ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Mecánica y Ciencias de la Producción

Coordinación de movimientos para un robot manipulador móvil holonómico

PROYECTO INTEGRADOR

Previo la obtención del Título de:

Ingeniero en Mecatrónica

Presentado por:

Andrés Alberto Castro Peñaranda

GUAYAQUIL - ECUADOR

Año: 2022

DEDICATORIA

El presente proyecto lo dedico a Dios por guiarme en este camino y brindarme la fuerza y el valor para alcanzar mis metas.

A mis padres, Javier Castro y Ligia Peñaranda, por su amor incondicional y por apoyarme en todo momento.

A mi hermano Felipe Castro, por ser mi confidente y mi amigo.

A mi amada novia Natasha Guzmán, por su amor y compañía en esta etapa de mi vida.

A mis amigos, por sus risas, apoyo y motivación. Gracias a todos ustedes por formar parte de mi vida y ayudarme a llegar hasta aquí.

Andrés Alberto Castro Peñaranda

AGRADECIMIENTOS

Quiero agradecer a Dios por iluminar mi camino y darme la fortaleza para completar esta tesis. Mi gratitud va también a mis queridos padres, Javier Castro y Ligia Peñaranda, por su amor incondicional y apoyo constante en todo momento. Agradezco a los profesores Jan Rosell y Leopold Palomo por su orientación y colaboración en el laboratorio, y al profesor Christian Tutivén por brindarme la oportunidad de realizar este proyecto. Gracias a todos ustedes por su contribución valiosa a mi crecimiento académico y profesional.

Andrés Alberto Castro Peñaranda

DECLARACIÓN EXPRESA

"Los derechos de titularidad y explotación, me corresponde conforme al reglamento de propiedad intelectual de la institución; ***Andrés Alberto Castro Peñaranda*** y doy mi consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

Andrés Alberto Castro
Peñaranda

EVALUADORES

Firmado electrónicamente por:
CARLOS XAVIER
SALDARRIAGA MERO

Carlos Saldarriaga, **Ph.D.**

PROFESOR DE LA MATERIA

Firmado electrónicamente por:
CHRISTIAN JAVIER
TUTIVEN GALVEZ

Christian Tutivén, **Ph.D.**

PROFESOR TUTOR

RESUMEN

Dentro de una universidad politécnica en España se tiene un manipulador móvil bibrazo equipado con manos mecánicas, que se llama MADAR (*Mobile Anthropomorphic Dual-Arm Robot*). Este robot solo cuenta con el sistema de control que permite la ejecución de movimientos para los brazos y las manos. Se desea desarrollar un sistema de control para mover la base del MADAR, obtener el posicionamiento de la plataforma dentro del laboratorio y la navegación autónoma entre diferentes estaciones de trabajo.

Para lograr el objetivo se desarrollaron librerías en el lenguaje de programación C++ para controlar la plataforma del MADAR, integrar los sensores y conectarse con los servomotores a través de una red EtherCAT. Se utilizaron paquetes de ROS para que los nodos puedan funcionar tanto en la simulación como en el robot real. Se requirieron métodos para obtener y localizarse en un mapa, así como para planificar rutas locales y globales, para hacer uso del paquete de navegación de ROS.

Se obtuvo un modelo cinemático, el cual se corroboró con una simulación dinámica del MADAR en gazebo. Se usó el mismo modelo en la plataforma real. Se consiguió que la plataforma del robot pueda desplazarse de forma autónoma hacia puntos de interés, teniendo en cuenta obstáculos y la capacidad de evadirlos de manera dinámica. Concluyendo el desarrollo exitoso del sistema de control de base móvil del robot MADAR, se consigue localización y navegación eficiente.

Palabras Clave: Base Móvil, Robot Holonómico, Navegación, ROS, Localización.

ABSTRACT

Within a polytechnic university in Spain there is a mobile manipulator equipped with mechanical hands, called MADAR (Mobile Anthropomorphic Dual-Arm Robot). This robot has only the control system that allows the execution of movements for the arms and hands. It is desired to develop a control system to move the base of MADAR, to obtain the positioning of the platform inside the laboratory and the autonomous navigation between different workstations.

To achieve the objective, libraries were developed in the C++ programming language to control the MADAR platform, integrate the sensors, and connect with the servomotors through an EtherCAT network. ROS packages were used so that the nodes can function both in the simulation and on the real robot. Methods for obtaining and locating on a map, as well as planning local and global routes, were required to make use of the ROS navigation package.

A kinematic model was obtained, which was corroborated with a dynamic simulation of the MADAR in gazebo. The same model was used on the real platform. It was achieved that the robot platform can move autonomously to points of interest, considering obstacles and the ability to avoid them dynamically. Concluding the successful development of the MADAR robot's mobile base control system, efficient localization and navigation is achieved.

Keywords: *Mobile Base, Holonomic Robot, Navigation, ROS, Localization.*

ÍNDICE GENERAL

ABREVIATURAS

ESPOL	Escuela Superior Politécnica del Litoral
MADAR	*Mobile Anthropomorphic Dual-Arm Robot*
ROS	*Robot Operating System*
URDF	*Unified Robotics Description Format*
SDF	*Simulation Description Format*
Xacro	XML *macro*
XML	*eXtensible Markup Language*
EtherCAT	*Ethernet for Control and Automation Technology*
AMCL	*Adaptive* Monte Carlo *Localization*
LIDAR	*Light Imaging Detection and Ranging*
RQT	*Report Quality Plotting Module*
SOEM	*Simple Open* EtherCAT
IP	*Internet protocol*

SIMBOLOGÍA

m	Metro
kg	Kilogramo
s	Segundo
ms	Milisegundo
kWh	Kilowatt por hora
rad	Radian
V	Voltaje
H	Hora
W	Watts

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1

1. Introducción

Desde 1990, se ha visto una notable influencia de la robótica en la sociedad y en la industria. La capacidad de los robots para realizar tareas repetitivas y pesadas ha revolucionado la forma en que las empresas abordan la producción y ha mejorado la eficiencia en los tiempos de trabajo. Además, al desempeñar estas tareas, los robots protegen a los trabajadores humanos de la fatiga y otros riesgos a su salud que podrían resultar al realizar estas tareas repetitivamente.

La implementación de la robótica ha sido ampliamente aceptada en el mercado y esto puede ser evidenciado por el aumento de la demanda de robots industriales en todo el mundo. Como se puede apreciar en la Figura 1.1, la tendencia ha sido hacia un aumento constante en la demanda de estos robots, lo que indica una fuerte adopción de esta tecnología en la industria [1].

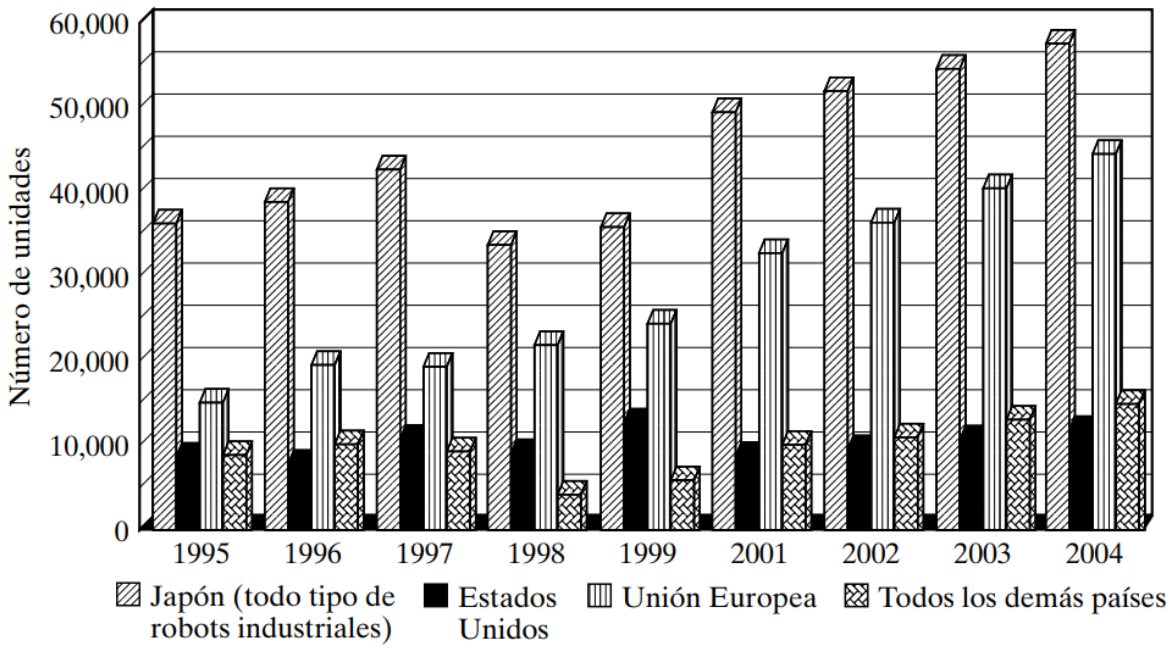

Figura 1.1 Instalaciones de robots industriales por año para los años 1995-2000 con proyección a los años 2001-2004 [1].

Hoy en día, en la industria, en algunos procesos se contempla la realización de tareas en conjunto con un robot. Esto se debe a que, aunque la máquina tenga una precisión y resiliencia superiores a las de un operador, en algunos casos, la supervisión humana es necesaria, ya sea para asignar tareas o detener procesos que esté realizando el robot. Es crucial considerar el riesgo asociado con el uso de este tipo de maquinaria. Como resultado, se han creado robots colaborativos que se pueden encontrar en diferentes sistemas flexibles de fabricación [2].

Además, existen varios tipos de robots colaborativos, que están diseñados en función de la aplicación y el comportamiento deseado. Entre las aplicaciones que se han diseñado se incluyen la manipulación, control de calidad, montaje y soldadura, entre otras [3].

Aunque este tipo de máquinas están diseñadas para operar junto a un humano, algunos robots pueden tener comportamientos no intuitivos debido a su estructura. Por esta razón, existe un interés en desarrollar robots que tengan una forma similar a la de una persona, para que cuando trabajen con un operador, éste espere un comportamiento o rango de movimiento similares a los de un humano, ya que es más fácil predecir el comportamiento y movimientos de este tipo de robots, conocidos como antropomórficos. Un ejemplo de este tipo de robots es el MADAR, que se puede ver en la Figura 1.2.

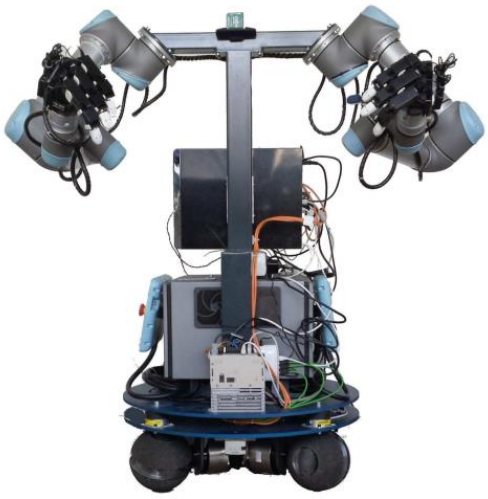

Figura 1.2 Robot móvil antropomórfico bibrazo [4].

1.1 Descripción del problema

En una universidad en España se ha construido un manipulador móvil bibrazo equipado con manos mecánicas, que se llama MADAR (*Mobile Anthropomorphic Dual-Arm Robot*). Al que se le ha desarrollado un sistema de control que permite la ejecución de movimientos coordinados de los dos brazos y las dos manos. La implementación se ha realizado en ROS (*Robotic Operating System*).

Actualmente, existe la necesidad de añadir el control de la base móvil (que permite movimientos holonómicos) proporcionando información necesaria para poder implementar algoritmos de planificación de rutas y navegación de forma autónoma con la capacidad de evadir obstáculos de forma dinámica.

Este proyecto se enfoca en dar a la plataforma del MADAR capacidades de navegación autónoma en el laboratorio universitario, teniendo en cuenta obstáculos que puedan surgir en tiempo real. Para alcanzar este objetivo, se desarrollará un modelo cinemático, se implementarán algoritmos de localización, planificación de rutas y navegación, permitiendo controlar eficazmente la base móvil del robot MADAR. Además, los paquetes deben ser compatibles con entornos simulados y reales, brindando información valiosa a través de la comunicación de ROS para coordinar los movimientos de brazos y manos del robot de manera eficiente.

1.2 Justificación del problema

El proyecto es crucial para el Laboratorio de Robótica de la Universidad Politécnica ya que será esencial para diversos proyectos de investigación que requieren la capacidad de manipular objetos en diferentes estaciones de trabajo y realizar una amplia gama de tareas. Actualmente, no existe un método para determinar la posición y orientación del robot en el laboratorio, lo que hace que el proyecto sea aún más importante.

1.3 Objetivos

1.3.1 Objetivo general

- Desarrollar un sistema de control de la base móvil del robot MADAR que permita la localización y navegación.

1.3.2 Objetivos específicos

- Modelar las ruedas omnidireccionales de la base del MADAR.
- Desarrollar herramientas de simulación para estimar los movimientos de la plataforma con el fin de validar modelos cinemáticos del robot.
- Crear librerías y paquetes en ROS para el control de la plataforma real y simulada.
- Diseñar el controlador de la base del MADAR verificado y operativo para facilitar su integración.
- Implementar algoritmos de localización y navegación.

1.4 Marco teórico

1.4.1 ROS

ROS es un conjunto de librerías y utilidades que sirven para desarrollar programas enfocados en el mundo de la robótica. En el que se puede implementar el paradigma de componentes de *software* aplicado a la robótica. El cual consiste en dividir un problema grande en varios subproblemas pequeños [5]. ROS está organizado en paquetes en el cada uno de estos posee una combinación de códigos, configuraciones y documentación.

Todos los paquetes dentro de su directorio de raíz deben poseer los archivos *CMakeList.txt* y *package.xml* [6], en donde se describen las instrucciones de construcción y las propiedades del paquete, así como el contenido, nombre, versión, autor, dependencias de otros paquetes. *Catkin* [7] es un sistema de construcción de ROS que genera los programas ejecutables, librerías e interfases.

Los paquetes relacionados con ROS deben ser agrupados en un solo directorio denominado "Espacio de Trabajo", que comúnmente se conoce como "catkin_ws". La Figura 1.3 muestra la estructura de un Espacio de Trabajo con varios paquetes.

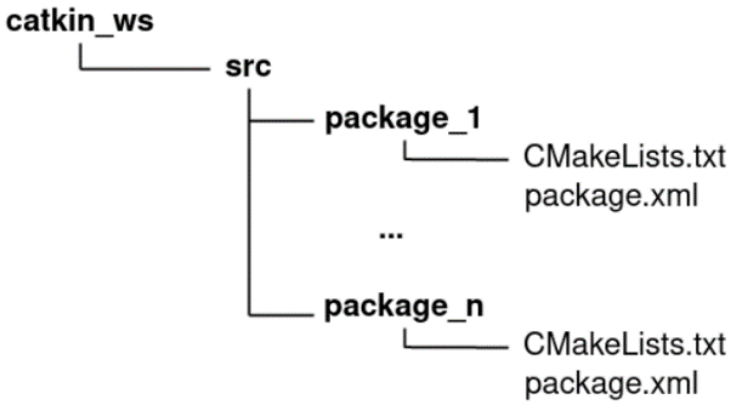

Figura 1.3 Estructura de espacio de trabajo [4].

1.4.1.1 Estructura

ROS utiliza una arquitectura basada en nodos, donde cada nodo se comunica y recibe mensajes de otros nodos en la red. Como se muestra en la Figura 1.4, existe un nodo maestro que se encarga de asignar nombres y registrar cada nodo conectado al sistema. Este nodo funciona como un servidor centralizado, coordinando la comunicación eficientemente entre todos los nodos en la red [8].

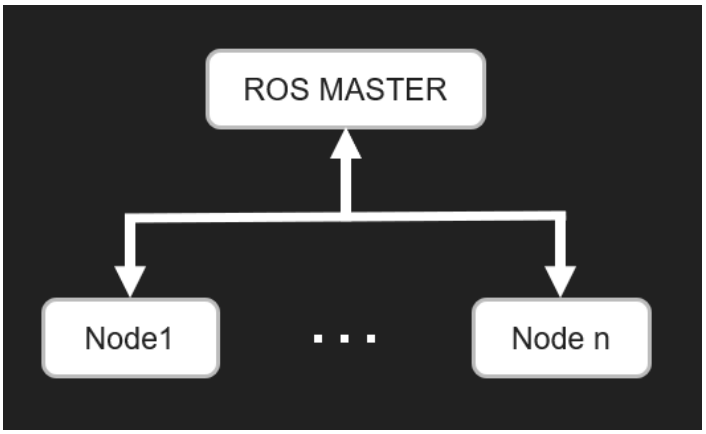

Figura 1.4 Estructura ROS.

1.4.1.2 Ejecución

Antes de la ejecución de cualquier nodo, se debe contar con un nodo “ROS MASTER” corriendo. Esto se realiza por medio del comando *roscore* en una terminal.

Uno de los métodos para realizar la ejecución de un nodo en ROS es por medio de la terminal, haciendo uso del comando *rosrun*, seguido del nombre del paquete y el nombre del archivo ejecutable.

rosrun nombre_del_paquete nombre_nodo

El comando *roslaunch* tiene como función principal ejecutar varios nodos a la vez y establecer algunos parámetros durante una única ejecución. Antes de iniciar la creación de los nodos, *roslaunch* verifica si existe un nodo maestro en la red y, en caso de que no

exista, crea uno. Este proceso no es secuencial y los nodos se inician casi de manera simultánea [9].

Roslaunch nombre_del_paquete nombre_del_archivo

1.4.1.3 Comunicación

La comunicación entre nodos se puede realizar por medio de metodologías suscriptor/publicador o cliente/servidor. Los tópicos (buses con nombres por el que varios nodos se comunican por medio de una semántica anónima) hacen uso del método de comunicación suscriptor/publicador, mientras que los servicios usan el método de comunicación de cliente/servidor [10].

En la Figura 1.5 se muestra un esquema de cómo sería la comunicación entre tres nodos a través de un tópico llamado 'Imagen'. El nodo publicador de la cámara publica la información en ese bus, al que los nodos detectores de carros y personas se suscriben para realizar sus procesos de manera independiente.

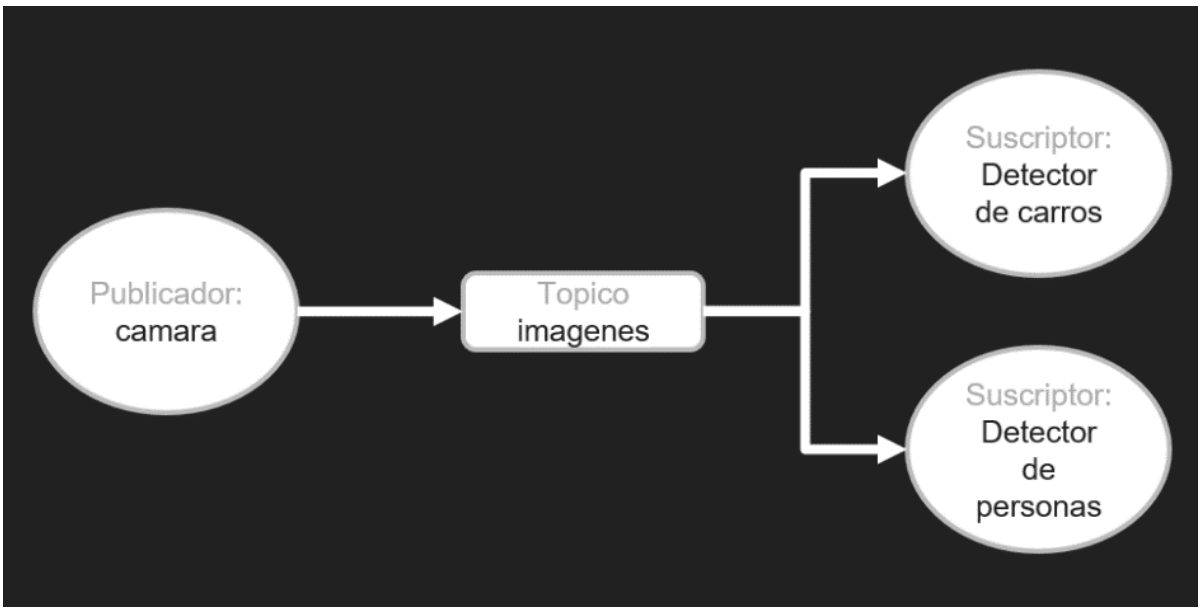

Figura 1.5 Ejemplo de comunicación por medio de tópico en ROS.

La comunicación a través de servicios se lleva a cabo mediante una estructura de solicitud y respuesta. Un nodo en ROS ofrece el servicio bajo un nombre único y un cliente puede invocar el servicio enviando un mensaje de solicitud y esperando una respuesta. La Figura 1.6 muestra un ejemplo de la comunicación a través de servicios de ROS, donde el nodo "Batería" solicita encender el LED 3 al servidor "Panel de LED".

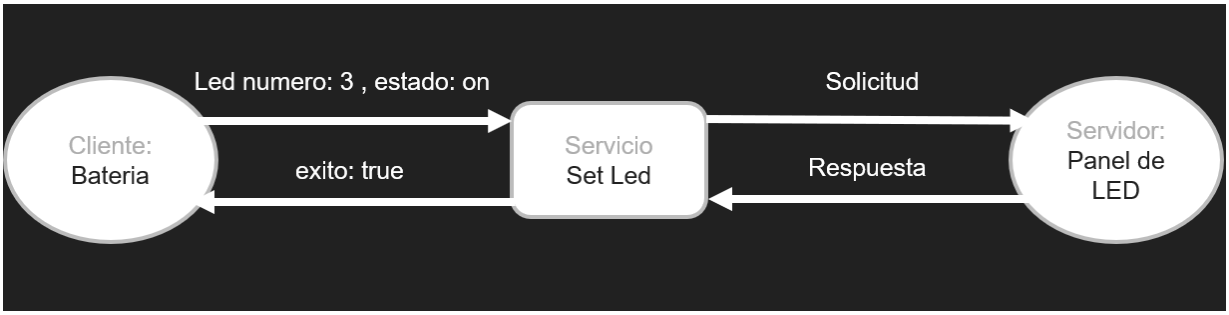

Figura 1.6 Comunicación por medio de servicios en ROS.

1.4.1.4 Visualización y simulación

ROS hace uso de URDF (*Universal Robotics Description Format*) se usa en Gazebo para describir la geometría, dinámica y apariencia de robots [11]. Es un formato de archivo XML que permite representar robots y sus componentes en tres dimensiones para su visualización y simulación en un entorno virtual. Las etiquetas principales las cuales se contemplan en la Figura 1.7 para describir un robot son:

- ***Link:*** Describe las propiedades inerciales del eslabón del robot, su modelo visual y el modelo de la colisión
- ***Joint:*** Describe la junta entre eslabones, el tipo de la junta, (definiendo los eslabones que se conectan por medio de una relación padre e hijo), la transformación que existen entre ellos y el eje de la junta.
- **Robot:** Describe el robot completo, el nombre, las etiquetas *links* y las etiquetas *joint*.

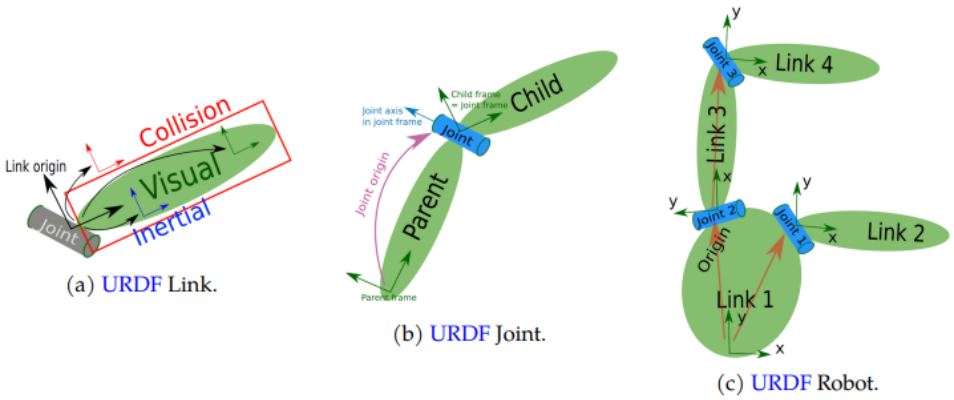

Figura 1.7 Etiquetas principales de un URDF [11].

1.4.1.5 Gazebo

Gazebo es un programa de código abierto que se utiliza en el campo de la robótica para mejorar la investigación y el desarrollo de robots. Con él se pueden crear ambientes virtuales que imitan el comportamiento y la interacción de los robots en un entorno real. La utilización de Gazebo permite a los investigadores y desarrolladores probar, mejorar y evaluar los controladores y algoritmos antes de implementarlos en robots reales, lo que reduce el tiempo y los costos de la investigación y el desarrollo, así como permite una evaluación exhaustiva de diferentes opciones de diseño y control antes de la implementación final. Se observa en la Figura 1.8 una simulación de un entorno en Gazebo en la que se está realizando el control de un brazo robótico dentro de un almacén [12] .

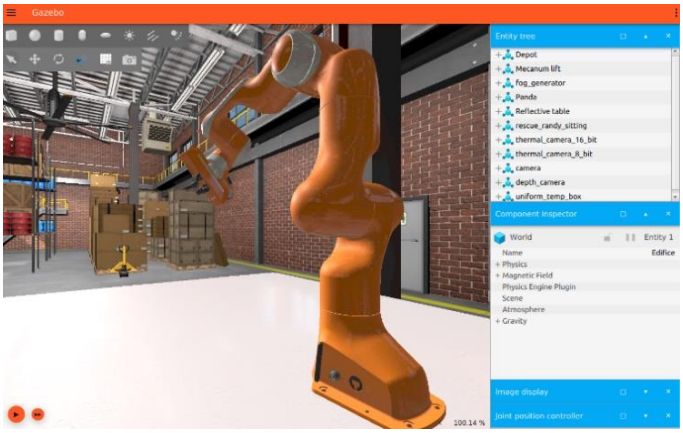

Figura 1.8 Simulación en Gazebo [12].

1.4.2 Navegación

Para los robots móviles, la navegación consiste en la habilidad de desplazarse de una posición inicial a una final, evitando situaciones peligrosas como colisiones. Para que se pueda realizar una navegación efectiva se deben usar las siguientes 3 ramas [13]:

- Localización
- Planificación de trayectorias
- Construcción de mapas

1.4.2.1 Localización y mapeo simultáneo

La localización y mapeo simultáneo es un problema que tienen los robots en el que deben construir un mapa del entorno conocido o desconocido por medio de sensores, para luego poder estimar la posición en la que se encuentra el robot dentro del mapa.

1.4.2.1.1 Gmapping

gmapping es una técnica de robótica que permite crear un mapa del entorno, utilizando datos recolectados de los sensores, como el LIDAR. Esta técnica emplea un filtro de Kalman para determinar la posición del robot y un algoritmo de Monte Carlo para generar un mapa más preciso [14]. La Figura 1.9 muestra un ejemplo de un mapa generado mediante la implementación de *gmapping*.

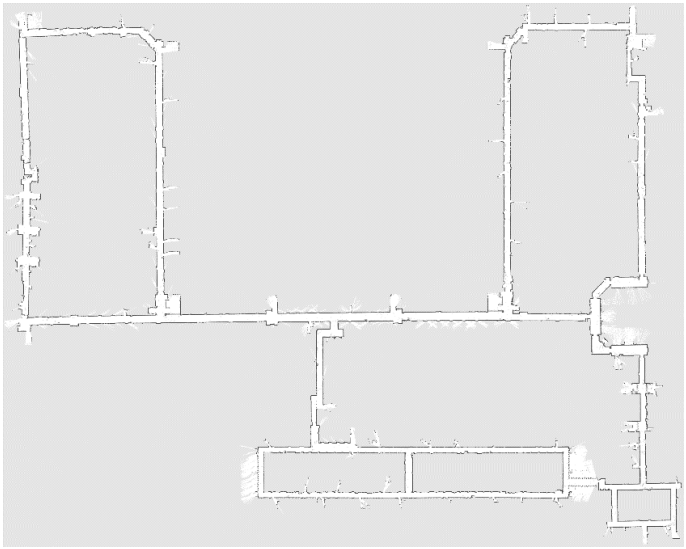

Figura 1.9 Visualización de mapa usando *gmapping.*

1.4.2.1.2 Filtro de kalman

El filtro de Kalman es una técnica de procesamiento de señales, que permite estimar el estado de un sistema dinámico a partir de mediciones inciertas. El algoritmo combina un modelo matemático del sistema con las mediciones, para calcular una estimación del estado del sistema, en un momento dado [15]. El filtro Kalman utiliza las mediciones pasadas y el modelo matemático para calcular la estimación del estado actual.

1.4.2.1.3 Monte Carlo

El algoritmo de Monte Carlo es un método estadístico que se utiliza para resolver problemas complejos, mediante la generación de un gran número de muestras aleatorias de una distribución desconocida, utilizando estas muestras para estimar la distribución desconocida.

Una de las aplicaciones del algoritmo es la generación de mapas precisos del entorno a partir de una gran cantidad de números de mediciones de sensores, mediante el uso de una técnica de filtrado de partículas para representar una distribución de probabilidad sobre la posición del robot y el mapa, finalmente, alcanzando una solución óptima a través de un gran número de muestreos aleatorios [16].

1.4.2.2 Odometría

La odometría es la estimación de la posición que tiene actualmente un robot móvil cuando este está desplazándose. Esto lo hace por medio de la información de los *encoders* que tienen las ruedas los cuales permiten saber la velocidad rotacional de las ruedas [17].

1.4.2.3 Planificación de trayectorias

La planificación de trayectorias consiste en tratar de encontrar un movimiento óptimo en el que no existan colisiones cuando el robot se desplaza entre una configuración inicial (inicio) y una final (objetivo) dentro de un entorno establecido.

1.4.3 Red EtherCAT

EtherCAT es un protocolo de red industrial que utiliza Ethernet para conectar y controlar dispositivos en aplicaciones industriales de automatización. Es rápido, flexible y eficiente en términos de ancho de banda, ideal para aplicaciones que requieren una transmisión de datos en tiempo real y una sincronización precisa. Además, EtherCAT es compatible con una amplia gama de dispositivos y es ampliamente utilizado en sectores como la automatización de la fábrica, la robótica, la aeroespacial y la automoción [18].

CAPÍTULO 2

2. Metodología

En este capítulo se explican los pasos realizados para poder conseguir los objetivos establecidos en el proyecto. También se explican las distintas partes del proceso del diseño. Planteando varias alternativas para poder resolver la problemática, de las cuales se seleccionó la más viable en base a los requerimientos necesarios del sistema. Por último, se muestra el desarrollo e implementación del sistema de control que satisficiera con los objetivos.

2.1 Selección de alternativas

Dado el problema que se definió, la solución se puede dividir en base a las alternativas para la simulación y en alternativas para el sistema de control que tiene la base móvil, las cuales se explican a continuación en la tabla 2.1 y tabla 2.2.

Tabla 2.1 Alternativas de solución de simulación

Alternativa 1	Simular todo el robot en gazebo. Incluir a su vez las matrices de inercias que deben tener todas las juntas del modelo, además de los controladores.
Alternativa 2	Simular en Gazebo de manera cinemática, ignorando la influencia de la gravedad, lo que ofrece la oportunidad de probar y desarrollar la interacción de los sensores con el entorno virtual.
Alternativa 3	Simular la plataforma del robot en Gazebo sin considerar los brazos y las manos.

Tabla 2.2 Alternativas de solución de control

Alternativa 1	Diseño de control de lazo abierto, asumir que no existen errores y operar a velocidades bajas para minimizar el error.
Alternativa 2	Diseño de control realizando fusión de sensores, de los cuales se tienen *encoders* para los motores y LIDAR.
Alternativa 3	Control cinemático solo usando información de IMU, para poder implementar un filtro de Kalman.

Con el fin de elegir la opción más adecuada de simulación, se llevó a cabo un análisis de cada una de las alternativas basándose en los criterios establecidos en la Tabla 2.3. A continuación, se presenta la importancia de cada criterio con relación a la solución:

- **Viabilidad:** facilidad realizar la solución y se puedan realizar pruebas con el robot.
- **Carga computacional optima:** solución con menor cantidad de recursos computacionales para realizar la simulación.
- **Similitud con robot real:** similitud al contrastar con el robot real los resultados, contando con el menor error posible.
- **Tiempo de operación:** solución que tenga menos pasos intermedios para realizar la prueba, será considerada la que tenga tiempo más optimo.

Tabla 2.3 Criterios de selección para solución de simulación

CRITERIOS DE SELECCIÓN			
Peso	**Criterio**	**Rango de Importancia**	**% de Decisión**
3	Viabilidad	1	37.5
2.5	Similitud con el robot real	2	31.25
1	Tiempo de operación	3	12.5
0.5	Carga computacional optima	4	6.25
8	**TOTAL**		**100**

Para la selección de la mejor alternativa del control, se hizo un análisis de cada una de las opciones en base a los criterios definidos en la Tabla 2.4, se muestra la importancia que tienen cada uno de los criterios que se encuentran en la tabla:

- **Viabilidad:** facilidad de realizar la solución y se puedan realizar pruebas con el robot.
- **Carga computacional óptima:** solución con menor cantidad de recursos computacionales para realizar la simulación.
- **Seguridad:** modelo de control que contenga menor probabilidad y rango de error.
- **Tiempo de operación:** la solución que tenga menos pasos intermedios para realizar la prueba será considerada la que tenga tiempo más optimo.

Tabla 2.4 Criterio de selección para solución de control.

CRITERIOS DE SELECCIÓN			
Peso	**Criterio**	**Rango de Importancia**	**% de Decisión**
3	Seguridad	1	37.5
2.5	Viabilidad	2	31.25
1	Tiempo de operación	3	12.5
0.5	Carga computacional optima	4	6.25
8	**TOTAL**		**100**

Se evaluaron, tanto las alternativas de control como las alternativas de simulación, teniendo como base a los criterios definidos en las tablas 2.3 y 2.4. Se pudo definir que la mejor solución para la simulación de la plataforma es la primera alternativa ver tabla 2.5 y que se va a intentar controlar la plataforma real y simulada con la segunda alternativa ver tabla 2.6.

Tabla 2.5 Matriz de decisión de alternativas de solución para simulación

	Criterios de Simulación					Resultados	
Pesos	2.5	3	1	0.5			
Opciones	Similitud con robot	Viabilidad	Tiempo de operación	Carga computacional optima	Puntaje sin peso	Puntaje con peso	Prioridad
Alternativa 1	2.5	2	1	0	5.5	13.25	1
Alternativa 2	1.5	2	0.5	0.5	4.5	10.5	2
Alternativa 3	1	3	0	0.5	4.5	8.75	3

Tabla 2.6 Matriz de decisión de alternativas de solución para control

	Criterios de Control					Resultados	
Pesos	3	2.5	1	0.5			
Opciones	Seguridad	Viabilidad	Tiempo de operación	Carga computacional optima	Puntaje sin peso	Puntaje con peso	Prioridad
Alternativa 1	1	2.5	1	0.5	5	10.5	2
Alternativa 2	3	1	0	0	4	11.5	1
Alternativa 3	1.5	1.5	0.5	0	3.5	8.75	3

2.2 Diseño conceptual

La Figura 2.1 presenta el boceto inicial para lograr una navegación eficiente en un robot. Se requieren dos componentes principales: un punto de inicio (representado en verde) y un objetivo (representado en rojo) para generar una trayectoria de desplazamiento. Además, se necesitó un método para definir una zona de seguridad (representado con verde) para evitar colisiones, en caso de que el robot detecte un objeto dentro de la zona este pueda reaccionar con tiempo para poder detenerse.

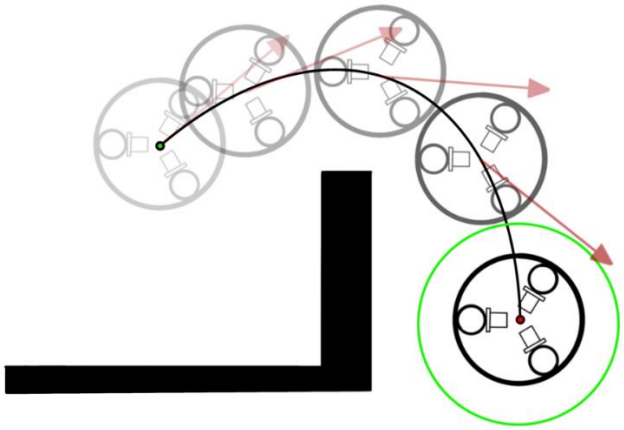

Figura 2.1 Diseño conceptual.

2.3 Proceso de desarrollo

Para lograr un desarrollo óptimo de la alternativa seleccionada, se desarrollaron paquetes que se usaron tanto en el robot real como en la simulación dinámica. Por lo tanto, se tomó en consideración el plan que se puede apreciar en la Figura 2.2. El plan consiste en comenzar con la creación del modelo dinámico, seguido de la implementación del control sobre las ruedas en la simulación. De esta manera, se podrán utilizar paquetes de navegación de forma segura y aplicarlos en el robot real.

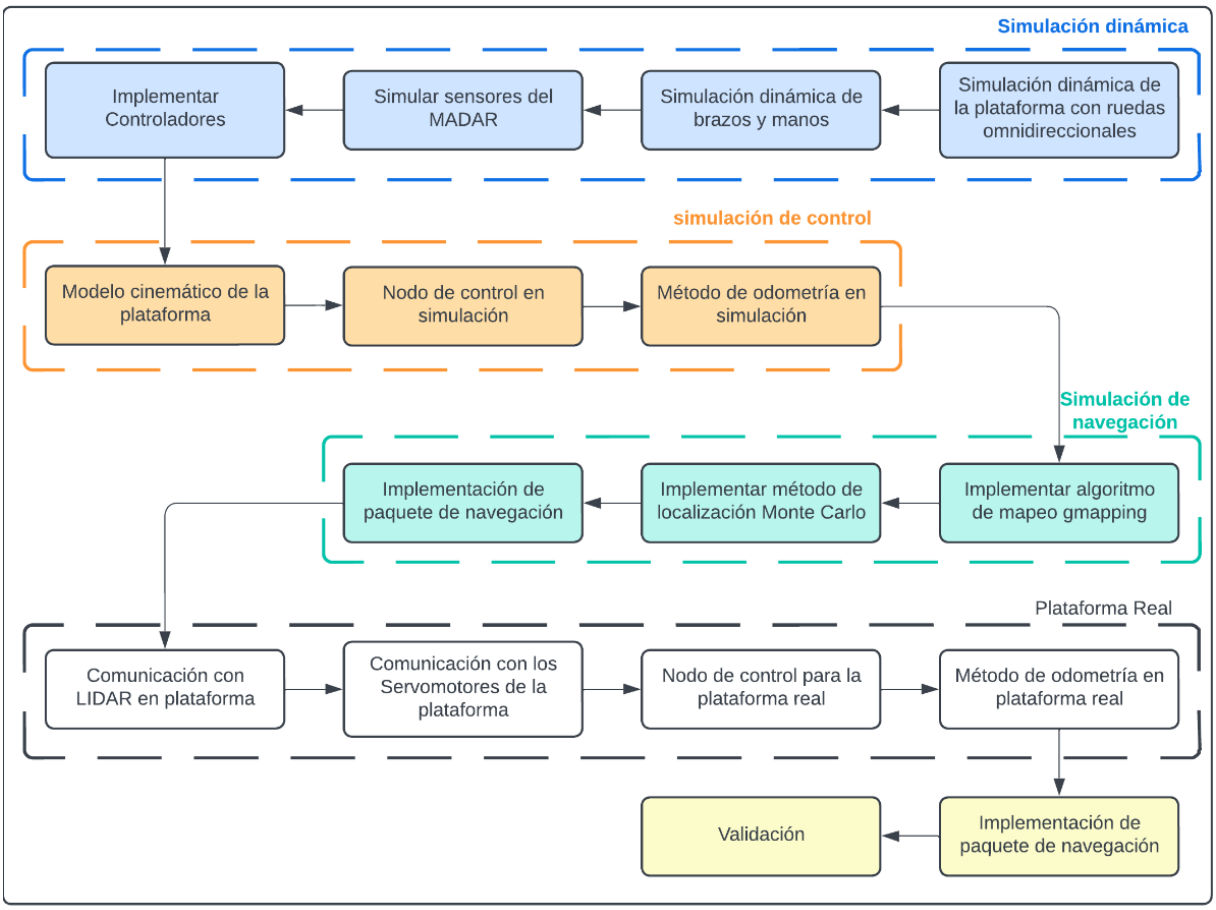

Figura 2.2 Plan de desarrollo de proyecto contemplando pasos para simulación y robot físico.

2.3.1 Simulación dinámica

Para poder realizar una simulación dinámica en gazebo se requirió de la descripción del robot, la cual esta especificada en un archivo XML. Se tuvo que crear el archivo que describen a las ruedas y la plataforma, este proceso se detalla en el Apéndice A, en este se explica cómo se obtuvieron las matrices de inercias, proceso para la obtención la malla de colisión de las ruedas y como se crea el archivo para la descripción de la plataforma.

Una vez obtenido el modelo dinámico de la plataforma, se procedió con la creación de un archivo XML principal en el que se incluyeron los archivos que describen cada miembro del robot (plataforma, brazos y manos). Esta implementación esta descrita en el apéndice B.

Se agregaron después los sensores LIDAR en descripción la plataforma, fusionando la información de cada uno para solo publicar en un tópico referirse al apéndice C y se implementaron los controladores para poder mover las articulaciones y ejes de ruedas que tiene el MADAR está implementación se puede ver en más detalle en el apéndice D.

2.3.2 Simulación de control

Para poder realizar el control de la plataforma se consideró el diagrama de la Figura 2.3 dando como resultado el modelo cinemático de la ecuación 2.1, el cual su obtención se detalla en el Apéndice E. se pudo obtener las velocidades angulares que deben tener los motores para poder realizar un movimiento en el plano deseado.

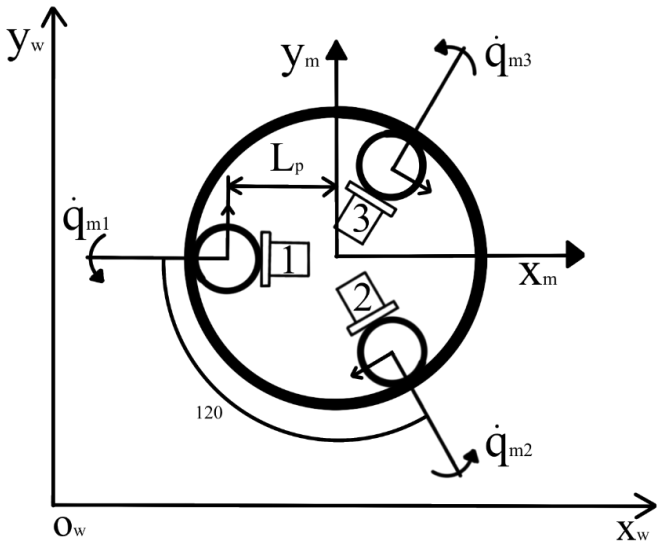

Figura 2.3 Diagrama de cuerpo libre de la plataforma.

$$A.\, v_m^m = \begin{bmatrix} 0 & 10 & -2.9 \\ -8.6603 & -5 & -2.9 \\ 8.6603 & -5 & -2.9 \end{bmatrix} \begin{bmatrix} u \\ v \\ r \end{bmatrix} = \begin{bmatrix} \dot{q}m1 \\ \dot{q}m2 \\ \dot{q}m3 \end{bmatrix} \quad (2.1)$$

Después de definir el modelo cinemático se creó un nodo de ROS el cual se suscribe a un tópico esperando recibir el vector v_m^m esto se hace con el fin de calcular la velocidad angular requerida en cada motor y publicar esa información a los controladores de velocidad para los motores que se encuentran instanciados en gazebo. La creación del nodo se especifica dentro del apéndice F.

Se implemento un método para la estimación de la odometría que solo aplica en la simulación dinámica es cual fue por medio de paquetes de gazebo, la implementación de los métodos esta descrita dentro del Apéndice G.

2.3.3 Simulación de navegación

Para realizar la navegación se requiere de un mapa, se hizo la utilización del paquete *gmapping* para poder crear un mapa del laboratorio usando los sensores del robot. La información necesaria para implementar el paquete se encuentra en el Apéndice H. Se ejecutó una simulación en gazebo con implementando el paquete de *gmapping* y se teleoperó el robot a para ir creando un mapa en la simulación.

Una vez que se había obtenido un mapa del laboratorio, se procedió a aplicar el método de localización basado en Monte Carlo. Este enfoque se eligió en lugar de utilizar el paquete de *gmapping*, ya que éste podía generar errores al intentar actualizar constantemente el mapa. En cambio, el algoritmo de Monte Carlo actualiza la posición del robot en el mapa mediante la modificación de la transformación entre el marco de odometría y el marco del mapa, como se detalló en el apéndice I.

Por último, se procedió con la implementación de la paquetería de navegación de ROS para la plataforma, el proceso de implementación esta descrito en el apéndice J. Al finalizar la implementación se pudo conseguir una simulación en gazebo del MADAR dentro del laboratorio, en la que se tiene la capacidad de navegar hacia una pose deseada evadiendo obstáculos de forma dinámica con el siguiente esquema que se ve en Figura 2.4.

El esquema de la Figura 2.4 indica que el robot intentará alcanzar una pose objetivo. En caso de detectar un obstáculo durante la ejecución del plan, seguirá los siguientes pasos para tratar de despejar el espacio:

1. **Reinicio conservador:** Los obstáculos ubicados fuera de una región específica definida por el usuario serán eliminados del mapa del robot.
2. **Rotación de limpieza:** El robot realizará una rotación en su sitio para liberar el espacio.
3. **Reinicio agresivo:** El robot liberará su mapa de manera más agresiva, eliminando todos los obstáculos ubicados fuera de la región rectangular en la que puede girar sobre su propio eje.
4. **Rotación de limpieza:** Luego, seguirá otra rotación en su sitio.

Si todos estos pasos fallan, el robot considerará que su objetivo es inviable y notificará al usuario que ha abortado la tarea.

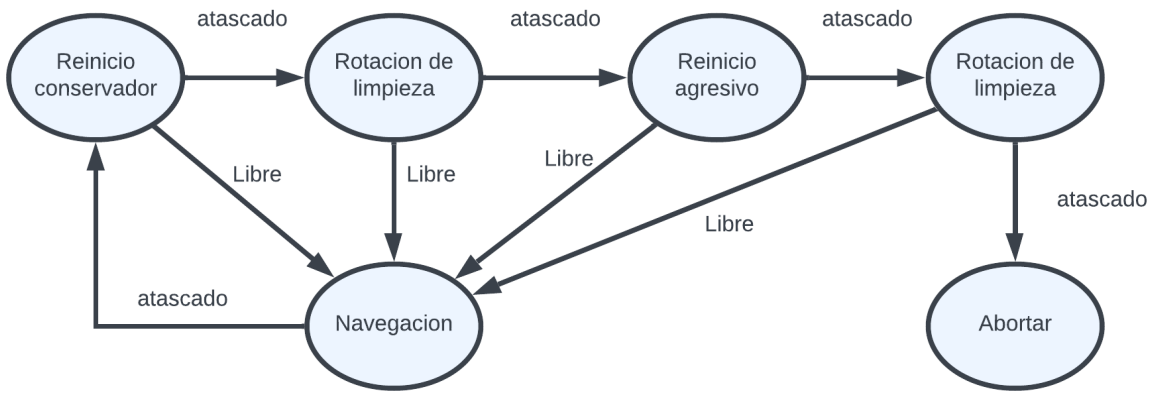

Figura 2.4 Comportamiento del robot para intentar navegar a una pose deseada.

2.3.4 Plataforma real

Dado el modelo cinemático del robot, se modificó una librería en C++ la cual se encarga de la comunicación de la plataforma con los motores, esto se hacía a través de la red EtherCAT ver apéndice L. Luego se procedió con la creación de un paquete en ROS que hace uso de las librerías modificadas con el fin de poder controlar la plataforma con tópicos de ROS ver apéndice M.

Los sensores LIDAR de la plataforma se comunican por medio de una red ethernet en el que cada sensor tiene su IP estática, se genera un nodo de ROS por cada uno de los sensores, los cuales publicaran la información a distintos tópicos los que estará suscrito otro nodo que se encarga de combinar esta información.

Dado que la plataforma real usa los mismos tópicos que los implementados para la simulación dinámica, estos se usaron de igual manera para realización la navegación en dentro del laboratorio de la universidad.

CAPÍTULO 3

3. Resultados y análisis

En esta sección, se presentan los resultados de la simulación dinámica y su aplicación real. Se analizaron los costos obtenidos por la integración del sistema de navegación autónomo en la plataforma del MADAR.

3.1 Resultados de simulación

3.1.1 Modelo dinámico del MADAR en Gazebo

Se género un modelo para la simulación dinámica del MADAR, donde se tiene en cuenta los sensores y controladores de las articulaciones, permitiendo una representación precisa y realista del robot móvil. La Figura 3.1 muestra el control de los brazos y manos dentro de Gazebo, ofreciendo una visualización concreta del modelo final del robot.

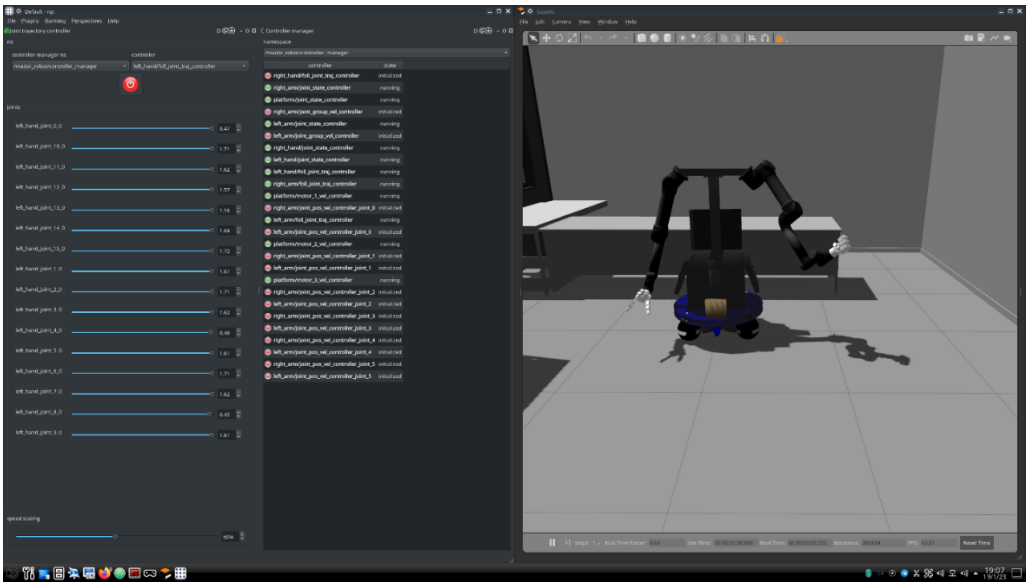

Figura 3.1 Control de posición de mano y brazo del MADAR

Revisar los videos del uno al tres del Apéndice O en los que se aprecia el control de las articulaciones como de la plataforma.

3.1.2 Localización y mapeo simultáneos en gazebo

Se desarrollo un paquete en ROS que permite la localización y mapeo simultáneos dentro del simulador dinámico, se puede ver en la Figura 3.2 como se comienza a generar el mapa del entorno de gazebo, en la Figura 3.3 se obtuvo el mapa del laboratorio dentro de la simulación.

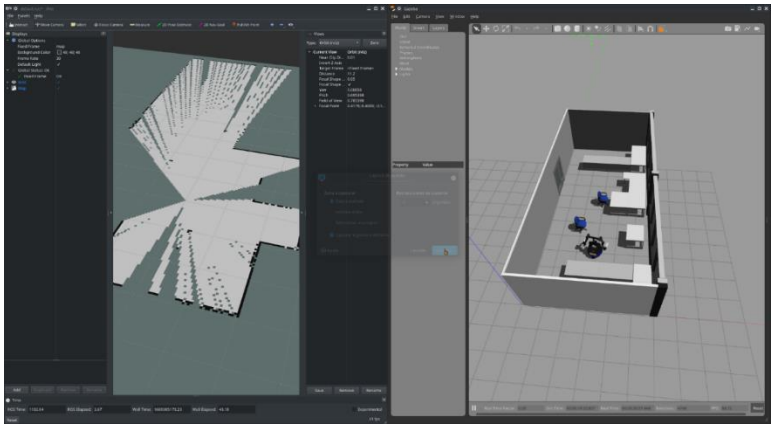

Figura 3.2 Prueba de nodo de *gmapping*.

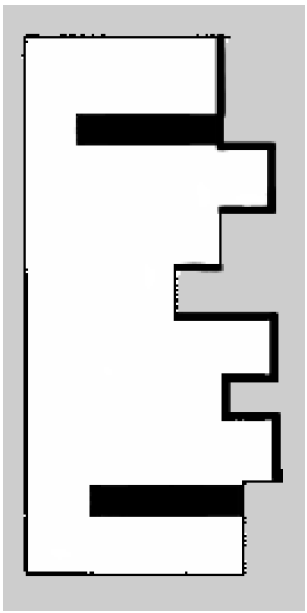

Figura 3.3 Mapa de laboratorio en simulación.

3.1.3 Navegación autónoma

La simulación del MADAR tiene la capacidad de realizar planificación de rutas y navegar hacia puntos de interés de forma autónoma, considerando obstáculos y evadirlos de forma dinámica, en la Figura 3.4 se puede ver el inicio de la navegación del lado izquierdo y el resultado de la navegación a la derecha de la imagen, el cuarto video del Apéndice O muestra una simulación de la navegación en gazebo.

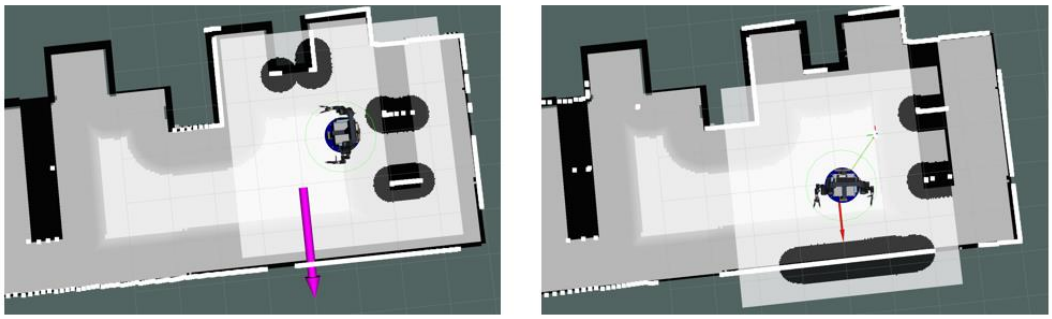

Figura 3.4 Navegación a pose deseada.

3.2 Resultados de plataforma real

3.2.1 Control de plataforma

Se puede controlar la plataforma por medio del tópico /cmd_vel el cual es un vector que está formado por tres elementos:

- Velocidad lineal en x
- Velocidad lineal en y
- Velocidad angular en z

En el video 5 del apéndice O, se ilustra el control de la plataforma mediante un control remoto. La comunicación que se estableció para lograr este control se visualiza en la figura 3.5, donde el nodo /f710_teleop_joy_node se suscribió a un mensaje de estado del control /joy para poder interpretar el mensaje de velocidad que debe publicarse en el nodo de la plataforma a través de /cmd_vel.

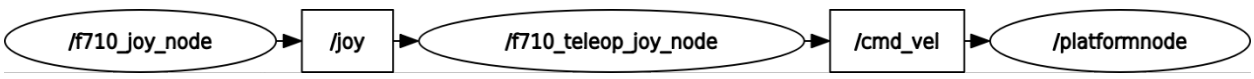

Figura 3.5 Comunicación de nodos para control de plataforma con control remoto.

3.2.2 Localización y mapeo simultáneos en el laboratorio

Se desarrolló un paquete en ROS que permite la localización y mapeo simultáneos usando los sensores LIDAR que tiene la plataforma, se puede apreciar en la Figura 3.6 el mapa del laboratorio generado por el robot.

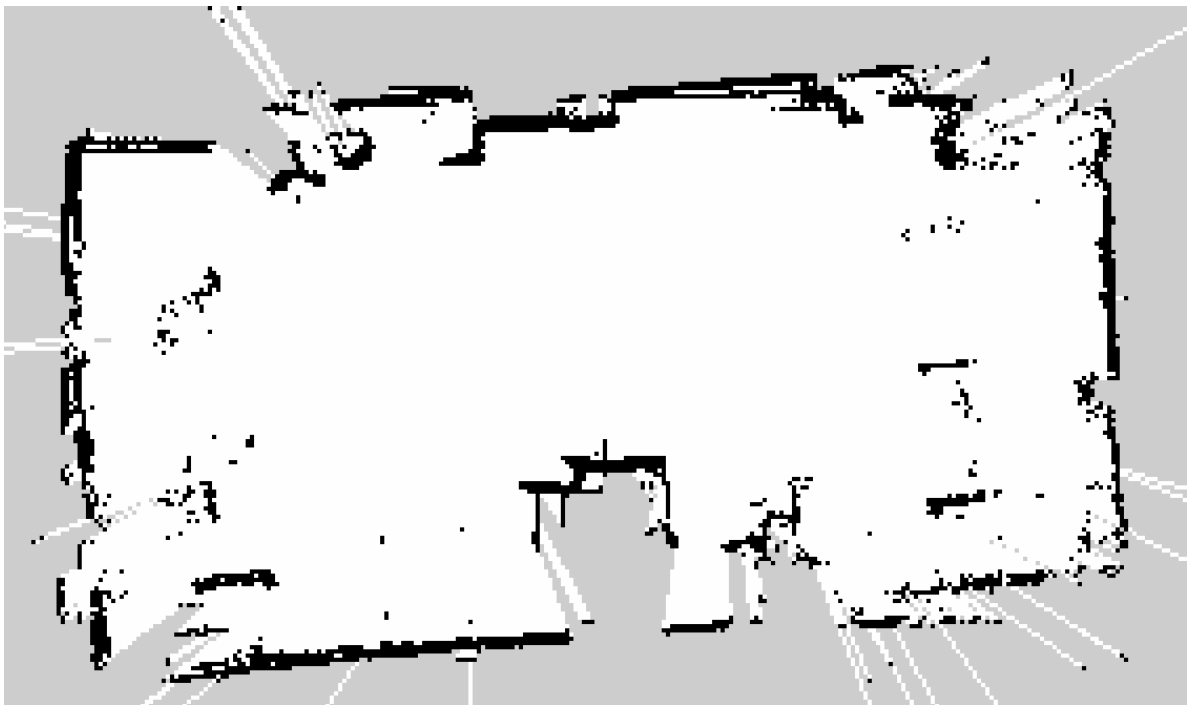

Figura 3.6 Mapa de laboratorio real.

3.2.3 Navegación autónoma dentro del laboratorio

Se creó un paquete en el que se realiza la comunicación con los sensores y *drivers* del MADAR con la finalidad de realizar planificación de rutas y navegar hacia puntos de interés de forma autónoma, esto hizo considerando obstáculos y evadirlos de forma dinámica, como se puede ver en la Figura 3.7 el MADAR está siguiendo un plan para evitar la colisión con la persona que se puede visualizar en la imagen, para observar la ejecución en tiempo real se provee el sexto video del Apéndice O.

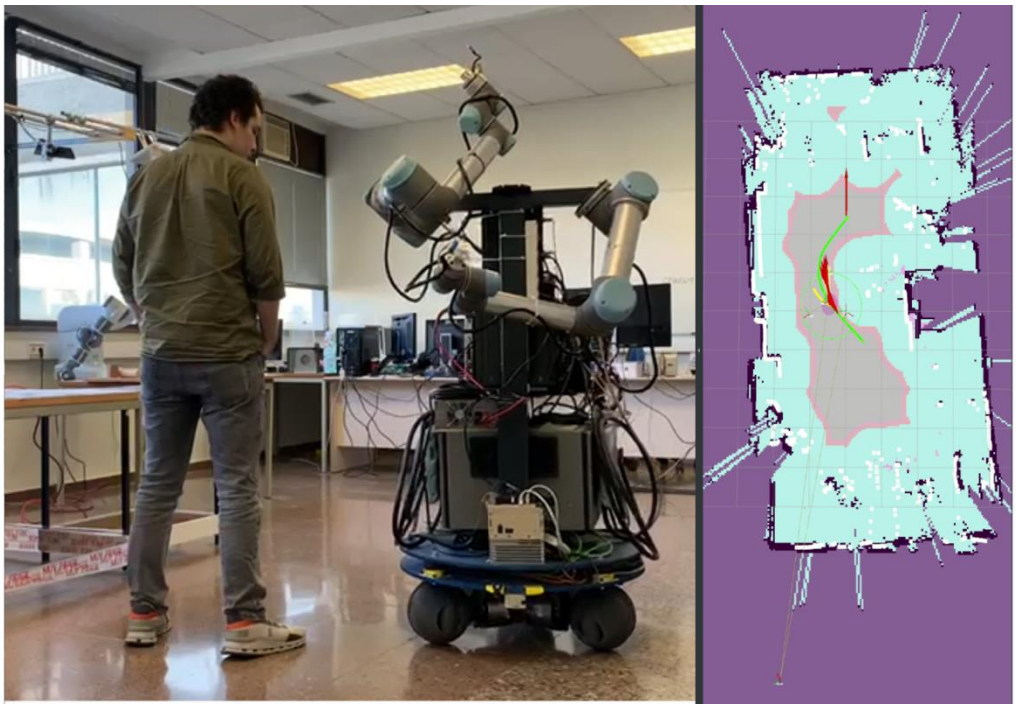

Figura 3.7 Prueba de navegación en el laboratorio.

3.2.4 Manual de usuario

En un repositorio de git se enlistaron los paquetes y librerías necesarias para la ejecución del proyecto, además en este se explica los comandos necesarios para realizar todos los resultados de simulación como en la plataforma real. Este repositorio se puede acceder con el siguiente enlace: https://gitioc.upc.edu/platform/bmmx.git

3.3 Análisis de costos

Los costos del proyecto se reparten en:

- Equipo para realizar pruebas.
- Equipo de Desarrollo.
- Consumo eléctrico.
- Horas de trabajo.

El equipo para realizar las pruebas incluyó la plataforma del MADAR, la cual contiene tres motores SGD7 y la computadora del robot. Costo de depreciación de los componentes se calculó en base al tiempo de uso, considerando un tiempo de vida útil de diez años. Tomando en cuenta que se estima un uso de cuatro horas al día, por cinco días a la semana, durante 48 semanas da una vida útil de 9600 hora. El tiempo de uso del equipo fue un 30% de las horas trabajadas.

El equipo de desarrollo estuvo constituido por una computadora. Contemplando una vida útil de cinco años, cuando esta se opera diez horas por los siete días de la semana. Dando como resultado 18,250 horas de vida útil. El tiempo de uso fue del 100% de las horas trabajadas.

Consecuentemente, dado que el departamento de internados considera un salario de 8 €/h para sus estudiantes. La supervisión de miembros del laboratorio puede ser estimada 200 horas, con un costo promedio de 30 €/h.

El consumo eléctrico que proviene de las computadoras de equipo de desarrollo como la de la plataforma. Abarcando un uso eléctrico promedio de 0.25 €/kWh, el gasto energético de la computadora para el desarrollo se encontraba alrededor de 0.20 kWh. El dispendio de los motores de 0.6 kWh, y el de la computadora del robot de 0.06 kWh.

El proyecto se realizó desde octubre hasta enero (18 semanas), con un promedio de siete horas al día, trabajando 5 días a la semana, acumulando un total de 630 horas. En la tabla 3.1 se puede apreciar el presupuesto descrito. Con un costo total de 12601.241€.

Tabla 3.1 Presupuesto del proyecto.

Encabezado de costos	**Descripción**	**Costo fijo €**	**Vida útil H**	**Variable de costo €**	**Tiempo de uso H**	**Costo €**
Equipo para realizar pruebas	SGD7(x3) Computadora Plataforma	9,000 2,000	9,600 9,600	0.9375 0.21	288 288	270(x3) 37.8
Equipo de Desarrollo	Computadora de desarrollo	2,000	18,250	0.109	630	69.041
Consumo eléctrico	SGD7(x3) Computadora plataforma	- -	- -	0.6 0.2	288 630	172.8(x3) 126
Horas de estudiante		-	-	8	630	5,040
Horas de supervisión		-	-	30	200	6,000
Costo total						12,601.241

CAPÍTULO 4

4. Conclusiones y recomendaciones

En esta sección, se muestran las conclusiones detalladas al integrar un sistema de navegación autónomo en la plataforma del MADAR. Además, se brindan valiosas recomendaciones para optimizar su uso y desarrollo en el futuro.

4.1 Conclusiones

- El modelado de las ruedas omnidireccionales de la base del robot MADAR para su implementación en simulaciones dinámicas en gazebo, se ha realizado de forma exitosa. Esto ha permitido una mejor comprensión de las capacidades de movimiento de estas ruedas. El modelo dentro de la simulación dinámica tiene una gran capacidad para estimar el comportamiento de las ruedas en diferentes entornos. Además, este modelo es útil para la optimización del control de movimiento y la planificación de rutas en la navegación del robot.
- El desarrollo de herramientas de simulación para estimar los movimientos de la plataforma del robot ha sido un éxito. Dentro de esta se validó el modelo cinemático del robot. Estas herramientas permiten predecir el comportamiento del robot en diferentes entornos y situaciones, lo que es esencial para garantizar la seguridad y eficiencia en su navegación. Además, estas herramientas también son útiles para la optimización de los modelos cinemáticos y para la investigación futura en robótica.
- Se consiguió la creación de librerías y paquetes en ROS para el control de la plataforma real y simulada del robot. Estos paquetes y librerías permiten un control preciso y eficiente tanto de la plataforma real como simulada, lo que es esencial para garantizar la seguridad y eficiencia en la navegación del robot. Además, estos paquetes y librerías también son útiles para la investigación futura en el laboratorio.

- El desarrollo de un sistema de control de la base móvil del robot MADAR ha sido efectuado de manera satisfactoria, ya que, este ha permitido la localización y navegación del robot de manera eficiente. A su vez, gracias a la implementación de los paquetes de navegación de ROS, se ha logrado un sistema que garantiza una mayor precisión en la navegación y una mayor seguridad en su movimiento.
- Según los resultados, se ha determinado que el 87,60% del costo total del proyecto está distribuido entre las horas invertidas por el estudiante y el supervisor. Del costo total, un 47,61% se deriva de las horas dedicadas por el supervisor y un 39,99% corresponde a las horas dedicadas por el estudiante.

4.2 Recomendaciones

- Revisar la parametrización de los componentes del MADAR, incluyendo computadoras, baterías, monitores, entre otros. Actualmente, no se utilizaron valores precisos de peso o geometría en la descripción del robot. Si se pueden mejorar las matrices de inercia, se logrará una simulación más acorde entre el modelo y el robot real.

- Se recomienda implementar un modelo dinámico en lugar del actual modelo de cinemática directa utilizado en el nodo de control. Esto permitirá una mejora en la precisión y eficiencia del control del robot, ya que un modelo dinámico considera tanto la cinemática como la dinámica del sistema.

- Se sugiere la creación de un nuevo nodo que permita cambiar la zona de seguridad del robot en función de la posición de sus brazos. Actualmente, todas las implementaciones consideran que los brazos se encuentran en el centro, lo que puede ser limitante en situaciones en las que los brazos se desvíen de su posición original. Con este nuevo nodo, se podrá adaptar la zona de seguridad en tiempo real, mejorando la seguridad y eficiencia del robot.

- Se propone agregar sensores adicionales, como cámaras de profundidad, que permitan analizar objetos a nivel de los brazos del robot. Actualmente, todos los sensores se encuentran en la base de la plataforma, lo que limita la capacidad del robot para interactuar con objetos a su alrededor. Con la adición de cámaras de profundidad en los brazos, se mejorará la capacidad de detección y manipulación de objetos, aumentando la eficiencia y versatilidad del robot.

BIBLIOGRAFÍA

[1] J. J. Craig, ROBÓTICA, México: PEARSON EDUCACIÓN, 2006.

[2] E. Vallejo, «Sistemas flexibles de manufactura,» 1998.

[3] E. Dominguez, C. Franklin, J. Fryman y M. Lewandowski, «Collaborative robotics: New era of human–robot cooperation in the,» 2020.

[4] O. Palacin, «Motion Coordination of a Dual-Arm,» Escola Tècnica Superior d'Enginyeria Industrial de Barcelona, España, 2022.

[5] AmandaDattalo, «ros.org,» Open Robotics, 08 08 2018. [En línea]. Available: http://wiki.ros.org/ROS/Introduction. [Último acceso: 20 10 2022].

[6] Open Robotics, «catkin/package.xml,» [En línea]. Available: http://wiki.ros.org/catkin/package.xml . [Último acceso: 05 02 2023].

[7] Open Robotics, «catkin,» [En línea]. Available: http://wiki.ros.org/catkin/package.xml. [Último acceso: 05 02 2023].

[8] AmmarAzab, «ROS.org,» Open Robotics, 20 09 2022. [En línea]. Available: http://wiki.ros.org/ROS/Concepts. [Último acceso: 20 10 2022].

[9] Open Robotics, «roslaunch,» [En línea]. Available: http://wiki.ros.org/roslaunch. [Último acceso: 05 02 2023].

[10] Open Robotics, «Topics,» [En línea]. Available: http://wiki.ros.org/Topics. [Último acceso: 05 02 2023].

[11] Open Robotics, «xacro,» [En línea]. Available: http://wiki.ros.org/xacro. [Último acceso: 05 02 2023].

[12] OpenRobotics, «gazebo,» Open Robotics, [En línea]. Available: https://gazebosim.org/home. [Último acceso: 20 10 2022].

[13] C. Stachniss, Robotic Mapping and Exploration, Springer, 2009.

[14] G. Grisetti, C. Stachniss y W. Burgard, «Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters,» *IEEE Transactions on Robotics,* vol. 23, nº 1, pp. 34-46, 2007.

[15] Durrant-Whyte, «Introduction to Estimation and the Kalman Filter,» The University of Sidney, 2001.

[16] D. F. W. B. Sebastian Thrun, «GRID AND MONTE CARLO LOCALIZATION,» de *PROBABILISTIC ROBOTICS*, 276, The MIT Press, 2005, p. 237.

[17] L. B. J. y. E. H. Feng, «Where am I? Sensors and methods for mobile,» Universidad de Michigan, 1996.

[18] E. Technology, «EtherCAT,» [En línea]. Available: https://www.ethercat.org/en/technology.html. [Último acceso: 22 10 2022].

APÉNDICES

APÉNDICE A

Simulación dinámica de ruedas omnidireccionales

Las ruedas que usa el MADAR son esféricas y están compuestas por siete elementos principales, con la finalidad de permitir un deslizamiento paralelo al eje de rotación de la rueda. Como se puede observar en la Figura A.1 el diseño de los elementos posee una geometría compleja. Esta complejidad resulta en un alto número de triángulos en la malla generada, lo que aumenta la carga computacional en las simulaciones.

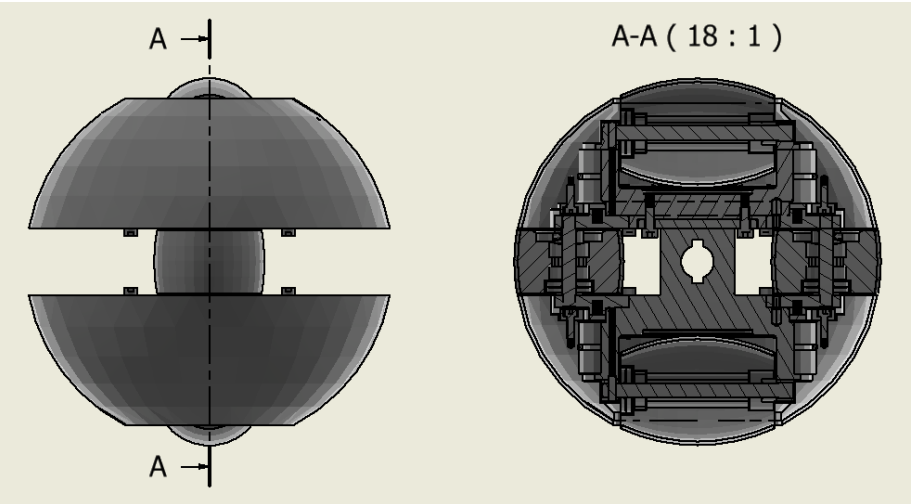

Figura A.1 Ensamble de rueda.

Para solucionar este problema, es necesario simplificar el modelo eliminando componentes innecesarios como pernos y rodamientos, y realizar una re-topología de las partes externas de las piezas. La Figura A.2 muestra la malla resultante de uno de los elementos principales simplificado, que representa la superficie de contacto de las ruedas y se utiliza para simular las colisiones en el sistema.

Figura A.2 Elemento de rueda simplificado.

Además, se usó Autodesk Inventor para estimar las matrices de inercia de cada uno de los componentes que forma la rueda esférica, así como de la estructura de la plataforma dando como resultado los siguientes valores que se visualizan en la tabla A.1.

Tabla A.1 Valores para matrices de inercias obtenidos en Autodesk Inventor.

Nombre	Masa (kg)	Ixx (kg m^2)	Iyy (kg m^2)	Izz (kg m^2)	Ixy (kg m^2)	Iyz (kg m^2)	Ixz (kg m^2)
Estructura	96	12.54	4.139	12.171	0	0	0
Eje	0.2414	0.000116	0.000112	0.000099	0	0	0
Rueda grande	3	0.005	0.009	0.005	0	0	0
Rueda pequena	1	0.000161	0.000255	0.000255	0	0	0
Rueda mediana	1	0.000087	0.000059	0.000059	0	0	0

Una vez obtenido los valores de las matrices de inercia de la plataforma, se creó un entorno de simulación en Gazebo. Para lograr esto, se usa un archivo XACRO que especifica la formación de los eslabones y juntas del robot, así como las matrices de inercia correspondientes a cada eslabón, como se puede ver en la Figura A.3. Se está creando la descripción de la junta entre la base de la plataforma con el eje de la rueda uno y después se procede a crear el eslabón del eje, este proceso se replica para cada uno de los elementos que forman la plataforma, el esquema de la relación de las juntas para la plataforma es el que se encuentra en la Figura A.4, haciendo que el código final para la descripción de la plataforma con las ruedas sea el que está en el primer elemento de la tabla en el Apéndice P:

```
<!-- ################################################################################################################### -->
<!--                                                START OF WHEEL 1                                                   -->
<!-- ################################################################################################################### -->
<joint name="motor_1" type="continuous">
  <parent link="chassis"/>
  <child link="wheel_1_link"/>
  <axis xyz="-1 0 0"/>
  <origin rpy="2.0944 0 1.5708" xyz="0 -0.2905 0.1"/>
</joint>
<link name="wheel_1_link">
  <visual>
    <origin xyz="0 0 0.018" rpy="0 0 0"/>
    <geometry>
      <mesh filename="package://madar_description/urdf/meshes/wheel_stl/shaft_hole.stl"/>
    </geometry>
  </visual>
  <collision>
    <geometry>
      <box size="0.01 0.01 0.01"/>
    </geometry>
    <material name="black"/>
  </collision>
  <inertial>
    <mass value="0.214"/>
    <!-- values calculated using inventor -->
    <inertia ixx="0.00011655" ixy="0.0" ixz="0.0" iyy="0.00011214" iyz="0.0" izz="0.00009944"/>
    <!-- values calculated using inventor -->
  </inertial>
</link>
<gazebo reference="wheel_1_link">
  <mu1>100</mu1>
  <mu2>100</mu2>

  <gravity>true</gravity>
  <material>Gazebo/Black</material>
</gazebo>
```

Figura A.3 Código de junta de la estructura con eje de rueda 1 y creación de eslabón del eje.

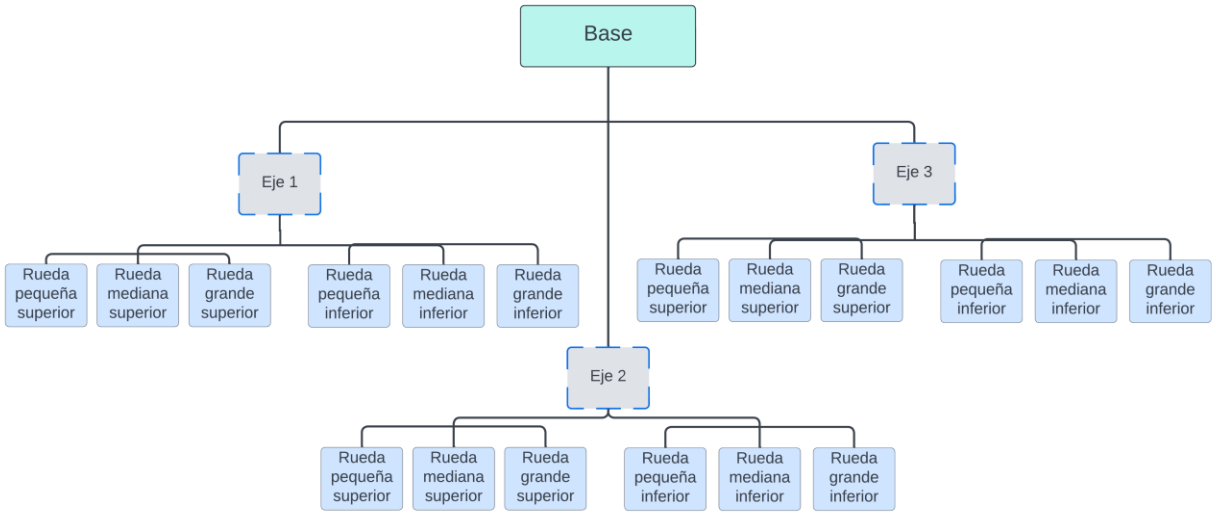

Figura A.4 Esquema de la relación de los eslabones para la plataforma.

Una vez que se cuenta con la descripción del robot se elabora un script de ejecución "*madar.launch*", este script es el que se va a encargar de generar una simulación en gazebo, además de cargar la descripción del archivo XACRO y luego intentar cargar el robot en gazebo, dando como resultado lo que se puede apreciar en la Figura A.5.

Figura A.5 Simulación dinámica de plataforma en Gazebo.

APÉNDICE B

Simulación dinámica de brazos y manos

La información para la descripción de los brazos y las manos fue provisionada por el cliente. Para poder agregar cada miembro en la descripción del robot se creó un archivo XACRO para cada uno de los robots que forman al MADAR, luego se procedió a definir en el archivo principal llamado “madar_robot.xacro” en el que se realizó la conexión de los elementos siguiendo el esquema de la Figura B.1, el código del archivo se puede observar en el segundo elemento de la tabla del Apéndice P.

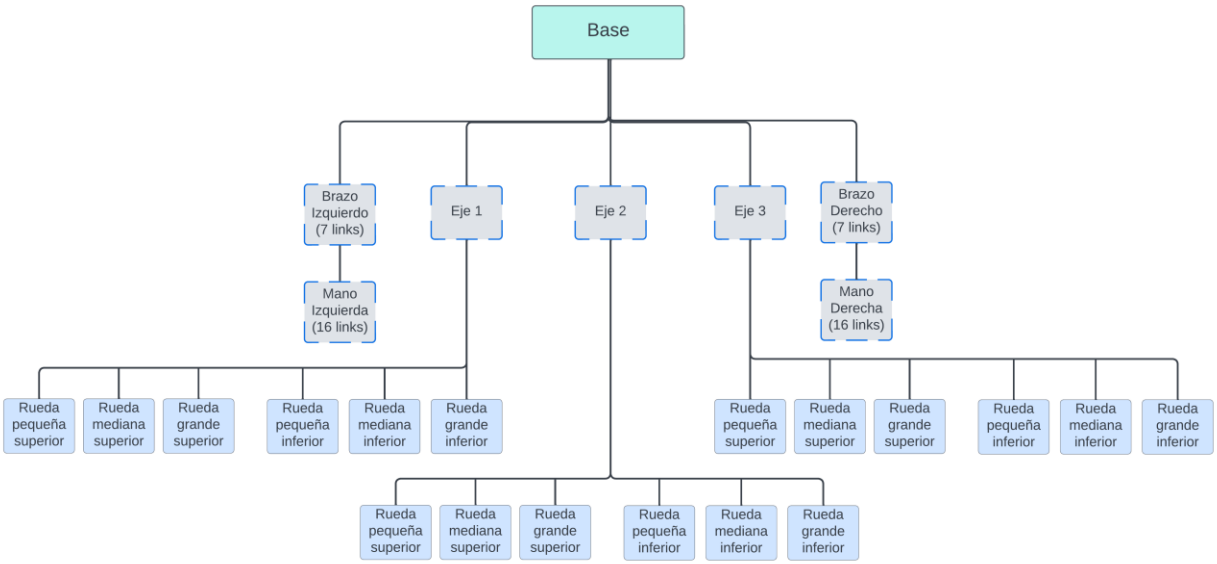

Figura B.1 Esquema de relación entre eslabones para la descripción del MADAR.

Una vez que se cuenta con la descripción completa del MADAR se usa el archivo *launch* creado en el apéndice A, para generar una simulación en gazebo, la cual va a cargar a el robot, generando lo que se puede apreciar en la Figura B.2.

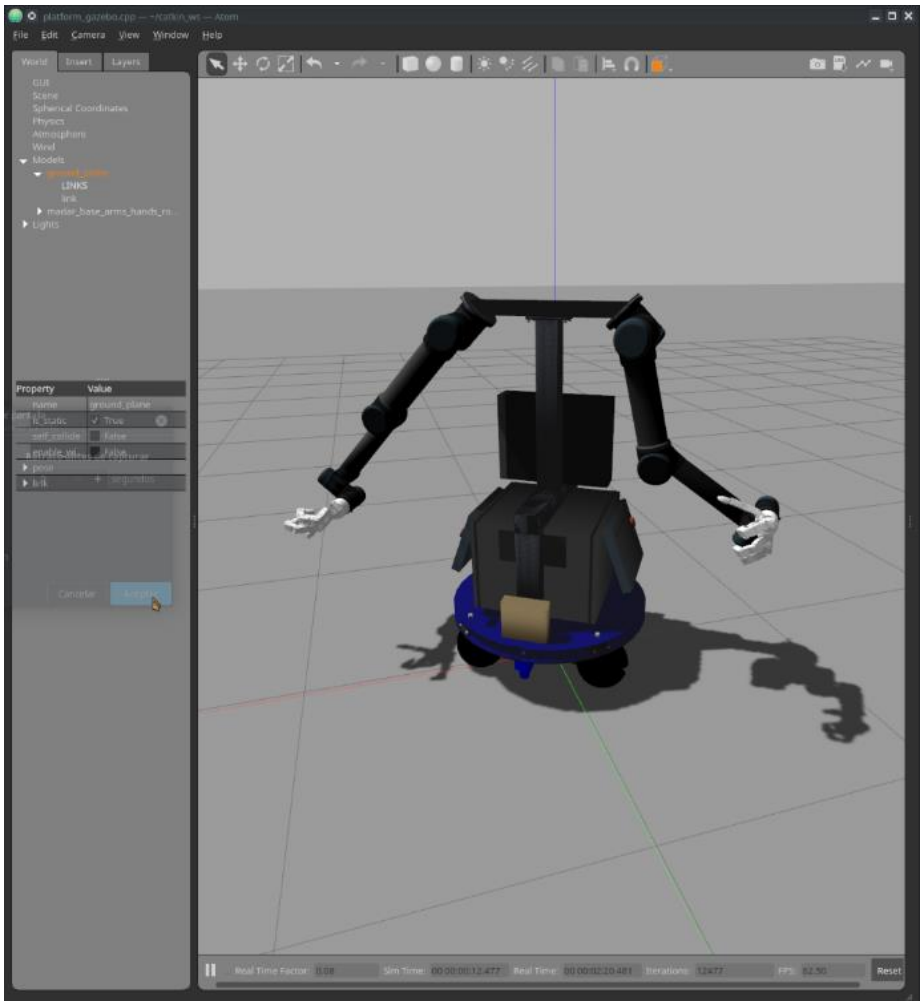

Figura B.2 MADAR simulado en Gazebo.

APÉNDICE C

Simulación de sensores

Se implementó el paquete de ROS de los sensores LIDAR de sick_tim, este paquete permite simular los sensores que están instalados en la plataforma. Para implementarlos en la simulación de gazebo, se deben agregar en la descripción de la plataforma las juntas que unen a los eslabones de los sensores con la base de la plataforma del MADAR, la cual cuenta con tres sensores LIDAR, que están separados a 120 grados entre ellos y a 390 mm del centro. Como se puede observar en la Figura C.1, dentro de la descripción se define el nombre del sensor, tópico al que se desea publicar, el ángulo máximo y mínimo.

```
<!-- ##################################################################################################### -->
<!--                                          START OF PLATFORM SENSORS                                    -->
<!-- ##################################################################################################### -->


<joint name="front_sick_tim_joint" type="fixed">
  <parent link="chassis"/>
  <child link="tim_front_mount_link"/>
  <origin rpy="3.14 0.0 1.57" xyz="0.0 0.350 0.215 " />
</joint>

<xacro:include filename="$(find sick_tim)/urdf/sick_tim.urdf.xacro" />
<xacro:sick_tim551 name="tim_front" ros_topic="/scan_tim_front" min_angle="-1.0472" max_angle="1.0472" />

<joint name="left_sick_tim_joint" type="fixed">
  <parent link="chassis"/>
  <child link="tim_left_mount_link"/>
  <origin rpy="3.14 0.0 ${1.57+2.0944}" xyz="-0.3031 -0.175 0.215 " />
</joint>

<xacro:include filename="$(find sick_tim)/urdf/sick_tim.urdf.xacro" />
<xacro:sick_tim551 name="tim_left" ros_topic="/scan_tim_left" min_angle="-1.0472" max_angle="1.0472" />

<joint name="right_sick_tim_joint" type="fixed">
  <parent link="chassis"/>
  <child link="tim_right_mount_link"/>
  <origin rpy="3.14 0.0 ${1.57-2.0944}" xyz="0.3031 -0.175 0.215 " />
</joint>

<xacro:include filename="$(find sick_tim)/urdf/sick_tim.urdf.xacro" />
<xacro:sick_tim551 name="tim_right" ros_topic="/scan_tim_right" min_angle="-1.0472" max_angle="1.0472" />
```

Figura C.1 Código de descripción de los sensores sick_tim.

Se debe combinar la información de los sensores en un solo tópico, para facilitar la implementación de los métodos de mapeo, localización y navegación de la plataforma. Con lo que se implementó un nodo en ROS que se suscribe a cada tópico de los scanner

y publica uno bajo el nombre de “/scan” el cual contiene la información de la señal fusionada de los scanner, para realizar la tarea mencionada, la comunicación de los nodos debe ser como la que se indica en la Figura C.2.

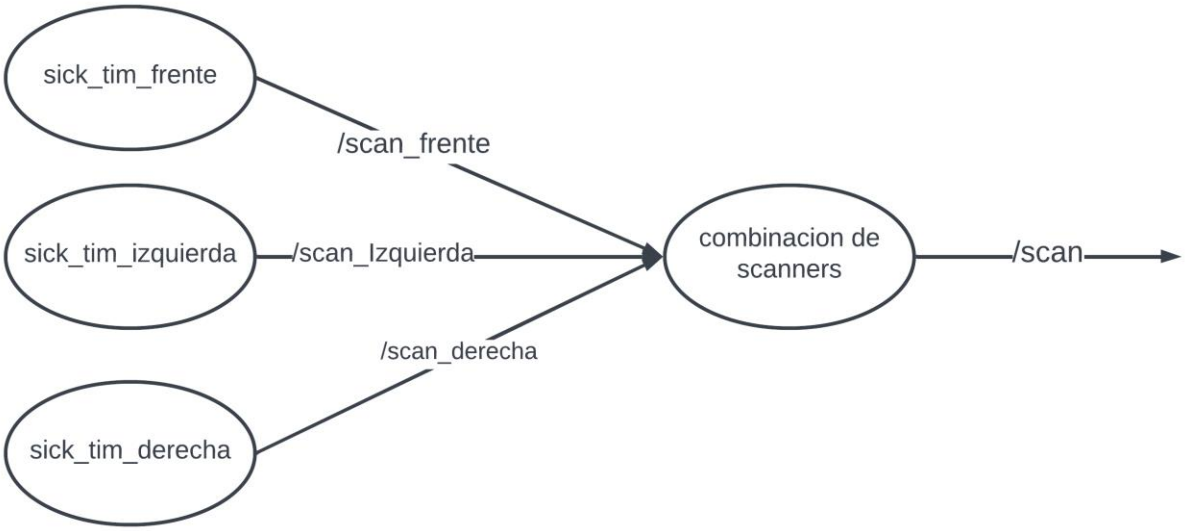

Figura C.2 Comunicación de nodos LIDAR.

APÉNDICE D

Implementación de controladores

Se empleo el uso de *plugins* de ROS para la creación de los controladores de las articulaciones y las ruedas. Las articulaciones cuentan con un control de trayectoria, mientras que las ruedas tienen un control de velocidad. Los valores de los controladores y las juntas que deben ser controladas se definen en un archivo YAML y en el XACRO de cada robot. Este archivo incluye los valores proporcionales, integrales y derivativos para cada actuador definido en la descripción del XACRO.

```yaml
hardware_control_loop:
  loop_hz: &loop_hz 125
joint_state_controller:
  type:         joint_state_controller/JointStateController
  publish_rate: *loop_hz
motor_1_vel_controller:
  type: velocity_controllers/JointVelocityController
  joint: motor_1
  pid:
    p: &kp 1.2
    i: &ki 0.01
    d: &kd 0.1
    i_clamp: &i_clamp 1
motor_2_vel_controller:
  type: velocity_controllers/JointVelocityController
  joint: motor_2
  pid:
    p: *kp
    i: *ki
    d: *kd
    i_clamp: *i_clamp
motor_3_vel_controller:
  type: velocity_controllers/JointVelocityController
  joint: motor_3
  pid:
    p: *kp
    i: *ki
    d: *kd
    i_clamp: *i_clamp
```

Figura D.1 Código de controladores para plataforma.

Se carga dentro del archivo *launch* los parámetros para crear los controladores en la simulación de gazebo esto se puede apreciar el código en el tercer elemento de la tabla del Apéndice P, en este archivo se crea una simulación en gazebo para luego cargar la descripción del robot y por último la carga de los controladores para las juntas.

APÉNDICE E

Modelo cinemático de la plataforma

Analizando la plataforma se puede ver que se cuenta con el esquema que está representado en la Figura E.1, con lo que para realizar el modelo cinemático de la plataforma se debe considerar que el centro de la circunferencia es el marco de referencia de la base, por estándares definidos se debe alinear el eje x de la base con el frente del robot, además de definir cuál será el orden de los motores que se considerarán en nuestra situación y a qué distancia se encuentran del centro, también considerar que los motores se encuentran a 120 grados entre ellos, esta información permite realizar el diagrama de cuerpo libre que se representa en la Figura E.2.

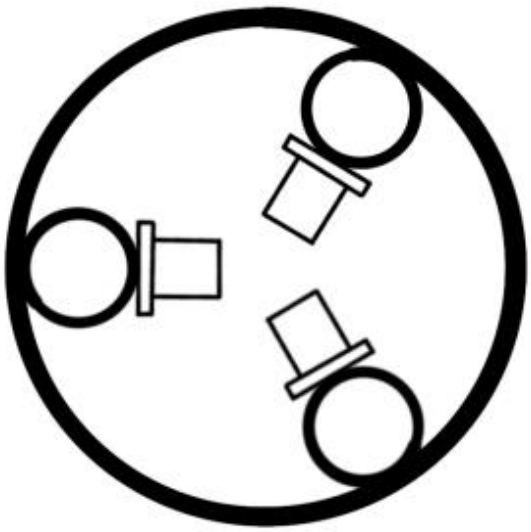

Figura E.1 Diagrama de base del robot.

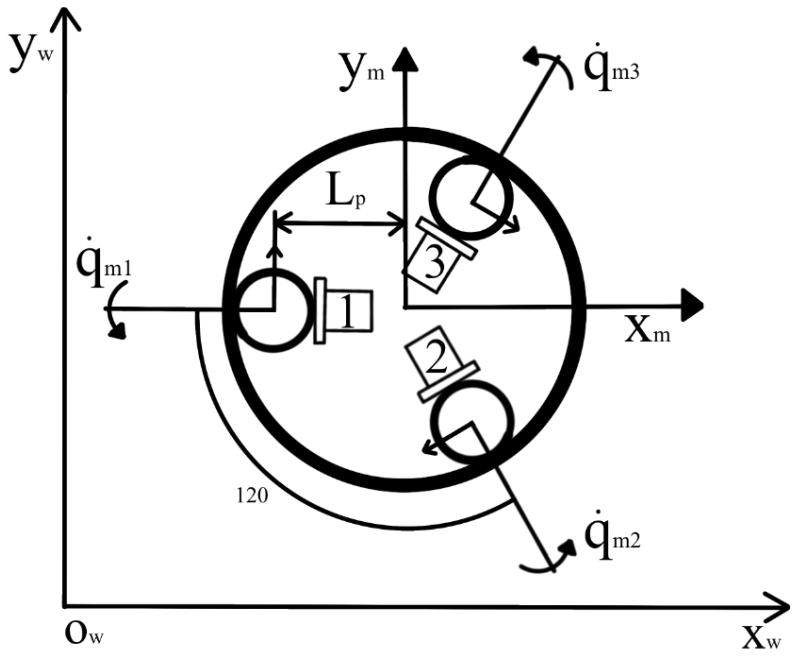

Figura E.2 Diagrama de cuerpo libre de la plataforma.

Dentro de la Figura E.2 se asume que el espacio de trabajo de la plataforma está representado en un sistema de coordenadas absoluto O_w,X_w,Y_w,Z_w , por el que se define un sistema de coordenadas relativo para la plataforma móvil el cual se expresa como O_m,X_m,Y_m,Z_m en donde el origen del sistema de referencia móvil está ubicado en el centro de gravedad de la plataforma y se sabe que las ruedas esta separadas a un ángulo de 120 grado entre ellas.

Al hacer el analis de una rueda como en la Figura E.3 se tienen los siguientes angulos:

- α_i: ángulo entre el eje x de la base con el centro de la rueda.
- β_i: ángulo de dirección de conducción
- γ_i: ángulo de dirección de deslizamiento

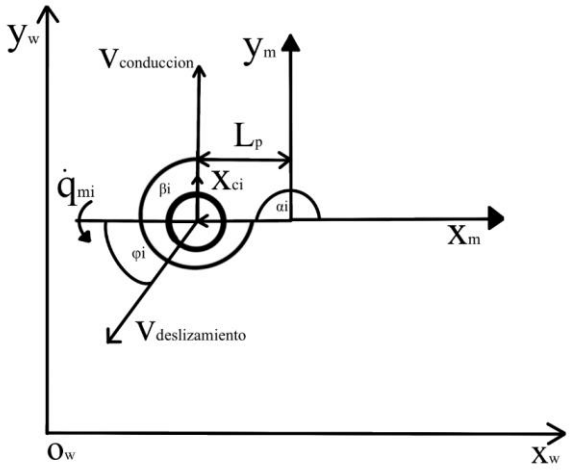

Figura E.3 Diagrama de cuerpo libre de rueda.

Para el caso de estas ruedas se puede considerar que el ángulo de deslizamiento es ortogonal con el vector de desplazamiento para las tres ruedas con lo que se puede decir que el ángulo $\gamma_i = 0$ entonces:

$$\gamma_1 = \gamma_2 = \gamma_3 = 0 \qquad \text{(E.1)}$$

El vector el cual va desde el centro de la circunferencia hasta el centro de la rueda cuenta con una norma de 0.290 m y se encuentran a un ángulo αi Dado que los motores se encuentran con un ángulo de 120 entre los mismos se establece la siguiente relación.

$$\alpha_2 = \alpha_1 + 120 \qquad \text{(E.2)}$$
$$\alpha_3 = \alpha_2 + 120$$

El ángulo entre el vector de conducción y el eje x del marco de referencia de la base se define como β_i , considerando que el ángulo entre las ruedas y el marco de referencia es α_i y el vector de deslizamiento es ortogonal con el vector de desplazamiento, se pueden definir los ángulos β_i como:

$$\beta_i = \alpha_i + 270 \qquad \text{(E.3)}$$

La velocidad de conducción va a depender de la velocidad lineal del centro de la rueda, la cual se puede expresar como el producto de la velocidad angular de la rueda con el radio de la rueda, y también la velocidad de conducción dependerá de la proyección del vector de velocidad de deslizamiento sobre el eje de la velocidad de conducción.

$$\dot{X_{cl}} = V_{conduccion} - V_{deslizamiento} \sin\varphi \qquad \text{(E.4)}$$
$$\dot{Y_{cl}} = V_{deslizamiento} \cos\varphi$$

$$V_{conduccion} = \dot{X_{cl}} + \dot{Y_{cl}} \tan\varphi \qquad \text{(E.5)}$$

$$V_{conduccion} = \omega_i L_{pi} \qquad \text{(E.6)}$$

$$\omega_i = \frac{1}{L_{pi}} (\dot{X_{cl}} + \dot{Y_{cl}} \tan\varphi)$$
$$\omega_i = \begin{bmatrix} \frac{1}{L_{pi}} & \frac{1}{L_{pi}} \tan\varphi \end{bmatrix} \begin{bmatrix} \dot{X_{cl}} \\ \dot{Y_{cl}} \end{bmatrix} \qquad \text{(E.7)}$$

Como se desea conseguir la componente de la velocidad en función al marco de referencia móvil por lo que se va a hacer uso de una matriz de rotación R para poder representar los componentes en el marco de referencia deseado.

$$V_{ci}^{ci} = \begin{bmatrix} \dot{X_{cl}} \\ \dot{Y_{cl}} \end{bmatrix}$$

$$V_{ci}^{m} = {}_{ci}^{m}R(\alpha i) \begin{bmatrix} \dot{X_{cl}} \\ \dot{Y_{cl}} \end{bmatrix}$$

$$V_{ci}^{m} = \begin{bmatrix} \cos(\beta i) & -\sin(\beta i) \\ \sin(\beta i) & \cos(\beta i) \end{bmatrix} \begin{bmatrix} \dot{X_{cl}} \\ \dot{Y_{cl}} \end{bmatrix} \qquad \text{(E.8)}$$

La velocidad lineal de la rueda desde el marco de la base se representa de la siguiente forma, donde (x_i, y_i) es el origen del marco de referencia de las ruedas ver Figura E.4:

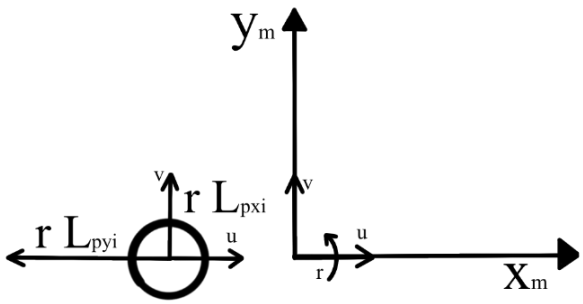

Figura E.4 Diagrama de cuerpo libre de rueda.

$$u - r\,L_{pyi} = V_{cxi}^{m} \qquad (E.9)$$
$$v + r\,L_{pxi} = V_{cyi}^{m}$$

$$V_{ci}^{m} = \begin{bmatrix} 1 & 0 & -L_{pyi} \\ 0 & 1 & L_{pxi} \end{bmatrix} \begin{bmatrix} u \\ v \\ r \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & -L_{pyi} \\ 0 & 1 & L_{pxi} \end{bmatrix} \begin{bmatrix} u \\ v \\ r \end{bmatrix} = \begin{bmatrix} \cos(\beta i) & -\sin(\beta i) \\ \sin(\beta i) & \cos(\beta i) \end{bmatrix} \begin{bmatrix} \dot{X}_{cl} \\ \dot{Y}_{cl} \end{bmatrix} \qquad (E.10)$$

Como la matriz de rotación es ortogonal se puede obtener la transpuesta de la matriz para la inversa

$$\begin{bmatrix} \cos(\beta i) & \sin(\beta i) \\ -\sin(\beta i) & \cos(\beta i) \end{bmatrix} \begin{bmatrix} 1 & 0 & -L_{pyi} \\ 0 & 1 & L_{pxi} \end{bmatrix} \begin{bmatrix} u \\ v \\ r \end{bmatrix} = \begin{bmatrix} \dot{X}_{cl} \\ \dot{Y}_{cl} \end{bmatrix} \qquad (E.11)$$

Con lo que para obtener la velocidad de conducción de una rueda se debe plantear la siguiente ecuación usando la relación (E.11) en la ecuación (E.7) para cada una de las ruedas.

$$\omega_i = \begin{bmatrix} \frac{1}{L_{pi}} & \frac{1}{L_{pi}}\tan\varphi \end{bmatrix} \begin{bmatrix} \cos(\beta i) & \sin(\beta i) \\ -\sin(\beta i) & \cos(\beta i) \end{bmatrix} \begin{bmatrix} 1 & 0 & -L_{pyi} \\ 0 & 1 & L_{pxi} \end{bmatrix} \begin{bmatrix} u \\ v \\ r \end{bmatrix} \qquad (E.12)$$

Al aplicar las ecuaciones E.2 y E.3 y teniendo en cuenta que el vector que va desde el centro de la circunferencia hasta el centro de la rueda tiene una norma de 0.290 m, se obtienen los valores presentados en la tabla E.1.

Tabla E.1 Valores de vectores referenciales.

I	$\boldsymbol{\alpha_i}$	$\boldsymbol{\beta_i}$	$\boldsymbol{Y_i}$	$\boldsymbol{X_i}$
1	180	450	0	-0.290
2	300	570	-0.2511	0.145
3	420	690	0.2511	0.145

Usando los valores de la tabla E.1 en la ecuación E.12 se tiene como resultado los siguientes vectores w_i:

- $W_1 = [0 \quad 10 \quad -2.9]$
- $W_2 = [-8.6603 \quad -5 \quad -2.9]$
- $W_3 = [8.6603 \quad -5 \quad -2.9]$

Con este modelo se pudo obtener las velocidades angulares que deben tener los motores para poder realizar un movimiento en el plano deseado.

$$A.v_m^m = \begin{bmatrix} W1 \\ W2 \\ W3 \end{bmatrix} \begin{bmatrix} u \\ v \\ r \end{bmatrix} = \begin{bmatrix} \dot{q}m1 \\ \dot{q}m2 \\ \dot{q}m3 \end{bmatrix}$$

$$A.v_m^m = \begin{bmatrix} 0 & 10 & -2.9 \\ -8.6603 & -5 & -2.9 \\ 8.6603 & -5 & -2.9 \end{bmatrix} \begin{bmatrix} u \\ v \\ r \end{bmatrix} = \begin{bmatrix} \dot{q}m1 \\ \dot{q}m2 \\ \dot{q}m3 \end{bmatrix} \quad \text{(E.13)}$$

Para expresar la relación entre las velocidades de las ruedas y la velocidad de la plataforma con respecto a un sistema de referencia fijo O_w,X_w,Y_w,Z_w, se requiere una matriz de transformación entre el marco de referencia absoluto y el de la base de la plataforma.

$$ {}^{w}_{m}R = \begin{bmatrix} \cos\ \varphi & -sin\varphi & 0 \\ \sin\ \varphi & cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{E.14} $$

Con lo que para representar la velocidad de la plataforma móvil con respecto al marco de referencia absoluto $v_m^w = [\dot{X}_m^w \quad \dot{Y}_m^w \quad \dot{\varphi}]$ donde los primeros 2 componentes representan la velocidad de la plataforma con respecto al marco fijo.

$$ \dot{q_r} = (A\ .\ {}^{w}_{m}R^T)v_m^w \tag{E.15} $$

Ahora es fácil obtener el modelo de la cinemática directa de la plataforma

$$ v_m^w = J_m(\varphi)\dot{q_r} \tag{E.16} $$

En donde $J_m(\varphi) = (A\ .\ {}^{w}_{m}R^T)^{-1}$ donde la expresión de la matriz del jacobiano de la plataforma se representa de la siguiente forma.

$$ J_m(\varphi) = \begin{bmatrix} 0 - 10\ sin\varphi & 10cos\varphi & -2.9 \\ -8.6603\ cos\varphi + 5\sin\ \varphi & -8.6603\ sin\varphi - 5cos\varphi & -2.9 \\ 8.6603\ cos\varphi + 5\ \sin\ \varphi & 8.6603\ sin\varphi - 5cos\varphi & -2.9 \end{bmatrix}^{-1} \tag{E.17} $$

Si se desea mover la plataforma en los ejes x y y a una velocidad de 1m/s se debe asignar al vector v_m^m los valores de u=1, v =1, r = 0.

$$ W\zeta\ = \begin{bmatrix} 0 & 10 & -2.9 \\ -8.6603 & -5 & -2.9 \\ 8.6603 & -5 & -2.9 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 10 \\ -13.6603 \\ 3.6603 \end{bmatrix} $$

La Figura E.5 muestra el desplazamiento de la plataforma después de un segundo, alcanzando su posición final en el punto (1,1). Durante este periodo, las tres velocidades angulares, w1 = 10 rad/s, w2 = -13.6603 rad/s y w3 = 3.6603 rad/s, se mantuvieron constantes.

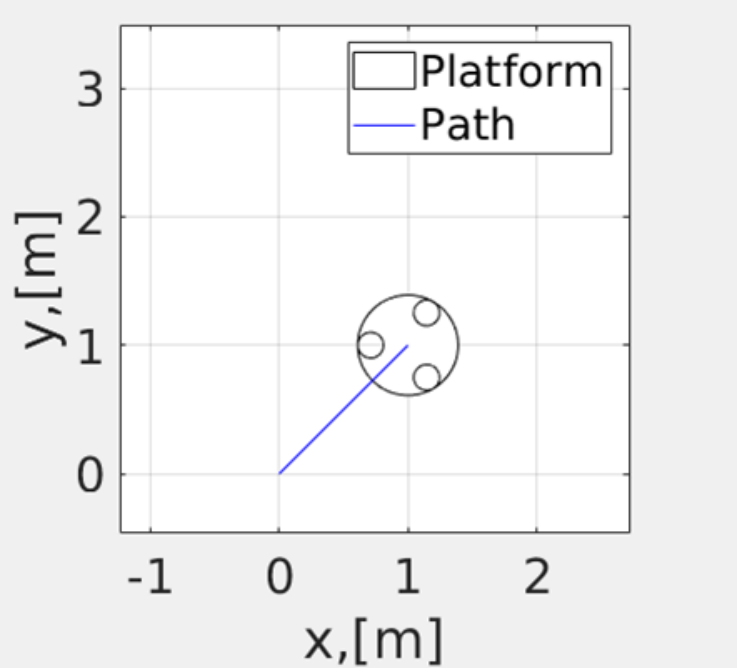

Figura E.5 Prueba del modelo en cinemático MATLAB de velocidad lineal en (x, y).

APÉNDICE F

Nodo de control en simulación

La creación del código para el nodo de control de la plataforma en gazebo se debe suscribir a un topico “/cmd_vel” el cual considerar que este mensaje está formado por:

- Velocidad lineal en x
- Velocidad lineal en y
- Velocidad angular en z

El nodo publica a los tópicos de los motores en gazebo para controlar las velocidades angulares que deberían tener los mismos. Esto lo realiza publicando a los siguientes tópicos:

- "/madar_robot/platform/motor_1_vel_controller/command"
- "/madar_robot/platform/motor_2_vel_controller/command"
- "/madar_robot/platform/motor_2_vel_controller/command"

Cuando se recibe un nuevo mensaje por medio del topico de "cmd_vel” este deberá calcular los valores de las velocidades angulares necesarios para le MADAR con la matriz de cinemática directa obtenida en el apéndice E, ver el cuarto elemento de la tabla del Apéndice P.

APÉNDICE G

Método de odometría en simulación

Para estimar la odometría dentro del simulador dinámico se hizo uso de un plugin de gazebo el cual provee la información de la ubicación exacta del modelo dentro de gazebo, la información la envía por medio de un mensaje de “nav_msgs/Odometry”, con lo que se debe crear un nodo que se suscriba al mensaje y publique la transformación que se genera entre el marco de odometría con la base de la plataforma esto se realiza en el código de la Figura G.1.

```
 geometry_msgs::TransformStamped odom_trans;

void chatterCallback(const nav_msgs::Odometry::ConstPtr& msg)
{
  ROS_INFO("Seq: [%d]", msg->header.seq);
  ROS_INFO("Position-> x: [%f], y: [%f], z: [%f]", msg->pose.pose.position.x,msg->pose.pose.position.y, msg->pose.pose.position.z);
  ROS_INFO("Orientation-> x: [%f], y: [%f], z: [%f], w: [%f]", msg->pose.pose.orientation.x, msg->pose.pose.orientation.y, msg->pose.pose.orientation.
  ROS_INFO("Vel-> Linear: [%f], Angular: [%f]", msg->twist.twist.linear.x,msg->twist.twist.angular.z);


  odom_trans.header.stamp = ros::Time::now();
  odom_trans.header.frame_id = "odom";
  odom_trans.child_frame_id = "base_link";

  odom_trans.transform.translation.x = msg->pose.pose.position.x;
  odom_trans.transform.translation.y = msg->pose.pose.position.y;
  odom_trans.transform.translation.z = msg->pose.pose.position.z;
  odom_trans.transform.rotation = msg->pose.pose.orientation;

}

int main(int argc, char **argv)
{

  ros::init(argc, argv, "odom_listener");


  ros::NodeHandle n;
  ros::Subscriber sub = n.subscribe("odom", 1000, chatterCallback);
  ros::Rate r(30.0);

    tf::TransformBroadcaster odom_broadcaster;


    //since all odometry is 6DOF we'll need a quaternion created from yaw
geometry_msgs::Quaternion odom_quat = tf::createQuaternionMsgFromYaw(0);

//first, we'll publish the transform over tf

odom_trans.header.stamp = ros::Time::now();
odom_trans.header.frame_id = "odom";
odom_trans.child_frame_id = "base_link";

odom_trans.transform.translation.x = 0.0;
odom_trans.transform.translation.y = 0.0;
odom_trans.transform.translation.z = 0.0;
odom_trans.transform.rotation = odom_quat;

//send the transform
odom_broadcaster.sendTransform(odom_trans);

   while(ros::ok()){
     ros::spinOnce();
     odom_broadcaster.sendTransform(odom_trans);

         r.sleep();
   }

  return 0;
}
```

Figura G.1 Código para obtención de odometría.

APÉNDICE H

Implementación de algoritmo de mapeo

Para realizar el mapeo se implementó el algoritmo de *gmapping* este paquete para poder funcionar requiere de los siguientes puntos:

- Macro de referencia de sensor que envía la señal del /*scan*
- información recibida por el scanner
- la odometría del robot

Se usa el código del paquete *gmappin*g en un *launch* en el que se especifican todos los parámetros que tiene el algoritmo para que funcione de forma óptima, ver Figura H.1.

```
<launch>
  <!-- SLAM: Gmapping -->

    <!-- Arguments -->
    <arg name="set_base_frame" default="base_link" />
    <arg name="set_odom_frame" default="odom" />
    <arg name="set_map_frame" default="map" />
    <arg name="set_scan_topic" default="scan"/>
    <arg name="node_start_delay" default="5.0" />
    <arg name="use_sim_time" default="false"/>
    <param name="use_sim_time" value="$(arg use_sim_time)"/>
    <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" output="screen" launch-prefix="bash -c 'sleep $(arg node_start_delay); $0 $@'">
      <param name="base_frame" value="$(arg set_base_frame)" />
      <param name="odom_frame" value="$(arg set_odom_frame)" />
      <param name="map_frame" value="$(arg set_map_frame)" />
      <param name="scan" value="$(arg set_scan_topic)"/>
      <param name="map_update_interval" value="5.0"/>
      <param name="maxUrange" value="16.0"/>
      <param name="sigma" value="0.05"/>
      <param name="kernelSize" value="1"/>
      <param name="lstep" value="0.05"/>
      <param name="astep" value="0.05"/>
      <param name="iterations" value="5"/>
      <param name="lsigma" value="0.075"/>
      <param name="ogain" value="3.0"/>
      <param name="lskip" value="0"/>
      <param name="srr" value="0.1"/>
      <param name="srt" value="0.2"/>
      <param name="str" value="0.1"/>
      <param name="stt" value="0.2"/>
      <param name="linearUpdate" value="1.0"/>
      <param name="angularUpdate" value="0.5"/>
      <param name="temporalUpdate" value="3.0"/>
      <param name="resampleThreshold" value="0.5"/>
      <param name="particles" value="30"/>
      <param name="xmin" value="-50.0"/>
      <param name="ymin" value="-50.0"/>
      <param name="xmax" value="50.0"/>
      <param name="ymax" value="50.0"/>
      <param name="delta" value="0.05"/>
      <param name="llsamplerange" value="0.01"/>
      <param name="llsamplestep" value="0.01"/>
      <param name="lasamplerange" value="0.005"/>
      <param name="lasamplestep" value="0.005"/>
    </node>
  </launch>
```

Figura H.1 Parámetros usado para usar *gmapping*.

APÉNDICE I

Implementación de algoritmo de localización Monte Carlo

La odometría es una técnica que permite estimar la posición del robot en un mapa a partir de los datos obtenidos por los sensores. Sin embargo, debido a la acumulación de errores en la pose del robot durante su movimiento, es necesario corregir estos errores para obtener una estimación más precisa de su posición. Para lograr esto, se implementa una transformación entre el marco de referencia del mapa y el marco de referencia de la odometría, utilizando los sensores LIDAR ubicados en la base del robot. La figura I.1 muestra la diferencia en los marcos de referencia cuando se utiliza el método de localización basado en la odometría sola y cuando se combina con el método de localización de monte Carlo.

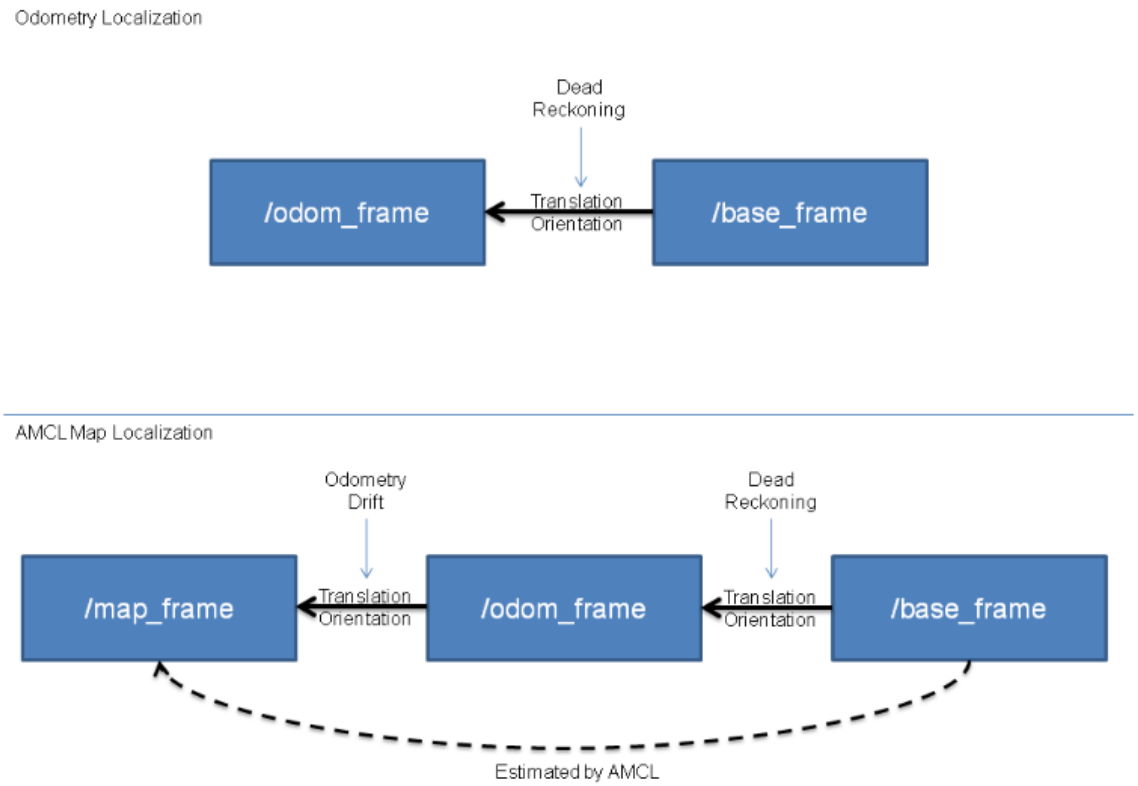

Figura I.1 Métodos de localización de odometría y AMCL.

Para la implementación el algoritmo de Monte Carlo es necesario realizar los siguientes puntos y argumentos de la Figura I.2:

- Macro de referencia del sensor que envía la señal del escaneo
- Definir marco de referencia de la odometría y la base de la plataforma
- Dar una estimación de la pose inicial
- Configurar parámetros relevantes

```
<launch>
    <!-- Arguments -->
    <arg name="scan_topic"     default="scan"/>
    <arg name="initial_pose_x" default="0.0"/>
    <arg name="initial_pose_y" default="0.0"/>
    <arg name="initial_pose_a" default="0.0"/>

    <!-- AMCL -->
    <node pkg="amcl" type="amcl" name="amcl">

        <param name="min_particles"            value="500"/>
        <param name="max_particles"            value="3000"/>
        <param name="kld_err"                  value="0.02"/>
        <param name="update_min_d"             value="0.20"/>
        <param name="update_min_a"             value="0.20"/>
        <param name="resample_interval"        value="1"/>
        <param name="transform_tolerance"      value="0.1"/>
        <param name="recovery_alpha_slow"      value="0.00"/>
        <param name="recovery_alpha_fast"      value="0.00"/>
        <param name="initial_pose_x"           value="$(arg initial_pose_x)"/>
        <param name="initial_pose_y"           value="$(arg initial_pose_y)"/>
        <param name="initial_pose_a"           value="$(arg initial_pose_a)"/>
        <param name="gui_publish_rate"         value="50.0"/>

        <remap from="scan"                     to="$(arg scan_topic)"/>
        <param name="laser_max_range"          value="-1"/>
        <param name="laser_max_beams"          value="180"/>
        <param name="laser_z_hit"              value="0.5"/>
        <param name="laser_z_short"            value="0.05"/>
        <param name="laser_z_max"              value="0.05"/>
        <param name="laser_z_rand"             value="0.5"/>
        <param name="laser_sigma_hit"          value="0.2"/>
        <param name="laser_lambda_short"       value="0.1"/>
        <param name="laser_likelihood_max_dist" value="2.0"/>
        <param name="laser_model_type"         value="likelihood_field"/>

        <param name="odom_model_type"          value="omni-corrected"/>
        <param name="odom_alpha1"              value="0.1"/>
        <param name="odom_alpha2"              value="0.1"/>
        <param name="odom_alpha3"              value="0.1"/>
        <param name="odom_alpha4"              value="0.1"/>
        <param name="odom_frame_id"            value="odom"/>
        <param name="base_frame_id"            value="base_link"/>

    </node>
</launch>
```

Figura I.2 Archivo de ejecución del paquete AMCL.

APÉNDICE J

Implementación de paquete de navegación

Como se puede observar en la Figura J.1 el *stack* de navegación genera cinco nodos principales los cuales se suscriben a tópicos para poder funcionar de forma correcta. Se compuso el paquete madar_navegation, el cual implementa el *stack* de navegación de ROS, requiere de la siguiente información para poder operar:

- Información de mapa (mapa creado por *gmapping*)
- Información de sensores (*scan* de LIDAR)
- Transformación de marco del mapa a odometría (método de Monte Carlo)
- Transformación del marco de odometría a la base del robot (*ground truth*)

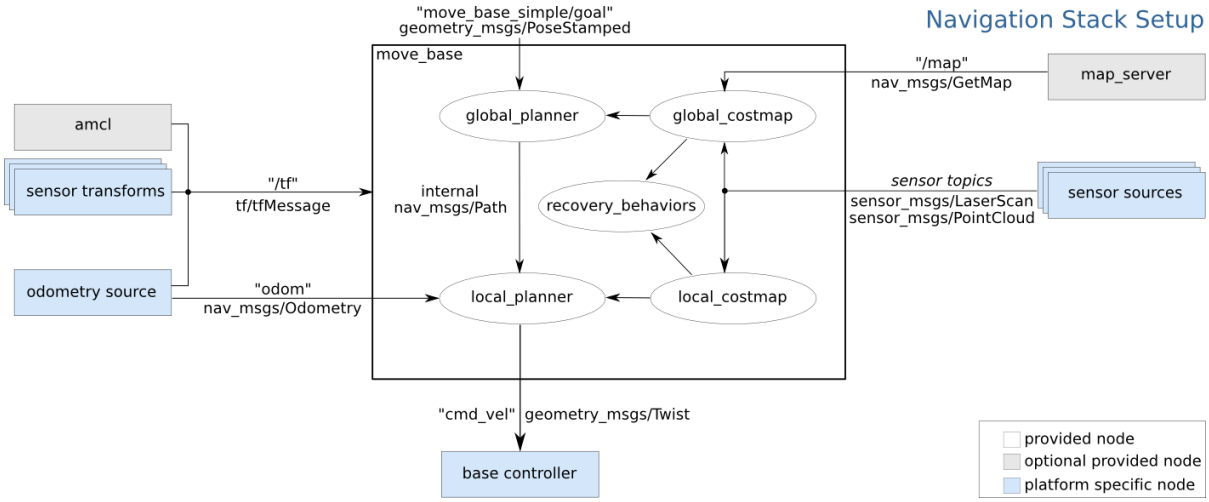

Figura J.1 Estructura principal de paquete de navegación.

En el paquete *madar_navegation* se implementan algoritmos para la planificación de rutas, también, dentro de este paquete se establece un mapa de costos globales y locales. De igual forma se define un área que va a ser la que el robot considera al momento de planificar rutas para desplazarse a una pose deseada. Es por eso por lo que, el paquete cuenta con seis archivos YAML (cuyos nombres se muestran en la Figura J.2) los que modifican los valores para generar el algoritmo de navegación y planificación de rutas, para ver el código de los ficheros ver el quinto elemento del Apéndice P.

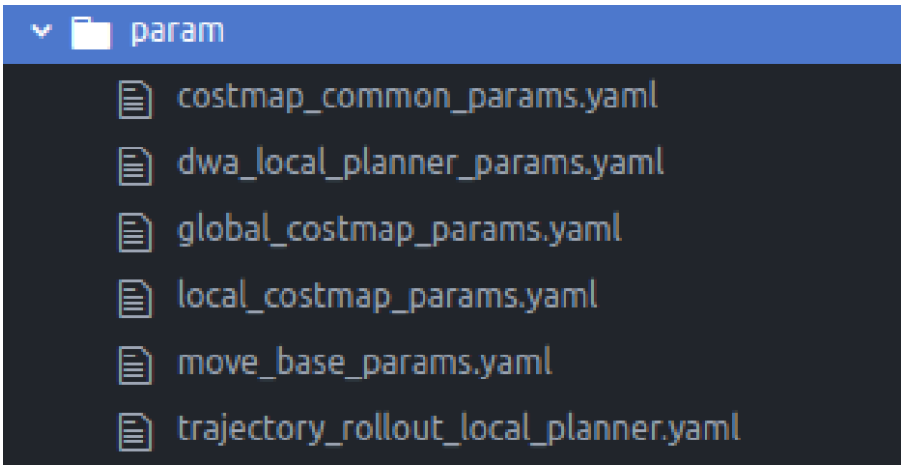

Figura J.2 archivos para implementar navegación.

Para la creación de los nodos en ROS se usa el siguiente archivo *launch* ver Figura J.3. En el archivo se cargan los parámetros de los ficheros YAML. esto es para la creación de los nodos que realizan la planificación del plan y controlan a el robot publicando a él típico de “/cmd_vel”.

```
<launch>
  <!-- Arguments -->
  <arg name="cmd_vel_topic" default="/cmd_vel" />
  <arg name="odom_topic" default="odom" />

  <!-- move_base -->
  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">

    <param name="base_local_planner" value="dwa_local_planner/DWAPlannerROS" />

  <!--load common configuration files -->
    <rosparam file="$(find madar_navigation)/param/costmap_common_params.yaml" command="load" ns="global_costmap" />
    <rosparam file="$(find madar_navigation)/param/costmap_common_params.yaml" command="load" ns="local_costmap" />

  <!--load local and global specific parameters -->
    <rosparam file="$(find madar_navigation)/param/local_costmap_params.yaml" command="load" ns="local_costmap" />
    <rosparam file="$(find madar_navigation)/param/global_costmap_params.yaml" command="load" ns="global_costmap"/>

  <!--load planner-->
    <rosparam file="$(find madar_navigation)/param/base_local_planner_params.yaml" command="load" />
    <rosparam file="$(find madar_navigation)/param/dwa_local_planner_params.yaml" command="load" ns="DWAPlannerROS"/>


    <remap from="cmd_vel" to="$(arg cmd_vel_topic)" />
    <remap from="odom" to="$(arg odom_topic)" />
  </node>
</launch>
```

Figura J.3 Archivos para creación.

APÉNDICE L

Comunicación con los servomotores de la plataforma

La topología de la red EtherCAT es como en la Figura L.1 en la que el maestro es la computadora la cual debe estar enviando información con un periodo de 4 ms en caso de que no se envié en ese tiempo los servomotores se detienen al instante.

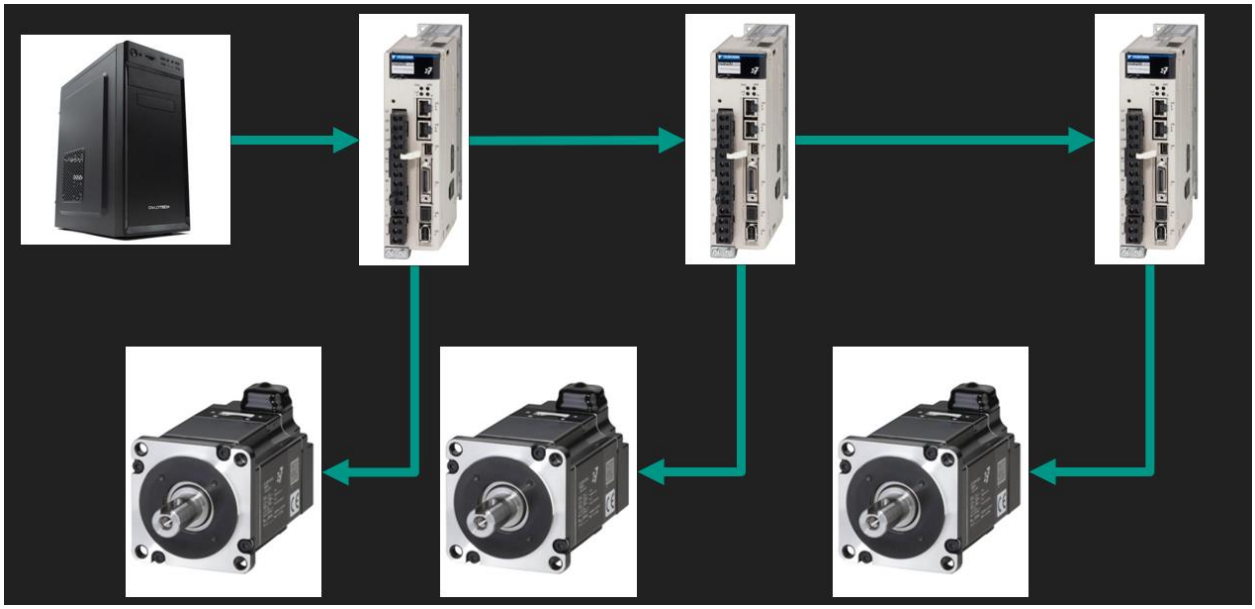

Figura L.1 Gráfico de comunicación de servomotores

Para Conseguir una comunicación con los servomotores se hace uso de la librería SOEM dado que los motores se encuentran conectados en una red EtherCAT. Se tenía desarrollado en el laboratorio la librería cpp4ec encima de la librería de SOEM para poder encapsular la librería SOEM en una clase que se puede usar en C++, esta librería estaba funcionando una interfaz en tiempo real, pero los sistemas operativos actuales que tienen las computadoras del laboratorio actualmente no soportan esa interfaz, con lo que se adecuo el código para que funcionen en las maquinas actuales.

Se uso la librería de cpp4ec para el desarrollo de la librería *platform*, esta se encarga de implementar los cálculos de la cinemática obtenidos en el apéndice E para asignar las velocidades angulares de los ejes a los drivers de la red EtherCAT por medio de las funciones que se crearon en cpp4ec, también, esta se encarga de calcular la odometría leyendo la información de los *encoders* el codigo de la librería se encuentra en el sexto elemento de la tabla en el apéndice P.

APÉNDICE M

Nodo de control para la plataforma real

El Código del ejecutable para el control del MADAR se puede acceder en el séptimo elemento de la tabla del Apéndice P, en este se crea un nodo el que se suscribe a un tópico con el nombre de /cmd_vel, esto hizo para calcular la velocidad que deben tener los motores con el modelo cinemático descrito en el apéndice E.

Como se puede ver en la Figura M.2, el nodo platformnode se ha suscrito al tópico /cmd_vel para controlar las velocidades angulares de los motores a través de la librería platform descrita en el apéndice L. Además, este nodo calcula la odometría de la plataforma y la publica en los tópicos /odom y /tf para que esta información esté disponible para otros nodos.

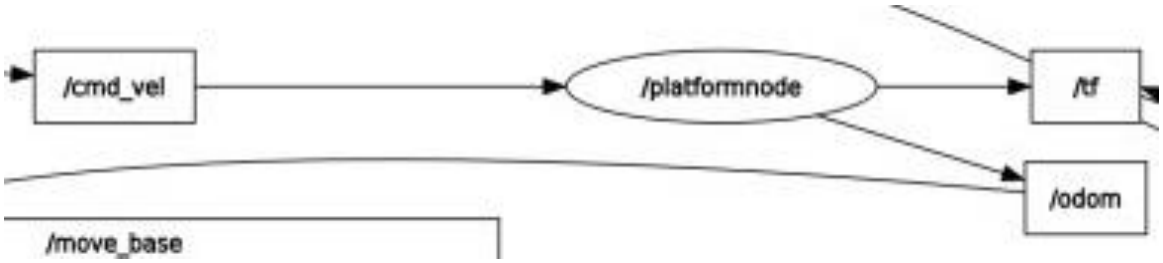

Figura M.2 Comunicación de los nodos para la navegación de la plataforma real.

APÉNDICE N

Código de odometría en plataforma real

Para implementar la odometría primero se requirió obtener la posición actual del motor y la velocidad del motor real.

```
motorPosition = currentMotorPosition * mdegToRad;
motorVelocity = currentMotorVelocity * mdegToRad;
```

Los *encoders* son absolutos, por lo que, se necesita el diferencial de posición de los motores y luego transformarlo a posición de la Plataforma

```
difMotorPosition = motorPosition - lastMotorPosition;
platPosition = (platformMatrix.inverse()) * difMotorPosition;
```

Estimación de la velocidad

```
int p = 100;
motorVelocity=exp(-p*time)*lastMotorVelocity+(1-(exp(-p*time)))*(motorPosition-
lastMotorPosition)/time;
estimatedMotorVelocity = motorVelocity / mdegToRad;
```

La rotación es el único componente que puede calcularse directamente mediante el incremento del codificador.

```
positionPlatform[2] += platPosition[2];
```

Calcular vx, vy, vfi

```
platVelocity = platformMatrix.inverse() * motorVelocity;
```

Calcular vx, vy, vfi al sistema fijo (que está definido por el sistema de plataforma al inicio)

```
velocityPlatform[0] = platVelocity[0] * cos(positionPlatform[2]) - platVelocity[1] *
                      sin(positionPlatform[2]);

velocityPlatform[1] = platVelocity[0] * sin(positionPlatform[2]) + platVelocity[1] *
                      cos(positionPlatform[2]);

velocityPlatform[2] = platVelocity[2];
```

Cálculo del incremento de posición integrando la velocidad

```
integratedPosition = velocityPlatform * time;
int i = 0;
```

Si el movimiento es holonómico, utiliza la posición de la plataforma. Si no lo es, se integra la velocidad.

```
if( ((std::abs(velocityPlatform[2])) > epsilon) && (atan(velocityPlatform[1]/velocityPlatform[0]) !=
velocityPlatform[2]) )
    {
```

integra

```
            positionPlatform[0] += integratedPosition[0];
            positionPlatform[1] += integratedPosition[1];
            i = 1;
        }else{
```

holonómico

```
            positionPlatform[0] += platPosition[0];
            positionPlatform[1] += platPosition[1];
            i = 2;
        }
```

Guardar la posición actual

```
        lastMotorPosition = motorPosition;
        lastMotorVelocity = motorVelocity;
        lastSampleTime = sampleTime[0];
    }
```

APÉNDICE O

Lista de videos

En esta sección se describen los videos de todas las pruebas realizadas durante el desarrollo del proyecto:

1. Primera prueba de simulación dinámica de la plataforma en gazebo: https://drive.google.com/file/d/1pAsr4b5rrSXc5WRuRSXn4XN9_gSE7-6-/view?usp=sharing
2. Control de plataforma en gazebo con brazos y manos: https://drive.google.com/file/d/1HokbdKK9X8uuwwUmcGNzd4c7AV5CG5VP/view?usp=sharing
3. Control de brazos y manos en simulador: https://drive.google.com/file/d/1kK8jGCRg09j3FbmmiQPLyiqdzkx37eYb/view?usp=sharing
4. Navegación autónoma del MADAR en simulación de gazebo https://drive.google.com/file/d/1TxNu2sifElukXYB2rKzyUb9HNW42_f9d/view?usp=sharing
5. control del movimiento de la plataforma real con control logitech f710: https://drive.google.com/file/d/1pldd36eQ9ak_RSBMJJQ0ac2drWMqZZ4F/view?usp=share_link
6. navegación del MADAR en laboratorio de forma dinámica: https://drive.google.com/file/d/19OvCtO6jTYi7E7fIJnX-y9v5o5Plg5mN/view?usp=sharing

APÉNDICE P

Lista de Repositorios

1	Archivo XACRO de la plataforma	https://gitioc.upc.edu/robots/madar_description/-/blob/gazebo_implementation/urdf/platform.xacro
2	Archivo XACRO del MADAR	https://gitioc.upc.edu/robots/madar_description/-/blob/gazebo_implementation/urdf/madar_robot.xacro
3	Launch de la simulación dinamica del MADAR	https://gitioc.upc.edu/platform/bmmx/-/blob/main/platform_sim/launch/madar.launch
4	Código para el control de motores en gazebo	https://gitioc.upc.edu/platform/bmmx/-/blob/main/platform_sim/src/platform_gazebo.cpp
5	Parámetros para stack de navegación	https://gitioc.upc.edu/platform/bmmx/-/tree/main/madar_navigation/param
6	Librería platform	https://gitioc.upc.edu/platform/platform
7	control de los motores del MADAR	https://gitioc.upc.edu/platform/bmmx/-/blob/main/bmmx/src/platform_ros.cpp