotores, los vehículos aéreos pueden ser clasificados en distintas categorías como las que se mencionan a continuación:

- Drones de ala fija
  Este tipo de drones tienen alas rígidas en el lugar de hélices, planean y pueden hacer uso de energía solar para lograr tiempos de vuelo y distancias mucho mayores en comparación con drones multirotor [8]. Un ejemplo de este tipo de robots se presenta en la Figura 1.3a.

- Drones multirotores
  Los drones multirotor tienen más de una hélice. Los tipos más comunes de este tipo de aeronaves tienen cuatro rotores, conocidos como quadirotores. El máximo número de motores que suelen tener los drones es de seis u ocho; cuanto mayor sea el número de rotores que tenga un dron, mayor será la carga útil que podrá levantar. Todos los drones con seis u ocho rotores se diseñan para tener aditamentos como cámaras y tanques con pesticidas según sea la actividad que se desea realicen. Un ejemplo de este tipo de robots se presenta en la Figura 1.3b.

- Drones híbridos de despegue y aterrizaje vertical (VTOL *Vertical Take-Off and Landing*)
  Los tipos de drones híbridos VTOL combinan los beneficios de los diseños de ala fija y basados en rotor. Este tipo de dron tiene rotores unidos a las alas fijas, lo que le permite mantenerse en el aire, despegar y aterrizar verticalmente. Esta nueva categoría de vehículos híbridos es aún escasa en el mercado, pero a medida

que avanza la tecnología, esta opción podría volverse mucho más popular en los próximos años. Un ejemplo de VTOL híbrido de ala fija es el dron de entrega Prime Air de Amazon [9]. Un ejemplo de este tipo de robots se presenta en la Figura 1.3c.

(a) Dron de ala fija (b) Dron multirotor (c) Dron híbrido

Figura 1.3: Robots aéreos

Al desarrollar y trabajar con vehículos autónomos surgen problemas de estudio, fuera del diseño propio del dispositivo (hardware). Estos problemas son abordados tanto en el ámbito académico como en el industrial, buscando dar soluciones que mejoren el desempeño de los vehículos. Algunos problemas de estudio incluyen los siguientes:

- Planificación de rutas.

  La planificación de rutas implica encontrar una ruta geométrica desde una configuración inicial hasta una configuración deseada de manera que cada estado en la ruta sea factible (si se tiene en cuenta el tiempo entonces se habla de trayectoria) [10].

- Detección y evasión de obstáculos.

  Es imprescindible que un vehículo autónomo sea capaz de detectar entidades mediante el uso de sensores, ya sea por medio de dispositivos ultrasónicos, ópticos o radares, para esquivar y evitar colisiones con las mismas. Esta capacidad de evasión y detección de obstáculos es fundamental para garantizar la seguridad tanto del vehículo como de los objetos que puedan encontrarse en su entorno. El estudio de este problema es fundamental en el desarrollo y la implementación de vehículos autónomos.

- Localización y mapeo simultaneo SLAM (*Simultaneous Localization and Mapping*).

  SLAM aborda el problema de percepción de un robot al navegar en un entorno desconocido. Al navegar en el entorno, el robot busca adquirir un mapa del ambiente

y, al mismo tiempo, estimar su localización utilizando dicho mapa. El objetivo de los sistemas SLAM es generar modelos detallados del entorno del robot al tiempo que mantiene un sentido preciso de la ubicación en el mismo [11].

- Formación y colaboración entre múltiples agentes.

  Un sistema de robot multiagente (MARS, por sus siglas en inglés) es uno de los temas más importantes en la actualidad. La tarea básica de este sistema se basa en el trabajo distribuido y cooperativo entre agentes (robots). Combina dos sistemas importantes; sistema multiagente (MAS, por sus siglas en inglés) y sistema de múltiples robots (MRS, por sus siglas en inglés). MARS se ha utilizado en muchas aplicaciones como sistemas de navegación, planificación de rutas, sistemas de detección, protocolos de negociación y control cooperativo [12].

- Navegación en ambientes desconocidos dinámicos.

  Generalmente fuera de los ambientes controlados de laboratorio o industriales, los ambientes en los que operan los robots móviles autónomos son desconocidos y dinámicos. Ejemplo de esta característica la encontramos en los taxis autónomos, los cuales operan en vialidades concurridas y cambiantes. Por lo tanto, el vehículo debe ser capaz de planificar rutas o trayectorias con información incompleta, evitando problemas que se presenten por la interacción con el entorno.

Como se puede notar en los problemas de estudio mencionados anteriormente la percepción del ambiente tiene un rol principal en su desarrollo. La visión por computadora es un campo de estudio el cual desarrolla técnicas con el objetivo de ayudar a los dispositivos mecánicos a obtener información del ambiente de manera análoga a como lo hace la visión humana. Esta característica la ha convertido en un componente altamete usado en el diseño de robots móviles.

La visión computacional consiste en la extracción automatizada de imágenes y el procesamiento de las mismas para la obtención de información. Por información podemos entender cualquier cosa en este contexto, como lo son: formas, colores, posición de las cámaras u objetos, reconocimiento de objetos, agrupación y búsqueda de contenido, también las deformaciones de imágenes, eliminación de ruidos y la realidad aumentada [13].

Esta área de investigación tiene más de cuarenta años de investigación, con lo cual se cuenta con diversas aplicaciones y técnicas desarrolladas.

- Reconocimiento óptico de caracteres (OCR)

La tecnologías OCR engloban un conjunto de técnicas basadas en estadística, la forma de los caracteres, transformadas y comparación, las cuales se complementan entre sí, se emplean para distinguir formas automáticamente entre los diferentes caracteres alfanuméricos existentes [14].

- Inspección robotizada

  Se emplean técnicas de reconocimiento de forma, procesamiento de imágenes e inteligencia artificial para automatizar y agilizar la inspección visual de piezas de fabricación industrial, con la finalidad de garantizar la calidad de los productos [15].

- Captura de movimiento

  El proceso de captura de movimiento (MoCap por sus siglas en inglés) es aquel que tiene como objetivo rastrear y registrar digitalmente el movimiento de un objeto o sujeto en el espacio. Se han desarrollado diferentes técnicas y tecnologías para llevar a cabo esta tarea. Uno de los ejemplos más comunes son los sistemas con cámaras infrarrojas (IR), las cuales se utilizan para estimar la ubicación de cuerpos rígidos con superficie reflectante adheridos al objeto de estudio [16].

1.2. Planteamiento del problema

En este trabajo se presenta el proceso de puesta en operación de un robot móvil rodante en un ambiente de laboratorio, empleando un sistema de visión. El sistema de visión se utiliza para estimar la pose del vehículo en tiempo real, asegurando que esta información esté disponible para definir o controlar el comportamiento del mismo.

En el campo de la robótica, se conoce como pose o postura de un objeto o robot a su configuración, definida por su posición y orientación. Con el objetivo de validar el funcionamiento del sistema, se propone implementar un esquema de navegación que utiliza técnicas de generación de rutas en ambientes estáticos con evasión de obstáculos.

1.3. Objetivos

1.3.1. Objetivo general

El objetivo general de este trabajo es implementar un sistema de navegación para robots móviles rodantes empleando un sistema de visión por computadora.

1.3.2. Objetivos particulares

Dentro de los objetivos particulares que permitan alcanzar el objetivo general, en este trabajo se han establecido los siguientes:

- Poner en operación el robot móvil rodante Amigobot.
- Obtener la pose del robot móvil rodante mediante el sistema de visión Optitrack.
- Integrar el vehículo rodante con el sistema de visión en un ambiente de programación que permita implementar esquemas de control y navegación.
- Implementar un sistema de navegación basado en la teoría de campos potenciales para el robot móvil.
- Validar el desempeño del sistema de navegación implementado.

1.4. Organización de la tesis

El resto del documento de tesis está organizado de la siguiente manera. En el Capítulo 2 se describe el sistema de visión empleado en el desarrollo de este trabajo. Los detalles de su instalación y su funcionamiento se presentan en detalle.

Las características del software y hardware necesarias para establecer la comunicación con el robot móvil rodante Amigobot y para poner el mismo en operación se presentan en el Capítulo 3. Además se expone el modelo cinemático el cual es necesario para abordar problemas de navegación y control.

El Capítulo 4 expone la estrategia de navegación y control empleada para validar el funcionamiento del sistema estudiado.

El esquema experimental empleado para la validación del sistema y los resultados obtenidos del proceso de prueba se discuten en el Capítulo 5. Finalmente, el Capítulo 6 presenta las conclusiones derivadas de este trabajo.

Capítulo 2

Sistema de visión

2.1. Introducción

El objetivo de un sistema de visión o *Machine Vision System*(MVS) es extraer información relevante del mundo real mediante imágenes. Un sistema de visión recoge información útil de una escena a través de su proyección en dos dimensiones. Debido a que las imágenes son proyecciones bidimensionales de un entorno tridimensional, es necesario recuperar la información dentro de las mismas, dado que no está disponible directamente. Este tipo de sistemas suelen utilizar cámaras u otros dispositivos de captura de imágenes, con el fin de obtener datos visuales, procesarlos mediante algoritmos y técnicas de procesamiento de imágenes, para así analizar, extraer información y tomar decisiones basadas en dichos datos.

Los sistemas de visión se utilizan en una gran variedad de aplicaciones, algunos de los usos más importantes dentro de esta área son:

- Seguimiento: Rastreo del movimiento de objetos a lo largo del tiempo en secuencias de imágenes.
- Medición: Cálculo de dimensiones, distancias, áreas y otros parámetros de interés en una imagen.
- Interpretación de escenas: Comprender y analizar la disposición espacial de los objetos en una imagen para extraer información más completa.

- Navegación autónoma: Permitir a vehículos autónomos, como drones o vehículos terrestres, interpretar su entorno y tomar decisiones de navegación.

En este trabajo se emplea un sistemas de visión comercial con la finalidad de estimar la posición de un vehículo terrestre y abordar la temática de la navegación autónoma.

2.2. Sistema de visión *Optitrack*

En este trabajo se emplea el sistema de visión *Optitrack*, esta es la marca más conocida de *NaturalPoint Inc*, empresa especializada en el desarrollo y fabricación de sistemas de *Motion Capture* o captura de movimiento. La empresa fue fundada en 1996 y tiene sede Corvallis, Oregon, Estados Unidos de América.

2.2.1. Componentes del sistema *Optitrack*

El sistema de visión *Optitrack* se constituye de distintos componentes, tanto de hardware como de software, los cuales se presentan a continuación.

Cámaras Flex 13

El sistema emplea seis cámaras *Flex 13*, de la marca *Optitrack*, la Figura 2.1 muestra una de las cámaras empleadas. Las *Flex 13*, son cámaras de captura de movimiento de volumen medio. Funcionan mediante luz infrarroja de 850nm, la cual es inperceptible al ojo humano. La Tabla 2.1 presenta las características más importantes de estas cámaras. Estas cámaras emiten luces infrarrojas por medio de un círculo de elementos ópticos, esta luz se propaga por medio del aire y rebota en marcadores reflejantes. Una vez que la luz rebota en los marcadores ésta es captada por los sensores de las cámaras. Otra característica de este tipo de cámaras es que permiten visualizar las imágenes captadas en escala de grises, esto con la finalidad de orientar su campo visual.

Para la colocación de cada una de las cámaras se emplea un tripié el cual permite ajustar su altura y orientación de manera sencilla. La Figura 2.2 muestra una de las cámaras colocada en el trípode.

Figura 2.1: Cámara *Optitrack Flex 13*.

Tabla 2.1: Características de las cámaras *Flex 13*

Característica	Valor
Resolución	1.3 MP
Latencia	8.3 ms
Precisión en 3D	±0.20 mm
Frecuencia de muestreo	120 FPS
Campo de visión	56°

Concentrador *Optihub 2*

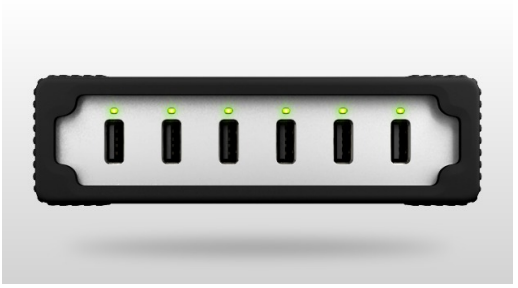
Las cámaras se conectan mediante un hub o concentrador *Optihub 2*, el cual sincroniza y alimenta cada una de las seis cámaras con las que se cuenta. Cada *Optihub 2* permite conectar hasta seis cámaras. La Figura 2.3a muestra la vista superior del hub con las etiquetas de cada uno de sus conectores.

En la Figura 2.3b se observa los seis conectores USB en los cuales se conectan las cámaras. Debido a que se tienen seis cámaras y dos hubs, se divide en grupos de tres cámaras para así conectar cada grupo a un hub. La Figura 2.3c muestra los puertos disponibles en el *Optihub 2* para llevar a cabo la sincronización entre los 2 hubs utilizados. Los números que aparecen en la figura hacen referencia a las siguientes conexiones.

1. Conexión de voltaje: Cada hub es alimentado por una fuente de voltaje de 24VCD.
2. Conexión de datos USB a la computadora: Este cable lleva la señal de salida de cada hub a la computadora.
3. Puerto de sincronización: Este puerto permite sincronizar la señal entre dos hubs, para que las cámaras funcionen de manera coordinada.

Figura 2.2: Cámara *Flex 13* colocada en su tripié.

(a)



(b)

(c)

Figura 2.3: Concentrador *Optihub 2*. (a) Vista superior; (b) vista frontal; (c) vista trasera.

Una vez que se tienen colocadas las cámaras y conectadas a cada uno de los hubs, es necesario conectar a la computadora el cable USB de uno de los *Optihub 2*. También es necesario emplear el software *Motive* para realizar la configuración del sistema de visión.

Software *Motive Optitrack*

La misma empresa desarrolladora de *Optitrack* proporciona un software para realizar tanto la iniciación del sistema de cámaras, como la captura de movimiento, este programa es *Motive*. La Figura 2.4 muestra el logo del software.

Figura 2.4: Logo *Motive Optitrack*.

La versión utilizada de este software es la 1.10.1, con fecha de lanzamiento diciembre del 2016. Esta se encuentra disponible para su descarga en el siguiente enlace: `https://s3.amazonaws.com/naturalpoint/software/Motive/Motive_1.10.1_Final_x64.exe`

Este programa, requiere tanto de una clave serial, como de una llave física USB para su activación y uso. La Figura 2.5 muestra la interfaz de *Motive* con la cual se cuenta al iniciar el programa.

2.2.2. Configuración y calibración del sistema de visión

Una vez que se han colocado las cámaras y se ha conectado todo el sistema, es indispensable realizar el proceso de calibración. El proceso de calibración del sistema de visión es necesario para la correcta obtención de información 3D a partir de imágenes 2D de la escena. Existen diferentes técnicas basadas en fotogrametría y auto-calibración. Como resultado se obtienen los parámetros intrínsecos y extrínsecos de la cámara [17]. Así, al calibrar es posible que el software estime la ubicación de las cámaras, además de definir el origen del sistema de referencia en ese momento. Con el origen establecido, *Motive* será capaz de estimar la posición relativa de los objetos distinguidos mediante los marcadores ópticos.

La interfaz de usuario del programa se muestra en la Figura 2.5.

Para realizar la calibración, se debe crear un proyecto nuevo, es importante tener presentes tanto el nombre del proyecto como la ruta.

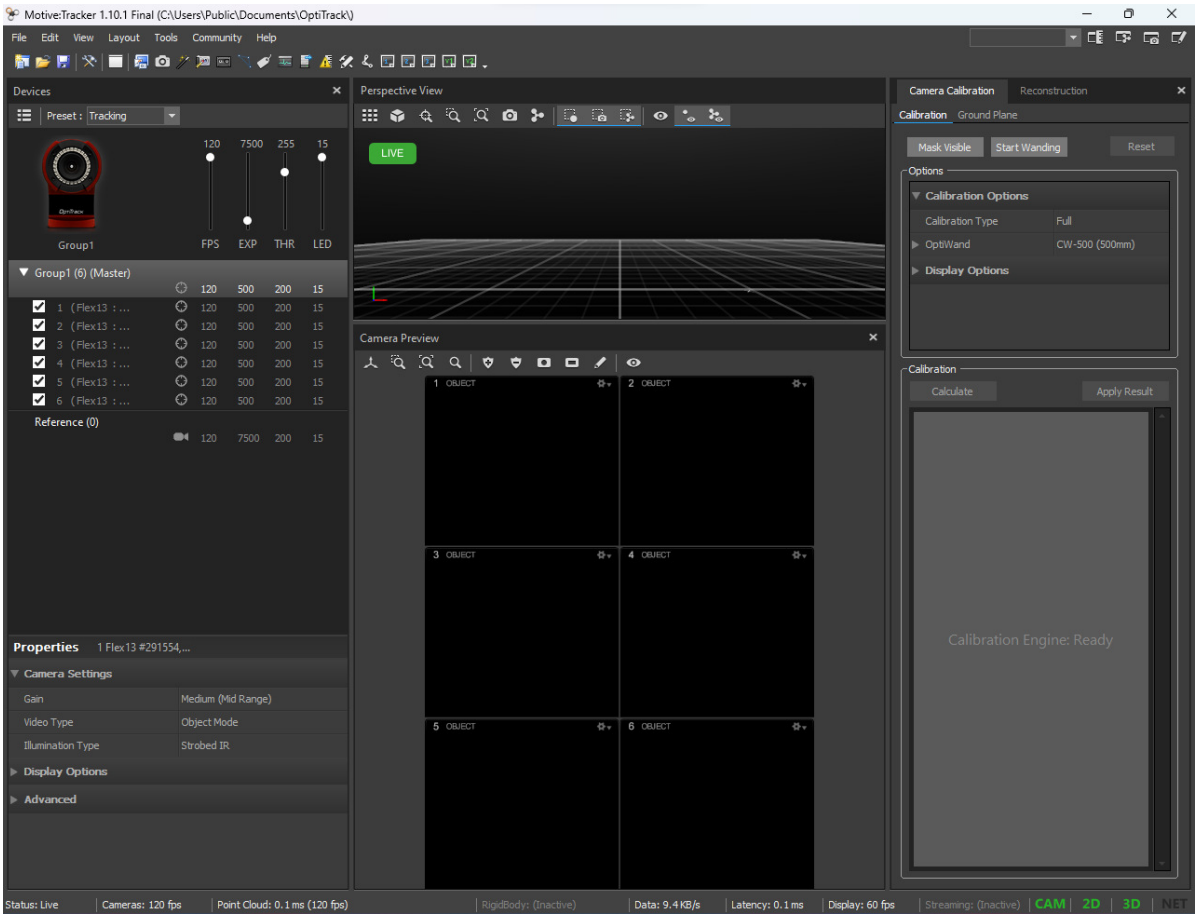


Figura 2.5: Interfaz de *Motive* Tracker.

Pasos para calibración

Para iniciar la calibración, es necesario retirar todos los marcadores y objetos que reflejen la luz infrarroja del área de trabajo, para evitar errores al momento de la calibración, como la detección de falsos objetos que, al reflejar la luz infrarroja, se vean en el programa.

1. Orientar las cámaras.

   Es fundamental para el buen funcionamiento del sistema que el campo visual de las cámaras cubra todo el espacio de trabajo. Para ello en *Motive* hay que seleccionar cada cámara en la ventana del lado izquierdo, como se ve en la Figura 2.6. En la sección *Preset* se tiene que seleccionar la opción *Aiming*. Una vez hecho lo anterior, se debe disminuir al mínimo el número de cuadros por segundos empleando la barra deslizante etiquetada con FPS (frames per second), y aumentar al máximo el valor de exposición con su correspondiente marcada con EXP.

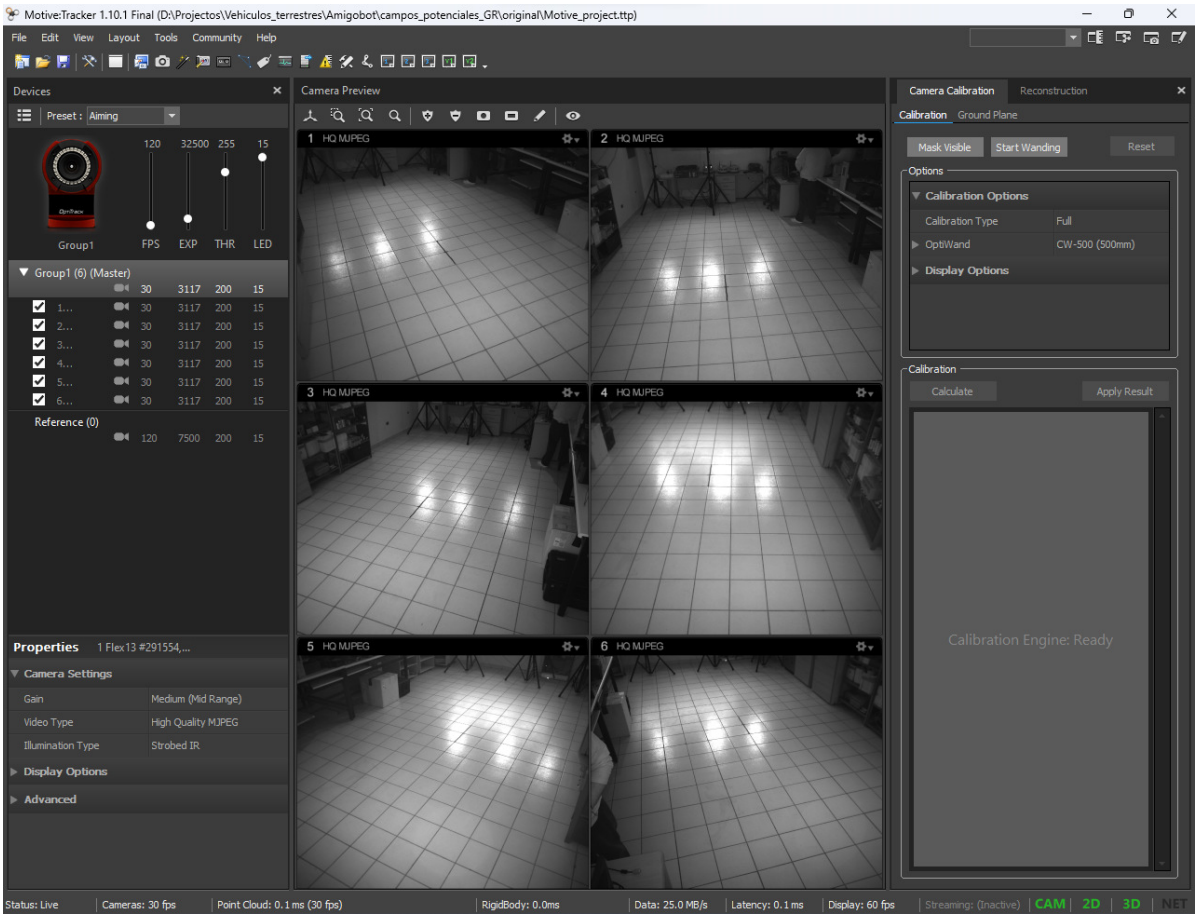


Figura 2.6: Visualización de área de trabajo Motive.

La Figura 2.6 muestra la visualización del área de trabajo al momento de realizar el acomodo de las cámaras, una vez que se ha configurado el software con las opciones ya mencionadas. Esta visualización permite orientar adecuadamente las cámaras.

2. Calibración

   La calibración de las cámaras se lleva a cabo empleando una vara de calibración, conocida como *Optiwand*, incluida en el paquete *Optitrack*. La vara de calibración de la Figura 2.7 es una estructura metálica en forma de T a la que se le colocan tres marcadores con una distribución conocida.

   Para iniciar la calibración es necesario presionar el botón *Start Wanding*, el cual se encuentra en la pestaña de *Camera Calibration* ubicada en la sección del lado derecho de la Figura 2.5. Es muy importante que no haya marcadores dentro del rango de visión de las cámaras. Así mismo, se debe subir el número de cuadros por segundo al máximo y bajar el valor de exposición al mínimo, es decir lo opuesto a lo mencionado en el punto anterior.

Figura 2.7: Bara de calibración Optiwand CW-500.

Una vez iniciada la calibración debemos mover la vara ensamblada, dentro del rango de visión de las cámaras. Al momento de mover la vara a través del rango de visión, se irán generando líneas en cada una de las vistas previas de las cámaras, la Figura 2.8a muestra las líneas generadas por *Motive* al detectar la herramienta de calibración.

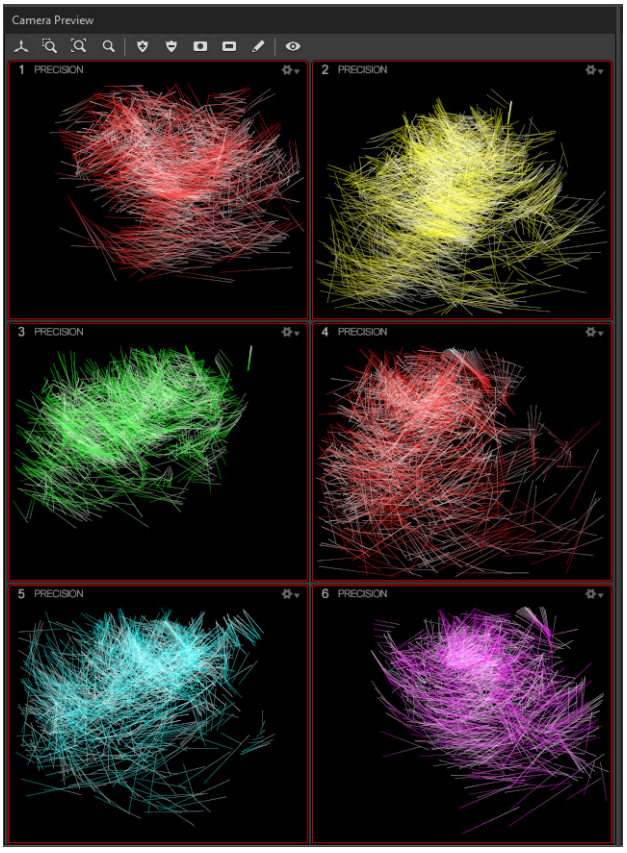
Se debe generar un número suficiente de líneas, como las que se muestran en la Figura 2.8a. Cada cámara tomará muestras de la posición de los marcadores, y mientras más se tomen, menor será el error de calibración. Un buen rango de muestras para cada cámara es de 4000, como se ilustra en la Figura 2.8b, donde se observa el numero de datos obtenidos a través del proceso de muestreo.

Una vez que se tiene una cantidad de muestras suficiente, , presionaremos el botón *Calculate*. De esta manera, el programa calcula los parámetros intrínsecos y extrínsecos de las cámaras, además de establecer el marco de referencia y calcular el error de estimación. En la Figura 2.9 se ilustran los errores obtenidos. La calibración tomará efecto al presionar el botón *Apply*.

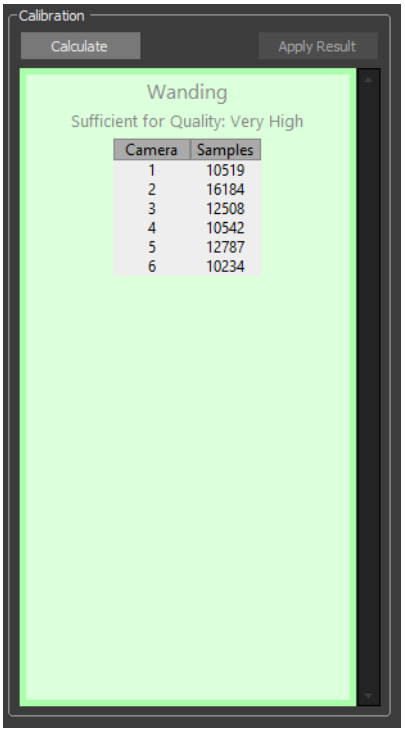
3. Establecer el origen del sistema coordenado

   Una vez finalizada la calibración es necesario seleccionar el origen del marco de referencia, dentro del área de trabajo. Para ello se emplea la escuadra de calibración que se muestra en la Figura 2.10, la cual cuenta con marcadores reflejantes.

   El proceso continúa colocando la escuadra con marcadores en la posición donde se desea establecer el origen del sistema de coordenadas en el área de trabajo. Para finalizar, se selecciona el botón *Set Ground Plate*, el cual está ubicado en la parte superior de una ventana del lado derecho dentro de la interfaz de *Motive*.



(a) Vista de cámaras durante la calibración.



(b) Muestras tomadas durante calibración.

Figura 2.8: Proceso de calibración de las cámaras

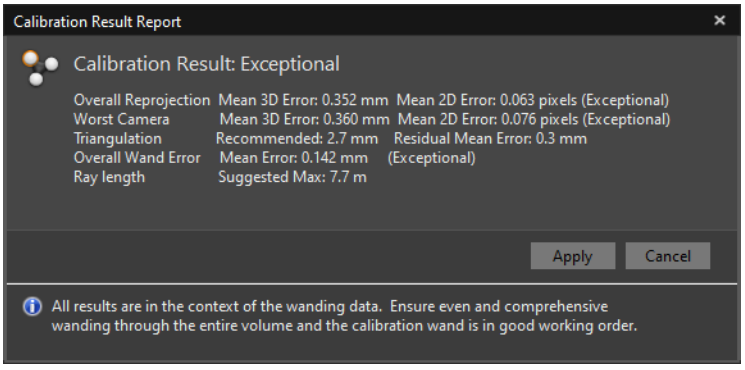


Figura 2.9: Resultado de la calibración. Mediciones del error.

2.2.3. Creación del objeto rígido

Una de las funciones del software *Motive* es que permite identificar y estimar las poses, es decir tanto la posición como la orientación, de objetos de interés usando los marcadores que reflejan la luz infrarroja. Para lograr esto es necesario colocar los marcadores sobre el

cuerpo del objeto a seguir. Una vez que se han agregado al objeto de interés es necesario posicionarlo en el área de trabajo.

Posteriormente, se procede a seleccionar cada uno de los marcadores del objeto, para ello dentro de la ventana *Perspective View* en *Motive*, se presiona el botón izquierdo del ratón sobre cada marcador azul mientras se presiona la tecla *Ctrl*. El siguiente paso consiste en presionar el botón derecho y se selecciona la opción *Create Rigidbody*. La Figura 2.11 muestra un ejemplo de los marcadores agrupados de un objeto creado siguiendo el procedimiento descrito anteriormente.

Una vez que se ha definido el objeto rígido, es necesario considerar el nombre del objeto, que se establece por defecto como Rigidbody1. Este identificador es de gran importancia para configurar la adquisición de datos en las aplicaciones del sistema de visión.

Como se mencionó anteriormente, el software del sistema de visión estima la posición y orientación del objeto definido. En la Figura 2.12 se muestran los valores estimados para un objeto en el ambiente de trabajo. Estos valores son accesibles a través del menú *View/Project*, el cual abre la ventana *Assets*. En esta ventana se selecciona el objeto del cual se desea visualizar su posición y orientación. Finalmente, se selecciona *Real-Time Info*, lo cual muestra los valores en tiempo real.

Figura 2.10: Escuadra de calibración usada para establecer el origen del sistema coordenado.

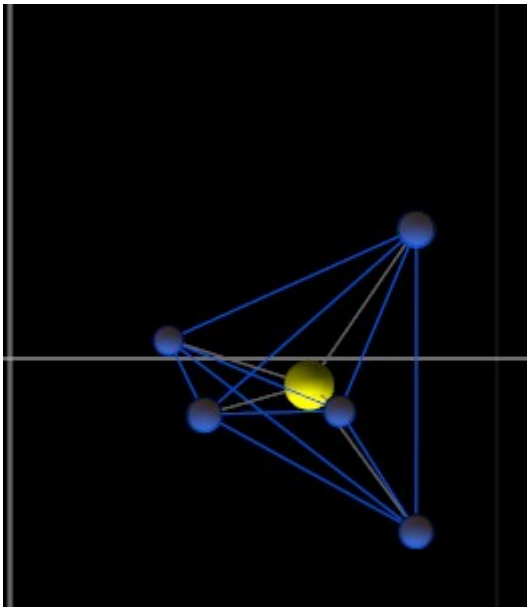


Figura 2.11: Objeto rígido creado con cinco marcadores.

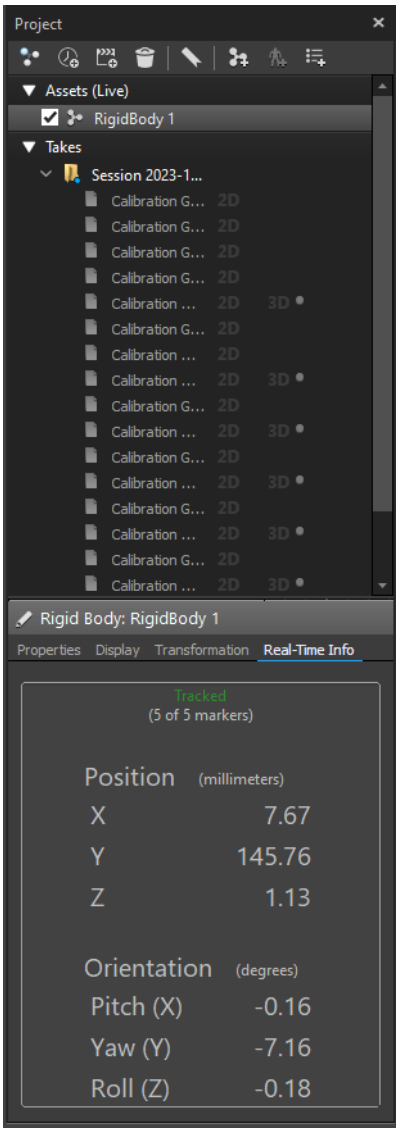


Figura 2.12: Estimación de pose del objeto creado

Capítulo 3

Robot móvil rodante Amigobot

3.1. Introducción

En la actualidad existe una gran cantidad de robots móviles empleados en diversas aplicaciones. Son usados tanto en ambientes domésticos como en ambientes industriales, y se han vuelto indispensables en campos como la exploración marítima, la fotografía aérea, la entrega de mercancía, el manejo de mercancías en almacenes o la nueva moda de taxis autónomos.

Existen diferentes clases de vehículos autónomos, los podemos clasificar en vehículos terrestres, aéreos y marítimos según su ambiente de trabajo. El sistema de navegación propuesto en esta tesis se aplicó en vehículos terrestres, sin embargo, se puede aplicar en diversos tipos de robots móviles.

El principal enfoque del sistema presentado en este trabajo es un vehículo móvil rodante o WMR (*Wheeled Mobile Robot* por sus siglas en inglés). El vehículo estudiado y puesto en operación es el robot Amigobot de la marca Adept MobileRobots [1] .

El robot móvil rodante Amigobot, es un vehículo terrestre tipo uni-ciclo de la familia *Pioneer*. Es un robot pequeño desarrollado con fines de investigación y desarrollo. La Figura 3.1 muestra el WMR Amigobot.

Este robot cuenta con dos ruedas de dirección y una tercera rueda de estabilidad. El cuerpo del Amigobot está hecho de policarbonato, el cual es soportado por un chasis de aluminio. La Figura 3.2 muestra las dimensiones y materiales en los cuales está construido el robot.

Figura 3.1: Robot móvil rodante Amigobot

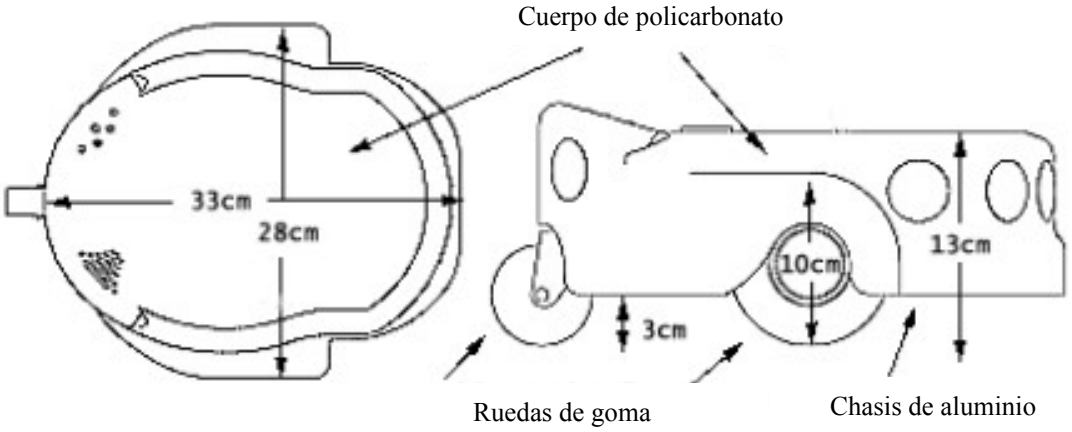


Figura 3.2: Dimensiones y materiales del robot Amigobot [1].

3.2. Características y Hardware

La plataforma Amigobot basa su sistema en un microprocesador Hitachi H8 a 20 MHz. Este sistema incluye memoria *FLASH* de sólo lectura, donde se guarda el sistema operativo y todo el software necesario para la operación de los sensores.

Sistema Operativo: AmigOS

El sistema operativo AmigOS gestiona todos los sistemas y componentes de bajo nivel del WMR, incluyendo la función de sonares, motores y la lectura de los codificadores

rotativos de las ruedas, por mencionar algunas tareas del SO. Al igual que muchos sistemas operativos, AmigOS es invisible para la mayoría de los usuarios, realizando su trabajo en segundo plano. Sin embargo, es una tecnología amigable al usuario, su interfaz está disponible libremente para el comando y control a través de su propio software.

El software del robot incluye una serie de programas con los cuales el usuario puede configurar, probar y operar de manera autónoma el vehículo. Estos programas hacen funcionar los motores y sonares incorporados, además de permitir al robot moverse de manera inteligente por sí mismo.

Sensores y actuadores

Al ser un robot para investigación y desarrollo, Amigobot presenta una variedad de sensores y actuadores los cuales serán descritos a continuación. La Figura 3.3 muestra los componentes y controles en el cuerpo del robot.

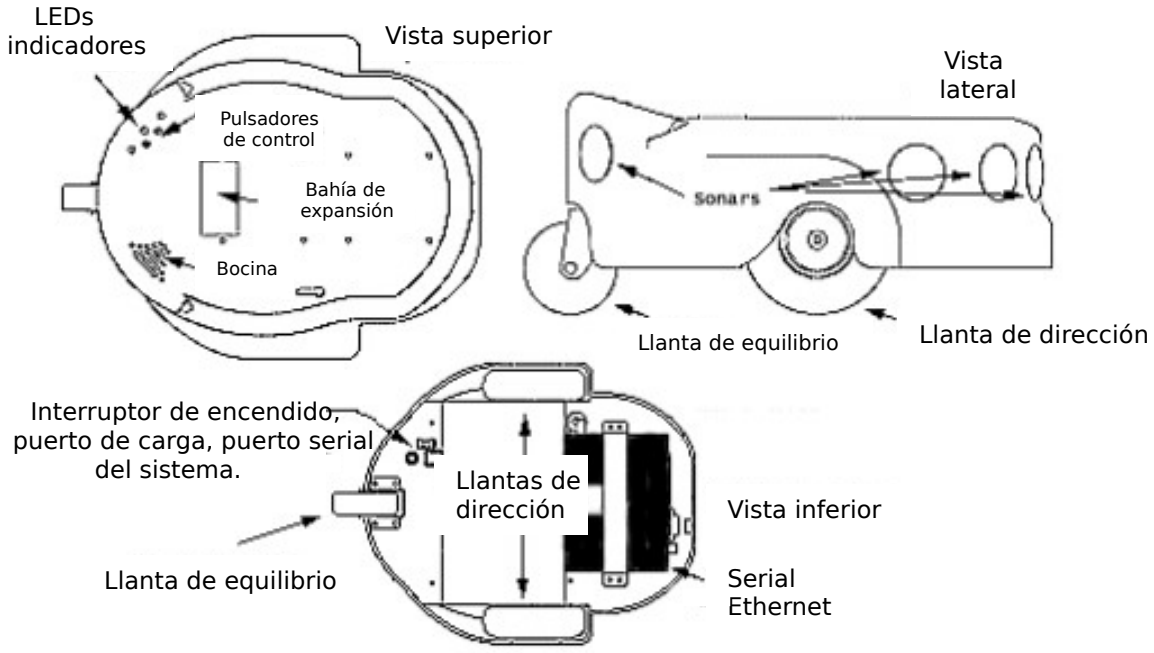


Figura 3.3: Amigobot componentes y controles [1].

El robot tiene dos llantas sólidas de cuatro pulgadas. Cada una es controlada por un motor reversible de corriente directa. La potencia de conducción se modula diferencialmente en ancho de pulso para un control preciso e independiente del movimiento de translación (adelante y atrás) y el movimiento de rotación (en sentido horario y en sentido anti-horario).

El robot cuenta con una tercera rueda la cual sirve de soporte trasero pasivo para el equilibrio y, a diferencia de muchos otros robots, puede girar en su propio eje para no quedarse atascado en las esquinas.

Cada motor incluye un codificador de alta resolución, el cuál es utilizado por el microcontrolador del robot para determinar las velocidades inmediatas de traslación y rotación del mismo. Las lecturas del codificador también ayudan a determinar la distancia que el robot ha recorrido y la dirección en la que se dirige.

El AmigoBot estándar viene equipado con ocho sonares, éstos se ubican en los seis discos dorados perforados alrededor de la parte delantera y los dos en la parte trasera. La Figura 3.4 muestra la ubicación y ángulos de cada uno de los sonares. El microcontrolador de AmigoBot utiliza los sonares de manera similar a un murciélago, no sólo para detectar objetos en la parte delantera, los lados y la parte trasera, sino también para determinar a qué distancia se encuentran ("estimación de distancia").

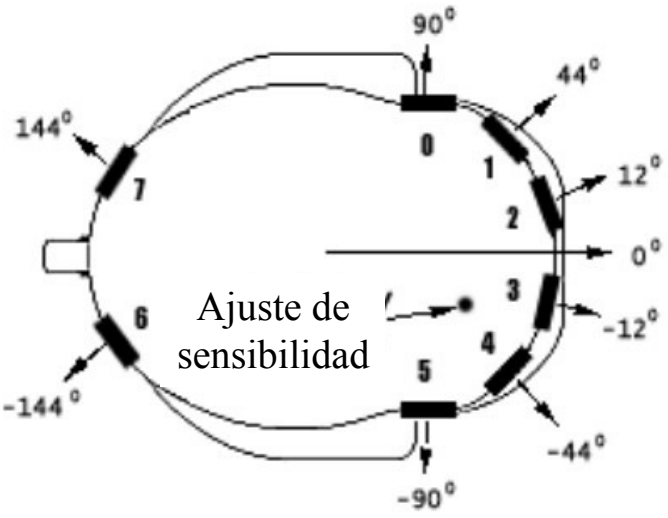


Figura 3.4: Ubicación y orientación de los sonares en Amigobot[1].

3.3. Comunicación y conectividad

El robot Amigobot no tiene comunicación inalámbrica, cuenta únicamente con comunicación serial mediante un puerto RS232. Este puerto se utiliza para configurar parámetros internos, como las velocidades máximas y las constantes de su controlador PID (Proporcional Integral Derivativo) interno, así como para recibir las velocidades necesarias para realizar la acción requerida.

Para solventar esta limitación, el robot cuenta con un módulo Lantronix *WiBox2100E*, el cual es un módulo serial a *WiFi*. Este módulo permite acceder, controlar, monitorear o compartir virtualmente cualquier dispositivo o equipo serial en una red inalámbrica 802.11b/g de forma remota.

Esta solución está bien integrada, ya que combina un sistema operativo, un servidor web integrado, una pila completa de protocolo TCP/IP con un transceptor 802.11b/g que admite seguridad WEP, WPA y 802.11i/WPA2-Personal, y dos puertos serie de alta velocidad en un paquete compacto y pequeño [18].

3.4. Operación y manejo de Amigobot

En esta sección se describe el método utilizado para la puesta en marcha y operación del robot Amigobot.

3.4.1. ROS/ARIA

El robot funciona mediante la interfaz de programación de aplicaciones (API por sus siglas en inglés) *ARIA* (Adept MobileRobots Advanced Robotics Interface for Applications). ARIA es un interfaz de programación orientada a objetos para los robots de la línea *Intelligent mobile robots*, de la marca Adept MobileRobots (y ActivMedia), los cuales incluyen los robots móviles Pioneer 2/3 DX y AT, PeopleBot, PowerBot, AmigoBot, PatrolBot/Guiabot, Seekur, SeekurJr y Pioneer LX.

Desarrollado en *C++*, ARIA proporciona una forma de comunicarse y administrar el microcontrolador dentro del robot Amigobot, además de brindar acceso a los sensores y actuadores de la plataforma. ARIA se proporciona como software de código abierto bajo la Licencia Pública General de GNU. Esto permite ver, modificar y reconstruir la biblioteca ARIA según se desee, siempre y cuando el software desarrollado con ARIA cumpla con los requisitos de la licencia.

Por otra parte, el Sistema Operativo de Robots (ROS, por sus siglas en inglés) es un conjunto de bibliotecas de software y herramientas de desarrollo para robots. Estas bibliotecas incluyen desde controladores hasta algoritmos de última generación para desarrolladores, siendo estas herramientas de código abierto.

El método para enviar consignas de control o parámetros de configuración al WMR Amigobot fue la implementación ROS/ARIA. Este método requiere únicamente una computadora con un SO Linux, el cual sea compatible con la versión 1 de ROS y el paquete de instalación de ARIA. En el caso de este trabajo se estableció la comunicación empleando distribuciones de Debian y Ubuntu, tanto en versiones de escritorio como en *Raspberry Pi*.

Existen distintas maneras en las cuales se pueden instalar las funciones de ROS/ARIA. En esta sección se describen dos métodos para su instalación, el primero en una computadora monoplaca *Raspberry Pi 4* y el segundo en una computadora personal. En ambos casos el primer paso es instalar el sistema ROS 1 en la versión seleccionada de Linux.

Para realizar la instalación del sistema operativo, consulte el sitio web de Debian 10 en `https://www.debian.org/releases/buster/` y para Ubuntu en *Raspberry Pi*, visite `https://learn.ubiquityrobotics.com/noetic_pi_image_downloads`.

La forma más sencilla de instalación de ROS Noetic (la última versión de ROS 1) en *Raspberry Pi*, es descargando la imagen de Ubuntu 20.04 LTS + GDM3 desde la página de Ubiquity Robotics, véase `https://learn.ubiquityrobotics.com/noetic_pi_image_downloads`, y grabar la imagen en una tarjeta microSD para ser utilizada como sistema de arranque por la *Raspberry Pi*. Este proceso se puede hacer mediante el software Etcher, véase `https://etcher.balena.io/`, también es posible realizarlo usando el software Raspberry Pi Imager, véase `https://www.raspberrypi.com/software/`.

Por otra parte la instalación de ROS Noetic en versiones de escritorio de Linux requiere una secuencia de pasos más extensa, a continuación se describe la misma, la cual fue probada tanto en Debian 10 como en Ubuntu 20.04. El proceso de instalación consiste en ejecutar las instrucciones listadas a continuación en la terminal del sistema.

1. Se agrega el repositorio de software ROS a la lista de fuentes *apt* en Linux usando el comando *sudo* para ejecutar la instrucción como administrador, por lo cual es necesario autentificar la cuenta de administrador.

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) "
\ > /etc/apt/sources.list.d/ros-latest.list'
```

2. Se agregan las firmas digitales para la identificación de los paquetes.

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 \
```

```
--recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

3. Se actualiza el sistema.

```
$ sudo apt update
```

4. Se instala ROS.

```
$ sudo apt install ros-noetic-desktop-full
```

5. Se incluye el ambiente de ejecución.

```
$ source /opt/ros/noetic/setup.bash
```

6. Se instalan los paquetes de Python necesarios para ROS.

```
$ sudo apt install python3-rosdep python3-rosinstall python3-rosinstall- \
  generator python3-wstool build-essential
$ sudo apt install python3 - rosdep
```

Finalizada la instalación de ROS se valida su correcto funcionamiento mediante las siguientes instrucciones ejecutadas en la terminal.

- Se inicializa el sistema de gestión de dependencias *rosdep*

```
$ sudo rosdep init
```

- Se actualiza el sistema de gestión de dependencias.

```
$ rosdep update
```

Una vez que ROS está funcionando adecuadamente, se procede con la instalación de ARIA. Para ello se ejecutan los comandos listados a continuación.

1. Se crea un entorno de trabajo *Catkin*, con la finalidad que la terminal acceda algunos comandos específicos de ROS.

```
$ mkdir -p ~/ catkin_ws / src
$ cd ~/ catkin_ws / src
$ catkin_init_workspace
$ cd ~/ catkin_ws
$ catkin_make
```

2. Cada vez que sea necesario compilar paquetes empleando el espacio *Catkin*, se debe ejecutar el script especial devel/setup.bash.

```
$ cd catkin_ws
$ . devel/setup . bash
```

Teniendo ROS instalado se procede a incluir los paquetes necesarios para el uso de ARIA, el proceso de instalación es similar al descrito anteriormente.

1. Se clona el repositorio de ARIA mediante el comando *git clone*.

```
$  git clone https://github.com/xh20/Aria.git
```

2. Se instala el paquete de ARIA empleando el comando *dpkg*.

```
$  sudo dpkg -i libaria_2.9.4+ubuntu16_amd64.deb # install Aria
```

3. Se obtiene de *github* las bibliotecas necesarias para integrar Aria en ROS en el espacio *Catkin* y se copila el espacio de nuevo.

```
$ cd ~/catkin_ws/src
$ git clone https://github.com/amor-ros-pkg/rosaria
$ cd ~/catkin_ws
$ catkin_make --force-make
```

Para inicializar ROS/ARIA y conectarse al vehículo Amigobot se ejecutan los siguientes comandos en una terminal.

```
$ roscore
$ rosrun rosaria RosAria _port:=192.168.0.107:8101
```

Estos comandos inicializan ROS y ejecutan al API de ARIA para conectarse con el robot. Note que el comando *roscore* no se debe ejecutar si se está empleando la versión de Ubuntu desarrollada por Ubiquity Robotics (versión instalada en la *Raspberry Pi*). Al momento de ejecutar el comando *rosrun rosaria* para conectarse con el robot se debe tener en cuenta la dirección *ip* y el puerto de red configurado en el módulo Lantronix del vehículo que se esté usando.

3.5. Modelo cinemático del robot Amigobot

En el caso de los robots móviles de tipo uni-ciclo es de gran utilidad conocer el modelo cinemático al momento de diseñar controladores o estrategias de navegación, por lo cual a continuación se describe dicho modelo el cuál corresponde a la plataforma de pruebas usada en este trabajo.

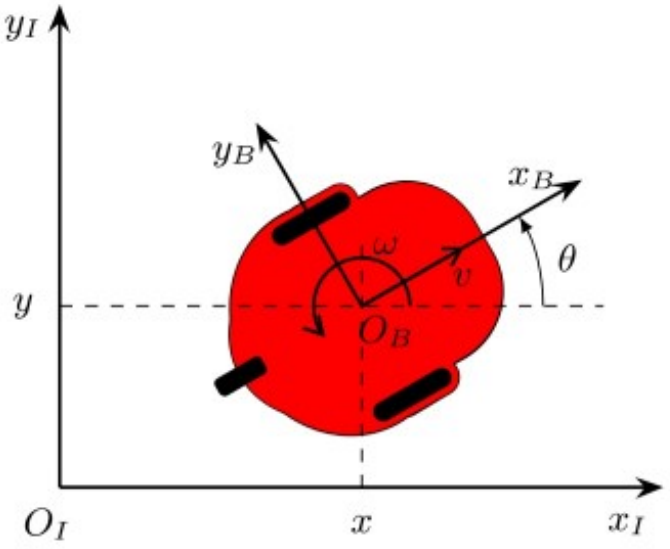


Figura 3.5: WMR Amigobot y sus marcos de referencia.

El modelo cinemático de un vehículo tipo uni-ciclo, bajo la restricción no holonómica de rodadura pura y sin deslizamiento, se expresa de la siguiente manera [19]

$$\dot{\boldsymbol{q}} = S(\boldsymbol{q})\boldsymbol{v}_e \tag{3.1}$$

donde $\boldsymbol{q}(t)$, $\dot{\boldsymbol{q}}(t) \in \mathbb{R}^3$ se definen como

$$\boldsymbol{q} = \begin{bmatrix} x_c \\ y_c \\ \theta \end{bmatrix}, \quad \dot{\boldsymbol{q}} = \begin{bmatrix} \dot{x}_c \\ \dot{y}_c \\ \dot{\theta} \end{bmatrix}, \tag{3.2}$$

denotando $x_c(t),\ y_c(t) \in \mathbb{R}$ la posición al centro de masa del vehículo con respecto del marco de referencia $\{I\}$ en coordenadas cartesianas y $\theta(t) \in \mathbb{R}$ representa la orientación del vehículo con relación al marco de referencia $\{B\}$, como se muestra en la Figura 3.5. Las variables $\dot{x}_c$ y $\dot{y}_c$ denotan las componentes cartesianas de la velocidad lineal $v(t) \in \mathbb{R}$ y la velocidad angular $\omega(t) \in \mathbb{R}$, respectivamente, por lo que la matriz $S(\boldsymbol{q}) \in \mathbb{R}^{2\times 3}$ se define como

$$S(\boldsymbol{q}) = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \tag{3.3}$$

y el vector de velocidad $\boldsymbol{v}_e(t) \in \mathbb{R}^2$ se define como

$$\boldsymbol{v}_e = \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{3.4}$$

Nótese que se asume que el centro de rotación y el centro de masa coinciden.

Capítulo 4

Esquema de navegación

4.1. Introducción

En este trabajo, se presenta un sistema de navegación con el cual es posible validar el funcionamiento de la plataforma de pruebas Amigobot. El sistema estudiado es adaptable a otras plataformas de prueba, como drones o robots móviles en general.

Hay diferentes técnicas para abordar la generación de rutas o trayectorias, una de las más populares, por su simplicidad y elegancia matemática, es la de campos potenciales [20]. Los campos de velocidad para guiado de robots se han estudiado a través de ecuaciones de hidrodinámica. Una de las técnicas más populares para evasión de obstáculos basada en campos potenciales es la función repulsiva convencional.

Varias alternativas han surgido para mejorar la función repulsiva convencional, la cual es especialmente útil para implementación en tiempo real debido a su dependencia exclusiva del gradiente local y la ausencia de necesidad de información global. No obstante, una desventaja significativa de este enfoque radica en la presencia de mínimos locales, lo que puede resultar en la incapacidad del vehículo para alcanzar la posición meta. Diversos métodos basados en campos potenciales han sido desarrollados como mejoras al esquema repulsivo básico. Ejemplos de estos incluyen el método de Objetivos No Alcanzables con Obstáculos Cercanos (GNRON) [21], las Funciones Potenciales Harmónicas (HPF) [22], el Campo Potencial Híbrido [23], los Campos Potenciales Artificiales Evolucionarios [24], y el Campo de Potencial Artificial Mejorado [25]. Una forma de mitigar la principal desventaja de los campos estáticos, es decir, la presencia de mínimos locales que puedan

obstaculizar la realización de la tarea, es mediante la adopción de campos potenciales de velocidad.

Este capítulo describe el proceso a seguir para realizar la integración de los elementos vistos en los capítulos anteriores, con el fin de obtener el sistema de navegación enfocado en robots móviles rodantes.

4.2. Campos potenciales de velocidad

La generación de trayectorias es un problema fundamental dentro de la navegación autónoma. El sistema de navegación estudiado en este trabajo permite abordar el problema de la generación de trayectorias y evasión de obstáculos, por lo cual se empleó el algoritmo de campos potenciales de velocidad.

La idea básica detrás de los campos potenciales es tratar al robot como una partícula puntual en el espacio Euclidiano bajo la influencia de un campo potencial artificial U. El campo U se construye de tal manera que el robot es atraído a su posición deseada mientras es repelido por los obstáculos. Este método requiere únicamente información de gradiente local, es decir, sólo hace uso de la posición actual del vehículo. Su matemática es sencilla y elegante, por lo cual es adecuado para implementarse en tiempo real. No obstante, este método presenta el problema de mínimos locales. En la Figura 4.1 se muestra un campo potencial, donde la partícula, en cierta configuración, puede quedar atrapada en un mínimo local, es decir una barrera, que le impida llegar a la posición meta.

En la mayoría de los planificadores de ruta por campos potenciales, el campo U se construye como un campo aditivo, que consta de un componente que atrae al robot a su posición deseada y un segundo componente que repele al robot de los obstáculos.

La planificación de ruta puede verse como un problema de optimización, a saber, el problema de encontrar el mínimo global de U empezando desde una condición inicial, y a menudo se utilizan métodos de descenso del gradiente para encontrar una solución. En física, un campo de fuerza conservador se puede escribir como el gradiente negativo de una función potencial. Por lo tanto, podemos interpretar el gradiente en descenso análogamente como una partícula que se mueve bajo la influencia de la fuerza $\boldsymbol{F} = -\nabla U$.

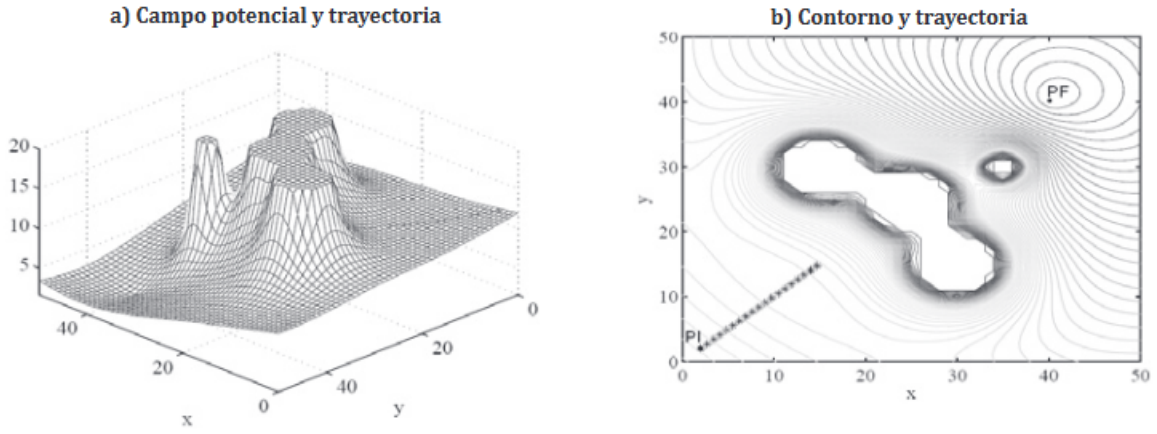


Figura 4.1: a) Campo potencial con la presencia de una barrera, b) Contorno del campo potencial con la presencia de una barrera [2].

4.3. Estrategia de navegación

La estrategia de navegación presentada a continuación está basada en la teoría de campos de velocidad, los cuales se encargan de generar una trayectoria deseada, y que al seguirla el vehículo evada los obstáculos del ambiente. La ruta deseada propuesta en este trabajo es una ruta circular. Para generar un campo de velocidad $\boldsymbol{V} = [V_x \; V_y]^T \in \mathbb{R}^2$ en una ruta circular, es necesario calcular dos campos vectoriales: uno de aproximación $\boldsymbol{V}_{ac} \in \mathbb{R}^2$ y otro tangencial $\boldsymbol{V}_{tg} \in \mathbb{R}^2$. El campo de aproximación está definido por el vector que apunta directamente a la ruta, y se obtiene mediante la sustracción normalizada del punto más cercano a la trayectoria, como se describe en [26]. Se desea seguir una ruta circular con radio r_{tr} y con centro localizado en el punto (o_x, o_y), el cual está definido por

$$(x - o_x)^2 + (y - o_y)^2 = r_{tr}^2. \tag{4.1}$$

Para obtener el campo de aproximación es necesario calcular el punto más cercano desde la posición del vehículo al círculo que describe la ruta deseada. Este punto es calculado por

$$\text{mín}\left(\sqrt{(x - x_{tr})^2 + (y - y_{tr})^2}\right) \tag{4.2}$$

donde x_{tr} y y_{tr} son todos los puntos que conforman la ruta circular, es decir, el grupo de puntos que satisfacen la ecuación (4.1). Ahora, definamos los vectores

$$\bar{\boldsymbol{\xi}} = \begin{bmatrix} \bar{x} \\ \bar{y} \end{bmatrix} = \begin{bmatrix} x - o_x \\ y - o_y \end{bmatrix}. \tag{4.3}$$

donde $\bar{x}$ y $\bar{y}$ denotan la diferencia entre cualquier punto del área de trabajo y el centro de la ruta circular. A su vez, $\tilde{x}$ y $\tilde{y}$ son los errores de posición desde cualquier punto del espacio de trabajo hasta su punto más cercano dentro de la trayectoria deseada. Las coordenadas del punto más cercano a la trayectoria se obtienen como

$$x_{cl} = o_x + r_{tr}\cos(\alpha_{cl}) \tag{4.4}$$

$$y_{cl} = o_y + r_{tr}\sin(\alpha_{cl}) \tag{4.5}$$

donde α_{cl} denota el vector formado por la posición actual y el punto más cercano a la ruta. De tal modo el error antes mencionado se define como

$$\tilde{\boldsymbol{\xi}} = \begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix} = \begin{bmatrix} x - x_{cl} \\ y - y_{cl} \end{bmatrix} \tag{4.6}$$

Tomando las Ecuaciones (4.4)-(4.5) para calcular x_{cl} y y_{cl}, podemos definir la distancia entre la posición actual y el punto más cercano a la ruta como

$$\|\tilde{\boldsymbol{\xi}}\| = \sqrt{\tilde{x}^2 + \tilde{y}^2}. \tag{4.7}$$

El ángulo α_{cl} se obtiene mediante

$$\alpha_{cl} = \text{atan2}(\bar{y}, \bar{x}). \tag{4.8}$$

El campo de aproximación se define como

$$\boldsymbol{V}_{ac} = \frac{\tilde{\boldsymbol{\xi}}}{\|\tilde{\boldsymbol{\xi}}\|}. \tag{4.9}$$

Denotamos el vector de las derivadas parciales de x_{cl} e y_{cl} como $\boldsymbol{v}_c = [V_{x_c}\ V_{y_c}]^T$, entonces el campo tangencial es definido por

$$\boldsymbol{V}_{tg} = \frac{\boldsymbol{v}_c}{\|\boldsymbol{v}_c\|}. \tag{4.10}$$

El campo de velocidad se obtiene mediante la suma ponderada normalizada del campo de aproximación y tangencial obtenidos en (4.9) y (4.10), respectivamente.

$$\boldsymbol{V} = \frac{F_1\boldsymbol{V}_{ac} + F_2\boldsymbol{V}_{tg}}{\|F_1\boldsymbol{V}_{ac} + F_2\boldsymbol{V}_{tg}\|}, \tag{4.11}$$

donde F_1 y F_2 son funciones de la distancia euclidiana entre un punto en el espacio y la trayectoria deseada, que vienen dadas por

$$F_1 = \frac{2}{1 + e^{-\gamma \|\tilde{\boldsymbol{\xi}}\|}} - 1, \quad F_2 = 1 - F_1. \tag{4.12}$$

4.3.1. Campo de evasión

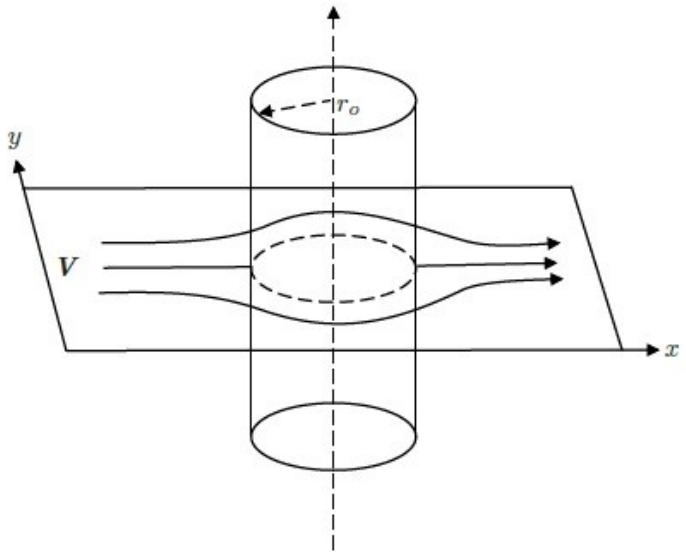


Figura 4.2: Flujo ideal alrededor de un cilindro.

El método de evasión de obstáculos se basa en el uso de la teoría hidrodinámica, específicamente mediante el flujo ideal generado alrededor de un cilindro, como se muestra en la Figura 4.2.

El flujo plano en estado estacionario se define mediante un campo de velocidad

$$\boldsymbol{V}_o(\boldsymbol{p}) = \begin{bmatrix} u(x,y) \\ v(x,y) \end{bmatrix} \tag{4.13}$$

en el punto

$$\boldsymbol{p} = \begin{bmatrix} x \\ y \end{bmatrix} \in \Omega \tag{4.14}$$

donde $\Omega \subset \mathbb{R}^2$ es el dominio ocupado por el fluido.

Se propone utilizar un obstáculo circular, el cual interfiere con el campo de flujo vectorial, donde (x_o, y_o) es la posición central del obstáculo, con radio r_o. Definimos el error

de posición con respecto del centro del obstáculo como

$$\tilde{\boldsymbol{\xi}}_e = \begin{bmatrix} x - x_o \\ y - y_o \end{bmatrix} = \begin{bmatrix} \tilde{x}_o \\ \tilde{y}_o \end{bmatrix}. \tag{4.15}$$

La función potencial de velocidad propuesta para un fluido ideal alrededor de un obstáculo circular es

$$\phi(\tilde{\boldsymbol{\xi}}_o) = \left(1 + \frac{r_o^2}{\tilde{x}_o^2 + \tilde{y}_o^2}\right)(\tilde{x}_o \cos\beta + \tilde{y}_o \sin\beta), \tag{4.16}$$

donde

$$\beta = \text{atan2}(V_y, V_x) \tag{4.17}$$

es el ángulo del vector de campo de velocidad en ese punto. La velocidad de flujo alrededor del cilindro está definida como el gradiente de la función ϕ, es decir, $\boldsymbol{V}_{ev} = \nabla\phi$. Entonces, la componente x del campo vectorial de velocidad es

$$V_{ev_x} = -\frac{r_o^2}{\tilde{x}_o^2 + \tilde{y}_o^2}(\tilde{x}_o \cos\beta + \tilde{y}_o \sin\beta)\tilde{x}_o + (1 + \frac{r_o^2}{\tilde{x}_o^2 + \tilde{y}_o^2})\cos\beta \tag{4.18}$$

y la función de velocidad en la coordenada y es

$$V_{ev_y} = -\frac{r_o^2}{\tilde{x}_o^2 + \tilde{y}_o^2}(\tilde{x}_o \cos\beta + \tilde{y}_o \sin\beta)\tilde{y}_o + (1 + \frac{r_o^2}{\tilde{x}_o^2 + \tilde{y}_o^2})\sin\beta. \tag{4.19}$$

Entonces, el vector velocidad para la maniobra evasiva es

$$\boldsymbol{V}_{ev} = \begin{bmatrix} V_{ev_x} \\ V_{ev_y} \end{bmatrix}. \tag{4.20}$$

Para visualizar el efecto del campo evasivo, se puede decir que éste redirige la dirección del vector del campo atractivo, modificando la referencia de velocidad para intentar evitar el obstáculo. Por lo tanto, el vector de velocidad evasiva se reemplaza por el vector de velocidad anterior y también se normaliza

$$\boldsymbol{V} = \frac{\boldsymbol{V}_{ev}}{\|\boldsymbol{V}_{ev}\|}. \tag{4.21}$$

Este vector de velocidad se definió mediante coordenadas rectangulares x e y; sin embargo, para dar dichas referencias de velocidad al vehículo, se necesita el vector de velocidad en componentes polares de magnitud y ángulo.

Cabe aclarar que el vector de velocidad $\boldsymbol{V}$ dado por (4.11), es obtenida en ausencia de obstáculos en el ambiente de trabajo. Posteriormente, el vector de velocidad es actualizado de manera iterativa empleando (4.17)–(4.21) por cada obstáculo en el ambiente.

4.4. Estrategia de control

En la sección 3.5 capítulo anterior se describió el modelo cinemático del robot móvil con ruedas. En la ecuación (3.1), la entrada de control del robot móvil corresponde con el vector $\boldsymbol{v}_e$ que contiene la velocidad lineal v y angular ω. A partir del campo de velocidad $\boldsymbol{V}$ generado, la entrada de velocidad lineal se obtiene como la norma euclidiana de dicho vector, es decir

$$v = \sqrt{V_x^2 + V_y^2}. \tag{4.22}$$

El ángulo asociado con el vector de velocidad V determina el ángulo de dirección deseado θ_d de la siguiente manera

$$\theta_d = \text{atan2}(V_y, V_x), \tag{4.23}$$

por lo que, para hacer un control de seguimiento de orientación, se debe obtener primero la derivada del vector V, mediante lo cual se puede calcular la velocidad angular deseada

$$\omega_d = \frac{V_x \dot{V}_y - V_y \dot{V}_x}{||V||^2}. \tag{4.24}$$

Finalmente, definiendo el error de orientación como $\dot{e}_i = \tilde{\theta} = \theta_d - \theta$, la velocidad angular usada como entrada de control está dada por

$$\omega = \omega_d + k_p \tilde{\theta} + k_i(e_i) \tag{4.25}$$

$$\dot{e}_i = \tilde{\theta} \tag{4.26}$$

donde k_p, $k_i \in \mathbb{R}$ son constantes positivas asociadas con la ganancia proporcional y la ganancia integral respectivamente.

Capítulo 5

Resultados y discusión

La descripción de la implementación del sistema de navegación en el robot móvil con ruedas Amigobot se presenta en este capítulo. Las características específicas de cada componente de la plataforma experimental que incluye al robot móvil, el sistema de visión y el esquema de navegación se han mencionado en los capítulos anteriores. Sin embargo, a continuación se describen, a manera de resumen, las características del sistema experimental y su funcionamiento en conjunto para obtener el desempeño deseado. Finalmente se presentan y discuten los resultados obtenidos de las pruebas hechas al sistema.

5.1. Esquema experimental

La Figura 5.1 muestra de manera esquemática el sistema experimental que se ha puesto en operación en este trabajo, y en el cual se implementó el algoritmo de navegación. El sistema de navegación implementado en este trabajo está compuesto del hardware y software que se menciona a continuación.

En cuestión de hardware se emplea:

- Sistema de visión *Optitrack*.
- *Raspberry Pi* 4
- Plataforma de pruebas Amigobot.
- Router

El software necesario es el siguiente:

- *Optitrack Motive*
- *MATLAB R2015* (es posible emplear hasta la versión R2017)
- ROS/ARIA
- Sistema operativo Linux

5.1.1. Descripción del funcionamiento del sistema

El esquema de navegación implementado se compone principalmente de dos partes, el sistema de visión *Optitrack* presentado en el Capítulo 2 y la plataforma de pruebas Amigobot. El primero sirve para la obtención de la posición del robot, mientras que el segundo proporciona una plataforma de prueba en la cual es posible validar los algoritmos de control y navegación estudiados.

En el esquema de navegación experimental presentado, existen distintos componentes que interactúan para alcanzar el objetivo establecido. En la Figura 5.1 se pueden observar de manera sencilla los distintos elementos que componen el esquema, tanto componentes de software como de hardware. Se puede notar que el esquema de control se ejecuta en una computadora que funciona como estación controladora, la cual cuenta con sistema operativo *Windows*. Dentro de este sistema se estima la posición del robot por medio del software *Motive* y se realiza el cálculo de las entradas de control que serán aplicadas al robot. Otro componente importante del sistema es la computadora *Raspberry Pi* que se emplea como puente de comunicación entre la estación controladora y el robot móvil con ruedas. Este componente permite procesar las entradas de control calculadas y enviarlas al vehículo para que sean ejecutadas. Finalmente es posible identificar al robot Amigobot el cual realiza la tarea asignada.

La implementación del controlador y el esquema de navegación se realizó en el entorno *Simulink* de *MATLAB*. El controlador obtiene la posición del robot mediante bibliotecas dinámicas *.dll* y bibliotecas estáticas *.lib* proporcionadas por el software de desarrollo de *Optitrack*. Una vez que se tiene la posición del robot, esta se utiliza en el cálculo de los campos de velocidad, como se describe en el Capítulo 4, dentro de la estrategia de navegación. Posteriormente, el algoritmo de navegación genera las consignas de control, es decir, se genera tanto la velocidad lineal como angular, las cuales son necesarias para

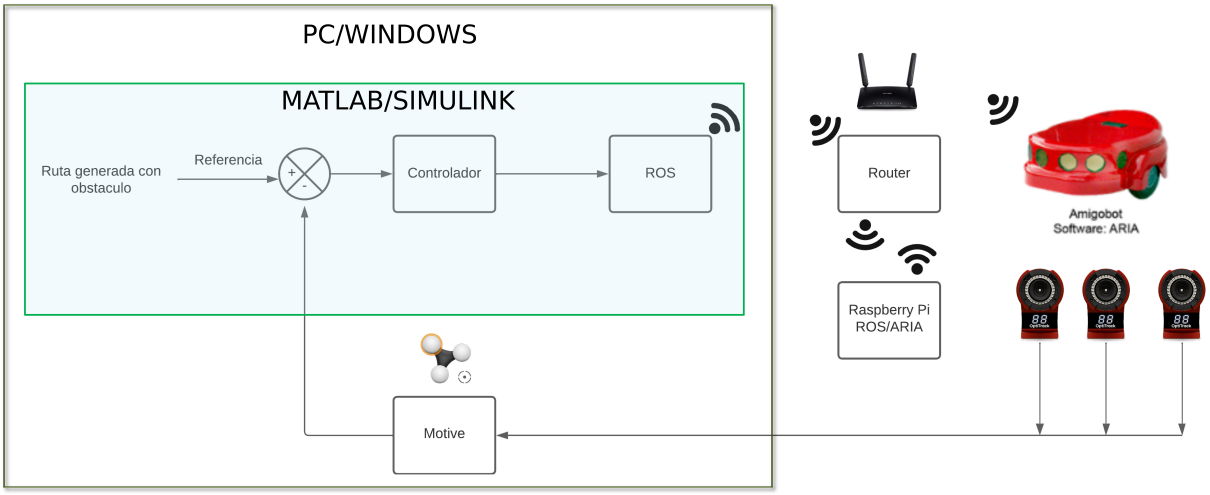


Figura 5.1: Diagrama esquemático del sistema experimental en que se implementó el algoritmo de navegación.

mover el robot y se envían a la estación intermedia que permite establecer la comunicación con el vehículo mediante los bloques de ROS en *Simulink*. Cabe mencionar que esta estación intermedia corresponde a la computadora *Raspberry Pi*, en la cual se tienen instaladas las herramientas de ROS/ARIA. Finalmente, la estación intermedia envía al robot las consignas calculadas para realizar la tarea establecida.

5.2. Implementación en software

La implementación de la estrategia de navegación fue desarrollada en *MATLAB/Simulink*, como se explicó en la sección anterior. La Figura 5.2 muestra el modelo de bloques con el cual se implementó el esquema de navegación.

Esta implementación se compone de diversos bloques, los cuales realizan tareas necesarias para la correcta generación y aplicación del método de navegación estudiado. Como se observa en la Figura 5.2 tenemos cuatro bloques o subsistemas principales, los cuales se explican a continuación.

Subsistema: TrackableLocation

Este subsistema se basa en el programa *Natural Point Motive (Optitrack) API interface to Matlab and Simulink 64bit*, desarrollado por Or Hirshfeld [27]. Este programa permite inicializar la comunicación con el software *Motive*, empleando sus bibliotecas dinámicas

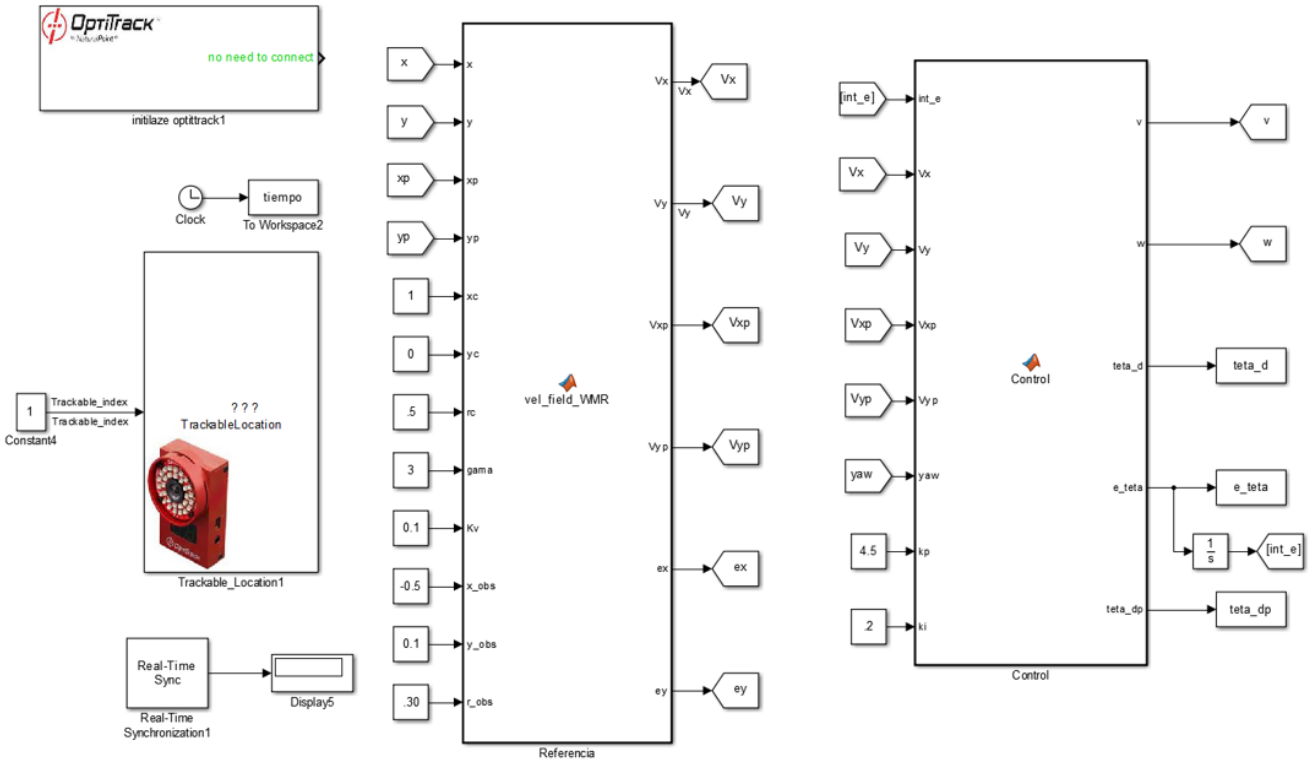


Figura 5.2: Modelo en Simulink para la implementación de la estrategia de navegación.

y estáticas con el fin de obtener la posición y orientación del robot a partir de los datos proporcionados por el mencionado software.

El subsistema realiza tres funciones principales:

- Estimar la posición del robot por medio del sistema de visión.
- Corregir el sistema de referencia y transformar el formato del ángulo de giro *yaw*, el cual es obtenido en un rango de valores de -179° a 180°, al formato de 0° a 359°.
- Calcular la velocidad estimada del robot por medio de la posición estimada por el sistema de visión.

Para más información acerca de este bloque, sus subsistemas y funciones véase el Apéndice B.

Subsistema: vel_field_WMR - Referencia

Este bloque contiene las funciones que calculan los campos potenciales de velocidad y la ruta a seguir por el robot. Genera todas las velocidaes calculando las Ecuaciones (4.1)-(4.21). Como parámetros de entrada solicita los datos:

- Una velocidad lineal Kv.
- Las coordenadas (x_{obs}, y_{obs}) del obstáculo y el radio r_{obs} del mismo.
- Las coordenadas (x_c, y_c) y radio r_c de la ruta circular a seguir.
- La ponderación γ, la cual se utiliza para calcular el campo de velocidad V, empleado la Ecuación (4.11).

Como variables de salida entrega los valores de:

- Componentes de velocidad lineal de referencia: V_x y V_y.
- Derivadas temporales de la velocidad lineal: $\dot{V}_x$ y $\dot{V}_y$.
- Los errores de posición entre el punto más cercano de la trayectoria circular y la posición actual: e_x y e_y.

Para más información acerca de este bloque y sus subsistemas y funciones véase el Apéndice C.

Subsistema: Control

Este bloque o subsistema es el encargado de generar las consignas de control, las cuales finalmente son enviadas al robot mediante el sistema ROS. Este bloque básicamente ejecuta las ecuaciones de control (4.22)-(4.26). Como parámetros de entrada, esta función solicita los valores de:

- Las velocidades lineales de referencia V_x y V_y.
- Las derivadas temporales de las velocidades, $\dot{V}_x$ y $\dot{V}_y$.
- El ángulo de orientación del robot θ.

- La constante proporcional k_p y la constante integral k_i.
- La integral del error de posición.

Las señales de salida de este bloque son

- Las consignas de velocidad lineal y angular que se deben enviar al robot: v y ω.
- El ángulo de orientación deseada y error de ángulo de orientación: θ_d y $\tilde{\theta}$.
- Una estimación de la velocidad angular deseada, a través de la derivada del ángulo de orientación deseado, es decir $\dot{\theta}_d$.

La señal de control asociada con la velocidad angular ω debe pasar por un bloque de saturación antes de ser aplicada al robot. Esto debido a que las ganancias k_p y k_i del controlador fueron seleccionadas para optimizar el desempeño en errores pequeños de orientación, por lo cual errores grandes generan velocidades angulares de alta magnitud causando inestabilidad al sistema. Para más información acerca de este bloque, sus subsistemas y funciones véase el Apéndice D.

Subsistema: ROS

Este subsistema contiene las funciones necesarias para enviar las consignas de control al intermediario, es decir, el sistema que ejecuta ROS/ARIA. Este subsistema realiza las siguientes funciones:

- Generar el mensaje a enviar vía ROS, seleccionando el tipo de mensaje. Para enviar las consignas, se emplea el mensaje de tipo *geometry_ msgs/Twist*.
- Incluir las consignas de control y el tipo de mensaje a enviar mediante un bloque *Bus Assignment*.
- Enviar el mensaje seleccionando el módulo de software de ROS, el cual es el encargado de mandar las consignas y que éstas sean ejecutadas por el robot. Este módulo es *RosAria/cmd_ vel*.

5.2.1. Ejecución del programa

Antes de ejecutar el programa y asegurarse de que el robot cumpla con la tarea propuesta, es necesario seguir los siguientes pasos para garantizar que el programa no devuelva errores.

1. Ejecutar en *MATLAB* el archivo *initlaze_startup_parametres_setup_motive_1_5.m* que se encuentra en el directorio del proyecto. Nótese que la llave física USB del programa Motive debe estar conectada para tener acceso a las bibliotecas del mismo. Véase el Apéndice A para obtener más información sobre el script.

2. Finalmente, se debe ejecutar el programa en Simulink que contiene los subsistemas ya mencionados. En el caso de este trabajo, el programa es "Control_cp_WMR.slx" y se encuentra en el directorio de ejecución de los archivos mencionados anteriormente. Este directorio también debe incluir las bibliotecas, archivos de inclusión del programa base propuesto por [27], además del proyecto con en el cual está definido el objeto de interés *Rigidbody* y el archivo de calibración que se obtienen del proceso explicado en el Capítulo 2.

5.3. Resultados experimentales

La validación del desempeño del sistema de navegación presentado en este trabajo se realizó considerando que se desea seguir una ruta circular tomando en cuenta las siguientes condiciones:

- Obstáculo circular colocado en las coordenadas $(x, y) = (-0.5, 0.1)$ m y con un radio $r = 0.30$ m.
- Trayectoria circular a cumplir con centro en las coordenadas $(x, y) = (1, 0)$ m y radio de $r = 0.50$ m.
- La velocidad lineal de referencia del robot fue de 0.1 m/s al momento de realizar pruebas experimentales.
- Las condiciones iniciales del robot se establecieron en $x(0) = -1.5$ m, $y(0) = 0$ m y $\theta(0) = 0°$.

- Las ganancias del controlador de orientación se determinaron por medio de un proceso de prueba y error seleccionando los valores $k_p = 4.5$ y $k_i = 0.2$.

La Figura 5.3 muestra la ruta seguida por el robot móvil con ruedas Amigobot. La línea

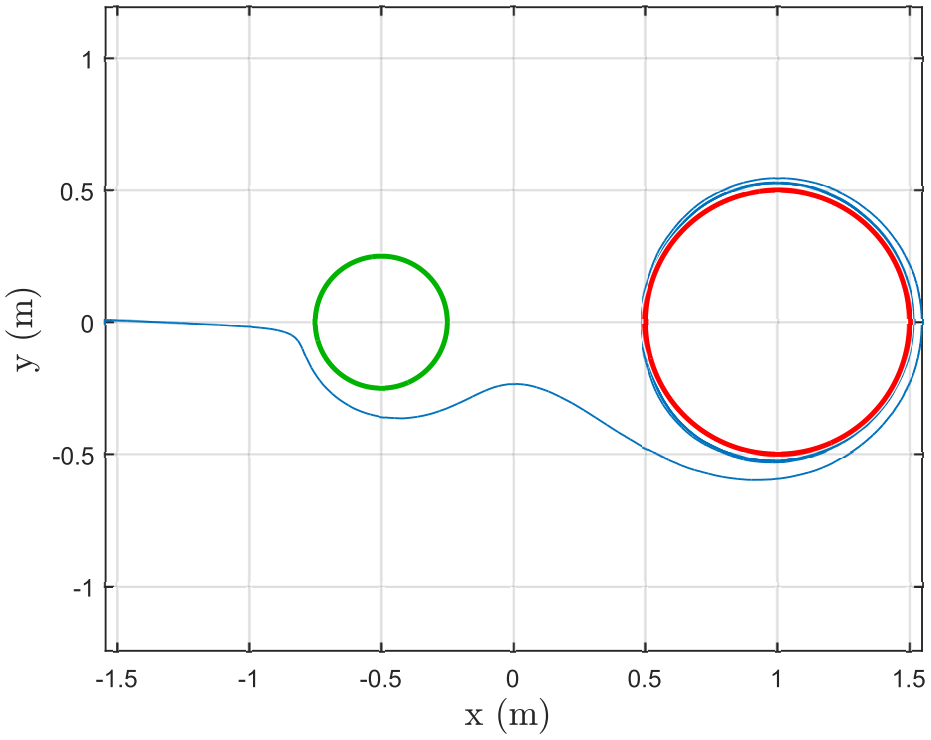


Figura 5.3: Ruta seguida por el robot en el experimento de navegación.

azul muestra la ruta seguida por el robot, mientras que en verde se presenta el obstáculo circular y en rojo el círculo que representa la ruta deseada. Como se puede observar en la ruta seguida, el robot esquiva perfectamente el obstáculo pasando a un costado y manteniendo la dirección hacia la ruta deseada. De igual manera se puede observar que el vehículo alcanza la ruta deseada y se mantiene cercana a ella.

La Figura 5.4 muestra la gráfica de las consignas de control enviadas al robot. Podemos observar cómo la velocidad lineal se mantiene cercana a 0.1 m/s. Adicionalmente, la velocidad angular aplicada al robot también es presentada. Se puede notar cómo su magnitud cambia constantemente con el objetivo de mantener al robot en la orientación deseada.

En la Figura 5.5 podemos observar los errores de posición, y orientación del robot obtenidos en el experimento de navegación. En la gráfica del error de posición, $\tilde{x}(t)$, se puede observar cómo el error varía de forma sinusoidal, de manera que se mantiene

acotado en magnitud de ±2.5 cm, lo cual es aceptable considerando que la ruta deseada tiene un diámetro de un metro.

La segunda gráfica presentada en la Figura 5.5 muestra el error de posición, $\tilde{y}(t)$, obtenido en el experimento. En este caso se nota cómo en un inicio el error es cercano a cero, posteriormente aumenta de manera considerable para finalmente quedar acotado en ±2.7 cm después del segundo treinta. Este comportamiento se presenta debido a que la ruta de navegación provoca que el robot se aleje del valor deseado para poder evitar el obstáculo. Finalmente, en la tercera gráfica tenemos el error de orientación, $\tilde{\theta}(t)$, de la ruta realizada por el robot, la cual se mantiene de manera regular por debajo de los ±2°. En todas las gráficas obtenidas se observa ruido debido a que la señal procesada de las cámaras introduce variaciones de estimación. Sin embargo, a pesar de este efecto indeseable se cumple adecuadamente con la tarea asignada.

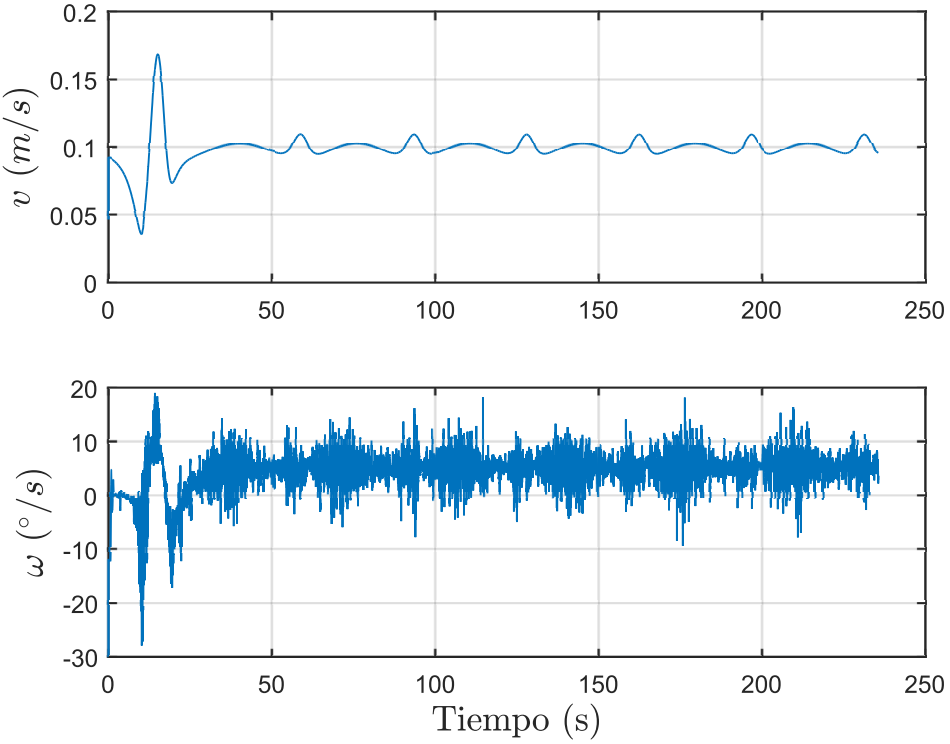


Figura 5.4: Entradas de control en el experimento de navegación.

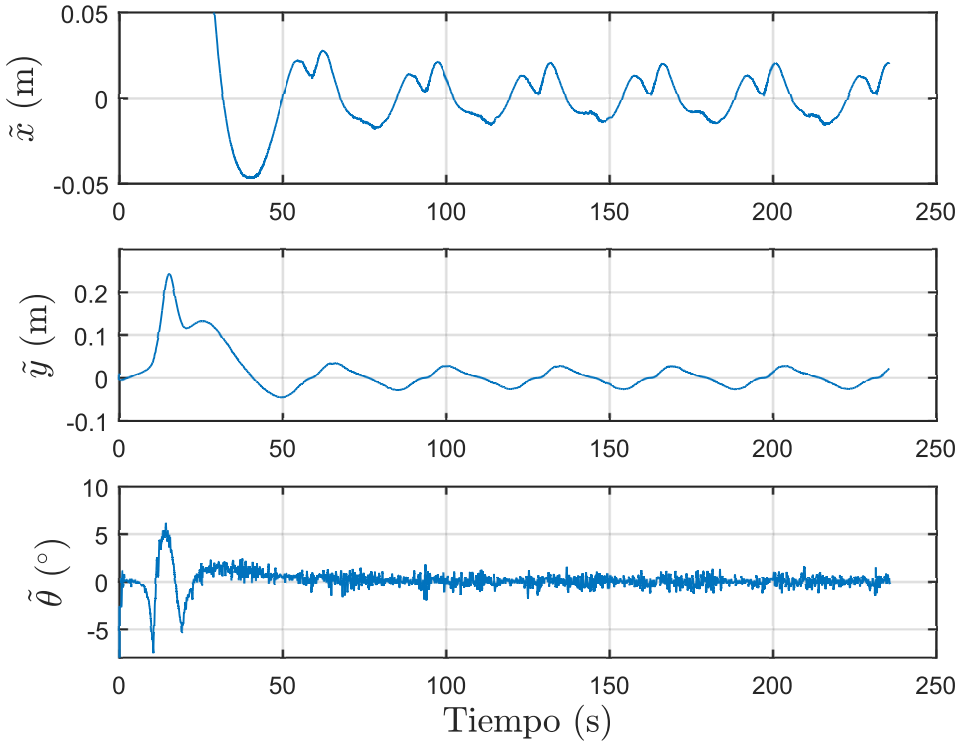


Figura 5.5: Gráficas de error de posición y orientación en el experimento de navegación.

Capítulo 6

Conclusiones

En este trabajo se ha demostrado que es posible implementar un sistema de navegación en el robot móvil rodante Amigobot empleando el sistema de visión por computadora *Optitrack*. Con esto se cumple el objetivo general establecido. Para alcanzar este resultado el sistema implementado es capaz de:

- Obtener la pose del vehículo de manera precisa.

  Para obtener la pose, se empleó el programa *Motive* y sus bibliotecas dinámicas. El mayor desafío al realizar la integración con *Simulink/MATLAB* fue la incompatibilidad entre *Motive* y las versiones actuales de MATLAB. Por lo tanto, al optar por una versión desactualizada de MATLAB, fue posible obtener la posición y orientación del WMR, además de emplear estos datos para lograr la trayectoria deseada propuesta en la Sección 5.3.

- Comunicar el robot con el entorno de desarrollo *Simulink/MATLAB* mediante ROS/ARIA

  Este fue un reto importante en el desarrollo del proyecto. El API de desarrollo ARIA es antiguo, ya que el robot Amigobot salió al mercado en 2004. Por lo tanto, al momento del desarrollo de este trabajo, no fue posible compilarlo y hacerlo funcionar en la versión de *MATLAB* que se utilizó. Como solución se empleó un sistema Linux como intermediario para lograr la comunicación con el robot.

- Navegar en el ambiente de trabajo.

  La integración del sistema se logró mediante el uso de MATLAB/Simulink versión 2015b, donde es posible ejecutar tanto el programa que interactúa con el sistema de visión como el sistema ROS.

El principal inconveniente para obtener el sistema de navegación fue la incompatibilidad de los componentes además de que parte del software se encuentra obsoleto para las herramientas disponibles actualmente.

El correcto funcionamiento del vehículo Amigobot con el sistema de visión *Optitrack* se validó experimentalmente implementando un sistema de navegación basado en campos potenciales de velocidad. De los resultados obtenidos se puede establecer que el sistema cumple con la tarea de navegación establecida evadiendo el obstáculo que se considera en el ambiente.

Un aspecto a considerar es que dadas las características del algoritmo de navegación se restringe la velocidad lineal de referencia proporcionada al robot. En el experimento realizado, el mencionado valor de velocidad se estableció en 10 cm/s provocando un movimiento lento. Este aspecto debe ser analizado con el propósito de presentar alguna mejora al algoritmo.

6.1. Trabajo futuro

Como trabajo futuro se propone lo siguiente:

- Actualizar el programa desarrollado para funcionar en versiones más actuales de *MATLAB* como R2021 o R2024.
- Copilar ROS noetic para funcionar en la versión actual de Ubuntu, siendo esta la versión 24.04 al año de redacción de este documento.
- Compilar el ambiente de desarrollo ARIA de forma que se pueda emplear en *MATLAB*, eliminando la necesidad de usar Linux como intermediario entre *MATLAB* y la mencionada biblioteca.
- Abordar el problema de navegación en ambientes cambiantes dinámicos, mediante campos potenciales u otras técnicas.
- Abordar el problema de mapeo y localización simultanea sacando el mayor provecho a las herramientas de hardware disponibles en robot Amigobot.

Apéndice A

Programa de inicialización de bibliotecas *Motive*

El programa de inicialización se debe ejecutar antes de correr el programa de Simulink. Es indispensable para que MATLAB ejecute y obtenga información del programa *Motive* a través de sus bibliotecas dinámicas (dll).

A continuación se presenta el programa encargado de inicializar y configurar el acceso a las bibliotecas de Motive.

```
% Limpiar todo el espacio de trabajo
clear all;

%Agregar la ruta para los iconos de los bloques
addpath('D:\Projectos\Vehiculos_terrestres\Amigobot\campos_potenciales_GR\original\');

% Archivo de proyecto Motiove que contiene la calibracion de la camara y la
    configuracion de los marcadores
project_file =
    'D:\Projectos\Vehiculos_terrestres\Amigobot\campos_potenciales_GR\original\';

% Ubicacion que contiene archivos .dll (bibliotecas dinamicas) y .lib (bibliotecas
    estaticas)
inc_location =
    'D:\Projectos\Vehiculos_terrestres\Amigobot\campos_potenciales_GR\original\inc\';

% Ubicacion que contiene archivos .h (encabezados) fijos
lib_location =
    'D:\Projectos\Vehiculos_terrestres\Amigobot\campos_potenciales_GR\original\lib\';
```

```
15
16 % Inicializacion de ROS
17 rosinit('192.168.0.100')
```

Apéndice B

Bloque de obtención de pose *TrackableLocation*

El subsistema *TrackableLocation*, se muestra en la Figura B.1. Este subsistema es el encargado de obtener los valores de posición y orientación del robot, entregado instantáneamente por el sistema de visión *Optitrack*. Debido a la orientación del marco de coordenadas manejado por el sistema de visión, hay que aplicar una conversión para que coincida con la orientación del marco inercial propuesta en el modelo cinemático que se presenta en esta tesis. Para ello, la coordenada y del marco inercial corresponde a $-z$ del sistema Motive, lo cual se puede observar con un bloque de ganancia -1. Se observa un bloque intermedio que sirve para corregir el ángulo de orientación del vehículo, para que el ángulo de salida θ presente valores entre $-\pi/2$ a $\pi/2$. Finalmente, se agrega un último bloque para estimar numéricamente la derivada de las variables de pose. Los valores de posiciones lineales, ángulos y derivadas, son transmitidos a otras partes del programa a través de bloques *goto*.

B.1. Funcion TrackableLocation

El primer bloque de la Figura B.1, de izquierda a derecha, es una función de usuario con el nombre *TrackableLocation*. Su función es obtener la pose de Amigobot mediante las bibliotecas de Motive inicializadas con el programa del Apéndice A.

```
1 function [x,y,z,roll,pitch,yaw,Current_Time_Stamp]= TrackableLocation(Trackable_index)
2
3     coder.extrinsic('calllib') % Define funcion de MATLAB en Simulink
```

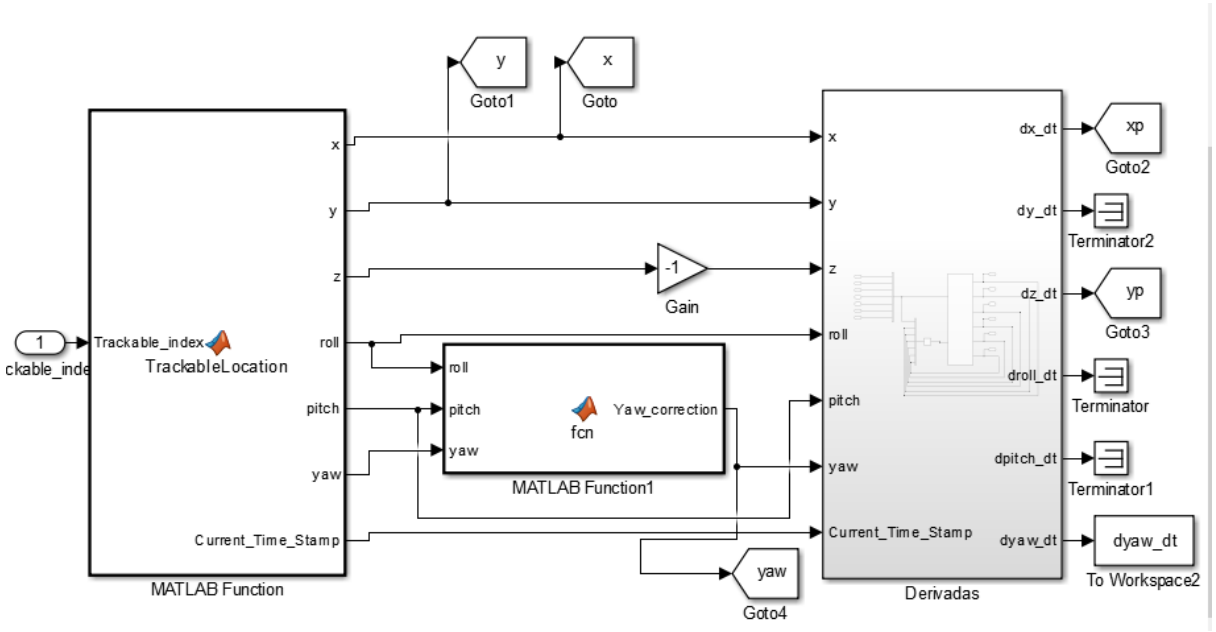


Figura B.1: Subsistema: Trackable Location.

```
4
5       % Actualiza el frame
6       calllib('NPTrackingTools', 'TT_Update'); % Actualiza los datos
7
8       % Define los tipos de salida de la funcion TT_TrackableLocation
9       % Se necesita que sea de tipo de datos unico para la funcion
10      X=single(0); Y=single(0); Z=single(0);
11      YAW=single(0); PITCH=single(0); ROLL=single(0);
12      qx=single(0); qy=single(0); qz=single(0); qw=single(0);
13      Current_Time=double(0);
14
15      % Encuentra los componentes de pose del objeto a rastear
16      [X,Y,Z,qx,qy,qz,qw,YAW,PITCH,ROLL] = calllib('NPTrackingTools',
        'TT_RigidBodyLocation',(Trackable_index-1),X,Y,Z,qx,qy,qz,qw,YAW,PITCH,ROLL);
17
18      % Obtiene la marca de tiempo
19      Current_Time = calllib('NPTrackingTools', 'TT_FrameTimeStamp');
20
21      % Calcula la diferencia de tiempo entre las capturas de dato
22
23      % Simulink acepta solo tipo de datos double
24      x=double(X); y=double(Y); z=double(Z);
25      yaw=double(YAW); pitch=double(PITCH); roll=double(ROLL);
26
27      Current_Time_Stamp=double(Current_Time);
28
```

```
29 end
```

B.2. Bloque de corrección de ángulo

Para aplicar estrategias de control, es necesario que el ángulo de orientación esté entre $\pm 180°$. Debido a que el bloque *TrackableLocation* hace una conversión a partir de cuaterniones, el ángulo θ no se entrega en el rango deseado, además que existen discontinuidades cuando el valor absoluto de *pitch* y *roll* supera los 90°. Para superar los inconvenientes mencionados, se agrega esta función en la obtención del ángulo de orientación del robot.

```
1  function Yaw_correction  = fcn(roll,pitch,yaw) % Sirve para corregir el angulo cuando
       pasa por la discontinuidad (cuando pasa del segundo al tercer cuadrante)
2      if abs(roll)>90 && abs(pitch)>90 % Comprueba si los valores absolutos de roll y
       pitch son mayores que 90
3          if yaw>=0
4              Yaw_correction=180-yaw; % Correccion si yaw es mayor o igual a 0
5          else
6              Yaw_correction=-180-yaw; % Correccion si yaw es menor que 0
7          end
8      else
9          Yaw_correction=yaw; % Si no se cumplen las condiciones anteriores, no hay
       correccion
10     end
11 end
```

B.3. Bloque derivadas

Este bloque calcula las derivadas temporales de las señales de posición x, z y de orientación θ. La Figura B.2 muestra el diagrama de bloques de este subsistema. Debido a que no está sincronizado cada valor de actualización de datos de Motive con los instantes de muestreo de ejecución del programa en Simulink, se deben hacer consideraciones en este sub-bloque, para estimar de la mejor manera posible las derivadas, según el método de Euler hacia atrás [28].

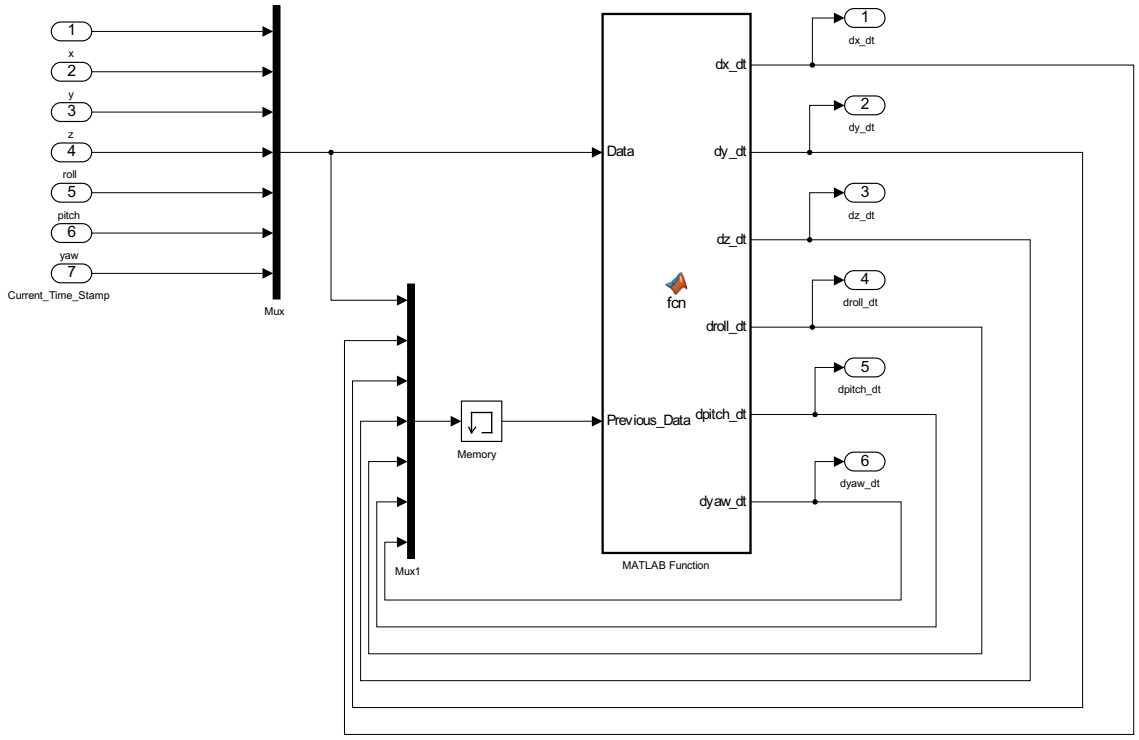


Figura B.2: Subsistema: Derivadas.

El siguiente programa es el encargado de calcular las derivadas.

```
function [dx_dt,dy_dt,dz_dt,droll_dt,dpitch_dt,dyaw_dt] = fcn(Data,Previous_Data)
    % Calcula la diferencia de tiempo entre muestras consecutivas
    Delta_Time = Data(7) - Previous_Data(7);
    % Calcula la norma de la diferencia entre los datos actuales y los anteriores
    norm_delta = norm(Data(1:6) - Previous_Data(1:6));

    % Comprueba si la norma es distinta de cero y si el tiempo transcurrido es
    positivo
    if ~((norm_delta == 0) && (Delta_Time <= 0))
        % Calcula las derivadas temporales de las coordenadas espaciales
        dx_dt = (Data(1) - Previous_Data(1)) / Delta_Time;
        dy_dt = (Data(2) - Previous_Data(2)) / Delta_Time;
        dz_dt = (Data(3) - Previous_Data(3)) / Delta_Time;
        dyaw_dt = (Data(6) - Previous_Data(6)) / Delta_Time;

        % Corrige las derivadas temporales de yaw si supera los limites de -180 a 180
    grados
```

```
16          if ((Data(6) - Previous_Data(6)) > 180)
17              dyaw_dt = ((Data(6) - Previous_Data(6)) - 360) / Delta_Time;
18          elseif ((Data(6) - Previous_Data(6)) < -180)
19              dyaw_dt = ((Data(6) - Previous_Data(6)) + 360) / Delta_Time;
20          end
21
22          % Calcula las derivadas temporales de los angulos de orientacion
23          dpitch_dt = (Data(5) - Previous_Data(5)) / Delta_Time;
24          droll_dt = (Data(4) - Previous_Data(4)) / Delta_Time;
25      else
26          % Si la norma es cero o el tiempo transcurrido es negativo, se usan los
        valores anteriores
27          dx_dt = Previous_Data(8);
28          dy_dt = Previous_Data(9);
29          dz_dt = Previous_Data(10);
30          dyaw_dt = Previous_Data(13);
31          dpitch_dt = Previous_Data(12);
32          droll_dt = Previous_Data(11);
33      end
34  end
```

Apéndice C

Programa de generación de campos potenciales

El siguiente programa es el encargado de generar la referencia para ser empleada en el controlador, se basa en la teoría de campos potenciales de velocidad revisada en la Sección 4.2.

```
1  function [Vx, Vy, Vxp, Vyp, ex, ey] = vel_field_WMR(x, y, xp, yp, xc, yc, rc, gama,
       Kv, x_obs, y_obs, r_obs)
2  %--------------------------------------------------------------------
3  % Entradas:  x, y, xp, yp, xc, yc, rc
4  % Salidas: [xpd, ypd, xppd, yppd]
5  %    g = 9.81;
6  %    x = in(1);    % Posicion actual del vehiculo en x
7  %    y = in(2);    % Posicion actual del vehiculo en y
8  %    xp = in(3);   % Posicion actual del vehiculo en x
9  %    yp = in(4);   % Posicion actual del vehiculo en y
10 %    xc = in(5);
11 %    yc = in(6);
12 %    rc = in(7);
13
14    hp = [xp; yp];
15
16
17 %--------------------------------------------------------------------------
18    % Calculo de teta_cl := qcl y sus derivadas
19    xt = x - xc;
20    yt = y - yc;
21    ht = [xt; yt];
```

```
22      qcl = atan2(yt, xt); % Theta_closer
23      qp_cl = cruz(ht, hp) / (ht' * ht);% (xt * yp - yt * xp) / d_cl^2;
24
25      % Calculo de derivadas de Vdes
26      [x_cl, y_cl] = TrayecCirculo(qcl, xc, yc, rc);
27      ex = x_cl - x;
28      ey = y_cl - y;
29      he = [ex; ey];
30      Vap = he / norm(he);        % Vap
31
32      Vxc = -rc * sin(qcl);
33      Vyc = rc * cos(qcl);
34      Vc = [Vxc; Vyc];
35      dist_Vc = norm(Vc);
36      Vtg = Vc / dist_Vc;         % Vtg
37
38      % V
39      gg = -gama * norm(he);
40      f = exp(gg);
41      F1 = (2 / (1 + f)) - 1;
42      F2 = 1 - F1;
43      Vnum = F1 * Vap + F2 * Vtg;
44      Vvf = Vnum / norm(Vnum);
45  %     norm(Vnum)
46
47      % Primera derivada -----1-----------1---------------1------------1------
48      xp_cl = qp_cl * Vxc;
49      yp_cl = qp_cl * Vyc;
50      e_xp = xp_cl - xp;
51      e_yp = yp_cl - yp;
52      hep = [e_xp; e_yp];
53      Vp_ap = Vp(Vap, he, hep);
54
55      Vp_xc = -qp_cl * Vyc; % -10*qp_cl * cos(qcl);
56      Vp_yc = qp_cl * Vxc;
57      Vp_c = [Vp_xc; Vp_yc];
58      Vp_tg = Vp(Vtg, Vc, Vp_c); % (Vp_c - (Vtg' * Vp_c) * Vtg) / dist_Vc;
59
60      % Vp
61      gp = -gama * (Vap' * hep);
62      fp = f * gp;
63      F1p = -2 * fp / (1 + f)^2;
64      F2p = -F1p;
65
```

```
66      Vp_num = F1 * Vp_ap + F2 * Vp_tg + F1p * Vap + F2p * Vtg;
67      Vp_vf = Vp(Vvf, Vnum, Vp_num);
68
69      Vxa = Vvf(1);
70      Vya = Vvf(2);
71      Vxpa = Vp_vf(1);
72      Vypa = Vp_vf(2);
73
74      in = [x; y; xp; yp; Vxa; Vya; Vxpa; Vypa; x_obs; y_obs; r_obs];
75      out = vel_ev(in);
76      Vx = Kv * out(1);
77      Vy = Kv * out(2);
78      Vxp = Kv * out(3);
79      Vyp = Kv * out(4);
80  end
81
82  function [x, y] = TrayecCirculo(q, xc, yc, rc)
83      x = xc + rc * cos(q);
84      y = yc + rc * sin(q);
85  end
86
87  function [out] = cruz(a, b)
88      % Producto cruzado
89      R = [0 1; -1 0];
90      out = a' * R * b;
91  end
92
93  function [out] = Vp(V, v1, v1p)
94      out = (v1p - (V' * v1p) * V) / norm(v1);
95  end
```

Apéndice D

Programa de control

El siguiente programa corresponde al bloque de control y es el encargado de generar las señales de velocidad lineal y angular aplicadas al robot.

```
% Calcula la velocidad lineal del robot
v = sqrt(Vx^2 + Vy^2);

% Calcula el angulo deseado de orientacion
teta_d = atan2(Vy, Vx);

% Convierte el angulo actual de yaw a radianes
teta = yaw * pi / 180;

% Calcula el error de orientacion
e_teta = teta_d - teta;

% Corrige el error de orientacion si esta fuera del rango [-pi, pi]
if (e_teta < -pi)
    e_teta = 2 * pi + e_teta;
elseif (e_teta > pi)
    e_teta = -2 * pi + e_teta;
end
    % Calcula la derivada del angulo de orientacion deseado

if (v == 0)
    teta_dp = atan2(Vyp, Vxp);
else
    teta_dp = (Vxp * Vy - Vyp * Vx) / v^2;
end

```

```
27      % Calcula la velocidad angular de control utilizando un controlador PI
28      w = kp * e_teta + teta_dp + ki * int_e;
29  end
```

Bibliografía

[1] ActiveMedia ROBOTICS, *Amigo User Guide*, 2003. Available: `https://www.generationrobots.com/media/AmigoGuide.pdf`.

[2] H. E. Espitia Cuchango and J. I. Sofrony Esmeral, "Algoritmo para planear trayectorias de robots móviles, empleando campos potenciales y enjambres de partículas activas brownianas," *Ciencia e Ingeniería Neogranadina*, vol. 22, no. 2, pp. 75–96, 2012.

[3] Z. Chong, B. Qin, T. Bandyopadhyay, T. Wongpiromsarn, E. Rankin, M. Ang, E. Frazzoli, D. Rus, D. Hsu, and K. Low, "Autonomous personal vehicle for the first-and last-mile transportation services," in *2011 IEEE 5th International Conference on Cybernetics and Intelligent Systems (CIS)*, pp. 253–260, IEEE, 2011.

[4] V. Engesser, E. Rombaut, L. Vanhaverbeke, and P. Lebeau, "Autonomous delivery solutions for last-mile logistics operations: A literature review and research agenda," *Sustainability*, vol. 15, no. 3, p. 2774, 2023.

[5] F. Inostroza, I. Parra-Tsunekawa, and J. Ruiz-del Solar, "Robust localization for underground mining vehicles: An application in a room and pillar mine," *Sensors*, vol. 23, no. 19, p. 8059, 2023.

[6] "El rover perseverance de la NASA 'al volante' - NASA ciencia." `https://ciencia.nasa.gov/sistema-solar/el-mars-rover-perseverance-de-la-nasa-al-volante/`. Accessed: 2024-02-09.

[7] L. García-Delgado, J. Noriega, D. Berman-Mendoza, A. L. Leal-Cruz, A. Vera-Marquina, R. Gómez-Fuentes, A. García-Juárez, A. Rojas-Hernández, and I. E. Zaldívar-Huerta, "Repulsive function in potential field based control with algorithm for safer avoidance," *Journal of Intelligent & Robotic Systems*, vol. 80, pp. 59–70, 2015.

[8] S. A. H. Mohsan, N. Q. H. Othman, Y. Li, M. H. Alsharif, and M. A. Khan, "Unmanned aerial vehicles (UAVs): Practical aspects, applications, open challenges, security issues, and future trends," *Intelligent Service Robotics*, vol. 16, no. 1, pp. 109–137, 2023.

[9] J. Rennie, "Drone types: Multi-rotor, fixed-wing, single rotor, hybrid VTOL." `https://www.auav.com.au/articles/drone-types/#4`, 2024. Visitado: 2024-02-24.

[10] J. R. Sánchez-Ibáñez, C. J. Pérez-del Pulgar, and A. García-Cerezo, "Path planning for autonomous mobile robots: A review," *Sensors*, vol. 21, no. 23, p. 7898, 2021.

[11] C. Stachniss, J. J. Leonard, and S. Thrun, "Simultaneous localization and mapping," in *Springer Handbook of Robotics*, pp. 1153–1176, Springer, 2016.

[12] A. A. A. Rasheed, M. N. Abdullah, and A. S. Al-Araji, "A review of multi-agent mobile robot systems applications.," *International Journal of Electrical & Computer Engineering (2088-8708)*, vol. 12, no. 4, 2022.

[13] "Visión por computadora - Libro online de IAAR." `https://iaarbook.github.io/vision-por-computadora/`. Visitado: 2024-02-24.

[14] S. Fernández, C. Javier, and V. S. Consuegra, "Reconocimiento óptico de caracteres (OCR)," *Univ. Carlo*, vol. 3, no. 7, p. 2008, 2008.

[15] J. Sánchez Diez, *Control de calidad del acabado superficial de bobinas de alambrón mediante inspección robotizada asistida por visión artificial.* PhD thesis, Escuela Técnica Superior de Ingenieros Industriales y de Telecomunicación, Jan 2016.

[16] M. Menolotto, D.-S. Komaris, S. Tedesco, B. O'Flynn, and M. Walsh, "Motion capture technology in industrial applications: A systematic review," *Sensors*, vol. 20, no. 19, p. 5687, 2020.

[17] C. R. Viala and A. J. S. Salmerón, "Procedimiento completo para el calibrado de cámaras utilizando una plantilla plana," *Revista Iberoamericana de Automática e Informática Industrial RIAI*, vol. 5, no. 1, pp. 93–101, 2008.

[18] Lantronix, Inc., *WIBOX x 2100E Device Server User Guide.* Lantronix, Inc., December 2017. Document Revision: December 2017.

[19] W. E. Dixon, D. M. Dawson, E. Zergeroglu, and A. Behal, *Nonlinear control of wheeled mobile robots*, vol. 175. Springer, 2001.

[20] P. Lin, J. H. Yang, Y. S. Quan, and C. C. Chung, "Potential field-based path planning for emergency collision avoidance with a clothoid curve in waypoint tracking," *Asian Journal of Control*, vol. 24, no. 3, pp. 1074–1087, 2022.

[21] S. H. A. Wahab, A. Saudi, N. Saad, and A. Chekia, "UAV path planning using rotated TOR in structured environment," in *2022 IEEE International Conference on Artificial Intelligence in Engineering and Technology (IICAIET)*, pp. 1–6, September 2022.

[22] S. Ionita, "Autonomous vehicles: from paradigms to technology," in *IOP Conference Series: Materials Science and Engineering*, vol. 252, p. 012098, IOP Publishing, 2017.

[23] X. Chen, Z. Huang, Y. Sun, Y. Zhong, R. Gu, and L. Bai, "Online on-road motion planning based on hybrid potential field model for car-like robot," *Journal of Intelligent & Robotic Systems*, vol. 105, no. 1, p. 7, 2022.

[24] P. Vadakkepat, K. C. Tan, and W. Ming-Liang, "Evolutionary artificial potential fields and their application in real-time robot path planning," in *Proceedings of the 2000 Congress on Evolutionary Computation, CEC00*, vol. 1, pp. 256–263, July 2000.

[25] R. D. Puriyanto, O. Wahyunggoro, and A. I. Cahyadi, "Implementation of improved artificial potential field path planning algorithm in differential drive mobile robot," in *14th International Conference on Information Technology and Electrical Engineering (ICITEE)*, pp. 18–23, October 2022.

[26] C. Pérez-D'Arpino, W. Medina-Meléndez, L. Fermín, J. Guzmán, G. Fernández-López, and J. C. Grieco, "Dynamic velocity field angle generation for obstacle avoidance in mobile robots using hydrodynamics," in *Ibero-American Conference on Artificial Intelligence*, pp. 372–381, Springer, 2008.

[27] O. Hirshfeld, "Natural point motive (optitrack) api interface to matlab and simulink 64bit." `https://github.com/orhirshfeld/Motive-API-interface-to-Matlab-and-simulink-in-Real-time`, 2023. GitHub. Retrieved December 20, 2023.

[28] K. Ogata, *Sistemas de Control en Tiempo Discreto*. Prentice Hall, 2 ed., 2000.