Universidad de la República
Facultad de Ingeniería

Robot Autónomo Móvil Terrestre con Énfasis en SLAM y Fusión Sensorial

Memoria de proyecto presentada a la Facultad de Ingeniería de la Universidad de la República por

Santiago Bernheim, Agustín Costa, Andrés De Luca

en cumplimiento parcial de los requerimientos para la obtención del título de Ingeniero Electricista.

Tutor

Rafael Canetti ..... Universidad de la República

Tribunal

Pedro Arzuaga ..... Universidad de la República
Rafael Canetti ..... Universidad de la República
Julio Pérez ..... Universidad de la República

Montevideo
martes 23 julio, 2019

Robot Autónomo Móvil Terrestre con Énfasis en SLAM y Fusión Sensorial, Santiago Bernheim, Agustín Costa, Andrés De Luca.

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 193 páginas.
Compilada el martes 23 julio, 2019.
http://iie.fing.edu.uy/

Agradecimientos

Agradecemos principalmente a Mariana, Martina, nuestros familiares y amigos por el apoyo incondicional, no solo a lo largo de este proyecto, sino durante toda la carrera, por su paciencia y cariño en todo momento.

Queremos hacer especial mención a Mauricio González, Roberto Rodríguez y Martha Delgado, que fueron claves en distintos puntos del desarrollo del proyecto.

Por último a nuestro Tutor, Rafael Canetti, que nos guió por este largo proceso y nos dio la oportunidad de llevar a cabo este proyecto.

Esta página ha sido intencionalmente dejada en blanco.

Resumen

El objetivo de este proyecto es la construcción de un robot autónomo móvil terrestre y la implementación de un sistema para el mapeo y navegación de entornos desconocidos para el robot. Dicho sistema consiste de dos componentes: el robot construido y una computadora externa a él. El usuario encomienda misiones que son recibidas por la computadora, generando mapas y planeando trayectorias acorde a lo indicado, mientras que el robot obedece los comandos generados por la computadora y está equipado con sensores que permiten caracterizar el entorno y evadir obstáculos.

El robot es un carro de cuatro ruedas omnidireccionales equipado con sensores y actuadores para interactuar con su entorno. Los sensores encargados de percibir el entorno son un LiDAR y cuatro sensores de ultrasonido. Por otra parte, para determinar la localización del robot se utilizaron encoders en cada una de las ruedas y una unidad de navegación inercial. Como actuadores se utilizaron cuatro motores, uno por cada rueda, para controlar el desplazamiento del robot.

La computadora se comunica con el robot a través de WiFi. Tanto el robot como la computadora corren sobre la plataforma ROS (Robot Operating System) los algoritmos de Fusión Sensorial, SLAM (Localización y Mapeo Simultáneos), y Planeación y Seguimiento de Trayectorias. En todo momento el usuario puede visualizar la ubicación del robot, el mapa construido y las trayectorias planeadas a través de la interfaz implementada en la computadora. Por último, al terminar la misión se pueden obtener los mapas generados por el robot para su futura utilización.

Esta página ha sido intencionalmente dejada en blanco.

Abstract

The target of this project is the construction of an autonomous mobile terrestrial robot and the implementation of a mapping and navigation system for unknown environments. Said system consists of two components: the robot and an external computer. The user defines missions which are received by the computer, generating maps and planning trajectories according to its indications, while the robot obeys the commands generated by the computer, and is equipped with sensors that allow to characterize the environment and avoid obstacles.

The robot is a four omnidirectional wheeled cart equipped with sensors and actuators to interact with the environment. The sensors in charge of perceiving the environment are a LiDAR and four ultrasonic sensors. On the other hand, in order to determine the location of the robot rotary encoders in each of the wheels and an inertial movement unit were used. Four motors were used as actuators, one for each wheel, to control the movement of the robot.

The computer communicates with the robot through WiFi. The robot, as well as the computer, run the Sensor Fusion, SLAM (Simultaneous Location and Mapping), and Path Planning algorithms on the ROS (Robot Operating System) platform. The user can visualize the location of the robot, the map that is being generated and the planned trajectories through the interface on the computer at all times. Lastly, when the mission has been finalized, the map generated by the robot can be obtained for future use.

Esta página ha sido intencionalmente dejada en blanco.

Tabla de contenidos

Agradecimientos i

Resumen iii

Abstract v

1. Introducción 1
- 1.1. Antecedentes . . . 2
- 1.2. Algunos ejemplos de uso . . . 3
- 1.3. Organización del documento . . . 5

2. Resumen del Sistema Implementado 7
- 2.1. Objetivo . . . 7
- 2.2. Aplicaciones . . . 7
- 2.3. Solución adoptada . . . 9
- 2.4. Arquitectura mecánica . . . 10
  - 2.4.1. Chasis . . . 11
  - 2.4.2. Primer nivel . . . 12
  - 2.4.3. Segundo nivel . . . 13
  - 2.4.4. Tercer nivel . . . 14
- 2.5. Movimiento . . . 14
- 2.6. Arquitectura eléctrica . . . 15
- 2.7. Descripción funcional . . . 20
  - 2.7.1. Fusión sensorial . . . 20
  - 2.7.2. Mapeo y Localización Simultáneos(SLAM) . . . 21
  - 2.7.3. Planeación de trayectorias y exploración del espacio . . . 21
- 2.8. Arquitectura lógica . . . 22
  - 2.8.1. Manejo de sensores . . . 22
  - 2.8.2. Control de motores . . . 22
  - 2.8.3. Localización . . . 25
  - 2.8.4. Manejo del LiDAR . . . 25
  - 2.8.5. Mapeo (SLAM) . . . 25
  - 2.8.6. Navegación: Planeación y Seguimiento de Trayectorias . . . 25
  - 2.8.7. Exploración . . . 25
  - 2.8.8. Interfaz de usuario . . . 26
  - 2.8.9. Comunicación entre módulos . . . 26

Tabla de contenidos

3. Componentes de Hardware **27**
3.1. Introducción . . . 27
3.2. Resumen de Componentes . . . 27
3.3. Conexión eléctrica . . . 28
3.4. Componentes Básicos . . . 32
3.4.1. Ruedas . . . 32
3.4.2. Motores . . . 32
3.4.3. Controladores . . . 36
3.4.4. Alimentación . . . 38
3.4.5. Instrumentación . . . 40
3.5. Inteligencia . . . 44
3.5.1. Teensy 3.5 . . . 44
3.5.2. Raspberry Pi 3 B+ . . . 45

4. Modelado y Caracterización **47**
4.1. Introducción . . . 47
4.2. Modelado . . . 47
4.2.1. Definición de sistemas de coordenadas del robot: . . . 47
4.2.2. Holonomía . . . 49
4.3. Cinemática: . . . 50
4.3.1. Introducción: . . . 50
4.3.2. Ecuaciones: . . . 50
4.4. Dinámica: . . . 55
4.5. Ecuaciones de movimiento: . . . 58

5. Algoritmos de Control, Localización, Mapeo y Navegación **61**
5.1. Control de Velocidad de las Ruedas . . . 61
5.1.1. Introducción . . . 61
5.1.2. Solución adoptada . . . 62
5.1.3. Modelado del Sistema . . . 63
5.1.4. Configuración del controlador . . . 66
5.1.5. Controlador y configuración utilizados . . . 70
5.2. Fusión Sensorial . . . 71
5.2.1. Introducción: . . . 71
5.2.2. Teoría estadística: . . . 73
5.2.3. Filtro de Kalman . . . 74
5.2.4. Extensión a sistemas no lineales: . . . 77
5.2.5. Diseño del algoritmo: . . . 78
5.2.6. Simulación y resultados: . . . 80
5.2.7. Implementación en el robot: . . . 83
5.3. Mapeo y Localización en Simultáneo . . . 84
5.3.1. Introducción . . . 84
5.3.2. Tipos de Mapas . . . 84
5.3.3. Nomenclatura a utilizar . . . 85
5.3.4. Definición formal del problema . . . 86
5.3.5. Algoritmo utilizado: Gmapping . . . 87

5.4. Planeación y Seguimiento de Trayectorias 91
5.4.1. Marco teórico 91
5.4.2. Global Planner : *Dijkstra* 96
5.4.3. Local Planner: Dynamic Window Approach 97
5.4.4. Mapas de Costo 100
5.4.5. Implementación: move_base 101
5.4.6. Exploración del espacio 102
5.5. Implementación 103

6. Implementación en Software **105**
6.1. ROS 105
6.1.1. Introducción 105
6.1.2. Estructura 106
6.1.3. Filosofía de desarrollo 107
6.2. Solución de software 108
6.2.1. Resumen 108
6.2.2. Interacción con el Entorno 108
6.2.3. Localización 111
6.2.4. Mapeo 111
6.2.5. Planeación de trayectorias 113
6.2.6. Exploración 119
6.2.7. Interfaz 119
6.2.8. Selección de misiones 122

7. Análisis y Validación de Funcionamiento **125**
7.1. Motores 125
7.1.1. Respuesta a la Rampa 125
7.1.2. Respuesta al Escalón 130
7.2. PID 133
7.2.1. Motivación 133
7.2.2. Implementación 133
7.2.3. Procesamiento de los Datos 133
7.3. Prueba de deslizamiento de ruedas 135
7.3.1. Motivación 135
7.3.2. Implementación 135
7.3.3. Resultados 136
7.4. Localización en el mapa 139
7.4.1. Motivación 139
7.4.2. Implementación 139
7.4.3. Procesamiento de Datos 139
7.5. Comparación de mapas 141
7.5.1. Motivación 141
7.5.2. Implementación 141
7.5.3. Resultados 141
7.5.4. Segundo mapa 143

Tabla de contenidos

8. Conclusiones **147**
8.1. Conclusiones generales 147
8.2. Trabajo futuro 148

9. Anexo **149**
9.1. Manual de usuario 149
9.1.1. Introducción 149
9.1.2. Instalación y configuración de máquina virtual 149
9.1.3. Inicialización 150
9.1.4. Interfaz 152
9.1.5. Selección de modo de funcionamiento 152
9.1.6. Descripción de modos de funcionamiento 153
9.1.7. Configuraciones 156
9.1.8. Links de interés 158
9.2. Guía del programador 158
9.2.1. rpi_localized_config.launch 159
9.2.2. manual_mapping.launch 161
9.2.3. autonomous_movement.launch 161
9.2.4. exploration.launch 162
9.3. Experimentos adicionales 164
9.3.1. Localización 164
9.3.2. Limitaciones de navegación autónoma 165
9.3.3. Prueba de mapeo autónomo 167

Índice de Tablas **175**

Índice de Figuras **176**

Capítulo 1

Introducción

Autonomía es una palabra proveniente del griego, formada por autos (propio) y nomos (ley o regla), es la capacidad de algo de regirse por sus propias reglas u órdenes.

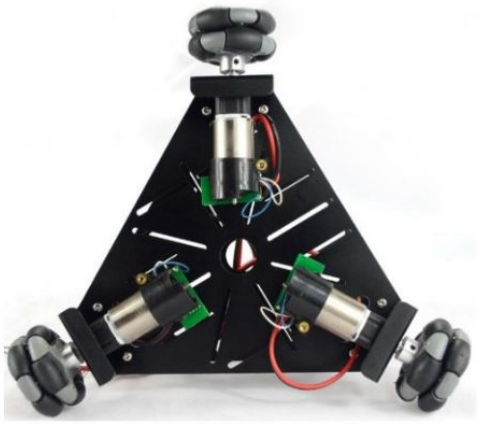
La robótica autónoma desarrolla robots capaces de ejecutar tareas con cierto grado de independencia. En general estos robots son móviles y se desplazan en el ambiente sin (o con poca) asistencia humana, pudiendo ser el ambiente un medio terrestre, aéreo o acuático.

Dentro del espectro de los robots autónomos móviles terrestres existen alguna de las siguientes variantes que se pueden ver en las Figuras [1] 1.0.1a,1.0.1b,1.0.1c y 1.0.1d.

A su vez los robots terrestres pueden dividirse en dos grandes grupos, robots para interiores y para exteriores. La situación en que se encuentre influirá por ejemplo sobre el diseño mecánico del robot y la naturaleza de los sensores utilizados.

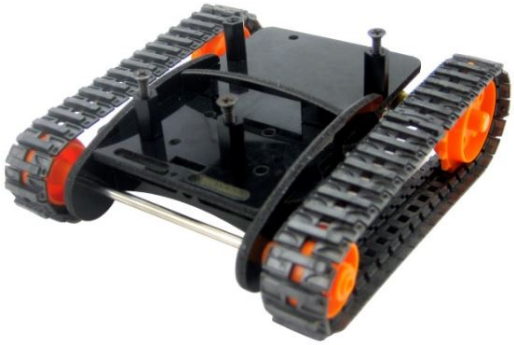
[1] Imágenes extraídas de productos disponibles en robotshop.com

(a) Robot con dos ruedas y un apoyo



(b) Robot con tres ruedas

(c) Robot con cuatro ruedas



(d) Robot con oruga

Figura 1.0.1: Ejemplos de Robots Terrestres Autónomos

1.1. Antecedentes

La historia de los carros autónomos comienza en el Siglo XV con Leonardo Da Vinci, creador de un carro autopropulsado al cual era posible definirle su trayectoria de antemano. Sin embargo no fue hasta el Siglo XX, una vez que los autos convencionales eran una realidad, que varias compañías comenzaron a trabajar en la fabricación de vehículos autónomos. Sobre finales del Siglo XX comenzaron a sentarse las bases matemáticas de lo que posteriormente se denominaría localización y mapeo simultáneo (SLAM), fundamental para el desarrollo de carros autónomos como el de este proyecto. Sin embargo fue recién entrado el Siglo XXI que se desarrollaron las primeras librerías de SLAM.

En cuanto a la Universidad de la República refiere, el Instituto de Ingeniería Eléctrica viene siguiendo hace unos años una línea de trabajo enfocada en proyectos de robótica móvil. Se desarrollaron robots acuáticos como Pez Robot I y II, aéreos

como UQUAD y Termodron, y terrestres como SULLA.

1.2. Algunos ejemplos de uso

1.2.0.1. Roomba

Roomba es una línea de aspiradoras autónomas fabricadas por iRobot. La Roomba 980 es una aspiradora autónoma ilustrada en la Figura 1.2.1, pero que a diferencia de otros productos similares, es capaz de hacer SLAM mientras realiza su trabajo. Para hacer SLAM [2] utiliza una técnica llamada VSLAM o Visual SLAM a partir del uso de una cámara en su parte superior.

La incorporación de SLAM revolucionó el funcionamiento de la Roomba ya que a partir de los mapas realizados y conociendo su ubicación, la aspiradora obtiene una autonomía mucho mayor. Dicho aumento se debe a que es capaz de volver a su lugar de carga cuando tiene poca batería, y una vez finalizada la carga puede regresar sin ayuda al lugar donde había dejado de aspirar para continuar con su tarea.

Figura 1.2.1: Roomba 980

1.2.0.2. TurtleBot

Willow Garage creó un robot de bajo costo y de código abierto con fines educativos y de investigación. Este robot, llamado Turtlebot3, es capaz de llevar a cabo funciones similares al robot construido en este proyecto.

TurtleBot3 se trata de una línea de robots basado en ROS[3] (software también creado por Willow Garage) , capaz de navegar por una habitación haciendo SLAM.

En la Figura 1.2.2 se puede ver el modelo *Waffle Pi*, que cuenta entre sus componentes con:

- LiDAR[4] 360

[2]Simultaneous Location And Mapping
[3]Robot Operating System
[4]Light Detection And Ranging

- Raspberry Pi: Computadora a bordo
- Módulo Bluetooth
- OpenCR: plataforma de desarrollo embebida basada en un microcontrolador Cortex

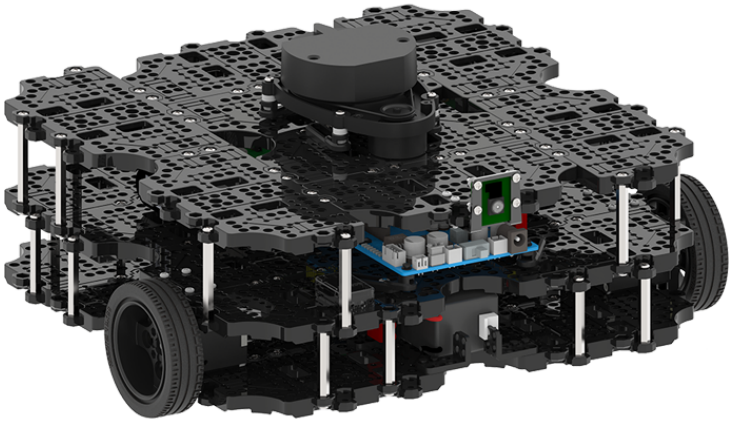


Figura 1.2.2: TurtleBot *Waffle Pi*

1.2.0.3. Robot de Amazon

La empresa estadounidense de ventas en línea Amazon utiliza en algunos de sus depósitos robots como el de la Figura 1.2.3. Estos robots fueron desarrolladas por la empresa Kiva y cumplen con la tarea de transportar los paquetes desde las estanterías hasta el lugar donde son empacados por los empleados. La utilización de robots autónomos permite dismunuir severamente los tiempos de traslado dentro del depósito.

Figura 1.2.3: Robot de Amazon

1.3. Organización del documento

El documento se encuentra dividido en 6 grandes secciones que se describen a continuación:

- **Resumen del Sistema Implementado:** Se brinda una visión general del proyecto, describiendo las principales características del robot y mencionando los algoritmos de navegación utilizados. Se detallan diagramas funcionales y de armado del robot.
- **Componentes de Hardware:** Se describen exhaustivamente los componentes utilizados desde sus características principales hasta su principio de funcionamiento. Se muestra también la conexión entre ellos y el armado.
- **Modelado y Caracterización:** Se explica el proceso de modelado físico matemático del robot. En este proceso se obtienen las ecuaciones de movimiento, que son la base del control del robot.
- **Algoritmos de Control, Localización, Mapeo y Navegación:** En este Capítulo se explica el proceso de creación de un modelo matemático para los motores, y del controlador correspondiente para la velocidad de las ruedas. Además, se describe la teoría detrás de los problemas asociados a la navegabilidad y autonomía del robot, y se describen los algoritmos utilizados como solución a esos problemas.
- **Implementación en Software:** En este Capítulo se documenta todo el desarrollo de Software realizado y se explican las funcionalidades de las librerías utilizadas. Se describe también el sistema operativo ROS, explicando la filosofía de diseño que conlleva y su estructura de funcionamiento.
- **Análisis y Validación de Funcionamiento:** Se explican los experimentos realizados para verificar el correcto funcionamiento del robot, relevar sus características y realizar los ajustes necesarios.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 2

Resumen del Sistema Implementado

2.1. Objetivo

El objetivo de este proyecto es diseñar y construir un robot autónomo móvil terrestre. Específicamente, un carro de cuatro ruedas con capacidad de mapear el entorno en el que se encuentra y poder navegar en él sin intervención de un usuario salvo por la configuración de misiones.

El robot debe ser capaz de completar misiones dado un entorno no estructurado pero confinado, es decir un ambiente cerrado del que no se tienen conocimientos previos. En particular, se puso foco en resolver las siguientes misiones:

- Realizar localización y mapeo simultáneo de forma eficiente, también conocido como el problema del SLAM.
- Navegar en dicho entorno de forma autónoma usando el mapa generado, o bien generando un nuevo mapa.

2.2. Aplicaciones

Una de las motivaciones para realizar este proyecto fue la cantidad de aplicaciones que tiene el producto final. A continuación se describirán algunas de las tantas que existen.

La aplicación que surge como primera opción es el uso de esta plataforma robótica para resolver problemas de logística. Utilizando la capacidad de mapeo y navegación para recorrer depósitos de manera autónoma, mejorando así la eficiencia del proceso. De esta manera, se podría trabajar en una modalidad colaborativa entre operarios humanos y robots.

Otra utilidad directa que surge de las capacidades del robot es su uso para relevar plantas de edificios. Utilizando la funcionalidad de mapeo se puede generar un archivo del mapa del edificio objetivo de manera autónoma y sin supervisión. Entonces, se puede lograr el mapeo en pocos minutos comparado a lo que le puede llevar a un operador idóneo con herramientas tradicionales midiendo las características principales del entorno.

Por último destacamos que la plataforma diseñada y construida en este proyecto puede tener un sinfín de aplicaciones debido a la versatilidad y modularidad del sistema. Esto significa que con cambios menores en el software se puede realizar una implementación similar pero con robots de otras dimensiones y características, pudiendo satisfacer gran variedad de misiones no contempladas en esta Sección.

2.3. Solución adoptada

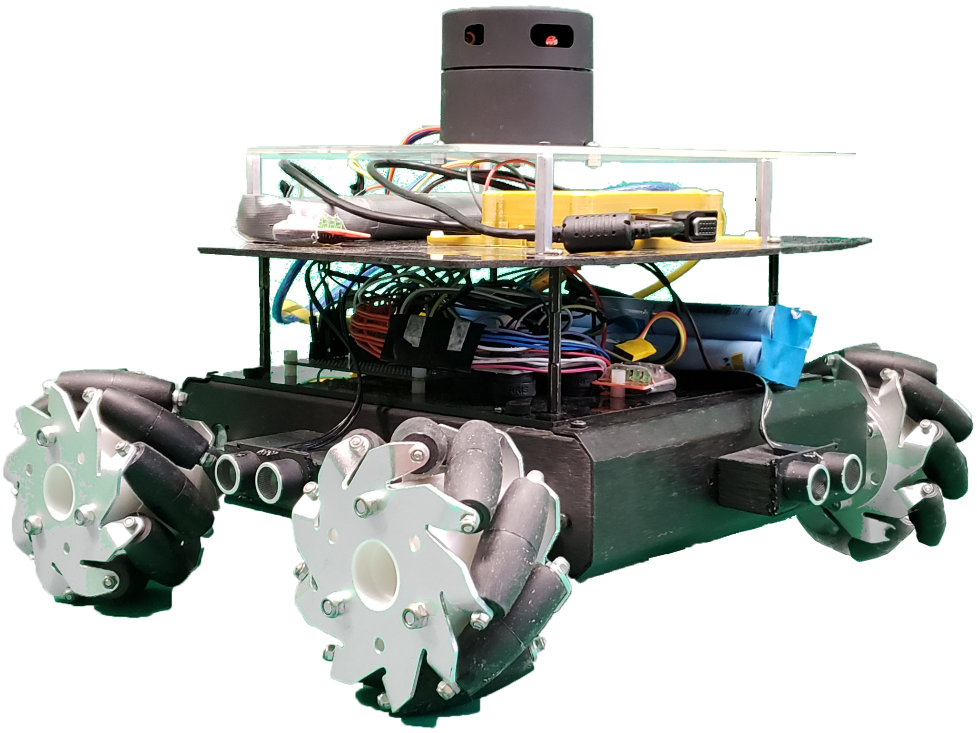


Figura 2.3.1: Rovert: robot construido para el Proyecto

El robot construido consiste de una plataforma móvil terrestre con cuatro ruedas Mecanum, ruedas omnidireccionales que permiten al robot desplazarse en todas las direcciones sin cambiar su orientación. Cada rueda es controlada individualmente por su propio motor, lo que permite lograr estos movimientos. Esto se explicará en detalle en la Sección 2.5.

Este robot cuenta con distintos sensores, basados en distintos principios de funcionamiento cada uno, para percibir el ambiente y la posición, orientación y velocidad del robot en el ambiente.

El procesamiento de información para cumplir con los objetivos es realizado en tres niveles distintos. Dos de ellos se encuentran a bordo del robot, un microcontrolador que maneja los sensores y los motores, y una computadora a bordo que realiza una parte menor del procesamiento. El resto del procesamiento se realiza en la computadora del usuario, conectada por WiFi con la computadora de a bordo.

El robot puede realizar tres funciones principales: el mapeo del ambiente con el robot teledirigido manualmente, mapeo y navegación autónoma a partir de metas fijadas por el usuario, y por último la exploración de un ambiente para completar el mapa del mismo.

Estas misiones las realiza a partir de la implementación de tres algoritmos principales. El primero es un algoritmo de fusión sensorial que permite combinar la información de los sensores para determinar la localización y velocidad del robot en todo momento. El segundo es un algoritmo de mapeo y localización simultánea (SLAM), que se encarga de crear el mapa del ambiente y ubicar al robot dentro del mismo. Por último, el tercer algoritmo es de planeación y seguimiento de trayectorias. Este se basa en la localización del robot y el mapa del ambiente provistos por los dos algoritmos antes mencionados para planificar la trayectoria óptima a una meta definida.

La elección de la misión a completar y otras configuraciones del robot se hacen a través de una interfaz gráfica disponible en la computadora del usuario. Además, esta permite visualizar el mapa creado en tiempo real, la posición del robot y guardar los mapas realizados.

2.4. Arquitectura mecánica

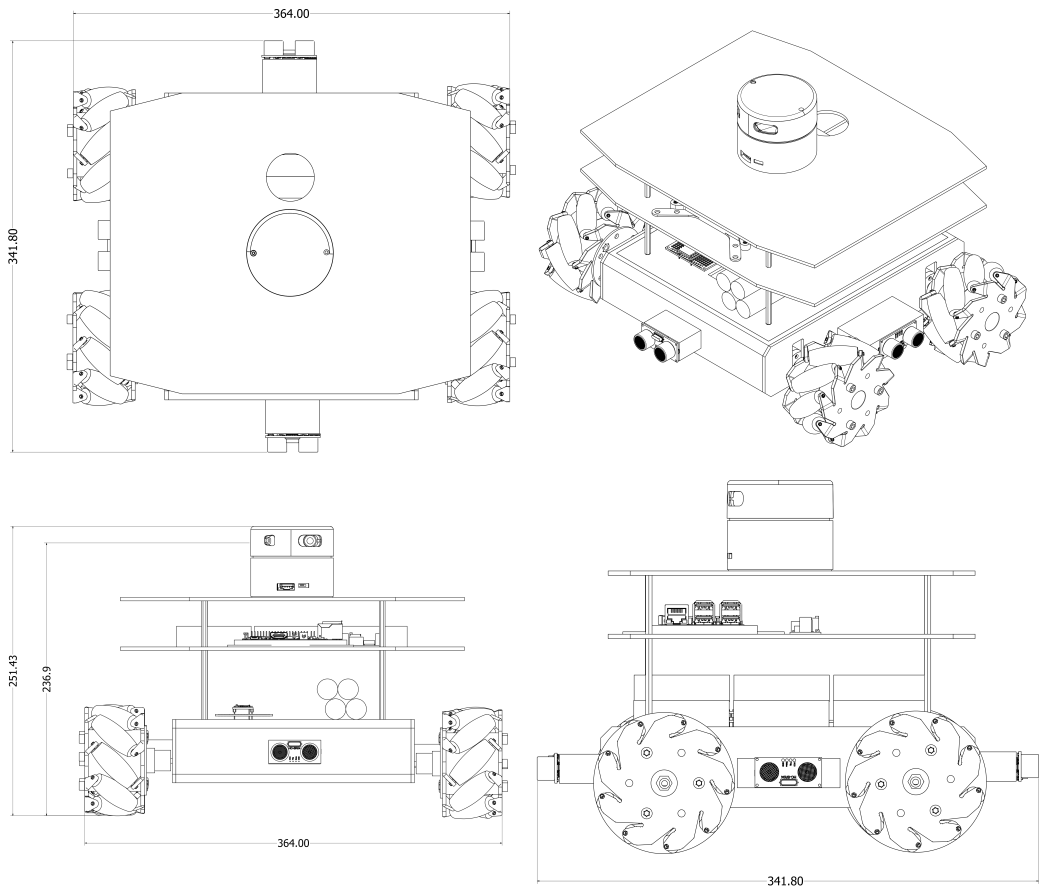


Figura 2.4.1: Diagrama dimensional del robot construido

En esta Sección se describe el vínculo mecánico entre los elementos del robot. A grandes rasgos la estructura del robot consiste en un chasis de aluminio, hueco y cerrado, y tres niveles formados por placas de acrílico. Los niveles están separados unos de los otros por tornillos separadores metálicos.

El elemento distintivo de este robot desde el punto de vista mecánico, es la utilización de ruedas Mecanum, ostensiblemente distintas a las tradicionales ruedas de goma usadas en aplicaciones similares. Las ruedas Mecanum permiten el movimiento omnidireccional del robot. Las peculiaridades del funcionamiento de las ruedas y el movimiento que estas permiten se tratan en las Secciones 3.4.1 y 2.5 respectivamente.

2.4.1. Chasis

El robot construido tiene como base un chasis de aluminio en forma de prisma de base rectangular hueco. A los lados de este chasis se encuentran las cuatro ruedas, dos a cada lado. A su vez, cada rueda es actuada por un motor individual, todos fijados con tornillos al interior del chasis.

La conexión entre las ruedas y los motores se hace a través de adaptadores que transfieren la energía de los motores a las ruedas. Estos se fijan por un lado con tornillos a las ruedas y por el otro con un tornillo prisionero que sujeta el eje del motor, que tienen forma de 'D'. Este acople solamente permite que las ruedas giren en torno al eje del motor, por lo que el direccionamiento no es análogo al de los automóviles comunes, sino que se realiza con combinaciones de giros de las ruedas individuales. Los motores están montados al chasis mediante tornillos, dejando fuera del chasis solo el eje del actuador. Los tornillos se colocan desde el exterior del chasis y tienen su correspondientes orificios roscados sobre la cara lateral del motor, la misma cara en la que se encuentra el eje. El interior del chasis además contiene los controladores de los motores.

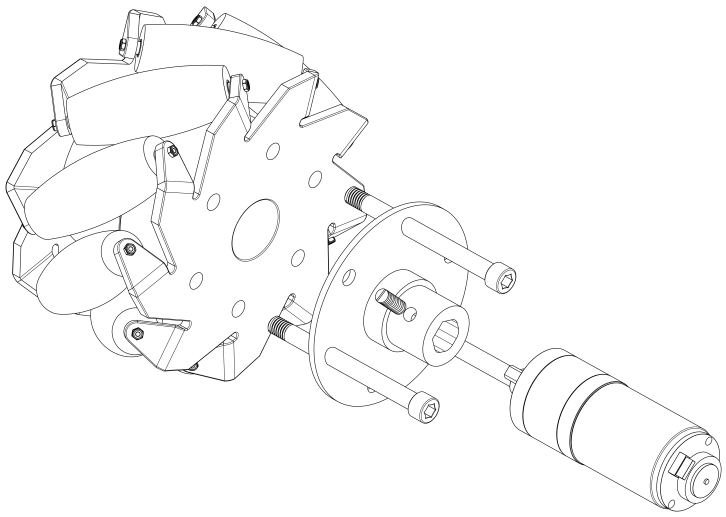


Figura 2.4.2: Diagrama de conexión entre rueda, adaptador y motor

En los cuatro laterales exteriores del chasis, en el punto medio de cada uno, hay sensores de ultrasonido para asistir en la navegación. Estos se encuentran adheridos con silicona a piezas de madera cortadas a medida para separar los sensores del chasis de modo que las ruedas no interfieran con los mismos.

En la Figura 2.4.3 se ilustra este primer nivel con los componentes descritos.

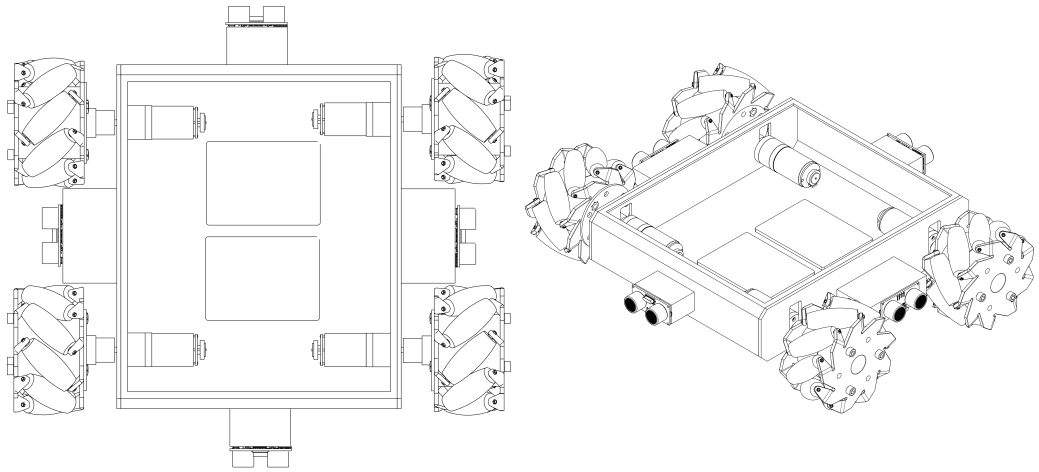


Figura 2.4.3: Diagrama de nivel de chasis

2.4.2. Primer nivel

Sobre la cara superior del chasis se encuentran la IMU[1], el microcontrolador para manejar sensores y motores, y la batería que alimenta los motores. Estos se muestran en el robot en diagrama de la Figura 2.4.4.

La IMU y el microncontrolador se encuentran cada uno en placas de circuito perforadas separadas con todas sus conexiones soldadas. Estas placas se fijan al chasis, en el caso de la IMU a través de amortiguadores de goma para absorber las vibraciones del robot y mitigar errores de medición, y en el caso del microcontrolador Teensy se fija con tornillos separadores de nylon.

Esta cara del chasis cuenta con dos orificios por los que se pasan los cables a los motores y controladores dentro del chasis.

[1]Inertial Movement Unit - Sensor que contiene giróscopo, acelerómetro y magnetómetro

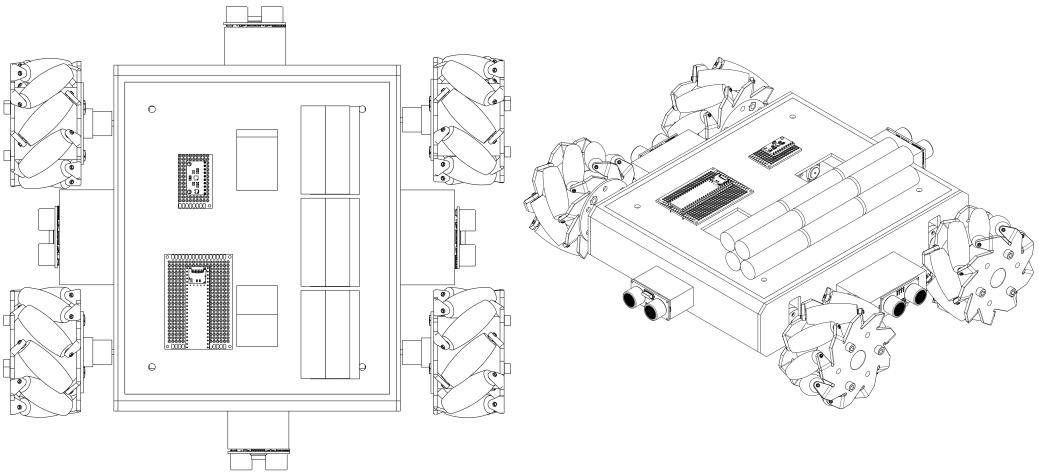


Figura 2.4.4: Diagrama de nivel de chasis

2.4.3. Segundo nivel

El segundo nivel, representado en la Figura 2.4.5, contiene la computadora a bordo, la batería encargada de alimentar la inteligencia y el convertidor DC-DC para regular el voltaje de la batería.

La computadora a bordo, la Raspberry Pi 3B+, se encuentra en una carcasa impresa en 3D en plástico PLA, fijada con tornillos a la placa de acrílico. Junto a esta, el convertidor DC-DC se encuentra también montado con tornillos a la placa de acrílico.

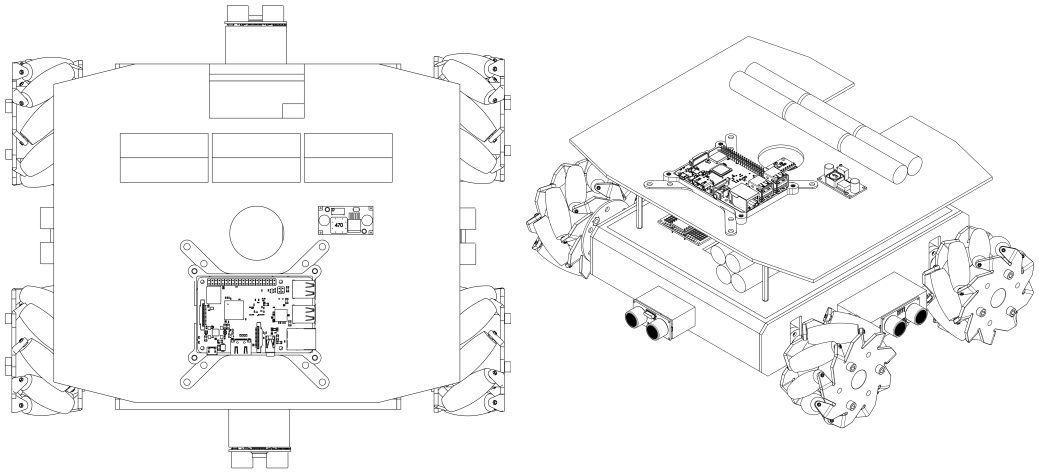


Figura 2.4.5: Diagrama de nivel de chasis

2.4.4. Tercer nivel

El tercer y último nivel solamente contiene al LiDAR, para que este tenga su campo visual ininterrumpido por otros objetos. El mismo cuenta con una placa metálica que le permite ser fijado con tornillos a la placa de acrílico.

La Figura 2.4.1 muestra el robot con todos sus niveles y en particular muestra la altura del sensor del LiDAR, a la que se tomarán las medidas. Los mapas generados serán todos cortes horizontales del ambiente a esta altura relativa al piso.

2.5. Movimiento

El robot se mueve gracias al desplazamiento conjunto de sus ruedas, sin embargo cada una de ellas se mueve de forma independiente para lograr diferentes desplazamientos y giros.

Para girar cada rueda por separado se utiliza un sistema como el de la Figura 2.4.2. A la izquierda se puede ver la rueda y el eje del motor, que se unen como se describió en la Sección 2.4 a través adaptadores por donde se transfiere la energía mecánica.

Cada motor cuenta con un encoder acoplado a su eje, que cuenta las revoluciones que dió cada rueda a partir de las vueltas del eje del motor, como se explica con más detalle en la Sección 3.4.2.3. El motor está alimentado por un controlador y recibe de este una señal que le indica qué velocidad y sentido debe tomar. El encoder está alimentado por el Teensy, al que le proporciona la cuenta actual de vueltas de la rueda. A su vez, la inteligencia a bordo transmite al controlador la velocidad que debe aplicar a cada motor.

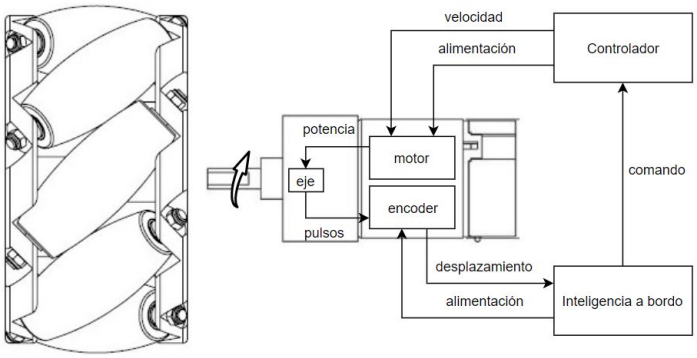


Figura 2.5.1: Diagrama de interacción rueda-robot

El direccionamiento del robot se logra controlando los de sentidos de rotación de las distintas ruedas, lo que permite el desplazamiento en todas las direcciones posibles sin necesidad de rotar el robot. Esto se logra gracias al diseño de las ruedas Mecanum, formadas por rodillos alrededor de la rueda que giran libremente y permiten que las ruedas se desplacen según estos. Estas ruedas se describen en detalle en los Capítulos 3 y 4. En la Figura 2.5.2 se puede observar cómo las distintas

combinaciones de direcciones de giro de las ruedas producen desplazamientos en todas las direcciones y rotaciones según diferentes ejes que se muestran en la Figura. Incluso es posible hacer combinaciones de dos o más movimientos de los descriptos en la Figura 2.5.2 siempre y cuando no cause el deslizamiento de las ruedas respecto al piso.

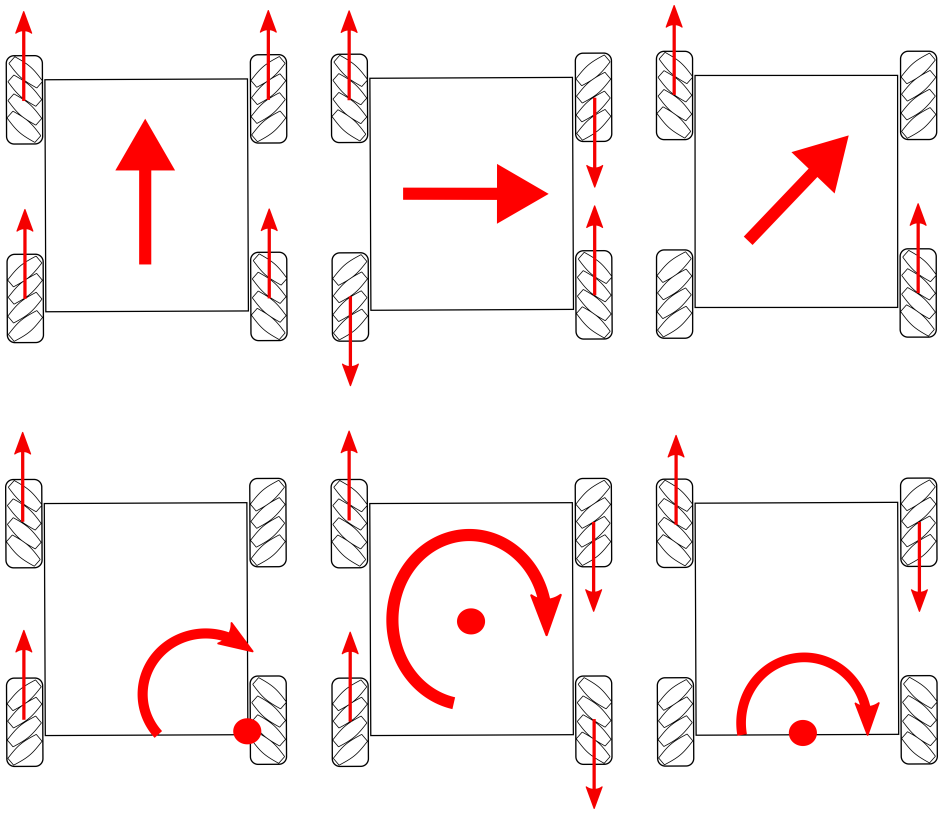


Figura 2.5.2: Diagrama de combinaciones de desplazamiento del robot

2.6. Arquitectura eléctrica

Eléctricamente, el robot consta de dos circuitos distintos, ambos alimentados por distintas baterías, pero compartiendo tierra para mantener un referencial común para la comunicación. Estas baterías serán llamadas: 'batería de motores', a la que alimenta a estos, y 'batería de la inteligencia' a la que alimenta a todo el equipamiento de sensado y procesamiento. En la Figura 2.6.1 se puede ver el diagrama eléctrico del sistema mientras que en la Figura 2.6.2 se puede observar el diagrama de conexionado.

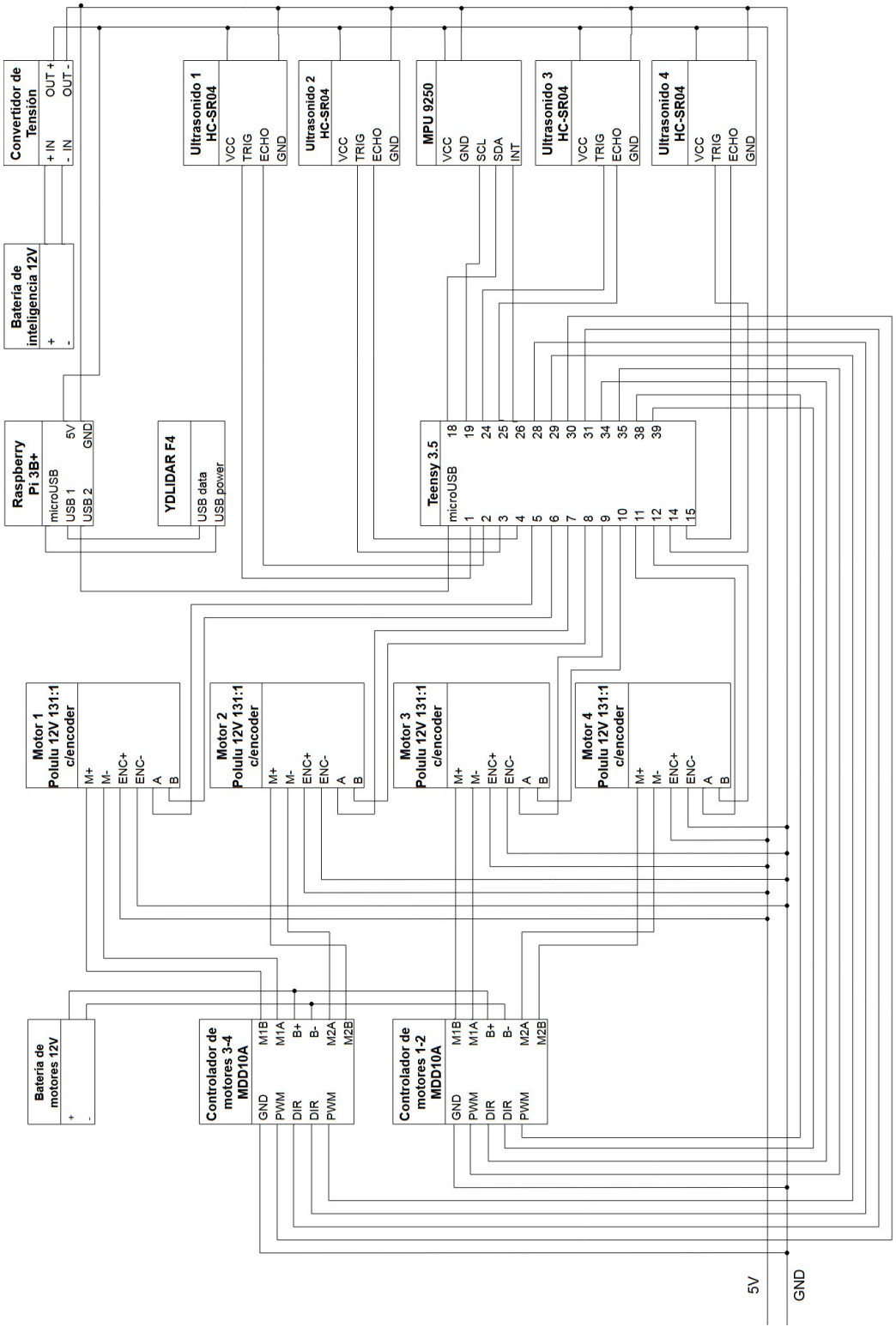


Figura 2.6.1: Diagrama eléctrico del robot

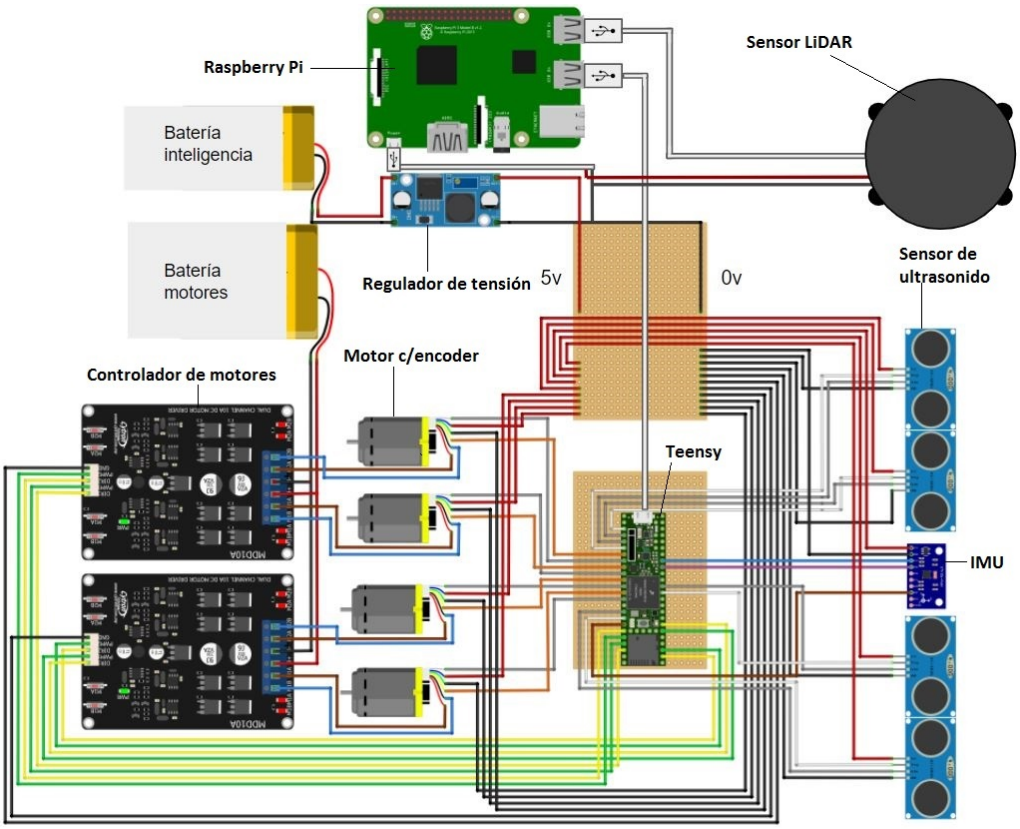


Figura 2.6.2: Diagrama de conexión de componentes

El circuito de la Figura 2.6.3 ilustra la conexiones eléctricas del circuito que alimenta los motores. Se puede observar la conexión entre la batería de motores, la cual alimenta los dos controladores, y cada uno de estos a su vez acciona dos motores.

Los controladores permiten manejar la velocidad y dirección de giro de cada motor individual produciendo una onda cuadrada a la salida que va al motor, cuyo ciclo de trabajo determina la velocidad y su polaridad determina la dirección.

Resulta muy importante la separación de este circuito del circuito de la electrónica, dado que la misma es más sensible al ruido y a la baja de nivel de voltaje. Un comando de velocidad a los motores que implique una aceleración súbita puede generar un pico de corriente en el circuito, que a su vez produzca una baja en la tensión de la batería. Esto, junto con el ruido eléctrico que generan los motores, sustentó la decisión de trabajar con un circuito separado para la inteligencia del robot.

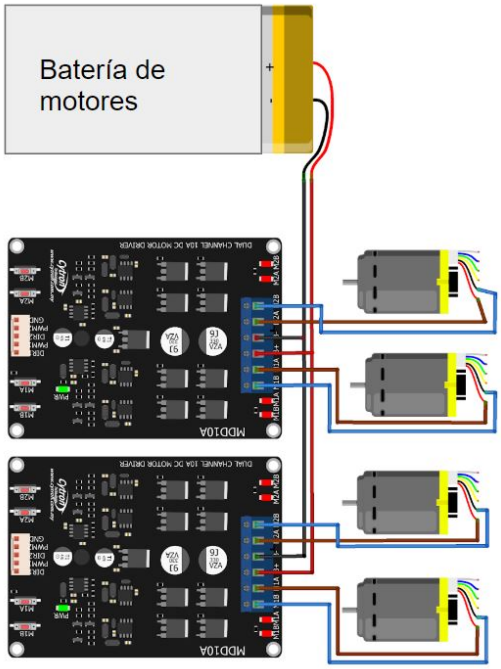


Figura 2.6.3: Diagrama de conexión de motores

Por otro lado, el circuito de la inteligencia, que puede observarse en la Figura 2.6.4, se encarga de alimentar los elementos de sensado y procesamiento del robot, que como se mencionó anteriormente, son sensibles a la baja de voltaje de alimentación. Por esto es que el circuito cuenta con un convertidor DC-DC que se encarga de regular el voltaje de la batería, que normalmente es de 12V aproximadamente, a 5V. Este regulador puede mantener una salida de voltaje estable aunque el voltaje de la batería baje con el uso.

La alimentación de 5V se distribuye a partir de una placa de pruebas con rieles para los terminales positivo y negativo del regulador, de donde se conectaron los encoders, la IMU y los sensores de ultrasonido. En paralelo a esta conexión, también de la salida del regulador, se alimenta directamente al LiDAR y el Raspberry Pi.

El Raspberry Pi a su vez alimenta a través de uno de sus puertos USB al Teensy. Esta conexión a través de las interfaces USB también sirve para transmitir la información de una plataforma a otra a través de un protocolo serial.

Finalmente, las conexiones de información se dividen en las de bajo nivel, que se concentran en el Teensy y las de alto nivel que se concentran en el Raspberry Pi. Las de bajo nivel son las lecturas de los encoders, sensores de ultrasonido e IMU y el comando de motores, todos enviados y recibidos a partir de pines digitales con diversos protocolos. Por otro lado, las de alto nivel son la lectura de los datos del LiDAR y la información de sensores ya analizada y procesada por el Teensy.

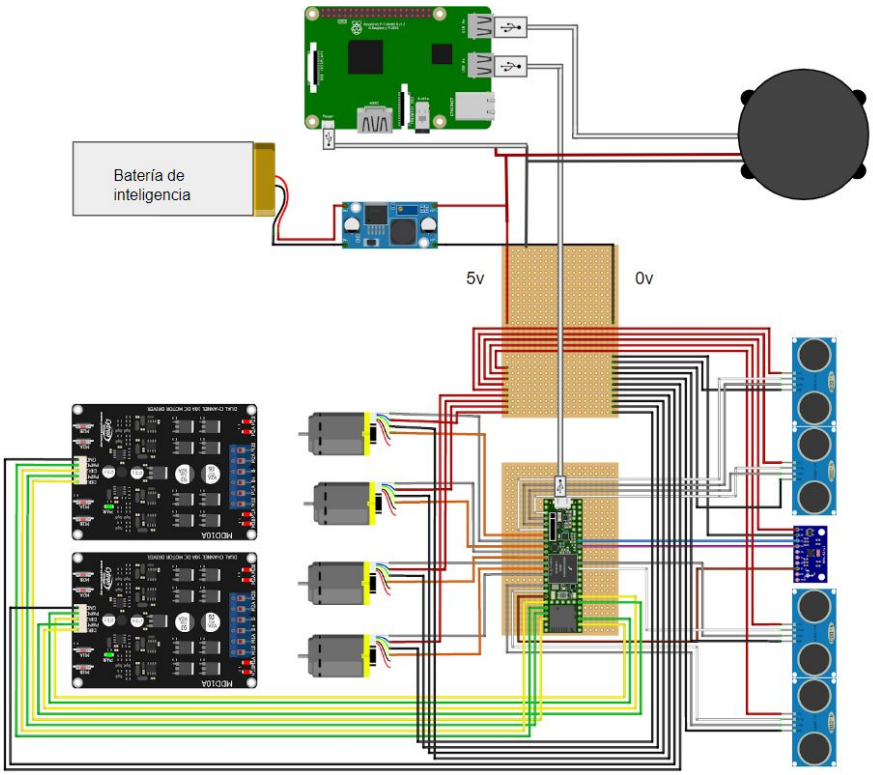


Figura 2.6.4: Diagrama de conexión de inteligencia

2.7. Descripción funcional

Los algoritmos que serán introducidos a continuación son los que permiten que a partir de la información de tres tipos de sensores, el robot sea capaz de recorrer el ambiente en que se encuentra de forma autónoma y generando en ese recorrido un mapa del entorno.

En la Figura 2.7.1, se puede ver un diagrama que refleja los aspectos más generales de la integración de los algoritmos.

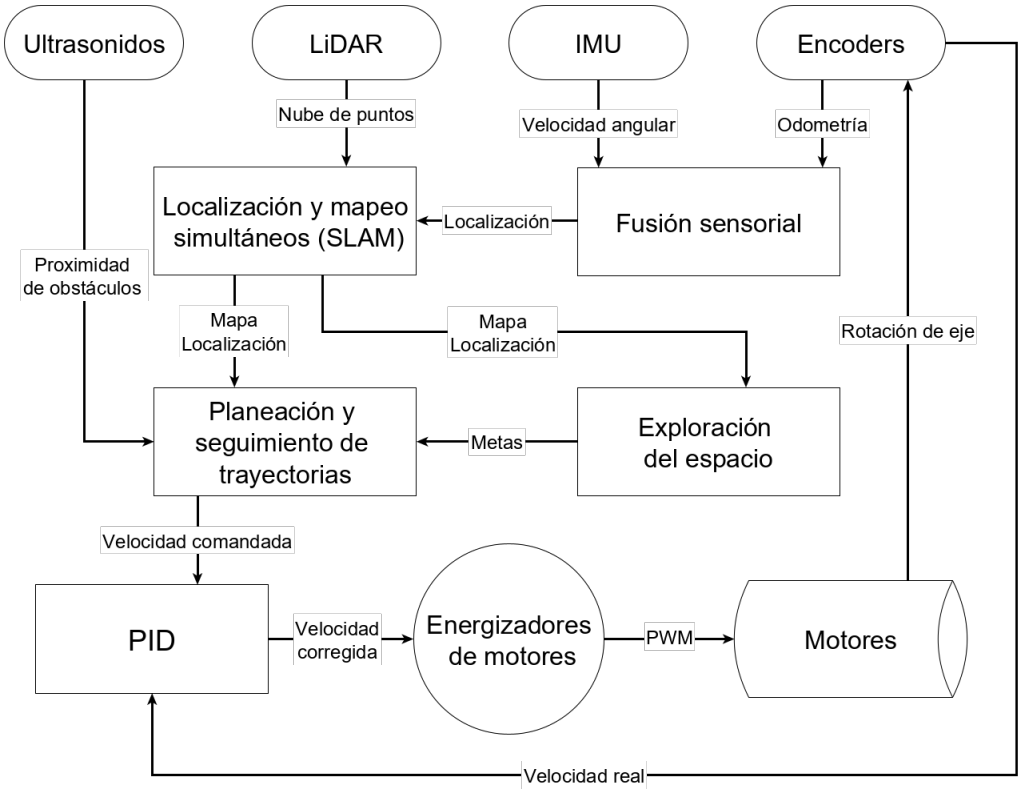


Figura 2.7.1: Diagrama de interacción de algoritmos y hardware

2.7.1. Fusión sensorial

La fusión sensorial consiste en combinar las observaciones de diferentes sensores (o un mismo sensor en distintos tiempos) para obtener información más significativa y certera que la dada por los sensores individualmente. La fusión sensorial es aplicada cuando no alcanza con un solo sensor para medir la magnitud deseada, o si la incertidumbre asociada a la medida es más de la tolerable por la aplicación.

La fusión sensorial en este proyecto fue implementada a través de un filtro de Kalman Extendido (EKF), un algoritmo basado en la estadística Bayesiana que combina información del giróscopo con la odometría para determinar la localización del robot con mayor precisión de lo que se lograría con estos sensores por separado.

2.7.2. Mapeo y Localización Simultáneos(SLAM)

El problema de mapeo y localización simultáneos consta de generar un mapa de un entorno desconocido mientras recorre el mismo. Además debe registrar el recorrido realizado mientras que se ubica esta trayectoria en el mapa.

Para resolver el problema antes detallado se debe equipar al robot con sensores y modelos matemáticos que le permitan identificar características del entorno y medir su movimiento. En este proyecto se utilizó por un lado un LiDAR para medir la distancia de paredes y otros obstáculos con respecto al robot, de esta manera se logra percibir las características del entorno. Por otro lado, se equipó al robot con encoders integrados a los ejes de los motores que en conjunto con la IMU y mediante fusión sensorial permiten estimar la posición. En este proyecto se utilizó el algoritmo Gmapping para resolver el problema. Sobre su funcionamiento se hablará en la Sección 5.3.

2.7.3. Planeación de trayectorias y exploración del espacio

La planeación de trayectorias resuelve el problema de encontrar un camino (si existe) entre una posición inicial y otra objetivo sin colisiones. Se trabaja bajo el supuesto de que se cuenta con la siguiente información: posición inicial, posición objetivo, descripción geométrica del ambiente y descripción geométrica del robot.

En este proyecto la planeación de trayectoria fue resuelta en dos niveles, uno global y otro local. En el nivel global, se dan las grandes directrices de comportamiento, atacando el problema como uno de caminos óptimos usando el algoritmo Djisktra [16]. Por otro lado, en el nivel local, se recibe del primero la trayectoria global y es el encargado de encontrar las velocidades relativas que debe seguir el robot para seguirla sin colisiones. Para ello se utilizó el método DWA [2].

Para solucionar el problema del mapeo autónomo se utilizó un módulo de exploración del espacio basado en detección de fronteras, que entrega posiciones objetivos al módulo de planeación de trayectorias para que pueda descubrir el ambiente de forma eficiente, en términos de tiempo.

[2]Dynamic Window Approach

2.8. Arquitectura lógica

El diagrama de la Figura 2.8.1 se divide en las tres unidades de hardware que se utilizan para el procesamiento de información del robot, que son: Teensy, Raspberry Pi y PC. Distribuidas en estas tres unidades de procesamiento, de acuerdo a sus características y limitaciones se implementaron las funciones principales que se describirán a continuación.

2.8.1. Manejo de sensores

El microcontrolador Teensy se encarga del manejo de los sensores con los que cuenta el robot para percibir el ambiente y su propio estado.

Sensores, como la IMU o los encoders, están continuamente tomando medidas y el Teensy simplemente se encarga de leer los valores entregados por estos sensores periódicamente. Por otro lado, los sensores de ultrasonido requieren que se inicie el proceso de medida cada vez, para luego devolver un valor de medición. Este proceso es también llevado a cabo por el microcontrolador.

Finalmente, el manejo de sensores se hace de tal modo de garantizar una frecuencia de muestreo constante para todos los sensores utilizando una arquitectura 'non-blocking'. Esto permite procesar los datos de los sensores en segundo plano, interrumpiendo el procesamiento para realizar medidas de acuerdo a la frecuencia configurada para cada sensor. Dichas configuraciones fueron:

- 1kHz para la cuenta de los encoders
- 5Hz para los Ultrasonidos
- 100 Hz para la IMU

2.8.2. Control de motores

El control de motores tiene como objetivo asegurar el seguimiento de los comandos de velocidad con el menor error posible y fijar la velocidad de cada rueda individual.

Los valores de velocidades relativas al robot deseadas son recibidos del control de más alto nivel y son procesados por el Teensy para generar la velocidad que debe adquirir cada rueda. Esto se hace a partir de las ecuaciones que se detallan en la Sección 4.3.

La velocidad calculada para cada rueda es introducida a un módulo de control PID que busca minimizar el error entre la velocidad deseada y la velocidad medida, obtenida a partir de las medidas de odometría. Además, se busca regular la velocidad de respuesta a los comandos de velocidad deseada para evitar que las ruedas deslicen al recibir una orden de acelerar súbitamente. Es de vital importancia poder asegurar un buen seguimiento de los comandos y el no deslizamiento de las ruedas para poder obtener buenos resultados en las misiones del robot. Se profundiza en el diseño del controlador PID en la Sección 5.1.

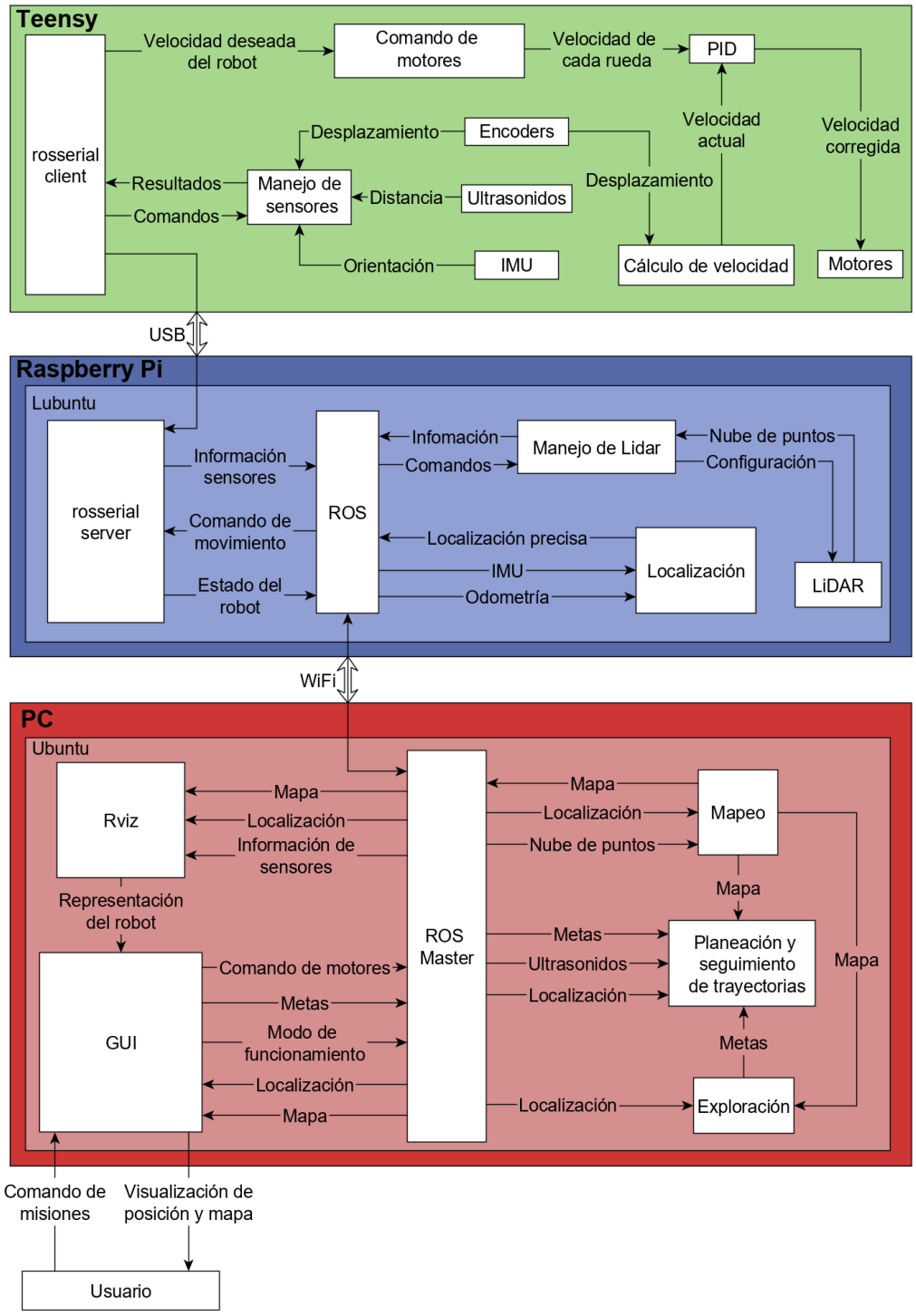


Figura 2.8.1: Arquitectura lógica del sistema

Finalmente, la comunicación de comandos y medidas de sensores con el control de más alto nivel se hace a través de la interfaz USB del Teensy utilizando un protocolo serial.

2.8.3. Localización

El módulo de localización corre en el Raspberry Pi y es el encargado de hacer la fusión sensorial en sistema. El módulo se basa en un filtro de Kalman extendido (referido también como EKF), en el que se profundizará en la Sección 5.2. El EKF se encarga de fusionar los datos de la IMU y odometría para producir un estimativo de la localización del robot respecto a su punto de comienzo, más preciso que la odometría o la IMU por separados.

2.8.4. Manejo del LiDAR

El LiDAR es un sensor láser de distancia giratorio que mide distancias abarcando 360° en un plano paralelo a su base. Por la cantidad de datos y la frecuencia con la que los produce, el LiDAR es controlado por el Raspberry Pi para no sobrecargar al Teensy. Los parámetros de funcionamiento del sensor son definidos al comienzo de la comunicación y luego el Raspberry Pi recibe periódicamente los datos de distancia del entorno que rodea al robot, llamados 'nube de puntos'.

2.8.5. Mapeo (SLAM)

El módulo de mapeo que corre en la PC, recibe del Raspberry Pi: la localización y la información del LiDAR. Estos datos se van utilizando para producir un mapa a medida que el robot va recorriendo los ambientes. Se van integrando los escaneos realizados por el LiDAR, ubicando cada uno de acuerdo a la localización recibida en ese momento. Este mapa se usa como base para los otros dos módulos principales: navegación y mapeo autónomo.

2.8.6. Navegación: Planeación y Seguimiento de Trayectorias

La funcionalidad de movimiento autónomo está dada por el módulo de planeación y seguimiento de trayectorias. Este se encarga de, dada una meta para el robot, planear la trayectoria que debe realizar, evitando obstáculos y siguiendo las restricciones puestas por el mapa. Además del mapa, se toman en cuenta los datos de distancia de los sensores de ultrasonido para evitar obstáculos que no sean identificados por el LiDAR.

2.8.7. Exploración

El módulo de exploración se encarga de definir las metas para el módulo de navegación. Este evalúa las fronteras del mapa construido hasta el momento y define metas en las zonas donde no tiene una frontera bien definida. Así es que el robot funcionando en este modo navega de forma autónoma por un ambiente hasta completar su mapa.

2.8.8. Interfaz de usuario

Finalmente, la interfaz de usuario en la PC permite al usuario interactuar con el programa para definir el modo de funcionamiento entre: el mapeo teledirigido, navegación autónoma, y la exploración del ambiente. Además, se envían las metas para el modo de funcionamiento autónomo a través de la interfaz.

A su vez, se puede visualizar en la interfaz la posición del robot, el mapa construido hasta el momento, los obstáculos detectados y la trayectoria planeada.

2.8.9. Comunicación entre módulos

La implementación de los módulos antes expuestos se realizó sobre ROS para facilitar y asegurar la comunicación entre ellos. ROS es un 'middleware', una capa de software que funciona por encima del sistema operativo pero por debajo de las aplicaciones. Provee múltiples funcionalidades orientadas al desarrollo de la robótica, asegurando la abstracción del hardware.

La ventaja de usar ROS es que provee gran flexibilidad para distribuir el trabajo computacional en varias máquinas separadas, conectadas simplemente a través de una red local. Se pueden correr los distintos programas o 'nodos', como se refiere comúnmente a estos, en las distintas máquinas y estos intercambian mensajes con formatos predefinidos por ROS.

Utilizando las estructuras de mensajes de ROS es que se transmite la información entre los dispositivos. Por un lado del Teensy al Raspberry Pi a través de la interfaz USB (con el protocolo rosserial de ROS) y por el otro del Raspberry Pi a la PC a través de WiFi. Aplicando la filosofía de cómputo distribuido de ROS es que se implementan los distintos módulos o nodos tanto en la PC como en el Raspberry Pi, resultando en un sistema modular e independiente del hardware.

Capítulo 3

Componentes de Hardware

3.1. Introducción

El Capítulo anterior describe las relaciones entre todos los componentes del sistema. En el presente Capítulo se detallan los componentes de hardware tanto eléctricos como mecánicos de forma individual. Además de presentar sus características principales, se explica brevemente el propósito de cada uno y la relación entre partes.

3.2. Resumen de Componentes

En la Tabla 3.1, se detallan los principales componentes utilizados, mientras que en la Tabla 3.2 se muestra como están dispuestos en el robot.

Categoría	**Elemento**	**Cantidad**	**Precio unitario (USD)**
Componentes Básicos	Rueda Mecanum	4	33
	Motor	4	40
	Batería de inteligencia	1	99
	Batería de motores	1	99
	Controladores	2	24
Instrumentación	LiDAR	1	169
	Ultrasonidos	4	5
	IMU	1	20
Inteligencia	Teensy 3.5	1	29
	Raspberry Pi 3B+	1	40

Tabla 3.1: Tabla de componentes

Piso	**Elemento**
Interior	Motores Controladores
Piso 1	Batería de motores Teensy 3.5 IMU
Piso 2	Raspberry Pi 3B+ Batería de inteligencia
Piso 3	LiDAR
Exterior	Ruedas Ultrasonidos

Tabla 3.2: Tabla de contenido por piso

3.3. Conexión eléctrica

Todos los componentes detallados en esta Sección se encuentran presentes en el diagrama de conexión de la Figura 3.3.2, y en el diagrama eléctrico de la Figura 3.3.1. En el mismo se puede observar tanto el circuito de motores como el de inteligencia a la vez, ambos descritos en la Sección 2.6, y tanto las conexiones de datos como las de energía.

Se puede ver en la parte inferior izquierda como la batería de inteligencia alimenta el conversor, que es quien provee una tensión de $5{,}4V$ a la salida. El conversor alimenta todos los sensores (LiDAR, encoders, IMU, ultrasonidos) y la placa Raspberry Pi. También alimenta indirectamente al Teensy ya que este se conecta por USB al Raspberry Pi, por donde también pasan los datos de los sensores conectados al Teensy. La información de las medidas del LiDAR se transmite desde el LiDAR mismo hacia el Raspberry via USB. Por otro lado la batería de motores se encarga de alimentar a los 2 controladores de motores, y estos últimos a dos motores cada uno.

En cuanto a conexión de datos refiere en la Tabla 3.3, se puede observar la conexión de pines del Teensy.

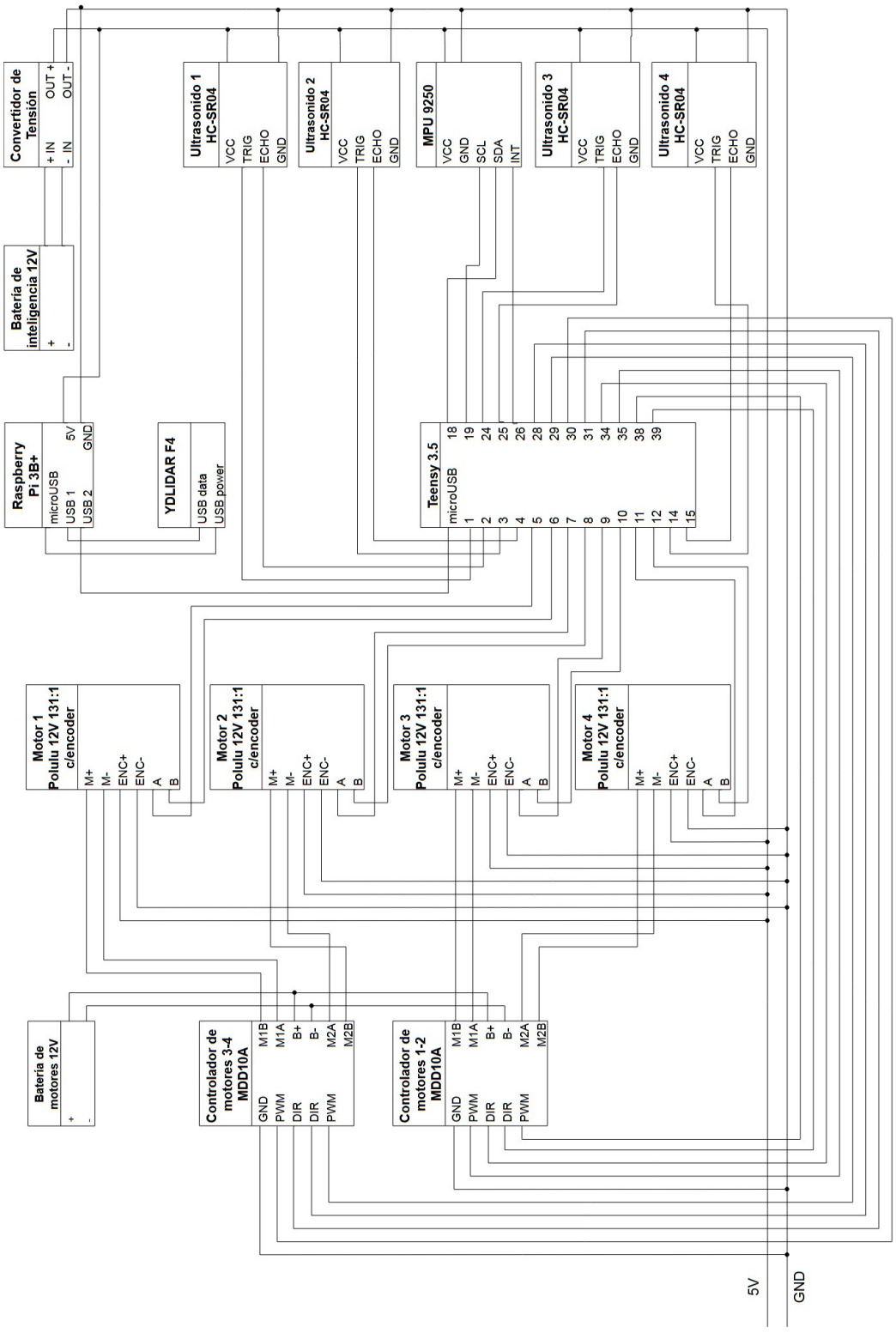


Figura 3.3.1: Diagrama eléctrico del robot

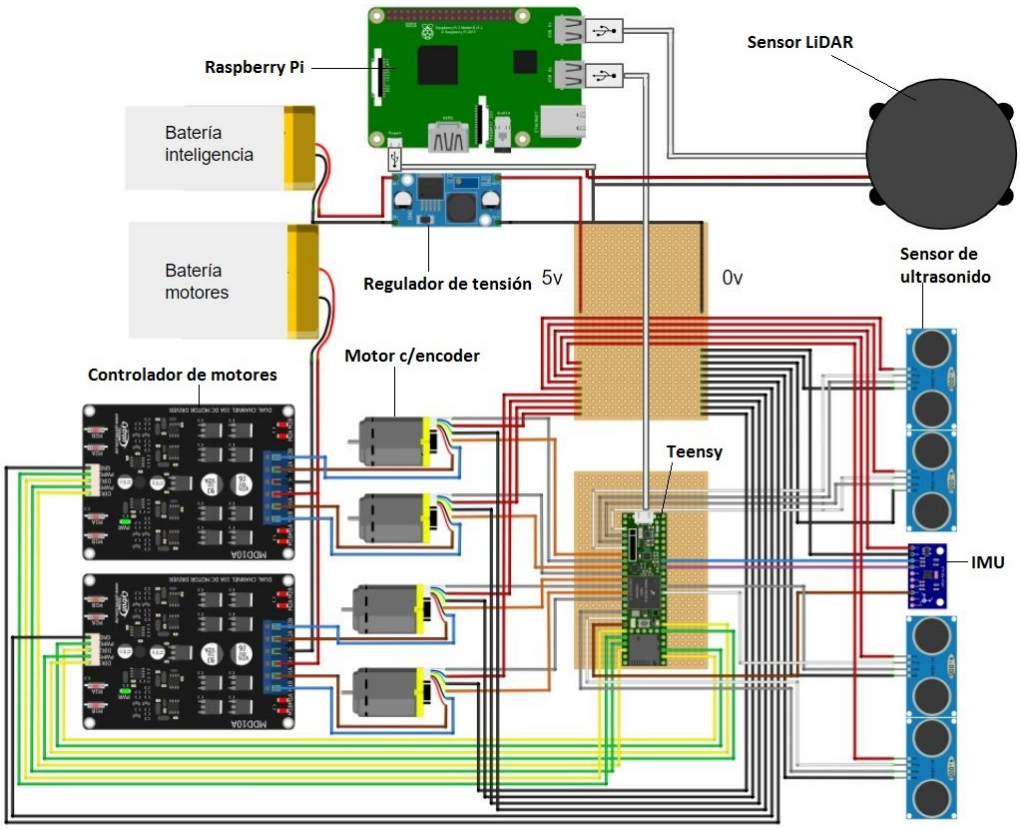


Figura 3.3.2: Diagrama de conexión de componentes

Pin	Descripción	Color
1	Trigger - US Der	
2	Echo - US Der	
3	Trigger - US Atr	
4	Echo - US Atr	
5	Encoder 1 - A	
6	Encoder 1 - B	
7	Encoder 2 - A	
8	Encoder 2 - B	
9	Encoder 3 - A	
10	Encoder 3 - B	
11	Encoder 4 - A	
12	Encoder 4 - B	
14	Trigger - US Izq	
15	Echo - US Izq	
18	IMU - SDA	
19	IMU - SCL	
24	Trigger - US Adel	
25	Echo - US Adel	
26	IMU - Int	
28	Motor 2 - DIR	
29	Motor 2 - PWM	
30	Motor 1 - PWM	
31	Motor 1 - DIR	
34	Motor 3 - DIR	
35	Motor 3 - PWM	
38	Motor 4 - PWM	
39	Motor 4 - DIR	

Tabla 3.3: Tabla de Pines

3.4. Componentes Básicos

3.4.1. Ruedas

Con el propósito de generar un movimiento omnidireccional, se utilizaron ruedas Mecanum. Este tipo de ruedas, está compuesta por rodillos colocados a 45 grados respecto del plano de la rueda que giran libremente. Estos permiten que la rueda se mueva libremente en la dirección del rodillo en contacto con el piso. La combinación de varias ruedas Mecanum en una configuración como la del robot construido hacen posible que se muevan en cualquier dirección, cambiando las velocidades y sentidos de giro de estas ruedas. Esto se debe a que las componentes de velocidad de cada rueda se suman o cancelan, logrando que la velocidad neta del robot tenga cualquier sentido y dirección. Se desarrollan estos conceptos de forma matemática en la Sección 4.3.

Cada rueda se integra al robot a partir de un adaptador que lo conecta mecánicamente al motor como se explica en la Sección 2.4.

Figura 3.4.1: Rueda Mecanum

Especificaciones:

- Masa: 400 g
- Díametro: 100 mm
- Ancho: 50 mm
- Compuesto por 9 rodillos por rueda
- Largo de los rodillos: 19 mm

3.4.2. Motores

Se utilizaron 4 motores DC con escobilla, uno por cada rueda. La particularidad de estos motores es que incluyen *encoder*, que permiten contar la cantidad de vueltas del eje del motor.

3.4.2.1. Dimensionamiento del robot

Para el dimensionamiento de los motores, como primera aproximación se despreció el rozamiento y se supuso un carro con las siguientes características:

Masa (m)	3 kg
Radio de la rueda (r)	3 cm
Velocidad máxima (v)	20 cm/s
Pendiente máxima (γ)	12°
Aceleración máxima (a_{neta})	10 cm/s^2

Tabla 3.4: Requerimientos mecánicos del robot

En la Figura 3.4.2 se puede observar el diagrama de cuerpo libre del robot. Se describe la dinámica del robot a partir de las siguientes dos ecuaciones:

$$F_{aplicada} = F_{neta} + mgsen\theta \tag{3.4.1}$$

$$T_{total} = F_{aplicada}.r \tag{3.4.2}$$

La cinemática se rige por esta ecuación que relaciona la velocidad angular de las ruedas con la velocidad lineal del carro:

$$\omega_{max} = \frac{v_{max}}{r} \tag{3.4.3}$$

Se puede escribir la corriente y potencia consumida por los motores como:

$$P_{max} = T_{total}\omega_{max} \tag{3.4.4}$$

$$I_{max} = \frac{P_{max}}{V} \tag{3.4.5}$$

A partir de los requerimientos del robot presentados en la Tabla 3.4, las ecuaciones 3.4.1, 3.4.2, 3.4.3, 3.4.4 y 3.4.5, y suponiendo una alimentación de 12 V, se calculan los requerimientos y se exponen en la siguiente Tabla 3.5.

	Total	Por motor
Potencia máxima	1,28W	0,32W
Torque máx	0,19Nm	0,048Nm
Corriente máxima	0,11A	0,0267A
RPM máx	63,8	63,8

Tabla 3.5: Requerimientos de los motores

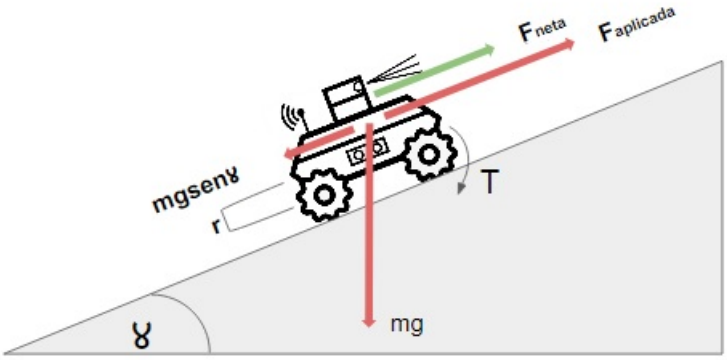


Figura 3.4.2: Diagrama del cuerpo libre del robot

3.4.2.2. Características de los motores

Con los requerimientos expuestos en la Tabla 3.5 y las opciones presentes en el mercado se decidió utilizar el siguiente motor, ilustrado en la Figura 3.4.3. Este satisface los requerimientos y además integra un encoder de gran precisión dentro del presupuesto disponible.

Figura 3.4.3: Motor DC 12 V

Especificaciones [31]:

- Modelo: Pololu 12V, 131:1 Metal Gear Motor w/64 CPR Encoder
- Voltaje nominal: 12 V
- Reducción 131.25:1
- Encoder: 64 (8400) cuentas por vuelta del motor (eje de salida).
- Corriente (rotor bloqueado/sin carga): $5A/0{,}3A$
- RPM(sin carga): 80
- Torque(rotor bloqueado/sin carga): $1{,}765Nm/0{,}353Nm$

3.4.2.3. Encoders

El motor está equipado con un encoder de efecto Hall de dos canales que mide la rotación de un disco magnético solidario al eje del motor. En la Figura 3.4.4 se ilustra un esquema de funcionamiento del encoder. Se puede ver que ambos canales están desfasados ya que apuntan a lugares distintos del disco, más aún están separados por construcción de tal manera que las señales eléctricas que emitan estén desfasadas $\frac{\pi}{2}$, es decir en fase y cuadratura.

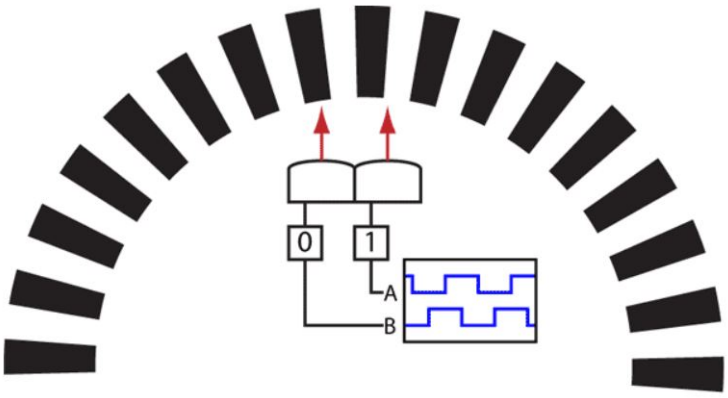


Figura 3.4.4: Diagrama de un encoder en cuadratura

En la Figura 3.4.5 se puede observar una captura de la pantalla de un osciloscopio donde se ven los dos canales. Se observa una onda cuadrada por canal que oscila entre 0 y VCC desfasada $\frac{\pi}{2}$ entre ellos. La frecuencia de oscilación permite obtener la velocidad de rotación, mientras que el orden de los canales el sentido. Contando los flancos de subida y bajada de ambos canales se obtiene una resolución de 64 cuentas por vuelta del eje del motor antes de la reducción.

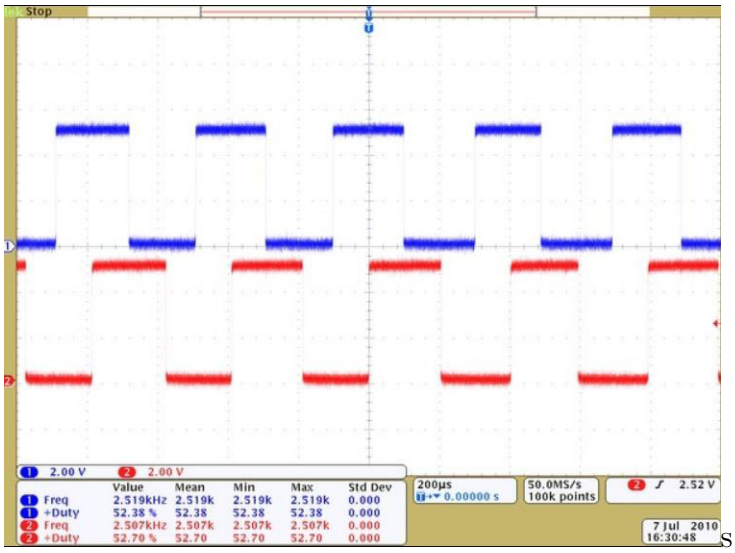


Figura 3.4.5: Captura de canales del encoder en un osciloscopio

El encoder equipado se puede observar en la Figura 3.4.6. Como se dijo anteriormente tiene una resolución de 64 cuentas por vuelta del eje del motor, teniendo en cuenta que la reducción del motor es 131.25:1 [1], se tiene una precisión 8400 cuentas por cada vuelta del eje de salida.

[1]Es decir que por cada vuelta del eje de salida, el eje del motor realizó 131.25 vueltas

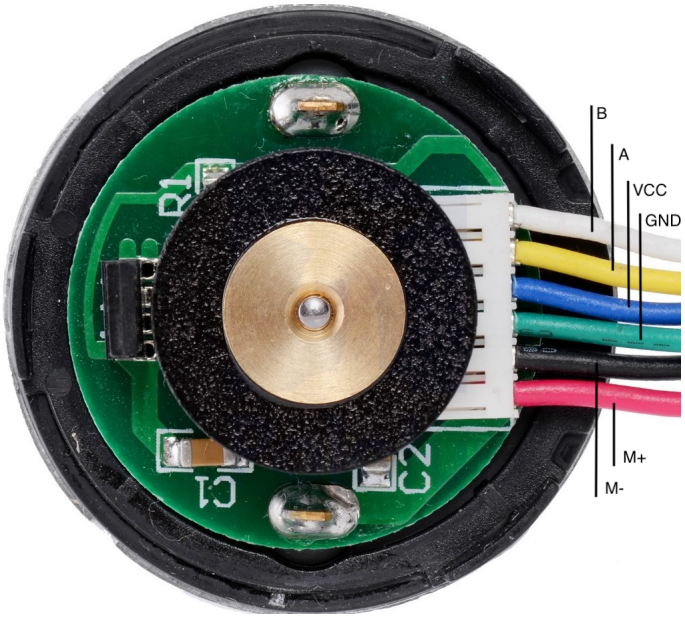


Figura 3.4.6: Terminales del motor y Encoder

En la Figura 3.4.6 se muestra el circuito impreso que contiene al encoder y los cables que alimentan tanto al motor como a la logica del encoder. Los cables M+ y M- estan conectados directo a los bornes del motor y son los encargados de suministrar la potencia al mismo. Por otro lado VCC y GND sirven para alimentar la lógica de los encoders siendo la salidas del mismo A y B.

3.4.3. Controladores

Los controladores son los encargados de mover los motores según las órdenes provenientes de la inteligencia a bordo.

El controlador utilizado es básicamente un puente H como el que se puede ver en la Figura 3.4.7. La inteligencia a bordo envía al controlador dos señales digitales de control, DIR y PWM. La señal DIR indica el sentido en que debe moverse el robot. Cuando DIR está HIGH se puede ver que las llaves S1 y S3 están cerradas mientras que S2 y S4 abiertas, sin embargo si DIR está en LOW se puede ver que la corriente circula en el sentido contrario ya que S1 y S3 están abiertas y S2 y S4 cerradas. El sentido en que pasa la corriente por el motor determinará el sentido en que gira el eje.

La señal PWM impone la velocidad a la que debe girar el eje. PWM [2] es una técnica de modulación por ancho de pulso que envía una onda cuadrada como las que se pueden ver en la Figura 3.4.8. Se varía el ciclo de trabajo de estas ondas cuadradas, es decir el porcentaje del período que se encuentran en HIGH. La ventaja que permite PWM es variar la tensión en bornes del motor sin variar

[2]Por sus siglas en inglés *Pulse Width Modulation*

la tensión de alimentación, ya que el motor será sensible a la tensión media, que se expresa en la ecuación 3.4.6.

$$V_{motor} = (\,\%ciclo\ de\ trabajo)VCC \tag{3.4.6}$$

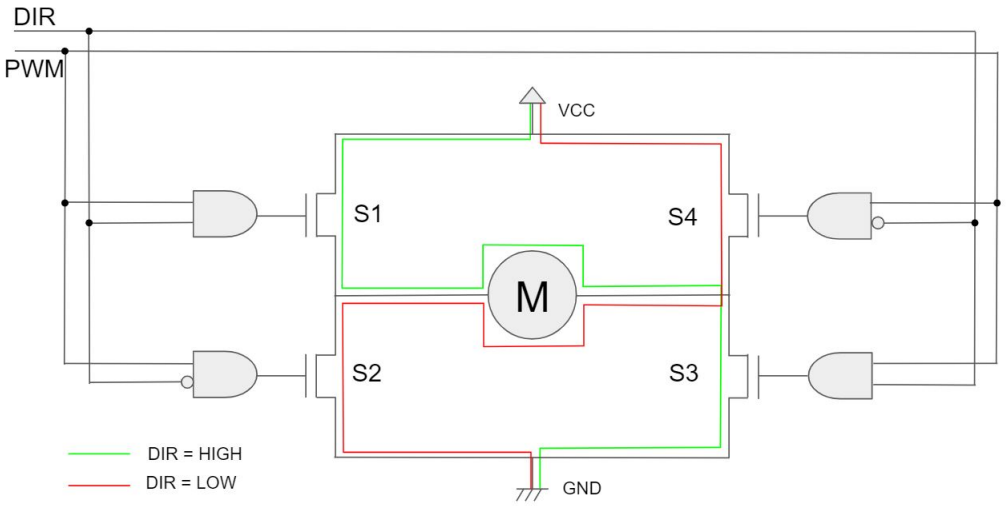


Figura 3.4.7: Esquemático del puente H

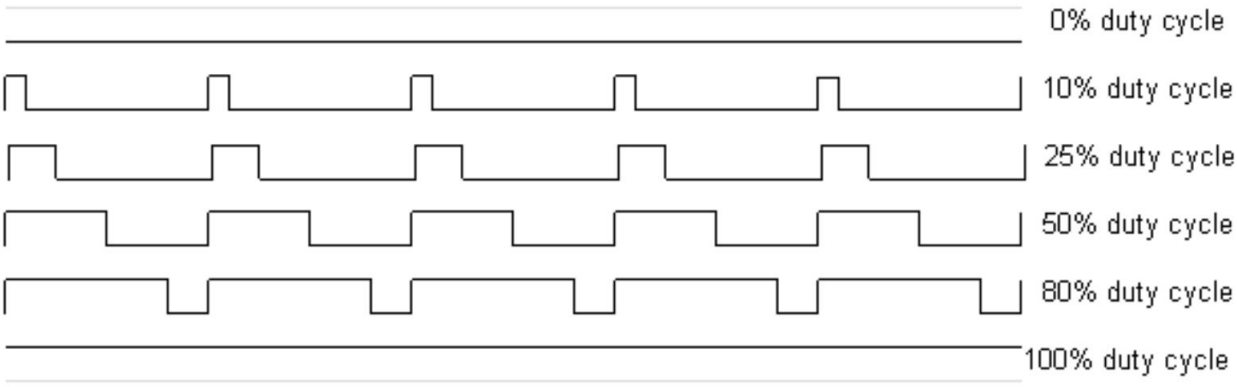


Figura 3.4.8: Ejemplo de distintos pulsos PWM

Se utilizaron dos controladores, de 2 canales cada uno. Un controlador trabaja con los motores delanteros, y otro con los traseros. En la Figura 3.4.9 se puede observar el controlador utilizado.

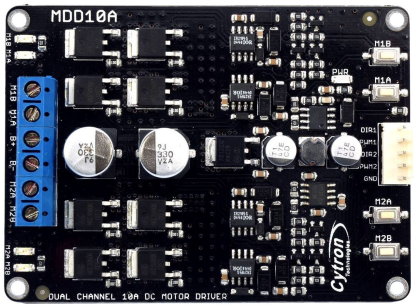


Figura 3.4.9: Controlador de Motores

Especificaciones [7]:

- Nombre: Cytron 10A 5-30V Dual Channel DC Motor Driver
- Modelo: MDD10A
- Corriente máxima continua: 10 A
- Corriente máxima de pico : 30 A
- Frecuencia máxima de PWM: 20 kHz
- Entrada lógica (alta): 3-5,5V
- Entrada lógica (baja): 0-0,5V

3.4.4. Alimentación

Se utilizaron dos paquetes de baterías del tipo *Li-Ion*, a partir de ahora serán llamadas batería de inteligencia y batería de motores, ubicándose en el segundo y primer nivel respectivamente.

Los paquetes de baterías fueron armados a partir de baterías de laptop disponibles en plaza. Esto permitió conseguir un precio razonable, evitando las complejidades de importar baterías. Se les desarmaron las carcasas y se les incorporaron a cada una un conector para carga balanceada.

Estas baterías están compuestas cada una por pilas de la serie *18650*, que cuentan con las características detalladas en la Tabla 3.6.

Adicionalmente se cuenta con un *buzzer* en cada batería como método de alarma ante la descarga de las celdas, y un convertidor DC-DC para la salida de la batería de inteligencia.

Todos los componentes que conforman el sistema de alimentación serán explicados a continuación. La conexión se mostró con detalle en la Sección 3.3.

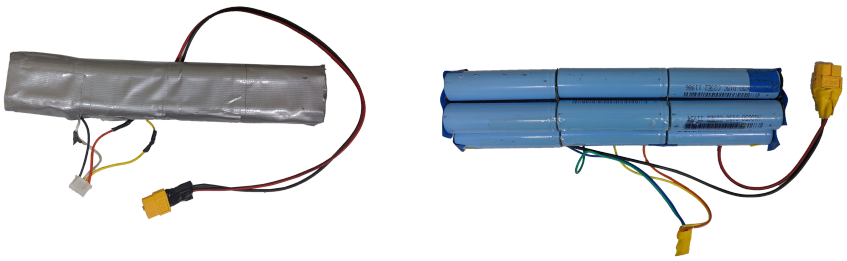
Capacidad	2200mAh
Voltaje nominal	3,7V
Voltaje de carga máximo	4,2V
Voltaje de corte	2,7V

Tabla 3.6: Principales características de la serie 18650

3.4.4.1. Baterías de inteligencia

La batería de inteligencia es la encargada de alimentar los dos módulos de inteligencia del robot, es decir las placas Teensy y Raspberry Pi. Además alimenta a todo el conjunto de sensores, incluyendo al LiDAR que es el sensor más demandante.

Para la batería superior se utilizó una configuración 2P3S, es decir 2 conjuntos de 3 pilas en serie, conectados a su vez en paralelo, teniendo un voltaje nominal de 11,1V y capacidad de 4400 mAh.



(a) Baterías de inteligencia (b) Baterías de motores

Figura 3.4.10: Baterías del robot

3.4.4.2. Baterías de motores

La alimentación de los motores, realizada a través de los dos controladores, es tarea de la batería de motores. Esta batería deberá asumir una carga mayor a la de la batería de inteligencia, por ello se utilizó una configuración 4P3S, también con un voltaje nominal de 11,1V y capacidad de 8800 mAh.

A pesar de que el voltaje nominal de los motores es de 12 V, puede funcionar a tensiones menores sin inconvenientes.

3.4.4.3. Regulador de Tensión

Se utilizó un *buck converter* para alimentar la Raspberry Pi, tiene como entrada la batería de inteligencia con una tensión de 11,1V y como salida la entrada de alimentación del Raspberry Pi a una tensión de 5,4V.

El modelo utilizado fue el *LM2596* [40], el cual se encuentra representado en la Figura 3.4.11.

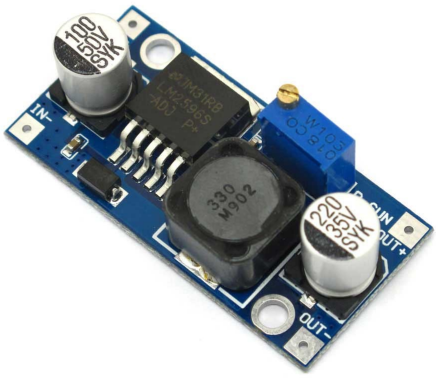


Figura 3.4.11: Regulador de tensión basado en el LM2596

3.4.4.4. *Buzzer* para baterías

Como método de alarma ante la descarga de las baterías, se conecta este componente al conector de balanceo. Este conector se encuentra conectado en cuatro punto al pack, tanto en sus extremos como en la unión de los paquetes en paralelo. De esta manera se puede utilizar tanto para recargar la batería de forma balanceada o bien relevar su voltaje en todo momento para indicar cuando una de sus celdas baja de un voltaje umbral Mediante luces LED indica el estado de las celdas y emite un pitido cada vez que la tensión de alguna celda baja de $3{,}3v$. Esto permite evitar el uso las baterías por debajo de la tensión de corte, para asegurar un mayor rendimiento a largo plazo.

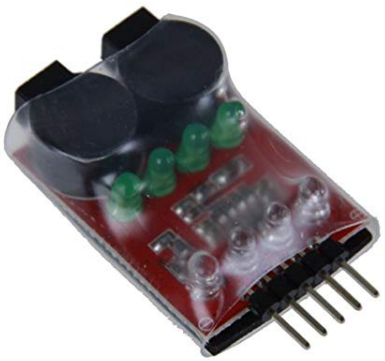


Figura 3.4.12: *Buzzer* para baterías

3.4.5. Instrumentación

Se detallan en la presente Sección los componentes de instrumentación que serán utilizados para sensar la distancia a los obstáculos y el movimiento del robot.

La capacidad sensorial del robot, dividida en tres tipos de elementos sensoriales, está compuesta por: un LiDAR, una IMU y cuatro sensores de ultrasonido.

3.4.5.1. LiDAR

El LiDAR es un escáner láser bidimensional. Se trata de un sensor infrarrojo de distancia giratorio, es decir que funciona como un radar.

La información que provee como salida es la distancia a la cual se encuentran los objetos para cada ángulo. Este ángulo es determinado a través de un encoder que se encuentra en el eje de giro del sensor. En cada rotación el LiDAR toma en el orden de cientos de medidas de modo de tener una gran resolución angular y así poder reconstruir fielmente el plano que escanea. Por esto puede detectar precisamente la forma de los objetos que lo rodean, y es sumamente usado en aplicaciones de mapeo.

Para detectar la distancia a la que se encuentran los objetos se basa en un sistema geométrico como el que se muestra en la Figura 3.4.13 [2]. El láser emite una señal infrarroja, que es reflejada en el objeto detectado. Este pulso reflejado pasa a través del lente e impacta contra el sensor CCD [3] , el cual es capaz de estimar el largo del segmento $b_k^{'}$.

Con la hipótesis de que la distancia al objeto es mucho mayor que la distancia entre el láser y el lente, podemos decir que distancia entre el láser y el punto del lente por el cual pasa el pulso reflejado (b), es constante.

Por la geometría del problema se puede ver que los triángulos (b, d_k) y $(b_k^{'}, d^{'})$ son semejantes ya que sus tres ángulos son iguales. La semejanza implica que sus lados y alturas homólogos serán proporcionales.

Dada la semejanza entre (b, d_k) y $(b_k^{'}, d^{'})$ y que los segmentos b y $d^{'}$ tienen largo fijo y conocido, se puede calcular la distancia al objeto detectado como $d_k = d^{'} \frac{b}{b_k^{'}}$

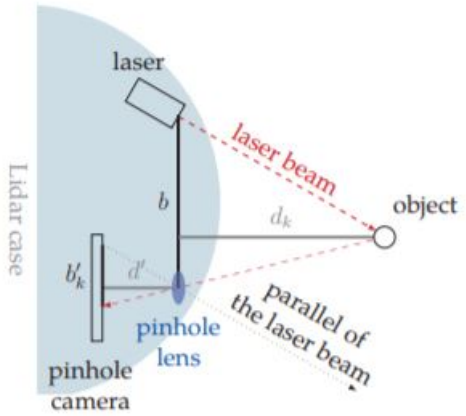


Figura 3.4.13: Principio de funcionamiento del LiDAR

[3] *charged-coupled device* por sus siglas en inglés. Se trata de un dispositivo electrónico que cuenta con un circuito integrado capaz de detectar los fotones incidentes sobre su superficie

Figura 3.4.14: LiDAR

Especificaciones [43]:

- Modelo: YDLIDAR F4 360° Laser Scanner
- Rango: 12 m
- Frecuencia de muestreo: 6000 muestras/segundo
- Frecuencia de giro: 5-12 vueltas/segundo

3.4.5.2. Sensores de ultrasonido

El sensor de ultrasonido es utilizado para evitar colisiones. Se trata de un sensor de distancia colocado a la altura de las ruedas para percibir objetos bajos que no son detectados por el LiDAR pero están presentes en el ambiente.

Cuenta con cuatro pines: *VCC*, *GND*, *Trigger* y *Echo*. Los dos primeros se utilizan para alimentación, mientras que los dos últimos se utilizan para realizar la medida.

El principio de funcionamiento de este sensor es sencillo, cuenta el tiempo de viaje de un pulso de ultrasonido a través del aire. A partir de este tiempo, siendo la velocidad del sonido conocida ($v = 343{,}2 m/s$) , se determina la distancia recorrida por el pulso con la siguiente ecuación.

$$distancia(m) = \frac{t.v}{2}$$

Para lanzar la medida se emite un pulso TTL de 10 μs en el pin de Trigger. A continuación un set de ocho pulsos de ultrasonido (40 kHz) es emitido. Después de emitidos los pulsos de ultrasonido el pin Echo sube a HIGH, hasta que se recibe el primer pulso de ultrasonido reflejado, volviendo a LOW. El tiempo de viaje da la onda sonora es igual al pulso generado por el pin de Echo. El proceso se descrito se puede observar en la Figura 3.4.15.

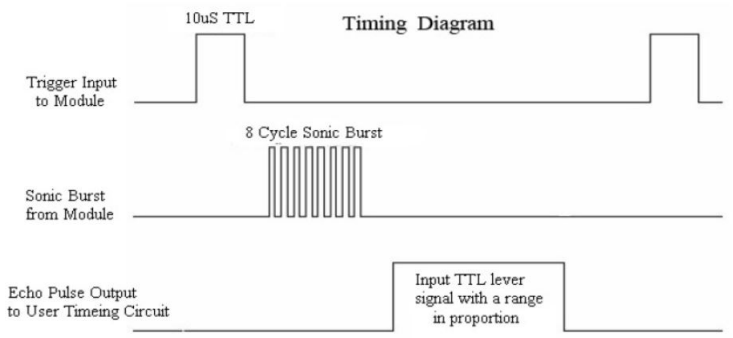


Figura 3.4.15: Diagrama de tiempos del ultrasonido

Figura 3.4.16: Sensor de Ultrasonido

Especificaciones [24]:

- Modelo: HC-SR04
- Alcance: 4 m
- Distancia Mínima: 2 cm
- Frecuencia: 40 kHz
- Ángulo de medida: 15 grados

3.4.5.3. IMU - Unidad de Navegación Inercial

La unidad de navegación inercial usada está compuesta por varios sensores. Se tiene a disposición: acelerómetro, giróscopo y magnetómetro. Se utilizó únicamente la información proveniente del giróscopo, para alimentar el módulo de fusión sensorial.

El giróscopo equipado en la IMU es de estado sólido o *MEMS*[4], basado en la aceleración de Coriolis.

Esta aparece si se tiene un objeto que se mueve en línea recta sobre un marco de referencia que está girando. Un observador exterior desde un marco inercial verá que la trayectoria del objeto es una curva, por lo cual debe haber una aceleración sobre el objeto hacia el interior de la curva. Esta es la aceleración de Coriolis.

Si el objeto tiene una velocidad lineal v, y el marco gira a una velocidad Ω, la aceleración que experimentará es:

$$a = 2v \times \Omega \tag{3.4.7}$$

Teniendo en cuenta lo anterior, para medir la velocidad angular (Ω), se induce una velocidad lineal (a través de una masa vibrante) v y se mide la aceleración de Coriolis resultante. Por último se despeja Ω de la ecuación 3.4.7.

[4]Micro-Electro-Mechanical Systems

La IMU se encuentra en el módulo de desarrollo GY9250 que contiene la MPU9250 y otros componentes como barómetro BMP280. Esta permite interactuar con los sensores con mayor facilidad a través de sus pines.

Figura 3.4.17: Placa de desarrollo GY 9250

Especificaciones [15]:

- Modelo: Invensense MPU 9250
- Protocolo de comunicación: I2C, SPI
- Sensores: giróscopo, acelerómetro y magnetómetro
- Giróscopo de estado sólido de escala programable con ADC de 16 bits

3.5. Inteligencia

Para el diseño de la inteligencia del robot se plantea una arquitectura en dos niveles, cada nivel con una placa de hardware correspondiente. Al tratarse de un robot funcionando en tiempo real, la división de tareas se realizó con el objetivo de poder asegurar la realización de las mismas en tiempo crítico.

Esta división de tareas se realizó separándolas por nivel de modo que la carga se distribuyó de la siguiente manera: Las tareas de alto nivel (procesamiento) fueron llevadas a cabo por la Raspberry Pi 3 B+ y las tareas de bajo nivel (como sensado y comando) fueron realizadas por el Teensy 3.5

3.5.1. Teensy 3.5

El Teensy 3.5 es un sistema de desarrollo basado en un microcontrolador ARM Cortex M4. Es compatible con el entorno de desarrollo de Arduino y sus librerías.

Se utiliza para tareas en las que el tiempo de procesamiento es crítico para el robot, entre ellas: comando de motores, lectura de IMU, encoders y ultrasonidos.

Figura 3.5.1: Teensy 3.5

Especificaciones [30]:

- Procesador de 120 MHz ARM Cortex-M4 con unidad de punto flotante.

- 512 Kb de memoria flash, 192 Kb de memoria RAM
- 62 pines de entrada/salida de 3.3V pero toleran 5V, todos pueden ser usados para interrupciones
- 25 entradas analógicas
- 20 salidas de PWM

3.5.2. Raspberry Pi 3 B+

Raspberry Pi es una single-board-computer que permite las mismas opciones que muchas computadoras de escritorio, pero en un tamaño mucho menor y menor memoria y poder de procesamiento.

Se utiliza para el procesamiento de módulos del sistema y es la encargada de centralizar toda la información proveniente de los sensores.

Trabaja en conjunto con el microcontrolador para manejar el robot y con una computadora externa para lograr el comando remoto y dividir la carga computacional.

La comunicación entre la computadora externa y la Raspberry Pi se hace mediante el uso de WiFi.

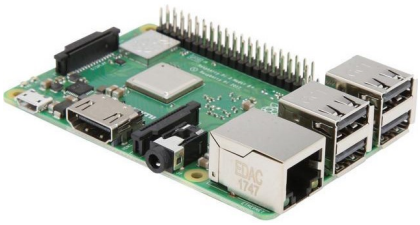


Figura 3.5.2: Raspberry Pi 3 B+

Especificaciones [34]:

- Procesador Cortex-A53 (ARMv8) 64-bit SoC[a] @ 1.4GHz
- 1 GB de memoria RAM
- 40 pines de GPIO[b]
- Sistema Operativo Lubuntu
- Redes WLAN de 2.4 y 5 GHz
- Bluetooth 4.2

[a] System On a Chip
[b] General Purpose Input Output

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 4

Modelado y Caracterización

4.1. Introducción

La siguiente Sección detalla el proceso de modelado físico-matemático del robot. Este proceso permite obtener las ecuaciones de movimiento, que son el centro del control del robot y la base para todos los algoritmos que se describirán en el Capítulo 5.

4.2. Modelado

4.2.1. Definición de sistemas de coordenadas del robot:

A efectos de poder representar las variables de interés del robot como su posición y velocidad con mayor facilidad y de forma coherente, en la Figura 4.2.1 se muestra la definición del sistemas de coordenadas del robot, en particular la definición de los ejes de referencia solidarios a las ruedas (O_{wi}), al robot completo (O_r) y el absoluto (O_q).

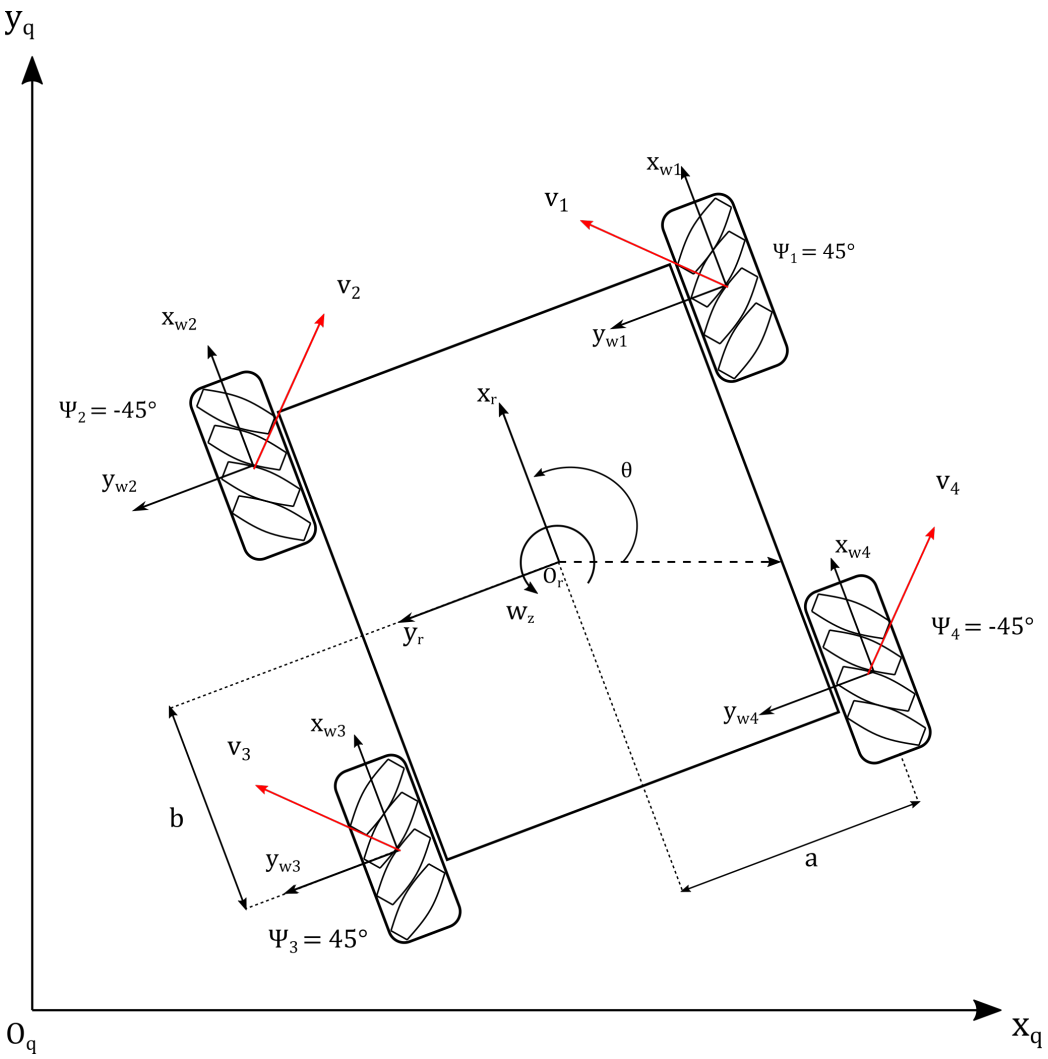


Figura 4.2.1: Definición de sistemas de coordenadas del robot

Como puede observarse en la Figura 4.2.1, se definen ejes de referencia solidarios a cada rueda orientados de la misma manera que el solidario al robot. Esto permite expresar la velocidad del robot en cualquiera de estos ejes de coordenadas y también deducir las ecuaciones para cambiar de referencial los parámetros de interés. También se representan en la Figura los vectores de velocidad de cada rueda.

En la Figura 4.2.2 se puede observar en más detalle como se definen estos vectores de velocidad para cada rueda.

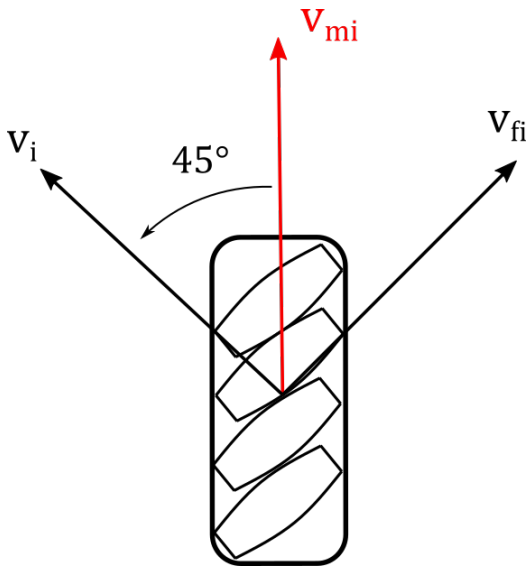


Figura 4.2.2: Modelo de la cinemática de una rueda

Estos están orientados a $\pm 45°$ respecto al eje x de cada rueda debido a que los rodillos de las ruedas descomponen la velocidad v_{mi} producida por el motor. Esta velocidad es descompuesta en dos vectores ortogonales, v_{fi} y v_i. Se tomará como hipótesis que la primera es cancelada por el rozamiento estático del rodillo con el piso por ser ortogonal al sentido de movimiento del rodillo en contacto con el piso. De este modo, se puede representar la velocidad de cada rueda con los respectivos vectores v_i.

4.2.2. Holonomía

La holonomía de un sistema mecánico describe la relación entre los grados de libertad controlables y los grados de libertad de movimiento del robot.

Un robot es holónomo si la cantidad de grados de libertad controlables es igual a los grados de libertad de movimiento, es decir que puede desplazarse en todas las direcciones de movimiento sin restricciones y de forma independiente. Un ejemplo de un robot holónomo es un cuadricóptero, que puede desplazarse en las 3 coordenadas espaciales (x, y, z) y realizar rotaciones en los 3 ángulos (roll, pitch, yaw).

Por otro lado, un robot anholónomo tiene menos grados de libertad controlables que grados de libertad de movimiento. Esto en la práctica significa que el robot no puede desplazarse en todas las direcciones de movimiento sin realizar maniobras, como es el caso de un automóvil.

En términos matemáticos, un sistema es holonómico si todas las restricciones a las que está sometido pueden ser expresadas por ecuaciones que dependan solamente de las variables de posición de dichos grados de libertad y del tiempo. Estas

se pueden escribir de la forma: [37]

$$f(x_1, x_2, x_3, ..., x_n, t) = 0 \tag{4.2.1}$$

Donde $x_1, x_2, x_3, ..., x_n$ son las variables de posición del sistema. En el caso del robot en cuestión, estas serían x, y, θ porque se mueve en el plano.

En contraposición, una restricción anholónoma es una que no puede expresarse de la manera anterior, sino que depende de derivadas temporales de las variables de estado del sistema ([35], parte A, Sección 2.3.6). Un sistema con al menos una restricción anholónoma se considera anholónomo.

En particular, el robot construido es holónomo, como se demostrará a continuación a través de las ecuaciones.

4.3. Cinemática:

4.3.1. Introducción:

Las ecuaciones cinemáticas describen el movimiento del robot respecto a parámetros constructivos del mismo, pero sin tener en cuenta las causas que producen el movimiento (fuerzas y pares).

El desarrollo de las siguientes ecuaciones se hace siguiendo el trabajo realizado en [1].

4.3.2. Ecuaciones:

Para comenzar se debe establecer la nomenclatura para los distintos parámetros que describen la dinámica:

- $\dot{\phi_{wi}}$: Velocidad angular de la rueda i alrededor del adaptador
- ψ_i: Ángulo de los rodillos respecto a la rueda i
- r: Radio de los rodillos
- R: Radio de las ruedas
- $\dot{\phi_{ri}}$: Velocidad angular del rodillo en contacto con el suelo de la rueda i
- $\dot{\theta_{wi}}$: Velocidad angular de la rueda i respecto a z (yaw), que es igual para todas porque están mecánicamente ligadas.

Para obtener las ecuaciones de movimiento del robot completo se debe comenzar por describir el movimiento de cada rueda.

Se puede comenzar por plantear el eje de coordenadas de la rueda i, que determina el referencial solidario para cada una de las ruedas. Sea O_{wi} el referencial de cada rueda, donde el centro del referencial está en el centro de cada rueda y el referencial mismo es solidario al carro completo.

Se puede observar la ubicación del referencial en la rueda en una vista de plano horizontal en la Figura 4.3.1.

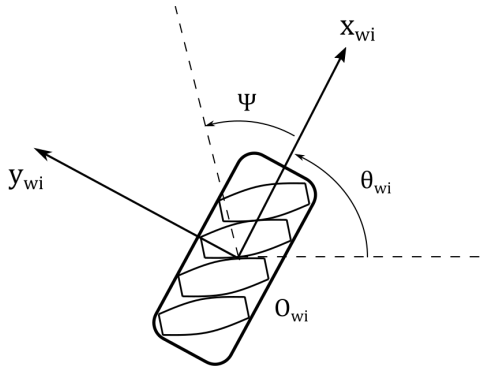


Figura 4.3.1: Definición de ejes de referencia para ruedas

A partir del referencial definido, se puede definir el vector de velocidades, $\dot{P}_{wi}$, en el referencial solidario a la rueda en la ecuación 4.3.1.

$$\dot{P}_{wi} = \begin{bmatrix} \dot{x_{wi}} \\ \dot{y_{wi}} \\ \dot{\theta_{wi}} \end{bmatrix} \tag{4.3.1}$$

Luego, se plantea la cinemática de las ruedas y sus partes en la Figura 4.3.2 aplicando las definiciones antes mencionadas. Nótese que se representa tanto la rueda en su conjunto como el rodillo en contacto con el suelo aplicando las definiciones al comienzo de esta Sección.

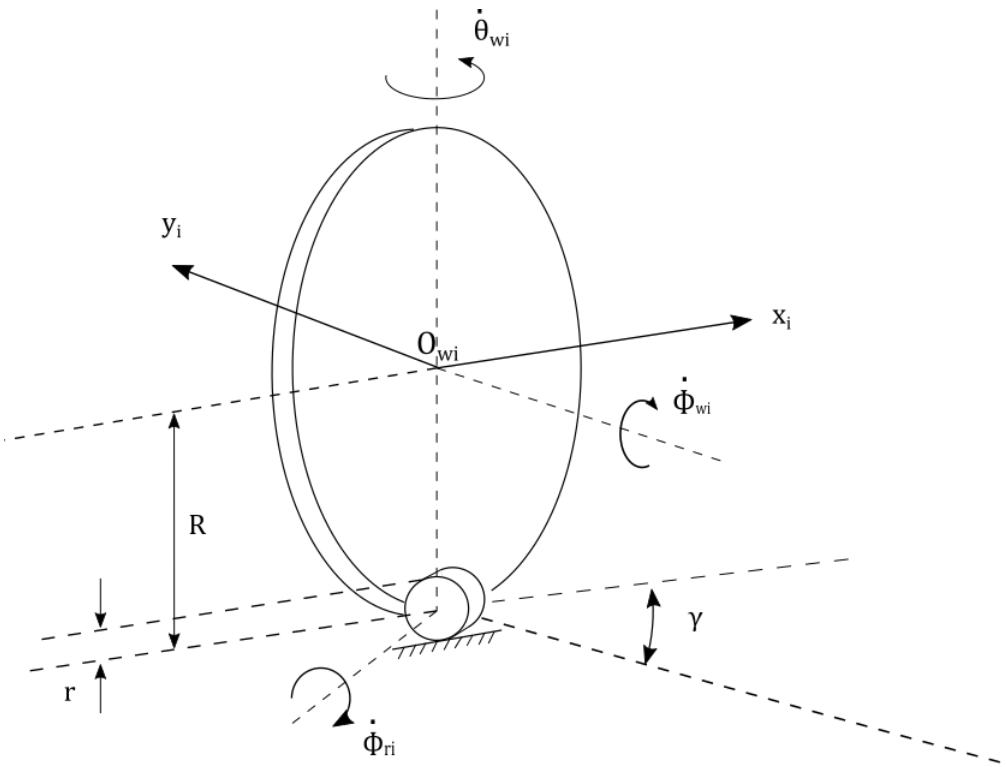


Figura 4.3.2: Cinemática de las ruedas mecanum

A partir de estas definiciones se puede escribir la velocidad de cada rueda en el referencial solidario a la misma a partir de las velocidades angulares de todas las partes de la rueda.

$$\begin{bmatrix} \dot{x_{wi}} \\ \dot{y_{wi}} \\ \dot{\theta_{wi}} \end{bmatrix} = \begin{bmatrix} R & -r.cos(\psi_i) & 0 \\ 0 & r.sen(\psi_i) & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} \dot{\phi_{wi}} \\ \dot{\phi_{ri}} \\ \dot{\theta_{wi}} \end{bmatrix} \tag{4.3.2}$$

Luego, a partir de la Figura 4.2.1, se puede deducir la matriz que permite transformar las velocidades del referencial solidario de cada rueda al referencial solidario del centro del robot.

$$T_{O_{wi}}^{O_r} = \begin{bmatrix} 1 & 0 & dx_{wi}^r \\ 0 & 1 & dy_{wi}^r \\ 0 & 0 & 0 \end{bmatrix} \tag{4.3.3}$$

Siendo dx_{wi}^r y dy_{wi}^r las distancias de la rueda i al centro del robot según los ejes x e y del referencial solidario al robot respectivamente.

Así se puede escribir la relación entre las velocidades de las ruedas en sus propios referenciales y la velocidad del robot en el referencial solidario al mismo.

$$\begin{bmatrix} \dot{x_r} \\ \dot{y_r} \\ \dot{\theta_r} \end{bmatrix} = T^{O_r}_{O_{wi}} . \begin{bmatrix} \dot{x_{wi}} \\ \dot{y_{wi}} \\ \dot{\theta_{wi}} \end{bmatrix} \qquad \Rightarrow \qquad \begin{bmatrix} \dot{x_r} \\ \dot{y_r} \\ \dot{\theta_r} \end{bmatrix} = \begin{bmatrix} 1 & 0 & dx^r_{wi} \\ 0 & 1 & dy^r_{wi} \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} \dot{x_{wi}} \\ \dot{y_{wi}} \\ \dot{\theta_{wi}} \end{bmatrix} \tag{4.3.4}$$

Combinando las ecuaciones 4.3.2 y 4.3.4 se obtiene la relación entre las velocidades angulares de las ruedas en el referencial solidario a las ruedas y la velocidad del robot entero en el referencial solidario al mismo.

$$\begin{bmatrix} \dot{x_r} \\ \dot{y_r} \\ \dot{\theta_r} \end{bmatrix} = \begin{bmatrix} R & -r.cos(\psi_i) & dx^r_{wi} \\ 0 & r.sen(\psi_i) & dy^r_{wi} \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} \dot{\phi_{wi}} \\ \dot{\phi_{ri}} \\ \dot{\theta_{wi}} \end{bmatrix} \tag{4.3.5}$$

Sea J_i matriz de transformación entre las velocidades angulares de las rueda i y las velocidades del robot en su referencial obtenida en la ecuación 4.3.5.Como para $\psi_i = 0$, $|J_i| = 0$, no existen singularidades para las ruedas mecanum. Adicionalmente, como $dim(J_i) = 3$ se deduce que cada rueda mecanum tiene 3 grados de libertad (x, y, θ).

Aplicando las propiedades geométricas del carro:

$$\begin{aligned} dx^r_{w1} &= b & dy^r_{w1} &= a \\ dx^r_{w2} &= b & dy^r_{w2} &= -a \\ dx^r_{w3} &= -b & dy^r_{w3} &= -a \\ dx^r_{w4} &= -b & dy^r_{w4} &= a \end{aligned}$$

Se obtienen las matrices J_i de cada rueda:

$$J_1 = \begin{bmatrix} R & -r/\sqrt{2} & b \\ 0 & r/\sqrt{2} & a \\ 0 & 0 & 1 \end{bmatrix} J_2 = \begin{bmatrix} R & -r/\sqrt{2} & b \\ 0 & -r/\sqrt{2} & -a \\ 0 & 0 & 1 \end{bmatrix} \tag{4.3.6}$$

$$J_3 = \begin{bmatrix} R & -r/\sqrt{2} & -b \\ 0 & r/\sqrt{2} & -a \\ 0 & 0 & 1 \end{bmatrix} J_4 = \begin{bmatrix} R & -r/\sqrt{2} & -b \\ 0 & -r/\sqrt{2} & a \\ 0 & 0 & 1 \end{bmatrix}$$

Usando las ecuaciones 4.3.5 y 4.3.6 se puede obtener la solución de la cinemática inversa siguiendo [25]. Así se obtiene la ecuación 4.3.7, que determina el movimiento del carro relativo a su referencial solidario a partir de las velocidades angulares de las ruedas $\dot{\phi_i}$ $(i = 1, 2, 3, 4)$ expresadas en el referencial solidario a cada rueda.

$$\begin{bmatrix} \dot{x_r} \\ \dot{y_r} \\ \dot{\theta_r} \end{bmatrix} = \frac{R}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ \frac{1}{a+b} & \frac{-1}{a+b} & \frac{-1}{a+b} & \frac{1}{a+b} \end{bmatrix} . \begin{bmatrix} \dot{\phi_1} \\ \dot{\phi_2} \\ \dot{\phi_3} \\ \dot{\phi_4} \end{bmatrix} \tag{4.3.7}$$

Nótese que a partir de la solución obtenida el robot puede moverse en cualquier trayectoria deseada sin cambiar su orientación.

Por último, se puede hallar la matriz de transición del referencial solidario al robot al referencial absoluto, una matriz de rotación de argumento θ, que corresponde a la rotación del robot respecto al referencial absoluto.

$$R(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.3.8}$$

Por último, aplicando las ecuaciones 4.3.7 y 4.3.8 se obtiene la ecuación de movimiento del robot en el referencial absoluto.

$$\begin{bmatrix} \dot{x_q} \\ \dot{y_q} \\ \dot{\theta_q} \end{bmatrix} = \frac{R}{4} \begin{bmatrix} cos(\theta_q) & -sin(\theta_q) & 0 \\ sin(\theta_q) & cos(\theta_q) & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ \frac{1}{a+b} & \frac{-1}{a+b} & \frac{-1}{a+b} & \frac{1}{a+b} \end{bmatrix} . \begin{bmatrix} \dot{\phi_1} \\ \dot{\phi_2} \\ \dot{\phi_3} \\ \dot{\phi_4} \end{bmatrix} \tag{4.3.9}$$

Que desarrollando esta última ecuación se obtiene la forma más compacta.

$$\begin{bmatrix} \dot{x_q} \\ \dot{y_q} \\ \dot{\theta_q} \end{bmatrix} = \frac{R}{4} \begin{bmatrix} cos(\theta_q) - sen(\theta_q) & cos(\theta_q) + sen(\theta_q) & cos(\theta_q) - sen(\theta_q) & cos(\theta_q) + sen(\theta_q) \\ cos(\theta_q) + sen(\theta_q) & -cos(\theta_q) + sen(\theta_q) & cos(\theta_q) + sen(\theta_q) & -cos(\theta_q) + sen(\theta_q) \\ \frac{1}{a+b} & \frac{-1}{a+b} & \frac{-1}{a+b} & \frac{1}{a+b} \end{bmatrix} . \begin{bmatrix} \dot{\phi_1} \\ \dot{\phi_2} \\ \dot{\phi_3} \\ \dot{\phi_4} \end{bmatrix} \tag{4.3.10}$$

4.4. Dinámica:

A continuación se deducirán las ecuaciones dinámicas que determinan el comportamiento del robot. Estas describen la relación entre el movimiento del robot y los factores físicos que lo afectan tales como las fuerzas, masa y energía.

En la Figura 4.4.1 se muestra el diagrama de cuerpo libre del robot, en el que figura el referencial solidario al mismo, que es idéntico al definido en la Sección de cinemática. Adicionalmente, en este modelo se supone que hay una fuerza externa F_{ex} que se aplica al robot en uno de sus lados, a una distancia h del frente del mismo y a un ángulo Ψ_{ex} respecto al lado sobre el cuál se aplica F_{ex}.

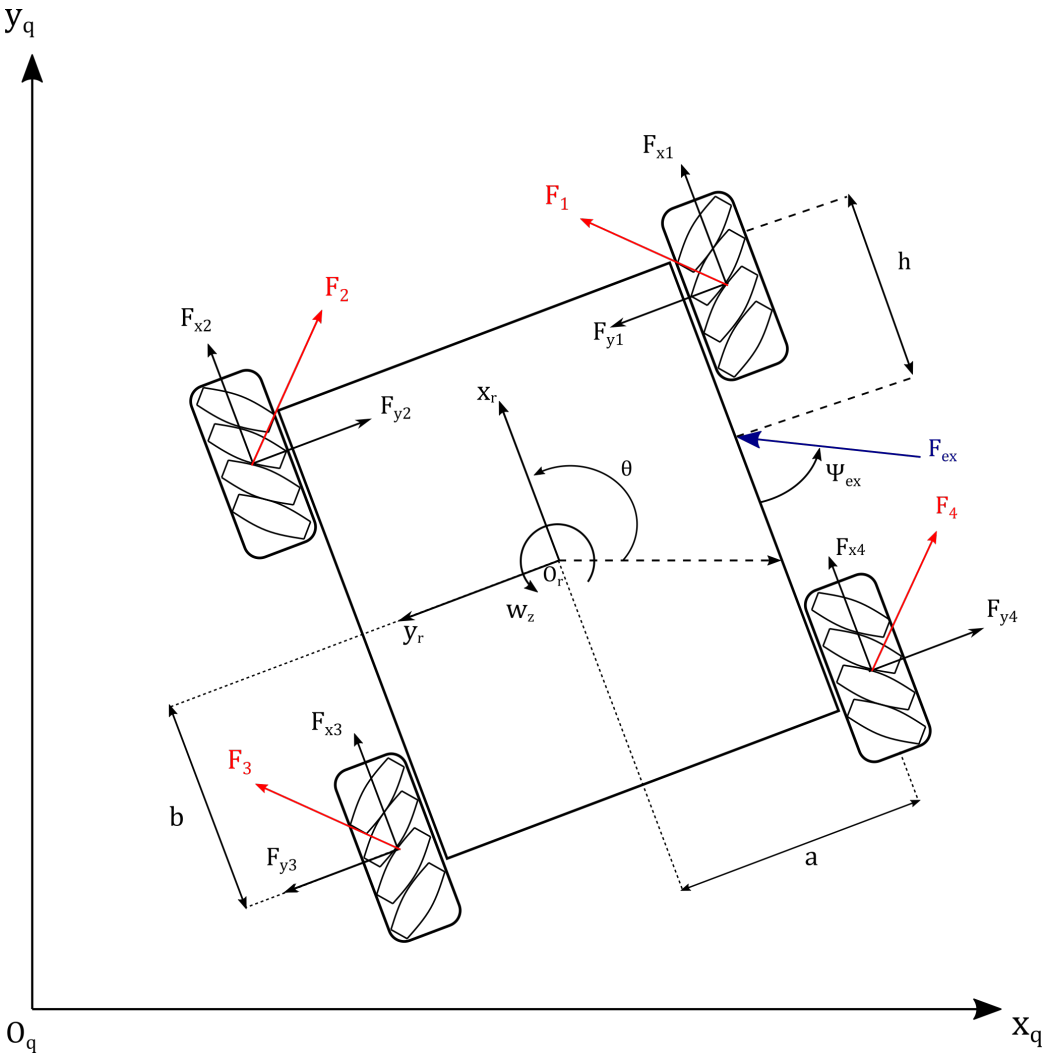


Figura 4.4.1: Modelo de cuerpo libre del robot

Para simplificar el desarrollo, se puede comenzar por analizar la rueda i individualmente.

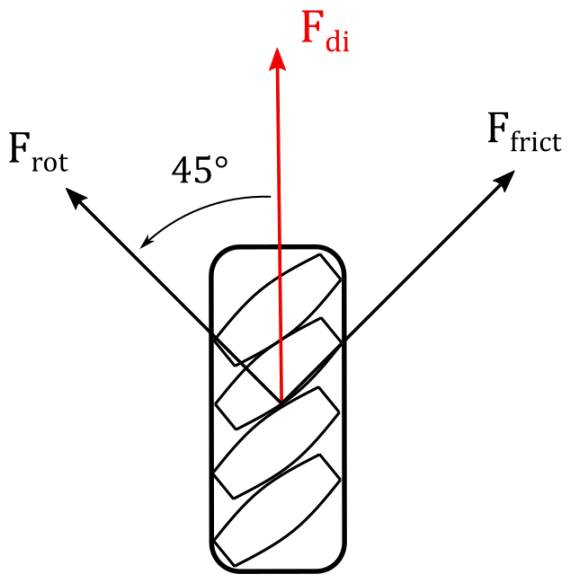


Figura 4.4.2: Diagrama de fuerzas en rueda mecanum

Sea F_{di} la fuerza generada por el motor DC que mueve la rueda i. Tomando un modelo simple de la fuerza producida por el motor DC de cada rueda:

$$F_{di} = \Gamma.u_i - \Delta.R.\dot{\phi}_i \tag{4.4.1}$$

Donde $u_i (i = 1, 2, 3, 4)$ es el voltaje aplicado a cada motor, Γ y Δ son constantes del motor. Gracias a la geometría particular de las ruedas Mecanum, los rodillos descomponen la fuerza F_{di} en dos direcciones, del mismo modo que se explicó en la Sección 4.3: una según el sentido de giro de los rodillos denominada F_{rot_i} y una en dirección perpendicular denominada F_{frict_i}.

Se trabajará bajo la hipótesis de que la fuerza en sentido perpendicular al movimiento de los rodillos, F_{frict_i}, no producirá movimiento en el robot, cancelándose con la fricción estática del piso. Por lo tanto, la fuerza $F_{rot_i} = \frac{F_{di}}{\sqrt{2}}$ será la única que genera un desplazamiento en el robot. En el diagrama de cuerpo libre se le llama F_i.

Esta última, a su vez se descompone nuevamente en dicho diagrama en F_{xi} y F_{yi}, ambas de valor $\frac{F_{di}}{2}$.

Sean M la masa del robot, $P_q = [x_q, y_q]^T$ el vector posición del robot en el marco de referencia estático, y $F_q = [F_{xq}, F_{yq}]$ el vector resultante de todas las fuerzas aplicadas al robot, referidas al referencial estático.

Aplicando la segunda ley de Newton, se puede escribir la ecuación de movimiento del robot respecto al referencial absoluto:

$$M_m.\ddot{P}_q = F_q \qquad \Rightarrow \qquad \begin{bmatrix} M & 0 \\ 0 & M \end{bmatrix} . \begin{bmatrix} \ddot{x_q} \\ \ddot{y_q} \end{bmatrix} = \begin{bmatrix} F_{xq} \\ F_{yq} \end{bmatrix} \tag{4.4.2}$$

Si llamamos $R_r^q(\theta) = \begin{bmatrix} cos(\theta) & -sen(\theta) \\ sen(\theta) & cos(\theta) \end{bmatrix}$ a la matriz de transformación del referencial solidario al robot al referencial absoluto, entonces:

$$\dot{P}_q = R_r^q(\theta).\dot{P}_r \tag{4.4.3}$$

$$F_q = R_r^q(\theta).F_r \tag{4.4.4}$$

A partir de esto, la ecuación 4.4.2 puede reescribirse con 4.4.3 y 4.4.4:

$$M_m(R_r^q(\theta).\ddot{P}_r + R_r^q(\dot{\theta}).\dot{P}_r) = R_r^q(\theta).F_r \tag{4.4.5}$$

Multiplicando de ambos lados por $R_r^q(\theta)^{-1}$:

$$M_m(R_r^q(\theta)^{-1}.R_r^q(\theta).\ddot{P}_r + R_r^q(\theta)^{-1}.R_r^q(\dot{\theta}).\dot{P}_r) = R_r^q(\theta)^{-1}.R_r^q(\theta).F_r \tag{4.4.6}$$

Como $R_r^q(\theta)^{-1} = R_r^q(\theta)^T$ y $R_r^q(\theta)^{-1}.R_r^q(\dot{\theta}) = \dot{\theta} \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$

Por lo tanto se puede reescribir 4.4.6 como:

$$\begin{bmatrix} M & 0 \\ 0 & M \end{bmatrix} . \begin{bmatrix} \ddot{x_r} - \dot{\theta}.\dot{y_r} \\ \ddot{y_r} - \dot{\theta}.\dot{x_r} \end{bmatrix} = F_r \tag{4.4.7}$$

Y F_r como:

$$F_r = \begin{bmatrix} F_{xr} \\ F_{yr} \end{bmatrix} - \begin{bmatrix} \beta.\dot{x_r} \\ \beta.\dot{y_r} \end{bmatrix} + \begin{bmatrix} F_{ex}.sin(\Psi_{ex}) \\ F_{ex}.cos(\Psi_{ex}) \end{bmatrix} \tag{4.4.8}$$

Donde F_{xr} y F_{yr} son las fuerzas totales en las direcciones x_r e y_r respectivamente, β el coeficiente de rozamiento estático, y F_{ex} es la fuerza externa actuando sobre el robot.

Finalmente, se puede escribir la segunda ley de Newton combinando 4.4.7 y 4.4.8.

$$\begin{bmatrix} M & 0 \\ 0 & M \end{bmatrix} . \begin{bmatrix} \ddot{x_r} - \dot{\theta}.\dot{y_r} \\ \ddot{y_r} - \dot{\theta}.\dot{x_r} \end{bmatrix} = \begin{bmatrix} F_{xr} \\ F_{yr} \end{bmatrix} - \begin{bmatrix} \beta.\dot{x_r} \\ \beta.\dot{y_r} \end{bmatrix} + \begin{bmatrix} F_{ex}.sin(\Psi_{ex}) \\ F_{ex}.cos(\Psi_{ex}) \end{bmatrix} \tag{4.4.9}$$

Considerando τ como el momento total sobre el centro de masa del robot, I_r el momento de inercia del mismo respecto al centro de masa. Se puede escribir la segunda cardinal como:

$$I_r.\ddot{\theta} = \tau - \beta_z.\dot{\theta} + F_{ex}.cos(\Psi_{ex}).a - F_{ex}.sen(\Psi_{ex}).(b - h) \tag{4.4.10}$$

Ahora puede reescribirse la ecuación 4.4.9:

$$\begin{bmatrix} M & 0 & 0 \\ 0 & M & 0 \\ 0 & 0 & I_r \end{bmatrix} \begin{bmatrix} \ddot{x_r} - \dot{y_r}.\dot{\theta} \\ \ddot{y_r} - \dot{x_r}.\dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} F_{xr} \\ F_{yr} \\ \tau_r \end{bmatrix} - \beta. \begin{bmatrix} \dot{x_r} \\ \dot{y_r} \\ \dot{\theta} \end{bmatrix} + F_{ex}. \begin{bmatrix} cos(\Psi_{ex}) \\ sen(\Psi_{ex}) \\ cos(\Psi_{ex}).a - sen(\Psi_{ex})(b-h) \end{bmatrix} \tag{4.4.11}$$

A partir del diagrama de cuerpo libre y la ecuación 4.4.1, F_{xr}, F_{yr} y τ pueden reescribirse:

$$F_{xr} = \frac{1}{2}[\Gamma.(u_1 + u_2 + u_3 + u_4) - \Delta.R.(\phi_1 + \phi_2 + \phi_3 + \phi_4)] \tag{4.4.12}$$

$$F_{yr} = \frac{1}{2}\Big[\Gamma.(u_1 - u_2 + u_3 - u_4) - \Delta.R.(\phi_1 - \phi_2 + \phi_3 - \phi_4)\Big] \tag{4.4.13}$$

$$\tau = \frac{a+b}{2}\Big[\Gamma.(u_1 - u_2 - u_3 + u_4) - \Delta.R.(\phi_1 - \phi_2 - \phi_3 + \phi_4)\Big] \tag{4.4.14}$$

Finalmente, aplicando juntas las ecuaciones 4.4.11, 4.4.12, 4.4.13 y 4.4.14 se pueden desarrollar las ecuaciones de movimiento finales del robot:

$$\ddot{x_r} = \frac{1}{2M}\Big[\dot{x_r}(-4\Delta - 2\beta) + 2F_{ex}.cos(\Psi_{ex})\Big] + \frac{1}{2M}\Big[\Gamma(u_1 + u_2 + u_3 + u_4)\Big] \tag{4.4.15}$$

$$\ddot{y_r} = \frac{1}{2M}\Big[\dot{y_r}(-4\Delta - 2\beta) + 2F_{ex}.sen(\Psi_{ex})\Big] + \frac{1}{2M}\Big[\Gamma(u_1 - u_2 + u_3 - u_4)\Big] \tag{4.4.16}$$

$$\ddot{\theta_r} = \frac{1}{I_r}\Big[\dot{\theta_r}.(-2\Delta(a+b)^2 - \beta) + 2F_{ex}.(cos(\Psi_{ex}).a - sen(\Psi_{ex}).(b-h))\Big] + \frac{1}{I_r}\Big[\frac{a+b}{2}.\Gamma.(u_1 - u_2 - u_3 + u_4)\Big] \tag{4.4.17}$$

4.5. Ecuaciones de movimiento:

Una vez deducidas las ecuaciones de la dinámica y cinemática del robot, se pueden obtener las ecuaciones de movimiento por las que se regirá el robot y que se utilizarán para el control.

En el caso del robot diseñado se demostrará que es suficiente con las ecuaciones de la cinemática para el control del mismo. Esto se puede demostrar si el robot cumple con las siguientes condiciones:

- El robot debe moverse sin deslizamiento entre sus ruedas y el piso.

- La respuesta a un escalón de velocidad de los motores debe ser suficientemente rápida para que el tiempo de establecimiento[1] sea insignificante comparado al cambio de velocidades que se le exige al sistema. Esto implica que se puede asegurar que en cada momento se está imponiendo directamente la velocidad de cada rueda, no una fuerza para llegar a dicha velocidad.

Dada la elección de hardware, las pruebas realizadas en el Capítulo 7, en particular en la Sección 7.3 , fueron concluyentes que las hipótesis antes mencionadas se cumplen.

Por lo tanto, las ecuaciones de movimiento del robot en el referencial solidario al robot se pueden resumir en:

$$\begin{bmatrix} \dot{x_r} \\ \dot{y_r} \\ \dot{\theta_r} \end{bmatrix} = \frac{R}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ \frac{1}{a+b} & \frac{-1}{a+b} & \frac{-1}{a+b} & \frac{1}{a+b} \end{bmatrix} . \begin{bmatrix} \dot{\phi_1} \\ \dot{\phi_2} \\ \dot{\phi_3} \\ \dot{\phi_4} \end{bmatrix} \tag{4.5.1}$$

Y para las ecuaciones en el referencial absoluto:

$$\begin{bmatrix} \dot{x_q} \\ \dot{y_q} \\ \dot{\theta_q} \end{bmatrix} = \frac{R}{4} \begin{bmatrix} cos(\theta) - sen(\theta) & cos(\theta) + sen(\theta) & cos(\theta) - sen(\theta) & cos(\theta) + sen(\theta) \\ cos(\theta) + sen(\theta) & -cos(\theta) + sen(\theta) & cos(\theta) + sen(\theta) & -cos(\theta) + sen(\theta) \\ \frac{1}{a+b} & \frac{-1}{a+b} & \frac{-1}{a+b} & \frac{1}{a+b} \end{bmatrix} . \begin{bmatrix} \dot{\phi_1} \\ \dot{\phi_2} \\ \dot{\phi_3} \\ \dot{\phi_4} \end{bmatrix} \tag{4.5.2}$$

Nótese que la ecuación anterior refiere a la velocidad del robot en un instante dado de tiempo, no puede aplicarse la misma ecuación a la posición del mismo dado que hay que integrar la misma y el parámetro θ depende del tiempo. La mejor solución para calcular la posición del robot es realizar una integral numérica de la siguiente manera:

$$\begin{bmatrix} x_q \\ y_q \\ \theta_q \end{bmatrix}_k = \begin{bmatrix} x_q \\ y_q \\ \theta_q \end{bmatrix}_{k-1} + \begin{bmatrix} \dot{x_q} \\ \dot{y_q} \\ \dot{\theta_q} \end{bmatrix}_k .T_s \tag{4.5.3}$$

Esta misma ecuación será la utilizada para el cálculo del desplazamiento dado por la odometría y lograr la localización del robot.

[1] ts, settling time - Tiempo que dura el transitorio entre que el robot parte del reposo y que se alcanza la velocidad final

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 5

Algoritmos de Control, Localización, Mapeo y Navegación

5.1. Control de Velocidad de las Ruedas

5.1.1. Introducción

El seguimiento de trayectoria del robot depende de la posibilidad del robot de mantener y variar su velocidad relativa al entorno de acuerdo a los comandos recibidos. El control de la velocidad del robot está ligado a la velocidad de cada rueda del carro, entonces es necesario controlar cada rueda individualmente.

Como se puede observar en la Figura 5.1.1, resultado de uno de los experimentos detallados en la Sección 7, la diferencia entre la velocidad comandada y la velocidad registrada puede llegar a ser de hasta 10 %.

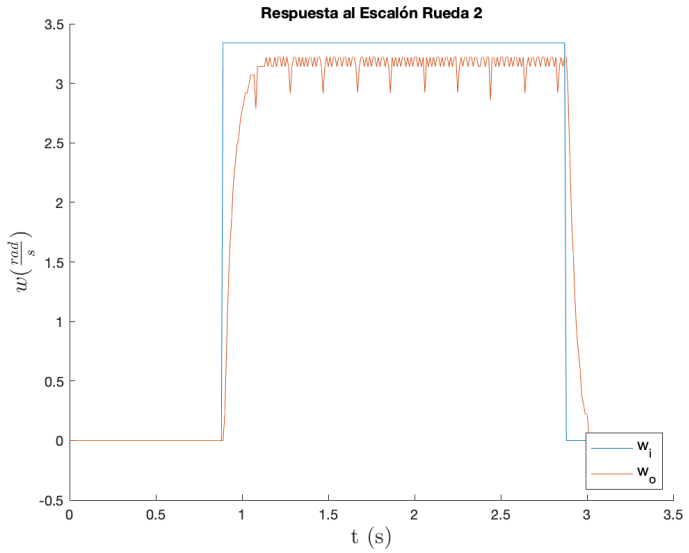


Figura 5.1.1: Respuesta al Escalón Rueda 2

5.1.2. Solución adoptada

Para resolver este problema se optó por utilizar un controlador PID (proporcional, integrador y derivador) para cada rueda. El objetivo de este controlador es minimizar la diferencia entre la velocidad deseada y la velocidad real de la rueda. Entonces, el error que alimenta el controlador del sistema es calculado a través de la diferencia entre la velocidad comandada a la rueda y la velocidad medida por los encoders incorporados, uno por motor.

El sistema de control implementado en cada rueda es el siguiente:

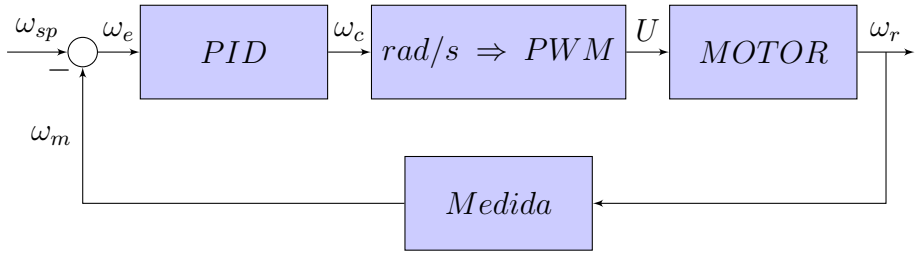


Figura 5.1.2: Diagrama de Control de cada Rueda

En la Figura 5.1.2, ω_{sp} representa la velocidad que debe alcanzar y mantener el motor, y ω_m la velocidad medida a partir de los encoders de cada motor. Luego, $\omega_e = \omega_{sp} - w_m$ es el error según el que actúa el controlador. La salida del controlador es w_c, que está en $\frac{rad}{s}$, y luego es convertida a PWM que es el comando que acepta el driver del motor, mencionado en la Sección 3, para mover la rueda al aplicarle un voltaje continuo en sus bornes. Como fue mencionado anteriormente la velocidad del motor, ω_r, es relevada a partir de los encoders en el eje de cada motor. La relación entre ω_r y ω_m se detalla a continuación.

5.1.2.1. Cálculo de velocidad a partir de cuentas de Encoders

En la descripción del Hardware del sistema en la Sección 3.4.2 se especifica que cada motor genera 64 cuentas por vuelta en su eje interno y además la reducción es 131.25:1, lo que resulta en 8400 cuentas por revolución en el eje externo, el cual es solidario a la rueda.

La velocidad del motor en $\frac{rad}{s}$ se calcula de la siguiente manera:

$$\omega_{medida} = \frac{2\pi}{T_{rueda}}$$

Luego T_{rueda} depende de la cuentas del encoder en un período de tiempo de medida. Donde el período de rotación de la rueda depende del período de medida del encoder y el valor obtenido en ese período:

$$T_{rueda} = \frac{8400 \times \tau_{encoders}}{cuenta\_encoder\_en\_\tau}$$

De esta manera se obtiene la variable de control w_m para comparar con la velocidad deseada por el usuario.

$$\omega_m = \frac{2\pi \times cuenta\_encoder\_en\_\tau}{8400 \times \tau_{encoders}}$$

Para poder crear el sistema de control antes detallado se prosiguió a relevar el comportamiento del sistema. De esta manera se generó un modelo que permitió caracterizar el sistema y simular un esquema de control para ajustar las variables del controlador PID de manera correcta.

5.1.3. Modelado del Sistema

Para poder sintonizar las constantes del PID y controlar de manera correcta cada rueda, se realizaron estudios de todas ellas para determinar qué transferencia $H(s)$ caracteriza al sistema. Por más que tanto los drivers como los motores utilizados son del mismo modelo y fabricante, pueden existir diferencias en los componentes que resulten en sistemas diferentes entre sí. Para asegurar que el modelo del sistema formulado se aproxima a cada rueda, los experimentos se realizaron en los cuatro motores. Una explicación detallada de los experimentos puede encontrarse en la Sección 7.1, esta Sección se centra en los resultados que surgieron a partir de los datos relevados.

Por un lado se observó la salida del sistema en lazo abierto al inyectar una rampa de subida y otra de bajada. Así se identificó la velocidad máxima y mínima que alcanza el motor, así como la zona de histéresis del mismo.

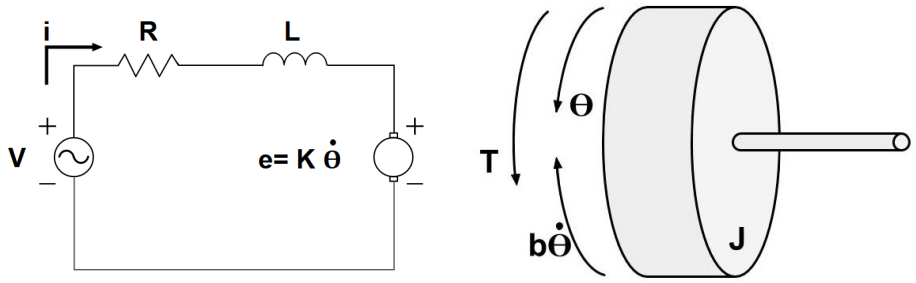
Luego, se estudió la respuesta al escalón para cada rueda. Con estos datos se utilizó una herramienta de Matlab , llamada 'System Identification', que permite ajustar una transferencia modelo a los datos relevados. Para obtener un transferencia fiel a los datos es de suma importancia tener un buen modelo del sistema de estudio.

Se importaron los datos a la aplicación de Matlab y se ajustó un modelo según cada set de datos. El modelo elegido para ajustar los datos se basó en el análisis del sistema de motor cargado expuesto en 'DC Motor Speed: System Modeling'[20] y que se expone a continuación.

Las variables del sistema de estudio en la Figura 5.1.3a y la Figura 5.1.3b, se definen a continuación:

- V: Voltaje en bornes del motor.
- R: Resistencia vista en bornes del motor.
- L: Inductancia vista en bornes del motor.
- e: Caída de Potencial en el motor
- i: Corriente Eléctrica introducida al motor
- T: Torque generado por el motor.

- dotθ: velocidad angular del eje del motor.
- K: Constante del motor.
- J: Momento de Inercia visto desde el eje del motor.
- b: Coeficiente de rozamiento viscoso.



(a) Circuito eléctrico del motor de continua

(b) Interacción Mecánica del motor y el ambiente

Figura 5.1.3: Sistema a Modelar

A partir de la ley de Kirchoff y estudiando la segunda cardinal de Newton del sistema obtenemos:

$$J\ddot{\theta} + b\dot{\theta} = T \tag{5.1.1}$$

$$L\frac{\partial i}{\partial t} + Ri = V - e \tag{5.1.2}$$

Luego, si suponemos que la constante del motor K permite relacionar el torque generado por el motor con la corriente y también deja establecer un vínculo entre la caída de potencial en el motor e y la velocidad del rotor $\dot{\theta}$

$$T = Ki \tag{5.1.3}$$

$$e = K\dot{\theta} \tag{5.1.4}$$

Entonces con todos los vínculos disponibles si combinamos 5.1.1 con 5.1.3 y 5.1.2 con 5.1.4 se obtiene:

$$J\ddot{\theta} + b\dot{\theta} = Ki \tag{5.1.5}$$

$$L\frac{\partial i}{\partial t} + Ri = V - K\dot{\theta} \tag{5.1.6}$$

A continuación realizando la transformada de Laplace a las ecuaciones 5.1.5 y 5.1.6 se llega a :

$$s(Js+b)\Theta(s) = KI(s) \tag{5.1.7}$$

$$(Ls+R)I(s) = V - Ks\Theta(s) \tag{5.1.8}$$

Por último resolviendo el sistema descrito por 5.1.7 y 5.1.8 se llega a la siguiente transferencia:

$$\frac{\dot{\Theta}}{V} = \frac{K}{(Ls+R)(Js+b)+K^2} \tag{5.1.9}$$

Como se explicó a partir de la Figura 5.1.2 la relación entre el voltaje en bornes del motor y la velocidad comandada por la inteligencia se relacionan de forma lineal. Siendo $V = K_{PWM}\theta_c$ y $\dot{\Theta}(s) = \omega_r(s)$ se obtiene:

$$T(s) = \frac{\omega_r}{\omega_c} = \frac{K_{PWM}\ K}{(Ls+R)(Js+b)+K^2} \tag{5.1.10}$$

Entonces la transferencia del sistema $H(s)$ consta de dos polos y sin ceros. La forma genérica puede ser expresada como:

$$T(s) = \frac{A}{s^2+Bs+C} \tag{5.1.11}$$

donde A, B y C son números reales.

Se buscó una aproximación de la transferencia para cada rueda y luego se comparó el comportamiento de cada modelo para los datos de todos los experimentos. Es así que se decidió qué modelo se ajustaba mejor a todos los experimentos según un índice de similitud que genera la aplicación. Para determinar dicho índice la aplicación utiliza la raíz de error cuadrático medio para contrastar las diferencias entre el modelo propuesto y los datos registrados. La transferencia obtenida según los experimentos es la siguiente:

$$T(s) = \frac{3035}{s^2+194{,}8s+3144} \tag{5.1.12}$$

Reescribiendo la transferencia de la siguiente manera :

$$T(s) = \frac{3035}{(s+17{,}759)(s+177{,}041)} \tag{5.1.13}$$

Se aprecia que la transferencia tiene un polo dominante en -17,759, ya que 17,759 $\ll$ 177,041, entonces si la transferencia simplificada es H(s) y verifica $T(s=0) = H(s=0)$, se puede asegurar que la transferencia $H(s)$ es una buena aproximación del comportamiento de T(s). Entonces la transferencia simplificada es:

$$H(s) = \frac{17{,}143}{s+17{,}759} \tag{5.1.14}$$

En la Figura 5.1.4 se muestra el modelo resultante en comparación con los datos experimentales previamente relevados.

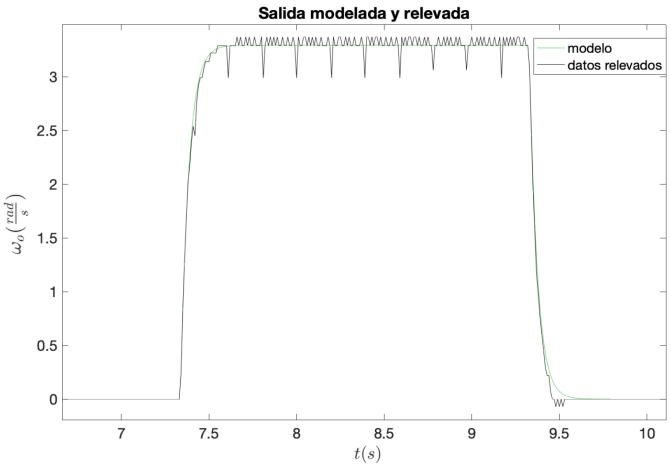


Figura 5.1.4: Modelo contra Datos relevados

Es importante notar e identificar el ruido presente en los datos relevados en la Figura 5.1.4, provenientes de diferentes fuentes. Por un lado, se introduce ruido porque la velocidad de la rueda no es constante, y por otro lado por la naturaleza de la medida del encoder, ya que las medidas de velocidad están cuantizadas. Debido a la velocidad de muestreo, $T_{SampleRate} = 10ms$, las cuentas que realiza el encoder en el período de muestreo son pocas, del orden de decenas. Entonces, puede ocurrir que en un período se cuente un pulso menos porque en el anterior se contó uno de más o viceversa, entonces aparecen picos por debajo de la media y lo mismo por encima.

Para mitigar estos errores se pudo haber aumentado el período de muestreo de los encoders pero se prefirió tener mayor información temporal para ver las variaciones de velocidad a costas de tener mayor ruido de medida.

Si se toma la media, a pesar del ruido, se aprecia que el modelo generado aproxima los rasgos característicos de la respuesta real en la Figura 5.1.4.

5.1.4. Configuración del controlador

A partir de este sistema se prosiguió con dos formas para calibrar el controlador, por un lado se realizó el método convencional, extraído de [36] y por el otro se usó la herramienta 'PID Tuner', que al igual que la herramienta de identificación de sistemas está incluida en el paquete de aplicaciones de Matlab.

5.1.4.1. Diseño del controlador por lugar de las raíces

Para lograr un controlador eficaz pero a la vez sencillo, se incrementó gradualmente la complejidad del controlador hasta lograr satisfacer todas las restricciones de diseño. En este caso se eligieron como restricciones: 2 % de sobretiro, 0.5 segundos de tiempo de subida. Es importante recordar que la función principal del controlador es eliminar el error en régimen, pero también es conveniente mantener

el tiempo de subida del mismo orden del sistema en lazo abierto, o bien disminuirlo para asegurar que las ruedas no deslizan .

Entonces se comenzó por un controlador proporcional con la siguiente transferencia en lazo cerrado:

$$G_{cl_P} = \frac{k_p H(s)}{1 + k_p H(s)} \tag{5.1.15}$$

El problema con este controlador es que no permite eliminar el error en régimen y por lo tanto no cumple con el requerimiento principal impuesto en este diseño. Para cubrir este defecto del controlador proporcional se introdujo una componente integral al controlador para lidiar con el error en régimen.

De esta manera el controlador propuesto para el sistema es de la forma:

$$C(s) = \frac{k_p s + k_i}{s} \tag{5.1.16}$$

A partir de los requerimientos de tiempo de respuesta y sobretiro se estudió el lugar geométrico de las raíces, es decir, se evaluaron los puntos del plano complejo que satisfacen $1+C(s)H(s) = 0$ y se ajustó $C(s)$ para llevar la respuesta del sistema en lazo cerrado a la forma más lenta posible y con menos sobretiro. El controlador puede ser reformulado como :

$$C(s) = \frac{K(1 + Zs)}{s} \tag{5.1.17}$$

Entonces, las variables que permiten determinar el diseño a partir del lugar geométrico de las raíces son la posición del cero $\frac{-1}{Z}$ del controlador y el factor K que mueve a los polos de la transferencia de lazo abierto dentro del lugar geométrico definido.

En la Figura 5.1.5 los marcadores en forma de círculo y cruz, esta última en el origen, representan el polo y el cero impuestos por el controlador $C(s)$ respectivamente. Mientras que la otra cruz, que se encuentra sobre el eje real negativo, es la del sistema de estudio. En la Figura 5.1.5 se muestran las zonas más claras como los lugares admisibles para las raíces del lazo cerrado, mientras que las zonas sombreadas no verifican las condiciones explicadas al comienzo de esta Sección. El lugar geométrico de las raíces se ajustó con el cero del controlador para que las curvas pertenecientes a la zona imaginaria sean tangentes a la condición de sobretiro como puede apreciarse en la Figura 5.1.5.

Por otro lado se varió K para que los polos del sistema en lazo cerrado, representados en la Figura 5.1.5 por los puntos a los lados de la abscisa 10 en el eje real, determinen la respuesta más lenta posible dentro de los requerimientos elegidos.

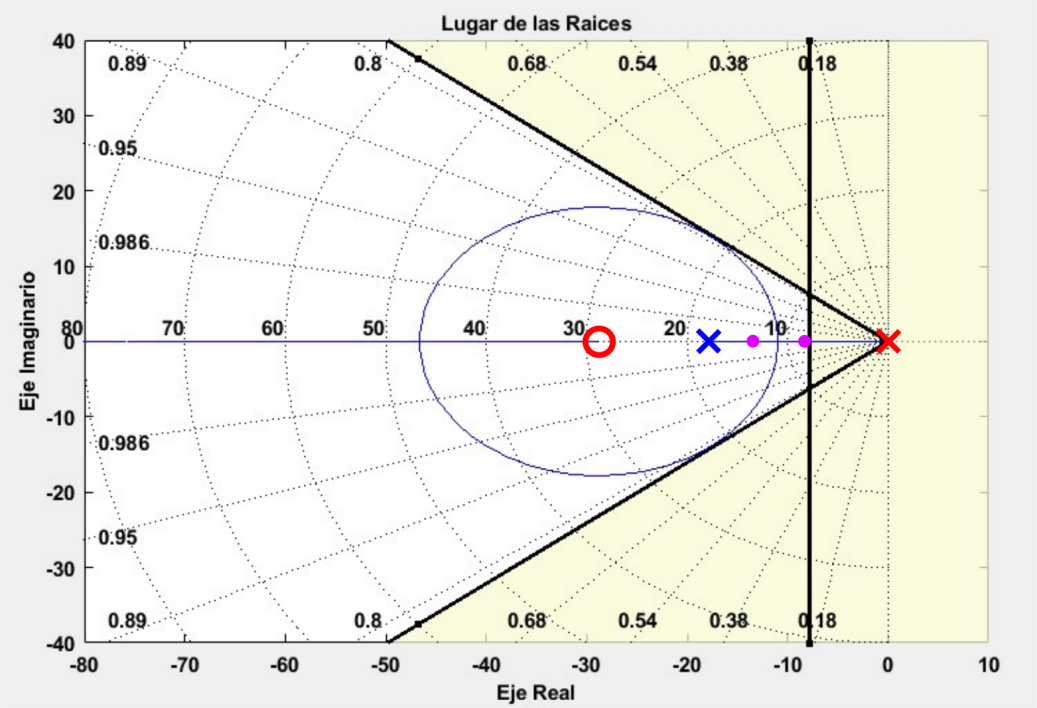


Figura 5.1.5: Controlador PI: Lugar de las Raíces.

La Figura 5.1.5 muestra la configuración final del PID según este método y es dada para el controlador con $K = 6{,}35$ y $Z = 0{,}035$.

Entonces en el controlador PID obtenido por este método es tal que $k_p = 0,222$ y $k_i = 6{,}35$.Con esta configuración el robot responde correctamente y además no se generan aceleraciones bruscas.

La respuesta al escalón para esta configuración se presenta en la Figura 5.1.6

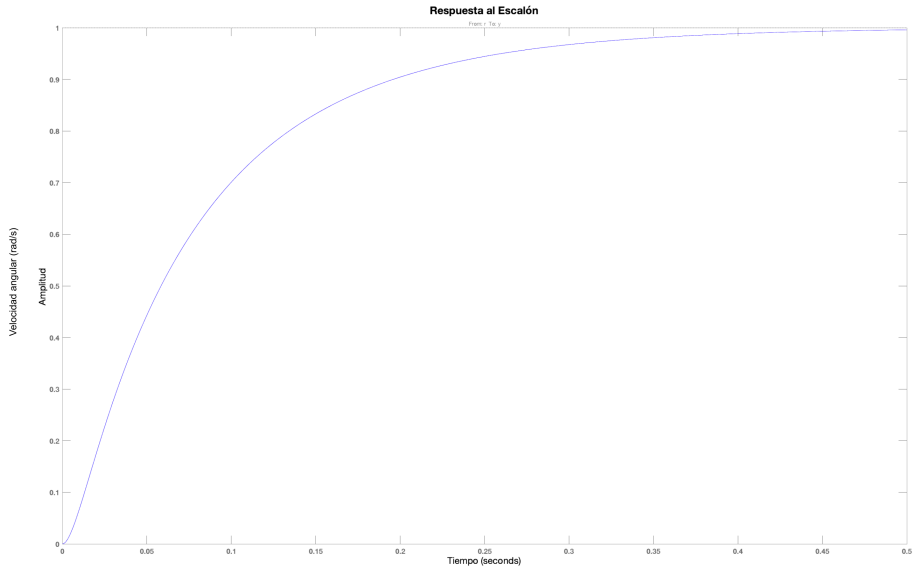


Figura 5.1.6: Respuesta al Escalón con controlado Diseñado

Con el fin de contrastar estos resultados con otras posibles configuraciones del controlador se acudió a la herramienta de MATLAB, que automatiza el proceso de diseño.

5.1.4.2. PID tuner

La herramienta de Matlab permite determinar el valor de las constantes del controlador PID necesario, dados: modelo de la planta a controlar, tiempo de subida deseado e índice de robustez en la transición. Esta última variable pretende controlar el sobretiro y tiempo de asentamiento.

Se realizaron tres configuraciones diferentes para probar su comportamiento en el robot y contrastarlo con el método descripto en la sección anterior. Para las tres configuraciones se ingresó el modo más robusto posible para el comportamiento transitorio y se fue variando el tiempo de subida deseado. Los tiempos elegidos fueron 0.05s , 0.1s y 0.5s. La Tabla 5.1 indica la configuración obtenida de las variables del controlador para cada tiempo de respuesta según el modelo que se obtuvo del motor.

Tiempo de Respuesta:	**0.05s**	**0.1s**	**0.5s**
K_p	2.092	1.068	0.215
K_i	29.934	17.225	4.032
K_d	0.009	0	0

Tabla 5.1: Configuraciones obtenidas según tiempo de respuesta

5.1.5. Controlador y configuración utilizados

Se evaluaron las cuatro configuraciones obtenidas en las secciones previas sobre el sistema completo, es decir con el robot realizando movimientos básicos y a partir de los resultados se decidió utilizar la configuración obtenida a partir del estudio del lugar geométrico de las raíces, debido a su andar continuo y sin aceleraciones bruscas, evitando el deslizamiento de las ruedas.

En la Sección 7 se encuentran los resultados de las configuraciones de PID. En la Figura 5.1.7 se observa la respuesta controlada comparada con el escalón de entrada para el controlador PID configurado con $k_p = 0,222$, $k_i = 6{,}35$ y $k_d = 0$.

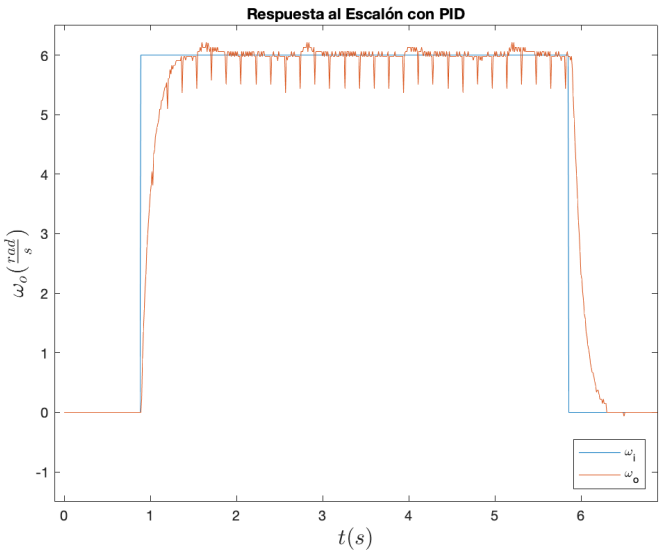


Figura 5.1.7: Respuesta al Escalón PID configuración B

5.2. Fusión Sensorial

5.2.1. Introducción:

La fusión de datos consiste en combinar las observaciones de diferentes fuentes (o una misma fuente en distintos tiempos), junto con modelos o información previa, para obtener información más significativa y certera que la dada por cada una individualmente, y así obtener una descripción más robusta y completa del ambiente o proceso de interés.

Este proceso es análogo a como los humanos y animales usan una combinación de múltiples sentidos, experiencias y la habilidad de razonar para tomar decisiones a partir de los estímulos externos.

La fusión de datos tiene aplicaciones en áreas como: tareas de monitoreo, control de procesos, sistemas de vigilancia y sistemas de información.

En esta Sección se detallará la aplicación de fusión de datos en el robot construido desde un enfoque probabilístico basándose en bibliografía especializada [21], [10], [6], [8] y [14].

En los sistemas robóticos en general, dado que se trabaja exclusivamente con sensores como fuentes de información, se denomina a estos métodos de fusión de datos como 'fusión de datos multisensorial' o simplemente 'fusión sensorial', como se referirá de aquí en adelante.

La fusión sensorial es aplicada cuando no alcanza con un solo sensor para medir la magnitud deseada, o si la incertidumbre asociada a la medida es más de la tolerable por la aplicación. Cobra especial importancia en sistemas robóticos donde se cuenta con un gran número de fuentes de información y es necesario poder combinar y filtrar estos datos para tomar decisiones.

Esta técnica juega un papel central en el desarrollo de software para el robot, ya que permite estimar con gran precisión la posición y orientación del robot en todo momento a partir de sensores que proveen información parcial y ruidosa.

En la práctica, lograr la localización de un robot móvil sin aplicar técnicas de fusión sensorial no brinda resultados aceptables por el ruido asociado a la medición con cualquier tipo de sensor y la información limitada que solo un sensor puede proveer. En la Figura 5.2.1 se puede observar un ejemplo de localización de un robot móvil terrestre usando solamente la odometría para lograrlo [6].

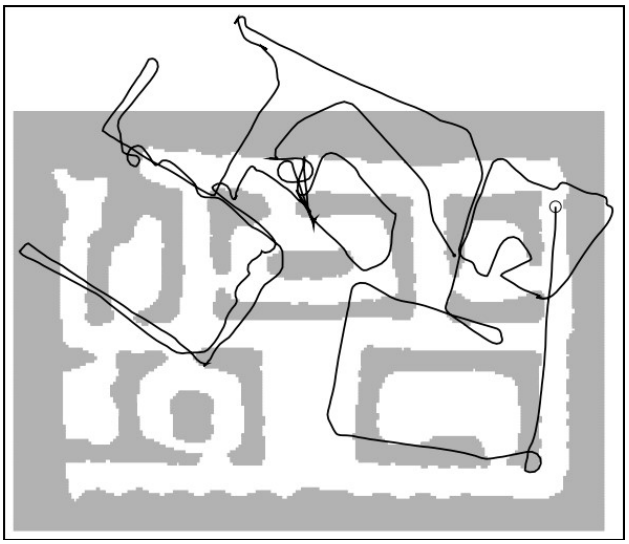


Figura 5.2.1: Localización sin aplicar fusión sensorial[14]

En este caso se comienza con una posición inicial conocida, marcada en la esquina superior derecha de la Figura 5.2.1, y a partir de esta se integran las medidas de velocidad dadas por la odometría para calcular el desplazamiento. Al estarse integrando las medidas, cualquier error sistemático, por más pequeño que sea, se integra en el tiempo y crece linealmente a medida que avanza el robot.

Como puede apreciarse, la trayectoria registrada rápidamente se desvía de la que habría realizado, comparando con el mapa de fondo. Esto se debe a que la odometría marca que la orientación del robot es levemente distinta a la real en todo momento. El error se integra con el tiempo a medida que se recorre el ambiente y la trayectoria se desvía conforme el robot continúa moviéndose.

Este ejemplo ilustra la necesidad de aplicar fusión sensorial a los sistemas robóticos móviles, a pesar de que en teoría no sea necesario, como sería el caso si la odometría fuera perfecta.

Tipos de fusión sensorial:

- Entre sensores: Todos los sensores miden la misma magnitud en las mismas condiciones. Se aplica para reducir la incertidumbre que tendría un solo sensor respecto a una magnitud específica.

- Entre atributos: Todos los sensores miden diferentes magnitudes asociadas a la misma situación experimental. Permite eliminar errores sistemáticos que ocurren en sensores que utilizan un mismo principio físico para medir o miden una misma característica del objeto de estudio.

- Entre dominios: Todos los sensores miden lo mismo pero en distintos dominios y escalas. Se aplica cuando ya sea el rango de medida o la precisión de un sensor no son suficientes para medir la magnitud deseada. En lugar de

esto se utilizan múltiples sensores de distintas escalas para abarcar todo el rango de medida necesario con buena precisión.

- A lo largo del tiempo: Para un mismo sensor se combinan las medidas actuales con las anteriores. Para poder tener una representación precisa del estado del sistema de estudio, se debe poder interpretar medidas históricas que ayudan a determinar la evolución del sistema a lo largo del tiempo.

Configuración de sensores:

- Complementaria: Los sensores no dependen uno del otro, pero combinados forman una imagen más completa.
- Competitiva: Cada sensor entrega una medida independiente de la misma propiedad.
- Cooperativa: Se usa información provista por dos o más sensores independientes para obtener información que no se lograría con estos individualmente.

5.2.2. Teoría estadística:

Los métodos de probabilísticos de fusión sensorial están principalmente basados en la estadística Bayesiana para combinar información previa con las observaciones. En la práctica esto puede realizarse a través de varios métodos como el uso de filtros de Kalman o métodos de Monte Carlo.

La regla de Bayes provee un método para hacer inferencias acerca de un sistema o proceso de interés descrito por un estado x, dada una observación z.

Esto requiere que la relación entre x y z esté dada por una distribución de probabilidad conjunta $P(x, z)$, y que se conozcan $P(x)$, la probabilidad de que el proceso se encuentre en el estado x, y $P(z)$, la probabilidad de que se realice la observación z.

La regla de Bayes permite calcular $P(x|z)$, la probabilidad de que el proceso de estudio se encuentre en el estado x, sabiendo que se realizó una observación z. Esta regla puede expresarse como:

$$P(x, z) = P(x|z).P(z) = P(z|x).P(x) \quad \Rightarrow \quad P(x|z) = \frac{P(z|x).P(x)}{P(z)} \tag{5.2.1}$$

Para la aplicación en el robot, se puede considerar que la probabilidad condicional $P(z|x)$ juega el rol de modelo de sensor. Se construye fijando el valor de x=x para averiguar la densidad de probabilidad $P(z|x =\text{x})$ en z. Se dice que funciona como modelo de sensor porque dado un estado fijo x, codifica cual debería ser la observación realizada por el sistema.

Del mismo modo, cuando el modelo del sensor es usado, se tiene $z =$z fijo, y una función de probabilidad $P(z =\text{z}|x)$ en x es inferida.

Para el caso multisensorial se requiere que haya independencia condicional, es decir [35]:

$$P(z_1, z_2, ..., z_n|x) = P(z_1|x).P(z_2|x)...P(z_n|x) = \prod_{i=1}^{n} P(z_i|x) \tag{5.2.2}$$

De modo que si se aplica la regla de Bayes (ecuación 5.2.1) al caso multisensorial, se obtiene:

$$P(x|Z^n) = C.P(x).\prod_{i=1}^{n} P(z_i|x) \tag{5.2.3}$$

Con C una constante normalizadora y Z^n las n observaciones. Esto permite deducir que la probabilidad a posteriori, $P(x|Z^n)$, es proporcional al producto de las probabilidades de cada una de las observaciones $P(z_i|x)$ y la probabilidad a priori $P(x)$.

A partir de 5.2.3, la forma recursiva de la regla de Bayes es:

$$P(x|Z^k) = \frac{P(z_k|x).P(x|Z^{k-1})}{P(z_k|Z^{k-1})} \tag{5.2.4}$$

La ecuación 5.2.4 que permite computar y almacenar solamente la densidad a posteriori $P(x|Z^{k-1})$, que contiene un resumen de toda la información anterior. Esto permite iterar para obtener la siguiente densidad a posteriori solamente a partir del paso anterior ([35], parte C, Sección 35.1.1).

Cuando una nueva observación $P(z_k|x)$ es recibida, la que antes era la probabilidad a posteriori pasa a ser la nueva probabilidad a priori, y el producto de ambas, una vez normalizado, se convierte en la nueva probabilidad a posteriori $P(x|Z^k)$.

De este modo se estima el estado x de un proceso de forma iterativa, recibiendo observaciones en cada paso y actualizando la probabilidad de que el proceso se encuentre en un cierto estado.

5.2.3. Filtro de Kalman

Los filtros de Kalman son ampliamente utilizados como algoritmos para fusión sensorial en robótica móvil. Estos son estimadores de estados que evolucionan en el tiempo, y lo hacen combinando sucesivamente las observaciones del estado con las predicciones del algoritmo mismo dadas por un modelo explícito de transición de estados.

En el caso de un sistema lineal con distribución Gaussiana, el filtro de Kalman es el mejor estimador lineal insesgado de estados, BLUE[1] por sus siglas en inglés. Esto significa que el filtro de Kalman es el estimador óptimo de estados, más que cualquier algoritmo en estas condiciones, ya que minimiza el error cuadrático medio del estado estimado.

[1]Best Linear Unbiased Estimator

El filtro de Kalman requiere un modelo lineal estadístico explícito de cómo el vector de estados $x(t)$ evoluciona a lo largo del tiempo, y de cómo las observaciones $z(t)$ se relacionan con dicho vector de estados. Dichos modelos se expresan de la forma descrita en las ecuaciones 5.2.5 y 5.2.6 para un sistema de tiempo discreto. Es en esta etapa que se modela el proceso matemáticamente y determinan los parámetros del filtro.

Estos modelos describen la evolución del vector de estados y las observaciones considerando un factor aditivo de ruido Gaussiano, w_k y v_k respectivamente, que representan las incertidumbres del modelado y en la medida.

Modelado:

$$x_k = A.x_{k-1} + B.u_k + w_k \tag{5.2.5}$$

$$z_k = C.x_k + v_k \tag{5.2.6}$$

A continuación se definen los parámetros que componen los modelos del sistema:

- La matriz A es la matriz de transición de estados, que explicita el modelo que describe la transición de estados del sistema. Esta permite, dado el sistema sin entradas, calcular el estado del sistema x_k solamente a partir de x_{k-1}.
- La matriz B representa el aporte de la entrada del sistema al nuevo estado.
- La matriz C modela el funcionamiento de los sensores, las fuentes de observaciones del sistema, y permiten determinar cuál será el vector de observaciones z_k dado un vector de estados x_k.
- La ganancia de Kalman, G_k, es una matriz que se calcula para que minimice la covarianza de la estimación a posteriori. En términos prácticos, la ganancia de Kalman determina en qué medida será el siguiente estado determinado por el modelo de transición de estados del sistema o por las observaciones de los sensores.
- El vector w_k conforma el ruido de proceso. Este es el que representa los errores de modelado del sistema. En una aplicación real, donde se simplifican los modelos para que sean lineales por ejemplo, el ruido de proceso marca la diferencia entre como el modelo predice la evolución de los estados y como realmente esta se da. Este es un vector de dimensión igual al vector de estados x_k y representa el error en cada coordenada individual, y caracteriza por una distribución gaussiana de media nula y con varianza Q_k en el instante k.
- El vector v_k es el ruido de medida, un parámetro característico de los sensores utilizados para las observaciones del sistema, que tienen un grado limitado de precisión, y por lo tanto tampoco serán exactamente acordes a su modelo. Los elementos de este vector representan el error de un sensor en cada una de sus coordenadas, y caracteriza por una distribución gaussiana de media nula y con varianza R_k en el instante k.

Una vez definido el modelo del sistema, se puede comenzar a ejecutar el filtro. Este algoritmo consta de 2 etapas, la predicción y la actualización, que se ejecutan de manera iterativa.

En la primera se realiza una estimación del estado actual, basada enteramente en el estado anterior, usando el modelo del sistema. Luego, una vez que se realizan las observaciones del estado actual, se puede computar la actualización de la estimación del estado actual (realizada en la etapa de predicción).

Este proceso se repite en cada paso del sistema para ir estimando el estado y obteniendo la covarianza de esa estimación. El resultado del algoritmo es la estimación del estado en cada paso $\hat{x_k}$ y la covarianza de dicha estimación P_k, por lo que se puede decir que el filtro de Kalman produce como resultado una función de densidad de probabilidad de media $\hat{x_k}$ y covarianza P_k [6].

A continuación se detalla el algoritmo:

Predicción:

$$\hat{x}_{k|k-1} = A.\hat{x}_{k-1|k-1} + B.u_k \tag{5.2.7}$$

$$P_{k|k-1} = A.P_{k-1|k-1}.A^T + Q_{k-1} \tag{5.2.8}$$

Actualización:

$$G_k = P_{k|k-1}.C^T(C.P_{k|k-1}.C^T + R)^{-1} \tag{5.2.9}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + G_k(z_k - C.\hat{x}_{k|k-1}) \tag{5.2.10}$$

$$P_{k|k} = (I - G_k.C).P_{k|k-1} \tag{5.2.11}$$

Es importante que al aplicar un filtro de Kalman se conozcan de forma certera las covarianzas de las observaciones y del proceso, ya que los filtros de Kalman son estimadores óptimos de estados si solo si el modelo implementado es correcto. En caso contrario, se puede probar que el filtro de Kalman convergerá a un resultado erróneo al tener datos estadísticos equivocados.

Finalmente, la Figura 5.2.2 muestra un diagrama funcional de la implementación del filtro de Kalman.

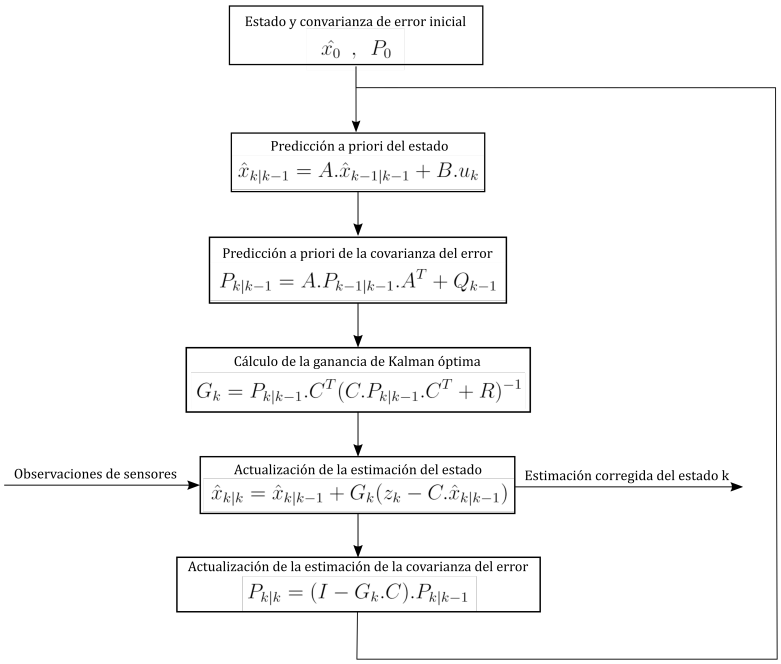


Figura 5.2.2: Diagrama funcional del filtro de Kalman

5.2.4. Extensión a sistemas no lineales:

La hipótesis principal del filtro de Kalman es que se aplica a sistemas lineales exclusivamente, lo que lo hace muy restrictivo para el uso en aplicaciones prácticas, ya que difícilmente se puedan modelar procesos complejos de forma lineal sin realizar simplificaciones o asunciones que difieren de manera notoria con el sistema de estudio.

En estos casos es que se puede aplicar una variante del algoritmo anterior llamado el Filtro de Kalman Extendido, o EKF por sus siglas en inglés. Este puede utilizarse como estimador óptimo de sistemas Gaussianos y no lineales pero linealizables a tramos, es decir que dado un proceso $x(t)$, existe un entorno para todo t tal que se puede aproximar $x(t)$ a un sistema lineal.

El EKF se adapta a sistemas no lineales alterando el algoritmo original de la siguiente manera:

Modelado:

$$x_k = f(x_{k-1}, u_k) + w_k \tag{5.2.12}$$

$$z_k = h(x_k) + v_k \tag{5.2.13}$$

Predicción:

$$\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_k) \tag{5.2.14}$$

$$P_{k|k-1} = F_{k-1}.P_{k-1|k-1}.F_{k-1}^T \tag{5.2.15}$$

Actualización:

$$G_k = P_{k|k-1}.H_k^T(H_k.P_{k|k-1}.H_k^T + R)^{-1} \tag{5.2.16}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + G_k(z_k - h(\hat{x}_{k|k-1})) \tag{5.2.17}$$

$$P_{k|k} = (I - G_k.H_k).P_{k|k-1} \tag{5.2.18}$$

En este algoritmo, el modelo del sistema está dado por la función $f(x, u)$ que engloba la evolución del sistema, tanto en función del estado anterior x, como de la entrada del mismo, u. La solución al problema de la no linealidad surge haciendo una linealización por tramos del modelo en cada instante de tiempo para predecir el estado en el instante siguiente. Esta linealización se representa en la matriz F_k:

$$F_k = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{k-1|k-1}, u_k} \tag{5.2.19}$$

La matriz definida en la ecuación 5.2.19 es la matriz jacobiana de la función f evaluada en la estimación del estado $k-1$, dada la observación realizada en el instante $k-1$, y evaluada también en la entrada del estado k. Por lo tanto se obtiene una linealización de la función $f(x_{k-1}, u_k)$ en el instante k. Esta matriz tiene su análogo en el filtro de Kalman original con la matriz A.

Del mismo modo, se linealiza la función $h(x_k)$ para adecuar el algoritmo a modelos de sensores no lineales:

$$H_k = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_{k|k-1}} \tag{5.2.20}$$

Obteniendo así la matriz jacobiana de la función h evaluada en la estimación del estado k dada la observación realizada en el instante $k-1$.

5.2.5. Diseño del algoritmo:

La localización precisa del robot en todo momento es de vital importancia para las misiones propuestas para el mismo, por eso se buscó implementar un algoritmo de fusión sensorial que fusionara la información de los sensores para reducir la incertidumbre y prevenir problemas de mapeo y navegación.

A partir de los sensores con los que cuenta el robot para localización, se puede diseñar un filtro de Kalman extendido para fusionar la odometría de las cuatro ruedas y la información del giróscopo. Cabe aclarar que se trata de un EKF porque, como se deduce en el Capítulo 4, las ecuaciones del robot describen un sistema no lineal, al igual que los modelos de los sensores como se verá a continuación.

En primer lugar se debe definir un vector de estados, las variables del robot que se desea estimar y corregir a través de la fusión sensorial. En este caso se eligió el vector

$$x_k = \begin{bmatrix} x_{qk} \\ y_{qk} \\ \theta_{qk} \\ \dot{x}_{qk} \\ \dot{y}_{qk} \\ \dot{\theta}_{qk} \end{bmatrix} \tag{5.2.21}$$

A partir de las ecuaciones de movimiento del robot deducidas en el Capítulo 4, se puede obtener la función $f(x_{k-1}, u_k)$ característica del sistema para aplicar en la fusión sensorial.

$$f(x_{k-1}, u_k) = \begin{bmatrix} x_{qk} \\ y_{qk} \\ \theta_{qk} \\ \dot{x}_{qk} \\ \dot{y}_{qk} \\ \dot{\theta}_{qk} \end{bmatrix} \tag{5.2.22}$$

$$\begin{bmatrix} x_{qk} \\ y_{qk} \\ \theta_{qk} \\ \dot{x}_{qk} \\ \dot{y}_{qk} \\ \dot{\theta}_{qk} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & Ts & 0 & 0 \\ 0 & 1 & 0 & 0 & Ts & 0 \\ 0 & 0 & 1 & 0 & 0 & Ts \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x_{qk-1} \\ y_{qk-1} \\ \theta_{qk-1} \\ \dot{x}_{qk-1} \\ \dot{y}_{qk-1} \\ \dot{\theta}_{qk-1} \end{bmatrix} +$$

$$\frac{R}{4} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ cos(\theta) - sen(\theta) & cos(\theta) + sen(\theta) & cos(\theta) - sen(\theta) & cos(\theta) + sen(\theta) \\ cos(\theta) + sen(\theta) & -cos(\theta) + sen(\theta) & cos(\theta) + sen(\theta) & -cos(\theta) + sen(\theta) \\ \frac{1}{a+b} & \frac{-1}{a+b} & \frac{-1}{a+b} & \frac{1}{a+b} \end{bmatrix} \begin{bmatrix} \dot{\phi}_{1k} \\ \dot{\phi}_{2k} \\ \dot{\phi}_{3k} \\ \dot{\phi}_{4k} \end{bmatrix} \tag{5.2.23}$$

Luego, se puede modelar los sensores del robot a fusionar. El giróscopo será utilizado solamente para medir la velocidad angular del robot en el eje z, por lo que el modelado de este sensor es muy sencillo:

$$h_{gyro} = \phi_q \tag{5.2.24}$$

Para modelar la odometría del carro a partir de su velocidad relativa se debe invertir la ecuación 4.5.2, ya que se desea obtener una ecuación de la forma:

$$\begin{bmatrix} \dot{\phi}_1 \\ \dot{\phi}_2 \\ \dot{\phi}_3 \\ \dot{\phi}_4 \end{bmatrix} = \Theta . \begin{bmatrix} \dot{x_q} \\ \dot{y_q} \\ \dot{\theta_q} \end{bmatrix} \tag{5.2.25}$$

Sin embargo, la ecuación 5.2.25 no se puede obtener directamente de la 4.5.2, dado que esta última no es cuadrada y por lo tanto tampoco es invertible.

Por eso, antes se debe agregar una condición más a la matriz para que esta sea invertible. Se deduce esta condición planteando que se desea que el movimiento del robot sea óptimo, es decir que no se produzcan movimientos en las ruedas que se cancelen entre sí o causen que las ruedas deslicen.

Cualquier vector de velocidades angulares de las ruedas que haga que se cancelen todas mutuamente debe pertenecer al núcleo de la transformación dada en 4.5.2, porque sería un vector v tal que $\Theta.v = 0$.

Ahora, para imponer esta condición de no deslizamiento dentro de la ecuación 4.5.2, se puede plantear que:

$$v \in ker(\Theta) \Rightarrow v = \begin{bmatrix} 1 & 1 & -1 & -1 \end{bmatrix} \tag{5.2.26}$$

De este modo la ecuación 4.5.2 queda:

$$\begin{bmatrix} \dot{x_q} \\ \dot{y_q} \\ \dot{\theta_q} \\ 0 \end{bmatrix} = \frac{R}{4} \begin{bmatrix} cos(\theta_q) - sen(\theta_q) & cos(\theta_q) + sen(\theta_q) & cos(\theta_q) - sen(\theta_q) & cos(\theta_q) + sen(\theta_q) \\ cos(\theta_q) + sen(\theta_q) & -cos(\theta_q) + sen(\theta_q) & cos(\theta_q) + sen(\theta_q) & -cos(\theta_q) + sen(\theta_q) \\ \frac{1}{a+b} & \frac{-1}{a+b} & \frac{-1}{a+b} & \frac{1}{a+b} \\ 1 & 1 & -1 & -1 \end{bmatrix} \cdot \begin{bmatrix} \dot{\phi_1} \\ \dot{\phi_2} \\ \dot{\phi_3} \\ \dot{\phi_4} \end{bmatrix} \tag{5.2.27}$$

Inviertiéndola, se obtiene la ecuación que permite obtener las velocidades de los motores a partir de la velocidad del robot en un referencial absoluto.

$$\begin{bmatrix} \dot{\phi_1} \\ \dot{\phi_2} \\ \dot{\phi_3} \\ \dot{\phi_4} \end{bmatrix} = \frac{1}{R} \begin{bmatrix} cos(\theta_q) - sen(\theta_q) & cos(\theta_q) + sen(\theta_q) & a+b & 1 \\ cos(\theta_q) + sen(\theta_q) & -cos(\theta_q) + sen(\theta_q) & -a-b & 1 \\ cos(\theta_q) - sen(\theta_q) & cos(\theta_q) + sen(\theta_q) & -a-b & -1 \\ cos(\theta_q) + sen(\theta_q) & -cos(\theta_q) + sen(\theta_q) & a+b & -1 \end{bmatrix} \cdot \begin{bmatrix} \dot{x_q} \\ \dot{y_q} \\ \dot{\theta_q} \\ 0 \end{bmatrix} \tag{5.2.28}$$

Por último, se puede escribir la función $h(x_k)$ del filtro de Kalman extendido:

$$h(x_k) = \begin{bmatrix} \omega_{gyro} \\ O_1 \\ O_2 \\ O_3 \\ O_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & cos(\theta_q) - sen(\theta_q) & cos(\theta_q) + sen(\theta_q) & a+b \\ 0 & 0 & 0 & cos(\theta_q) + sen(\theta_q) & -cos(\theta_q) + sen(\theta_q) & -a-b \\ 0 & 0 & 0 & cos(\theta_q) - sen(\theta_q) & cos(\theta_q) + sen(\theta_q) & -a-b \\ 0 & 0 & 0 & cos(\theta_q) + sen(\theta_q) & -cos(\theta_q) + sen(\theta_q) & a+b \end{bmatrix} \cdot \begin{bmatrix} x_{qk} \\ y_{qk} \\ \theta_{qk} \\ \dot{x}_{qk} \\ \dot{y}_{qk} \\ \dot{\theta}_{qk} \end{bmatrix} \tag{5.2.29}$$

5.2.6. Simulación y resultados:

En una etapa previa a la implementación en el robot se optó por simular en Matlab el algoritmo de fusión sensorial, dado que el mismo debió ser diseñado para los sensores en el sistema y se requería comprobar su funcionamiento para la localización.

Para esto se implementaron las funciones antes descritas, con los cálculos de sus matrices jacobianas y la implementación del EKF. Se puede encontrar el código utilizado en el anexo de la documentación. Se tomaron como entradas las covarianzas de los sensores y las entradas de velocidad de las ruedas en todo momento.

Debajo se detallan los resultados de dicha simulación.

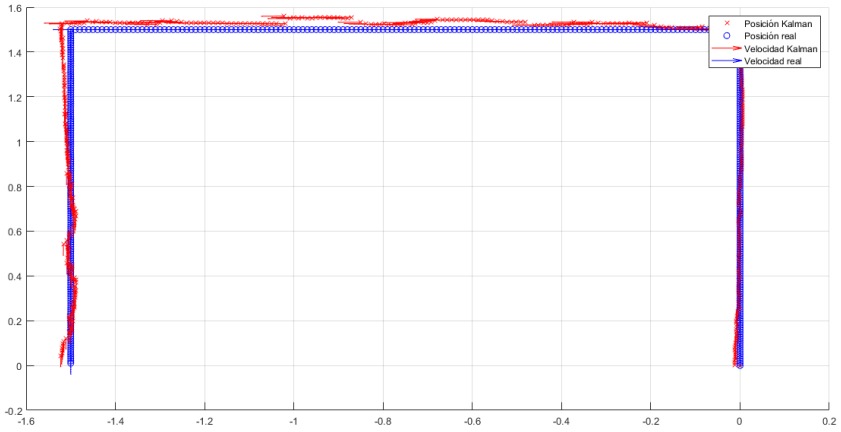


Figura 5.2.3: Posición del modelo vs Posición estimada por el EKF

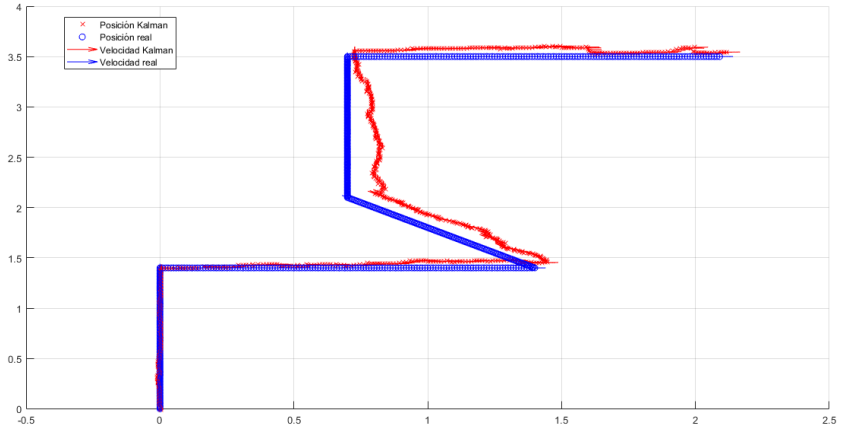


Figura 5.2.4: Posición del modelo vs Posición estimada por el EKF

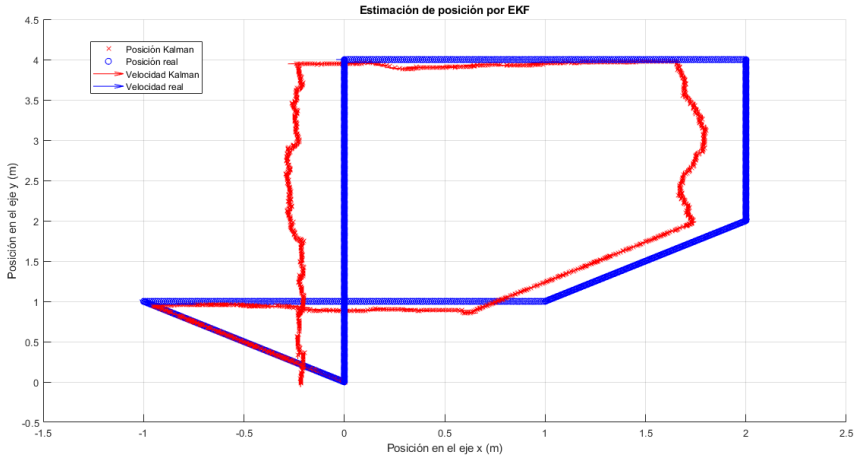


Figura 5.2.5: Posición del modelo vs Posición estimada por el EKF

Como se puede observar en las Figuras 5.2.3, 5.2.4, 5.2.5, se grafican los resultados de la simulación para varias entradas distintas. En estos gráficos figuran la posición 'real', calculada directamente por el modelo de transición de estados y sin considerar ruido en el mismo, y la estimación de la posición dada por el filtro de Kalman extendido.

A pesar que la trayectoria tomada como referencia no cuenta con una covarianza en su modelo, lo que sería equivalente a un robot que se mueve perfectamente y por lo tanto no es óptimo para comparación, los resultados de la estimación del filtro de Kalman diseñado no alcanzan el nivel de precisión necesario para una adecuada localización del robot.

En particular, en la Figura 5.2.5 se puede ver una simulación de una trayectoria que cierra un lazo, es decir que comienza y finaliza en el mismo punto. La estimación del filtro no logra replicar este comportamiento por un margen importante, de $0,2m$ aproximadamente.

Graficando el error de la estimación del filtro de Kalman diseñado, en la Figura 5.2.6, se puede observar que se obtiene un error de hasta $0,4m$ en módulo en algunos casos. Esta precisión no es aceptable para las misiones que debe realizar el robot y los ambientes que debe recorrer.

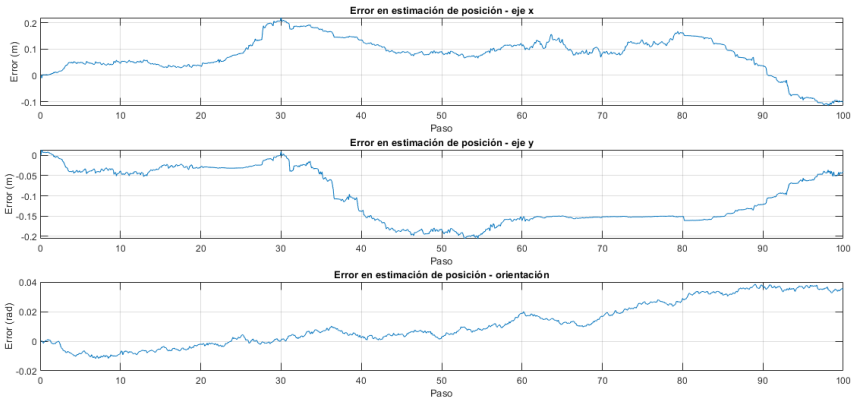


Figura 5.2.6: Error en la estimación del EKF

En conclusión, se decidió descartar esta implementación de un algoritmo de fusión sensorial dado que no cumplió con los objetivos del proyecto, luego de diseñarlo e implementarlo. En lugar de seguir invirtiendo recursos en mejorar el funcionamiento de este algoritmo, se lo descartó en favor de uno ya probado y optimizado para robots móviles terrestres, que permitió la fácil integración del mismo al robot.

5.2.7. Implementación en el robot:

En función de los resultados anteriores se optó por utilizar un algoritmo provisto por el navigation stack de ROS, del que se hablará en el Capítulo 6. Este algoritmo es *robot_pose_ekf*, un filtro de Kalman extendido que necesariamente requiere información de odometría y al menos un sensor más, una IMU, cámara o GPS. Se detalla la implementación de este algoritmo en la Sección 6.2.3.

En lugar de disponer de un modelo de transición de estados del sistema y de los sensores, este algoritmo recibe como entradas de odometría el desplazamiento relativo del robot (Δx_r, Δy_r, y $\Delta \theta_r$) en cada paso y, para la implementación realizada, lo fusiona con datos de la IMU. Esto logra un funcionamiento muy similar al deseado con la simulación.

Como se verá en el Capítulo 7, la implementación de este algoritmo mejora de forma apreciable la localización del robot y permite la navegación y el mapeo cuando imperfecciones en el piso hacen que las ruedas deslicen.

5.3. Mapeo y Localización en Simultáneo

5.3.1. Introducción

5.3.1.1. Motivación

La localización y mapeo simultaneo o SLAM por sus siglas en inglés (Simultaneous Location And Mapping) es clave para lograr un robot autónomo móvil, ya que permite crear un modelo del entorno y relacionarlo con la dinámica del robot, para facilitar el desarrollo de funciones más complejas como la navegación autónoma basada en mapas.

5.3.1.2. Objetivo

Se busca que el robot pueda moverse en un ambiente desconocido y ser capaz de estimar su movimiento relativo en el sistema. Además el robot debe estar equipado con sensores que permitan tener una percepción de su entorno. Tanto la estimación del movimiento como el sensado del entorno se realizan con sensores reales que introducen ruido y por tanto incertidumbre en las variables que se desean estimar.

5.3.1.3. Guía del conocimiento

Como guía principal en el tema SLAM se utilizó el contenido expuesto en el libro 'Springer Hanbook of Robotics' [35] para poder entender el estado del arte. En específico la Sección 46 en la parte E. También fueron de gran utilidad los tutoriales de introducción al tema de Hugh Durrant-Whyte y Tim Bailey [9].

5.3.2. Tipos de Mapas

Para poder modelar el entorno y poder ubicarse en el mismo es importante definir un tipo de dato asociado al modelo del mapa para tener una representación útil.

Debe elegirse un modelado que permita incluir la mayor cantidad de detalles del mapa real pero que no impida su actualización en tiempo real por una sobrecarga de información.

Existen diferentes maneras de modelar un entorno y generar un mapa, pero las más utilizadas en SLAM son mapas basados en características y mapas de grillas de ocupaciones.

Las características son comúnmente denominadas 'landmarks' y son rasgos distintivos en el mapa, reconocibles para el robot y sus sensores. El robot debe ser capaz de reconocer dichos landmarks y poder interpretar su ubicación y orientación relativa a la posición actual en la que se encuentra. Entonces el mapa pasa a ser un vector que en cada entrada se encuentran las coordenadas y la orientación relativa del landmark.

Por otro lado un mapa basado en una grilla de ocupaciones es representado mediante una matriz que cada entrada refiere a una posición en el mapa y el valor

que dicha entrada toma depende de la probabilidad de que esa ubicación en el espacio esté ocupada por un objeto.

La resolución del problema de SLAM utilizada en este proyecto, que es tratada más adelante en esta Sección, modela el mapa a base de grillas.

5.3.3. Nomenclatura a utilizar

Para tratar la resolución del problema de forma concisa, comprensible y ordenada se pasa a detallar la nomenclatura fundamental del problema de SLAM.

A la secuencia de posiciones del robot se la llamará $X_T = \{x_0, x_1, ..., x_T\}$, este es el recorrido del robot entre el tiempo 0 y T, integrado por posiciones discretas x_i. Es importante notar que x_i son variables de dimensión n y que depende del tipo de mapa que se desea construir y el tipo de robot que se manipula. En el robot de este proyecto x_i está compuesto por tres dimensiones (x_i,y_i,θ_i) como se puede observar en la Figura 5.3.1. Siendo x_i e y_i la posición registrada respecto al referencial X,Y, mientras que θ_i es el ángulo que forma el eje X con la dirección en la que apunta el frente del robot en el instante i.

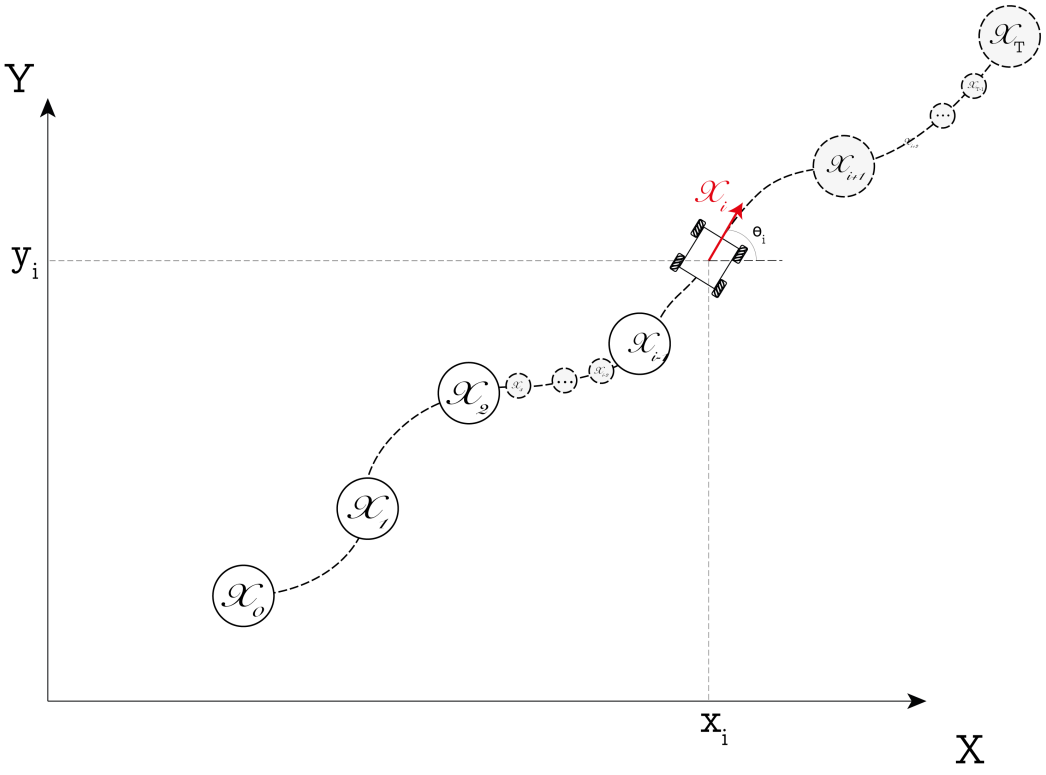


Figura 5.3.1: Trayectoria en SLAM

Se denomina u_t a la odometría que caracteriza el movimiento entre el paso t y $t+1$. Se define el vector de odometrías como $U_T = \{u_0, u_1, ..., u_T\}$. Este contiene todas las transiciones que permiten inferir la trayectoria sabiendo el punto de partida.

Asumiendo que el robot toma una medida, z_i, en cada instante de tiempo, se obtiene $Z_T = \{z_0, z_1, ..., z_T\}$ que es el conjunto de medidas realizadas a lo largo del camino del robot. Las medidas de Z_T relevadas dependen del sistema y los sensores elegidos para relevar las características del mismo. Podría utilizarse solo un sensor como una conjunto para tener mayor información.

El mapa verdadero del ambiente se denota como $\underline{m}$ y es el que se quiere estimar en conjunto con la secuencia de posiciones del robot X_T.

5.3.4. Definición formal del problema

El problema de SLAM consiste en generar un modelo del mundo m y una secuencia de la ubicación del robot$X_T = \{x_0, x_1, ..., x_T\}$ a partir de la odometria U_T a lo largo de las posiciones y los datos obtenidos Z_T por los sensores.

Las variables antes mencionadas son probabilísticas, ya que los sensores son ruidosos e incorporan incertidumbres que deben ser modeladas con distribuciones probabilísticas. Por lo tanto resolver el problema de SLAM es equivalente a hallar $p(X_T, m|Z_T, U_T)$. Esto se interpreta como la probabilidad de la trayectoria X_T y el mapa m dados un set de medidas Z_T y un set de odometrías U_T.

Se distinguen dos formas principales de resolver SLAM:

- **El problema de SLAM completo** ('Full SLAM problem' en inglés):

  En esta modalidad se desea calcular $p(X_T, m|Z_T, U_T)$ para hallar el recorrido $X_T = \{x_0, x_1, ..., x_T\}$ del robot en el mapa con todos los datos necesarios a disposición. Es un proceso que se corre luego de que el robot se movió en el ambiente. No puede ser utilizado para navegar en tiempo real ya que es necesario haber realizado todo el recorrido con el robot.

- **El problema de SLAM en vivo** ('Online SLAM problem'):

  A diferencia del problema completo se desea calcular $p(x_T, m|Z_T, U_T)$, es decir que se quiere calcular la posición para el momento $t = T$ dentro del mapa m y no todas las posiciones del recorrido de principio a fin como en el problema completo. La solución del problema depende solo de la odometría desde el inicio, las medidas del entorno y una posición inicial. Como no es necesario realizar todo el recorrido para procesar los datos, es posible hallar la solución a este problema en tiempo real y de manera iterativa. Por lo tanto este es el tipo de algoritmos que debe ser usados para hacer navegación autónoma.

Para poder resolver SLAM el robot debe estar equipado con dos modelos:

- Modelo matemático que relacione la odometría con el movimiento del robot:

$$p(x_t|x_{t-1}, u_t) \tag{5.3.1}$$

- Modelo que relacione la medida z_t con el ambiente m y la posición del robot: $p(z_t|x_t, m)$. Esta distribución de probabilidad muestra qué tan probable es

la medida z_t según la posición x_t en el mapa m. Generalmente $p(z_t|x_t, m)$ no se conoce directamente ya que modelar la medida realizada por el robot según la posición del robot en el mapa resulta un problema no lineal y de difícil solución. Pero se puede calcular mediante la regla Bayesiana, entonces se tiene :

$$p(z_t|x_t, m) = \frac{p(z_t, x_t, m)}{p(x_t, m)} \tag{5.3.2}$$

5.3.5. Algoritmo utilizado: Gmapping

Para poder resolver el problema de SLAM se decidió utilizar Gmapping: un filtro de partículas de tipo Rao-Blackwell para generar mapas de grillas a partir de un sensor de distancia de tipo láser (LiDAR). La teoría detrás del algoritmo utilizado se puede encontrar en 'Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling'[13] que es una iteración del trabajo 'Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filter' [12].

En este documento se detallarán los conceptos fundamentales de la teoría pero no se profundiza como en los documentos antes mencionados ya que escapan del alcance de este proyecto.

5.3.5.1. Filtro de Partículas

Los filtros de partículas, también conocidos como métodos secuenciales de Montecarlo, son un conjunto de algoritmos usados generalmente para estimar el estado interno de un sistema dinámico a partir de observaciones parciales con perturbaciones provenientes de los sensores o bien del sistema.

El objetivo de los algoritmos de esta rama es computar la distribución a posteriori del estado en un proceso de Markov, dada una observación parcial y ruidosa. De esta manera se puede tomar una muestra de esta distribución para obtener un buen estimador del próximo estado o del estado actual.

El filtro de partículas usa un conjunto de partículas para representar la distribución a posteriori del estado. Cada una de las partículas representa un posible estado en el que se puede encontrar el sistema.

El proceso de filtrado de las partículas consta de dos etapas predicción y actualización. Por un lado, en la predicción se utiliza información del sistema para generar el set de partículas, cada partícula tiene asociado un peso de verosimilitud w. Este peso determina la probabilidad de que esa partícula sobreviva al proceso de actualización, ya que se retiran las partículas menos probables para mantener el conjunto más cercano al estado real del problema. Además, el peso de verosimilitud es el que determina que tan probable es que se muestree la partícula del conjunto para representar la magnitud que se quiere calcular.

El peso de verosimilitud tiene como propósito lograr que la distribución calculada se asemeje a la distribución real del estado del sistema mediante iteraciones.

Entonces si a cada partícula se indexa según k se tiene que para cada instante el peso de verosimilitud representa:

$$w^k{}_t = \frac{\text{distribución objetivo}}{\text{distribución propuesta}} \tag{5.3.3}$$

Luego, si llevando este concepto abstracto a una formula real y definiendo la distribución objetivo como $p(X_T^{(k)}|Z_T, U_T)$ y la distribución propuesta a $\pi(X_T^{(k)}|Z_T, U_T)$ obtenemos:

$$w^k{}_t = \frac{p(X_T^{(k)}|Z_T, U_T)}{\pi(X_T^{(k)}|Z_T, U_T)} \tag{5.3.4}$$

Por último según el desarrollo de la Sección III A de 'Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters'[12] se obtiene:

$$w_t^{[k]} = \mathcal{N}(z_T|X_T^{[k]}, \mu_{T,n}^{[k]}, \Sigma_{T,n}^{[k]}) \tag{5.3.5}$$

Donde $\mu_{T,n}^{[k]}$ es la media y $\Sigma_{T,n}^{[k]}$ la varianza asociada a la distribución Gaussiana.

De esta manera se obtiene una relación entre la medida realizada por el robot z_T y la importancia que se le debe dar a la partícula según si el modelo que contiene se asemeja a la realidad o no.

5.3.5.2. Filtro de Partículas aplicado a SLAM

En el caso de SLAM se utiliza el filtro de partículas para representar la distribución de probabilidad asociada a $p(x_T, m|Z_T, U_T)$ mediante un set de partículas.

Un problema con este método es que el espacio de mapas y caminos posibles es demasiado grande y por lo tanto el número de partículas necesarias para lograr resolver el problema puede elevarse aumentando el costo computacional. Por cada partícula se debe realizar el cálculo de odometría para estimar la nueva posición y a su vez interpretar los datos obtenidos para esa posición. Esto lleva a querer minimizar el número de partículas necesarias y por tanto tener un método preciso para sacar con reposición del conjunto de partículas de una manera eficiente.

Las primeras implementaciones de un filtro de partículas para resolver SLAM de una manera eficiente, en términos de procesamiento en tiempo real y bajo uso de recursos computacionales, fueron: 'Rao–Blackwellized particle filtering for dynamic Bayesian network'[26] y 'FastSLAM: a factored solution to the simultaneous localization and mapping problem with unknown data association' [22].

Tanto esas primeras implementaciones como Gmapping utilizan un proceso que se conoce como la Rao-Blackwellización del filtro de partícula y se remonta a [33] y [4].

La Rao-Blackwellización del filtro de partículas tiene como objetivo principal simplificar el cálculo de la probabilidad conjunta a posteriori de $p(x_T, m|Z_T, U_T)$. Rao-Blackwell propone usar la siguiente igualdad para hallar el valor antes mencionado.

$$p(x_T, m|Z_T, U_T) = p(x_T|Z_T, U_T)p(m|x_T, Z_T) \tag{5.3.6}$$

En la Figura 5.3.2 se puede apreciar que si se retiran las variables que llevan la información del camino, aquellas con fondo gris, el resto queda desconectada y por lo tanto independientes entre si. Este es el fundamento que permite hacer la factorización mencionada 5.3.6.

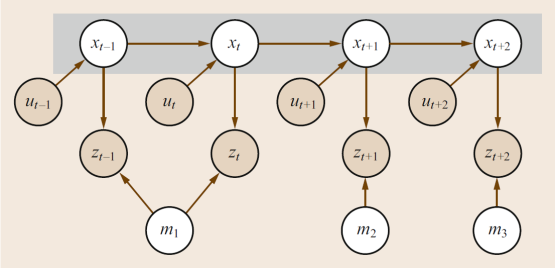


Figura 5.3.2: Representación de SLAM y la independencia entre landmarks sin la trayectoria

Utilizando la factorización mostrada en 5.3.6 se puede calcular por un lado la trayectoria del robot y luego a partir de ella el mapa en el que se mueve. Esto resulta conveniente ya que el mapa se construye a partir de la superposición de medidas del robot a lo largo de la trayectoria recorrida.

La probabilidad a posteriori del mapa $p(m|x_T, Z_T)$ se puede computar analíticamente utilizando 'mapeo con poses conocidas'[23] ya que se conoce la trayectoria del robot X_T y las medidas realizadas en cada punto Z_T.

Entonces solo se aplica el filtro de partículas a la probabilidad a posteriori de la trayectoria del robot $p(X_T|Z_T, U_T)$. Así cada partícula representa una trayectoria potencial del robot y tiene asociado un mapa , ya que a partir de cada trayectoria se puede calcular un nuevo mapa.

El proceso de actualización de las partículas se realiza de la siguiente manera:

- Cuando se calcula la odometría a partir de los encoders se generan estocásticamente nuevas variables de localización, una por cada partícula. La distribución para generar las ubicaciones está basada en el modelo de movimiento:

  $$x_t^{[k]} \approx p(x_t|x_{t-1}^{[k]}, u_t) \tag{5.3.7}$$

  Este paso de muestreo probabilístico puede realizarse fácilmente para cualquier robot que se pueda computar su cinemática.

- Cuando se recibe una medida z_t ocurren dos cosas:

  1. Se calcula, para cada partícula, la probabilidad de la nueva medida z_t, es decir qué tan probable es la medida obtenida según la posición en la que se encuentra el robot. Siendo n el índice del landmark sensado, entonces la probabilidad que se desea obtener es la siguiente:

$$w_t^{[k]} = \mathcal{N}(z_t|x_t^{[k]}, \mu_{t,n}^{[k]}, \Sigma_{t,n}^{[k]}) \tag{5.3.8}$$

Como se explicitó anteriormente $w_t^{[k]}$ es conocido como el peso de verosimilitud o importancia, ya que mide qué tan importante es la partícula según los nuevos datos obtenidos. Luego se normalizan los pesos de todas partícula para que la suma de los pesos de importancia dé uno.

2. Una vez calculado los pesos el algoritmo se muestrea del set de partículas actual, y se genera uno nuevo. Las partículas que no son seleccionadas se descartan para dar lugar a nuevas, generadas a partir del promedio de las partículas sobrevivientes. La probabilidad de muestrear una partícula es el peso de importancia normalizado. Este paso es denominado remuestreo.

- Por último, se estima el mapa para cada partícula a partir de la trayectoria almacenada y los datos de los sensores calculando así $p(m|x_T, Z_T)$ en cada partícula.

Se puede muestrear del conjunto de partículas para obtener una representación del mapa y la trayectoria del robot. Para tomar una muestra se utiliza el peso de importancia, como fue mencionado anteriormente, para determinar con que probabilidad se puede muestrear cada una de las partículas.

En este proyecto, el vector de medidas del entorno Z_T se genera a partir del LiDAR que permite tomar un corte bidimensional paralelo al piso de la habitación en la que se encuentra. Por otro lado, para calcular la odometría del dispositivo se utilizaron los encoders de los motores según las condiciones mencionadas en el Capítulo 4.

5.3.5.3. Implementación en el robot

Además, Gmapping utiliza la información de la odometría para ajustar el peso verosimilitud ponderando la probabilidad de las partículas según qué tanto se asemeje la medida realizada con el LiDAR a la esperada según la posición.

Entonces mediante Gmapping se resuelve el mapeo del entorno mientras que se calcula la trayectoria que se realiza al desplazarse por el mismo.

5.4. Planeación y Seguimiento de Trayectorias

5.4.1. Marco teórico

La planeación de trayectorias es uno de los pilares de la autonomía del robot, le permite al mismo navegar por el ambiente sin necesidad de asistencia y evitando la colisión con los obstáculos presentes. En este capítulo se utilizó como base [5] y [19].

Su importancia fue expresada por uno de los referentes de la planeación de movimientos en robótica Jean Claude Latombe como:

'... es eminentemente necesario ya que, por definición, un robot cumple tareas moviéndose en el mundo real' [18]

El problema de planeación de trayectorias consiste en encontrar un camino (si existe) que mueva el robot desde una posición inicial a una objetivo sin sufrir colisiones con ningún obstáculo, dados:

- Posición inicial del robot
- Posición objetivo del robot
- Descripción geométrica del robot
- Descripción geométrica del ambiente

A pesar de que el problema formulado anteriormente está definido en el mundo real, para solucionarlo se trabajará en el espacio de configuraciones que se definirá a continuación.

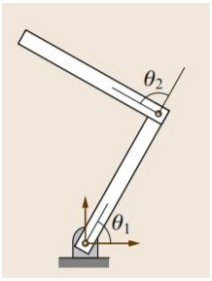
5.4.1.1. Espacio de configuración

Se define una configuración q como la especificación de todos los puntos del robot relativo a un referencial de coordenadas fijo. Usualmente se expresa como un vector que incluye información de su posición y de su orientación.

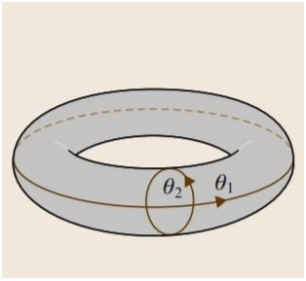
El espacio de configuración de un sistema es el espacio de todas las configuraciones posibles.

En la Figura 5.4.1a, se puede ver un ejemplo del concepto de configuración. Se tiene un brazo con dos articulaciones libres en el plano de la Figura. Por lo cual alcanza con conocer θ_1 y θ_2, para saber la posición de todos los puntos del brazo, esta es la idea detrás de las configuraciones.

En la Figura 5.4.1b se puede observar el mapeo del espacio de configuración en un toroide. Cualquier punto del toroide se puede describir también utilizando solamente θ_1 y θ_2, y se corresponde de forma biunívoca con una disposición del brazo.



(a) Brazo con dos articulaciones



(b) Espacio de configuración

Figura 5.4.1: Primer ejemplo de espacio de configuración

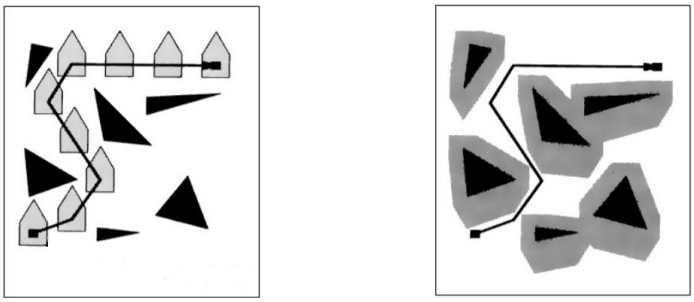
En la ecuación 5.4.1 se puede observar una configuración genérica q_i, que pertenece al espacio de configuración C_{brazo} para todo $\theta_1 \in [0, 2\pi]$, $\theta_2 \in [0, 2\pi]$.

$$q_i = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \tag{5.4.1}$$

En la Figura 5.4.2a se puede observar un ejemplo más parecido al de este proyecto. Se puede ver un robot poligonal, que tiene su movimiento restringido a la traslación únicamente. Entonces al moverse por todo el ambiente alrededor de los obstáculos, se pueden obtener los límites del espacio de configuración, es decir los puntos en los cuales se puede encontrar el robot.

En la Figura 5.4.2b se puede observar el espacio de configuración, y es importante notar que en el nuevo problema ya no se trabaja con un robot en un ambiente, sino con un punto o configuración en el espacio de configuraciones. Cabe aclarar que como sólo puede trasladarse, la información sobre su orientación no forma parte de la configuración. Por ende si se quiere usar un robot que rota, el espacio de configuración ya no podrá ser mapeado en $\mathbb{R}^2$ sino en $\mathbb{R}^3$ y por supuesto incrementando la dificultad de trabajar con él.

Sin embargo, si se tiene en cuenta que el robot utilizado en el proyecto es holonómico y que tiene ancho y largo similares (lo que nos dice que se podría pensarlo como un robot circular), un espacio de configuración bidimensional independiente de la orientación es una muy buena aproximación.



(a) Espacio de trabajo

(b) Espacio de configuración

Figura 5.4.2: Segundo ejemplo de espacio de configuración

5.4.1.2. Formulación del problema

A partir de la introducción teórica sobre espacios de configuración explicada anteriormente se puede definir el problema de la planeación de trayectoria de manera más precisa.

Sea $W = \mathbb{R}^m$ el ambiente, $O \in W$ el conjunto de obstáculos, $A(q)$ donde q es la configuración en que se encuentra el robot A.

Se definen los siguientes dos espacios, C_{libre} aquellas configuraciones donde el robot puede estar dentro del espacio C y $C_{ocupado}$ su complemento :

- $C_{libre} = \{q \in C / A(q) \cap O = \emptyset\}$
- $C_{ocupado} = C - C_{libre}$

Se definen además dos configuraciones:

- q_i: configuración inicial del robot.
- q_f: configuración objetivo del robot.

El propósito de la planeación de trayectorias es encontrar un camino τ, tal que $\tau : [0, 1] \rightarrow C_{libre}$ con $\tau[0] = q_i$ y $\tau[1] = q_f$.

5.4.1.3. Discretización del espacio

El espacio de configuración C_{libre} necesita ser discretizado para poder aplicar algoritmos de planeación de trayectorias. Existen dos grandes maneras de discretizar el espacio: planeación combinatorial o planeación basado en muestreo.

planeación basado en muestreo: El planeación basado en muestreo[2] tiene como objetivo discretizar el espacio C_{libre} utilizando un algoritmo de evasión de obstáculos para moverse en el ambiente.

[2]Se puede encontrar en la bibliografía como *Sampling-based planning*.

Toma muestras aleatorias de configuraciones de todo el espacio C y si están en C_{libre}, son guardadas como vértices en un grafo. A su vez estos vértices se tratan de conectar mediante un módulo de planeación local, que genera las aristas entre los vértices.

Por lo cual, lo que se obtendrá como resultado de aplicar este tipo de algoritmos es una cantidad de configuraciones libres mapeadas como vértices de un grafo, y entre ellos caminos libres de colisión mapeados como aristas que conectan los vértices.

Un punto central que diferencia a los algoritmos de este tipo entre ellos, es la manera en que toman las muestras. Los algoritmos basados en muestreo más conocidos son:

- $A*$
- $D*$
- RRT[3]
- PRM[4]

planeación combinatorial: Los algoritmos de planeación combinatorial describen explícitamente C_{libre} a través de un grafo. Se caracterizan por ser completos, es decir para cualquier problema el algoritmo encontrará una solución o reportará correctamente que no existe.

Estos métodos generan caminos[5], que son grafos en C_{libre}, en el cual cada nodo es una configuración de C_{libre} y cada arista una camino libre de colisión.

Existen varios algoritmos de esta clase como pueden ser los *Grafos de visibilidad* o los *Diagramas de Voronoi*. Sin embargo en el caso del proyecto será utilizado una descomposición en celdas como forma de discretizar el espacio. Esta descomposición en celdas utilizada, será referida más adelante como mapa de costos.

5.4.1.4. Niveles de planeación

Dada la complejidad del problema de planeación de trayectorias, es común utilizar una arquitectura en dos capas para plantear la solución.

La capa superior es llamada Global Planner tiene como entrada una discretización global del espacio de configuraciones, y con él se construirá un grafo. Una particularidad del Global Planner es que realiza el camino solución sin preocuparse de la cinemática ni de la dinámica del robot, transformando el problema inicial en la búsqueda de caminos óptimos a través del grafo mencionado anteriormente.

El algoritmo utilizado en este proyecto para realizar esta búsqueda es *Dijkstra* que será explicado en la Sección 5.4.2.

El Local Planner está encargado de calcular las velocidades relativas del robot para moverse hacia cada uno de los objetivos parciales, dados por el Global Planner.

[3] *Rapidly-exploring random tree*, por sus siglas en inglés

[4] *Probabilistic Road Map*

[5] En la bibliografía se encuentra como *roadmap*

Para ello en vez de trabajar con el total del espacio de configuración, trabaja con una porción más pequeña (una discretización local) pero más precisa del mismo. El método utilizado en este proyecto para generar las trayectorias locales es el *Dynamic Window Approach*, que se describe en la Sección 5.4.3.

A modo de resumen, en la Figura 5.4.3 se puede observar un diagrama de bloques que muestra el funcionamiento en dos niveles. Se puede ver que el bloque Global Planner define las grandes directrices de movimiento del robot, proveyendo al nivel inferior de metas intermedias, que son en realidad una cadena de vértices (o configuraciones) del grafo. Pero el Local Planner indica las velocidades que debe tomar el robot. Las entradas de cada bloque de planeación son las respectivas discretizaciones del espacio, estos bloques de discretización se implementan a partir de la utilización de mapas de costo, que serán explicados con detenimiento en la Sección 5.4.4.

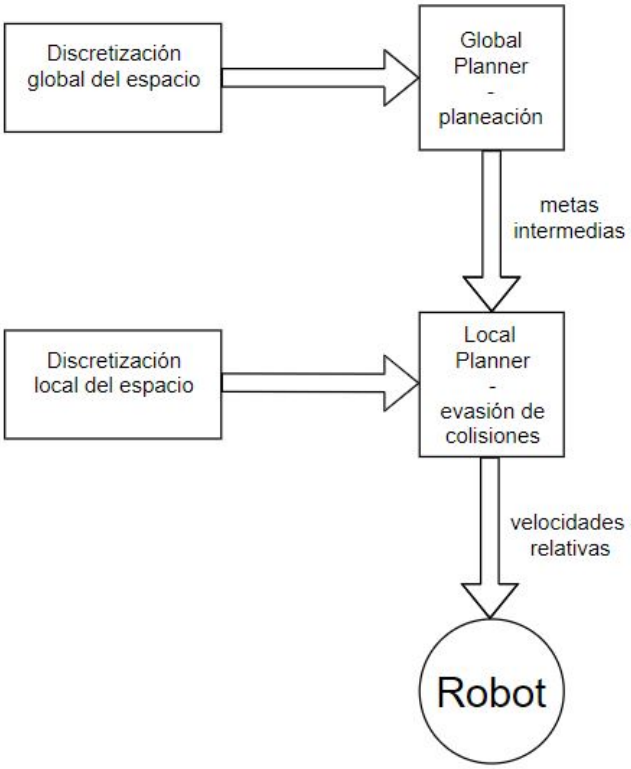


Figura 5.4.3: Diagrama de arquitectura en dos niveles

5.4.2. Global Planner : *Dijkstra*

Dijkstra es uno de los algoritmos más populares en la búsqueda de caminos mínimos[6], y se describe de forma accesible en [16] . Para describir este algoritmo partimos de la base de un grafo ponderado. En un grafo ponderado como el de la Figura 5.4.4, las aristas (i, j) que conectan los nodos tienen un peso $w(i, j)$, y la longitud de una trayectoria entre dos nodos es la suma de los pesos de las aristas recorridas.

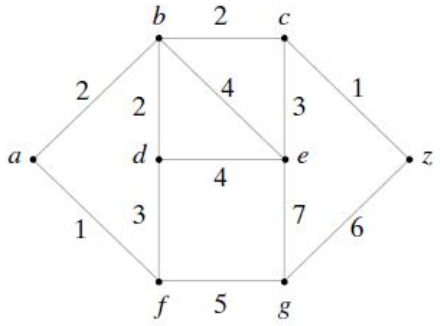


Figura 5.4.4: Ejemplo de grafo ponderado

El objetivo de este algoritmo es: dado dos vértices del grafo, a(inicio) y z(final), encontrar la ruta más corta entre ellos. Para lograr dicho propósito se deben asignar etiquetas a los vértices, que como se verá a continuación pueden ser temporales o permanentes.

Se llama $L(v)$ la etiqueta del vértice v. Al principio las etiquetas de todos los nodos serán temporales, y en cada iteración del algoritmo una pasará a ser permanente. Se puede demostrar que si la etiqueta de v es permanente, entonces $L(v)$ es la longitud mínima entre a y v.

A continuación se describe el pseudocódigo del algoritmo.

```
1  L(a)=0;
2  for x from b to z do
3  |  L(x) = ∞;
4  end
5  while z ∈ T do
6  |  seleccionar v ϵ T con L(v) mínimo;
7  |  T=T-{v};
8  |  for x ∈ T /∃(x, v) do
9  |  |  L(x) = min{L(x), L(v) + w(v, x)}
10 |  end
11 end
```

En la Figura 5.4.5 se puede ver la inicialización del algoritmo, donde todas las etiquetas valen ∞, menos la del vértice origen a que vale 0.

[6]Tiene particular relevancia en el área de las telecomunicaciones, sobretodo en cuanto a protocolos de ruteo refiere

En la primera iteración se elige el vértice con la menor etiqueta, que como se dijo anteriormente es a. Se puede ver en la Figura 5.4.6 que los vértices adyacentes son b y f, se etiquetan como el peso de sus aristas hacia a, que son 2 y 1 respectivamente.

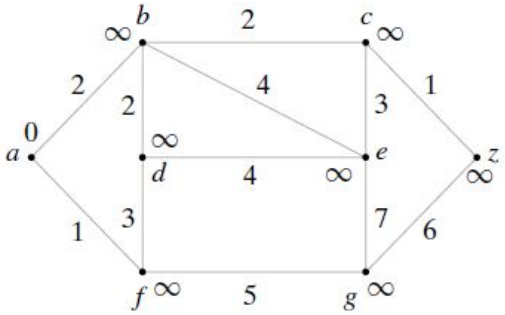


Figura 5.4.5: Inicialización del algoritmo

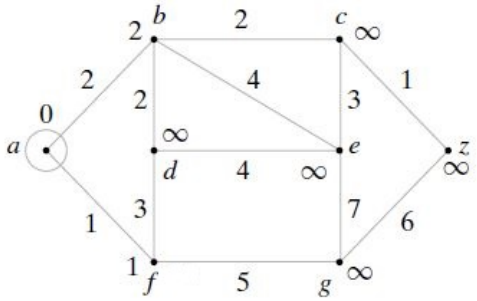


Figura 5.4.6: Primera iteración

Para la segunda iteración (Figura 5.4.7, el vértice de la menor etiqueta es f, y sus adyacentes a etiquetar son d y g.

En la Figura 5.4.8, se puede ver la situación después de la tercera iteración, usando el vértice b como referencia. Un comentario es que antes de la tercera iteración el valor de la etiqueta de d, era 4. Si el peso de la arista (f, d) hubiese sido 1, por ejemplo, en vez de 3, el valor de la etiqueta de d después de la tercera iteración se hubiese actualizado a 2.

El algoritmo seguirá avanzando hasta que todas la etiqueta de z sea permanente. Es fácil observar que el valor de la etiqueta permanente de z será 5, y el camino óptimo será $a - b - c - z$

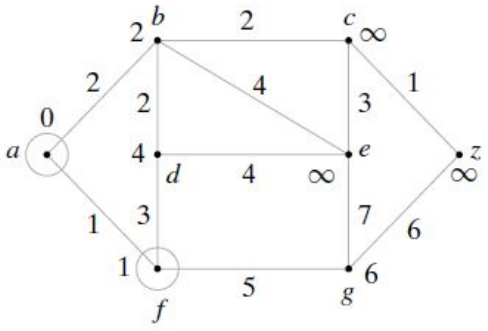


Figura 5.4.7: Segunda iteración

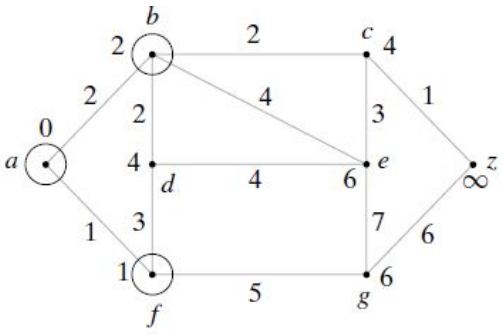


Figura 5.4.8: Tercera iteración

5.4.3. Local Planner: Dynamic Window Approach

Dynamic Window Approach es un método de evasión de obstáculos utilizado como Local Planner, descrito en la publicación 'The dynamic window approach to collision avoidance' [11].

Este algoritmo considera solamente un intervalo de tiempo pequeño para computar cada orden. En este tiempo simula trayectorias de arcos de circunferencia tratando de aproximar la trayectoria global y evitando obstáculos.

Para generar dichos arcos se tendrá un espacio de búsqueda de dos dimensiones, que incluye las velocidades de traslación y rotación. Un punto importante es que sólo se incluyen en el espacio de búsqueda aquellas velocidades que se consideran admisibles. Lo que significa que son velocidades que se pueden alcanzar en el intervalo de tiempo, y permiten frenar al robot de forma segura ante los obstáculos.

Se define el conjunto de velocidades posibles $V_s = v, w$, siendo v y w las velocidades lineal y angular que puede alcanzar el robot. Sea t el tiempo durante el cual las aceleraciones , $\dot{v}$ y $\dot{w}$ serán aplicadas. Sea (v_a, w_a) la velocidad actual. Se define la ventana dinámica V_d como:

$$V_d = \{(v, w)/v \in [v_a - \dot{v}.t, v_a + \dot{v}.t], w \in [w_a - \dot{w}.t, w_a + \dot{w}.t]\} \tag{5.4.2}$$

Para funcionar como evasor de colisiones es necesario tener en cuenta la distancia a los objetos para delimitar el espacio de búsqueda. Por lo cual se define el conjunto de velocidades admisibles V_a como:

$$V_a = \{(v, w)/v \leq \sqrt{2.dist(v, w).\dot{v}_b}, w \leq \sqrt{2.dist(v, w).\dot{w}_b}\} \tag{5.4.3}$$

Siendo $dist(v, w)$ una función que devuelve la distancia al objeto más cercano y será explicada en el párrafo 5.4.3. Además se definen $\dot{v}_b$ y $\dot{w}_b$ como las aceleraciones de colisión.

A partir de los dos conjuntos de velocidades anteriores V_a y V_d, y del conjunto de velocidades posibles V_s, se conforma el espacio de búsqueda de velocidades V_r como:

$$V_r = V_s \cap V_d \cap V_a \tag{5.4.4}$$

En la Figura 5.4.9 , se puede ver un ejemplo de la ventana dinámica. Se puede observar V_s y V_d señaladas en la Figura. En cuanto a V_a, las velocidades admisibles se encuentran en gris claro y las no admisibles en gris oscuro. La intersección de las 3 ventanas, por ende el espacio de búsqueda V_r, se encuentra en blanco.

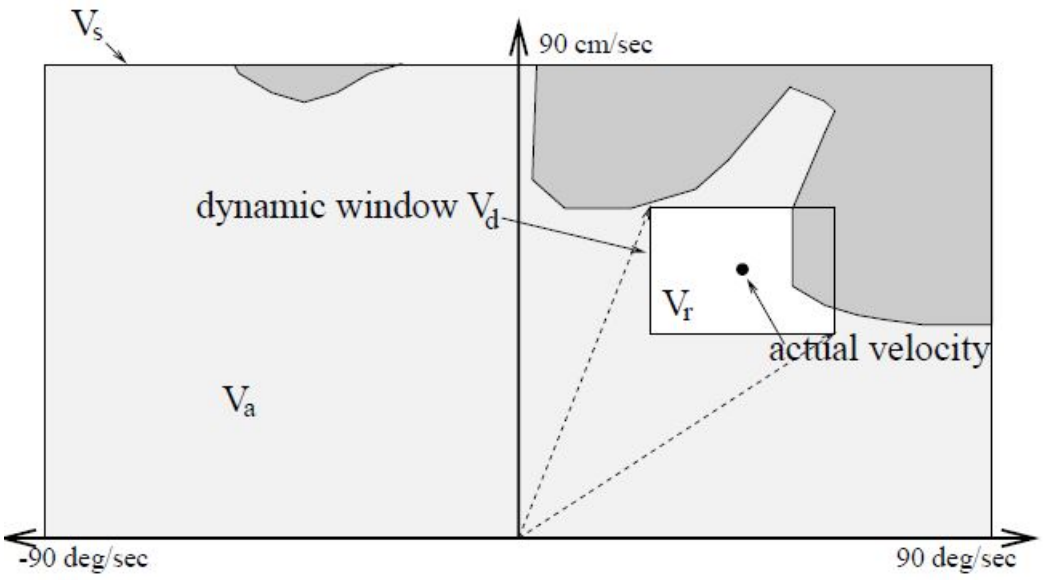


Figura 5.4.9: Venta dinámica

Dentro de las velocidades que conforman el espacio de búsqueda, se elegirá aquella que maximice la siguiente función de costo.

$$G(v, w) = \sigma(\alpha.heading(v, w) + \beta.dist(v, w) + \gamma.vel(v, w)) \tag{5.4.5}$$

Teniendo en cuenta las siguientes entradas a la función: *heading*, *dist* y *vel*. Las tres entradas son ponderadas por el usuario con sus respectivos parámetros α, β, y γ. Esto permite modificar el comportamiento del módulo de planeación.

Heading: Dado (v, w), $heading(v, w)$ devuelve el ángulo entre la dirección principal de robot y la predicción objetivo. Para predecir la posición al final del intervalo de tiempo se asume que el robot se mueve con velocidades (v, w) durante ese lapso.

Dist: La función $dist(v, w)$ representa la distancia al objeto más cercano que intersecta con la curvatura. En el caso de que no existan objetos detectados, la función devuelve una constante del valor máximo definido.

Vel: La función $vel(v, w)$, devuelve la proyección de la velocidad de traslación v.

Función Objetivo La función $G(v, w)$ como vimos en la ecuación 5.4.5 es la suma ponderada de los tres términos que se explicaron anteriormente. Estos términos están normalizados entre $[0, 1]$, y tendrán valor 0 para velocidades que no se encuentren el espacio de búsqueda.

El par de velocidades (v, w) elegidos para mover el robot, como se dijo anteriormente, será aquel que maximice esta función.

5.4.4. Mapas de Costo

Los mapas de costo son grillas de ocupación que se utilizan para crear las discretizaciones del espacio, que utilizan los algoritmos de planeación. Se generan a partir de los mapas provenientes de $SLAM$.

Se utilizaron dos mapas de costo, uno para cada nivel de planeación. Para el nivel global el mapa de costo contiene información sobre el ambiente, mientras que para el local solo sobre una reducida porción alrededor del robot. El mapa local incluye además información del sensor de ultrasonidos para la detección de obstáculos de baja altura.

En una grilla de ocupación el mapa está divido en celdas, las cuales pueden tener un valor entre 0 y 255. Aquellas que tengan costo 0 serán consideradas libres, mientras que las que tengan un valor distinto de 0 tendrán un grado de ocupación. Las celdas que no tengan un valor de costo asociado serán consideradas desconocidas.

En la Figura 5.4.10 se puede observar la representación del robot. Desde adentro hacia afuera se puede observar en negro la celda central del robot, es decir la que contiene al centro del mismo. En rojo se puede ver la silueta del robot, mientras que en azul se ven dos circunferencias: la circunferencia inscrita y la circunscrita. Se le llamará al radio de la circunferencia inscrita radio menor, y al de la circunscrita radio mayor.

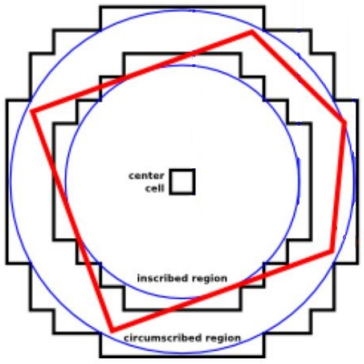


Figura 5.4.10: Silueta del robot

En la Figura 5.4.11 es donde se introduce el concepto de inflación, es decir darle costo a celdas que no están ocupadas, pero que transitar por ahí implica una posible colisión para el robot.

Una celda donde hay un obstáculo es considerada letal, teniendo colisión asegurada, para todos las celdas dentro del radio menor también habrá colisión asegurada. Por otro lado existirá una colisión posible, dependiendo de la orientación del robot para todas las celdas que no estén entre el radio mayor y menor.

También es posible asignarle costo a las celdas que no están en zona de colisión. La idea detrás de esto es desalentar a que el robot circule por las áreas cercanas a los obstáculos. En la Figura 5.4.11 se puede ver la curva de decaimiento del costo

en función de la distancia. Tanto el radio de inflación como el factor de decaimiento son parámetros configurables.

Otra de las razones por la cual se genera esta zona de inflación alrededor de la celda ocupada es para ganar robustez frente a la forma y la orientación del robot.

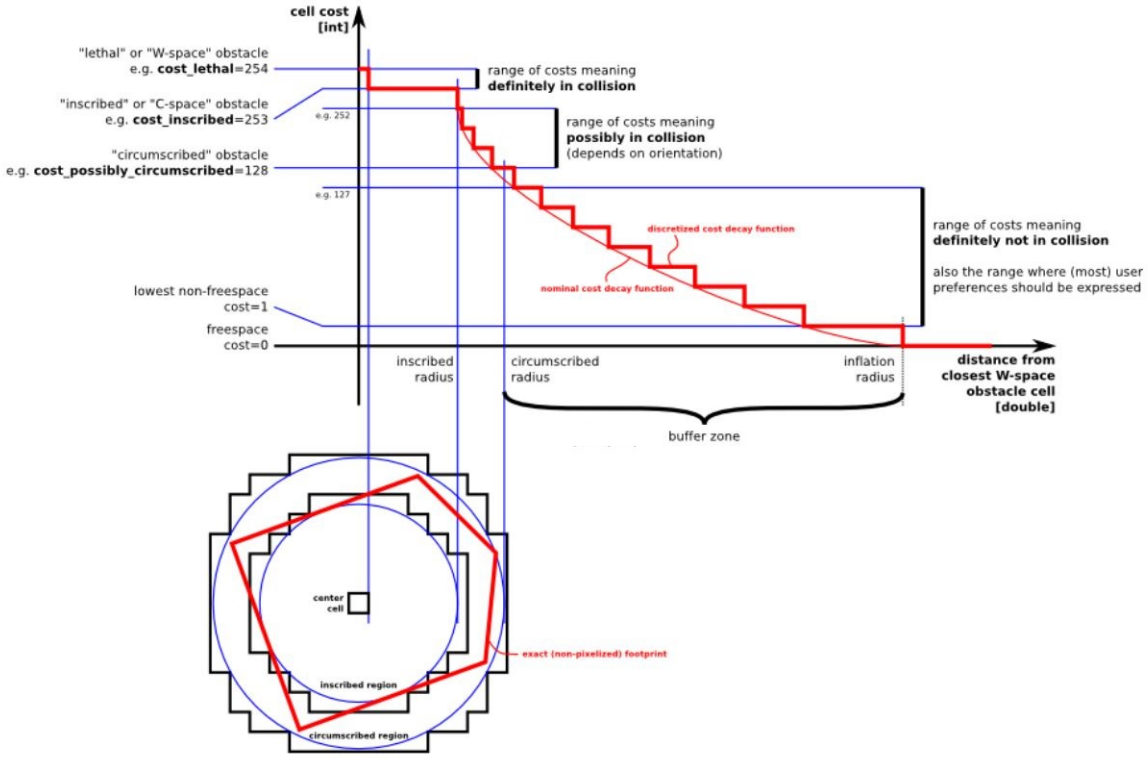


Figura 5.4.11: Inflación

El mapa de costo está organizado en capas, en este caso se utilizaron las siguientes tres, pero es posible añadir más:

- **Capa estática:** Incorpora información proveniente del algoritmo de mapeo.
- **Capa de obstáculos:** Se alimenta de la información de sensores para incluir los obstáculos detectados en el mapa.
- **Capa de inflación:** Se agregan nuevos valores alrededor de los obstáculos con el propósito de acercar lo más posible el mapa de costos al espacio de configuración del robot.

5.4.5. Implementación: move_base

La implementación de todos los algoritmos antes descritos para la planificación y seguimiento de trayectorias se realiza a través de módulos de ROS de los que se hablará en más detalle en la Sección 6. Estos se centralizan en un nodo principal

llamado 'move_base' que se encarga de combinar toda la información de los demás y controlar el robot.

5.4.6. Exploración del espacio

Para lograr la autonomía total se debe agregar la función de exploración que permite al robot navegar por ambientes desconocidos sin ayuda mientras explora y genera un mapa del ambiente. Este mapa se podrá utilizar posteriormente en la navegación. El algortimo utilizado en este proyecto implementa una técnica llamada Exploración basada en fronteras, que será explicada a continuación.

5.4.6.1. Exploración basada en fronteras

Esta Sección está basada el artículo 'A frontier-based approach for autonomous exploration' [42], que marca la siguiente premisa detrás de la exploración basada en fronteras:

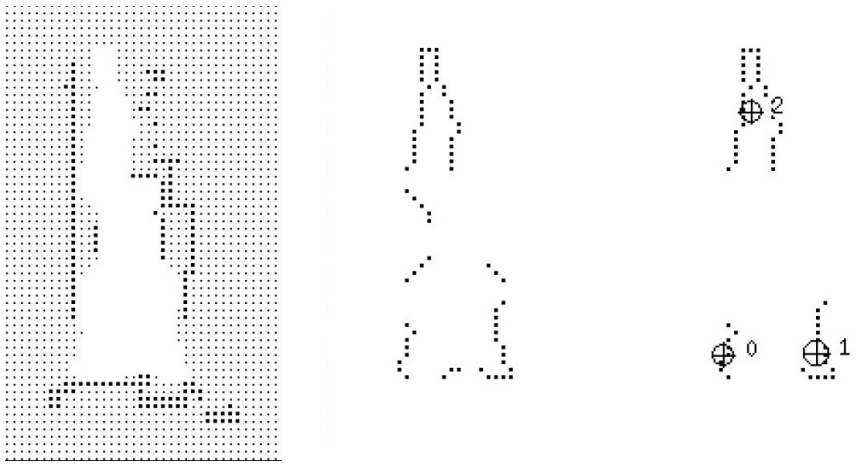
'Para ganar la mayor información sobre el ambiente, es necesario moverse a la frontera entre el espacio abierto y el desconocido.'

Se define la frontera como la región entre el espacio desconocido y el espacio libre y conocido. Moviéndose hacia una frontera se descubre el ambiente y nuevas fronteras, que serán próximos objetivos.

Detección de fronteras: Dada una probabilidad de ocupación fijada de antemano (Por ejemplo P_{priori} =0.5). Se clasifican en función de ella, las celdas del mapa de costo, en tres tipos:

- ocupadas: probabilidad de ocupación $>P_{\text{priori}}$
- libres: probabilidad de ocupación $<P_{\text{priori}}$
- desconocidas: probabilidad de ocupación $= P_{\text{priori}}$

Cuando se tiene una celda libre adyacente a una celda desconocida, se etiqueta como celda borde. A su vez si se tienen varias celda borde adyacentes se tiene un región de frontera. Por último si esta región supera un tamaño mínimo configurado por el usuario, se le considera una frontera.



(a) Etapa inicial (b) Bordes detectados (c) Centros de fronteras

Figura 5.4.12: Proceso de detección de fronteras

El proceso descrito anteriormente se ejemplificará a continuación. En la Figura 5.4.12a, podemos ver las celdas libres, desconocidas y ocupadas, en blanco, gris y negro respectivamente.

En la Figura 5.4.12b se pueden observar los bordes detectados, mientras que por último en la Figura 5.4.12c están marcados con círculos los centros de las regiones que se calificaron como fronteras.

Navegación entre fronteras: Una vez que las fronteras fueron detectadas, el robot navega hasta la más cercana. Cuando cumple con este objetivo agrega la frontera a la lista de fronteras visitas y vuelva a realizar la detección de fronteras, comenzando otra vez la iteración.

En el caso de que después de un cierto tiempo el robot no pueda lograr llegar a la frontera objetivo, determinará que la frontera es inaccesible y se dirigirá a la frontera accesible más cercana.

5.5. Implementación

Se utilizó el módulo *Explorer Lite*, basado en exploración de fronteras. Dicho módulo genera posiciones objetivo que le encarga a *move base* a partir de los mapas disponibles.

Una de las ventajas del *Explorer Lite* frente a otros opciones de exploración como *frontier_ exploration*, es que no genera sus propios mapas de costo sino que utiliza los ya disponibles, reduciendo la carga computacional del sistema.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 6

Implementación en Software

Este Capítulo engloba todo el desarrollo de software que se realizó en este proyecto y como se usaron las librerías preexistentes. También se describe el middelware usado como cimiento del proyecto: ROS [1].

6.1. ROS

6.1.1. Introducción

ROS (Robot Operating System) es un 'middleware', es decir una capa de software que funciona por encima del sistema operativo principal, generalmente una distribución de Linux, y por debajo de las aplicaciones de alto nivel. Provee múltiples funcionalidades, herramientas y buenas prácticas orientadas al desarrollo de la robótica. Un pilar de la filosofía detrás de ROS es no 'reinventar la rueda', proveyendo librerías bien documentadas y fáciles de usar para todo tipo de funciones como: navegación, manipulación, control e intercambio de mensajes, entre otras [17].

ROS fue desarrollado por Willow Garage y la Universidad de Stanford como parte del programa STAIR [2] que comenzó en el año 2007. El propósito de este era crear un 'middleware' gratuito y de código abierto para el desarrollo a gran escala de sistemas robóticos complejos. Gracias a que el software es de código abierto, el desarrollo no está restringido solamente a estas dos entidades, sino que hay una gran comunidad de desarrolladores que mantienen, y continuamente crean y mejoran las distintas partes que componen este software.

Para utilizar ROS es necesario superar un curva de aprendizaje. Durante el tiempo de adaptación a ROS el usuario comprende la importancia de generar código modular, reutilizable y de fácil comprensión para un lector ajeno al proyecto.

[1] Robot Operating System
[2] Standford Artificial Intelligence Robot

6.1.2. Estructura

ROS se basa en una estructura de grafos interconectados para formar una red, donde se dividen todos los procesos en nodos y estos se comunican mediante mensajes. Esta estructura de grafos obliga a modularizar el código y permite reutilizar muchos módulos entre proyectos. Entonces se define una red ROS como un conjunto de nodos comunicados por mensajes, mensajes que viajan a través de buses llamados "topics".

6.1.2.1. Nodos

Los Nodos se pueden interpretar, en un intento de crear un análogo a la programación convencional, como funciones que toman variables y devuelven resultados, o realizan acciones según los valores adquiridos. Para que la reutilización de código sea más sencilla, se debe procurar crear un nodo por cada acción o proceso que deba realizar el robot. Por ejemplo, el driver de un sensor que realiza y publica la medida debe ser un nodo independiente de otro que luego realice una operación sobre esa medida, a modo de ejemplo para estimar un estado del robot.

6.1.2.2. Topics

Los topics funcionan como vías de información y en el esquema de grafos representan los arcos que unen los nodos. Cada topic es único y los nodos pueden subscribirse a ellos para recibir la información que por otro lado otros nodos publican. De esta manera la comunicación entre los nodos es descentralizada. Además los topics tienen asociado un tipo de mensaje para ordenar aún más la comunicación entre los nodos.

6.1.2.3. Nodo Fundamental: ROS Master

Para que la Red de Nodos ROS sea funcional se debe definir un nodo principal. Este nodo se llama "ROS Master", y provee como servicio el registro de los topics y nodos de la red. Es importante destacar que luego de que un nodo se registra en una red ROS, reportando a qué topics subscribe y a cuáles publica, la comunicación no está centralizada por el nodo Master si no que funciona de manera independiente. Es decir que sólo funciona a modo de registro y para hacer dinámica la adición de un nuevo nodo a la red.

6.1.2.4. Mensajes

Los mensajes entre nodos también tienen estructuras predefinidas, se organizan primero en tipos, como mensajes de sensores, navegación, geometría, trayectoria, entre otros. Dentro de los tipos de mensajes, se categoriza a partir del origen de los mismos, como por ejemplo dentro de los mensajes de sensores existen los mensajes de rango, IMU, campo magnético, temperatura, entre otros.

Todos los mensajes contienen un encabezado conformado por un número secuencial, un timestamp y una identificación del nodo de origen. Adicionalmente,

todos los valores numéricos que se transmiten mediante mensajes incluyen la correspondiente covarianza asociada a la medida. Esta estructura facilita el envío y recepción de información entre nodos y permite tener siempre un seguimiento de qué tan actual es la información recibida. La cronología de los mensajes es clave para los algoritmos que se mencionarán más adelante, donde se debe mantener información actualizada en todo momento para el correcto funcionamiento.

6.1.2.5. Frames

ROS mantiene la información de posición y orientación de los distintos elementos del robot en forma de "frames". Estos son marcos de referencia solidarios a cada elemento del sistema. Los frames permiten transformar la información asociada a un eje de referencia, por ejemplo la medida de un sensor en la parte delantera de un robot, y transformarla a otro eje de referencia, por ejemplo uno fijo en el ambiente.

Esta transformación se realiza a partir de un package incluido en las funcionalidades básicas de ROS llamado "tf". tf permite publicar en un topic especial las transformaciones entre los distintos frames. Cada mensaje contiene la identificación de los frames entre los que se desea transformar, y la traslación y rotación necesarias para transformar de uno a otro.

Durante el funcionamiento del robot se puede ejecutar un comando para visualizar la transformación que se está publicando entre dos frames dados obteniéndose un resultado del estilo del siguiente:

```
    At time 1263248513.809
- Translation: [2.398, 6.783, 0.000]
- Rotation: in Quaternion [0.000, 0.000, -0.707, 0.707]
in RPY [0.000, -0.000, -1.570]
```

Esta es la base de la comunicación de información entre elementos físicamente distribuidos en ROS, desde un robot queriendo publicar su posición respecto a un frame absoluto en el ambiente hasta la comunicación entre robots en un enjambre que juntos forman un mapa a partir de múltiples pequeñas observaciones.

A partir de las transformaciones publicadas se puede crear un 'árbol' de transformaciones en el que se detalla la dependencia entre los distintos frames y por lo tanto las transformaciones que se deben aplicar para cambiar la información de referencial.

6.1.3. Filosofía de desarrollo

La reutilización del código existente es uno de los puntos más importantes de la filosofía de desarrollo que se intenta promover para la utilización de ROS.

Como se explicó anteriormente, una de las grandes ventajas de utilizar ROS como plataforma de desarrollo de software para un robot es que cuenta con librerías, para todo tipo de aplicaciones, que son fácilmente implementables.

Parte de la facilidad para la implementación deriva de que todas las librerías o nodos disponibles cuentan con extensa documentación, detallando la funcionalidad

del nodo en su totalidad, así como los topics a los que subscribe y publica. Adicionalmente, la documentación de los nodos presenta los parámetros modificables de los mismos y los rangos de valores que pueden tomar. Esta filosofía de diseño brinda gran flexibilidad al desarrollador para implementar con facilidad un nodo que se adapte a las características del robot en cuestión.

También en pos de la flexibilidad para el desarrollador, es que ROS es agnóstico en cuanto a lenguaje de programación, pudiendo elegir entre C++ y Python para el desarrollo de nodos, pero con la salvedad de que un nodo tiene que estar escrito en un único lenguaje. No es necesario elegir uno solo de estos lenguajes para todo el proyecto a desarrollar dado a que ROS cuenta con las APIs[3] 'roscpp' y 'rospy' respectivamente, que hacen la interfaz entre nodos y permiten la comunicación entre estos aunque estén escritos en distintos lenguajes.

6.2. Solución de software

6.2.1. Resumen

En esta Sección se profundizará en la implementación en software de todas las funcionalidades descritas en la Sección 2.8, detallando los packages de ROS utilizados y sus resultados.

6.2.2. Interacción con el Entorno

Para controlar los motores del robot y utilizar todos los sensores salvo el LiDAR se utilizó un Teensy como fue comentado en la Sección3. Para programar el microcontrolador en el sistema embebido se decidió utilizar el Arduino IDE con un complemento que permite utilizar dicha placa con el entorno de programación, el complemento se llama Teensyduino.

Las funciones que cumbre el microcontrolador son:

- Configuración de IMU y manejo de datos provenientes del sensor
- Comando de Ultrasonidos
- Interacción con Encoders
- Cálculo de Odometría
- Control de las Ruedas
- Comunicación con la red ROS

Además cada función se lleva bajo un control de tasas de muestreo para asegurar un funcionamiento coherente y consistente a lo largo del uso del robot. Asegurando funciones de tiempo real para nodos de más alto nivel, como la planeación de trayectorias.

[3]Application Programming Interface

6.2.2.1. Comunicación con la IMU

Para convertir la información registrada por la IMU, y que pueda ser utilizada por los algoritmos implementados en ROS, se utiliza una librería creada por Sparkfun llamada 'IMU9250' [38]. Esta biblioteca permite abstraer la comunicación I^2C entre el Teensy y la IMU, creando un objeto asociado al sensor. De esta manera simplifica la lectura del código, permitiendo crear un código más útil pero más conciso.

En el código final la tasa de muestreo asignada fue de 100Hz. Entonces, cada 10ms se releva y publica en ROS: las velocidades angulares de cada eje del robot y las aceleraciones, una por eje cartesiano. En esta Sección se utilizaron los mismos ejes de referencia que los detallados en el Capítulo 4.

6.2.2.2. Comando de sensores de ultrasonido

A diferencia del manejo de la IMU, para los sensores de ultrasonido, se implementó una librería a medida para resolver el problema. Como el sistema consta de 4 sensores de ultrasonido que son lentos respecto al resto de las funciones del robot, es necesario realizar la medida de ultrasonido de manera no intrusiva con respecto al resto del procesamiento del sistema. Es importante remarcar esto ya que si las medidas de los sensores de ultrasonido se realizaran sin paralelizar, es decir ocupando tiempo del procesador, el período del sistema estaría restringido a el período de los sensores de ultrasonido y sería demasiado lento para controlar el robot. A modo de ejemplo, asumiendo que la velocidad del sonido en el aire es $343,2\frac{m}{s}$ entonces si se limita a sensar hasta 4 metros desde la ubicación del sensor, el tiempo de la medida de cada sensor sería de $T_{US} = \frac{8m}{343,2\frac{m}{s}} = 23,33ms$. Entonces, medir los 4 sensores a bordo llevaría $93,24ms$, restringiendo la frecuencia del sistema a $10,725Hz$.

Para sobre pasar este sistema se utilizaron las llamadas de rutinas causadas por interrupciones. Entonces se puede seguir utilizando el resto del sistema mientras que la onda de sonido viaja a través del aire.

Teniendo presente la frecuencia máxima de los ultrasonidos como conjunto presentada anteriormente se decidió configurar el sistema para muestrear el conjunto de ultrasonidos a 5 Hz. Entonces cada 200ms se publica el último set de medidas de ultrasonido y se lanzan las nuevas medidas.

6.2.2.3. Interacción con Encoders

Para llevar la cuenta de los encoders, en cada rueda se utilizó una librería[39] dedicada que funciona plenamente por interrupciones. Para cada encoder se creo un objeto de tipo 'Encoder' especificando los pines asociados. Las cuentas de los encoders se realizan de manera independiente del transcurso del programa principal y se consultan los registros de las cuentas para saber cuanto se movió la rueda en el período seteado en la configuración principal.

Luego se estima la velocidad de cada rueda mediante el cálculo expuesto en la Sección 5.1.2.1 para poder utilizar el resultado, tanto para estimar la odometría

como realizar control activo sobre cada rueda.

Para asegurar un buen control de las ruedas y tener información a la hora de calcular la odometría se configuraron los encoders para ser muestreados cada 1 ms en conjunto con el cálculo de odometría. Entonces ambas partes tienen una frecuencia de muestreo de 1 kHz

6.2.2.4. Cálculo de Odometría

El cálculo de la Odometría permite ubicar el robot en el entorno respecto a una posición inicial.

Este cálculo se engloba en una función única que transforma las velocidades de las ruedas del robot a la velocidad del dispositivo como conjunto. Los detalles de la operatoria fueron aclarados en la Sección 4. La actualización de la odometría se renueva cada 1 ms al igual que los encoders.

6.2.2.5. Control de las Ruedas

La velocidad deseada de cada rueda se calcula a partir de la velocidad relativa que se espera del robot. Esta velocidad relativa se obtiene, a través de la red ROS, de otros nodos como el de planeación de trayectorias.

Entonces, el Teensy recibe los comandos de velocidad a través de la red ROS y calcula la velocidad que debe tener cada rueda para alcanzar el movimiento comandado. Como se detalló en el Capítulo 5, cada rueda tiene un controlador PID asociado para mantener la velocidad necesaria para realizar los movimientos de manera correcta.

Para implementar el control PID se utilizó la librería recomendada por Arduino [3], ya que cuenta con la funcionalidad de 'anti-windup' y realiza los cálculos de manera eficiente en términos temporales. La actualización de la salida del controlador se renueva a una tasa de 500 Hz.

6.2.2.6. Comunicación con la red ROS

Para asociar los datos generados por el microcontrolador con la red ROS y recibir mensajes de otros nodos, se utilizó la librería dedicada rosserial-arduino. Esta biblioteca permite abstraerse de la comunicación serial por UART y manejar la interacción entre el Raspberry Pi y el Teensy como si el código de este último fuera un nodo más de la red ROS.

Para utilizar la librería hay que correr una rutina en la computadora en la que se quiere conectar, que en este caso es el Raspberry Pi, y agregar una configuración inicial en el código del microcontrolador que incluye los topics a los que se subscribe y en cuales publica. Además en el microcontrolador se debe incluir el llamado periódico a una función, incluida en la biblioteca mencionada en esta Sección, que resuelve la comunicación con ROS de manera dinámica. Dicha función sirve como punto de control para resolver sincronización de tiempos con la Red y atender llamados por subscripciones.

Entonces, se logra publicar la información generada por los sensores en cada topic elegido y con el formato de mensaje indicado. Por otra parte se actúa sobre los mensajes recibidos, generados por otros nodos en los topics seleccionados.

6.2.3. Localización

Tal como se discutió en el Capítulo 5, Sección 5.2, el EKF implementado no dio resultados aceptables por lo que se optó por implementar un package dentro de los provistos por el Navigation Stack de ROS, *robot_pose_ekf*.

Este package es usado para estimar la pose 3D de un robot, ubicación y orientación en el espacio, basado en mediciones parciales de la pose obtenidas a través de sensores. Usa un filtro de Kalman extendido 6D (3D de posición, 3D de orientación) para combinar medidas de odometría de las ruedas, una IMU y odometría visual de una cámara [28]. En esta implementación se desactivó la función de odometría visual dado a que el robot no cuenta con una cámara.

El nodo que implementa este package subscribe a los topics '*odom*' e '*imu_data*', de los que recibe los datos de los sensores. Los mensajes que recibe de los sensores son de acuerdo a la estructura detallada en la Sección 6.1.2.4.

Una vez procesada la pose del robot, se publica en el topic '*odom_combined*', que contiene el resultado del filtro. Además, se publica una transformación tf del frame 'odom_combined' al 'base_link' que indica la localización y orientación del robot, al que corresponde el frame base_link, respecto al referencial absoluto, al que corresponde el frame odom_combined.

Esta transformación se usa para obtener la localización del robot en todo momento y también para transformar todas las medidas y comandos relativos al robot al referencial absoluto y viceversa.

6.2.4. Mapeo

El mapeo del entorno se lleva a cabo mediante el algoritmo de SLAM llamado Gmapping, mencionado y explicado en la Sección 5.3. Específicamente se utilizó una implementación en C++ creada por la organización OpenSLAM [29] que se encarga de difundir sobre el tema y ayudar a desarrolladores a llevar a tierra los conceptos abstractos de diferentes trabajos de investigación. Esta implementación del algoritmo se encuentra dentro de un nodo ROS, llamado 'slam_gmapping', que se encarga de hacer de interfaz entre el algoritmo y ROS, publicando el mapa en un topic de mapeo si se atienden los requerimientos.

El nodo 'slam_gmapping' se subscribe a los datos de un sensor láser 2D paralelo al plano del piso,en nuestro caso el LiDAR, y a un proveedor de odometría. En el sistema implementado en este proyecto la odometria se obtiene a través del nodo de localización detallado en la Sección 6.2.3. La odometría se usa en parte para ubicar el mapa con respecto a la posición del robot mediante transformaciones tf. También es necesario tener una transformación tf entre el eje de coordenadas del LiDAR y la base del robot ,'base_link' mencionada anteriormente. Entonces se creo un nodo que publica periódicamente la transformada entre la base del robot y

la posición del sensor laser que consta de una translación en un eje ya que comparte los otros dos. De esta manera se puede relacionar siempre la fuente de información, el LiDAR , con la posición inicial.

Con los requerimientos antes mencionados el nodo publica periódicamente el mapa generado, siendo este de grillas de ocupaciones.

Por otro lado el mapa generado por el nodo 'slam_gmapping' puede ser guardado por comando del usuario. Para guardar el mapa se utilizo otro nodo 'map_server' que registra los mensajes publicados y los guarda en un formato que permite ser levantado posteriormente para su uso. Los archivos generados por este nodo son dos: un '.YAML' que contiene los metadatos asociados al mapa, es decir toda la información relevante que no es el mapa en sí, y una imagen que contiene el mapa generado basado en grillas de ocupaciones.

Un mapa resultante se puede apreciar en la Figura 6.2.1

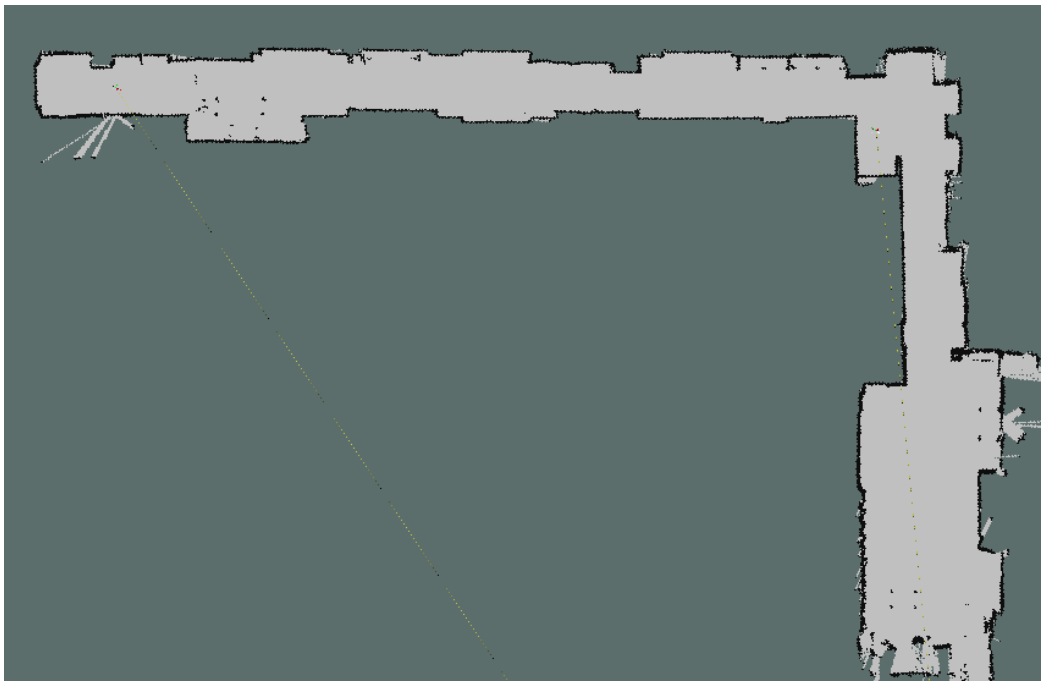


Figura 6.2.1: Ejemplo de mapa generado por el robot

6.2.5. Planeación de trayectorias

La planeación de trayectorias se realiza implementando los packages '*move_base*', '*costmap_2d*', '*dwa_local_planner*' y '*global_planner*' del navigation stack. En conjunción con los algoritmos de sensado, control, mapeo y localización antes descritos es que esta combinación de nodos logra la planeación y seguimiento de trayectorias de forma autónoma. A continuación se detallará el propósito y el funcionamiento de cada package y como se integra al software del robot.

6.2.5.1. Descubrimiento de obstáculos

El package costmap_2d crea un mapa de costos a partir de los sensores del robot. Este mapa de costos, formado por celdas de tamaño configurable, cuantifica el ambiente alrededor del robot, dándole un valor entre 0 y 255 a cada celda, indicando el costo que implica al robot moverse por ella.

En la escala antedicha, un valor de 0 significa que la celda está completamente libre y por lo tanto el robot puede transitarla sin problemas. Por otro lado, un costo de 255 significa que el costo para el robot es 'letal', es decir que debe evitarse a toda costa al planear una trayectoria. Los valores intermedios a estos dos implican que por esas celdas puede planearse una trayectoria, pero que esto se lleve a cabo dependerá de las características de la situación y como haya sido configurado el planeador.

Para asignar costos a cada celda, este nodo utiliza la información del LiDAR principalmente, que complementa con los sensores de ultrasonido en el robot. Al detectarse un obstáculo, las celdas que este cubre son marcadas con un costo de 255 en el mapa.

Alrededor del mismo se genera una zona de costo no nulo, cuyo valor es máximo donde termina el obstáculo y disminuye progresivamente a medida que se aumenta la distancia respecto al mismo. Esta zona se genera para lograr que el robot navegue tomando cierta distancia de los obstáculos cuando sea posible y es configurable a través de dos parámetros *'inflation_radius'* y *'cost_scaling_factor'*. El primero determina el tamaño de esta zona de inflación alrededor del obstáculo y el segundo es qué tan rápido decae el costo con la distancia. Estos conceptos se discuten también en la Sección 5.4.4.

En la Figura 6.2.2 se puede observar el mapa de costos local con los obstáculos o zonas letales marcadas en color gris más oscuro. Estos se superponen con los obstáculos que detecta el LiDAR y conforman el mapa del ambiente. En las celdas adyacentes a los obstáculos se puede apreciar como el color del mapa de costos se va volviendo más claro, indicando menor costo, debido a los parámetros inflation_radius y cost_scaling_factor elegidos.

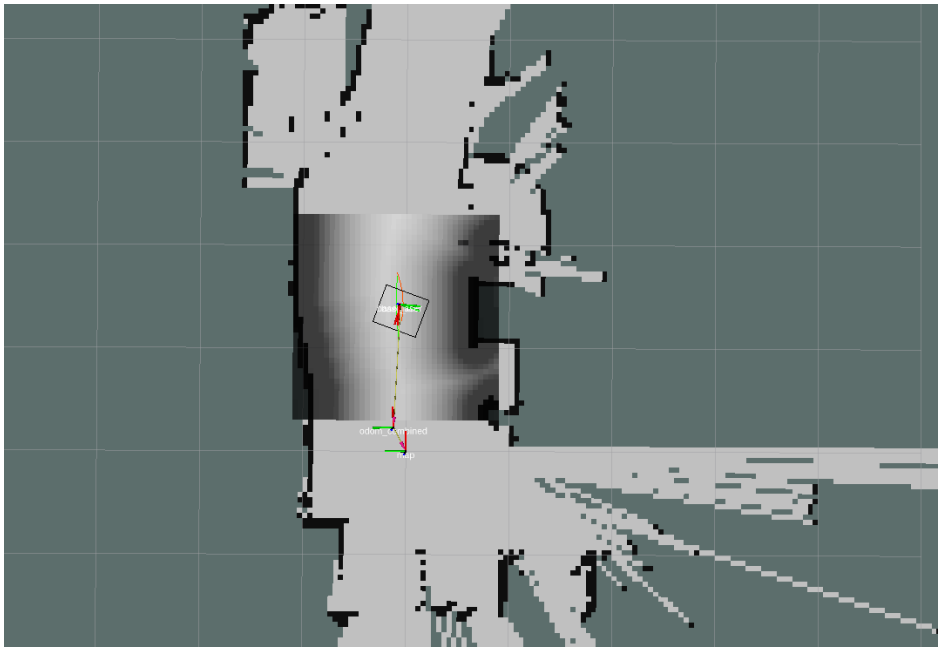


Figura 6.2.2: Mapa de costos superpuesto con mapa del ambiente

Por último, este nodo publica la '*footprint*' del robot, un polígono centrado en el frame solidario al robot que representa la geometría del mismo. Esta footprint puede observarse en la Figura 6.2.2 como el rectángulo negro en el centro del mapa de costos. La footprint permite visualizar en el mapa la dimensión real del robot, que es utilizada por el planeador de trayectorias para decidir que trayectorias son viables.

6.2.5.2. Planeador global de trayectorias

Como se explicó en la Sección 5.4, la planeación de trayectorias se realiza en dos etapas, una local y otra global. En esta Sección se detallará el funcionamiento del planeador global.

Este nodo calcula una trayectoria para el robot desde el punto de partida hasta la meta. Permite elegir el algoritmo utilizado para determinar la trayectoria de los posibles que se discutieron en la Sección 5.4. En particular se seleccionó el algoritmo Dijkstra.

Tomando como base el mapa de costos elaborado por el package anterior y, si existe, el mapa del ambiente, se pueden definir parámetros como: a partir de qué costo se debe considerar una celda como letal, o cuál es el umbral a partir del cuál no se considera más una celda como neutral. Otro parámetro muy importante para el desarrollo de un robot que debe mapear a medida que recorre el ambiente es que se tenga en cuenta el espacio desconocido, todavía no mapeado, como posible camino hacia la meta.

Finalmente, este nodo publica una trayectoria para que el planeador local pueda definir los comandos del robot. Este nodo constantemente reevalúa la situación del

robot a partir del mapa que se va construyendo y recalcula la trayectoria, de ser necesario. Para esto se define el último parámetro, la frecuencia de planeación del nodo.

En la Figura 6.2.3 se puede visualizar la trayectoria global en color verde, marcando el camino por la zona de menor costo del mapa de costos, y que termina en la flecha roja que simboliza la meta fijada. Adicionalmente, en la Figura 6.2.4 se puede observar que el robot es capaz de planear su trayectoria global aún cuando esta es definida en una zona de la que no se tiene información.

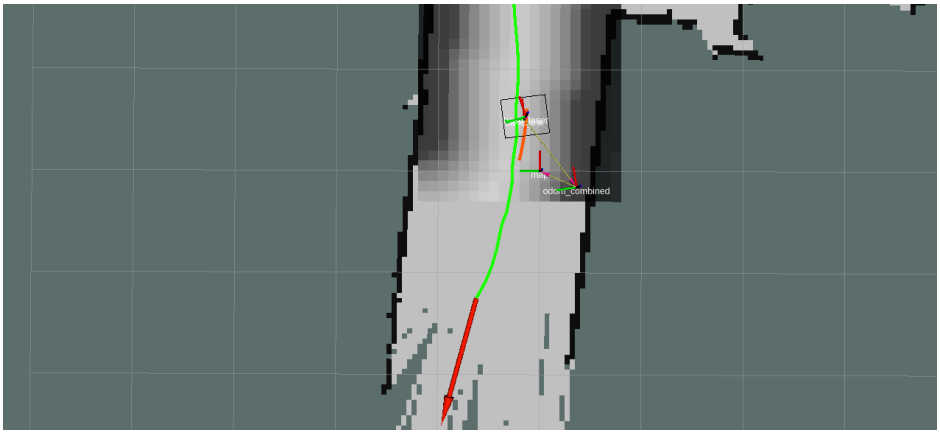


Figura 6.2.3: Robot con una trayectoria global planeada

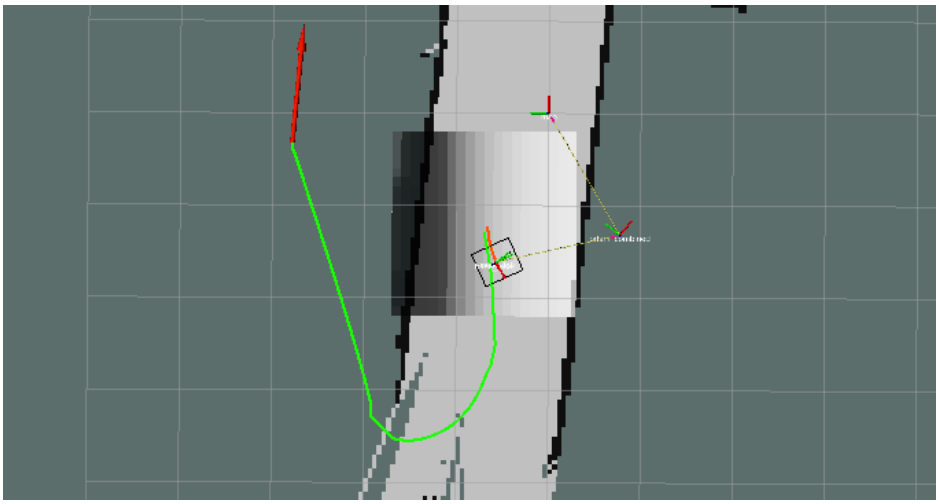


Figura 6.2.4: Robot con una trayectoria global hacia un objetivo en una zona desconocida

6.2.5.3. Planeador local de trayectorias

El nodo planeador local se implementa a través del package dwa_local_planner, que aplica la técnica Dynamic Window Approach que se discute en la Sección 5.4.3, que basándose en el mapa de costos creado por el nodo anterior y una trayectoria

global, planea la trayectoria óptima en el entorno local a partir de un enfoque de ventana dinámica. Esto significa que la trayectoria local está restringida a una ventana de dimensiones configurables que se desplaza junto con el robot.

Este nodo permite un gran espectro de configuraciones en cuanto a cómo se determina la trayectoria del robot y las velocidades para recorrer la misma.

Para determinar la trayectoria óptima, el nodo simula varias posibles trayectorias basadas en la información obtenida de los nodos anteriormente descritos. Algunos de los parámetros más importantes que se pueden configurar son cuánto tiempo hacia adelante se simula, el tamaño del paso entre puntos de una trayectoria simulada, la cantidad de muestras que se utilizan para simular las velocidades y la frecuencia con la que se simula. Incrementar cualquiera de estos parámetros produce mejores resultados pero aumenta a su vez el costo computacional del sistema, que puede implicar que no se logre mantener la frecuencia de funcionamiento requerida.

Una vez configurado el nodo para seleccionar la óptima trayectoria, se deben ajustar los parámetros que gobiernan el seguimiento de dicha trayectoria. Estos incluyen las aceleraciones y velocidades, lineales y angular, para el seguimiento de la trayectoria.

Adicionalmente, se pueden definir tolerancias respecto a cómo se debe seguir la trayectoria elegida, a qué distancia de la meta y con qué error en la orientación se puede considerar la meta alcanzada.

La trayectoria local se muestra tanto en la Figura 6.2.5 como en la 6.2.3 en color rojo, en este caso muy cercana a la trayectoria global en verde. Puede apreciarse que esta tiene la forma de un arco de circunferencia, como se explicó en la Sección 5.4.3 y está solamente acotada al entorno local.

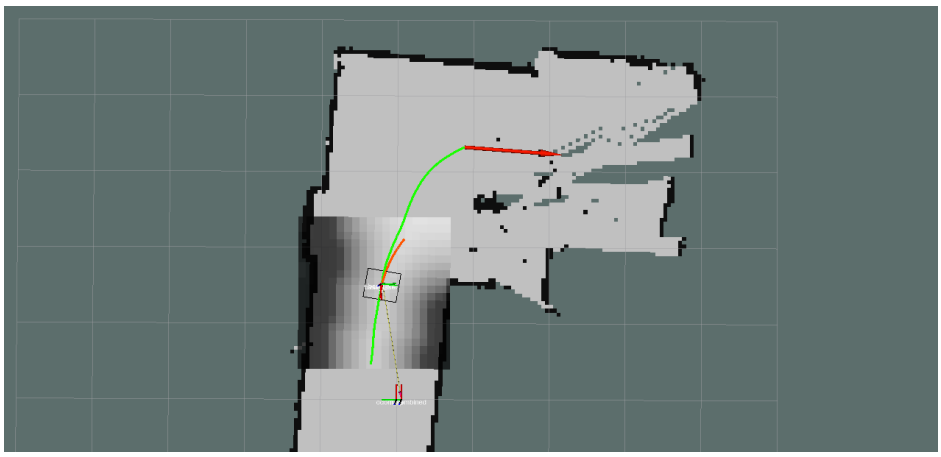


Figura 6.2.5: Planeación de trayectorias local

Finalmente, los comandos de velocidad calculados para el seguimiento de esta trayectoria por el nodo dwa_local_planner son enviados al último nodo, move_base. Al igual que el planeador global, este nodo también actualiza constantemente la trayectoria local en función de la información recibida y también puede configurarse la frecuencia con la que lo hace.

6.2.5.4. Seguimiento de trayectorias

Por último, una vez que se cuenta con una trayectoria global, una local y los comandos de velocidad necesarios para el seguimiento de estas, estos comandos son recibidos por el nodo move_base. Este nodo, funcionando en un robot con los parámetros de navegación antedichos correctamente seleccionados y en la ausencia de obstáculos dinámicos, hará que el robot eventualmente alcance la meta elegida dentro de las tolerancias configuradas.

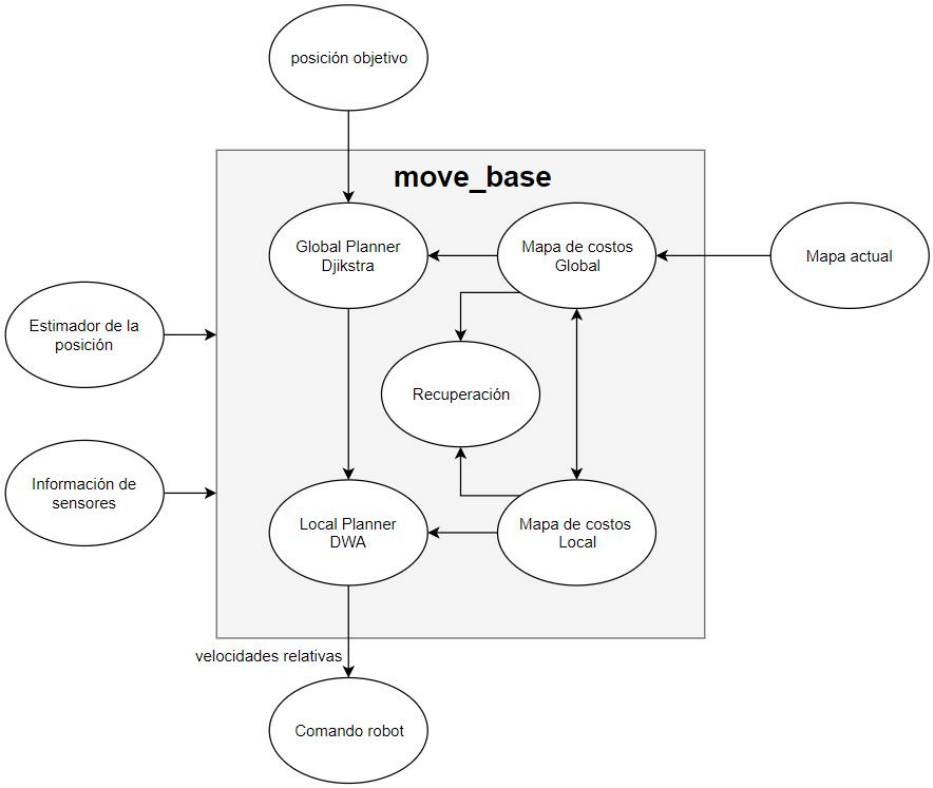


Figura 6.2.6: Diagrama de bloques de move_base

Este nodo realiza la importante tarea de centralizar toda la información provista por los anteriores y producir los comandos para el robot, como puede observarse en la Figura 6.2.6. Para esto también cuenta con parámetros como la frecuencia de control, la distancia lineal o angular que el robot debe recorrer para considerar que no se encuentra oscilando, el tipo de acciones de recuperación que puede realizar y la cantidad de veces que debe reintentar planear un camino cuando el elegido no es posible.

Cuando el robot detecta un atascamiento, es decir que después de un tiempo configurado por el usuario es incapaz de planear una trayectoria hacia la posición objetivo, se activa el mecanismo de recuperación.

En la Figura 6.2.7 se puede observar la máquina de estados del sistema de recuperación. Una vez que el robot detecta un atascamiento, irá a un estado de

recuperación. Si en este estado logra desatascarse, vuelve al estado de navegación, si no lo logra pasa a un estado siguiente con un medida más agresiva.

La iteración se repite 3 veces, cada vez yendo a un estado donde las medidas para recuperarse son mayores. Si en el último estado no logra desatascarse se avisa al usuario y se aborta la misión.

Los estados de recuperación son los siguientes:

- **Reset Conservativo:** Los obstáculos fuera de un área configurada por el usuario son eliminados del mapa
- **Rotación de despeje:** El robot realiza una rotación en el lugar, con el objetivo de despejar nuevas celdas. En caso de que haya detectado obstáculo erróneamente.
- **Reset agresivo:** Se despejan todos los obstáculos exteriores al footprint.

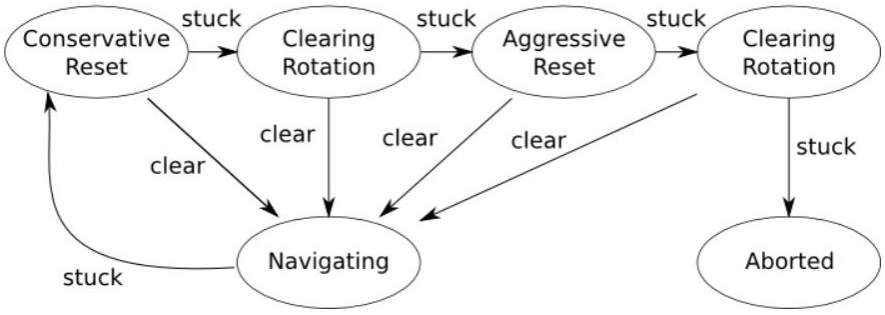


Figura 6.2.7: Máquina de estados del sistema de recuperación

Finalmente, este nodo también es el encargado de imprimir mensajes al usuario respecto al estado del robot, si se está cumpliendo con las frecuencias de funcionamiento de los distintos nodos, si el robot se encuentra trancado, y si la meta fue alcanzada o esta tarea no puede lograrse dadas las condiciones.

6.2.5.5. Calibración del navigation stack

Como se puede apreciar de las secciones anteriores, el navigation stack de ROS permite personalizar una gran variedad de aspectos de la navegación autónoma para ajustar el mismo al robot en cuestión y sus misiones.

Este es un proceso complejo y repetitivo ya que requiere en gran parte que se trabaje a partir de ensayo y error. Para facilitar este proceso se utilizaron las fuentes bibliográficas [44] y [27], junto con la documentación de cada package individual para realizar este proceso de forma más efectiva e informada.

Es importante notar que solamente con la información de los packages separados no se puede garantizar el mejor resultado de configuración de navegación ya que se debe entender cómo estos interactúan y cómo los parámetros modificados en uno afectan el funcionamiento del otro.

6.2.6. Exploración

Para la exploración del espacio se utilizó el package Explorer_lite. Se trata de un explorador basado en detección de fronteras, como se explicó en la Sección 5.4.6.1, que se utiliza para generar posiciones objetivo que al ser utilizadas por move_base, permiten explorar el ambiente de forma eficiente.

Claro está que para poder utilizar este explorador, es imprescindible que se encuentre funcionando correctamente move_base, siendo posible navegar por el ambiente a través de la interfaz de Rviz.

Se suscribe al mapa proveniente de gmapping a través del topic 'map' y le aplica el algoritmo de detección de fronteras. Una vez detectadas, las fronteras son publicadas en el topic 'frontiers' y podrán verse en Rviz. Estas fronteras serán la posiciones objetivos que tomará move_base.

6.2.7. Interfaz

La interfaz entre el usuario y el robot está formada por dos partes principales que interactúan entre sí para permitir al usuario visualizar en tiempo real el estado del robot, el mapa y las trayectorias, así como elegir las misiones y guardar el mapa realizado.

6.2.7.1. Rviz

La primera de estas partes es un nodo llamado *'Rviz'*, incluido dentro de las funcionalidades básicas de ROS. Este nodo es un visualizador 3D que permite mostrar la información de sensores y el estado del robot.

Rviz es el único package de ROS implementado en este desarrollo que genera una ventana separada para visualizar los topics existentes de manera gráfica, interactuar con el sistema publicando mensajes y realizar configuraciones. A continuación se profundizará en estas tres funciones.

La interfaz de Rviz permite visualizar una lista de todos los topics existentes a los que corresponda una representación física y desplegar dicha representación en la interfaz, como puede observarse en la Figura 6.2.8. Los de principal importancia para el robot en cuestión son los topics en los que se publica el mapa, las trayectorias local y global, y el mapa de costos local.

Adicionalmente, también se encuentran disponibles para visualización los distintos frames que componen el sistema. Esto permite observar la posición de cada uno y como se desplazan uno respecto al otro.

La conjunción de los dos puntos anteriores permite representar el movimiento del robot respecto al referencial absoluto a medida que se construye el mapa del ambiente. Adicionalmente, en modo de funcionamiento autónomo se puede visualizar la ventana deslizante en torno al robot con el mapa de costos indicando los obstáculos, con la trayectoria local actualizándose a partir de los mismos.

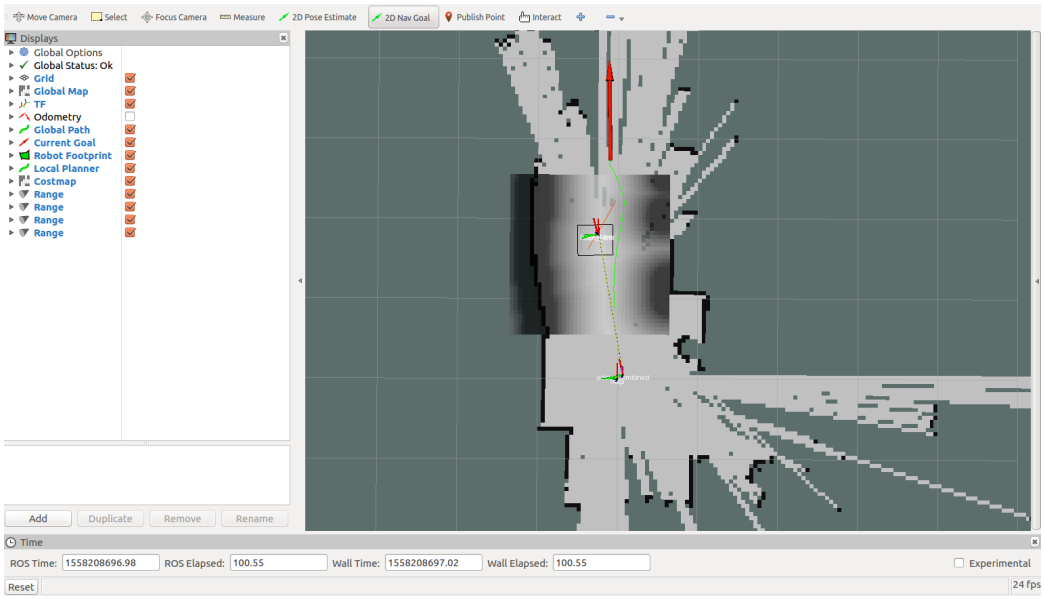


Figura 6.2.8: Imagen de Rviz con sus menús y la ventana de visualización

Como se puede observar en la Figura 6.2.8, en la Sección superior se encuentran los botones que permiten interactuar con el sistema. Los primeros 5 botones, comenzando desde la izquierda, tienen como función interactuar con el modelo 3D para cambiar el punto de vista, seleccionar elementos y tomar medidas. Estas acciones no afectan al robot sino simplemente a la visualización.

Por otro lado, las siguientes opciones permiten interactuar directamente con los demás nodos. En particular, en la implementación actual estos botones permiten enviar mensajes a los que subscribe el nodo move_base para definir un estimado de la posición y fijar metas. Esta última es la opción utilizada en modo autónomo para definir las metas del robot con el botón *'2D Nav Goal*, que permite crear una flecha en la interfaz con el origen de esta flecha definiendo la ubicación de la meta y la dirección definiendo la orientación asociada a esa meta. Se puede observar la flecha verde dibujada por el usuario con la nueva meta de navegación en le Figura 6.2.9.

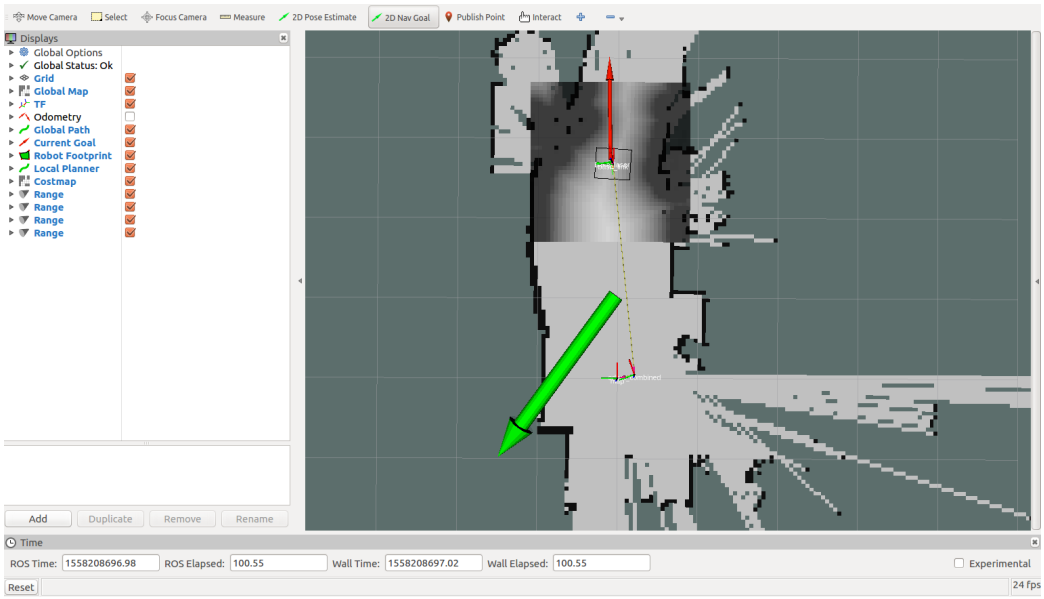


Figura 6.2.9: Definición de metas para navegación a través de Rviz

Finalmente, Rviz permite guardar la configuración utilizada en la ventana abierta, y esto permite cargarla por defecto cada vez que se corra el nodo Rviz nuevamente para siempre mostrar la representación de los mismos nodos en pantalla. Adicionalmente, Rviz soporta plugins, funcionalidades extra desarrolladas por usuarios que pueden integrarse al nodo con facilidad. En la implementación del robot se utilizó un plugin para controlar el robot de forma remota en modo manual a través de una ventana secundaria dentro de Rviz, como se ve en la Figura 6.2.10.

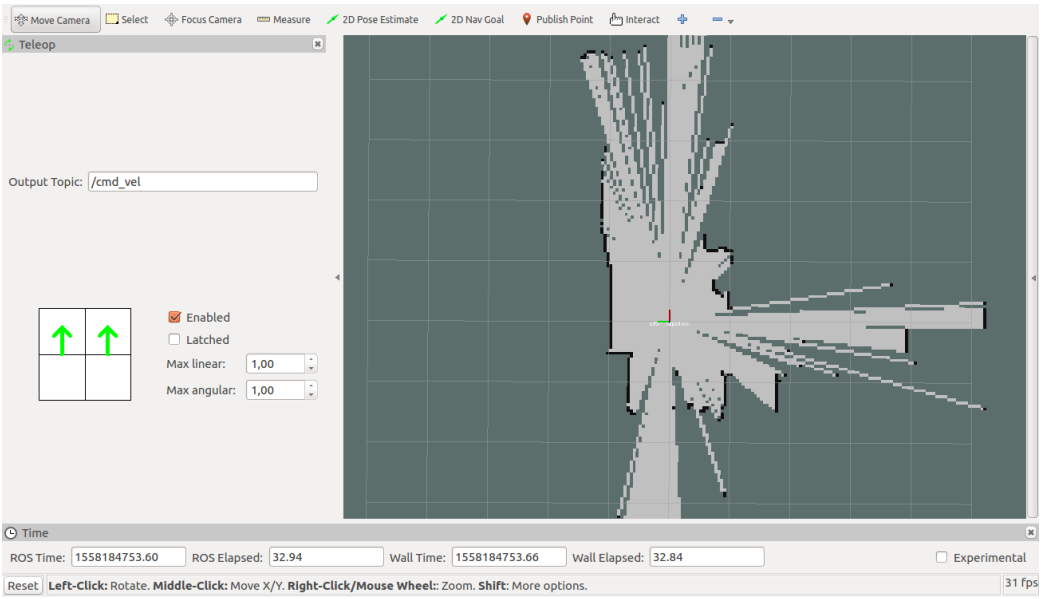
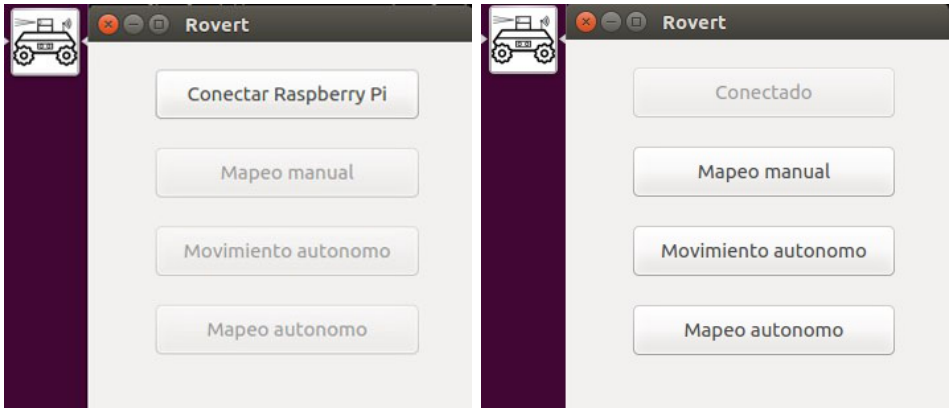


Figura 6.2.10: Plugin de comando de velocidad en Rviz

6.2.8. Selección de misiones

La segunda parte principal de la interfaz de usuario es una ventana que permite: establecer una conexión por WiFi con el robot, elegir las misiones a realizar, cambiar de misión seleccionada y guardar el mapa realizado. Esas funcionalidades fueron implementadas a través de un código de Python utilizando la librería PyQt[32], basada en el framework Qt de desarrollo de interfaces gráficas[41].

Como primer paso se debe seleccionar la opción de conexión con el robot, esto ejecuta un proceso que intenta establecer una conexión SSH con el Raspberry Pi. Si esta conexión es exitosa, procede a activar todos los nodos que corren en esta plataforma, la comunicación serial, el LiDAR y el EKF. En la Figura 6.2.11a se muestra la ventana de la interfaz para realizar la conexión.



(a) Interfaz de conexión con el robot (b) Interfaz de selección de modo

Figura 6.2.11: Interfaz de usuario: menú principal

Una vez completada la conexión, se habilitan las opciones de seleccionar misión. Las disponibles son 'Mapeo manual', 'Movimiento autónomo' y 'Mapeo autónomo', como se puede apreciar en la Figura 6.2.11b.

La primera solamente activa la función de mapeo y abre una instancia de Rviz, con el plugin antes mencionado, para observar el mapa y controlar el robot en una sola ventana.

La segunda activa todos los nodos descritos en la Sección 6.2.5 y permite, visualizando en Rviz, a medida que se crea el mapa, navegar de forma autónoma el ambiente definiendo metas con la función '2D Nav Goal' de Rviz, detallada en la Sección anterior.

Por último, la misión de exploración no requiere ninguna interacción por parte del usuario. Una vez seleccionada, el robot recorrerá el ambiente de forma autónoma, definiendo sus propias metas, hasta haber completado el mapa. El proceso puede observarse en la ventana de Rviz.

Una vez seleccionada cualquiera de las misiones, una ventana secundaria permite guardar el mapa o cambiar a un modo de funcionamiento distinto y así retornar al menú anterior.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 7

Análisis y Validación de Funcionamiento

El siguiente Capítulo plantea y describe la implementación y resultados de las pruebas de validación para el robot. En este Capítulo se incluyen solamente las pruebas cuyos resultados son centrales para la evalución del robot.

7.1. Motores

7.1.1. Respuesta a la Rampa

7.1.1.1. Motivación

Para caracterizar los puntos máximos y mínimos de funcionamiento del motor según el comando del driver integrado se introdujo al sistema una entrada en forma de rampa ascendente y descendente. Como fue repetido en diferentes partes de este documento el driver del motor toma como entrada una señal PWM. Por lo cual la variable a modificar es el ciclo de trabajo del pulso PWM y en el experimento se comenzó de 0 % del ciclo de trabajo y se subió hasta 100 % de ciclo de trabajo de manera gradual. Para luego volver de 100 % a 0 %.

Con esta experiencia se pretende encontrar la histéresis del motor y comprobar que el manejo del motor puede considerarse lineal, ya que se pretende utilizar una relación lineal entre la velocidad en rad/s que se necesita de la rueda en cada instante y el ciclo de trabajo del PWM que debe ser generado por el microcontrolador.

7.1.1.2. Implementación

El código responsable de realizar esta experiencia esta escrito en C++. Utiliza funciones fundamentales de la Librería de Arduino como analogWrite() para generar el pulso de ancho modulado para controlar el driver y escribe por serial la velocidad comandada y la medida por el encoder.

Los datos generados en la experiencia fueron procesados con un script de Matlab, ya que para cada velocidad había más de una medida. El script de Matlab promedia las salidas para cada velocidad comandada para filtrar el ruido.

Para obtener una relación entre el ciclo de trabajo del pulso y la velocidad comandad en radianes por segundo a la rueda se impuso un pulso de 100 % de ciclo de trabajo y se midió la velocidad mediante el encoder del motor.

La velocidad registrada fue 8.507 $\frac{rad}{s}$. Por otro lado la función de Arduino analogWrite() permite utilizar la función de PWM en el microprocesador y toma como máximo valor 255(el máximo valor que se puede obtener con un byte). Es decir que un pulso con ciclo de trabajo de 100 % se consigue pasando como variable 255. Es por esto que la linealización del comando se realiza con respecto a 255. Entonces para generar el ciclo de trabajo según la velocidad que se necesita se realiza la siguiente operación: $PWM_{\omega_i} = \frac{\omega_i \times 255}{8{,}507 \frac{rad}{s}}$

Todos los motores se encontraban con una rueda montada en su eje exterior pero sin rozamiento. Es decir que los motores trabajaron en vacío

7.1.1.3. Procesamiento de los Datos

A continuación se exponen los resultados de las cuatro experiencias superpuestas. Graficando la velocidad de salida ω_o según la velocidad configurada ω_i para cada rueda .

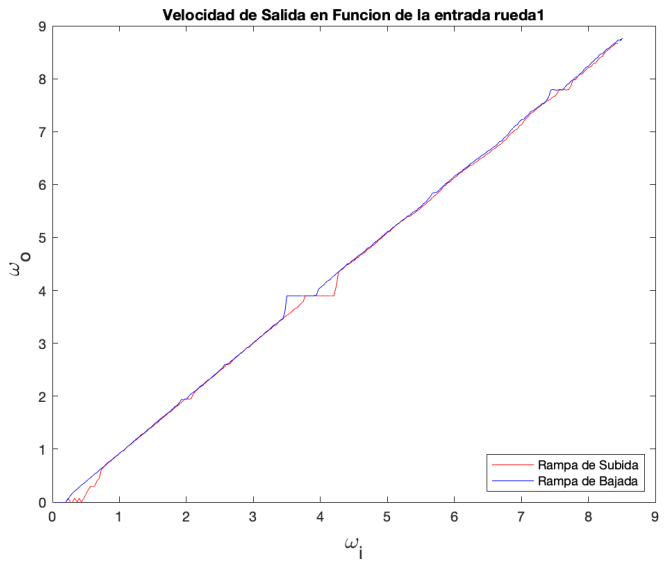


Figura 7.1.1: Velocidad de Salida según la entrada Rueda 1

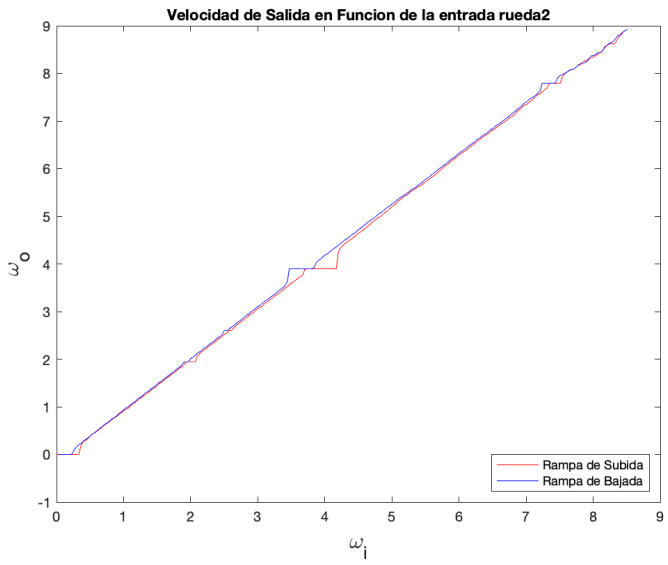


Figura 7.1.2: Velocidad de Salida según la entrada Rueda 2

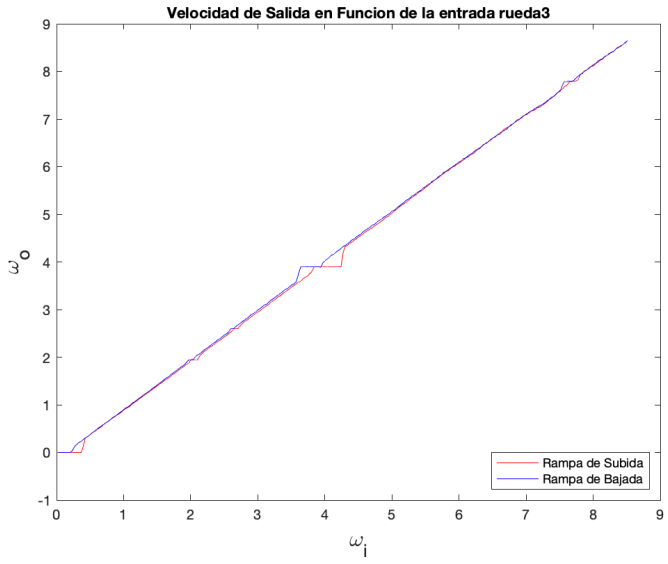


Figura 7.1.3: Velocidad de Salida según la entrada Rueda 3

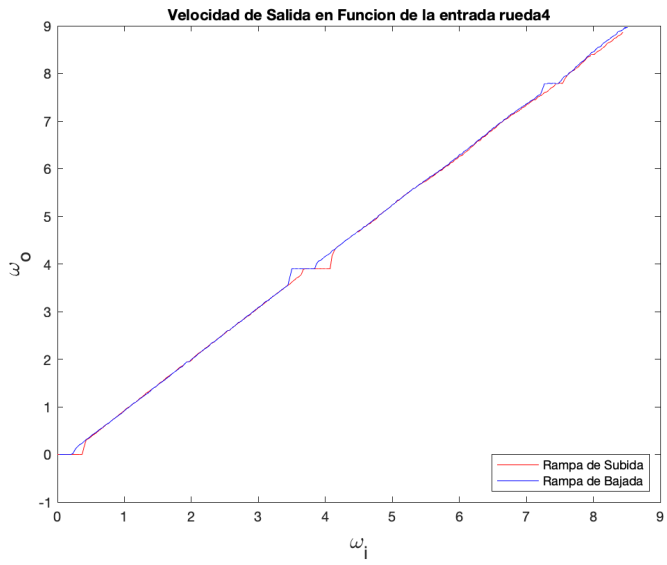


Figura 7.1.4: Velocidad de Salida según la entrada Rueda 4

De los datos antes expuestos es necesario notar que para los 4 motores encontramos un comportamiento muy similar, lo que permite generar un controlador idéntico para cada rueda en una etapa más avanzada. Por otro lado tenemos que según los cuatro experimentos, en el peor caso el motor no puede arrancar con comandos menores a 0.73 $\frac{rad}{s}$. Es importante destacar que la relación entre la entrada y la salida es considerablemente lineal salvo en velocidades puntuales que mediante el sistema de control de velocidad son corregidos.

Por último se muestra que al comparar las cuatro respuestas en la Figura 7.1.5 se encuentra una variación pequeña que también es corregida por el sistema de control.

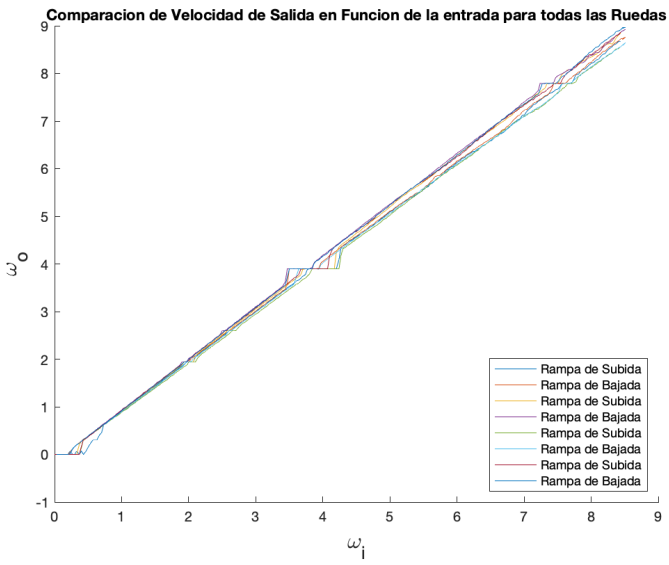


Figura 7.1.5: Velocidad de Salida según la entrada para todas las Ruedas

7.1.2. Respuesta al Escalón

7.1.2.1. Motivación

Para caracterizar la respuesta del sistema, el motor con rueda, se inyecta un escalón y se observa la salida. Se realiza el experimento con cada par rueda-motor para asegurar un modelo común a las cuatro ruedas.

7.1.2.2. Implementación

En este experimento se introducen escalones de la misma amplitud consecutivos en el sistema mientras que se registra la salida en $\frac{rad}{s}$ para ver la respuesta del sistema.

El código responsable de este experimento puede encontrarse en el Anexo pero la lógica básica es un cambio de estado de la salida entre 0 y 3,34 $\frac{rad}{s}$ cada 2 segundo. Se decidió usar este tiempo ya que es mucho mayor al tiempo de subida de la respuesta, observación que se hizo en las primeras pruebas.

Se tomo una muestra cada 10 ms en cada experimento.

Al igual que el experimento anterior todos los motores actuaron en vacío con la salvedad de tener las ruedas montadas a los mismos.

7.1.2.3. Procesamiento de los Datos

Los resultados de la experiencia para cada rueda se muestran a continuación:

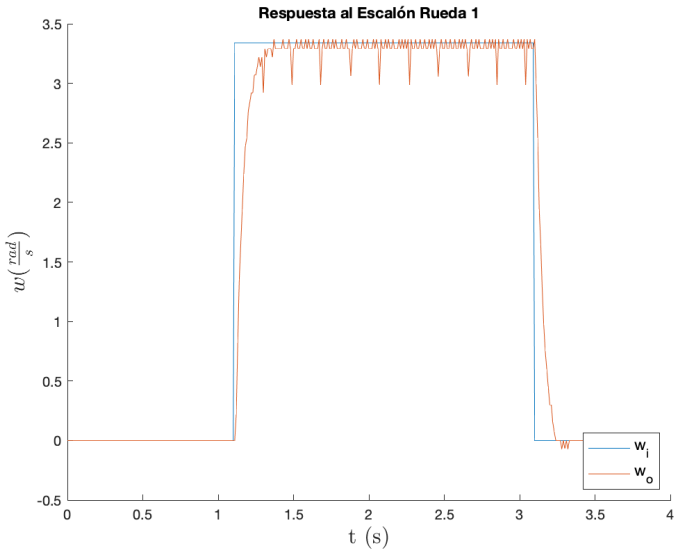


Figura 7.1.6: Respuesta al Escalón Rueda 1

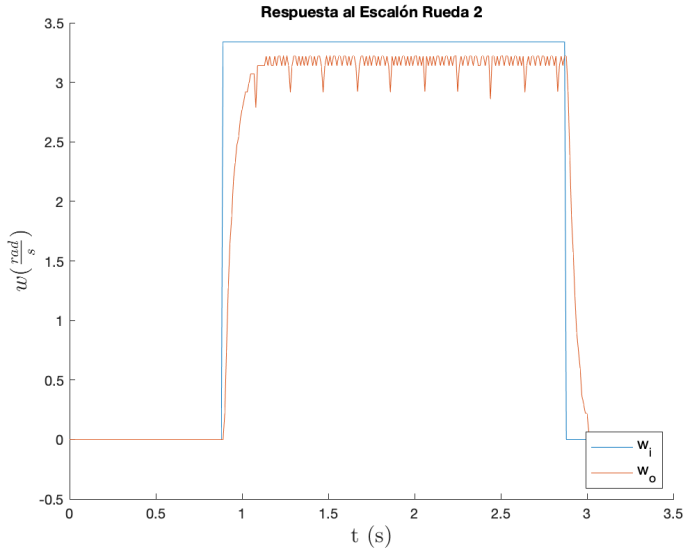


Figura 7.1.7: Respuesta al Escalón Rueda 2

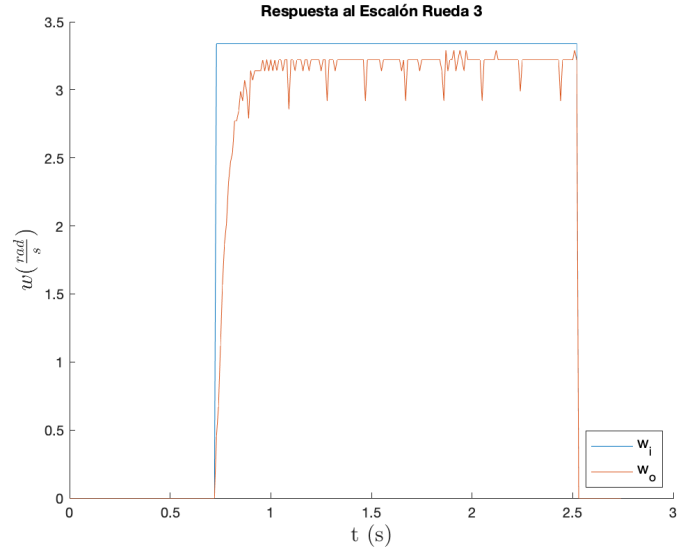


Figura 7.1.8: Respuesta al Escalón Rueda 3

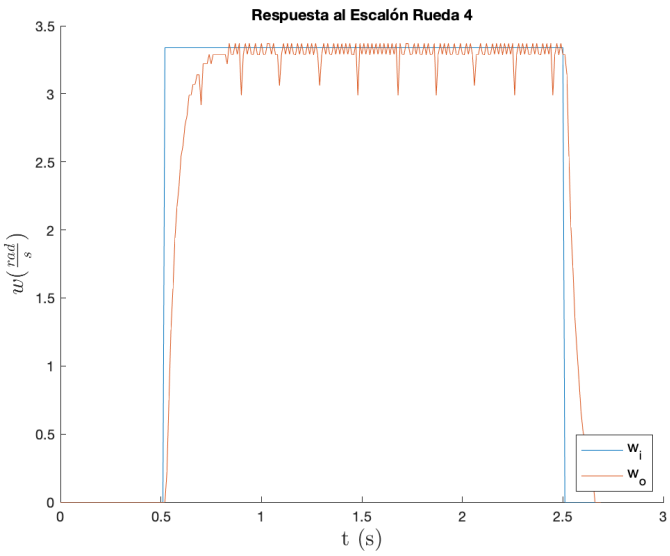


Figura 7.1.9: Respuesta al Escalón Rueda 4

El promedio del tiempo de subida en los experimentos fue de 0.11 segundos. Por otro lado en los tres casos no hubo sobretiro y existe la presencia de un error constante con $t \to \infty$. A partir de los experimentos antes formulados se genera un modelo del sistema. El modelado fue detallado en la Sección 5.1.3

7.2. PID

7.2.1. Motivación

Para verificar el funcionamiento del sistema de control PID tuneado mediante los experimentos antes realizados y con el procedimiento detallado en la Sección 5.1 se realizan experimentos para ver el tiempo de respuesta obtenido. Además se evaluó el andar del robot para determinar qué configuración era la más adecuada tratando de lograr un movimiento continuo y sin sobresaltos.

7.2.2. Implementación

Al sistema utilizado para los otros experimentos de los motores se le modificó la salida mediante un controlador PID. El controlador PID fue alimentado mediante la velocidad medida por el encoder en el eje del motor. Todo el código fue escrito en C++ y la salida grabada por el puerto serial.Es importante destacar que el controlador PID generaba una nueva salida cada 50 ms.

En esta prueba también se trabajó con los motores enganchados a las ruedas y estas sin rozamiento externo. Es decir los motores se encontraban en vacío.

7.2.3. Procesamiento de los Datos

Se probaron tres opciones de tuneo para encontrar el que lograba un mejor andar. Por un lado se intentó obtener un tiempo de respuesta de 0.5 segundos, por otro un tiempo de respuesta de 0.1 segundo y por último uno de 0.05 segundos .

Para la configuración A determinada por el lugar de las raíces, con tiempo de respuesta 0.5, con las constantes del controlador $k_p = 0{,}222$, $k_i = 6{,}35$,$k_d = 0$ se obtuvo la respuesta presentada en la Figura 7.2.1. La configuración A mostró un tiempo de subida de 0,37 segundos en los experimentos en vacío.

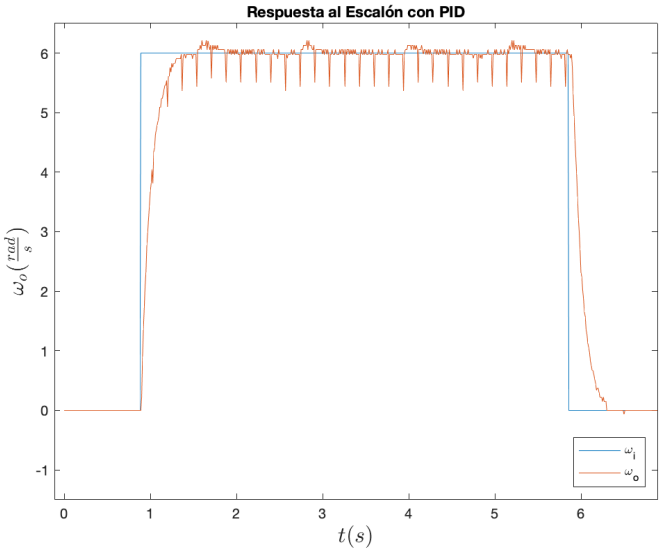


Figura 7.2.1: Respuesta al Escalón con PID configuración A

Para la configuración B que se impuso un tiempo de respuesta de 0.1 segundos para calibrar el controlador PID se obtuvo que se deben configurar los valor $k_p = 1.068$, $k_i = 17{,}225$ y $k_d = 0$. Para estos valores de calibración se obtuvieron la siguiente respuesta al escalón del sistema controlado:

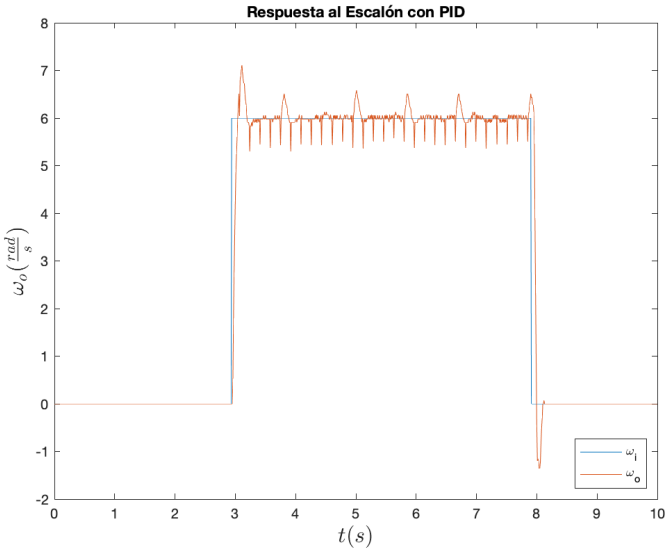


Figura 7.2.2: Respuesta al Escalón con PID configuración B

En promedio el tiempo de respuesta con la configuración correspondiente a la Figura 7.2.2 es de 0,0937 segundos.

Por último la configuración a la que se acotó a un tiempo de respuesta de 0.05 segundos se obtuvo los siguientes valores para su controlador: $k_p = 2{,}092$, $k_i = 29{,}934$,$k_d = 0{,}0$ denominados configuración C.

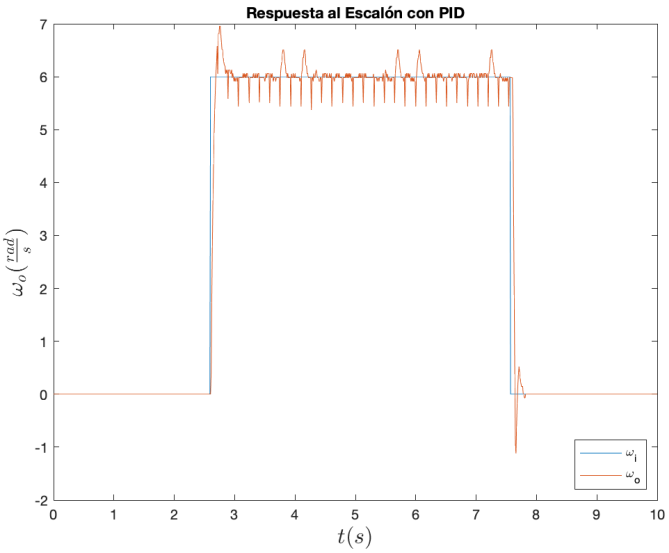


Figura 7.2.3: Respuesta al Escalón PID configuración C

En el experimento expuesto en la Figura 7.2.3 se esperaba ver una disminución en el tiempo de respuesta con respecto a la configuración A y B, y se obtuvo un tiempo promedio de 0,0586 segundos.

Todos los casos anulan el error en régimen presente en el sistema en lazo abierto que es el principal objetivo. Esta característica es lo que se buscó principalmente pero debido al sobretiro presente en las configuraciones B y C, que llevaban a sobresaltos en el andar del robot, se decidió utilizar la configuración A.

7.3. Prueba de deslizamiento de ruedas

7.3.1. Motivación

Se diseñó y realizó este experimento para ver sí, dadas las características constructivas del robot, se logra fijar la velocidad del carro en cada momento sin que se produzcan deslizamientos en ninguna de las ruedas.

7.3.2. Implementación

Para medir el deslizamiento se configuraron dos movimientos, uno hacia adelante y otro hacia atrás. El experimento consistió de 5 pulsos de movimiento hacia adelante durante 2 segundos cada uno, deteniendo el robot 1 segundo entre pulsos. Una vez realizados los 5 pulsos de movimiento hacia adelante, se realizó un

solo movimiento hacia atrás de 10 segundos. Este tipo de movimiento es útil para determinar si en los momentos de aceleración súbita el robot patina. Para poder afirmar que no hubo deslizamiento de las ruedas la posición final del robot debe ser la misma que la posición de inicio.

Se realizó el experimento sin control activo por PID, dado que así se presenta el peor caso. Ya que el controlador PID enlentece la respuesta asegurando una menor aceleración y previniendo el deslizamiento.

Se tomaron como referencias medidas en el suelo, tomadas en cada instante que el robot se detiene entre pulsos, además de la posición inicial y la final. También se registraron las cuentas de los encoders en cada tramo para contrastar con las medidas de distancia.

Se realizó el mismo experimento para 5 velocidades distintas entre 0 y 255: 50, 100, 150, 200 y 250.

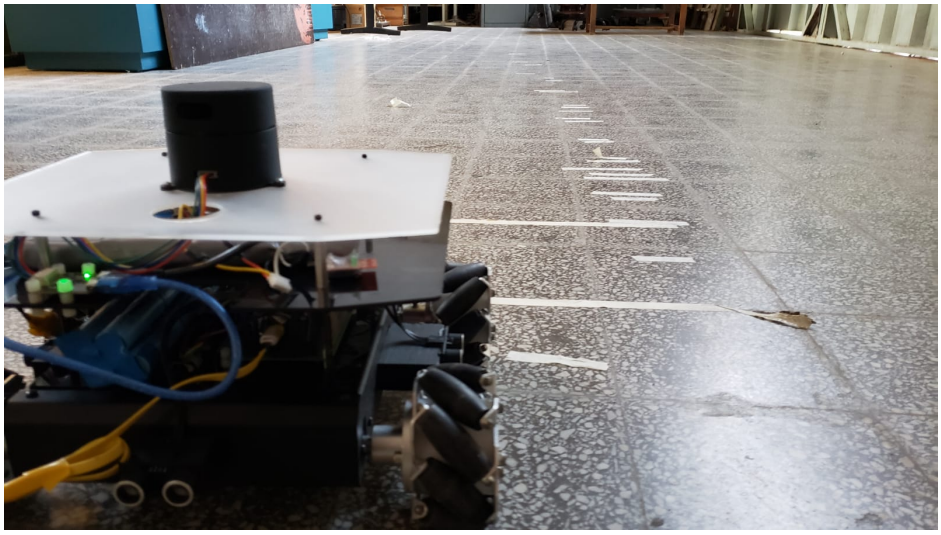


Figura 7.3.1: Método de testeo de deslizamiento

7.3.3. Resultados

En primer lugar, en la Tabla 7.1 se exponen los resultados completos de las cuentas de los encoders para el experimento. Nótese que luego de los 5 pulsos de cada velocidad hay una fila con esa misma velocidad pero de signo negativo. Esta representa el movimiento hacia atrás que se realiza de manera continua por 10 segundos para retornar a la posición inicial.

Velocidad PWM	**Rueda 1**	**Rueda 2**	**Rueda 3**	**Rueda 4**
50	4100	4058	4019	4079
50	4174	4094	4142	4148
50	4204	4104	4132	4150
50	4254	4045	4149	4028
50	4205	4109	4123	4209
-50	-20883	-20224	-20430	-20549
100	8553	8352	8392	8538
100	8942	8639	8701	8796
100	8916	8662	8717	8866
100	8977	8658	8708	8837
100	8982	8643	8752	8824
-100	-44147	-42515	-42944	-43723
150	12976	12620	12721	12893
150	13586	13142	13227	13466
150	13639	13142	13251	13452
150	13582	13167	13246	13503
150	13592	13146	13228	13502
-150	-67031	-64895	-65287	-66331
200	17355	16704	16815	17106
200	18332	17474	17595	17989
200	18292	17451	17643	18001
200	18324	17461	17643	18011
200	18293	17438	17601	18006
-200	-90038	-86779	-86741	-89159
250	20787	19697	20193	19949
250	22025	20822	21179	21095
250	21962	20783	21192	21360
250	22007	20817	21114	21312
250	22015	20764	21108	21354

Tabla 7.1: Cuentas de encoders de cada para cada pulso de movimiento con su velocidad correspondiente

A partir de estos datos se puede obtener la información del desplazamiento y en particular la de la diferencia de desplazamiento medida por los encoders, ya que cuando las ruedas deslizan los encoders siguen contando.

Calculando la suma de todas las cuentas medidas en los desplazamientos hacia adelante para cada velocidad, se les puede restar las cuentas medidas en el desplazamiento hacia atrás a la misma velocidad. En la Figura 7.3.2 se muestra la gráfica de estos datos.

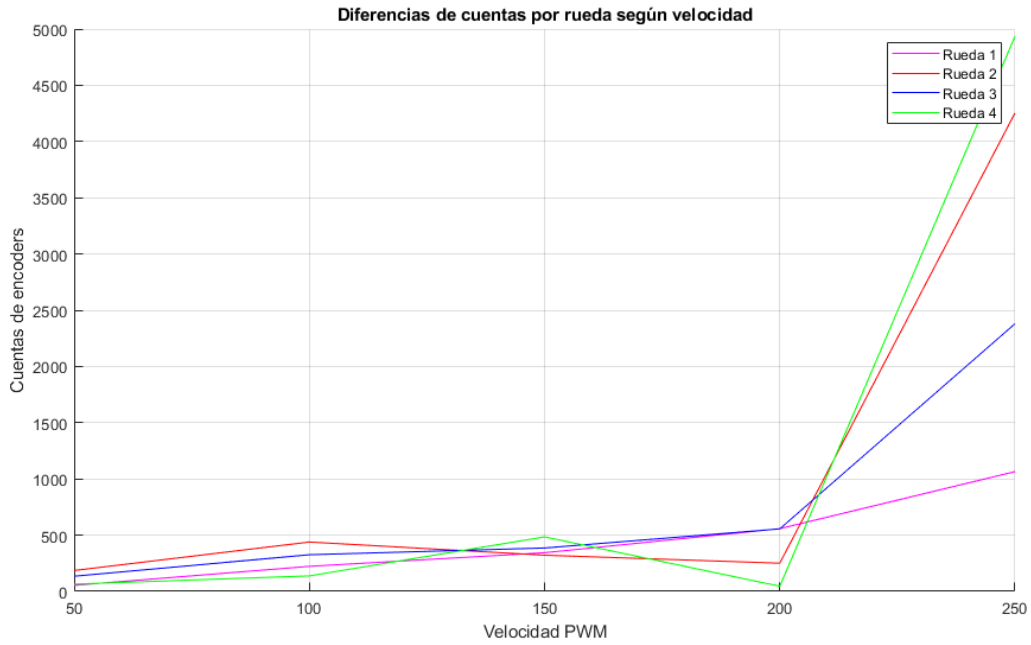


Figura 7.3.2: Diferencia de cuentas de encoder para cada rueda

Se puede apreciar que la diferencia hasta una velocidad de 200 en PWM (78 % de la alcanzable) es de menos de 500 cuentas. Esto es equivalente a 0,06 vueltas de cada rueda, que en milímetros se traduce a 37,4mm. Esta diferencia es despreciable en recorridos de 3,24m como los realizados en este experimento, por lo que la diferencia porcentual es del 1,15 %.

Por otro lado, es notoria la diferencia medida una vez que se alcanza la velocidad de 250 en PWM (98 % de la alcanzable). En la Figura 7.3.2 se indica una diferencia de hasta 5000 cuentas para una de las ruedas, equivalente a 0,6 vueltas de cada rueda, 374mm de desplazamiento que es más que el largo del robot. Esto no solo indica que hay deslizamiento de las ruedas a esta velocidad, sino que también hay ruedas que se están moviendo más rápido que otras. Estos dos puntos refuerzan la necesidad de un controlador PID como el implementado, que regula la velocidad de cada rueda individualmente, además de evitar aceleraciones súbitas y por lo tanto el deslizamiento.

El resultado de este experimento fue tomado en cuenta para configurar las velocidades máximas que alcanza el robot pero la implementación del PID mitiga estos problemas.

7.4. Localización en el mapa

7.4.1. Motivación

El objetivo de este experimento es validar la localización del robot en el mapa. Para ello se compara la medida desde el centro del robot hacia obstáculos entre dos métodos: manual y a través de la interfaz gráfica (Rviz).

7.4.2. Implementación

Primero se mapeó la planta baja del IIE, posteriormente se hicieron diez paradas cada un par de metros donde se tomaron dos medidas desde el robot hacia obstáculos cercanos. Se decidió que estos puntos fuesen esquinas de los obstáculos ya que eran más fáciles de identificar en la interfaz gráfica. Las paradas realizadas se pueden observar en la Figura 7.4.1.

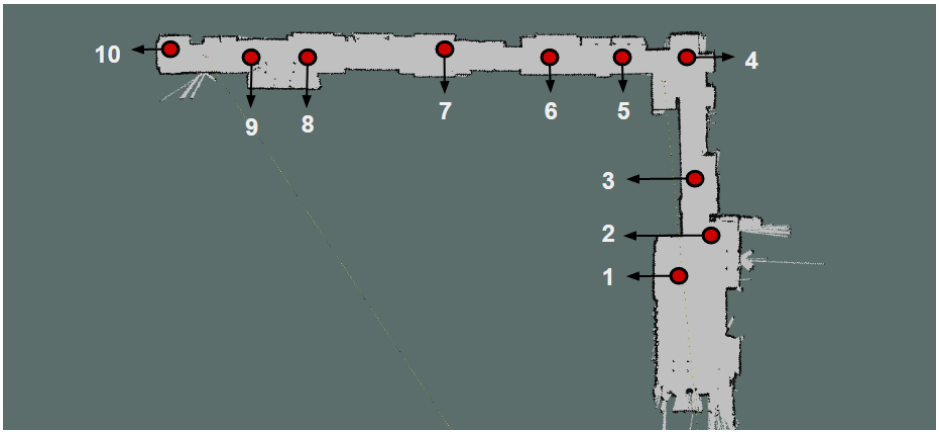


Figura 7.4.1: Mapa de localización de puntos de medida

7.4.3. Procesamiento de Datos

En la Tabla 7.2 se pueden observar para cada medida los resultados obtenidos usando los dos métodos, manual e interfaz gráfica (Rviz). Sobre estos datos se gráfica el error absoluto y relativa en la Figura 7.4.2.

Para medir que tan buena es la localización del robot en el mapa, se pasa a calcular la raíz del error cuadrático medio[1] utilizando la ecuación 7.4.1, obteniendo un valor de 6.28 cm.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{x}_i - x_i)^2} \tag{7.4.1}$$

[1]RMSE por sus siglas en inglés Root Mean Squared Error

Punto	Número de medida	Distancia medida (cm)	
		Rviz	Manual
1	1	237	231
1	2	118	126
2	3	155	160
2	4	141	146
3	5	300	309
3	6	117	113
4	8	197	197
4	9	218	220
5	10	231	237
5	11	138	151
6	12	193	196
6	13	216	225
7	14	84	86
7	15	240	245
8	16	119	120
8	17	184	186
9	18	106	94
9	19	303	305
10	20	177	182
10	21	113	107

Tabla 7.2: Medidas de localización del robot en el mapa

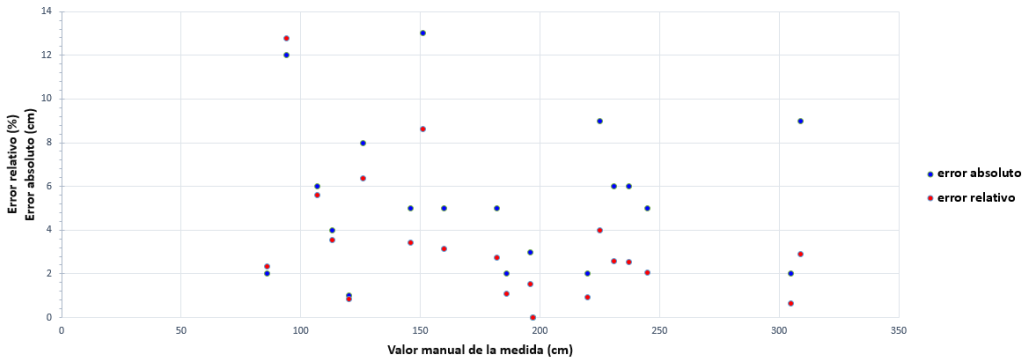


Figura 7.4.2: Gráfica del error absoluto y relativo para cada medida

7.5. Comparación de mapas

7.5.1. Motivación

La principal funcionalidad del robot construido es generar mapas de los ambientes que recorre, por lo que para validar y cuantificar la precisión con la que realiza esta tarea se realizó este experimento.

7.5.2. Implementación

Se realizó el mapeo de dos áreas distintas de las que se contaba con planos digitales a priori para compararlos. Una vez realizados los mapas, se definieron varias distancias dentro de los mapas que se midieron tanto en el mapa generado como en el plano de cada área. A partir de la comparación de las medidas se derivaron los resultados de la precisión de los mapas generados.

7.5.3. Resultados

7.5.3.1. Primer mapa

El primer mapa realizado fue de la planta baja del Instituto de Ingeniería Eléctrica, un área de aproximadamente 110 metros cuadrados. En la Figura 7.5.1 se muestra en blanco el plano de la planta con el mapa generado por el robot superpuesto en color gris. Se puede observar en esta Figura que el mapa generado por el robot se ajusta bien al plano con algunas salvedades. Las diferencias entre el mapa generado y el plano se tratan en su mayoría de muebles en los pasillos, que no figuran en el plano.

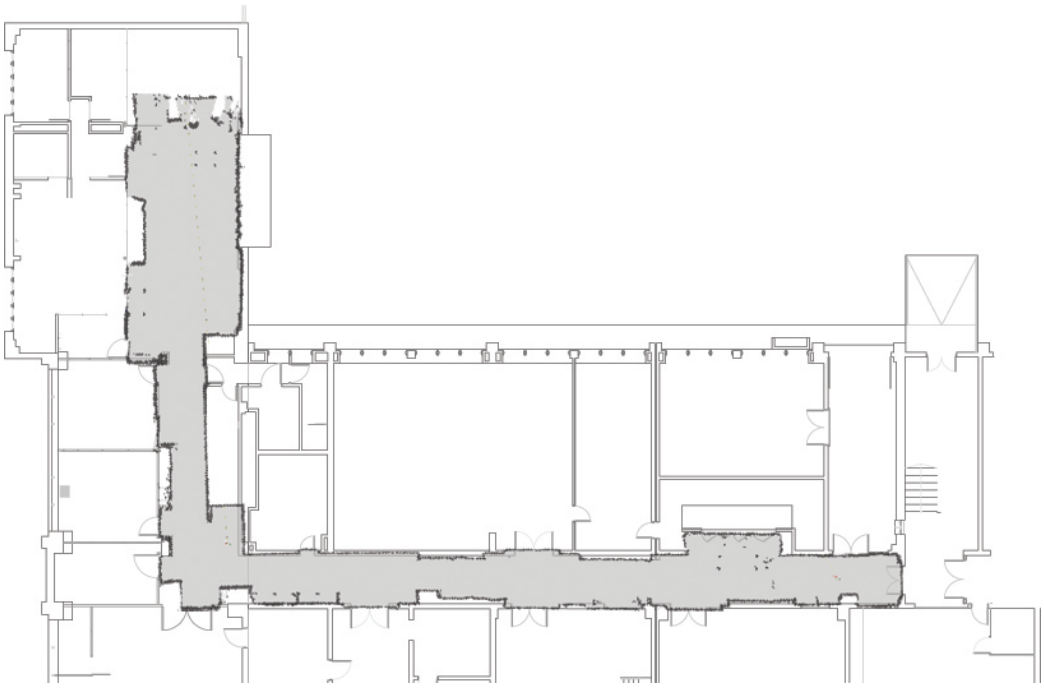


Figura 7.5.1: Primer mapa generado por el robot superpuesto sobre el plano

Para este mapa se tomaron 10 puntos de referencia para tomar medidas y comparar. Dichos puntos de referencia se marcan en el plano de la Figura 7.5.2.

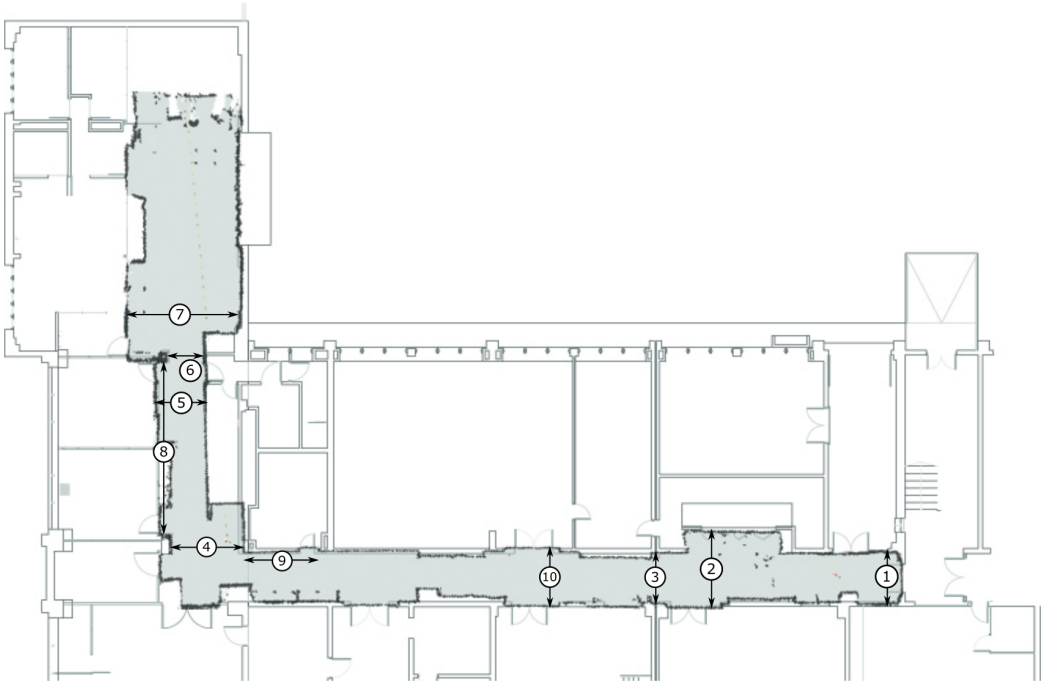


Figura 7.5.2: Distancias de referencia marcadas para comparación

A partir de estas distancias de referencia se tomaron medidas tanto en el plano como en el mapa generado. Los resultados se exponen en la Tabla 7.3.

Medida	**Plano (cm)**	**Rviz (cm)**
1	214	216
2	321	322
3	194	189
4	309	312
5	198	208
6	152	157
7	506	494
8	765	751
9	336	327
10	238	243

Tabla 7.3: Datos comparativos de distancia para el primer mapa

A partir de estos se puede derivar que la raíz del error cuadrático medio para este mapa es de: $RMSE = 7,80cm$. Esta desviación es muy buena para las distancias medidas, donde la mínima es de 150cm y el promedio de las medidas realizadas es de 323cm, significando un error porcentual de 2,7 %.

7.5.4. Segundo mapa

El segundo mapa realizado fue de la planta alta del Instituto de Ingeniería Eléctrica, un área de aproximadamente 80 metros cuadrados. En la Figura 7.5.3 se muestra en blanco el plano de la planta con el mapa generado por el robot superpuesto en color gris. Se puede observar en esta Figura que el mapa generado se asemeja en forma al plano, pero difiere en varias medidas como puede apreciarse en la Tabla 7.4.

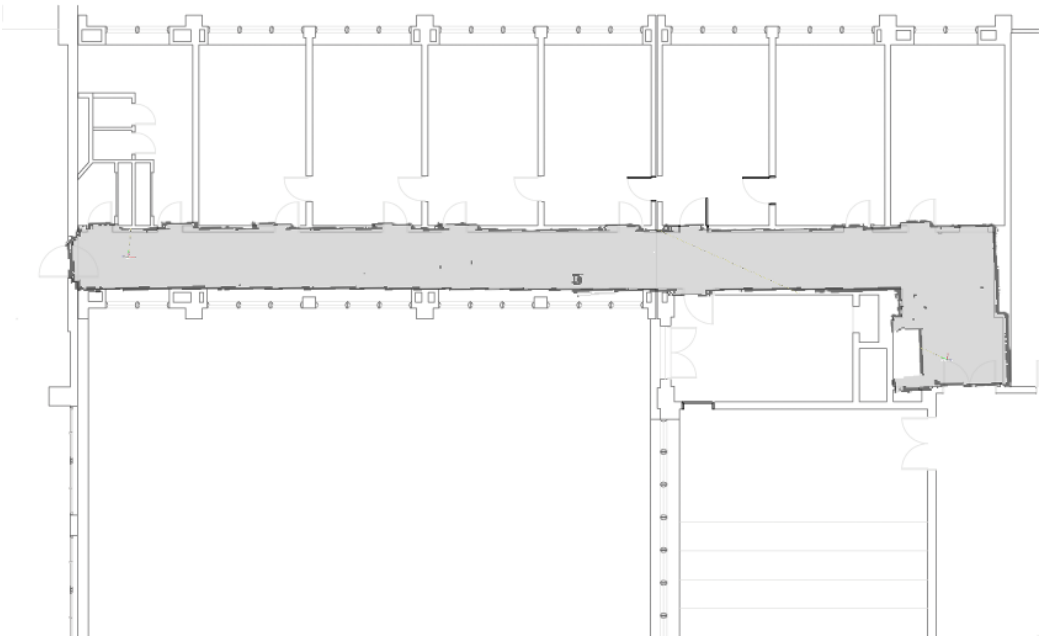


Figura 7.5.3: Segundo mapa generado por el robot superpuesto sobre el plano

Para este mapa se tomaron 8 distancias de referencia para tomar medidas y comparar. Dichas distancias se marcan en el plano de la Figura 7.5.4.

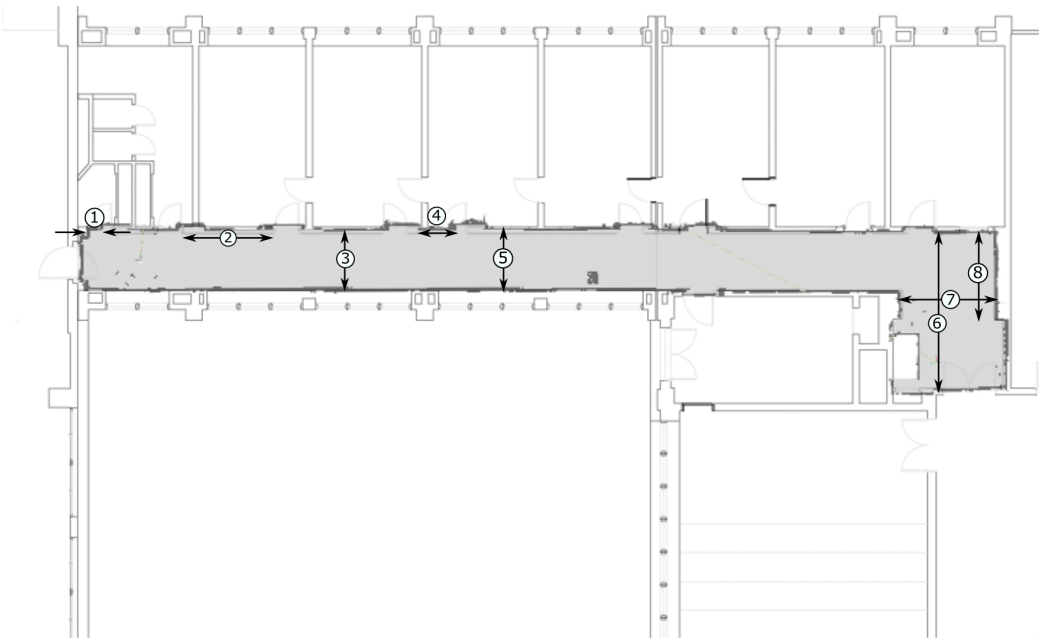


Figura 7.5.4: Distancias de referencia marcadas para comparación

Medida	**Plano (cm)**	**Rviz (cm)**	**Medida real(cm)**
1	28	39	38
2	286	202	201
3	178	176	174
4	106	115	115
5	178	177	176
6	484	483	484
7	288	288	288
8	267	265	266

Tabla 7.4: Datos comparativos de distancia para el segundo mapa

A partir de los datos obtenidos de los tres métodos de medición se puede calcular la raíz del error cuadrático medio como indicador de la desviación en el mapa generado por el robot. Este error se calculó a partir de la diferencia entre las medidas hechas en Rviz y las realizadas a mano en el pasillo, ya que, como se observa en la Tabla 7.4, las distancias obtenidas en el plano difieren en gran medida en relación a las reales. Así se obtuvo $RMSE = 1{,}07cm$, lo que significa un error porcentual de $0.48\,\%$ para el promedio de las medidas realizadas de 217cm.

Además, cabe destacar que el robot con su función de mapeo permitió hallar errores significativos en los planos oficiales de la Facultad.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 8

Conclusiones

8.1. Conclusiones generales

Se logró construir un robot autónomo móvil terrestre e implementar un sistema para el mapeo y navegación de entornos desconocidos para el robot.

En cuanto a la construcción, el robot es robusto frente a las exigencias mecánicas y capaz de cumplir con las condiciones planteadas en los objetivos.

Se considera positiva la elección de los componentes tanto mecánicos como eléctricos. Se destaca en particular la inclusión de tres componentes: las ruedas Mecanum, que permiten el movimiento omnidireccional del robot, el microcontrolador Teensy, que permitió implementar todas las tareas necesarias con fluidez y el LiDAR, que posibilitó la tarea de mapeo. Sin embargo, no fue posible realizar todo el procesamiento del sistema dentro del robot, ya que el Raspberry Pi representa un cuello de botella en términos computacionales. Este fue seleccionado antes de determinar los algoritmos necesarios, por lo que la elección fue guiada por otros trabajos similares. Como consecuencia, se tuvo que delegar la mayor parte del procesamiento a la computadora del usuario.

La elección de usar ROS para este proyecto fue acertada, más allá de la curva de aprendizaje asociada a la misma, ya que permitió sobrellevar las carencias computacionales, marcadas por el Raspberry Pi, al facilitar la distribución de las cargas de procesamiento. Además colaboró para el desarrollo de un proyecto complejo en un tiempo reducido. Un mérito logrado gracias a ROS es la independencia del hardware, ya que el sistema permite intercambiar los componentes utilizados, en caso de querer agregar nuevas funcionalidades o mejorar el desempeño del mismo. Otra ventaja de la decisión del uso de ROS es la modularidad del código, favoreciendo la reutilización y comprensión del mismo.

El modelo teórico del movimiento del robot fue adecuado, ya que permitió lograr una odometría precisa a partir de la velocidad de las ruedas del robot. Es importante mencionar que la confiabilidad del movimiento modelado es muy dependiente de la interacción entre las ruedas y el piso. Por otro lado, el modelo en conjunto con el controlador PI mostraron resultados satisfactorios en cuanto al control del movimiento del robot. Así se logró un movimiento omnidireccional

manejable y consistente.

Aunque no se logró implementar exitosamente el filtro de Kalman extendido diseñado para fusión sensorial, sí se estudiaron y comprendieron los conceptos en profundidad de este tema. Adicionalmente, tomando como base los conocimientos de filtros de Kalman, se pudo elegir e implementar un algoritmo de fusión sensorial ya existente que mejora notoriamente la localización del robot.

La implementación de gmapping como algoritmo de SLAM fue exitosa dado que se lograron obtener mapas muy fieles al entorno real del robot, además de robustez frente a imperfecciones en las medidas de sensores y características cambiantes del ambiente.

Los objetivos de planeación y seguimiento de trayectorias de forma autónoma fueron cumplidos satisfactoriamente, ya que el robot puede planear trayectorias en ambientes con obstáculos de todo tipo sin colisiones hasta alcanzar su meta. Además, se puso especial énfasis en lograr trayectorias que lleven al robot por caminos con un margen de seguridad respecto a los obstáculos.

El módulo de exploración fue desarrollado como un modo de funcionamiento adicional, ya que no se encontraba dentro de los objetivos del proyecto. Dadas las limitaciones de tiempo y que se trata de un módulo adicional, este no pudo ser implementado de modo que funcionara consistentemente de forma correcta. De todos modos, con una mayor cantidad de pruebas y tiempo de desarrollo esta funcionalidad podría ser implementada en el sistema.

La interfaz implementada permite interactuar con el robot de forma sencilla y eficiente, además de facilitar la visualización de los mapas generados por el robot y su ubicación en ellos. Esta presenta un gran aporte a la usabilidad del robot, sobre todo en modo autónomo, cuando es útil conocer las trayectorias y metas del robot.

En resumen, el robot construido y el software implementado en él, cumplen con los objetivos propuestos y resuelven problemas no planteados inicialmente. Esta solución tiene un verdadero potencial para ser usado en aplicaciones relacionadas a la arquitectura, para mapear plantas en tiempos reducidos, y la logística o transporte de cargas.

8.2. Trabajo futuro

Se plantea como primer medida sustituir la Raspberry Pi por una mini PC Intel NUC, migrando todo el procesamiento del sistema a esta última. Una vez realizado el paso anterior todas las tareas se realizarían dentro del robot.

En segunda instancia se plantea agregar una cámara, esta podría usarse para corregir la odometría de las ruedas con la técnica odometría visual o para implementar un tipo distinto de localización y mapeo simultáneo llamado Visual SLAM.

En tercer lugar, se requeriría más tiempo de trabajo centrado en el módulo de exploración autónoma para lograr que funcione de la forma esperada consistentemente.

Finalmente se propone estudiar opciones para mejor la suspensión del robot permitiendo mejorar la confiabilidad en el uso de las ruedas al independizar la misma del tipo de suelo en el que se usan.

Capítulo 9

Anexo

9.1. Manual de usuario

9.1.1. Introducción

El propósito de este documento es introducir los conceptos básicos para el manejo del robot, las posibles configuraciones y los modos de funcionamiento. Este manual sin embargo no profundizará en la programación, ni en como modificar los algoritmos o funcionamiento del robot. Para entender el funcionamiento se recomienda utilizar la documentación oficial de ROS para guiar al lector en las metodologías de desarrollo y estar capacitado para modificar el código fuente.

9.1.2. Instalación y configuración de máquina virtual

Para el funcionamiento del robot se debe contar con una computadora de usuario con los siguientes **requerimientos**:

- 20GB de espacio de disco libre
- Al menos 4GB de RAM que se puedan destinar para la máquina virtual
- Conectividad WiFi 2.4GHz
- Instalación de gestor de máquinas virtuales VMware Workstation[1]

El funcionamiento del robot depende de que la computadora de usuario corra una distribución de Linux con ROS instalado con los packages correspondientes.

Para facilitar esta tarea se provee una máquina virtual basada en Ubuntu 16.04 LTS y con ROS Kinetic instalado, ya configurada con todo el software necesario.

En primer lugar la máquina virtual debe agregarse al entorno de VMWare Workstation:

1. Abrir VMWare Workstation

[1]Puede ser que algún otro software de máquinas virtuales funcione (por ejemplo Parallels para MacOS)

2. Seleccionar ‘Archivo’ → ‘Abrir’
3. Seleccionar el archivo de la máquina virtual y abrirlo
4. Una vez agregada la máquina a VMWare ejecutarla con el botón en el panel de la izquierda

Al ejecutarla debería abrirse la máquina de Ubuntu. Se debe loggear en el usuario Ubuntu con la contraseña ‘**ubuntu**’.

Se debe verificar que la máquina virtual esté configurada con el adaptador de red como ‘Bridged’ (‘Menú de la VM’ → ‘Ajustes’ → ‘Adaptador de red’). Esta configuración permite la comunicación directa entre la máquina virtual y el robot, explicada más adelante.

9.1.3. Inicialización

Una vez configurada la PC debe inicializarse el robot.

9.1.3.1. Nivel de carga de baterías

Antes de encenderse el robot debe verificarse que las baterías se encuentren cargadas. En primer lugar se deben conectar los respectivos Buzzers al conector balanceador de las baterías. Si uno de estos prende un LED rojo, ya sea de forma continua o intermitente[2], significa que una de las celdas está por debajo de su voltaje de funcionamiento. En este caso se debe cargar la batería con la celda baja. En la Figura 9.1.1 se muestra como deben conectarse los Buzzers para ambas baterías.

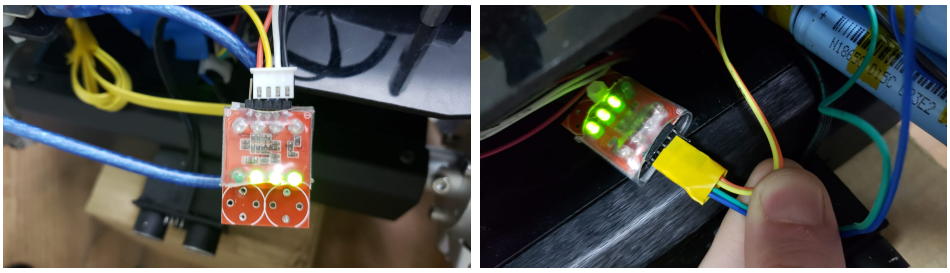


Figura 9.1.1: Buzzers del Sistema

Si es necesario cargar alguno de los paquetes de baterias se deben conectar a un cargador de baterías Li-Ion/Li-Po, con las respectivas conexiones para el balanceo de las celdas. En la Figura 9.1.2 se muestra como conectarlos. Para el uso del cargador referirse a la documentación específica del mismo. En caso de tener

[2]Con las baterías recién cargadas puede ser que los buzzers marquen batería baja, ignorar esta advertencia. En cualquier caso verificar con el voltímetro cada celda

opciones diferentes de carga, se recomienda cargar la batería de inteligencia a 4A y la de Motores a 6A.

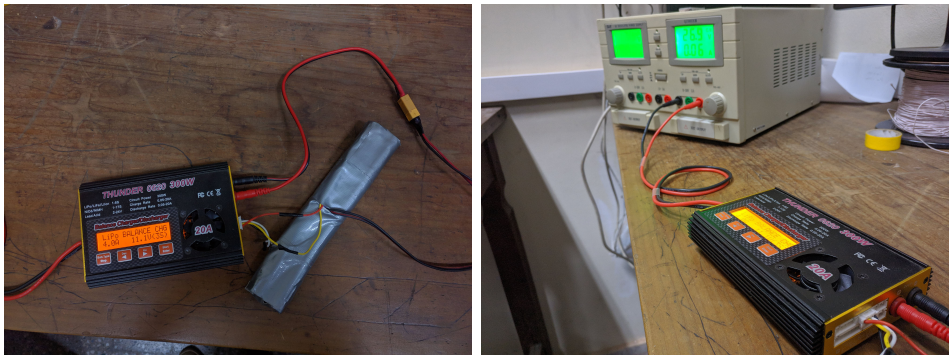


Figura 9.1.2: Conexiones de carga de baterías

9.1.3.2. Creación de WiFi AP

El Raspberry Pi viene con todo lo que necesita ya precargado, solamente requiere que se inicie activando su WiFi access point, al que se conectará la PC para comunicarse. En caso de que no aparezca la red 'ubiquityRobotD9C2' se recomienda, antes de conectar las baterías, conectar un monitor, mouse y teclado al Raspberry Pi para configurarlo.

Los pasos para crear el WiFi AP son:

1. Hacer click derecho en el símbolo de WiFi en la esquina inferior derecha de la pantalla.
2. Seleccionar 'Edit connections'
3. Eliminar todas las redes en la lista salvo 'ubiquityRobotD9C2' o 'WifiAP'
4. Reiniciar el Raspberry Pi

La red '**ubiquityRobotD9C2**' debería aparecer como disponible para conectarse con la PC una vez reiniciado el Raspberry. La contraseña es '**robotseverywhere**'.

9.1.3.3. Conexión y encendido

Una vez cargadas ambas baterías, se deben conectar al robot para alimentar tanto a los motores como al Raspberry Pi, LiDAR y Teensy.

Al conectar la batería de la inteligencia el LiDAR debería encenderse, girar por un segundo y luego detenerse de nuevo. Si gira más lento de lo normal, haciendo un ruido inusual o no gira, se debe verificar el voltaje de la batería y la salida del regulador de tensión. Este último puede medirse fácilmente en los pines de VCC y GND de los sensores de ultrasonido. Si fuera menor a 5,3V, se debe ajustar el

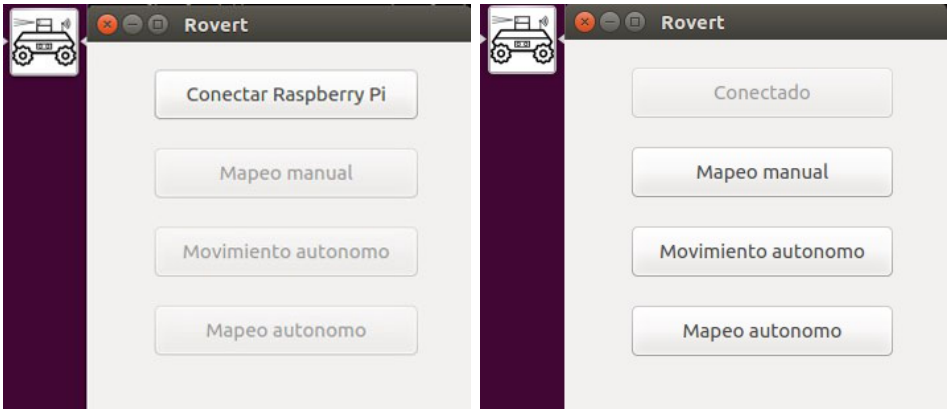
potenciómetro del regulador de tensión hasta superar este voltaje pero sin llevarlo por encima de 5,5V.

Finalmente, la PC debe conectarse a la red WiFi del robot para terminar la configuración. Para este punto recordamos verificar que la máquina virtual esté configurada con el adaptador de red como 'Bridged' ('Menú de la VM' → 'Ajustes' → 'Adaptador de red').

9.1.4. Interfaz

Para comenzar a usar el robot se debe ejecutar el programa '**Rovert**' en el Escritorio de la PC. Este lanzará la selección de funcionamiento.

9.1.5. Selección de modo de funcionamiento



(a) Interfaz de conexión con el robot (b) Interfaz de selección de modo

Figura 9.1.3: Interfaz de usuario: menú principal

Primero se debe presionar el botón 'Conectar Raspberry Pi' asegurándose que la conexión de red sea correcta entre la maquina virtual y el robot (ver Figura 9.1.3a). En caso de falla se mostrará una advertencia.

Una vez que se logra la conexión el botón queda deshabilitado y muestra el texto 'Conectado', habilitando el resto de los botones. Estos botones permiten iniciar un modo de funcionamiento que se pasan a describir a continuación (ver Figura 9.1.3b).

Cuando se selecciona un modo de funcionamiento se abre una ventana adicional en segundo plano que permite cambiar de modo o guardar el mapa. Al seleccionar 'Guardar mapa' se debe elegir un nombre de archivo y un directorio para guardarlo. Allí se crean los archivos .pgm y .yaml, ambos con el nombre elegido, que conforman el mapa. Seleccionar 'Cambiar modo' hace que se cierre esta ventana secundaria y Rviz también, volviendo a la ventana principal de la interfaz.

9.1.5.1. Abrir mapas guardados

Para abrir un mapa guardado en Rviz se deben abrir 3 terminales separadas:

- En la primera se debe ingresar el comando 'roscore', que lanza el ROS Master.
- En la segunda se debe ingresar el siguiente comando 'rosrun map_server map_server [ruta al archivo .yaml del mapa]'. Este comando corre el package 'map_server' que permite publicar mapas guardados como archivos en el topic '/map'.
- En la última terminal se debe ejecutar 'rosrun rviz rviz'. Al abrirse Rviz debería visualizarse el mapa. De no ser así se puede seleccionar para agregar una representación de mapa en el panel izquierdo y se debe ingresar '/map' como topic del mismo.

9.1.6. Descripción de modos de funcionamiento

9.1.6.1. Mapeo manual

En este modo, se puede manejar el robot por comandos desde la PC. A medida que este recorre el ambiente, se irá construyendo el mapa. Se puede observar este proceso en la interfaz de Rviz. Los comandos de velocidad pueden ser dados directamente desde la interfaz con el mouse, o a través de una terminal con el teclado usando el package teleop_twist_keyboard. Para poder ejecutar comandos de ROS en cualquier terminal nueva se debe antes escribir la línea: 'export ROS_IP=[ip de la máquina virtual]'.

Advertencia: el LiDAR no funciona correctamente cuando la luz solar lo ilumina directamente sobre el sensor o bien si ilumina con mucha intensidad una superficie que se desea mapear.

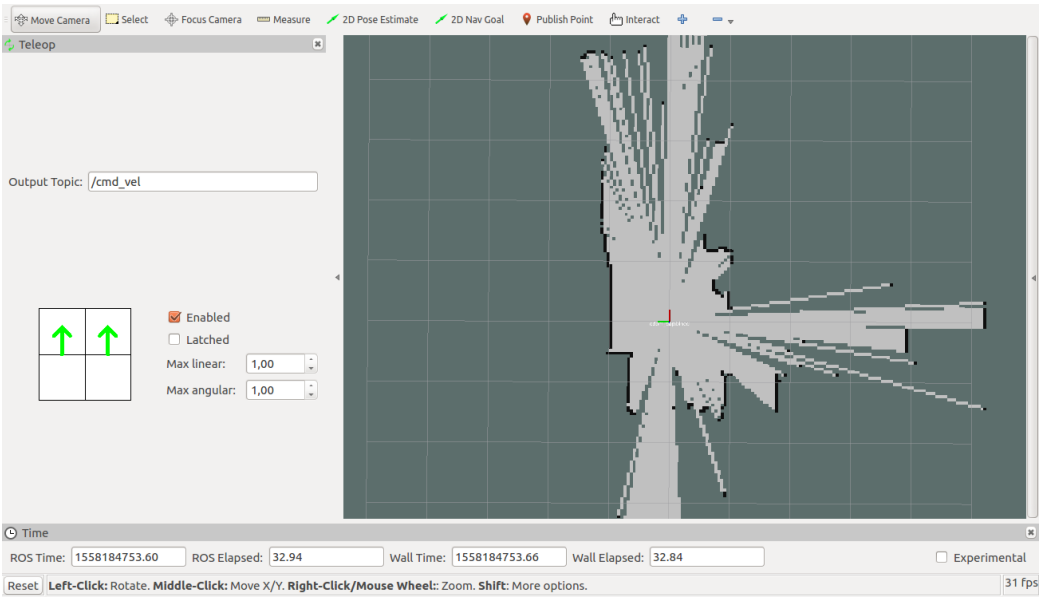


Figura 9.1.4: Rviz en Modo Manual

Los comandos de velocidad se dan a partir de la ventana ubicada en el panel izquierdo. Para activar la herramienta tildar la opción 'Enabled' y ajustar las velocidades máximas (se recomienda 0,4 en ambos casos). Luego, haciendo click en el panel y arrastrando con el mouse se indica hacia donde se quiere mover el robot y dos flechas verdes en la ventana muestran el comando actual (se pueden apreciar en la Figura 9.1.4). Esta herramienta no permite el manejo omindireccional del carro.

9.1.6.2. Movimiento autónomo

En este modo al igual que el anterior se abre el Rviz que permite visualizar la posición actual del robot y el mapa generado hasta el momento. A diferencia del modo anterior, para mover al robot se debe configurar una posición de destino. Para hacer esto primero se debe clickear en la barra superior '2D Nav Goal' y luego indicar en el mapa haciendo click donde se desea enviar el robot y sin soltar apuntar para ajustar la dirección final en la que debe quedar el robot, esto creará una flecha verde que muestra la meta que se está definiendo hasta que se suelte el botón del mouse. La meta quedará marcada con una flecha roja. En la Figura 9.1.5 se muestran todos los elementos descritos.

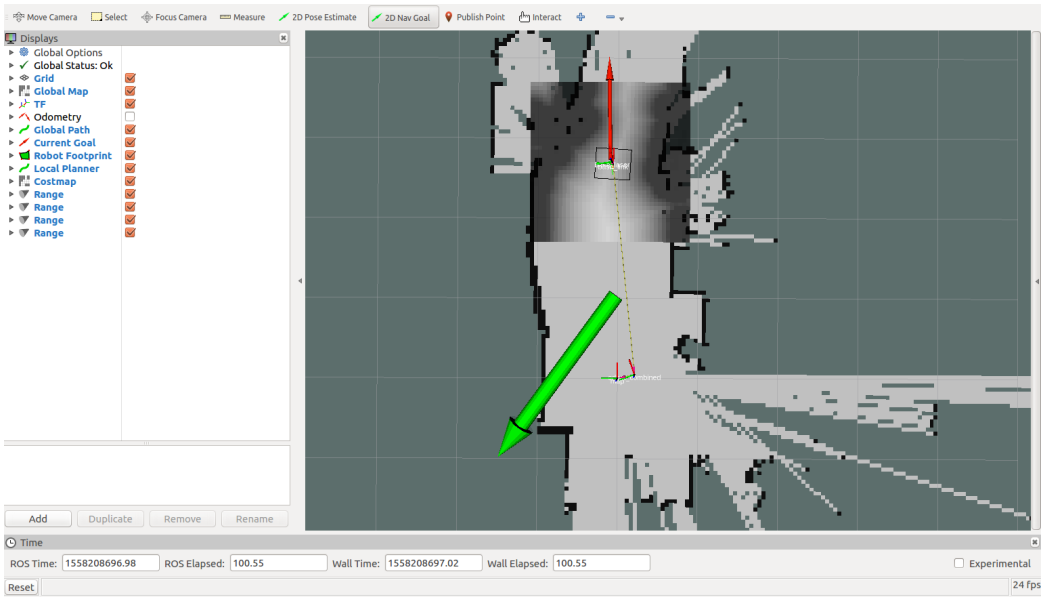


Figura 9.1.5: Selección de meta en modo autónomo

A medida que se mueve el robot se pueden monitorear las trayectorias calculadas, siendo la verde la del planeador global y la roja la del local. El recuadro con sobras al rededor de la representación del robot es al mapa de costos que muestra por donde se debería mover el robot con áreas mas claras. La meta puede ser cambiada en cualquier momento, hasta cuando el robot se encuentra en movimiento rumbo a otra meta. Si el robot no encuentra camino o bien se atasca por las condiciones presentadas será notificado el usuario por terminal. Si el robot se atasca comienza comportamientos de recuperación. El robot no fue diseñado para funcionar con obstáculos dinámicos por lo que se recomienda trabajar en un ambiente libre de personas y obstáculos móviles para funcionar en modo autónomo.

9.1.6.3. Exploración autónoma

Por último, este modo integra el movimiento autónomo con una inteligencia que genera metas en las fronteras del mapa no definidas para completar el mapa actual. No requiere intervención del usuario pero se recomienda acompañar el desarrollo dado que esta funcionalidad no es del todo estable. Al igual que las otras misiones se puede monitorear el proceso mediante Rviz y la terminal.

9.1.7. Configuraciones

En esta Sección se comentarán las configuraciones del módulo 'move_base' pero para tener mayor información recomendamos leer la documentación oficial y consultar 'ROS Navigation Tuning Guide' en los links de interés.

En las siguientes secciones se describen solamente algunos de todos los parámetros que existen en cada archivo. Se eligieron describir solamente los que pueden ser modificados y tienen un efecto real en la navegación. Muchos de los parámetros que no se discuten no deben ser modificados porque las funcionalidades del robot dependen de ello. De todos modos, si se desea hacer cambios más profundos se puede encontrar más información de cada parámetro en la documentación de los packages correspondientes en la Wiki de ROS y los repositorios de GitHub de los módulos.

Los archivos que configuran la navegación autónoma pueden encontrarse en catkin_rovert_ws/src/rovert/move_base_config

9.1.7.1. costmap_common_params

Este archivo contiene los parámetros que rigen para tanto el costmap local como el global.

Los primeros parámetros de interés son: *'inflation_radius'* y *'cost_scaling_factor'*. El primero determina el tamaño de la zona de inflación alrededor del obstáculo y el segundo es que tan rápido decae el costo con la distancia. Se discuten en detalle en la Navigation Tuning Guide, donde se muestran varios ejemplos de los mismos.

Además pueden configurarse parámetros de la range_sensor_layer de como detectan los obstáculos:

- *clear_threshold*(0-1): Celdas con probabilidad menor que esta son marcadas como libres
- *mark_threshold*(0-1): Celdas con probabilidad mayor que esta son marcadas como obstáculos
- *clear_on_max_reading*(True, False): Limpiar mapa a leer máximo rango
- *no_reading_timeout*(float): Si en este tiempo no le llega un msg nuevo imprime warning y marca la layer como 'not current'.
- *topics*(["/topic1", "/topic2", ...]): Topics con la información de sensores de rango que se quieren utilizar.

Si se desea, se puede deshabilitar la range_sensor_layer comentando todas las líneas relacionadas a esta de este y los demás archivos de configuración de move_base.

9.1.7.2. local_costmap_params

Este archivo contiene la configuración del mapa de costos local. Se identifican el nombre del referencial global 'global_frame' y la base del robot 'robot_base_frame'.

Luego se declara la frecuencia de actualización y la de publicación. Una variable que debe tenerse en cuenta es ' transform_tolerance' ya que es la causante de muchos warnings en la terminal. Generalmente se utiliza para comprobar que el resto del sistema esta corriendo de manera correcta y ningún modulo se esta atrasando. En caso de ver el warning correspondiente en terminal revisar el resto del sistema y si no hay errores subir la variable para poder continuar con el uso.

Luego se aclara si el mapa es estático o dinámico, el tamaño de la ventana y la resolución asociada al mapa de costos.

Por último, se definen las capas adicionales o 'plugins' a incluir en el mapa de costos local. Aquí se puede comentar la range_sensor_layer para deshabilitarla. Las demás deben permanecer habilitadas.

9.1.7.3. global_costmap_params

En este archivo se declaran las variables de configuración del mapa de costos global.

Al igual que en el anterior se identifican el referencial global y la base del robot. Además se cuenta con parámetros de frecuencia de actualización interna del costmap y de publicación del mismo. También se define la transform_tolerance para este, que fija la tolerancia de este módulo a los atrasos en el sistema. Esta es fuente de muchos warnings al igual que para el costmap local y se deben manejar de la misma manera.

Finalmente, se definen los 'plugins' para el mapa de costos global. En particular, solo resulta necesario cambiar la range_sensor_layer, que se comenta para deshabilitarla, ya que los demás son necesarios.

9.1.7.4. base_global_planner_params

El archivo de esta Sección se encarga de configurar el planeador global de trayectorias, siendo el que genera la trayectoria principal para el robot.

La variable 'allow_unknown' indica a move_base que es aceptable generar trayectorias por zonas del mapa desconocidas para el robot. De esta manera las trayectorias pueden ser siempre en los caminos más cortos en vez de obligar a pasar por zonas conocidas que pueden resultar en caminos de mayor recorrido.

El algoritmo de seguimiento de trayectorias puede elegirse, siendo las opciones Dijkstra y A*. La variable encargada de diferenciar el algoritmo a utilizar es 'use_dijkstra' que al ponerla en True activa Dijkstra y de lo contrario utiliza A*.

Por otro lado como el mapa esta cuantizado en celdas, se puede seleccionar seguir la trayectoria por los bordes de las mismas o bien se utiliza un método de gradiente descendente para generar un camino entre celdas. La variable encargada de modificar este comportamiento es 'use_grid_path'.

Luego 'visualize_potential' y 'publish_potential' deciden si se expone el potencial para Rviz y si debe ser publicado respectivamente.

La variable 'cost_factor' funciona de igual manera que el mencionado en 'costmap_common_params'.

Finalmente se definen los valores neutros y letales para el mapa de costos. Es decir, para decidir que grillas del mapa son o no accesibles se chequea el costo de cada celda. Si el costo de la celda es menor al neutral_cost, entonces la celda es considerada como segura, si esta entre el neutral y el letal entonces es preferente no pasar esa zona y por último si está por encima del costo letal es mandatorio eludir ese lugar.

9.1.7.5. base_local_planner_params_v2

El archivo tiene todos los comentarios necesarios para poder entender el efecto de cada variable y el valor por defecto.

9.1.8. Links de interés

- Descarga de imagen de Lubuntu + ROS Kinetic para Raspberry Pi: `https://downloads.ubiquityrobotics.com/pi.html`
- Tutoriales Fundamentales de ROS: `http://wiki.ros.org/ROS/Tutorials`
- Referencia de uso teleop_twist_keyboard: `http://wiki.ros.org/teleop_twist_keyboard`
- Tutoriales de uso de rviz: `http://wiki.ros.org/rviz/Tutorials`
- Documentación de YDLIDAR: `http://ydlidar.com/download`
- Doumentación de gmapping: `http://wiki.ros.org/gmapping`
- ROS Navigation Tuning Guide: `http://kaiyuzheng.me/documents/navguide.pdf`
- ROS Navigation Tutorial: `http://wiki.ros.org/navigation/Tutorials/Navigation%20Tuning%20Guide`
- Referencia de uso de map_server: `http://wiki.ros.org/map_server`

9.2. Guía del programador

En esta Sección se incluyen los *launchfiles*, los archivos que se utilizan para lanzar todos los nodos de ROS del robot. Estos enuncian el nodo a lanzar y los parámetros u opciones con los que se debe configurar dicho nodo al ser lanzado. Por más información sobre los launchfiles y el comando 'roslaunch', utilizado para

ejecutar estos archivos, se recomienda leer la página de referencia de uso: `http://wiki.ros.org/roslaunch`.

Los archivos presentados en las siguientes secciones contienen código que solamente será comprensible para un lector familiarizado con el funcionamiento de ROS y los launchfiles. Se recomienda también leer las páginas de cada uno de los nodos que se hablará a continuación para más información sobre las opciones de cada uno.

9.2.1. rpi_localized_config.launch

Este primer archivo es el único que se encuentra en el Raspberry Pi y lanza nodos que corren esta plataforma. Aquí se lanza el nodo 'serial_node' para la comunicación con el Teensy, el nodo 'ydlidar' para el manejo del LiDAR, el nodo 'robot_pose_ekf' para la localización a través de un EKF y por último se crean cuatro nodos del tipo 'tf static transforms'. Estos nodos realizan transformaciones de tf que, a diferencia de otras en el sistema como '/odom_combined→/base_link', no cambian con el tiempo porque transforman entre cada uno de los ultrasonidos y el frame solidario al robot 'base_link'.

```
1  <launch>
2
3          <node pkg="rosserial_python" type="serial_node.py" name="
               ↪ serial_node">
4              <param name="port" value="/dev/ttyACM0"/>
5              <param name="baud" value="230400"/>
6          </node>
7
8          <node name="ydlidar_node" pkg="ydlidar" type="ydlidar_node"
               ↪ output="screen" respawn="false">
9              <param name="port" type="string" value="/dev/ttyUSB0"/>
10             <param name="baudrate" type="int" value="230400"/>
11             <param name="frame_id" type="string" value="base_laser
                   ↪ "/>
12             <!--param name="angle_fixed" type="bool" value="true"/
                   ↪ -->
13             <param name="low_exposure" type="bool" value="false"/>
14             <!--param name="heartbeat" type="bool" value="false"/
                   ↪ -->
15             <param name="resolution_fixed" type="bool" value="true
                   ↪ "/>
16             <param name="auto_reconnect" type="bool" value="true"/
                   ↪ >
17             <param name="reversion" type="bool" value="false"/>
18             <param name="angle_min" type="double" value="-180" />
19             <param name="angle_max" type="double" value="180" />
20             <param name="range_min" type="double" value="0.08" />
```

```
21              <param name="range_max" type="double" value="16.0" />
22              <param name="ignore_array" type="string" value="" />
23              <param name="samp_rate" type="int" value="9"/>
24              <param name="frequency" type="double" value="7"/>
25
26      </node>
27
28      <!-- Definicion de transformaciones de coordenadas para
            ↪ ultrasonidos -->
29
30      <!--node pkg="tf" type="static_transform_publisher" name="
            ↪ tf_name" args="x y z yaw pitch roll frame_id
            ↪ child_frame_id period (ms) -->
31
32    <node pkg="tf" type="static_transform_publisher" name="tf_us_adel
          ↪ " args="0.18 0.0  0.0  0.0  0.0  0.0 /base_link /us_adel 5
          ↪ " />
33    <node pkg="tf" type="static_transform_publisher" name="tf_us_izq"
          ↪  args="0.0  0.21  0.0 1.5708  0.0  0.0 /base_link /us_izq
          ↪ 5" />
34    <node pkg="tf" type="static_transform_publisher" name="tf_us_der"
          ↪  args="0.0 -0.21  0.0 -1.5708  0.0  0.0 /base_link /us_der
          ↪  5" />
35    <node pkg="tf" type="static_transform_publisher" name="tf_us_atr"
          ↪  args="-0.18  0.0  0.0 3.1416  0.0  0.0 /base_link /us_atr
          ↪  5" />
36
37
38    <!-- Localizacion - robot_pose_ekf -->
39
40    <node pkg="robot_pose_ekf" type="robot_pose_ekf" name="
          ↪ robot_pose_ekf">
41          <param name="output_frame" value="odom_combined"/>
42          <param name="base_footprint_frame" value="base_link"/>
43          <param name="freq" value="30.0"/>
44          <param name="sensor_timeout" value="1.0"/>
45          <param name="odom_used" value="true"/>
46          <param name="imu_used" value="true"/>
47          <param name="vo_used" value="false"/>
48          <param name="debug" value="true"/>
49          <param name="self_diagnose" value="false"/>
50      </node>
51
52  </launch>
```

9.2.2. manual_mapping.launch

Este archivo se lanza cuando se selecciona el modo de 'Mapeo manual' en la interfaz de usuario. Este genera los nodos de 'gmapping' para hacer SLAM y de la herramienta 'rviz' para visualizar el mapa y mandar comandos de velocidad al robot. El visualizador rviz es lanzado con una configuración dada por el archivo 'rviz_teleop_plugin.rviz' que puede encontrarse dentro del package 'rovert', donde se encuentran todas las demás configuraciones.

```
1  <launch>
2
3      <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping"
           ↪ output="screen">
4          <param name="base_frame" value="base_link"/>
5          <!-- param name="odom_frame" value="odom"/ SI NO SE USA
               ↪ ROBOT_POSE_EKF -->
6          <param name="odom_frame" value="odom_combined"/>
7          <param name="map_frame" value="map"/>
8          <param name="map_update_interval" value="3.0"/>   <!-- En
               ↪ segundos -->
9          <param name="transform_publish_period" value="0.02"/> <!--
               ↪ Tiempo entre publicaciones de transforms -->
10         <param name="throttle_scans" value="1"/>     <!-- Procesa 1
               ↪ escaneo de cada "throttle_scans" -->
11     </node>
12
13     <node type="rviz" name="rviz" pkg="rviz" args="-d $(find rovert)/
           ↪ Rviz/rviz_teleop_plugin.rviz"/>
14
15 </launch>
```

9.2.3. autonomous_movement.launch

Este archivo se lanza al seleccionar el modo de funcionamiento 'Movimiento autónomo' y además de gmapping y rviz lanza el package 'move_base'. Este es el que concentra otros packages del navigation stack de ROS y permite la navegación autónoma. Las opciones de este package son muy amplias y se describen las más significativas en el Manual de Usuario, Sección de Configuraciones.

```
1  <launch>
2
3    <master auto="start"/>
4
5    <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping"
         ↪ output="screen">
6      <param name="base_frame" value="base_link"/>
```

```
7       <!-- param name="odom_frame" value="odom"/ SI NO SE USA
            ↪ ROBOT_POSE_EKF -->
8       <param name="odom_frame" value="odom_combined"/>
9       <param name="map_frame" value="map"/>
10      <param name="map_update_interval" value="3.0"/> <!-- En
            ↪ segundos -->
11        <param name="transform_publish_period" value="0.01"/> <!--
            ↪ Tiempo entre publicaciones de transforms -->
12      <param name="throttle_scans" value="1"/> <!-- Procesa 1 escaneo
            ↪  de cada "throttle_scans" -->
13    </node>
14
15        <node pkg="move_base" type="move_base" respawn="false" name="
            ↪ move_base" output="screen">
16              <rosparam file="$(find rovert)/move_base_config/
                    ↪ costmap_common_params.yaml" command="load" ns="
                    ↪ global_costmap" />
17            <rosparam file="$(find rovert)/move_base_config/
                ↪ costmap_common_params.yaml" command="load" ns="
                ↪ local_costmap" />
18            <rosparam file="$(find rovert)/move_base_config/
                ↪ local_costmap_params.yaml" command="load" />
19            <rosparam file="$(find rovert)/move_base_config/
                ↪ global_costmap_params.yaml" command="load" />
20            <rosparam file="$(find rovert)/move_base_config/
                ↪ base_local_planner_params_v2.yaml" command="load" /
                ↪ >
21            <rosparam file="$(find rovert)/move_base_config/
                ↪ base_global_planner_params.yaml" command="load" />
22        </node>
23
24        <node type="rviz" name="rviz" pkg="rviz" args="-d $(find
            ↪ rovert)/Rviz/rviz_fullscreen.rviz"/>
25
26  </launch>
```

9.2.4. exploration.launch

Este último archivo se lanza al seleccionar el modo de funcionamiento 'Mapeo autónomo' y consta de los mismos nodos que el anterior, agregando el package 'explore_lite' que implementa el descubrimiento de fronteras.

```
1   <launch>
2
3     <master auto="start"/>
```

```
4
5   <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping"
        ↪ output="screen">
6       <param name="base_frame" value="base_link"/>
7       <!-- param name="odom_frame" value="odom"/ SI NO SE USA
            ↪ ROBOT_POSE_EKF -->
8       <param name="odom_frame" value="odom_combined"/>
9       <param name="map_frame" value="map"/>
10      <param name="map_update_interval" value="3.0"/>   <!-- En
            ↪ segundos -->
11      <param name="transform_publish_period" value="0.01"/> <!--
            ↪ Tiempo entre publicaciones de transforms -->
12      <param name="throttle_scans" value="1"/>     <!-- Procesa 1
            ↪ escaneo de cada "throttle_scans" -->
13   </node>
14
15      <node pkg="move_base" type="move_base" respawn="false" name="
            ↪ move_base" output="screen">
16          <rosparam file="$(find rovert)/move_base_config/
                ↪ costmap_common_params.yaml" command="load" ns="
                ↪ global_costmap" />
17          <rosparam file="$(find rovert)/move_base_config/
                ↪ costmap_common_params.yaml" command="load" ns="
                ↪ local_costmap" />
18          <rosparam file="$(find rovert)/move_base_config/
                ↪ local_costmap_params.yaml" command="load" />
19          <rosparam file="$(find rovert)/move_base_config/
                ↪ global_costmap_params.yaml" command="load" />
20          <rosparam file="$(find rovert)/move_base_config/
                ↪ base_local_planner_params_v2.yaml" command="load" /
                ↪ >
21          <rosparam file="$(find rovert)/move_base_config/
                ↪ base_global_planner_params.yaml" command="load" />
22      </node>
23
24      <node type="rviz" name="rviz" pkg="rviz" args="-d $(find
            ↪ rovert)/Rviz/rviz_fullscreen.rviz"/>
25
26      <node pkg="explore_lite" type="explore" respawn="false" name=
            ↪ "explore" output="screen">
27          <param name="robot_base_frame" value="base_link"/>
28          <param name="costmap_topic" value="map"/>
29          <param name="costmap_updates_topic" value="map_updates
                ↪ "/>
30          <param name="visualize" value="true"/>
```

```
31              <param name="planner_frequency" value="0.33"/>
32              <param name="progress_timeout" value="30.0"/>
33              <param name="potential_scale" value="3.0"/>
34              <param name="orientation_scale" value="0.0"/>
35              <param name="gain_scale" value="1.0"/>
36              <param name="transform_tolerance" value="0.5"/>
37              <param name="min_frontier_size" value="0.75"/>
38      </node>
39
40
41
42 </launch>
```

9.3. Experimentos adicionales

9.3.1. Localización

9.3.1.1. Motivación

La siguiente experiencia se diseñó para verificar los resultados de añadir el filtro de Kalman extendido (EKF) implementado (descrito en la Sección 6.2.3) a la localización del robot, respecto a utilizar Gmapping con la odometría.

9.3.1.2. Implementación

Se comandó el robot de manera remota, siguiendo una trayectoria en forma de bucle cerrado. Este recorrido se realizó dos veces, con dos métodos distintos para determinar la localización del robot. En la primera se utilizó el sistema completo, es decir EKF junto con Gmapping y odometría, mientras que en la segundo se realizó sin el filtro.

Se marcó la ubicación de inicio del robot y siguiendo una trayectoria cerrada se intentó llevar al robot a la misma posición y la misma pose que la inicial.

9.3.1.3. Resultados

El recorrido realizado fue de 14.7m de largo en ambos casos. Se relevaron los datos de distancia entre la posición inicial y final, medidas a partir de la posición real del robot y de la determinada por el algoritmo de localización usado en cada caso. Los datos se muestran en la Tabla 9.1.

	Diferencia medida	**Diferencia real**
Sin EKF	2cm	6cm
Con EKF	3cm	3cm

Tabla 9.1: Diferencia entre la posición final y la posición inicial

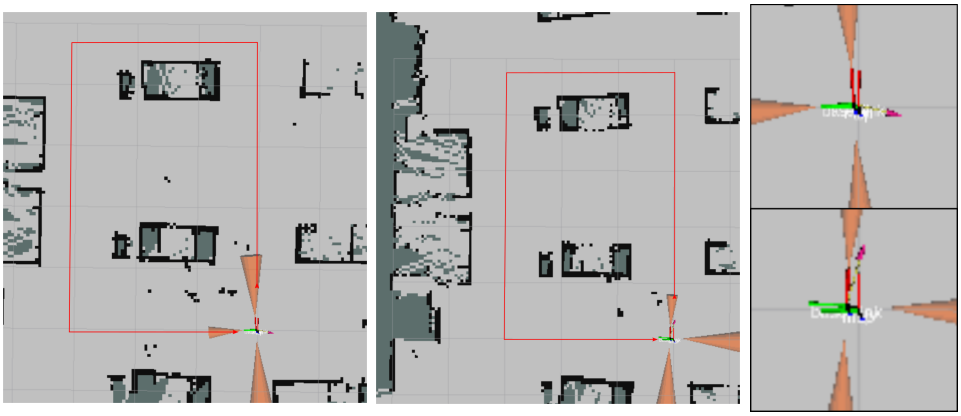


Figura 9.3.1: Resultados de experimento de localización con EKF (izquierda) y sin EKF (centro), y comparación de posiciones (derecha)

9.3.2. Limitaciones de navegación autónoma

9.3.2.1. Abertura mínima para navegación autónoma

Motivación Se ideó este experimento para determinar el ancho mínimo de una abertura, como el marco de una puerta por ejemplo, por el que puede planear una trayectoria el robot y atravesarla de forma consistente.

Implementación Se comenzó por configurar el navigation stack (ver Sección 6.2.5) de modo que permita navegar adecuadamente en los ambientes para los que está destinado el robot. Una vez realizada esta configuración, se posicionaron dos planchas de madera en sentido vertical con una separación de 1m.

Se comandó al robot que planeara una trayectoria al otro lado de las maderas para que pasara entre ellas y se fue disminuyendo el espacio entre las maderas con cada pasada exitosa. Se repitió al menos una vez cada prueba antes de cambiar la distancia entre maderas para asegurarse que la planeación exitosa no fuera accidental. En la misma línea una vez que se encontró la primer distancia para la cual el robot no pasaba, se repitió varias veces antes de terminar el experimento.

Resultados Una separación de 51cm fue la mínima que se pudo alcanzar para que el robot planeara consistentemente una trayectoria y pudiera recorrerla atravesando la abertura. En la Figura 9.3.2 se muestra el robot antes y después de atravesar la abertura.

Figura 9.3.2: Robot navegando a través de la abertura de ancho variable

9.3.2.2. Objetivos fuera del mapa

Motivación Si se está navegando de forma autónoma sin un mapa del ambiente ya disponible, es difícil que las metas definidas por el usuario queden todas dentro del área alcanzable por el robot, por lo que este debe poder funcionar correctamente cuando se le definen metas fuera del mapa conocido. Esta experiencia comprobó esta funcionalidad.

Implementación Habiendo seleccionado 'Mapeo manual' como modo de funcionamiento, se define una meta de navegación en una ubicación que se encuentra fuera del mapa.

Resultados Al fijar la meta fuera del mapa el robot planea trayectorias posibles pasando por secciones donde el mapa no contaba con una frontera definida. Una vez que el robot descubre dichas fronteras no definidas y se determina que no hay una trayectoria posible, se detiene e imprime un mensaje para informar al usuario que no se puede crear una trayectoria. TODO: encerrar al robot y sacar foto con meta fuera

9.3.2.3. Obstáculos dinámicos

Motivación Un ambiente con obstáculos dinámicos no forma parte del alcance de este proyecto, pero de todos modos se ensayó el funcionamiento del robot en el caso de que se cuente con obstáculos móviles. Esto permite determinar la robustez del robot frente a personas caminando por el ambiente, una de las principales dificultades para un robot autónomo.

Implementación Con el robot navegando de forma autónoma, se coloca un objeto en la trayectoria del robot cuando este está por recorrer ese segmento de la trayectoria. Se analizaron los casos en que el obstáculo, una vez interpuesto en el camino del robot, se queda quieto, desaparece, o se mueve dentro del campo visual del robot.

Resultados Cuando el robot detecta el obstáculo colocado en su trayectoria, el mismo reduce su velocidad o se detiene, dependiendo de la distancia a la que se coloque el obstáculo.

En el caso de que el obstáculo permanezca estático, el robot logra planear una trayectoria para esquivarlo, cuando sea posible.

Si el obstáculo desaparece luego de ser detectado, el robot planea su trayectoria nuevamente, pudiendo pasar por la ubicación que antes permanecía inhabilitada. Por otro lado, si el obstáculo se mueve en el campo visual del robot, a velocidades similares a las del robot, la planeación de trayectorias se ve comprometida. El mapa del ambiente comienza a tener errores, dado que el obstáculo puede tomarse como punto de referencia para el mapa, algo que el algoritmo de mapeo no soporta. Esto a su vez hace que la trayectoria planeada sea errónea, dado que esta se basa en el mapa construido.

9.3.3. Prueba de mapeo autónomo

9.3.3.1. Motivación

Se planteó este experimento con el fin de determinar el funcionamiento del módulo de exploración de ambientes. Este tiene como fin mapear un ambiente completo de forma autónoma, eligiendo las metas de cada movimiento de modo de completar un mapa con fronteras definidas.

9.3.3.2. Implementación

Se colocó el robot dentro del Laboratorio de Medidas Eléctricas del IIE, un ambiente cerrado de aproximadamente 70 metros cuadrados de área y que cuenta con obstáculos para el robot. Se procedió a realizar el mapeo autónomo del ambiente.

9.3.3.3. Resultados

Los resultados que se presentarán a continuación corresponden a una sesión de mapeo exitosa de un total de tres que se intentaron. Esto se debe a que esta funcionalidad no está completamente desarrollada y por lo tanto no tiene un funcionamiento estable.

En la Figura 9.3.3 se muestra el ambiente mapeado por el robot.

Figura 9.3.3: Laboratorio mapeado de forma autónoma

El mapa obtenido se puede observar en la Figura 9.3.4, donde se marca también el punto de vista del que fue tomada la imagen 9.3.3. Este mapa es una buena aproximación del ambiente, al igual que se demuestra en la Sección 7.5. También se cronometró el tiempo necesario para completar el mapa, que tomó 4 minutos y 17 segundos.

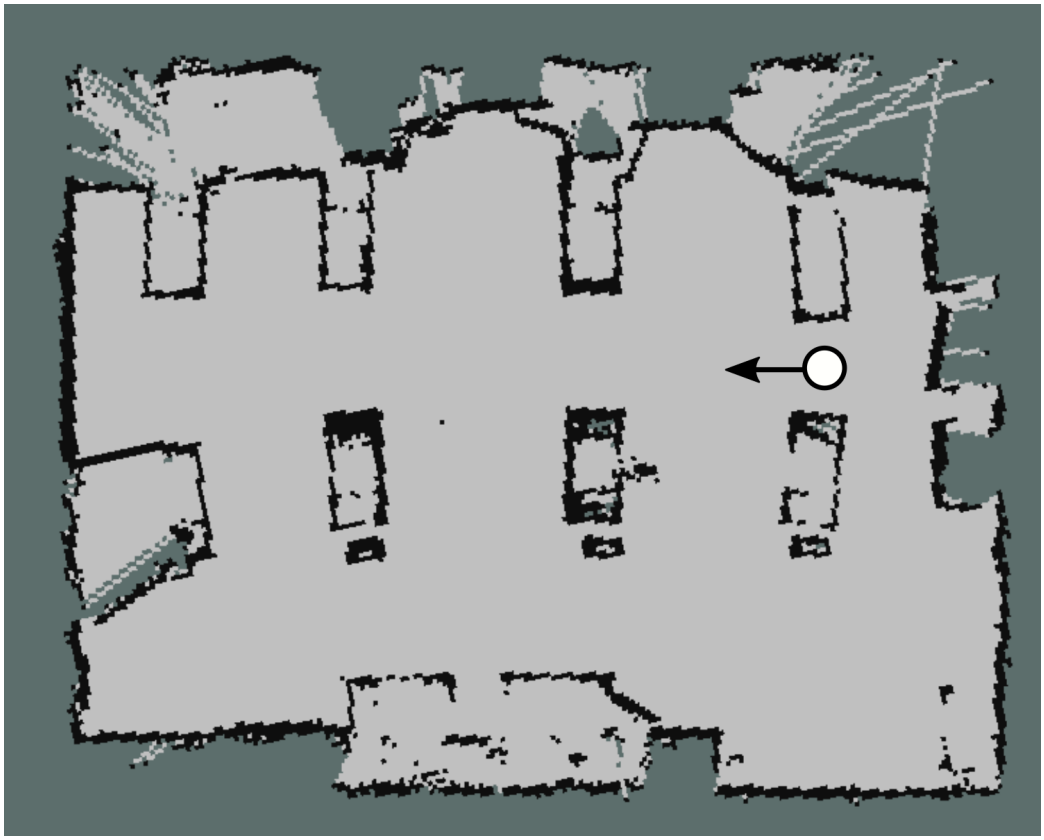


Figura 9.3.4: Mapa generado de forma autónoma

Esta página ha sido intencionalmente dejada en blanco.

Bibliografía

[1] V. Alakshendra y S. S. Chiddarwar. «A robust adaptive control of mecanum wheel mobile robot: simulation and experimental validation». En: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. de 2016, págs. 5606-5611. DOI: 10.1109/IROS.2016.7759824.

[2] Anas W. Alhashimi, Damiano Varagnolo y Thomas Gustafsson. «Statistical Modeling and Calibration of Triangulation Lidars». En: *ICINCO*. 2016.

[3] Brett Beauregard. *Arduino PID Library*. Ver. 1.2.1. Mar. de 2019. URL: https://github.com/br3ttb/Arduino-PID-Library/.

[4] David Blackwell. «Conditional Expectation and Unbiased Sequential Estimation». En: *Ann. Math. Statist.* 18.1 (1947), págs. 105-110. DOI: 10.1214/aoms/1177730497. URL: https://doi.org/10.1214/aoms/1177730497.

[5] Wolfram Burgard y col. *Introduction to Mobile Robotics Robot Motion Planning*. 2011. URL: http://ais.informatik.uni-freiburg.de/teaching/ss11/robotics/slides/18-robot-motion-planning.pdf.

[6] Howie Choset y col. «Principles of Robot Motion: Theory, Algorithms, and Implementations». English. En: MIT Press, mayo de 2005. ISBN: 0262033275.

[7] Cytron. *MDD10A Dual Channel 10A DC Motor Driver*. URL: https://www.robotshop.com/media/files/pdf2/rb-cyt-153_-_mdd10a_users_manual_v2.0_-_2017-06.pdf.

[8] Gregory Dudek y Michael Jenkin. «Computational Principles of Mobile Robotics». En: New York, NY, USA: Cambridge University Press, 2010. ISBN: 0521692121.

[9] Hugh Durrant-Whyte y Tim Bailey. «Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms». En: *IEEE ROBOTICS AND AUTOMATION MAGAZINE* 2 (2006), pág. 2006.

[10] Hugh Durrant-Whyte y Thomas C. Henderson. «Multisensor Data Fusion». En: ene. de 2008, págs. 585-610. DOI: 10.1007/978-3-540-30301-5_26.

[11] D. Fox, W. Burgard y S. Thrun. «The dynamic window approach to collision avoidance». En: *IEEE Robotics Automation Magazine* 4.1 (mar. de 1997), págs. 23-33.

[12] G. Grisetti, C. Stachniss y W. Burgard. «Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters». En: *IEEE Transactions on Robotics* 23.1 (feb. de 2007), págs. 34-46. ISSN: 1552-3098. DOI: 10.1109/TRO.2006.889486.

[13] G. Grisettiyz, C. Stachniss y W. Burgard. «Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling». En: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. Abr. de 2005, págs. 2432-2437. DOI: 10.1109/ROBOT.2005.1570477.

[14] David L. Hall, James Llinas y Martin E. Liggins. «Handbook of Multisensor Data Fusion». En: (sep. de 2008), pág. 870. DOI: 10.1201/9781420038545.

[15] Invensense. *MPU-9250 Product Specification*. URL: http://www.invensense.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf.

[16] Richard Johnsonbaugh. «Matemáticas Discretas». En: 6.ª ed. Vol. 4. PEARSON EDUCACIÓN, 2005. Cap. 8, págs. 347-351.

[17] Anis Koubaa. *Robot Operating System (ROS): The Complete Reference (Volume 1)*. 1st. Studies in Computational Intelligence. Springer Publishing Company, Incorporated, 2016. ISBN: 978-3-319-26052-5.

[18] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991. ISBN: 0792391292.

[19] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006. DOI: 10.1017/CBO9780511546877.

[20] MATLAB. *DC Motor Speed: System Modeling*. http://ctms.engin.umich.edu/CTMS/index.php?example=MotorSpeed&section=SystemModeling. [Online; accessed 20-May-2019]. 2016.

[21] H.B Mitchell. *Multi-sensor Data Fusion. An Introduction*. 1.ª ed. Springer-Verlag Berlin Heidelberg, ene. de 2007, pág. 282. ISBN: 978-3-540-71559-7. DOI: 10.1007/978-3-540-71559-7.

[22] Michael Montemerlo y col. «FastSLAM: a factored solution to the simultaneous localization and mapping problem with unknown data association». Tesis de mtría. Carnegie Mellon University, the Robotics Institute, 2002.

[23] Hans P. Moravec. «Sensor Fusion in Certainty Grids for Mobile Robots». En: *AI Magazine* 9 (1988), págs. 61-74.

[24] Mouser Electronics. *HC-SR04 Ultrasonic Sensor Datasheet*. URL: https://www.mouser.com/ds/2/813/HCSR04-1022824.pdf.

[25] P. Muir y C. Neuman. «Kinematic modeling for feedback control of an omnidirectional wheeled mobile robot». En: *Proceedings. 1987 IEEE International Conference on Robotics and Automation*. Vol. 4. Mar. de 1987, págs. 1772-1778. DOI: 10.1109/ROBOT.1987.1087767.

[26] Kevin Murphy y Stuart Russel. «Rao–Blackwellized particle filtering for dynamic Bayesian networks». En: *Sequential Monte Carlo Methods in Practice* 1.3 (2001), págs. 499-516.

[27] Open Source Robotics Foundation. *navigation/Tutorials/Navigation Tuning Guide - ROS Wiki*. URL: http://wiki.ros.org/navigation/Tutorials/Navigation%5C%20Tuning%5C%20Guide.

[28] Open Source Robotics Foundation. *robot_pose_ekf - ROS Wiki*. URL: http://wiki.ros.org/robot_pose_ekf.

[29] OpenSLAM. *Gmapping*. Ver. 1. Mayo de 2019. URL: https://openslam-org.github.io/gmapping.html.

[30] PJRC. *Teensy 3.5 Development Board*. URL: https://www.pjrc.com/store/teensy35.html.

[31] Pololu. *131:1 Metal Gearmotor 37Dx73L mm with 64 CPR Encoder*. URL: https://www.pololu.com/product/2827.

[32] Python Software Foundation. *PyQt - Python Wiki*. URL: https://wiki.python.org/moin/PyQt.

[33] Calyampudi Radhakrishna Rao. «Information and accuracy obtainable in estimation of statistical parameters». En: *Bull. Calcutta Math. Soc.* 37.3 (1945), págs. 81-91.

[34] Raspberry Pi Foundation. *Raspberry Pi Model 3B+*. URL: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/.

[35] Bruno Siciliano y Oussama Khatib. *Springer Handbook of Robotics*. Springer-Verlag Berlin Heidelberg, 2016. ISBN: 978-3-540-30301-5. URL: https://www.springer.com/us/book/9783540303015.

[36] Departamento de Sistemas y Control. *Prácticas de Laboratorio*. https://eva.fing.edu.uy/mod/resource/view.php?id=29168. [Online; accessed 20-Abril-2019]. 2016.

[37] S. K. Soltakhanov, M. P. Yushkov y S. A. Zegzhda. «Mechanics of Nonholonomic Systems: A New Class of Control Systems». En: Springer Publishing Company, Incorporated, 2009. ISBN: 3540858466.

[38] Sparkfun. *MPU-9250 9 DOF IMU Arduino Library*. Ver. 1.0. Dic. de 2018. URL: https://github.com/sparkfun/SparkFun_MPU-9250_Breakout_Arduino_Library.

[39] Paul Stoffregen. *Encoder Library*. Ver. 1.2. Mar. de 2019. URL: https://github.com/PaulStoffregen/Encoder.

[40] Texas Instruments. *LM2596 SIMPLE SWITCHER Power Converter 150-kHz 3-A Step-Down Voltage Regulator*. URL: http://www.ti.com/lit/ds/symlink/lm2596.pdf.

[41] The Qt Company. *Qt | Cross-platform software development for embedded & desktop*. URL: https://www.qt.io/.

[42] B. Yamauchi. «A frontier-based approach for autonomous exploration». En: *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*. Jul. de 1997, págs. 146-151. DOI: 10.1109/CIRA.1997.613851.

[43] YDLIDAR. *YDLIDAR F4 PRO*. URL: http://ydlidar.com/download.

[44] Kaiyu Zheng. «ROS Navigation Tuning Guide». En: *CoRR* abs/1706.09068 (2017). eprint: 1706.09068. URL: http://arxiv.org/abs/1706.09068.

Índice de Tablas

3.1. Tabla de componentes 27
3.2. Tabla de contenido por piso 28
3.3. Tabla de Pines 31
3.4. Requerimientos mecánicos del robot 33
3.5. Requerimientos de los motores 33
3.6. Principales características de la serie 18650 38

5.1. Configuraciones obtenidas según tiempo de respuesta 69

7.1. Cuentas de encoders de cada para cada pulso de movimiento con su velocidad correspondiente 137
7.2. Medidas de localización del robot en el mapa 140
7.3. Datos comparativos de distancia para el primer mapa 143
7.4. Datos comparativos de distancia para el segundo mapa 145

9.1. Diferencia entre la posición final y la posición inicial 164

Esta página ha sido intencionalmente dejada en blanco.

Índice de Figuras

1.0.1. Ejemplos de Robots Terrestres Autónomos 2
1.2.1. Roomba 980 3
1.2.2. TurtleBot *Waffle Pi* 4
1.2.3. Robot de Amazon 4

2.3.1. Rovert: robot construido para el Proyecto 9
2.4.1. Diagrama dimensional del robot construido 10
2.4.2. Diagrama de conexión entre rueda, adaptador y motor 11
2.4.3. Diagrama de nivel de chasis 12
2.4.4. Diagrama de nivel de chasis 13
2.4.5. Diagrama de nivel de chasis 13
2.5.1. Diagrama de interacción rueda-robot 14
2.5.2. Diagrama de combinaciones de desplazamiento del robot 15
2.6.1. Diagrama eléctrico del robot 16
2.6.2. Diagrama de conexión de componentes 17
2.6.3. Diagrama de conexión de motores 18
2.6.4. Diagrama de conexión de inteligencia 19
2.7.1. Diagrama de interacción de algoritmos y hardware 20
2.8.1. Arquitectura lógica del sistema 23

3.3.1. Diagrama eléctrico del robot 29
3.3.2. Diagrama de conexión de componentes 30
3.4.1. Rueda Mecanum 32
3.4.2. Diagrama del cuerpo libre del robot 34
3.4.3. Motor DC 12 V 34
3.4.4. Diagrama de un encoder en cuadratura 35
3.4.5. Captura de canales del encoder en un osciloscopio 35
3.4.6. Terminales del motor y Encoder 36
3.4.7. Esquemático del puente H 37
3.4.8. Ejemplo de distintos pulsos PWM 37
3.4.9. Controlador de Motores 38
3.4.10. Baterías del robot 39
3.4.11. Regulador de tensión basado en el LM2596 40
3.4.12. *Buzzer* para baterías 40
3.4.13. Principio de funcionamiento del LiDAR 41

3.4.14. LiDAR 42
3.4.15. Diagrama de tiempos del ultrasonido 43
3.4.16. Sensor de Ultrasonido 43
3.4.17. Placa de desarrollo GY 9250 44
3.5.1. Teensy 3.5 44
3.5.2. Raspberry Pi 3 B+ 45

4.2.1. Definición de sistemas de coordenadas del robot 48
4.2.2. Modelo de la cinemática de una rueda 49
4.3.1. Definición de ejes de referencia para ruedas 51
4.3.2. Cinemática de las ruedas mecanum 52
4.4.1. Modelo de cuerpo libre del robot 55
4.4.2. Diagrama de fuerzas en rueda mecanum 56

5.1.1. Respuesta al Escalón Rueda 2 61
5.1.2. Diagrama de Control de cada Rueda 62
5.1.3. Sistema a Modelar 64
5.1.4. Modelo contra Datos relevados 66
5.1.5. Controlador PI: Lugar de las Raíces. 68
5.1.6. Respuesta al Escalón con controlado Diseñado 69
5.1.7. Respuesta al Escalón PID configuración B 70
5.2.1. Localización sin aplicar fusión sensorial[14] 72
5.2.2. Diagrama funcional del filtro de Kalman 77
5.2.3. Posición del modelo vs Posición estimada por el EKF 81
5.2.4. Posición del modelo vs Posición estimada por el EKF 81
5.2.5. Posición del modelo vs Posición estimada por el EKF 82
5.2.6. Error en la estimación del EKF 83
5.3.1. Trayectoria en SLAM 85
5.3.2. Representación de SLAM y la independencia entre landmarks sin la trayectoria 89
5.4.1. Primer ejemplo de espacio de configuración 92
5.4.2. Segundo ejemplo de espacio de configuración 93
5.4.3. Diagrama de arquitectura en dos niveles 95
5.4.4. Ejemplo de grafo ponderado 96
5.4.5. Inicialización del algoritmo 97
5.4.6. Primera iteración 97
5.4.7. Segunda iteración 97
5.4.8. Tercera iteración 97
5.4.9. Venta dinámica 99
5.4.10. Silueta del robot 100
5.4.11. Inflación 101
5.4.12. Proceso de detección de fronteras 103

6.2.1. Ejemplo de mapa generado por el robot 112
6.2.2. Mapa de costos superpuesto con mapa del ambiente 114
6.2.3. Robot con una trayectoria global planeada 115

6.2.4. Robot con una trayectoria global hacia un objetivo en una zona desconocida . . . 115
6.2.5. Planeación de trayectorias local . . . 116
6.2.6. Diagrama de bloques de move_base . . . 117
6.2.7. Máquina de estados del sistema de recuperación . . . 118
6.2.8. Imagen de Rviz con sus menús y la ventana de visualización . . . 120
6.2.9. Definición de metas para navegación a través de Rviz . . . 121
6.2.10. Plugin de comando de velocidad en Rviz . . . 122
6.2.11. Interfaz de usuario: menú principal . . . 123

7.1.1. Velocidad de Salida según la entrada Rueda 1 . . . 126
7.1.2. Velocidad de Salida según la entrada Rueda 2 . . . 127
7.1.3. Velocidad de Salida según la entrada Rueda 3 . . . 127
7.1.4. Velocidad de Salida según la entrada Rueda 4 . . . 128
7.1.5. Velocidad de Salida según la entrada para todas las Ruedas . . . 129
7.1.6. Respuesta al Escalón Rueda 1 . . . 130
7.1.7. Respuesta al Escalón Rueda 2 . . . 131
7.1.8. Respuesta al Escalón Rueda 3 . . . 131
7.1.9. Respuesta al Escalón Rueda 4 . . . 132
7.2.1. Respuesta al Escalón con PID configuración A . . . 134
7.2.2. Respuesta al Escalón con PID configuración B . . . 134
7.2.3. Respuesta al Escalón PID configuración C . . . 135
7.3.1. Método de testeo de deslizamiento . . . 136
7.3.2. Diferencia de cuentas de encoder para cada rueda . . . 138
7.4.1. Mapa de localización de puntos de medida . . . 139
7.4.2. Gráfica del error absoluto y relativo para cada medida . . . 140
7.5.1. Primer mapa generado por el robot superpuesto sobre el plano . . . 142
7.5.2. Distancias de referencia marcadas para comparación . . . 142
7.5.3. Segundo mapa generado por el robot superpuesto sobre el plano . . . 144
7.5.4. Distancias de referencia marcadas para comparación . . . 144

9.1.1. Buzzers del Sistema . . . 150
9.1.2. Conexiones de carga de baterías . . . 151
9.1.3. Interfaz de usuario: menú principal . . . 152
9.1.4. Rviz en Modo Manual . . . 154
9.1.5. Selección de meta en modo autónomo . . . 155
9.3.1. Resultados de experimento de localización con EKF (izquierda) y sin EKF (centro), y comparación de posiciones (derecha) . . . 165
9.3.2. Robot navegando a través de la abertura de ancho variable . . . 166
9.3.3. Laboratorio mapeado de forma autónoma . . . 168
9.3.4. Mapa generado de forma autónoma . . . 169

Esta es la última página.
Compilado el martes 23 julio, 2019.
http://iie.fing.edu.uy/