UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

SISTEMA DE APOYO VISUAL PARA LOCALIZACIÓN Y NAVEGACIÓN EN ROBOT DE SERVICIO USANDO CÁMARAS IP

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN ELÉCTRICA

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO

GIOVANNI MATÍAS PAIS LUCERO

PROFESOR GUÍA:
JAVIER RUIZ DEL SOLAR SAN MARTÍN

MIEMBROS DE LA COMISIÓN:
CESAR AZURDIA MEZA
PATRICIO LONCOMILLA ZAMBRANA

Este trabajo ha sido parcialmente financiado por la ANID mediante la beca
CONICYT-PFCHA/Magíster Nacional/2020 - 22201763

SANTIAGO DE CHILE
2024

Resumen de la tesis para optar al grado de
Magíster en Ciencias de la Ingeniería, mención Eléctrica
Resumen de la memoria para optar al título de
Ingeniero Civil Eléctrico
Por: Giovanni Matías Pais Lucero
Fecha: 2024
Profesor Guía: Javier Ruiz del Solar

Resumen

Esta tesis propone un enfoque para mejorar la navegación de un robot móvil mediante un sistema de apoyo visual. Dado que la navegación del robot depende de la cantidad y calidad de información de sus sensores, se propone aprovechar la información disponible en la red de cámaras de seguridad del entorno del robot. El sistema propuesto se conecta a las cámaras para detectar la posición del robot y obstáculos usando aprendizaje profundo y regresiones, integrando esta nueva información al sistema de localización y navegación en tiempo real. Además, se implementa un sistema de ajuste automático basado en metodología profesor-estudiante para mejorar la precisión de las cámaras fijas con mínima calibración humana. El trabajo se divide en varias etapas, incluyendo implementación, validación en ambientes simulados y reales, y evaluación de modelos para cada componente del sistema. Finalmente, se evalúa el funcionamiento del sistema en el robot real para comprobar que el sistema es capaz de ejecutarse en su CPU en tiempo real, demostrando que el sistema de apoyo visual logra aportar información que mejora la localización y navegación en algunos de los escenarios planteados.

Dedicado a mi yo del futuro y mi querida familia. Mamá, Papá y Hermano.

Agradecimientos

Quiero partir agradeciendo a mi profesor guía y mentor por muchos años, Javier Ruiz del Solar. Agradecer su apoyo es este proceso final de la carrera y durante la mayor parte de esta, por muchas muchas enseñanzas como su estudiante, su ayudante en algunos cursos y como miembro del laboratorio de robótica por años. Aprendí muchas cosas de él siendo clave en mi desarrollo profesional y personal, aprendiendo conocimiento teórico, conocimiento aplicado, conocimiento de gestión y liderazgo. Agradecer también las oportunidades y confianza depositada en mí para distintos proyectos.

Quiero agradecer a mi familia, papá, mamá y mi hermano por darme siempre el apoyo para entrar en la carrera, mantenerme y terminarla. Siempre han sido mi apoyo en mis maratones interminables de trabajo y estudio, mis momentos difíciles y mi guía cuando no veo las cosas claras.

Agradecer a mis amigos durante la universidad Manu, Bitar, Ale, Javi, Andy, Agus por muchas risas y buenos momentos sumados a noches de estudio y apoyo. Especialmente a mi gran amigo Nico Cárdenas, por ser un hermano para mi, recibirme en su departamento y pasar muchas muchas horas estudiando y haciendo trabajos en su departamento. Muchas noches de pizza y cerveza también. Agradecer al laboratorio de robótica y su gente, agradecer a Nico Marticorena, Cristopher, Lukas, Peluka, Ulises, Luz, Mati, Rodrigo, Gonzalo y al resto del equipo. Aprendí muchas cosas en el laboratorio, cree muy buenos amigos y muy buenos momentos con maratones de pega, el esperanto, eventos y viajes. Agradecimiento especial a Bender y Pepper.

Agradecer a mi amigo, socio y hermano Juan Pablo Cáceres, por el apoyo constante en el fin de la carrera, últimos ramos y la tesis. Por largas noches y fines de semana de tesis y buena conversa.

Agradecer a personas que fueron claves en este proceso y ahora están lejos por distintas razones.

Agradecer a mis amigos del colegio y mis amigos de la U, con quienes he compartido en estos años y me han apoyado de distintas maneras. Por aguantar mis desapariciones por semanas y meses por estar trabajando en algo.

Finalmente, agradezco el financiamiento otorgado por la Agencia Nacional de Investigación y Desarrollo (ANID) mediante la beca CONICYT-PFCHA/Magíster Nacional/2020 - 22201763

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Lista de acrónimos

- **CCTV:** Closed-Circuit Television. Televisión de Circuito Cerrado.
- **YOLO:** You Only Look Once. Sólo Miras una Vez.
- **ROS:** Robot Operating System. Sistema Operativo para Robots.
- **RTSP:** Real-Time Streaming Protocol. Protocolo de Transmisión en Tiempo Real.
- **CPU:** Central Processing Unit. Unidad Central de Procesamiento.
- **LiDAR:** Light Detection and Ranging. Detección y Rango por Luz.
- **IMU:** Inertial Measurement Unit. Unidad de Medición Inercial.
- **SLAM:** Simultaneous Localization and Mapping. Localización y Mapeo Simultáneos.
- **RViz:** ROS Visualization. Visualización en ROS.
- **IA:** Artificial Intelligence. Inteligencia Artificial.
- **RPN:** Region Proposal Network. Red de Propuestas de Regiones.
- **SSD:** Single Shot MultiBox Detector. Detector de Multibox de Tiro Único.
- **NAS:** Neural Architecture Search. Búsqueda de Arquitectura Neuronal.
- **IOU:** Intersection over Union. Intersección sobre Unión.
- **RAM:** Random Access Memory. Memoria de Acceso Aleatorio.
- **OS:** Operating System. Sistema Operativo.
- **DVR:** Digital Video Recorder. Grabador de Video Digital.
- **NVR:** Network Video Recorder. Grabador de Video en Red.
- **SGD:** Stochastic Gradient Descent. Descenso del Gradiente Estocástico.
- **ADAM:** Adaptive Moment Estimation. Estimación Adaptativa de Momentos.
- **FPS:** Frames per Second. Fotogramas por Segundo.
- **mAP:** Mean Average Precision. Precisión Media Promedio.
- **SVR:** Support Vector Regression. Regresión de Vectores de Soporte.
- **MLP:** Multi-Layer Perceptron. Perceptrón Multicapa.
- **SR:** Success Rate. Tasa de Éxito.

Capítulo 1

Introducción

1.1. Motivación

La robótica es un campo de investigación de gran potencial y crecimiento, con una variedad de robots diseñados para asistir a los seres humanos en tareas industriales y domésticas. Los robots domésticos, en particular, deben ser eficientes en sus labores y accesibles económicamente para que puedan integrarse ampliamente en la vida diaria. Sin embargo, la navegación de robots presenta múltiples desafíos, incluyendo la necesidad de que los robots comprendan su entorno, planifiquen trayectorias y eviten colisiones con obstáculos, tanto estáticos como móviles.

Esta tarea presenta distintos desafíos, tales como que el robot sea capaz de entender su entorno para ubicarse en este, armar un plan de trayectoria para llegar a sus objetivos evitando colisiones con obstáculos y, adicionalmente, desplazarse en ambientes domésticos con personas, sumando la dificultad de tener obstáculos en movimiento constantemente. Para afrontar los desafíos del problema de navegación, se disponen de distintas metodologías y algoritmos, pero todos se basan en la información que el robot es capaz de recolectar a través de sus propios sensores. Un problema de esto es que los sensores con mejor desempeño y aporte a la navegación tienden a ser costosos, por lo que existen distintos modelos de robots que incorporan una menor cantidad de sensores o sensores de menores prestaciones. Por otro lado, un robot por lo general solo tiene información de su entorno cercano, limitado a lo que se encuentra en el alcance de los sensores.

Para mejorar la navegación de los robots sin incurrir en altos costos, es crucial explorar el uso de los recursos disponibles en el entorno. Actualmente, muchos edificios están equipados con cámaras de seguridad que capturan continuamente información valiosa del entorno. Esta tesis propone utilizar estas cámaras, junto con técnicas de procesamiento de imágenes y aprendizaje profundo, para complementar la percepción de los robots y mejorar su capacidad de navegación. La integración de datos de cámaras de seguridad puede proporcionar a los robots información adicional sobre obstáculos y objetivos, incrementando su capacidad para desplazarse de manera segura y eficiente.

Trabajos previos han explorado el uso de cámaras para la generación de información

utilizada en la localización, como es el caso de los trabajos de ORB-SLAM [1][2][3], donde se emplean cámaras montadas en el robot para reconocer puntos de referencia que este utiliza para ubicarse. Por otro lado, el trabajo de Xu et al. [4] aborda la detección de distintos obstáculos, incluyendo personas, usando cámaras RGB-D y modelos de aprendizaje profundo, integrando esta información al sistema de navegación del robot. Singh et al. [5] plantean la detección de información visual por parte de un enjambre de robots que comparten dicha información entre ellos para mejorar la posición de cada uno. A partir de estos trabajos surge la idea de aprovechar las cámaras de vigilancia de los edificios para generar información visual del robot, como si fueran otro agente del enjambre, y así mejorar el desempeño en la tarea de navegación. Para esto, se revisaron trabajos donde se usa aprendizaje profundo en cámaras de seguridad. Ahmed et al. [6] emplean cámaras cenitales para detectar objetos y comportamientos, complementando el trabajo de los robots y del personal de seguridad. Por otro lado, Bultmann et al. [7] han explorado la mejora de la localización del robot utilizando cámaras CCTV (Circuito Cerrado de Televisión) del edificio para detectar al robot y estimar su posición, integrando esta información con la de los sensores. Estos trabajos utilizan distintos modelos de aprendizaje profundo para la detección de objetos y emplean computación externa para este propósito. Esta tesis propone el uso de modelos livianos del estado del arte, como YOLO [8][9][10][11][12], que puedan ejecutarse directamente dentro del robot y conectarse a las cámaras de CCTV para detectar al robot, obstáculos y objetivos que apoyen la tarea de navegación.

Adicionalmente, en los trabajos previos se observa la necesidad de una gran cantidad de horas humanas en etiquetado y calibración de modelos, o el uso de datos sintéticos para mitigar esto. En esta tesis se plantea el uso de un entrenamiento con método profesor-estudiante [13][14], donde un modelo grande con alta precisión transfiere conocimiento a un modelo liviano capaz de ejecutarse en el robot y que aprovecha que las cámaras tienen una vista controlada.

El aporte de esta solución incluye una mejora en la tarea de navegación del robot sin costos adicionales en hardware, apoyando la localización, planificación global y detección de objetivos. Todo esto con un sistema que requiere de una baja cantidad de horas humanas para su ajuste. Sin embargo, se anticipan desafíos técnicos, como la integración de datos de múltiples cámaras y la necesidad de un procesamiento eficiente de imágenes. Esta investigación abordará estos desafíos mediante el desarrollo y la validación de nuevos algoritmos y modelos.

1.2. Hipótesis

Se plantea como hipótesis que es posible desarrollar un sistema de apoyo visual basado en aprendizaje profundo que permita a un robot mejorar su localización en el espacio y detectar la posición de obstáculos que faciliten su navegación. Este sistema debe aprovechar la información visual de cámaras IP disponibles dentro de la red, utilizar un modelo de aprendizaje profundo que se ejecute en tiempo real dentro del robot e integrar la nueva información en el sistema de localización y navegación del robot.

1.3. Objetivo general

Desarrollar e implementar un sistema de apoyo visual basado en aprendizaje profundo para mejorar la localización y navegación de un robot de servicio usando cámaras IP.

1.4. Objetivos específicos

1. Implementar un nodo de procesamiento de imágenes en ROS (Robot Operating System) basado en modelos de aprendizaje profundo que se conecte a cámaras WiFi vía protocolo RTSP (Real Time Streaming Protocol).
2. Evaluar y implementar un modelo de detección que permita su ejecución eficiente en la CPU del robot. La ejecución del modelo no debe interferir con las otras funciones del robot.
3. Implementar un algoritmo de calibración y ajuste fino para cámaras fijas, aprovechando que gran parte de los píxeles y el contexto de una cámara fija no cambia. Se plantea una metodología Profesor-Estudiante (Teacher-Student) entrenado en diversos entornos operativos, optimizando así el rendimiento en vistas fijas.
4. Integrar la información de detección del robot y obstáculos como apoyo del sistema de localización y de navegación del robot.
5. Evaluar y analizar el desempeño de los modelos de detección, calibración y el sistema de apoyo visual completo realizando pruebas de navegación del robot en ambientes simulados y reales.

1.5. Estructura de la tesis

El presente documento se estructura como sigue:

- En el Capítulo 2 se presentan los antecedentes necesarios para la comprensión del trabajo expuesto en este documento. Se realiza una revisión de distintos temas como: conceptos de robótica y procesos asociados, aprendizaje de máquinas, visión computacional y modelos de aprendizaje profundos asociados.
- En el Capítulo 3 se presenta la metodología del trabajo realizado, se presentan las decisiones tomadas para la implementación, herramientas de software y equipamiento a utilizar, ambientes de prueba, interacciones entre los distintos nodos y ecuaciones utilizadas.
- En el Capítulo 4 se presentan los resultados de las pruebas realizadas, se describen estas y se analizan los resultados obtenidos vinculándolos con las hipótesis del trabajo.
- Finalmente, en el Capítulo 5 se presentan las conclusiones del trabajo realizado, cumplimiento de la hipótesis y objetivos, y algunos nuevos planteamientos pensados en la extensión del trabajo a futuro.

Capítulo 2

Antecedentes

2.1. Robótica móvil

Consiste en un área de la robótica asociada a los robots que deben desplazarse para cumplir con sus labores, en general por un terreno delimitado. Estos robots a diferencia de robots de manufactura que están fijos, tienen ruedas o extremidades que les permiten desplazarse cambiando su ubicación. Dado lo anterior estos robots también cuentan con capacidades para localizarse en un espacio respecto a un mapa usando sus sensores y cuentan con habilidades para navegar por estos espacios, definiendo trayectorias que pueden modificarse en caso de que hayan cambios en el entorno. La robótica móvil combina la información de distintos sensores y métodos para determinar la posición del robot en el espacio y calcular las rutas de desplazamiento para alcanzar un objetivo.

La robótica de servicio doméstica comprende un área de la robótica más cercana a las tareas de las personas en su día a día, estos robots fueron ideados inicialmente para ayudar a las familias en sus casas. Estos robots, a diferencia de los robots industriales, deben interactuar regularmente con seres humanos por lo que tienen habilidades sociales. En el caso de los robots de servicio móviles, estos deben desplazarse en ambientes con una gran cantidad de "obstáculos"móviles como pueden ser las personas u objetos que las personas mueven. El trabajar en ambientes con personas propone nuevos desafíos para el método en que los robots realizan la navegación por el espacio y mueven sus actuadores para realizar sus tareas. Este trabajo de tesis se centra en la robótica de servicio móvil, por lo que se abordan desafíos particulares de esta área.

2.1.1. Arquitectura de control de un robot móvil

El proceso de desplazamiento de un robot móvil considera distintos componentes que interactúan entre sí, compartiendo información y actualizando indicadores del proceso para la lograr llegar al objetivo.

En la Figura 2.1 se muestra un diagrama de la interacción entre los distintos componentes

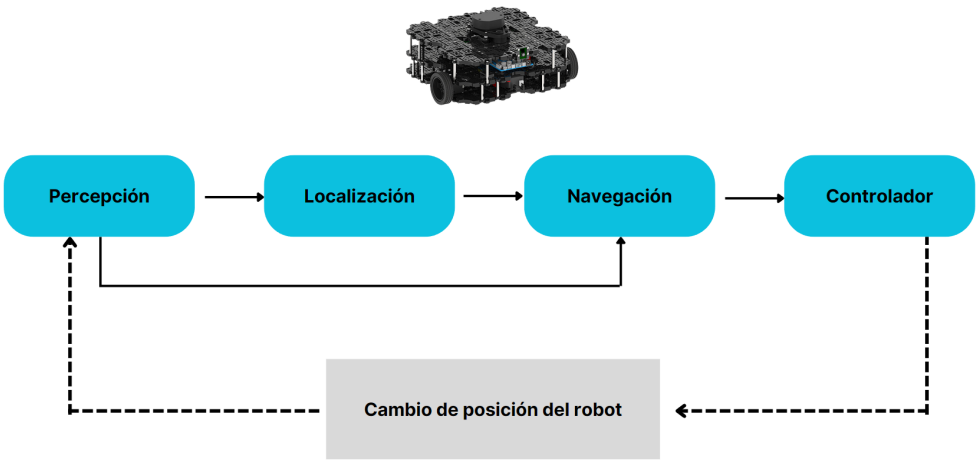

Figura 2.1: Diagrama flujo de proceso navegación en robótica móvil

del proceso de desplazamiento.

- **Percepción**: Corresponde a los sensores que recopilan datos del entorno del robot para ser usados en los algoritmos asociados al proceso. Dentro de los sensores más comunes se encuentran los LiDAR, cámaras, sensores de proximidad, IMU (Unidad de Medición Inercial), entre otros.
- **Localización**: Corresponde a la componente de software encargada de recibir la información de los sensores y, a través de algoritmos, determinar la posición del robot respecto al mapa.
- **Navegación**: Corresponde a la componente de software encargada de la tarea de desplazamiento como tal. Esta debe definir la trayectoria del robot dentro del mapa para llegar al objetivo sin colisionar con obstáculos. Este módulo utiliza la información del mapa y la información de localización para definir la trayectoria y utiliza la información de los sensores para ir corrigiendo la trayectoria para evitar obstáculos.
- **Controlador**: Corresponde a la componente que entrega los comandos de movimiento a los motores de ruedas o extremidades que permiten los movimientos del robot. Los comandos entregados provienen de las decisiones tomadas en tiempo real por el módulo de navegación y permiten la conexión del software con el hardware.

El proceso completo descrito en el diagrama de la Figura 2.1 muestra cómo el robot obtiene información de sus sensores para que el módulo de localización determine la posición del robot respecto al mapa. Tomando la información de los sensores y del módulo de localización, junto con un objetivo de navegación, el módulo de navegación genera una trayectoria global y local con el fin de evitar obstáculos. Finalmente, recibiendo la información del módulo de navegación, los controladores envían comandos a los motores, generando el desplazamiento del robot. Este proceso genera el movimiento del robot y, al realizarse, se repite para actualizar la nueva posición del robot. Este ciclo ocurre varias veces por segundo.

2.1.2. Localización

El proceso de localiación busca determinar la posición del robot respecto a un sistema de referencia, en general, asociado a un mapa previamente designado y memorizado por este. Dentro de la robótica móvil es necesario tener un mapa asociado al entorno de trabajo del robot, este mapa puede cargarse de distintas maneras pero lo más común es construir el mapa usando los sensores del robot dado que gracias a los sensores el robot estimará su posición. Existen casos en los cuales el robot es capaz de realizar el "mapeo" del ambiente y su localización al mismo tiempo, esto se conoce como SLAM (Simultaneous localization and mapping) [15] pero este caso no se considerará dentro del trabajo de tesis. Se tomará como punto de partida que el robot ya posee un modelo preciso de mapa asociado a los ambientes de pruebas. Dentro de la tarea de navegación del robot es necesario que el robot conozca su posición respecto al mapa para poder planificar sus trayectorias para objetivos y cumplir con sus tareas. La tarea de localización del robot se realiza a través de mediciones de los sensores en conjunto con algoritmos a través de los cuales se detectan puntos de referencia o landmarks. Al detectar estos puntos de interés junto a una posición relativa al robot y dado que tienen una posición dada en el mapa, el robot usa algoritmos para manejar la información de estos landmarks con el fin de estimar su posición y la corrige a medida que se desplaza. La integración de las detecciones de landmarks junto la posición anterior del robot y su desplazamiento se maneja usando distintos algoritmos dado que los sensores no tienen mediciones 100 % precisas y poseen ruido, por lo que se debe manejar la información cuidadosamente. Algunos algoritmos comúnmente usados en estas tareas son el filtro de Kalman y el filtro de partículas, en esta tesis se trabaja con métodos de localización basados en filtros de partículas.

2.1.3. Filtro de partículas

El filtro de partículas[16] es un método de estimación utilizado en robótica móvil para encontrar la posición y orientación de un robot en un ambiente. Es un método útil en casos de gran incertidumbre y donde el comportamiento de sensores y movimiento son no lineales.

El filtro de partículas modela una distribución probabilística de la posición del robot a través de distintas muestras o partículas. Estas partículas corresponden a vectores que codifican una posible posición y orientación del robot. Este método logra modelar distribuciones de probabilidad complejas por lo que es útil para distintos problemas de búsquedas de distribuciones, donde un mayor número de partículas permite tener una mejor representación de la función de distribución, pero a su vez requiere una mayor carga computacional por lo que es necesario encontrar un equilibrio para cada caso. La teoría plantea que si se tuvieran infinitas partículas se tendría una representación de alta exactitud acercándose a la distribución real.

El proceso del filtro de partículas se puede describir en las siguientes etapas:

1. **Inicialización:** Se generan N partículas distribuidas aleatoriamente en el espacio de estados, donde cada partícula representa una posible posición y orientación del robot (x_i, y_i, θ_i). Cada partícula se pondera con un peso inicial, que generalmente es igual para todas las partículas.

2. **Predicción (Movimiento):** Las partículas se actualizan basándose en el modelo de movimiento del robot. Si el robot se mueve, la posición de cada partícula se ajusta según el movimiento estimado, añadiendo ruido para simular la incertidumbre:

$$x_i' = x_i + \Delta x + \varepsilon_x \tag{2.1}$$
$$y_i' = y_i + \Delta y + \varepsilon_y \tag{2.2}$$
$$\theta_i' = \theta_i + \Delta\theta + \varepsilon_\theta \tag{2.3}$$

   Donde ε_x, ε_y y ε_θ son términos de ruido que representan la incertidumbre del movimiento.

3. **Actualización:** Las partículas se reponderan en función de cuán bien predicen las observaciones de los sensores al recibir nuevas mediciones u observaciones. Si una partícula predice bien la posición del robot, su peso w_i aumenta:

$$w_i' = w_i \cdot p(z|x_i') \tag{2.4}$$

   Donde $p(z|x_i')$ es la probabilidad de observar z dado el estado de la partícula x_i'.
   Al final de esta etapa se normalizan los pesos para que la suma total sea igual a 1.

4. **Re muestreo:** Se realiza un re muestreo de las partículas basado en sus pesos. Las partículas con pesos mayores tienen una mayor probabilidad de ser seleccionadas, mientras que las partículas con pesos bajos pueden ser descartadas. Esto permite concentrar las partículas en las regiones del espacio de estados más probables.

5. **Estimación:** La estimación de la posición y orientación del robot se calcula como la media ponderada de las partículas:

$$\hat{x} = \sum_{i=1}^{N} w_i x_i \tag{2.5}$$
$$\hat{y} = \sum_{i=1}^{N} w_i y_i \tag{2.6}$$
$$\hat{\theta} = \sum_{i=1}^{N} w_i \theta_i \tag{2.7}$$

El filtro de partículas en el caso de uso de este trabajo de tesis para estimación de la posición del robot, considera la esperanza de la distribución de las partículas como la posición que el algoritmo entrega al robot. Al inicio del funcionamiento se generar partículas con poses aleatorias en el espacio y a través de la detección de puntos de referencia se van modificando manteniendo las partículas con mayor probabilidad y renovando partículas que tienen menor probabilidad. Esto permite que a medida que el robot obtiene mediciones de detecciones de puntos de referencia las partículas se concentren en torno a la posición real del robot y disminuyendo la incertidumbre de su posición.

Para este trabajo se utiliza particularmente el algoritmo AMCL (Adaptive Monte Carlo Localization), el cuál es una variante del filtro de partículas comúnmente utilizada en robótica

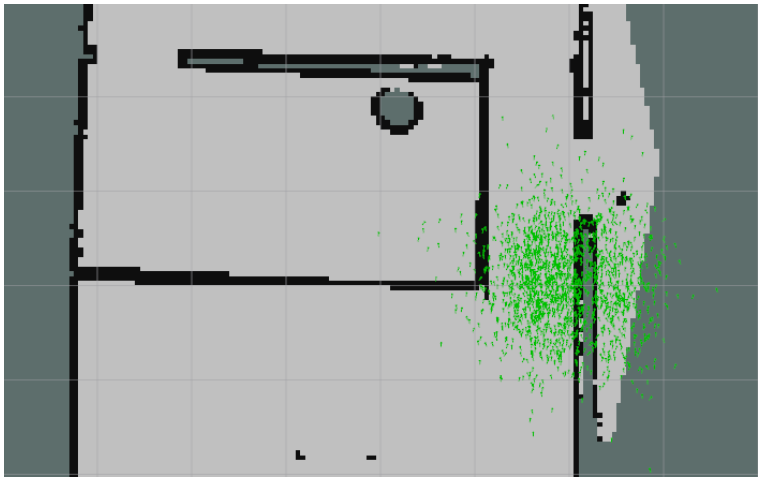

Figura 2.2: Partículas al inicio de la exploración.

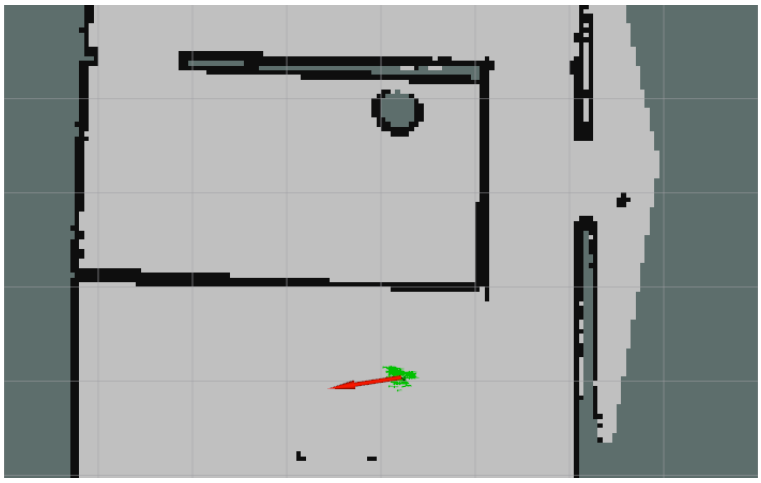

Figura 2.3: Partículas al avanzar en la exploración.

móvil. Este algoritmo es utilizado por defecto en el framework ROS dado su desempeño en entornos dinámicos y que su adaptabilidad en la cantidad de partículas permite ser más eficiente en costos computacionales. Una diferencia de este algoritmo respecto a la explicación previa del filtro de partículas es que adapta la cantidad de partículas dependiendo de la incertidumbre de la estimación. Esto se traduce agregando un paso 4.5 dentro de los pasos antes explicados, donde se adapta la cantidad total de partículas.

En las Figuras 2.2 y 2.3 se observa como cambia la distribución de las partículas a medida que pasa el tiempo que el robot explora su entorno. En la Figura 2.2 se observa como las partículas en verde están más dispersas y acumuladas en un sector dada las primeras mediciones del robot iniciar el algoritmo. Posterior a enviar al robot a explorar el lugar designado con la flecha roja en la Figura 2.3 se observa que las partículas se concentran en el punto objetivo dado que al sumar nuevas mediciones de puntos de referencia, en este caso muros y muebles, mejora la posición.

2.1.4. Navegación

Consiste en la tarea de desplazarse por el espacio designado para el robot o mapa. La tarea de navegación considera la generación del plan de trayectoria, la interacción con los actuadores para desplazarse y la actualización de trayectoria dependiendo de los cambios de estado del robot respecto al entorno. Los cambios en el entorno, por ejemplo obstáculos, deben ser detectados en tiempo real a través de los distintos sensores que poseen los robots. En la Figura 2.4 se observa como el robot designa la trayectoria para llegar un objetivo.

Planificación de trayectoria

La planificación de la trayectoria es esencial para la navegación de robots móviles. El proceso se inicia con la creación de un mapa de costos, el cual asigna valores de dificultad a distintas áreas del entorno, incluyendo obstáculos fijos y móviles. Este mapa de costos ayuda a determinar las rutas más eficientes y seguras para el robot.

Se utilizan algoritmos de planificación de trayectoria, como A* (A-star)[17] y D* Lite [18], para calcular la mejor ruta desde la posición actual del robot hasta su objetivo. Estos algoritmos consideran el mapa de costos para evitar obstáculos y minimizar la distancia y el tiempo de viaje.

Evasión de Obstáculos

Durante la navegación, el robot debe ser capaz de detectar y evadir obstáculos en tiempo real. Los sensores, como LiDAR, cámaras y ultrasonidos, proporcionan datos continuos sobre el entorno. Si se detecta un obstáculo en la trayectoria planificada, el robot utiliza estos datos para ejecutar una replanificación inmediata.

La replanificación puede implicar ajustar ligeramente la ruta para rodear el obstáculo o, en casos más complejos, recalcular una nueva trayectoria desde cero. Este proceso es crítico para asegurar que el robot pueda navegar de manera segura y eficiente, especialmente en entornos dinámicos donde los obstáculos pueden moverse.

Para estimar esta trayectoria el robot utiliza un mapa de costos construido en función de los obstáculos detectados y el modelo del mapa previamente cargado en el robot. El mapa de costos codifica las posiciones o puntos del mapa, representado por píxeles, con un valor entre 0 y 255 que representa la probabilidad de colisión en esos puntos. Esto es utilizado por el cálculo de trayectoria para penalizar las rutas con un costo asociado mayor. El mapa de costos está separado en 2 tipos: el mapa de costos global y el mapa de costos local.

Mapa de costos global

El mapa de costos global es aplicado sobre todo el mapa del ambiente del robot, este se construye usando la información de muros y obstáculos del mapa del robot previamente

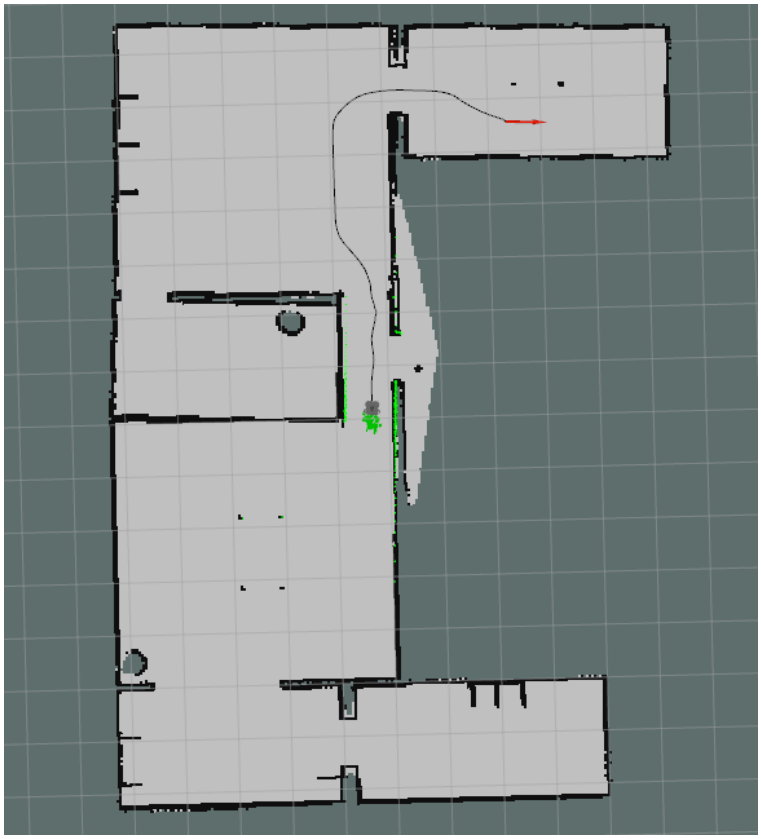

Figura 2.4: Ejemplo de trayectoria generado por sistema de navegación sobre mapa visualizado en RViz (Robot Vizualization) de ROS .

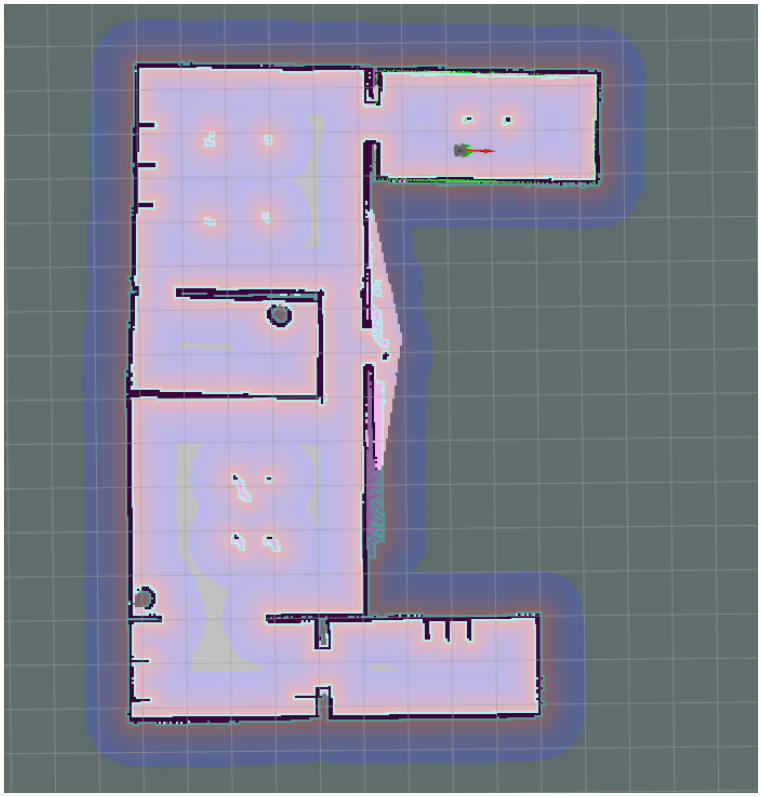

Figura 2.5: Mapa de costos global visualizado sobre mapa en RViz.

cargados, asignando a estos puntos del mapa la identificación posición prohibida u obstáculo `LETHAL` en ciertos frameworks. La asignación de estos obstáculos demarca que es un punto del mapa por el cual el robot no puede navegar dado que una colisión es segura. Alrededor de los puntos de colisión anteriores se genera un mapa de costos en función de la probabilidad de colisión y usando un método llamado inflación que asigna un valor en función de la distancia al obstáculo. En conjunto a la información del mapa original, el robot va completando el mapa de costos en función de la detección de obstáculos a partir de sus sensores como por ejemplo LiDAR o cámaras de profundidad. Al detectar un obstáculo este se marca en el mapa de costos global hasta tener una nueva medición que identifique que ese obstáculo ya no está. Este mapa de costos se mantiene en el tiempo incluso si el robot no está en ese sector del mapa hasta que el robot y sus sensores estén en alcance de corregirlo. Este mapa de costos es la base para el cálculo de trayectorias a un punto final en el mapa.

En la Figura 2.5 se observa el mapa de costos del robot. El mapa de costos se representa a través de una escala de colores en función del valor de cada punto. Se puede observar como la información del mapa marcada en negro tiene un color más rojizo alrededor dado que es una posición prohibida o obstáculo `LETHAL`. Adicionalmente, se pueden ver en marcados en celeste obstáculos que detectó el robot al pasar por ese lugar previamente, indicando que son coordenadas del mapa prohibidas. En este caso, algunos de estos puntos corresponden a las patas de una mesa que no estaba cuando se realizó el mapeo del lugar.

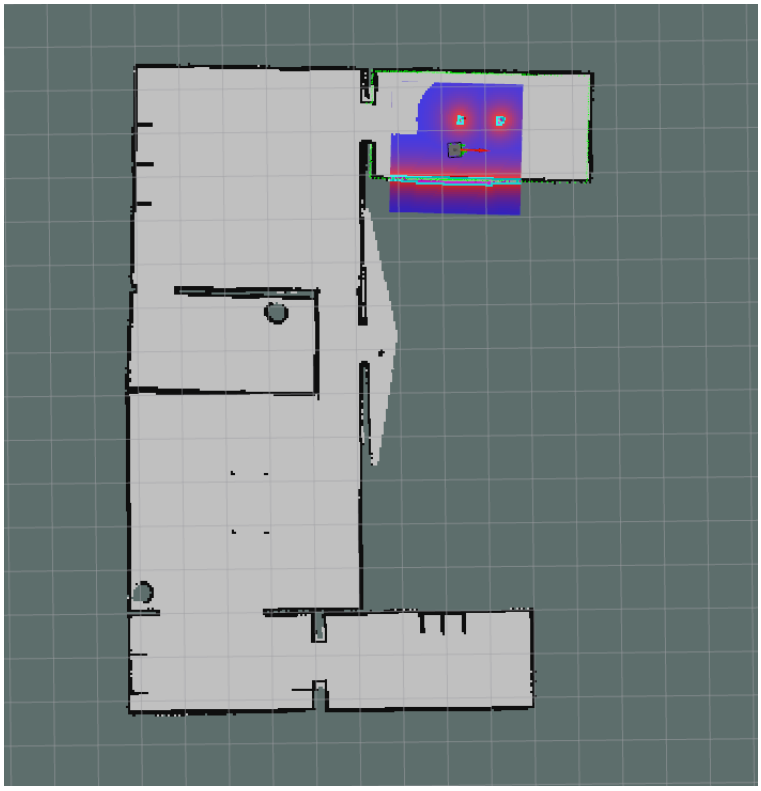

Figura 2.6: Mapa de costos local visualizado sobre mapa en RViz.

Mapa de costos local

El mapa de costos local está asociado a los obstáculos inmediatos y dentro del alcance de los sensores del robot, marcando al igual que el mapa de costos global las zonas de mayor probabilidad de colisión. Este mapa de costos se utiliza para la navegación local para alcanzar el siguiente punto de la trayectoria global previamente calculada y también colabora en la actualización del mapa de costos global. Esta capa del mapa de costos también funciona como una capa de seguridad teniendo una gran prioridad en el desplazamiento del robot en caso de que aparezca un nuevo obstáculo en las cercanías y previo a recalcular la ruta. En la Figura 2.6 se observa el mapa de costos del robot a través de RViz. Ahí se ve que al igual que el mapa de costos global se representa en escala de colores las probabilidades de colisión y en este caso se tiene definido un mapa de costos local de forma cuadrada, la zona del cuadrado que parece faltar corresponde a una zona sin riesgo según la captura de sus sensores.

2.2. Visión por Computador

La visión por computador es una subdisciplina de las ciencias de la computación y la inteligencia artificial que se centra en el análisis e interpretación de la información contenida en imágenes y videos por parte de un computador o máquina. Para ello, se utilizan diversas herramientas, algoritmos y modelos de IA con el fin de capturar, procesar e interpretar imágenes, generando automáticamente información útil para distintas tareas.

La visión por computador ofrece numerosas oportunidades para resolver diversos tipos de problemas, generando valor en múltiples ámbitos. Algunos de los casos de uso más comunes incluyen: en robótica para comprender entornos, en medicina para el análisis de exámenes, en navegación autónoma para segmentar el espacio, en seguridad para detectar delitos y en manufactura para identificar anomalías, entre otros.

En la aplicación a distintos casos de uso, la visión por computador abarca varias tareas, tales como clasificación, detección de objetos, segmentación de objetos y estimación de poses. Para llevar a cabo estas tareas, se emplean algoritmos clásicos basados en filtros y transformaciones, así como algoritmos de aprendizaje automático y profundo.

2.2.1. Aprendizaje de máquinas o Machine Learning

El aprendizaje de máquinas es una subdisciplina de las ciencias de la computación y una rama de la inteligencia artificial que comprende a algoritmos que poseen la capacidad de aprender a realizar tareas en base a datos. Este aprendizaje consiste en la búsqueda de parámetros óptimos dentro de un modelo para cumplir con una tarea. La búsqueda de estos parámetros se realiza a través de un entrenamiento que consiste en utilizar datos para ir ajustando los parámetros a través de la optimización de una función de desempeño o la disminución de una función de error o pérdida. Existen distintos tipos de algoritmos de aprendizaje de máquinas como por ejemplo: las regresiones logísticas, máquinas de soporte vectorial y las redes neuronales artificiales. Dentro de los algoritmos se aprendizaje de máquina existen distintos paradigmas:

- Aprendizaje supervisado: En este paradigma se tienen conjuntos de datos donde se conoce la entrada y salida al modelo. Se llama supervisado dado que se le entrega el resultado esperado para que el modelo encuentre los parámetros que a partir de una entrada obtenga la salida esperada. Un caso de uso de esto es la clasificación en imágenes, donde se tiene de entrada la imagen y se tiene como salida la etiqueta que indica la clase del objeto en ella.
- Aprendizaje no supervisado: En este caso no se tienen las salidas esperadas, se tiene un set de datos con distintos elementos y se busca que el algoritmo de aprendizaje no supervisado sea capaz de encontrar patrones y relaciones entre los datos. Un caso de uso de esto es separar conjuntos de datos en clusters
- Aprendizaje reforzado: El último paradigma comprende los casos en que no se tiene la salida esperada pero se tiene una forma de medir el desempeño de un algoritmo, por lo

que se optimiza una función de recompensa acumulada. En estos casos el problema a abordar debe ser capaz de ser representado como un problema de decisión de Markov. Un ejemplo de esto es un robot andando en bicicleta, no se tiene la salida exacta de los comandos de los motores en todo momento pero se puede medir el desempeño como la cantidad de distancia recorrida sin caerse. Por lo anterior, se puede optimizar el modelo para maximizar esa función de recompensa incluso sin saber los comandos exactos de los motores.

2.2.2. Redes Neuronales Convolucionales

Las redes neuronales son un modelo de aprendizaje de máquinas basado en el funcionamiento de las neuronas del cerebro humano. Estas están compuestas por múltiples unidades llamadas neuronas que se relacionan entre ellas a través de conexiones y capas para modelar así sistemas complejos a través de una representación matemática. Cada una de estas capas de neuronas reciben una entrada y a través de sumas y multiplicaciones entregan una salida a la siguiente capa. Entre estas capas se aplican funciones no lineales para otorgar así a estos modelos la capacidad de representar sistemas no lineales. Los parámetros de estas operaciones matemáticas corresponden a los parámetros que se optimizan en a partir de una función de pérdida o error.

Las redes convolucionales son un tipo de red neuronal artificial en donde el procesamiento de las capas de esta se basa en la operación de convolución, estas al ser aplicadas en imágenes tienen el efecto análogo de aplicar filtros. A partir de estas operaciones de convolución y el aprendizaje de los parámetros de estas, se logra extraer información de imágenes que posteriormente se pasa a otro tipo de bloques o capas, por ejemplo capas *fully connected* para el caso de clasificación. Estas redes son muy utilizadas en visión computacional dado los buenos resultados que tienen, donde se le entregan las imágenes directo sin extraer características y logran generar la información de interés para el usuario. Estas redes se caracterizan por tener 3 tipos de capas: capas de convolución, capas de pooling y capas de clasificación.

Capa de convolución

Esta capa realiza las operación de convolución utilizando distintos kernels, en este caso los kernels corresponden a los pesos a entrenar. El objetivo de estas capas es aplicar convoluciones lo que es equivalente a aplicar filtros a las imágenes para así obtener distintas características de la imagen. En la ecuación 2.8 se muestra como es la operación entre capas.

$$Y_j = g(b_j + \sum_i K_{ij} \circledast Y_i) \tag{2.8}$$

Donde Y_j corresponde a la salida de una neurona j de la capa de salida que es calculada con la sumatoria de las convoluciones de las neuronas de la capa anterior Y_i con el kernel K_{ij}, a esto se suma el bias b_j y se aplica una función de activación no lineal g.

Capa de pooling

La capa de pooling corresponde a una capa que realiza sub-muestreo de las imágenes, esto otorga a la red robustez para algunas transformaciones geométricas. La función de pooling comunmente más usada es max-pooling la cuál, dependiendo de cuanto se debe reducir la dimensionalidad de la imagen en cierta vecindad, busca el valor máximo y ese se transfiere a la imagen de salida de menor dimensión.

Capa de clasificación

La tercera y última capa corresponde a la capa *fully-connected* que se encarga de la clasificación de la red. Estás capas reciben las características extraídas por las capas anteriores y proceden a realizar la clasificación, los pesos de cada *perceptron* de ajustan para lograr la mejor clasificación. La operación en la capa de clasificación se presenta en la ecuación 2.9.

$$y_j = g(b_j + \sum_i w_{ij} * y_i) \tag{2.9}$$

2.2.3. Aprendizaje Profundo o Deep Learning

Las redes neuronales profundas o *Deep Neural Networks* son un tipo de modelos de *Machine Learning* basado en redes neuronales convolucionales que contienen una gran cantidad de capas. Estos modelos a diferencia de otros modelos de *Machine Learning* no requieren una extracción de características manual o pre procesamiento de un experto. Los modelos de *Deep Learning* reciben en la entrada los datos directos y las primeras capas se encargan de la extracción de características. Por ejemplo, en el caso de procesamiento de imágenes, al usar modelos de *Deep Learning* se le entrega al modelo la imagen directamente, en cambio, en otros modelos de *Machine Learning* es necesario extraer características manualmente de las imágenes e ingresar estas al modelo.

Desde la aparición de AlexNet [19] en 2012 resolviendo ImageNet [20] con un desempeño considerablemente mayor que otros algoritmos, han surgido cada vez redes convolucionales profundas más grandes y que implementan nuevos bloques como son VGG16 [21] y ResNet [22]. El Deep Learning muestra resultados sobresalientes en distintos problemas, pero dado que la red tiene que extraer las características de los datos directamente se necesita una gran cantidad de datos y entrenamiento. Estas redes son de gran tamaño y por tanto requieren de hardware con alto rendimiento para su uso y aún mayor para su entrenamiento, teniendo tiempos de entrenamiento de días o meses en computadores convencionales. Actualmente, existen múltiples trabajos donde se plantean nuevas arquitecturas con nuevas capas y bloques para resolver distintos tipos de problemas como son Detección de objetos, Segmentación de objetos, Reconocimiento facial, etc.

El trabajo con modelos de aprendizaje de máquinas y aprendizaje profundo depende de la optimización de los parámetros del modelo con tal de optimizar una función de pérdida.

Dentro de este trabajo existen hiper parámetros, variables y decisiones de diseño que se configuran previo al proceso de optimización y búsqueda de parámetros. A continuación se describen brevemente algunas de estas variables importantes para este trabajo:

- Conjunto de entrenamiento: Corresponde a un sub conjunto de datos del conjunto de datos total a utilizar. Este sub conjunto de datos es una fracción representativa del conjunto total y será el sub conjunto a utilizar por el modelo de aprendizaje de máquinas para optimizar sus parámetros. Este subconjunto corresponde a la mayor parte de los datos, en general corresponde al 75-80 %.
- Conjunto de prueba: Corresponde a un sub conjunto de datos del conjunto de datos total, distinto al conjunto de entrenamiento. El conjunto de prueba es reservado para evaluar el desempeño de un modelo de aprendizaje de máquina posterior a todo el proceso de entrenamiento y ajuste. Este sub conjunto de datos no debe ser conocido por el modelo hasta la evaluación final, en general corresponde al 10-25 %
- Función de pérdida: Corresponde a la función que calcula el error que tiene un modelo en una tarea determinada, el proceso de entrenamiento busca reducir este error para optimizar los parámetros buscando el mejor desempeño del modelo.
- Optimizador: Corresponde al método de optimización de parámetros a utilizar, estos varían dependiendo del tipo de modelo, tarea del modelo y el conjunto de datos.
- Tasa de aprendizaje: Corresponde a un factor que es multiplicado al valor de error de la función de pérdida. Este factor regula qué tanto se modificarán los parámetros del modelo en función del error, por lo que corresponde a la velocidad de aprendizaje.
- Épocas: Corresponde a la cantidad de veces que el modelo procesará el conjunto de datos de entrenamiento en el proceso de entrenamiento. Donde una época corresponde a recorrer cada dato del conjunto de datos de entrenamiento.
- Tamaño de batch: Dentro del proceso de entrenamiento a los modelos se le entregan los datos en lotes o grupos con el fin de aprovechar el procesamiento paralelo y aumentar la velocidad del proceso de entrenamiento. El tamaño de batch corresponde a cuantos datos le entregaré al modelo simultáneamente para su cálculo de error. En el caso de este trabajo corresponde a la cantidad de imágenes que se le entrega al modelo agrupadas.

2.2.4. Detección de objetos usando aprendizaje profundo

Este trabajo de tesis se centrará en el problema de detección de objetos, donde se busca encontrar en una imagen la ubicación del objeto y reconocer que clase es. Para esto tiene como salida común del modelo de aprendizaje profundo una lista con los objetos detectados. Para cada objeto se entrega su clase y 4 coordenadas que parametrizan el recuadro que encierra al objeto buscado (también llamado *bounding box*). Un ejemplo de esto se puede observar en la Figura 2.7 donde se detectan distintos objetos en la escena de una habitación indicando qué clase de objetos son.

Los modelos de aprendizaje profundo de detección de objetos se pueden separar en dos enfoques principales: detectores de dos etapas y detectores de una etapa.

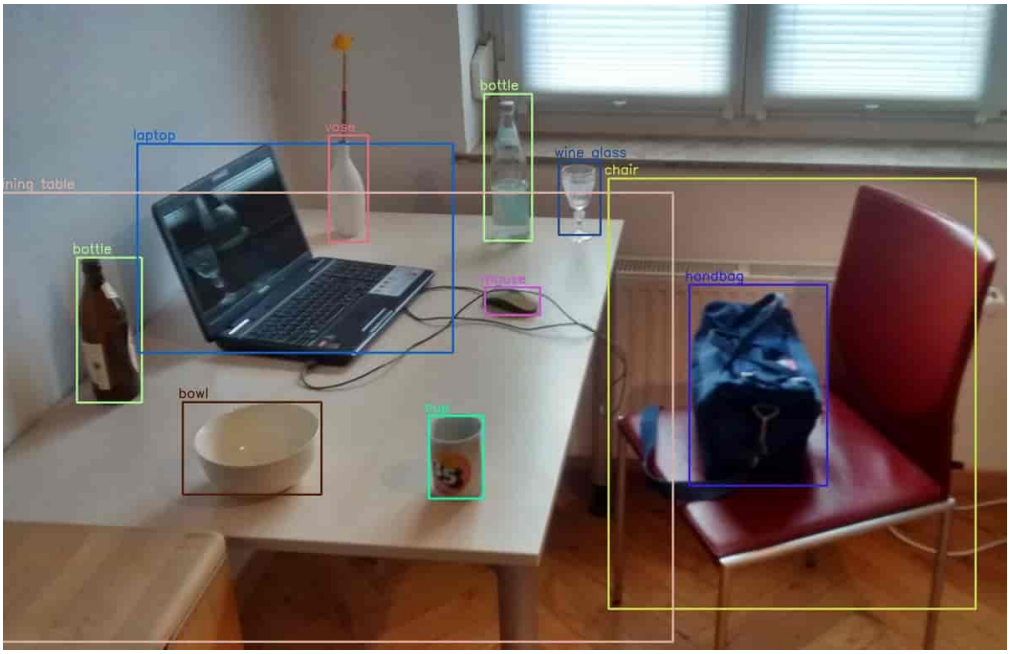

Figura 2.7: Ejemplo de detección de objetos sobre una imagen de habitación.

- **Detectores de dos etapas**: Estos detectores primero generan propuestas de regiones donde podrían estar los objetos y luego clasifican estas regiones. La generación de regiones propuestas se realiza con búsqueda selectiva en modelos como *R-CNN* [23] y *Fast R-CNN*[24] o por un modelo encargado de esto, *Region Proposal Network (RPN)*, en modelos como *Faster R-CNN*[25].

- **Detectores de una etapa**:Estos detectores unifican el proceso de detección en una sola etapa, directamente prediciendo las clases y las posiciones de los objetos en un solo modelo. Son más rápidos y se utilizan en aplicaciones donde la velocidad es crítica. Dentro de este tipo de modelos también se encuentran 2 sub tipos: modelos que usan *anchors* o modelos sin *anchors*. Los *anchors* o cajas ancla son cajas predefinidas de diferentes tamaños y relaciones de aspecto que se utilizan como referencias para predecir las ubicaciones de los objetos en una imagen. Los **modelos con anchors** colocan estos anchors en puntos de referencia (como los centros de las celdas en una cuadrícula) y luego ajustan estos anchors para adaptarse a los objetos detectados. Los modelos más conocidos que utilizan anchors son: *YOLO (You Only Look Once)* [8], *SSD (Single Shot MultiBox Detector)* [26] y *RetinaNet* [27]. Por otro lado, los **modelos sin anchors** no dependen de cajas predefinidas para detectar objetos. En cambio, utilizan diferentes técnicas para predecir las ubicaciones y tamaños de los objetos directamente. Algunos modelos destacados que no utilizan anchors son: CornertNet [28] y CenterNet[29].

2.2.5. Modelos livianos

El uso de algoritmos de Aprendizaje Profundo se ha masificado dado el buen resultado que tiene en distintas tareas, dado esto, la investigación sobre estos modelos se ha orientado en los últimos años en su masificación para dispositivos cotidianos con una menor capacidad

de cómputo: computadores con GPU de bajo desempeño, computadores sin GPU, dispositivos embebidos, smartphone, etc. Estos trabajos que buscan estos objetivos de modelos con menores requerimientos generan nuevos modelos livianos capaces de ejecutarse en dispositivos con capacidades limitadas. Para lograr esto se emplean técnicas de búsqueda de arquitectura (NAS o Network Architecture Search) [30][31][32] y Compresión de arquitectura [33][34], donde se busca disminuir requerimientos de computo, manteniendo un procesamiento en tiempo real y precisión. Algunos modelos livianos revisados previo a la selección de YOLO V8n fueron: EfficientDet [35][36], NanoDet [37], PicoNet [38], YoloV5, MobileNet2 SSD [39]

2.2.6. YOLO V8

YOLO (You Only Look Once) [8] es un modelo de detección de objetos de una sola etapa basado en anchors, altamente utilizado en visión por computadora debido a su buen desempeño en términos de precisión y velocidad. El trabajo original de este detector [8][40][9] ha sido mejorado por distintos autores, generando nuevas versiones [10][11][12] que buscan mejorar aún más la precisión y velocidad. Entre las mejoras incorporadas a lo largo de las distintas versiones se incluyen la incorporación de anchors, modelos más profundos (Darknet), detección a múltiples escalas, técnicas de entrenamiento y optimización de rendimiento.

La base del trabajo asociado a YOLO consiste en la detección unificada, que realiza el proceso de *region proposals* y clasificación en un solo procesamiento utilizando un modelo único. Esto permite que el modelo del detector se entrene de extremo a extremo, logrando una mayor velocidad de inferencia.

Para realizar la detección unificada, YOLO divide la imagen en un cuadrícula de *SxS* y, para cada celda de la cuadrícula, se predicen una cantidad fija de posiciones de objetos usando recuadros o *bounding box*. Junto a cada bounding box de un posible objeto, se obtienen 5 valores:

- x: Coordenada x dentro de la imagen del centro del *bounding box*.
- y: Coordenada y dentro de la imagen del centro del *bounding box*.
- w: Ancho del *bounding box*.
- h: Alto del *bounding box*.
- confianza: Nivel de confianza o seguridad de la detección del modelo.

Además de estos valores, se estima una distribución de probabilidad por cada clase de objetos. Para ajustar todos estos parámetros en una sola etapa, se utiliza un función de pérdida unificada que considera la ubicación y la clase de los objetos. La función de pérdida toma en cuenta la ubicación (x,y) y el tamaño de los objetos (w,h) mediante la intersección (IOU). Adicionalmente, se considera la existencia de objeto o no en las celdas y, finalmente, la probabilidad de clase por celda.

En la ecuación 2.10 se presenta la función de pérdida utilizada en YOLO, donde las 2 primeras filas están asociadas a las coordenadas de la detección y las últimas 3 filas asociadas al objeto y clase.

$$
\begin{aligned}
\mathcal{L} = & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
& + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\
& + \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\
& + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\
& + \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
\tag{2.10}
$$

Donde:

- λ_{coord}: Ponderación para la pérdida de las coordenadas (x, y, w, h) del *bounding box*.
- S: Tamaño de la cuadrícula en la que se divide la imagen.
- B: Número de *bounding boxes* predichos por celda de la cuadrícula.
- $\mathbb{1}_{ij}^{\text{obj}}$: Indicador de si el *bounding box* j en la celda i es responsable de la detección del objeto.
- x_i, y_i: Coordenadas del centro del *bounding box* predicho.
- $\hat{x}_i, \hat{y}_i$: Coordenadas del centro del *bounding box* real.
- w_i, h_i: Ancho y alto del *bounding box* predicho.
- $\hat{w}_i, \hat{h}_i$: Ancho y alto del *bounding box* real.
- C_i: Confianza del *bounding box* predicho.
- $\hat{C}_i$: Confianza del *bounding box* real.
- λ_{noobj}: Ponderación para la pérdida de confianza cuando no hay objeto presente en la celda.
- $p_i(c)$: Probabilidad predicha de la clase c en la celda i.
- $\hat{p}_i(c)$: Probabilidad real de la clase c en la celda i.
- $\mathbb{1}_{ij}^{\text{noobj}}$: Indicador de si no hay objeto presente en el *bounding box* j en la celda i.

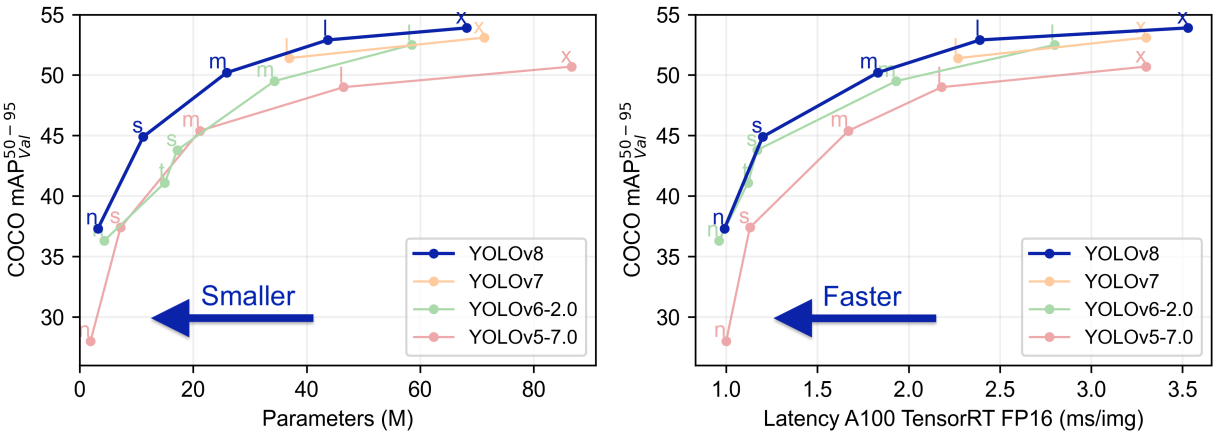

Figura 2.8: Gráfico de comparación de versiones y variaciones de YOLO. (Figura extraída de https://github.com/ultralytics)

En este trabajo de tesis se utiliza el modelo YOLO V8 implementado en la librería de Python Ultralytics. Dentro de las distintas mejoras realizadas en estos trabajos, se tiene la selección de tamaños para que el usuario escoja el modelo que se ajuste a sus necesidades de velocidad y capacidad de cómputo. Las variaciones de YOLO V8 corresponden a: n (nano), s (small), m (medium), l (large) y x (x large).

Se decide trabajar con YOLO V8 debido a su desempeño en distintos benchmark y a la disponibilidad de variaciones de tamaño que colaboran en la evaluación de la metodología profesor-estudiante que se quiere implementar. Junto a esto, se evaluó el desempeño de la versión más pequeña de YOLO v8 mostrando un buen desempeño a nivel de precisión y velocidad en hardware limitado.

En la Figura 2.8 se presentan las variaciones de precisión y velocidad entre distintas versiones de YOLO y sus variaciones de tamaño.

Recientemente, al finalizar este trabajo, se publicó YOLO V9 [41], por lo que su evaluación para implementar en este trabajo puede ser considerado en el futuro.

2.2.7. Destilamiento de conocimiento

El Destilamiento de Conocimiento o método profesor-estudiante (teacher-student) en deep learning es un enfoque donde un modelo grande y complejo (*teacher* o profesor) se utiliza para entrenar un modelo más pequeño y eficiente (*student* o estudiante)[13][14]. Esta metodología es especialmente útil para facilitar el entrenamiento de un modelo más pequeño aprovechando el aprendizaje del modelo grande, permitiendo desplegar el modelo pequeño en dispositivos con recursos limitados. El proceso de entrenamiento por método profesor-estudiante se puede describir en los siguientes pasos:

- **Entrenamiento de modelo profesor**: Se entrena el modelo grande y complejo, con el objetivo de lograr el mayor nivel de precisión posible. Se busca que este entrenamiento

una sola vez o se reentrene con poca frecuencia.

- **Generación de etiquetas**: En el nuevo conjunto de datos de entrenamiento donde se busca que el modelo estudiante tenga buen desempeño se generan predicciones usando el modelo profesor. Esto genera etiquetas de manera automática sin proceso de etiquetado manual realizado por humanos. De este proceso se pueden generar etiquetas reales y también se pueden generar etiquetas suaves o *soft labels*. Las etiquetas suaves corresponden a las probabilidades de clases que también representan parte del aprendizaje del modelo profesor.
- **Entrenamiento de modelo estudiante**: Se realiza el entrenamiento del modelo estudiante buscando que imite el desempeño del profesor en base a las etiquetas generadas. En caso de utilizar etiquetas suaves se debe intregrar una pérdida adicional en la función de error.

Una vez terminado el proceso el modelo pequeño logra mejorar su precisión imitando el comportamiento del modelo profesor. Esto en ciertas bases de datos genera un resultado positivo acercándose a la precisión del modelo profesor pero permitiendo menor complejidad, mayor velocidad y la posibilidad de desplegarse en hardware con recursos limitados.

Junto a lo anterior se plantea que esta metodología puede ser más efectiva en modelos pequeños que tienen ambientes de operación controlados, cómo una cámara fija. El concepto detrás de esto es que la red pequeña no puede generalizar de la misma forma que la red grande para distintos contextos, pero se propone que puede imitar el comportamiento en un caso más restringido encontrando patrones particulares del contexto de uso.

2.3. Herramientas y Software

2.3.1. ROS

Roboting Operating System (ROS) [42] [43] consiste en una framework de programación de robótica que incluye una gran cantidad de paquetes, herramientas y desarrollos orientados a potenciar la robótica a través de un concepto de comunidad. Este framework plantea el funcionamiento de los distintos procesos de los robots a través de un sistema de nodos. Los nodos realizan procesos y comparten información enviando mensajes a través de tópicos entre ellos . A través de este sistema se logra un manejo eficiente de la información entre los distintos nodos que comprenden distintas partes de una tarea objetivo en un robot. En la Figura 2.9 se muestra con un ejemplo simple cómo se comunican los nodos a través de mensajes por los tópicos. Los mensajes de estos tópicos pueden ser enviados por un publicador y pueden ser recepcionados por suscriptores que están revisando la información del tópico. Adicionalmente bajo el mismo sistema de mensajes y tópicos se establecen servicios donde interactúa un servidor y un cliente en el traspaso de información.

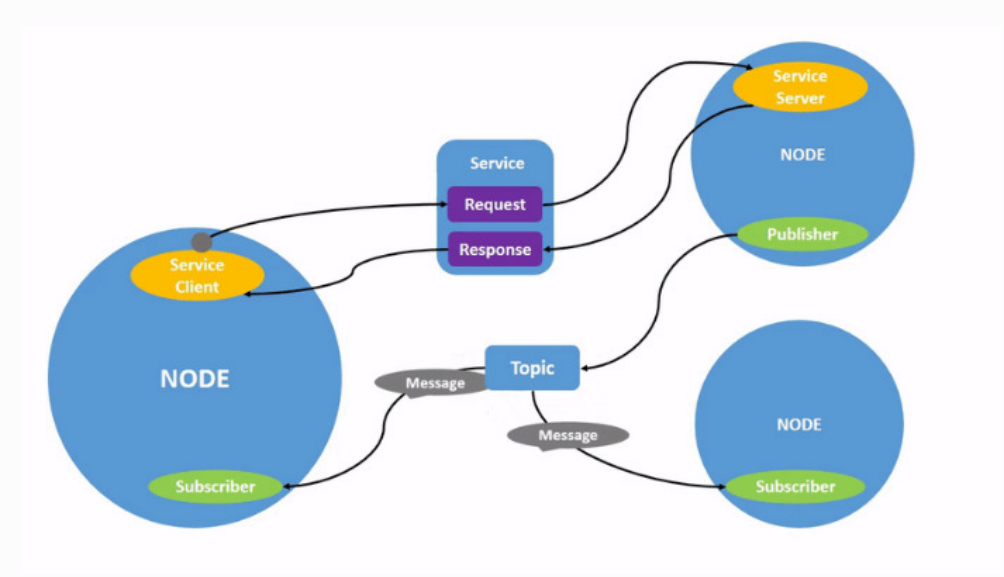

Figura 2.9: Ejemplo de interacción entre nodos de ROS compartiendo mensajes a través de tópicos. (Extraída de http://wiki.ros.org/ROS/Tutorials)

2.3.2. Simulador Gazebo

Gazebo [44] es un simulador capaz de crear ambientes 3D multi-robot con reglas físicas. Este simulador es altamente usado en robótica dado que permite la integración con ROS para entregar a través de un puente (ros-bridge) la información del robot simulado vía tópicos como si fuera el robot real. Dado que el simulador aplica reglas físicas, es muy preciso en la simulación de mediciones de los sensores y la interacción de los robots con distintos espacios, personas, vehículos y otros robots. El uso de simuladores en muy importante en este trabajo y robótica dado que es una forma de proteger a los robots de sufrir daños en la etapa de experimentación cuando se está realizando investigando o desarrollando nuevas características.

2.4. Trabajo relacionado

En esta tesis, se propone utilizar las cámaras de CCTV presentes en el edificio donde se desplaza un robot móvil para generar información que mejore su localización y navegación en el entorno. Para lograr este objetivo, se plantea el uso de modelos de aprendizaje profundo para detectar al robot dentro del campo visual de las cámaras, así como otros objetos de interés, tales como obstáculos y objetivos.

Un trabajo relevante en el ámbito del uso de cámaras para resolver problemas de localización y navegación es el sistema ORB-SLAM [1][2][3]. En sus diversas versiones, este sistema aborda estos desafíos mediante el uso de cámaras monoculares, estéreo o RGB-D para la detección de características en el entorno, con el fin de generar un mapa del lugar, localizarse y navegar. ORB-SLAM utiliza descriptores locales, sin aplicar detección de objetos mediante aprendizaje profundo, y se enfoca en ofrecer un funcionamiento en tiempo real con un

costo computacional reducido. Estos trabajos integran las características detectadas para la localización del robot utilizando filtros, como el filtro de Kalman extendido, que permiten gestionar la incertidumbre en las mediciones. Además, el sistema está diseñado para detectar las mismas características cuando el robot vuelve a recorrer el mismo lugar. Diversos estudios han ampliado los principios de ORB-SLAM para abordar temas relacionados con la navegación visual.

En el contexto del desafío de la navegación, se han desarrollado trabajos que buscan detectar obstáculos utilizando cámaras de profundidad, para ser considerados en la planificación de rutas y en la evasión de obstáculos. Xu et al. [4] destacan la necesidad de desarrollar algoritmos basados en información visual que faciliten la detección de obstáculos, teniendo siempre en cuenta las limitaciones computacionales de los robots. En su trabajo, utilizan cámaras de profundidad para la detección de obstáculos, combinando métodos que no utilizan aprendizaje profundo con métodos de aprendizaje de bajo consumo. Por ejemplo, procesan la nube de puntos generada por la cámara de profundidad para detectar obstáculos con sus formas exactas, aunque este enfoque no permite identificar fácilmente el tipo de obstáculo. Para ello, emplean modelos basados en aprendizaje, como YOLO en una de sus versiones más ligeras, con el objetivo de clasificar el tipo de obstáculo detectado, como si se trata de una persona. Esto es crucial, ya que la respuesta del robot varía dependiendo del tipo de obstáculo, especialmente si es estático o dinámico. Posteriormente, procesan la información mediante un filtro de Kalman para determinar la posición del obstáculo, que luego se integra en los algoritmos de navegación del robot. Las pruebas de este trabajo se llevaron a cabo utilizando dispositivos de capacidad limitada, como un Intel NUC y una Nvidia Jetson Xavier NX, para comparar su rendimiento.

La detección de objetos en el proceso de navegación puede orientarse a diversas tareas, pero debe coexistir con otros procesos del robot. Singh et al. [5] proponen el uso de modelos de detección basados en aprendizaje profundo para robots de servicio en sus tareas cotidianas. Este trabajo destaca la importancia de la detección de distintos tipos de objetos, así como de la estimación de su posición en el mapa, para su posterior uso en la realización de tareas, como cuando uno de estos objetos es el objetivo de una tarea específica. El estudio sugiere el uso de YOLO V2 con modificaciones que reducen su consumo computacional, reduciendo capas de convolución, permitiendo que funcione simultáneamente con los procesos del sistema de navegación. En esa sección también se hace mención a otros modelos livianos, como MobileNet para abordar ese desafío.¨En paralelo al sistema de detección de objetos, el robot debe procesar información de sensores y de imágenes RGB-D en tareas de localización y navegación. En este contexto, el sistema prioriza la localización y navegación, saltando frames del procesamiento de imágenes cuando la CPU del robot está comprometida, para asegurar el correcto funcionamiento de las tareas críticas. Las pruebas se llevaron a cabo utilizando una Nvidia Jetson TX2.

Otro enfoque diferente para abordar el problema de SLAM es el planteado por CoVOR-SLAM [45]. Lee et al. proponen el uso de comunicación entre varios robots en forma de enjambre para reducir el error en la estimación de la posición de cada robot o agente. En este caso, es crucial equilibrar la precisión de la posición con el consumo de recursos computacionales, considerando también los recursos requeridos para la transmisión de información entre agentes. Para gestionar este compromiso, el trabajo sugiere compartir solo la información esencial, donde cada agente realiza un procesamiento inicial de los datos antes de transmitir

la posición de otros agentes junto con su incertidumbre. Para ello, se utilizan tres tipos de mensajes que contienen información sobre la posición del agente con posibles errores, la distancia entre agentes y la distancia a las antenas de comunicación. La posición de los agentes se parametriza en matrices de 7 grados de libertad para optimizar la transmisión de información. La posición de estos robots o agentes se genera mediante odometría visual utilizando las cámaras a bordo, y no es necesario que todos los agentes estén comunicados; la adición de la cámara monocular de un agente extra ya reduce significativamente el error de posición de los agentes. Además, se plantea una integración sencilla con la información de otros sensores utilizados en el proceso de SLAM. Otra ventaja de este trabajo es que el procesamiento no es centralizado, lo que elimina la necesidad de un equipo central con gran capacidad de cómputo, ya que cada agente procesa la información localmente y la comparte ya procesada, logrando así un procesamiento descentralizado. Este trabajo emplea simuladores como Gazebo para las pruebas, complementándolas con experimentos en el mundo real.

Revisando los distintos trabajos mencionados relacionados con la navegación y el uso de modelos de aprendizaje automático, se establecen los lineamientos para esta tesis. Muchos de los trabajos revisados utilizan cámaras instaladas en el propio robot para generar información de puntos de referencia u obstáculos que apoyen el proceso de localización y navegación. Además, en el trabajo mencionado anteriormente, se observa cómo múltiples robots colaboran utilizando su información visual para apoyar al conjunto de agentes. A partir de esto, se propone el uso de cámaras instaladas en el entorno, que podrían considerarse como nuevos agentes del enjambre, generando información útil para el robot o agente principal. Todos los trabajos subrayan la necesidad de que el procesamiento sea eficiente, de modo que pueda ejecutarse dentro del robot sin comprometer otras tareas críticas, como la localización y la navegación. Por ello, se revisan trabajos relacionados con la generación de información utilizando modelos de aprendizaje profundo y cámaras externas ya presentes en el entorno.

Ahmed et al. [6] exploran la capacidad de las cámaras de seguridad actuales para combatir delitos, destacando la dificultad de monitorear eficazmente una gran cantidad de cámaras o pantallas. Proponen que los modelos de aprendizaje profundo pueden ofrecer una solución a este problema y sugieren la posibilidad de colaboración entre robots móviles y cámaras fijas de seguridad para la detección de comportamientos delictivos. Además, subrayan la ventaja de las cámaras ubicadas en altura, ya que presentan una menor oclusión y permiten cubrir una mayor cantidad de terreno. Esto es relevante en su trabajo, dado que utilizan trackers para monitorear el comportamiento de personas y vehículos. En su estudio, evalúan dos tipos de detectores, YOLO y SSD, en distintas vistas de una cámara robótica para medir la precisión de la detección. Además, complementan esta evaluación con el análisis de distintos tipos de trackers, como Boosting, MedianFlow y GoTurn, para seguir a una persona o vehículo dentro de la imagen con fines de seguridad contra el delito. Finalmente, este trabajo resalta el potencial de utilizar cámaras de vigilancia junto con modelos de aprendizaje profundo para generar información valiosa.

Bultmann et al. [7] proponen el uso de la red de cámaras estáticas del edificio para la detección del robot, mejorando así su localización. Este trabajo engloba distintos conceptos revisados previamente y es el más relacionado con el trabajo de esta tesis. Este trabajo utiliza redes neuronales convolucionales para la detección del robot y la estimación de puntos clave (keypoints) con el fin de determinar la posición del modelo 3D del robot. En lugar de utilizar sistemas de transformadas entre sistemas de referencia de las cámaras, emplean la información

de los puntos clave y modelos de aprendizaje automático. Para el entrenamiento, usan datos sintéticos debido a la complejidad de crear una base de datos en la realidad. Todo este sistema se ejecuta en una unidad de procesamiento externa conectada a la red de cámaras (Nvidia Jetson Xavier NX), la cual obtiene las imágenes, ejecuta el detector y estima la pose del robot para integrarla en el sistema de localización del robot. Este trabajo plantea un método novedoso sin etiquetado que ayuda en la navegación, particularmente con el problema del robot secuestrado (kidnapped robot problem).

Basado en la revisión de estos trabajos sobre la colaboración entre robots y cámaras externas, y considerando los avances en modelos de aprendizaje profundo eficientes, se plantea el diseño e implementación de un sistema que utilice la vista de las cámaras para apoyar la localización y navegación del robot. Dada la motivación y la facilidad de utilización del sistema, se propone el uso de un modelo liviano que sea capaz de ejecutarse dentro del robot sin necesidad de computación externa, a diferencia de los trabajos mencionados anteriormente. En esta línea, se plantea utilizar modelos más actuales, como las últimas versiones de YOLO [10][11][12], que pueden presentar un mejor desempeño en la detección de objetos comparado con los modelos utilizados en los trabajos revisados. El trabajo de Bultmann et al. solo plantea la detección del robot para apoyar el sistema de localización, mientras que el trabajo de Ahmed et al. se enfoca en la detección de personas y vehículos asociados a temas delictivos, pero no mencionan al robot. En esta tesis, se plantea la detección del robot para colaborar en su localización, así como la detección de dos obstáculos móviles comunes en ambientes cerrados: personas y sillas. La incorporación de estos obstáculos tiene como objetivo apoyar el sistema de navegación para la planificación de trayectorias globales y la búsqueda de objetivos.

Otro desafío para la utilización de un sistema de este tipo es la calibración y el entrenamiento en un nuevo ambiente donde deba trabajar el robot. Estos trabajos generalmente implican una gran cantidad de horas humanas en ajustes de parámetros y etiquetado para que el modelo liviano que corre dentro del robot obtenga un buen nivel de precisión. Para abordar esto, se plantea la utilización de un entrenamiento del modelo liviano usando el método profesor-estudiante [13][14]. Este método permite que un modelo de gran tamaño o profesor, que tiene un buen nivel de detección del robot y los obstáculos, transfiera su conocimiento a un modelo liviano o estudiante que es capaz de ejecutarse dentro del computador del robot. El hecho de que las cámaras tengan una vista fija facilita el entrenamiento de este método, permitiendo una transferencia en modelos pequeños con una precisión cercana a la del modelo profesor. El uso de este método también permite utilizar datos e imágenes reales, reduciendo la necesidad de usar tantos datos sintéticos y evitando así el arduo trabajo de etiquetado manual mencionado en otros estudios.

En varios de los trabajos mencionados, se realizan pruebas de las soluciones propuestas utilizando hardware con capacidad computacional limitada, dado que los sistemas deben funcionar en tiempo real. Los tipos de hardware más comúnmente utilizados incluyen NVIDIA Jetson, Intel NUC y Raspberry Pi. Además, muchos de estos trabajos consideran el desarrollo, implementación y experimentación utilizando simuladores, complementados con pruebas experimentales en robots o drones reales. Este enfoque permite acelerar el proceso de validación y proteger la integridad del hardware durante las primeras etapas de desarrollo, cuando los algoritmos aún son poco robustos y arrojan resultados insatisfactorios.

Capítulo 3

Metodología

Posterior a la revisión de la bibliografía presentada en la sección de marco teórico, se plantea la metodología a utilizar en este trabajo.

La metodología abarca tres secciones principales para describir el trabajo realizado en función de los objetivos específicos planteados:

1. Diseño general y ambientes de trabajo.
2. Procesamiento información visual e integración.
3. Compresión y calibración modelo liviano.

Las secciones 2 y 3 contienen la mayor parte de la investigación y desarrollo de código orientado a los objetivos específicos planteados. La sección 1 describe los lineamientos generales del trabajo y los ambientes de trabajo que corresponden al contexto de las pruebas realizadas para la validación de la hipótesis planteada en este trabajo de tesis.

3.1. Diseño general y ambientes de trabajo

En la Figura 3.1 se muestra un diagrama de flujo del sistema planteado en este trabajo de tesis. El primer paso consiste en la conexión del robot a las cámaras a través de la red local donde el robot está realizando sus labores. La conexión se realiza a través de WiFi. Posteriormente, se realiza la detección de objetos a través de un modelo de aprendizaje profundo que se ejecuta dentro del robot. La siguiente etapa consiste en estimar la pose respecto al mapa en 3 dimensiones del robot y los obstáculos. Para esto, se utilizan modelos de regresión basados en las salidas del modelo de detección de objetos. Una vez recibida la salida del modelo de regresión con la pose de los objetos, se incorpora como información útil para el robot. En el caso de la posición del robot, este la obtiene al llamar a un servicio que entrega una nueva referencia para el sistema de localización basado en filtro de partículas. En el caso de la posición de obstáculos, se incorporan en el mapa de costos global del robot

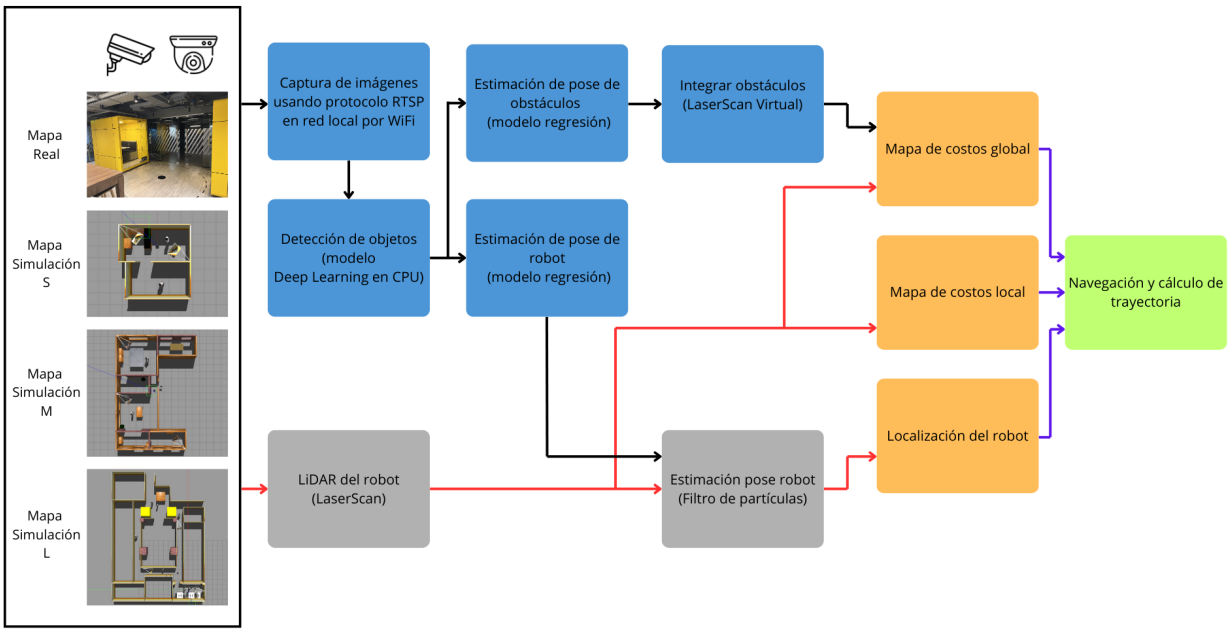

Figura 3.1: Diagrama de flujo de sistema de apoyo visual.

a través de la simulación de un sensor láser virtual que marca puntos donde hay obstáculos. Finalmente, toda esta información es utilizada por el nodo de navegación para el cálculo de trayectoria y alcanzar un objetivo.

Para acelerar el trabajo se plantea el uso de simuladores, dado que esto ayuda a la integración del algoritmo principal, una vez recibida la información de las detecciones generadas por la conexión a las cámaras. La representación en el simulador del robot y su interacción con el mapa es fidedigna, entregando información de los sensores (cámaras, LiDAR, encoders) bajo los mismos mensajes de ROS. Sin embargo, la información del simulador es perfecta, es decir, no hay ruido en las mediciones y los sensores presentan mayor estabilidad. Esto genera diferencias en la calidad de la información que pueden dificultar la implementación en un ambiente real.

El *Reality Gap* es un problema común al trabajar en simuladores, dado que el ambiente del simulador es mucho más simple y controlado. Por lo tanto, en general, el desempeño al pasar del simulador a la realidad decae considerablemente. En este caso, el *Reality Gap* afecta a las mediciones del LiDAR y las cámaras, proporcionando mejores detecciones de objetos a través de los modelos y un mayor *overfitting*. Para resolver el problema con los modelos de procesamiento de imágenes, donde es notoria la diferencia con la realidad, se plantea el reentrenamiento y calibración usando la metodología profesor-estudiante para ajustar el sistema al ambiente de trabajo real.

Por lo tanto, el uso del simulador para la descomposición del problema por etapas, donde la etapa en simulación corresponde a la navegación del robot y la transformación de referencias de la información, muestra muchas ventajas para acelerar este trabajo y resguardar la seguridad del robot real.

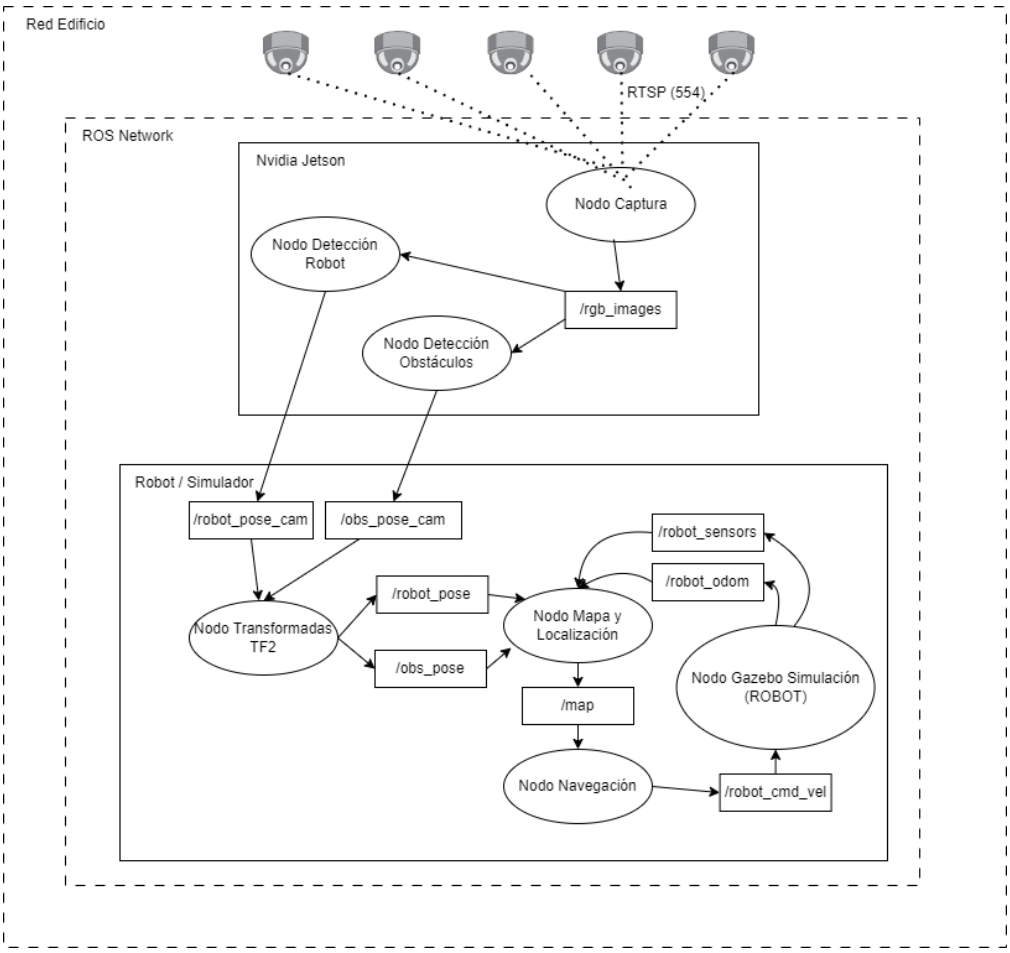

Figura 3.2: Diagrama de interacción en simulador entre nodos y tópicos ROS.

Para el trabajo y las pruebas en el simulador, se presenta la arquitectura de la Figura 3.2, donde se muestra la interacción entre los nodos a través de sus tópicos mediante *publishers*, *subscribers* y *server-clients*.

En esta simulación se utilizarán 3 escenarios para el desarrollo y pruebas:

- Mapa Sim Tamaño S: Simulación de un departamento de 2 ambientes.
- Mapa Sim Tamaño M: Simulación de una casa. Este modelo es proporcionado por los paquetes del robot turtlebot3.
- Mapa Sim Tamaño L: Simulación de una oficina y espacios comunes de un cowork en Santiago de Chile.

El modelo simulado del robot a utilizar (TurtleBot 3 Waffle Pi) se encuentra disponible y público en GitHub, dado que este modelo de robot tiene bastante documentación y es popularmente usado en investigación.

Junto con las paredes y muebles incluidos en la simulación de cada mapa se incorporan cámaras usando un plugin de Gazebo para sensores. Estas cámaras publican las imágenes

e información relacionada en un tópico de ROS para ser utilizadas, simulando así un caso real. Esta representación de cámaras permite configurar su resolución, distancia focal y otros parámetros con el fin de simular distintos modelos de cámaras.

3.1.1. Instalación y preparación de ambiente

El robot y los ambientes simulados en Gazebo se hospedan en un PC que mantiene el proceso principal de ROS asociado a la simulación en ejecución. Este PC se utilizará únicamente en las pruebas de simulación. Al realizar las pruebas en un ambiente real, el robot ejecutará el proceso principal de ROS, por lo que no será necesario tener los nodos asociados a Gazebo en ejecución. Las especificaciones del PC a utilizar en la etapa de simulación se presentan a continuación, con el fin de demostrar que el trabajo realizado es capaz de ser replicado en un computador convencional.

- CPU: AMD Ryzen 5 3600 6-Core Processor 3.59 GHz
- RAM: 16 GB
- GPU: Nvidia RTX 2060
- OS: Ubuntu 20.04 Desktop

El desarrollo de todo el trabajo de tesis se realiza usando ROS Noetic [46], posterior a una evaluación de versiones de ROS1[42] y ROS2[43]. Esta versión es compatible con el equipamiento a utilizar y es una de las últimas versiones de ROS1 que posee documentación, soporte y actualizaciones. Se realiza la instalación de las distintas dependencias de ROS, se configuran sus variables en el entorno del sistema operativo y se instalan las herramientas recomendadas para el trabajo. Junto a esta versión de ROS se instala Gazebo 11 [47] para el trabajo de simulación.

El workspace de ROS donde se desarrolla el trabajo se separa en paquetes asociados a cada componente del trabajo y aprovecha también paquetes disponibles de ROS. Los paquetes principales del trabajo son:

- `gazebo_sim`: Encargado de la simulación del ambiente en Gazebo, configuraciones de mapas y bridge de datos.
- `vision`: Encargado de la conexión a las cámaras IP, la ejecución de modelos de Deep Learning y la estimación de posición de objetos respecto al mapa.
- `vision_calibration`: Encargado del sistema de calibración automática para nuevos mapas.
- `vision_costmap_layer`: Encargado de simular un sensor virtual que marque obstáculos en el mapa de costos del robot, en base a la información de detección del paquete principal de visión.
- `navigation`: Encargado de las pruebas aleatorias del trabajo y recuperar métricas.

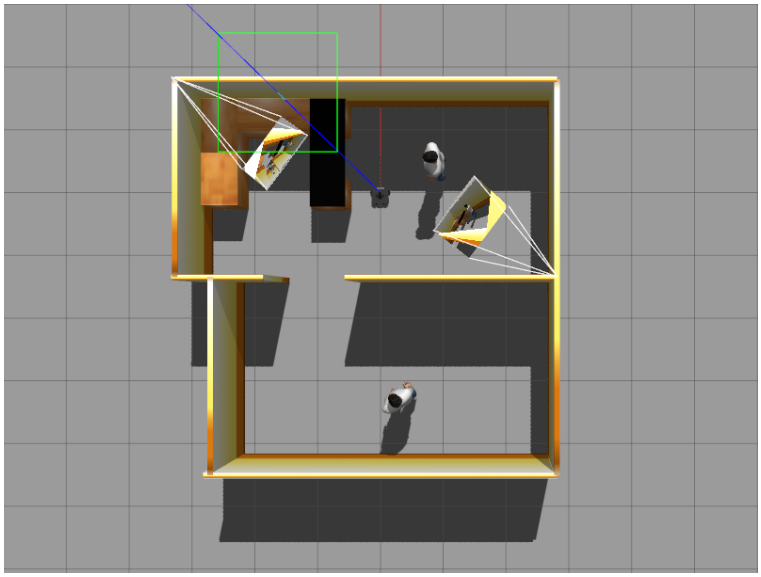

Figura 3.3: Vista de mapa S: Departamento.

3.1.2. Escenarios

Usando las distintas funcionalidades de Gazebo se construye el mapa pequeño llamado **Departamento** y el mapa grande llamado **iF Hendaya**. El mapa mediano a utilizar corresponde a la **TurtleBot3 house**, disponible en los paquetes de ROS asociados al robot TurtleBot3. Cada uno de estos mapas considera muros, obstáculos fijos y obstáculos móviles (personas y sillas). Cada mapa incluye cámaras instaladas en esquinas, simulando las cámaras de seguridad comunes de CCTV que se encuentran en los edificios.

En las Figuras 3.3, 3.4, 3.5 se muestran los mapas a utilizar desde una vista superior. En las figuras se puede observar el robot para dimensionar la escala y se muestran en las esquinas las cámaras de cada uno.

Para las pruebas en un escenario real, se define como lugar de pruebas los espacios comunes de un cowork y una oficina a la que se tiene acceso. Los espacios comunes incluyen una cocina, cabinas de reuniones, pasillos, auditorio y oficinas privadas. El mapa L de simulación se construye para este trabajo basado en los planos de este cowork. En las Figuras 3.6, 3.7 y 3.8 se pueden ver zonas del espacio y su representación en Gazebo por el mapa L.

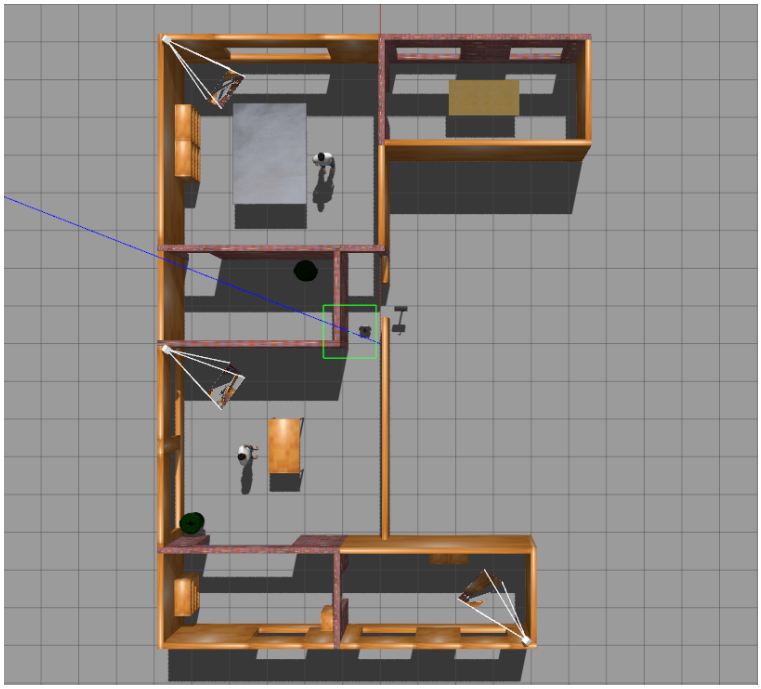

Figura 3.4: Vista de mapa M: TurtleBot3 House.

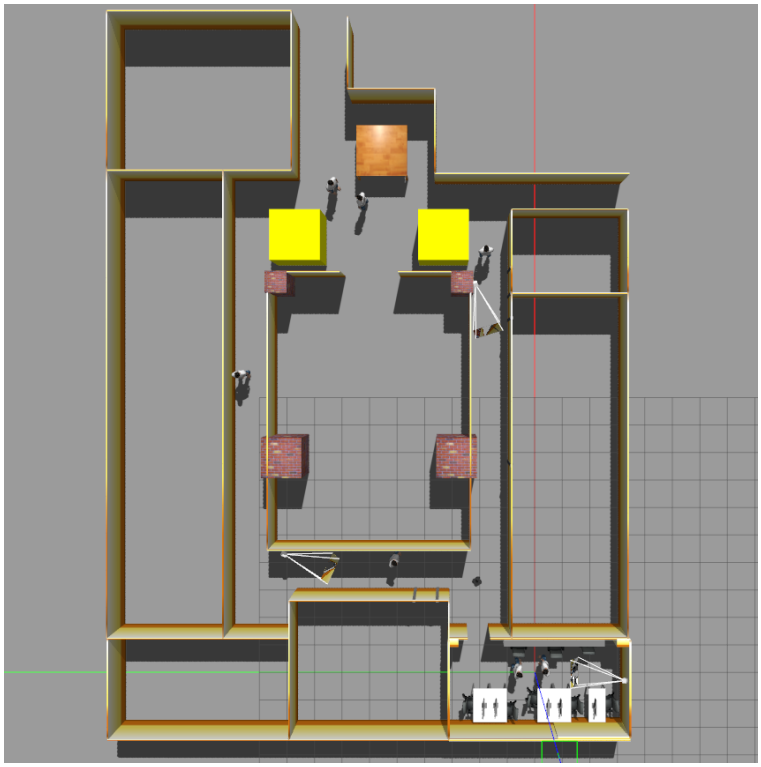

Figura 3.5: Vista de mapa L: iF Hendaya.

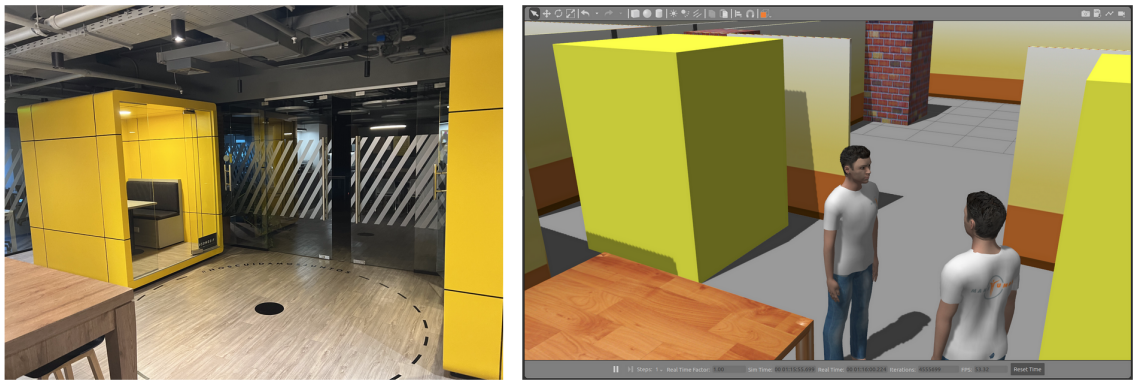

Figura 3.6: Sector cocina y cabinas reunión. A la izquierda foto real. A la derecha simulación en Gazebo.

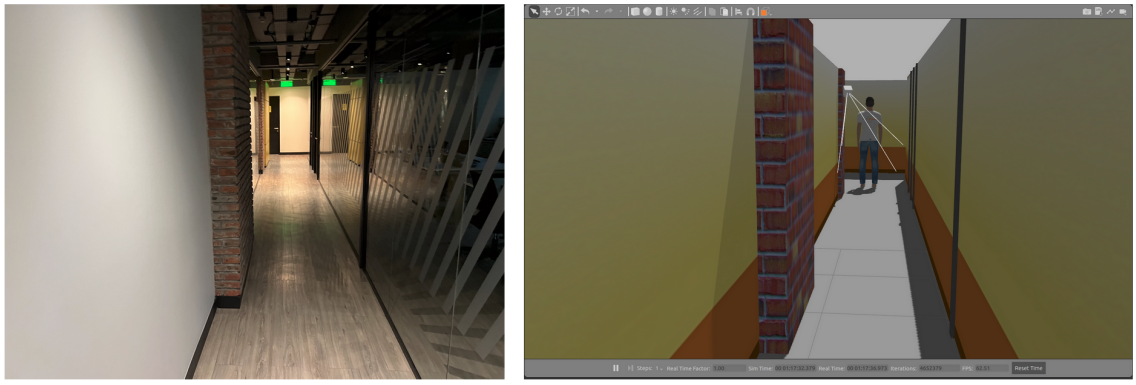

Figura 3.7: Sector pasillos con oficinas privadas. A la izquierda foto real. A la derecha simulación en Gazebo.

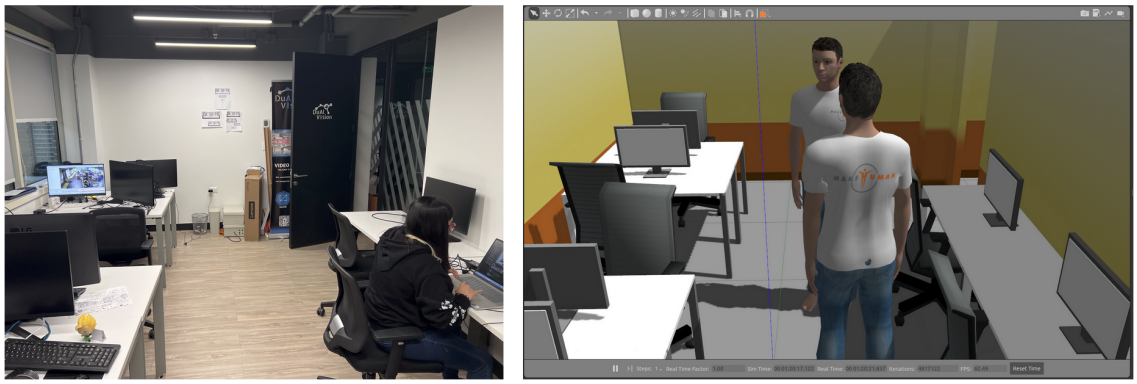

Figura 3.8: Oficina privada 7. A la izquierda foto real. A la derecha simulación en Gazebo.

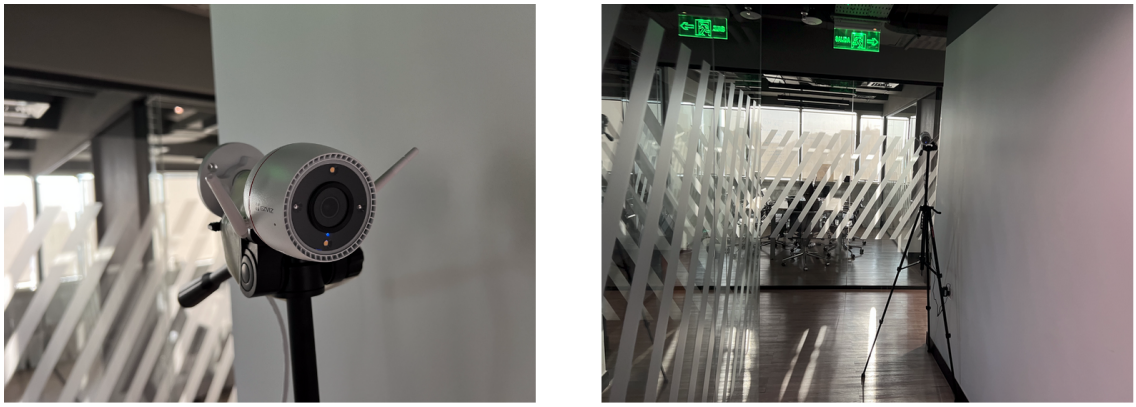

Figura 3.9: Imagen de una de las cámaras WiFi instaladas en Cowork.

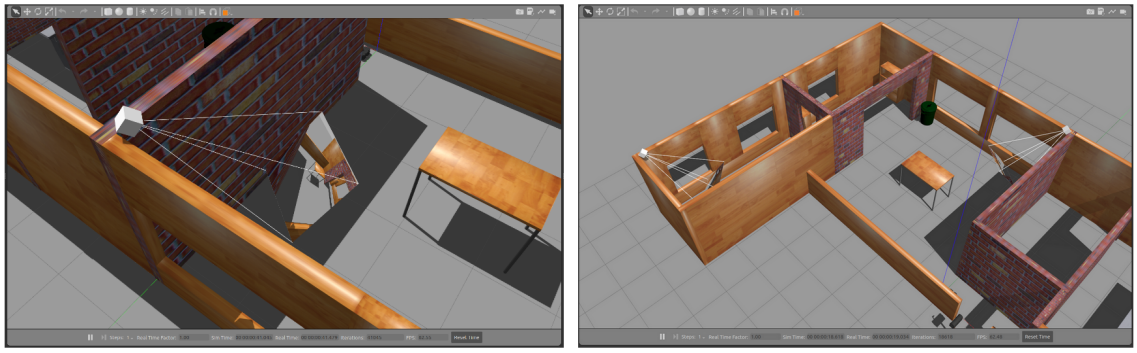

Figura 3.10: Imagen de una de las cámaras implementadas en gazebo.

3.1.3. Cámaras

Para la simulación y las pruebas reales se utilizarán cámaras IP WiFi de seguridad comunes. Dado esto, se configuran en Gazebo cámaras con las mismas características, las cuales se indican a continuación:

- Resolución: 1920x1080
- Velocidad Captura: 30 fps
- Tamaño de lente: 2.8 mm
- Campo de visión: 105°
- Modelo cámara real: EzViz CS-H3c

Estas cámaras se ubican con trípodes en los puntos seleccionados en los mapas, a una altura aproximada de 1.6 metros, a diferencia de la simulación donde las cámaras están a 2.0 y 2.4 metros de altura. En las Figuras 3.9 y 3.10 se puede ver la ubicación de algunas cámaras en las simulaciones y en el mapa real.

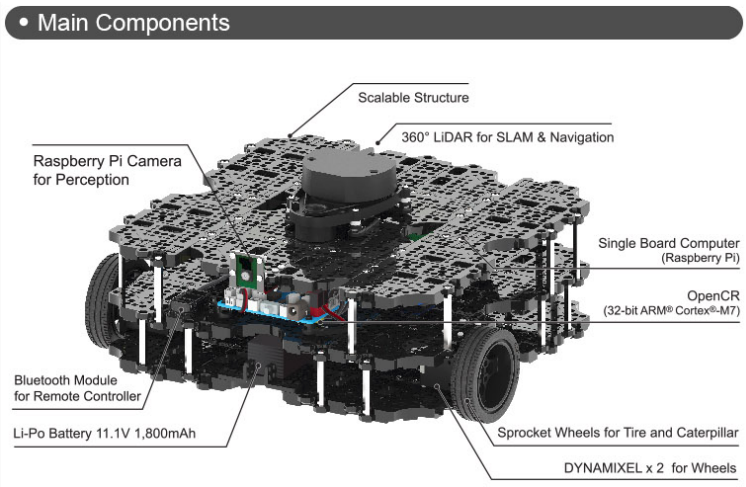

Figura 3.11: Sensores y componentes Turtlebot3 Waffle Pi. (Extraída de https://emanual.robotis.com).

3.1.4. Robot

El robot utilizado en el trabajo de tesis es un TurtleBot3 Waffle. Este robot se selecciona dado que tiene una gran documentación, lo que permite concentrar el trabajo en el desafío principal de la tesis: la integración de la información visual de las cámaras del ambiente en el sistema de navegación del robot. El TurtleBot3 posee una base diferencial para su desplazamiento, cuenta con una cámara frontal y un LiDAR en la parte superior. El sistema del robot funciona con una Raspberry Pi 4, lo que permite que sea fácil de intervenir e implementar el trabajo de tesis en él. En la Figura 3.11 se puede observar el robot junto a sus sensores y en la Figura 3.12 se pueden observar las dimensiones del robot.

Para la simulación del robot y sus sensores se utilizará el modelo del TurtleBot3 Waffle disponible en la documentación del robot[48] . Junto a esto se utilizarán los paquetes de controladores, sensores y navegación del robot.

Para la localización, se tiene la información de odometría basada en los datos de los motores de la base y se dispone de la información del LiDAR. . En la Figura 3.13 se muestra el robot en el cowork.

Cabe destacar que estos robots modulares, en general, llevan como computador principal dispositivos embebidos o tarjetas de desarrollo. Estas unidades tienen capacidades limitadas, pero su tamaño compacto, bajo costo y bajo consumo energético presentan ventajas para diversos casos de uso. El trabajo de tesis está orientado a este tipo de dispositivos.

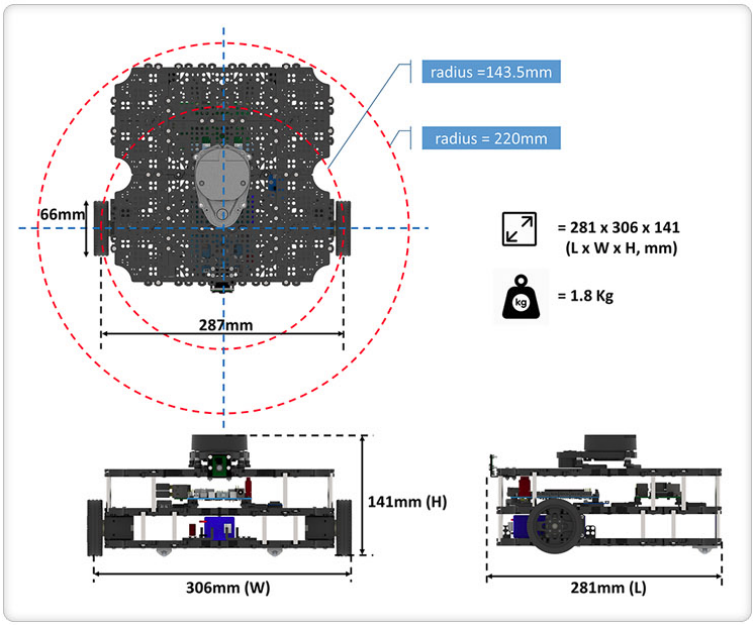

Figura 3.12: Dimensiones Turtlebot3 Waffle Pi. (Extraída de https://emanual.robotis.com).

Figura 3.13: TurtleBot3 Waffle en Cowork.

3.2. Procesamiento información visual e integración

La parte central del trabajo consiste en la generación de información útil para la navegación del robot aprovechando la vista de las cámaras disponibles en el edificio. Para esto, se utilizan nodos de conexión a las cámaras, modelos de detección de objetos livianos basados en aprendizaje profundo, modelos para la transformación de sistemas de referencia y sistemas de calibración semiautomática para los modelos de procesamiento de imágenes.

3.2.1. Módulo de conexión a cámaras

El nodo encargado de conectar la imagen de las cámaras es *ipcamerabridge*. Este nodo de ROS se conecta a las cámaras a través de la librería de procesamiento de imágenes OpenCV [49]. OpenCV permite conectarse a distintos tipos de cámaras, incluyendo cámaras IP a través del protocolo RTSP (Real Time Streaming Protocol) por el puerto 554. Este protocolo está disponible en la gran mayoría de cámaras de seguridad IP, en grabadores de video digitales o Digital Video Recorder (DVR) y grabadores de video en red o Network Video Recorder (NVR). Dependiendo del fabricante, pueden variar los parámetros para acceder en cada caso. Este nodo se encarga de solicitar imágenes a las cámaras IP de la escena donde está el robot cada 5 segundos o cuando el robot lo solicite como apoyo. Al conseguir la imagen, el nodo la publica en un tópico de ROS por cada cámara, permitiendo que otros nodos de ROS se suscriban a las imágenes para su procesamiento. Para el uso de este nodo, solo se debe tener un usuario de la cámara con autorización para utilizar ese puerto, no se requiere tener la cuenta de administrador necesariamente. Distintas empresas en la industria ofrecen servicios de video analítica a través de esta conexión poco invasiva al sistema de seguridad. En el caso de simulación, este nodo se define como una función identidad, que se suscribe al tópico de gazebo y publica en el mismo tópico del caso con el robot real, con tal de mantener la arquitectura principal del sistema.

3.2.2. Detección de objetos: robot y obstáculos

Para la detección del robot en la escena, se tiene un nodo de ROS encargado de suscribirse al tópico de las imágenes de cada cámara. Una vez obtenida la imagen, se utiliza un detector liviano que corre en la CPU del robot, optimizado para hardware limitado. Se realizó la evaluación de distintos modelos de detección livianos capaces de ser ejecutados por el computador del robot, evaluando precisión y consumo de recursos. Finalmente, se seleccionó YOLO V8 en su versión más pequeña, versión n, la cual es capaz de ejecutarse en la CPU del robot mostrando buena precisión.

Para el entrenamiento del modelo liviano, se utiliza el método profesor-estudiante, utilizando imágenes de cada escenario de prueba y generando las etiquetas con el modelo grande o profesor ejecutándose dentro de un PC con GPU. El entrenamiento del modelo estudiante se realiza con un modelo liviano preentrenado en MS COCO [50], con una cantidad aproximada de 2000 imágenes etiquetadas por el profesor y 300 épocas de entrenamiento. Los detalles del entrenamiento y parámetros se presentan en la sección de resultados.

Para el modelo profesor se utilizó la versión L de YOLO V8, después de evaluar el entrenamiento del estudiante usando los modelos L y XL. Aunque hoy en día se tienen grandes bases de datos de imágenes de personas y muebles en datasets públicos como MS COCO, no existen imágenes etiquetadas de robots para su detección. En este trabajo se crea una base de datos específica para el robot seleccionado, TurtleBot3 Waffle, para su detección tanto en el mundo real como en el simulador. Este dataset generado permite el entrenamiento del modelo profesor, que luego se usa para reentrenar el modelo estudiante en el ambiente acotado de cada cámara.

Para la creación de esta base de datos del robot se realizaron grabaciones del robot en movimiento en el simulador Gazebo y en el mundo real en distintos espacios, entre ellos un departamento y distintos espacios de un cowork. Estas grabaciones se realizaron con cámaras IP WiFi montadas en distintos lugares y con un smartphone. Posteriormente, se realizó un muestreo sobre los fotogramas del video para generar las imágenes de la base de datos. Finalmente, se realizó el etiquetado de las imágenes y se guardaron en el formato de YOLO. Este formato corresponde a archivos de texto con el mismo nombre que las imágenes, y dentro del archivo, en cada fila, se guarda la información en el formato [Clase, x, y, w, h], donde:

- Clase: Corresponde a la clase en un valor numérico entero respecto a la lista de clases totales.
- x: Coordenada x dentro de la imagen del centro del *bounding box*.
- y: Coordenada y dentro de la imagen del centro del *bounding box*.
- w: Ancho del *bounding box*.
- h: Alto del *bounding box*.

Los valores de x, y, w y h están normalizados entre 0 y 1.

La cantidad de imágenes totales etiquetadas para el dataset asociadas a la clase turtlebot corresponde a 3000, donde 1000 son del simulador y 2000 de escenarios reales.

En la sección 3.3 se explica en detalle el método de calibración automática para el detector de objetos.

Teniendo el modelo liviano ya ajustado al escenario de operación, este se despliega en el nodo de ROS, procesando las imágenes de las cámaras y detectando los objetos de interés.

En la Figura 3.14 se puede observar como el sistema en simulador detecta a las personas conectándose a la vista de las cámaras.

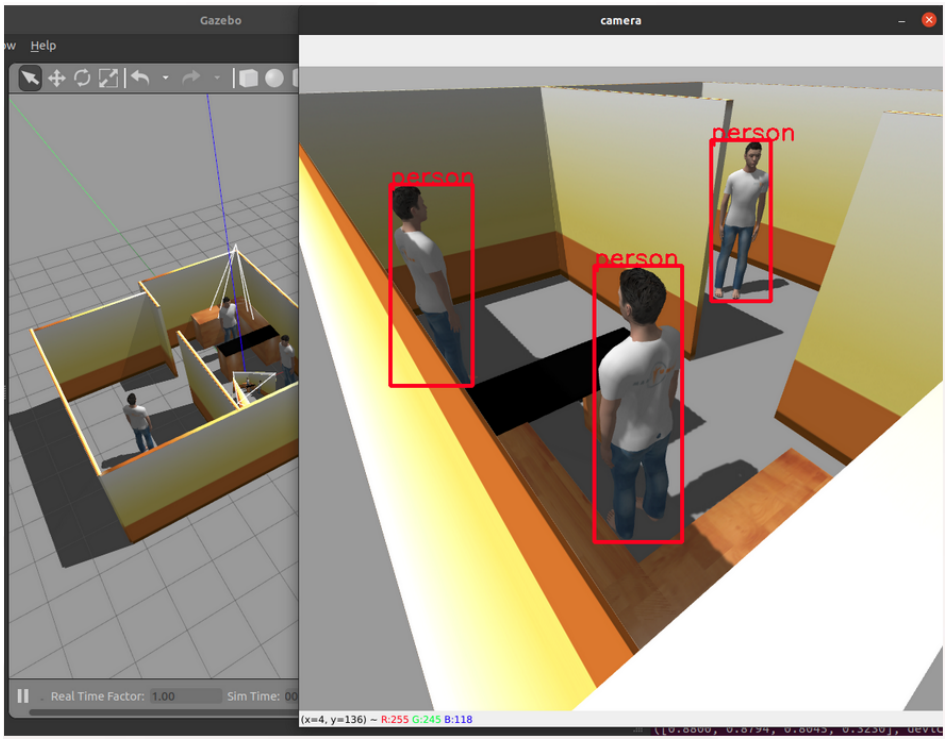

Figura 3.14: Mapa de simulación S: Departamento. Detección de personas.

Una vez generadas las detecciones por el modelo liviano, el nodo de ROS procede a publicar una lista de detecciones 2D a través de un tópico de ROS diseñado para esto. Este tópico contiene la información de las detecciones por cada cámara y permite que otros nodos utilicen esta información. El mensaje de ros utilizado para publicar la información de las detecciones es `Detection2DArray` y tiene los campos presentados en Listado 3.1:

Listado 3.1: Campos del mensaje Detection2DArray en ROS

```
# Detection2DArray.msg

Header header  # Header timestamp and frame_id
Detection2D[] detections  # Array of Detection2D messages

---

# Detection2D.msg

ObjectHypothesisWithPose[] results  # Array of detection hypotheses
BoundingBox2D bbox  # 2D bounding box of the detection
---

# ObjectHypothesisWithPose.msg

uint32 id  # Class ID
float64 score  # Confidence score
Pose2D pose  # 2D pose of the detected object
---

# BoundingBox2D.msg
```

```
22
23 float64 center_x  # Center of the bounding box (x)
24 float64 center_y  # Center of the bounding box (y)
25 float64 size_x  # Width of the bounding box
26 float64 size_y  # Height of the bounding box
27 ---
28
29 # Pose2D.msg
30
31 float64 x  # X position
32 float64 y  # Y position
33 float64 theta  # Orientation
```

En el caso de la simulación, este tipo de proceso puede optimizarse ya que se pueden posicionar los objetos aleatoriamente a través de los servicios de ROS y Gazebo para la posición de los modelos del simulador. Esto permite tener un dataset con una mejor distribución de datos y representatividad del espacio completo.

3.2.3. Estimación de pose de objetos en mapa

Para incorporar la información de las detecciones en el sistema de localización y navegación, es necesario tener la posición de estos objetos (robot, personas, sillas) respecto al sistema de referencia del mapa. ROS proporciona un servicio de transformadas de sistemas de referencia; sin embargo, este servicio de transformadas funciona con cámaras de profundidad, ya que traslada posiciones en 3D a otros sistemas de referencia en 3D, como el mapa.

Dado que no se tiene información de profundidad para determinar una posición 3D respecto a la cámara, se propone utilizar la detección 2D generada por el nodo de detección sobre la imagen de las cámaras CCTV. Utilizando la información de las detecciones, se aplica un modelo de regresión para estimar la posición de los objetos respecto al sistema de referencia del mapa. Se considera que estos objetos se encuentran al nivel del piso, lo cual es válido para las clases robot, persona y silla.

Para la estimación de poses, se dispone de un nuevo nodo de ROS que se suscribe al tópico de detecciones generado en la sección anterior. Se propone realizar una regresión que, para cada cámara y cada tipo de objeto, reciba como entrada la posición del bounding box con sus dimensiones y obtenga como salida la posición (x, y) respecto al mapa del robot.

$$[x_{mapa}, y_{mapa}] = funcion_{regresion}([x_{det}, y_{det}, w_{det}, h_{det}]) \tag{3.1}$$

La coordenada z se considera igual a 0, dado que se asume que los obstáculos están en contacto con el piso.

Para la generación de la base de datos a utilizar en el entrenamiento de la regresión, se plantea el uso del mapa generado por el robot, una simulación simple en Gazebo y el módulo de detección de objetos de la sección anterior. Posterior al proceso de mapeo del robot, donde se define la forma del mapa del ambiente de operaciones utilizando sus sensores, se emplea

este mapa para generar un ambiente de Gazebo automático, donde se levantan paredes en los contornos de las habitaciones detectadas. Una vez generado este mapa de Gazebo automático, se agregan las cámaras en las posiciones del mapa real. Se procede a ubicar modelos de cada clase de objeto en posiciones aleatorias en el mapa y, a través del módulo de detección de objetos, se obtienen los *bounding box* asociados a cada modelo. Dado que en Gazebo se tiene la posición de los objetos generados aleatoriamente, se obtiene el par de entrada y salida de la regresión para su entrenamiento.

Para la regresión, se prueban distintos tipos de modelos con el fin de evaluar cuál se ajusta mejor al problema planteado, seleccionando el modelo con mejor resultado en los distintos ambientes. Entre los modelos de regresión evaluados están: Regresión Lineal, Regresión por Gradient Boosting, Regresión por Support Vector Machine, Regresión por Random Forest, Regresión por KNeighbors y Regresión por Multi Layer Perceptron.

En la sección de resultados se presentan las métricas de los entrenamientos realizados y la selección del mejor modelo.

3.2.4. Integración posición del robot en sistema de localización

La estimación de la posición del robot respecto al mapa, generada por los módulos anteriores, se utiliza para mejorar el sistema de localización del robot. Esto es especialmente relevante en entornos donde no se disponen de buenos puntos de referencia o en áreas extensas donde el alcance del LiDAR no es suficiente para detectar dichos puntos. Se busca incorporar la información de posición generada por las cámaras CCTV para complementar el proceso de localización.

El sistema de localización del robot se lleva a cabo mediante el paquete ROS amcl[51] (Adaptive Monte Carlo Localization), una variación del algoritmo de filtro de partículas que estima la posición y orientación del robot con una cantidad de partículas que varía dependiendo de la incertidumbre, lo cuál es computacionalmente más eficiente.

La nueva información de posición se integra en el sistema de localización como una medición adicional para actualizar el sistema de estimación con un nuevo *prior*, con el objetivo de reducir la incertidumbre y la dispersión de las partículas. Esta integración se realiza publicando una nueva posición del robot en un tópico específico, el cual se utiliza para proporcionar manualmente una posición de referencia al robot al iniciar los nodos de localización y navegación.

El algoritmo AMCL se puede descomponer en las siguientes etapas (explicadas previamente en la sección de Antecedentes):

1. Inicialización
2. Predicción (Movimiento)
3. Actualización
4. Re uestreo

5. Adaptación de partículas
6. Estimación

La nueva información de posición se integra durante la etapa de Actualización, donde se generan nuevas partículas con un alto peso asociado a la nueva posición de referencia, y se ajustan las ponderaciones de las partículas existentes en función de su proximidad a esta nueva referencia.

Este proceso se modela con la siguiente ecuación:

$$w_i' = w_i \cdot p(z|x_i') \tag{3.2}$$

donde $p(z|x_i')$ es la probabilidad de observar z dado el estado de la partícula x_i'.

En la etapa de Re muestreo, se eliminan las partículas con menor peso o aquellas que difieren significativamente de la nueva posición de referencia. Una vez hecho esto, el algoritmo completa el ciclo y repite los pasos normalmente, actualizando las partículas según el proceso original.

Este proceso generalmente se realiza de forma manual al iniciar las tareas del robot. El filtro de partículas se inicializa con un estado o posición en el origen (0,0,0) y genera partículas con una distribución uniforme dentro de las posiciones permitidas del mapa. Posteriormente, se puede proporcionar manualmente una posición estimada de inicio del robot en el mapa, la cual sigue el proceso de actualización con nuevas partículas asociadas a la referencia.

El sistema de apoyo visual aprovecha el tópico utilizado en este proceso para proporcionar una nueva referencia sin necesidad de intervención humana. El sistema de apoyo visual se activa automáticamente cuando la posición detectada por la cámara difiere significativamente de la posición estimada por AMCL.

Cuando el robot se encuentra dentro del campo de visión de las cámaras IP, su posición estimada por el sistema de cámaras se compara con la posición determinada por el sistema de localización basado en AMCL. Si la diferencia entre ambas posiciones supera un umbral de error predeterminado, el sistema de apoyo visual envía una nueva referencia de posición al filtro de partículas, generando una actualización de la distribución de las partículas considerando esa referencia. Esto es útil en casos donde el robot confunde un sector del mapa con otro, proporcionando una localización errónea que afecta el proceso de navegación. Estos casos generalmente ocurren en espacios con distribuciones de muros y muebles muy similares.

El algoritmo de localización se basa en un modelo de cadena de Markov. Cuando se obtiene nueva información, el sistema actualiza la estimación de la posición del robot según el siguiente modelo:

$$p(x_t|z^t, u^t) = \text{const.} \cdot p(z_t|x_t) \int p(x_t|u_t, x_{t-1}) \cdot p(x_{t-1}|z^{t-1}, u^{t-1})\, dx_{t-1} \tag{3.3}$$

donde $p(x_t|z^t, u^t)$ corresponde a la posición del robot que estamos buscando, $z^t = z_0, ..., z_t$

corresponde a las mediciones del sensor, en este caso el LiDAR, de los puntos de referencia, y $u^t = u_0, ..., u_t$ corresponde a las acciones de control relacionadas con el movimiento del robot.

El uso de la referencia entregada a AMCL se puede entender como la actualización forzada en el estado anterior para la nueva estimación, ya que al ser un proceso de Markov se cumple que $p(x_t|u_t, x_{t-1})$, el estado t está definido por el estado anterior y la acción asociada a la transición. Por lo tanto, se define un nuevo valor $p(x_0|z^0, u^0) = p(x_0)$ como condición previa cuando se detecta un error considerable, aprovechando la detección del sistema de apoyo visual.

La pose del robot también considera un ángulo de orientación respecto al mapa. Se evaluó la posibilidad de estimar este ángulo utilizando la información visual, pero se determinó que sería necesario un modelo con detección de *keypoints* para realizar una estimación precisa de la orientación de los objetos en un plano 3D. Los modelos de detección de *keypoints* son más complejos y grandes para ser ejecutados en la CPU del robot. Además, la tasa de detección de *keypoints* es menor que la de detección de objetos. Dado lo anterior y considerando que el objetivo de este trabajo es generar un sistema de apoyo que sea capaz de funcionar sin computación externa, se descartó la estimación de la orientación del robot y los objetos en este trabajo. La detección de los obstáculos como tal no se ve afectada por la falta de orientación en su ubicación en el mapa. Sin embargo, la falta del parámetro de orientación para la posición del robot en el sistema de localización sí es relevante, por lo que se aplica un comportamiento de recuperación en el robot. Durante la actualización de la referencia en el filtro de partículas, ya se cuenta con la ubicación del robot más cercana a la real, pero sin orientación. Por lo tanto, se aplica un comportamiento de recuperación que consiste en girar hasta que los sensores del robot corrijan el ángulo de orientación.

A modo de representar el flujo de este algoritmo se presentan las etapas en pseudo código en Algoritmo 1.

Algoritmo 1 Algoritmo de Filtro de Partículas Modificado para Localización de Robot

Input: Conjunto inicial de partículas $\{x_i, w_i\}$, entrada de control u_t, observación z_t
Output: Conjunto de partículas actualizado $\{x'_i, w'_i\}$, posición estimada del robot $\hat{x}_t$
Inicializar conjunto de partículas $\{x_i, w_i\}$ con distribución uniforme.
while robot está operativo **do**
    for cada partícula x_i **do**
        Predicción: Aplicar modelo movimiento $x'_i = f(x_i, u_t)$
        Actualización: Calcular peso ponderado $w'_i = w_i \cdot p(z_t|x'_i)$
    end for
    if Diferencia entre posición estimada y detección por cámara mayor a umbral **then**
        Incorporar posición de referencia externa, generando nuevas partículas y ajustando pesos
    end if
    Normalizar pesos $w'_i \leftarrow \frac{w'_i}{\sum w'_i}$
    Re muestreo: Re muestrear partículas basadas en w'_i para obtener un nuevo conjunto $\{x'_i, w''_i\}$
    Estimación: Estimar la posición del robot $\hat{x}_t = \sum w'_i \cdot x'_i$
end while

En la Figura 3.15 se puede observar al robot perdido dentro del mapa, con las partículas distribuidas en toda la imagen (puntos verdes), esto es previo a recibir apoyo de las cámaras. En la Figura 3.16 se presenta el estado posterior a recibir una estimación de posición de las cámaras, lo que le da al robot un nuevo $p(x_0|z^0, u^0) = p(x_0)$ y actualiza las partículas, pero con una orientación incorrecta dado que el sistema de detección solo captura la posición. Se observa que las partículas comienzan a concentrarse alrededor del robot dada la generación de nuevas partículas y re muestreo posterior. En la Figura 3.17 finalmente se muestra el estado del robot posterior a dar una vuelta de 360 grados para disminuir la incertidumbre y encontrar su orientación. Se observa que el filtro de partículas es capaz de centrarse en la posición del robot luego de los primeros pasos.

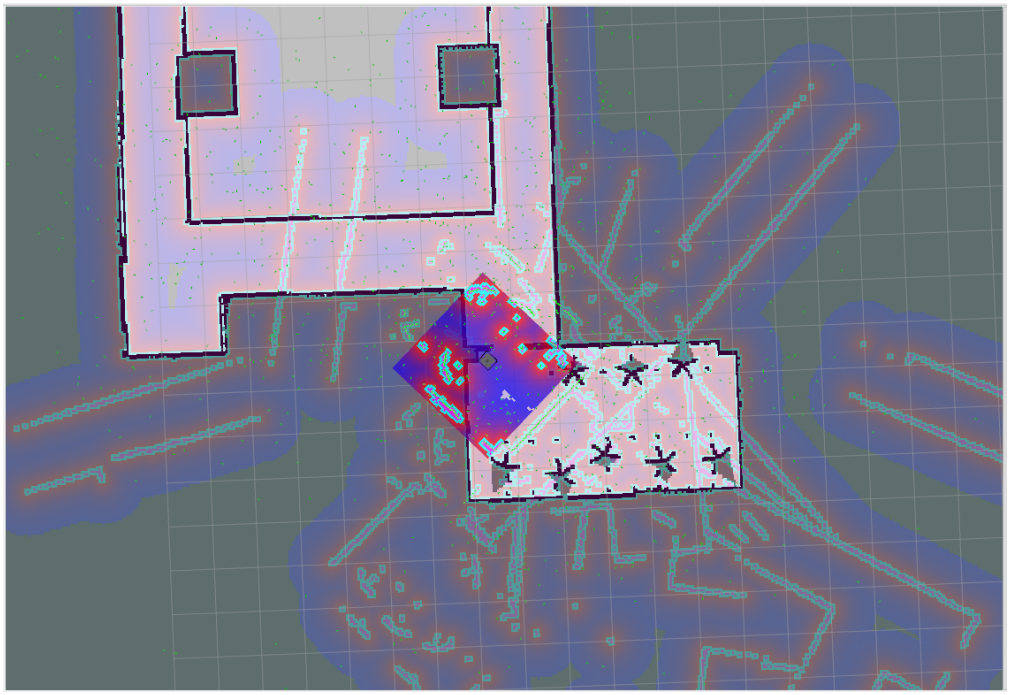

Figura 3.15: Localización con apoyo visual. Estado previo a recibir apoyo de cámaras.

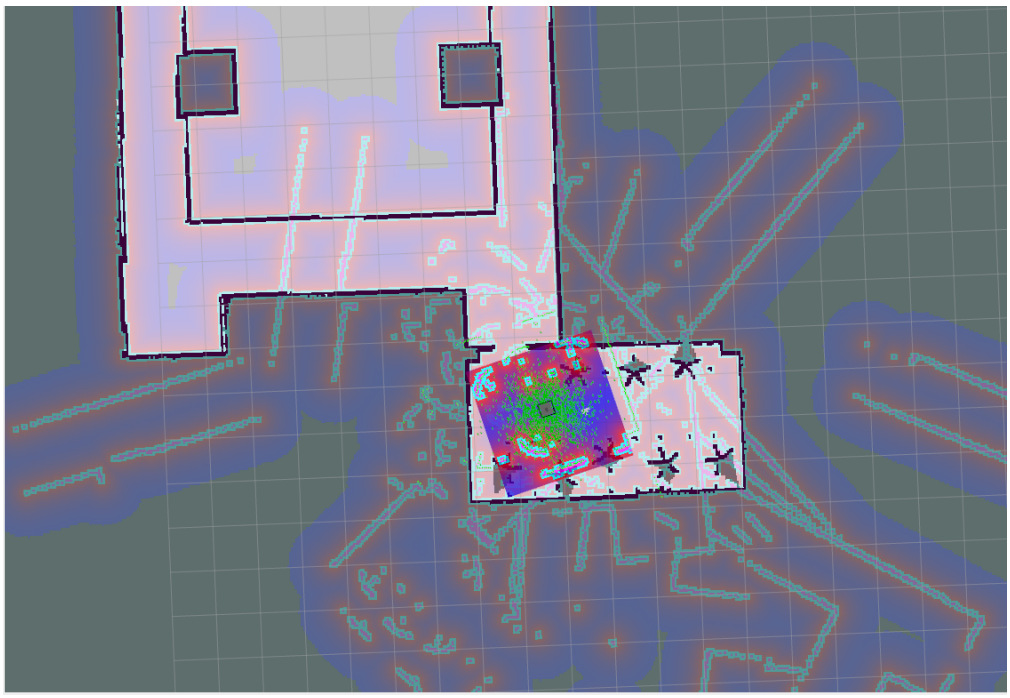

Figura 3.16: Localización con apoyo visual. Estado posterior a recibir apoyo de cámaras.

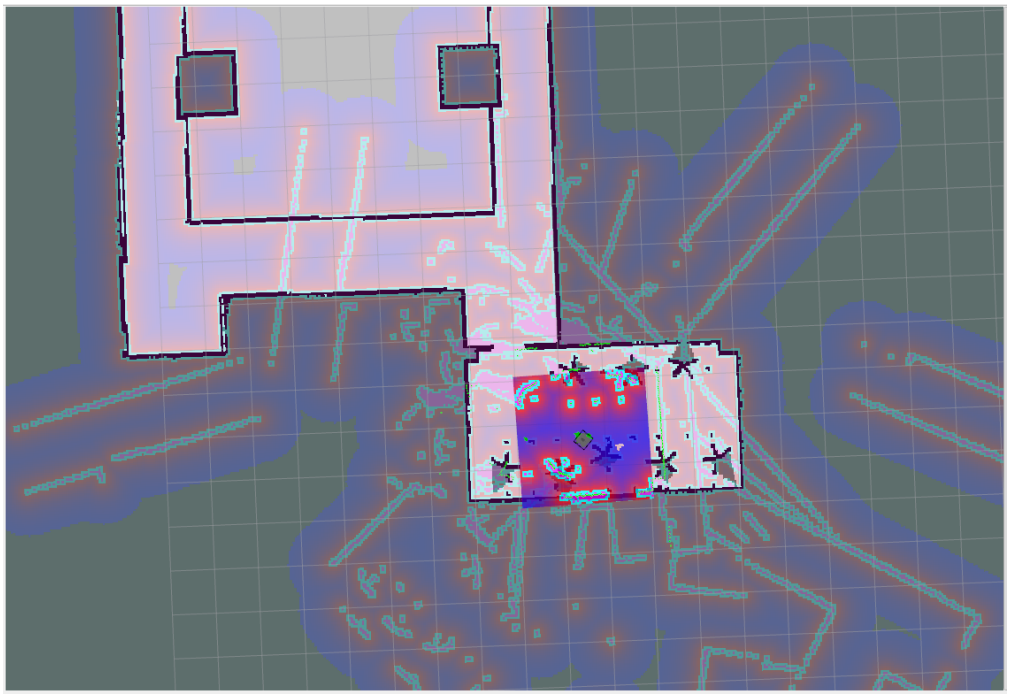

Figura 3.17: Localización con apoyo visual. Estado posterior a recibir apoyo y ejecutar una vuelta de 360 grados como comportamiento de recuperación.

3.2.5. Integración obstáculos en mapa navegación

El sistema de detección mediante las cámaras CCTV busca proporcionar información adicional de obstáculos fuera del campo de percepción del robot, que está limitado por el alcance de sus sensores. Se decide integrar esta información detectada por las cámaras en el Mapa de Costos Global del robot. Esto es de gran utilidad, dado que el robot, al calcular la trayectoria, solo utiliza la información obtenida por sus sensores en la última visita a los distintos lugares del mapa. Como resultado, no puede detectar cambios en la posición de los obstáculos que podrían afectar o bloquear su trayectoria hasta que llega al lugar y lo verifica con sus propios sensores.

Para integrar la información de la detección de obstáculos de las cámaras, se genera un LiDAR virtual, el cual publica en un tópico de ROS un mensaje tipo `LaserScan` con los mismos campos que un LiDAR. El nodo encargado de esto se suscribe a los nodos de detección de obstáculos, después de realizar la regresión y tener la ubicación de los obstáculos respecto al mapa. Teniendo la información de los obstáculos, tipo y posición, el nodo encargado de la integración entrega la información marcando en el LiDAR virtual detecciones de obstáculos como si fueran detectados por este LiDAR en las mismas posiciones detectadas por las cámaras. Se toma la decisión de utilizar un LiDAR virtual, dado que el paquete de navegación por defecto de ROS basa su navegación en este tipo de sensor, por lo que la integración es directa.

Se ubica este sensor virtual en el origen del mapa del robot y, posteriormente, recibe la posición de los obstáculos detectados respecto al mapa en coordenadas (x, y, z) y lo transforma a una posición radial de ángulo y distancia (ϕ, ρ), dado que el láser hace una medición de distancias para distintos ángulos desde su origen. Al tener el ángulo y distancia del objeto respecto al origen, se procede a buscar el punto del láser virtual más cercano a ese ángulo para asignarle el valor de distancia, marcando así una detección de obstáculo en el mapa. Para considerar el tamaño del obstáculo, se agregan más puntos del LiDAR con la misma distancia detectada al obstáculo; esto ayuda a indicar mejor las dimensiones de la zona de colisión al momento de cargar esto en el mapa de costos global del robot. El nodo de ROS encargado de simular este sensor publica el mensaje con esta información y es agregado como segunda fuente de información de obstáculos en el archivo de configuración del mapa de costos global del robot.

Al inicializar el sistema de navegación, los obstáculos indicados por el LiDAR virtual son considerados en el mapa de obstáculos como un `LETHAL Object`. La construcción del mapa de costos del robot se genera usando la posición de estos `LETHAL Object` y realizando un proceso de inflación, el cual define que el punto donde está el obstáculo es una posición no permitida y los puntos aledaños se les asigna un costo en función de la distancia al obstáculo. La incorporación de esta información en el mapa de costos global se hace con el fin de mejorar la estimación inicial de la trayectoria de navegación del robot, evitando elegir rutas imposibles o tener que recalcular la ruta una mayor cantidad de veces. Así, la información visual incorporada es un prior útil para su navegación y, a medida que el robot tiene en el alcance del LiDAR real los obstáculos detectados, se corrige la posición junto al mapa de costos generada por la inflación de estos obstáculos.

En la Figura 3.18 se muestra como el sistema de apoyo visual detecta obstáculos, en este

caso personas, que están caminando dentro del mapa. Los puntos rojos o marcadores indican la posición que el sistema de apoyo visual estimó para los obstáculos. Se puede observar que el marcador de arriba está dentro del rango de los sensores del robot, pero el marcador de abajo está fuera del alcance por los muros. Usando la cámara de esa habitación se estima la posición y zonas de caminata de la persona. Inmediatamente al lado del marcador se observar zonas prohibidas por el mapa de costos del robot, esto debido a que las personas han estado circulando por esas trayectorias. El sistema de navegación detecta las zonas prohibidas y aumenta el costo de las zonas cercanas para luego ser considerados en la trayectoria de navegación.

A modo de representar el flujo de este algoritmo se presentan las etapas en pseudo código en Algoritmo 2.

Algoritmo 2 Integración de Obstáculos en el Mapa de Navegación

Input: Lista de obstáculos detectados $\{obst_1, obst_2, \ldots, obst_n\}$, Mapa de costos M, Parámetros de actualización del mapa
Output: Mapa de costos actualizado M'
Inicializar LiDAR Vitual para simulación de obstáculos
while el robot está operativo **do**
    Detección: Obtener posición de los obstáculos en el entorno usando la cámara y el sistema de detección
    Convertir las posiciones de los obstáculos al sistema de coordenadas del mapa de navegación
    for cada obstáculo $obst_i$ en la lista **do**
        Transformar posición del obstáculo a sistema de referencia LiDAR virtual
        Actualización: Publicar obstáculo en tópico de LiDAR virtual
    end for
    Realizar una expansión del área del obstáculo en el mapa para considerar márgenes de seguridad
    Recalcular las rutas de navegación usando el mapa actualizado M'
    if el tiempo de recalculado es aceptable **then**
        Actualizar la ruta de navegación del robot
    else
        Mantener la ruta actual hasta la próxima actualización
    end if
    Enviar el nuevo mapa de costos M' al sistema de navegación del robot
end while

La hipótesis de este trabajo de tesis plantea que la incorporación de esta información en conjunto a la información de localización ayuda a disminuir los tiempos promedios de navegación del robot, mejorando la tasa de éxito de alcanzar los *goals* planteados.

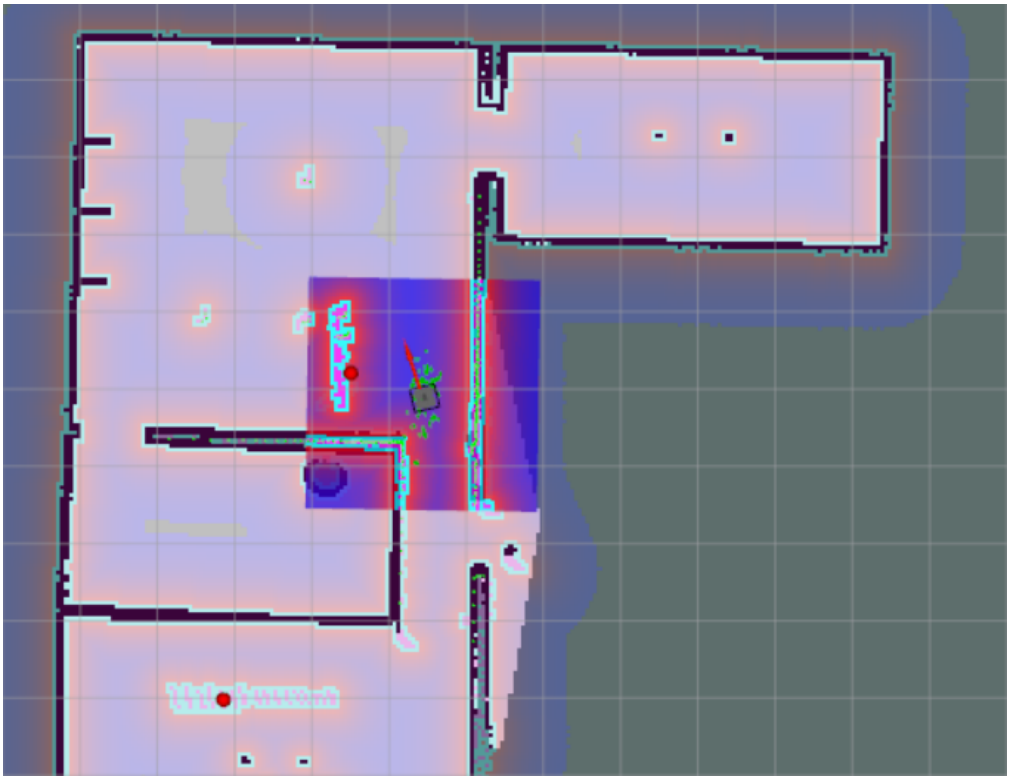

Figura 3.18: Vista de mapas de costos en RViz. Markers de posición de obstáculos en rojo.

3.3. Calibración de modelo de detección liviano

Dado que el sistema propuesto debe ejecutarse en los recursos limitados del robot, se plantea usar modelos de detección livianos y entrenamiento por método profesor-estudiante (o destilación).

La hipótesis plantea que un modelo liviano, capaz de ejecutarse con bajos recursos en el robot, puede lograr una buena tasa de precisión en un ambiente acotado con un entrenamiento adicional automático en la escena. Es decir, se aprovechará que las condiciones del ambiente de operación del robot son limitadas para ajustar el modelo y lograr una mayor precisión. Para evitar el proceso de etiquetado manual que conlleva el reentrenamiento, asociado a una gran cantidad de horas hombre, se utilizan imágenes del ambiente etiquetadas por un modelo pesado denominado modelo profesor (teacher), previamente entrenado con una gran cantidad de imágenes de distintos datasets. Este modelo, diseñado para funcionar en distintos ambientes y en GPU, logra mejores resultados comparado con el modelo liviano que se ejecuta en el robot. Por lo tanto, puede generar etiquetas para corregir los errores del modelo liviano, denominado modelo estudiante (student).

El modelo profesor es un YOLO V8 en su versión grande (YOLO V8-L), mientras que el modelo estudiante es la versión más pequeña (YOLO V8-n). Ambos modelos fueron preentrenados con el dataset COCO. En las Figuras 3.19 y 3.20 se realiza una prueba rápida para visualizar cualitativamente el desempeño de ambos modelos. El modelo profesor y estudiante no han visto las imágenes de prueba antes. Se observa que el modelo profesor logra una mayor cantidad de detecciones, aunque comete errores como falsos negativos y positivos, mostrando

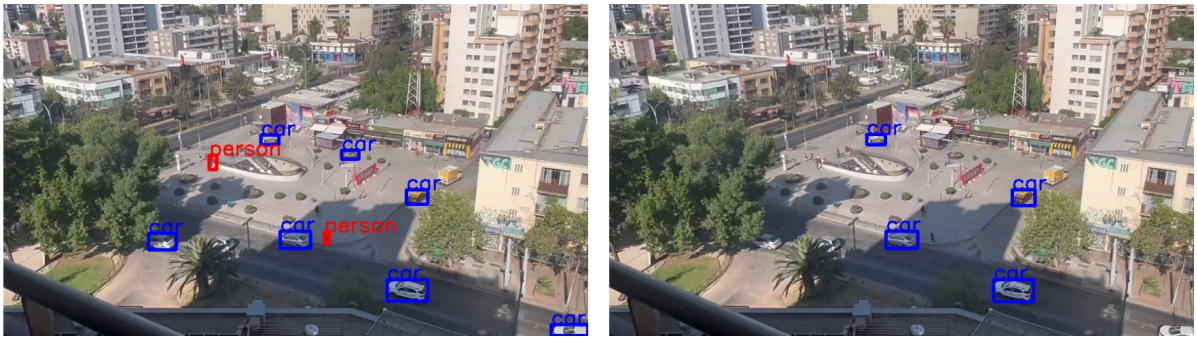

Figura 3.19: Video de Prueba 1. A la izquierda detecciones de modelo profesor. A la derecha detecciones de modelo estudiante.

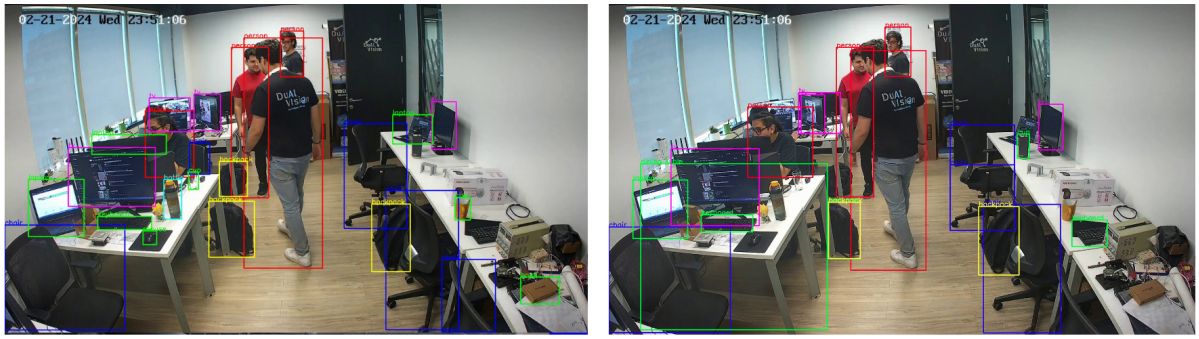

Figura 3.20: Video de Prueba 3. A la izquierda detecciones de modelo profesor. A la derecha detecciones de modelo estudiante.

que el modelo estudiante comete una mayor cantidad de errores. Esto indica que el modelo estudiante puede aprender del desempeño del modelo más pesado.

Para este trabajo de tesis se realizaron pruebas iniciales de este método con videos de distintas cámaras usando las clases del dataset COCO para validar su efectividad.

Para el objetivo de esta tesis se consideraron solo 3 clases de objetos: robot, personas y sillas. Esto se debe a que se requiere la detección del robot para su estimación de posición, apoyando así al sistema de localización. Además, para la tarea de navegación, se busca tener información sobre obstáculos que están fuera del campo sensorial del robot. Dado que el robot ya ha realizado el mapeo del espacio, los obstáculos cuya posición no se conoce corresponden a obstáculos móviles. Los obstáculos móviles más comunes en espacios de trabajo o domicilios son las personas y las sillas, que son muebles que comúnmente se mueven. Finalmente, considerando la detección de objetivos dentro del edificio, se incluye el caso en que se deba buscar a una persona en particular. Dado lo anterior, se definen estas clases para esta tesis.

Considerando que el modelo grande o profesor ya se encuentra entrenado previamente con buenas métricas de precisión sobre las clases objetivo, el procedimiento realizado para el sistema de ajuste del modelo liviano consta de los siguientes pasos:

1. **Generación de Base de Datos:** Para la generación de bases de datos que se usarán en el proceso, se debe grabar el proceso de mapeo del robot para tener imágenes del robot en distintas posiciones del campo de visión de la cámara. Adicionalmente, se graba por

algunas horas el funcionamiento del lugar y el movimiento de los objetos. En caso de querer dar mayor calidad a los datos, se pueden movilizar objetos artificialmente. A partir de estos videos se generan muestras de imágenes y se obtiene la base de datos. En general, para las pruebas se utilizó una cantidad de imágenes que varió entre 500 y 2000.

2. **Generación de etiquetas:** El modelo grande o profesor realiza inferencias sobre las imágenes de la base de datos generadas para el ajuste. Los resultados de las inferencias se guardan como etiquetas. Este proceso de inferencia se realiza en un PC o servidor con GPU para acelerar el proceso.
3. **Entrenamiento del modelo estudiante:** El modelo estudiante se entrena con estas etiquetas para aprender del desempeño del modelo profesor. En esta etapa, se ajusta el modelo estudiante para tener un buen desempeño solo en la vista de las cámaras de donde se capturaron las imágenes. Esto facilita el proceso de aprendizaje, pero no permite que generalice de buena manera si se aplica en nuevas cámaras. Este proceso de entrenamiento se realiza en un PC o servidor con GPU para acelerar el proceso.
4. **Despliegue del modelo estudiante:** Una vez entrenado el modelo estudiante en el equipo con GPU y una vez que se hayan validado las métricas de desempeño de este en validación, se procede a cargar el nuevo modelo liviano en el robot para su ejecución en tiempo real y así aportar al sistema de navegación.

Para la elección de parámetros del proceso de entrenamiento se hicieron distintas pruebas, principalmente en cantidad de imágenes, épocas y tamaño de batch. Los parámetros comunes utilizados en entrenamiento del modelo estudiante son:

- **Cantidad de clases:** 3
- **Cantidad de imágenes:** 500-2000
- **Épocas de entrenamiento:** 300
- **Tamaño de batch:** 2
- **Learning rate:** 0.01
- **Weight decay:** 0.0005
- **Optimizador:** SGD, Adam

Este proceso no requiere etiquetado manual, lo cual es una gran ventaja para su ejecución. Las métricas y resultados del proceso de entrenamiento por método profesor-estudiante se presentan en la sección de resultados.

Capítulo 4

Resultados y Análisis

Para la realización de pruebas y la extracción de resultados, se definieron distintos tipos de experimentos y métricas a evaluar en cada etapa del proceso.

4.1. Método de entrenamiento profesor-estudiante

Para la primera exploración se tomaron videos de pruebas que los modelos no conocían y se extrajeron imágenes cada 1 segundo. Sobre estas imágenes se realizó la inferencia usando el modelo grande o profesor para la generación de etiquetas.Los modelos profesor utilizados en esta exploración fueron YOLOv8L y YOLOv8X, mientras que el modelo estudiante fue YOLOv8n.

4.1.1. Métricas

Dado que se está realizando la destilación de aprendizaje para la tarea de detección de objetos se utilizará como métrica la Precisión Media Promedio o *Mean Average Precision* (mAP). La métrica *Mean Average Precision* (mAP) es ampliamente utilizada en tareas de detección de objetos para evaluar la precisión de los modelos. Esta métrica combina la *Precision* (Precisión) y el *Recall* (Cobertura) en una sola medida, proporcionando una visión integral del desempeño del modelo.

La Precisión se define como la fracción de detecciones verdaderamente positivas (TP) sobre todas las detecciones realizadas (la suma de verdaderamente positivas (TP) y falsamente positivas (FP)):

$$\text{Precisión} = \frac{TP}{TP + FP} \tag{4.1}$$

Por otro lado, la Cobertura se define como la fracción de detecciones verdaderamente positivas (TP) sobre todas las instancias reales en la clase (la suma de verdaderamente

positivas (TP) y falsamente negativas (FN)):

$$\text{Cobertura} = \frac{TP}{TP + FN} \tag{4.2}$$

Para calcular el mAP, primero se obtiene la *Average Precision* (AP) para cada clase. La AP es el promedio de la precisión para cada umbral de *recall* de 0 a 1. En la práctica, esto se realiza calculando la precisión en varios puntos de *recall* y luego promediándolos.

Finalmente, el mAP se obtiene promediando las APs de todas las clases. Si hay N clases, el mAP se calcula de la siguiente manera:

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}_i \tag{4.3}$$

donde AP_i es la Average Precision para la clase i.

El mAP es una métrica útil porque proporciona una única cifra que resume tanto la precisión como la cobertura del modelo en todas las clases, permitiendo una comparación sencilla entre diferentes modelos.

4.1.2. Experimentos

Se utilizaron vistas de distintas cámaras con el fin de evaluar la capacidad de este método para mejorar la precisión en modelos livianos sin intervención humana.

En la Tabla 4.1 se presenta un resumen de información de los videos obtenidos de distintas cámaras para evaluar el desempeño de la calibración. Se indica el número de imágenes utilizadas en la calibración y el tiempo que tomó el proceso. La medición de tiempo de ajuste o calibración del modelo liviano se realizó utilizando YOLOv8L como profesor.

Tabla 4.1: Resumen información videos de cámaras de prueba profesor-estudiante.

Caso	Cámaras	Imágenes	Tiempo calibración [m]
1. cámara celular balcón	1	66	12.15
2. cámaras seguridad municipal	4	546	78.32
3. cámara seguridad oficina	1	251	29,31

Para las pruebas realizadas se utilizaron las variables e hiperparámetros indicados en la Tabla 4.2.

Se procedió a realizar las pruebas de transferencia de conocimiento en los tres conjuntos de datos, con los parámetros antes indicados y para dos modelos profesor distintos.

En la Tabla 4.3 se presentan los resultados al aplicar el método en distintas vistas de cámaras usando YOLOv8L como modelo profesor. Se muestra el desempeño del modelo pro-

Tabla 4.2: Variables e hiper parámetros utilizados en las pruebas.

Variable o Hiper parámetro	Valor
GPU utilizada	Nvidia RTX 2060 [6GB]
Conjunto de entrenamiento	80 %
Conjunto de prueba	20 %
Optimizador	SGD
Función pérdida clasificación	Binary Cross Entropy
Función pérdida regresión	Distribution Focal Loss, Complete IoU
Tasa de aprendizaje	0.01
Early Stopping	Si
Tamaño de batch	4
Épocas	300

fesor entrenado previamente con el dataset COCO, el estudiante base (previo a la destilación) entrenado también con el dataset COCO y finalmente el estudiante final, que ya pasó por el proceso de destilación de aprendizaje usando las etiquetas del modelo profesor. Además, se incluye el tiempo que duró el proceso de destilación de aprendizaje.

Tabla 4.3: Resultados método de calibración usando YOLOv8l y YOLOv8n.

Prueba	mAP Profesor	mAP Estudiante	mAP Estudiante Final	Variación %	Tiempo[m]
Caso 1	0.263	0.102	0.242	+137.25 %	9.72
Caso 2	0.377	0.305	0.338	+10.81 %	57.59
Caso 3	0.723	0.339	0.693	+104.42 %	18.96

En la Tabla 4.4 se presentan los resultados al aplicar el método en distintas vistas de cámaras usando YOLOv8X como modelo profesor.

Tabla 4.4: Resultados método de calibración usando YOLOv8x y YOLOv8n.

Prueba	mAP Profesor	mAP Estudiante	mAP Estudiante Final	Variación %	Tiempo[m]
Caso 1	0.286	0.120	0.279	+132.25 %	12.15
Caso 2	0.339	0.272	0.251	-7.38 %	78.32
Caso 3	0.942	0.339	0.692	+103.13 %	29.31

En las Tablas 4.3 y 4.4 se observa que la red liviana mejora su desempeño tras el proceso de aprendizaje en 5 de los 6 experimentos realizados. En los experimentos donde el modelo profesor corresponde a YOLOv8L, se aprecia que en los tres casos hay una mejora en la métrica mAP. Sin embargo, en un caso, la tasa de mejora es menor. Para los casos 1 y 3, donde se utilizaron imágenes de una sola cámara, se observa una mejora superior al 100 %. En contraste, en el caso 2, con imágenes de cuatro cámaras, la mejora es solo del 10 %. Esto sugiere que el sistema tiene un mejor desempeño en entornos más controlados y con menor variabilidad, lo que concuerda con la hipótesis planteada, donde se discute que el modelo

Figura 4.1: Video de Prueba 3. A la izquierda, detecciones del modelo estudiante antes de la sesión de aprendizaje. A la derecha, detecciones del modelo estudiante después de la sesión de aprendizaje.

liviano, con menor capacidad de generalización, puede alcanzar un desempeño similar al modelo grande o profesor en ambientes semi-estáticos.

En los experimentos con el modelo profesor basado en YOLOv8X, se tiene un comportamiento similar: los casos 1 y 3 muestran una mejora superior al 100 %, aunque con una variación menor que con YOLOv8L. En el caso 2, se observa un desempeño negativo, empeorando el rendimiento del modelo estudiante. Comparando la métrica mAP final entre ambos modelos profesor, YOLOv8L tiene una precisión final menor en el caso 1, una precisión considerablemente menor en el caso 2, y en el caso 3 son similares. Esto indica que, a pesar de que YOLOv8X tiene un mejor desempeño base, esto no se traduce en un mejor proceso de transferencia de aprendizaje, posiblemente debido a diferencias arquitectónicas significativas con YOLOv8N.

Analizando el caso 2, se observa que, aunque la tasa de mejora es menor, el modelo YOLOv8L sigue mejorando el desempeño del estudiante. Sería beneficioso realizar más pruebas con una mayor cantidad de videos, cámaras y escenarios para caracterizar mejor los casos en los que el sistema de transferencia de conocimiento tiene una mejor tasa de mejora. En este experimento, el caso 2 involucra distintas cámaras de vía pública con gran variabilidad en sus escenas, lo cual impacta negativamente el sistema.

En conclusión, este experimento demuestra que el método de transferencia es capaz de mejorar el desempeño del modelo liviano o estudiante utilizando como guía la inferencia del modelo más pesado sin intervención humana.

Para apreciar visualmente el cambio, se muestra en la Figura 4.1 la comparación del modelo estudiante liviano antes y después de la transferencia de aprendizaje. Se puede observar que aumenta la cantidad de detecciones y se elimina algunas detecciones erróneas.

Posterior a la validación de que el sistema logra mejorar la precisión del modelo pequeño, se procede a evaluar el desempeño de los modelos en términos de velocidad de inferencia en distintos tipos de hardware. La métrica utilizada para esta medición es la cantidad de *frames* o imágenes por segundo (FPS) que el modelo es capaz de procesar. En la Tabla 4.5 se observan los resultados de inferencia sobre un conjunto de validación en tres arquitecturas distintas y dos casos para la arquitectura del robot. Los resultados muestran que la diferencia

de precisión es la misma, ya que se usa el mismo modelo, con una disminución del mAP de 4.15 % respecto al modelo profesor basado en YOLOv8L.

La importancia de este experimento es medir la velocidad de inferencia del modelo estudiante en las distintas arquitecturas y comparar este desempeño con el del modelo profesor. Para esta métrica, se observa una mejora de velocidad considerable que varía entre +696.96 % y +928.98 % en los distintos casos. Esto indica resultados favorables para el sistema, donde se obtiene una pérdida de precisión de aproximadamente el 4 %, que aunque considerable, viene acompañada de una mejora significativa en la velocidad.

Tabla 4.5: Tiempo de inferencia en distintas CPU para el modelo profesor y el modelo estudiante posterior al aprendizaje.

CPU	Diff. % mAP	FPS Profesor	FPS Estudiante	Mejora % FPS
AMD® Ryzen 5 3600	-4.15 %	2.53	21.49	+749.41 %
Intel® Core i3 8145U	-4.15 %	0.73	7.14	+878.57 %
TBot3 (RPI4) Standby	-4.15 %	0.07	0.76	+928.98 %
TBot3 (RPI4) ROS bringup	-4.15 %	0.03	0.33	+696.96 %

Estos resultados validan que, utilizando modelos livianos junto al proceso de transferencia de conocimiento desde un modelo profesor, es posible ejecutar la detección sobre las cámaras dentro del robot, permitiendo utilizar el sistema de apoyo visual sin la necesidad de computación externa, como en otros trabajos. La evaluación de tiempos muestra que el robot es capaz de ejecutar el modelo de detección junto a sus condiciones de operación completas, es decir, con los nodos de ROS comunicando la información de los sensores y ejecutando otros algoritmos para sus tareas básicas. El sistema de detección funciona a 0.3 FPS en el caso antes mencionado, indicado como *ROS Bringup* en la Tabla 4.5. Esto demuestra que esta detección de objetos puede operar en tiempo real en paralelo con otras funcionalidades del robot, proporcionando un apoyo efectivo en sus labores mediante consultas a las cámaras para obtener información fuera del alcance de sus sensores.

4.2. Pruebas de integración con el robot

4.2.1. Simulación

Como se planteó al inicio de la metodología, se desarrollaron tres entornos simulados para la implementación del trabajo de tesis y la realización de pruebas. Estos entornos representan posibles áreas de operación para un robot móvil doméstico, como un departamento, una casa o una oficina. Los entornos incluyen diversas habitaciones y cámaras estratégicamente ubicadas, simulando condiciones reales de uso.

En la Tabla 4.6 se presenta un resumen de información de los mapas seleccionados, indicando dimensiones y cantidad de cámaras.

Tabla 4.6: Resumen de información de los mapas de simulación.

Mapa	Área	Cámaras
(S) Departamento	36 m^2	2
(M) TurtleBot House	82 m^2	3
(L) iF Hendaya Cowork - Oficina Dual Vision	150 m^2	3

4.2.2. Escenarios reales

Para la validación del trabajo y la obtención de resultados, se seleccionó un entorno en el cual se instalaron cámaras CCTV conectadas a la red WiFi para la interacción con el robot. Este entorno corresponde a un cowork en Santiago de Chile, que incluye áreas comunes y la oficina privada de una empresa de tecnología. Similar a los entornos simulados, se consideró un espacio de trabajo adecuado para un robot móvil doméstico. El mapa de mayor tamaño simulado corresponde a una representación 3D detallada de este entorno.

En la Tabla 4.7 se presenta un resumen de información de los ambientes reales seleccionados, indicando dimensiones y cantidad de cámaras.

Tabla 4.7: Resumen de información de los mapas reales.

Mapa	Área	Cámaras
(L) iF Hendaya Cowork - Oficina Dual Vision	$\tilde{1}50$ m^2	3

4.3. Detección de objetos

Las pruebas iniciales en los ambientes descritos anteriormente se centran en evaluar la capacidad de los modelos para detectar los objetos seleccionados: robot, persona y silla. Para ello, se realiza una comparación de desempeño entre el modelo profesor, el modelo estudiante previo al proceso de aprendizaje y el modelo estudiante posterior al aprendizaje.

4.3.1. Métricas

Dado que nuevamente se busca evaluar el desempeño en la tarea de detección se utiliza cómo métrica la *Mean Average Precision* (mAP) sobre las tres clases de objetos, en este caso: robot, persona y silla.

4.3.2. Experimentos

En la Tabla 4.8 se presentan los resultados de detección en los ambientes simulados y el escenario real.

Tabla 4.8: Resultados de detección en ambientes simulados y reales.

Prueba	mAP Profesor	mAP Estudiante base	mAP Estudiante Final	Variación %
Mapa Sim S	0.89	0.76	0.86	13.16 %
Mapa Sim M	0.93	0.71	0.88	23.94 %
Mapa Sim L	0.87	0.68	0.84	23.53 %
Mapa Real L	0.73	0.36	0.66	83.33 %

Los resultados de estos experimentos demuestran que el sistema de transferencia de aprendizaje mejora el desempeño del modelo estudiante. A pesar de utilizar 2 o 3 cámaras, se observan mejoras en los cuatro escenarios evaluados. Sin embargo, no se alcanza una tasa de mejora superior al 100 % como en las pruebas de video con una sola cámara. Esto confirma que, a mayor variabilidad en los ambientes, la tasa de mejora disminuye.

Una mejora superior al 13 % en la métrica mAP en tareas de detección es considerablemente buena. En los casos simulados, la tasa de mejora es menor en comparación con el caso real. Los resultados muestran que la precisión del modelo estudiante base en el caso real es significativamente baja en comparación con los casos simulados, debido a la menor capacidad de generalización y menor número de parámetros del modelo más liviano. El mundo real presenta una mayor cantidad de información y variabilidad en comparación con el simulador, lo que también se refleja en la métrica del modelo real.

A pesar de esto, el proceso de transferencia de conocimiento mejora el desempeño del modelo estudiante, mostrando la mayor tasa de mejora en los cuatro casos con un 83.33 % de incremento, en comparación con el 23.94 % del siguiente caso. Esto demuestra que el método de transferencia ayuda a mitigar el problema del *Reality Gap*, como se planteaba en los objetivos de este trabajo.

Estos experimentos proporcionan información relevante para esta tesis. Por un lado, se demuestra que la transferencia de conocimiento es efectiva incluso con tres cámaras distintas y ayuda a enfrentar el *Reality Gap*. Por otro lado, los experimentos indican que la tasa de precisión del modelo liviano es alta; un valor en torno al 70 % en la métrica mAP es adecuado para tareas de detección de objetos, verificando que el sistema puede detectar objetos de interés y generar valor para el robot.

4.4. Estimación de posición

Posterior a la evaluación del desempeño en la detección de objetos, se procede a evaluar el rendimiento del proceso de estimación de posición de los objetos respecto al mapa. Para ello, se evalúan distintos modelos de regresión aplicados a los objetos dentro de los escenarios de pruebas.

4.4.1. Métricas

Para la estimación de la pose del robot y de los obstáculos, se utiliza el error absoluto promedio respecto a la posición real de cada detección. La posición real de cada objeto se obtiene a través de los datos proporcionados por el simulador Gazebo mediante su integración con ROS. Además, se evalúa la velocidad de inferencia, dado que es crítico que todo el sistema funcione en tiempo real.

4.4.2. Resultados

En la Tabla 4.9 se presentan los resultados de la estimación de pose utilizando distintos modelos de regresión. Se evaluaron diversos modelos para los tres tipos de objetos detectados por el sistema, calculando el error promedio en metros y el tiempo de inferencia. La regresión para estos tres tipos de objetos se realizó en los tres mapas simulados, obteniendo métricas representativas de distintos escenarios y orientaciones de cámaras. Las métricas de los experimentos realizados se promedian para cada objeto y modelo con el objetivo de hacer la comparación entre ellos.

Entre los seis modelos evaluados, se observan resultados bastante diversos para ambas métricas, error y tiempo de inferencia. Dado lo anterior, se descartan los modelos que tienen resultados considerablemente inferiores al resto. El modelo de regresión basado en Multi Layer Perceptron (MLP) muestra el menor tiempo de inferencia, pero un gran error en metros, por lo cual no es aplicable para la tarea de estimación de pose. Por otro lado, el modelo de regresión basado en RandomForest, a pesar de tener un error comparable con el resto de los modelos, presenta un tiempo de inferencia considerablemente mayor, por lo cual se descarta por afectar el desempeño en tiempo real.

Tabla 4.9: Resultados de estimación de poses con diferentes modelos.

Modelo regresión	Error robot [m]	Error personas [m]	Error sillas [m]	Inferencia [ms]
Lineal	0.1493	0.1521	0.1545	0.2487
GradientBoosting	1.9977	0.0768	2.0006	0.3871
SVR	**0.0767**	**0.0532**	**0.0814**	0.2690
RandomForest	0.9801	0.3150	0.9863	5.5260
KNeighbors	0.3216	0.1285	0.3274	0.5688
MLP	125.4891	2.4068	84.4957	**0.2454**

Comparando los modelos de regresión lineal, regresión por Gradient Boosting, regresión por Support Vector Machine y regresión por KNeighbors, se pueden ver resultados más cercanos en ambas métricas, logrando un error de estimación en centímetros en general y tiempos de inferencia inferiores a 1 milisegundo. Dado que el objetivo del módulo de regresión es estimar la posición de los objetos detectados con el menor error posible, se selecciona el modelo de regresión basado en Support Vector Machine (SVR). Este modelo logra un error por debajo de los 10 centímetros, siendo un error válido para la tarea asignada y está entre los dos modelos con menor tiempo de inferencia, logrando un tiempo de inferencia promedio de 0.26 milisegundos.

Con este experimento se valida que el uso de modelos de regresión cumple con las necesidades del trabajo de esta tesis, dado que se logra un error de posición bajo y un tiempo de inferencia reducido, asociado a un bajo consumo de la CPU del robot.

4.5. Navegación con información visual

Posterior a las distintas validaciones de los componentes del sistema de apoyo visual, se realizaron pruebas de navegación en ambientes simulados y reales. En estos experimentos, se ubicaron obstáculos en distintas posiciones del mapa, algunos fijos y otros en movimiento. Se generaron objetivos de navegación aleatorios para el robot con el fin de evaluar su desempeño al navegar por los espacios y evitar los obstáculos. Para ello, se comparó el desempeño del robot navegando sin el sistema de apoyo visual con su desempeño utilizando el sistema de apoyo visual.

4.5.1. Métricas

En el problema de navegación se definen dos métricas principales, adoptadas de estudios relacionados sobre tareas de navegación con colaboración con cámaras [7]. La primera métrica es la tasa de éxito (*Success Rate*, SR), que se define como el ratio de trayectorias en las que el robot logra alcanzar el objetivo propuesto.

La tasa de éxito es una métrica fundamental que mide la capacidad del robot para alcanzar su objetivo en diversos escenarios, proporcionando una indicación clara de la efectividad del sistema de navegación [52]. Por otro lado, el tiempo de navegación es esencial para evaluar la eficiencia del sistema, ya que indica cuánto tiempo le toma al robot llegar a su destino [53]. Estas métricas son ampliamente reconocidas y utilizadas en la literatura sobre navegación de robots, como se menciona en trabajos previos [54].

Para esta métrica, se contabilizan tres posibles resultados: navegación exitosa, navegación sobre tiempo límite y navegación abortada. Esta métrica se define como:

$$Success\ Rate\ (SR) = \frac{Trayectorias\ con\ navegacion\ exitosa}{Trayectorias\ evaluadas} \tag{4.4}$$

donde la suma de navegación exitosa, navegación sobre tiempo límite y navegación abortada corresponde a las trayectorias evaluadas.

Además, se considera una métrica de tiempo de navegación, que mide el tiempo transcurrido desde que se recibe un nuevo objetivo y se inicia el cálculo de la trayectoria. El tiempo de navegación se separará en dos casos: navegación exitosa y navegación que supera el tiempo límite. Esta separación es necesaria porque el objetivo del trabajo es reducir los tiempos de navegación, por lo que es crucial distinguir los casos de timeout.

Se evaluó también contabilizar los cambios de trayectoria durante la navegación, pero

esta medida se descartó debido a su alta variabilidad y subjetividad, ya que depende en gran medida de la posición y orientación de los obstáculos.

4.5.2. Definición pruebas

Para las pruebas, se utilizan los distintos mapas seleccionados y se generan posiciones aleatorias para el robot, los obstáculos y los objetivos. Cada mapa tendrá múltiples pruebas con diferentes configuraciones para obtener una muestra representativa de resultados. A cada prueba, compuesta por un mapa y las poses de robot, obstáculos y objetivos, se le denomina escenario.

Cada uno de los cuatro escenarios mencionados tendrá cuatro tipos de pruebas para permitir una comparación exhaustiva de los resultados:

- Navegación base (Odometría+LiDAR). Con Obstáculos fijos.
- Navegación base (Odometría+LiDAR). Con Obstáculos móviles.
- Navegación con localización y detección de obstáculos visual. Con Obstáculos fijos.
- Navegación con localización y detección de obstáculos visual. Con Obstáculos móviles.

La ubicación de los obstáculos en cada prueba se realizará considerando tanto obstáculos fijos como móviles. El movimiento de los obstáculos móviles se definió como un movimiento periódico entre un punto A y un punto B. En las Figuras 4.2, 4.3, 4.4 y 4.5 se muestran las ubicaciones de los obstáculos para cada mapa. Para las pruebas con obstáculos fijos, se utilizan las ubicaciones designadas con un punto amarillo o celeste. En el caso de los obstáculos móviles, se utilizan los pares de ubicaciones marcadas con un punto amarillo, indicando las posiciones A y B con una línea segmentada para mostrar la trayectoria periódica de los obstáculos.

Para las trayectorias del robot, se generan posiciones de origen y destino aleatorias dentro de las zonas permitidas del mapa. Se generan 20 pares de puntos aleatorios, que se fijan para cada una de las cuatro pruebas en cada uno de los cuatro escenarios.

Para las pruebas, se utilizan las variables e hiperparámetros indicados en la Tabla 4.10.

Tabla 4.10: Variables e hiperparámetros utilizados en las pruebas.

Variable o Hiper parámetro	Valor
Modelo Profesor	YOLOv8L
Modelo Estudiante	YOLOv8n
Modelo Regresión	SVR
Umbral error distancia visual para recibir apoyo localización	0.5m
Obstáculos	personas, sillas
Escenarios	4
Trayectorias a evaluar por escenario	20

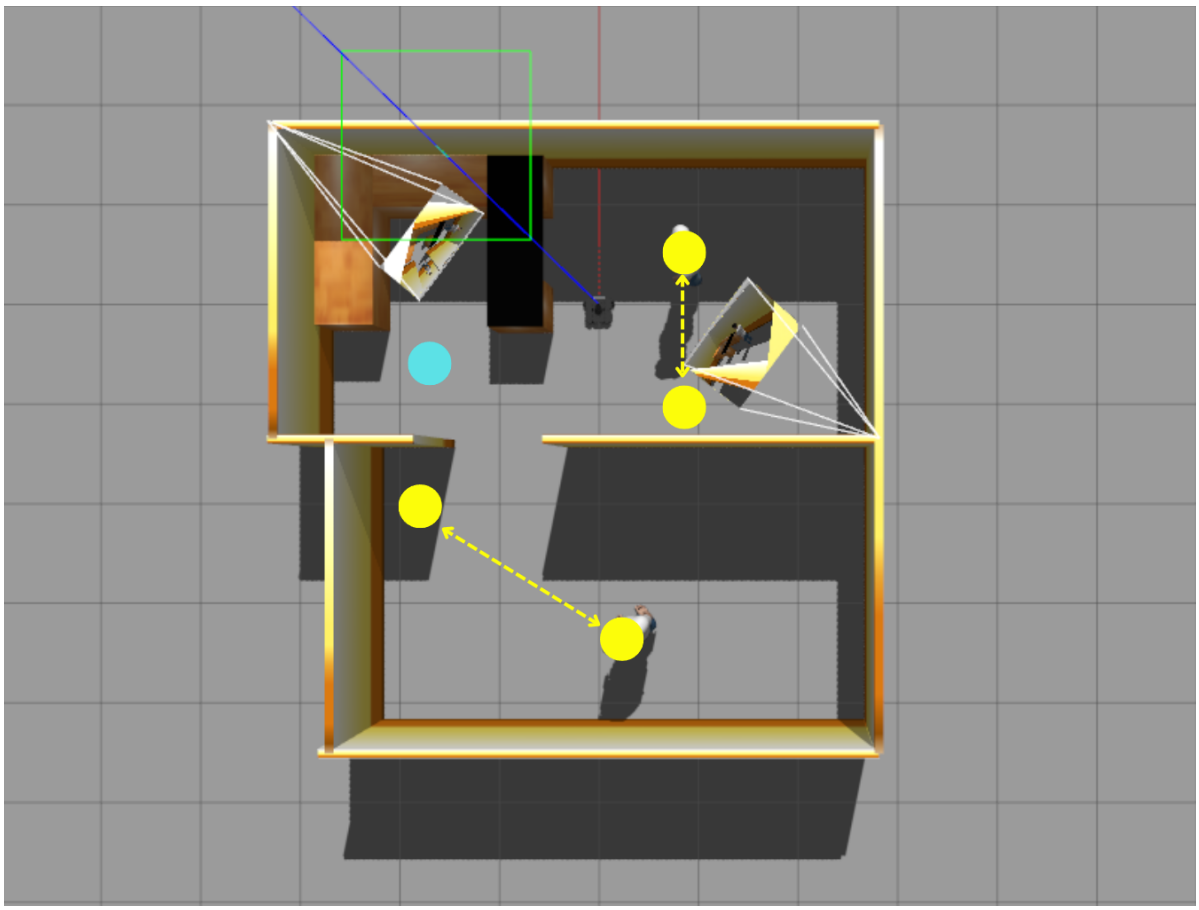

Figura 4.2: Obstáculos prueba mapa simulado S.

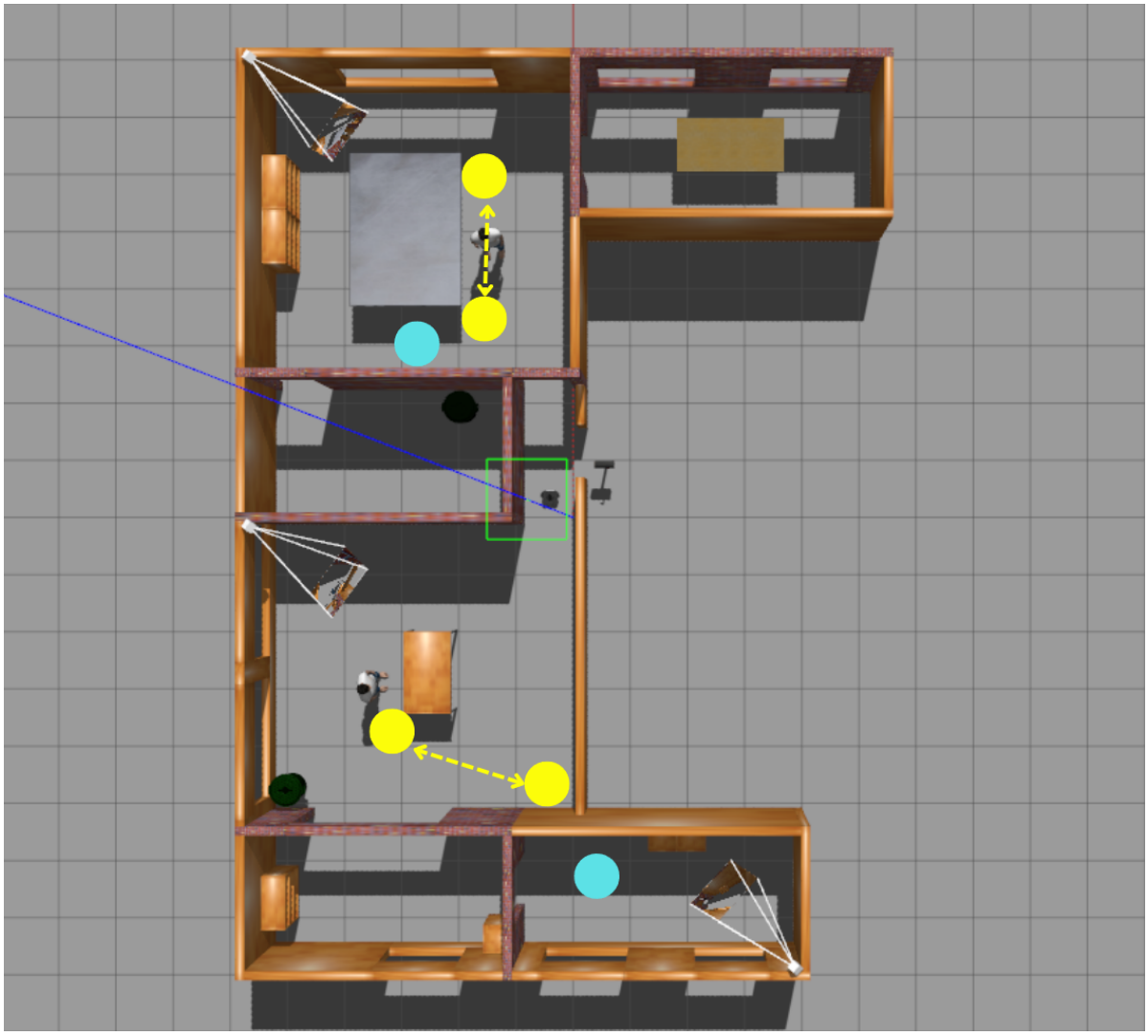

Figura 4.3: Obstáculos prueba mapa simulado M.

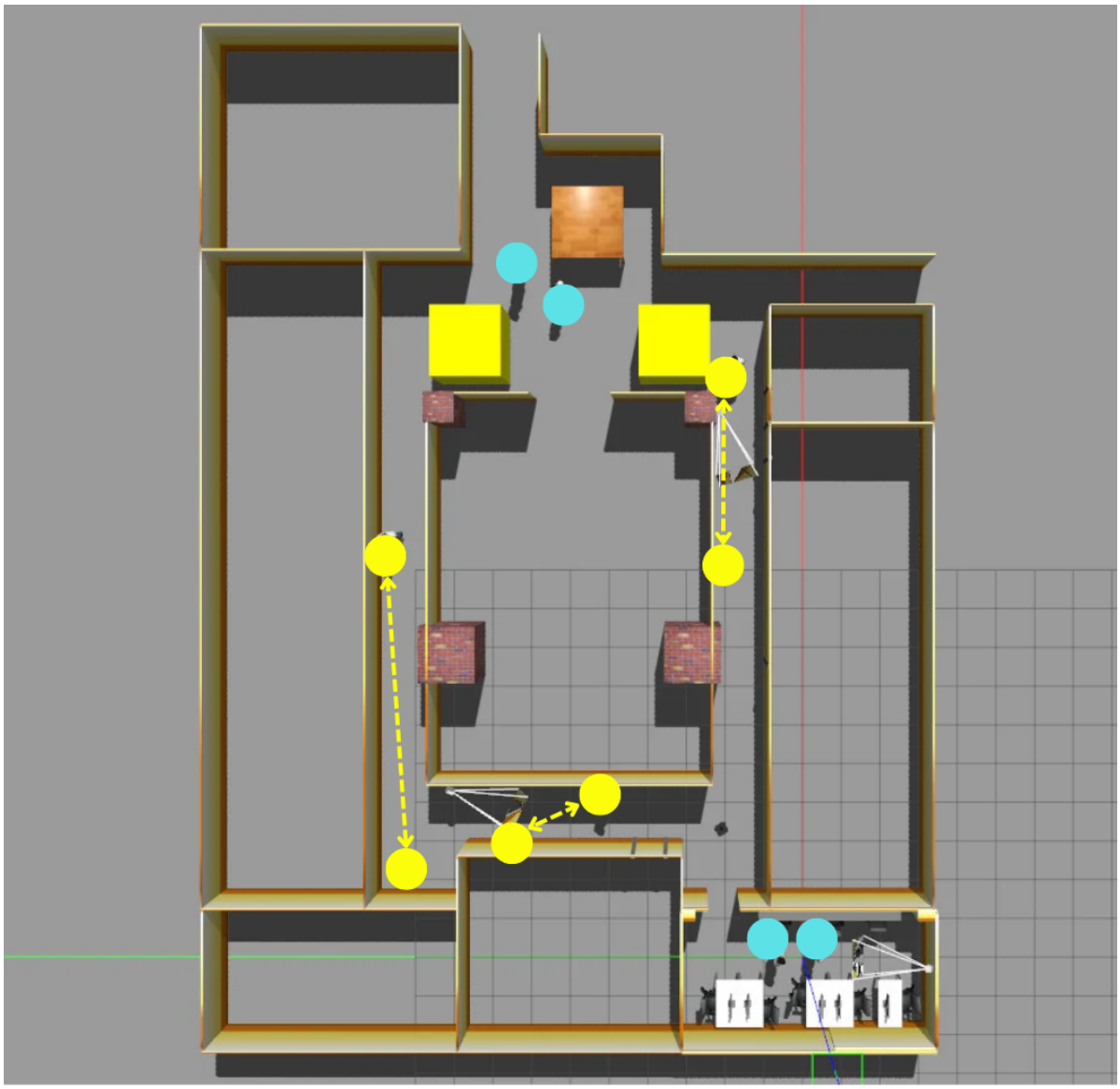

Figura 4.4: Obstáculos prueba mapa simulado L.

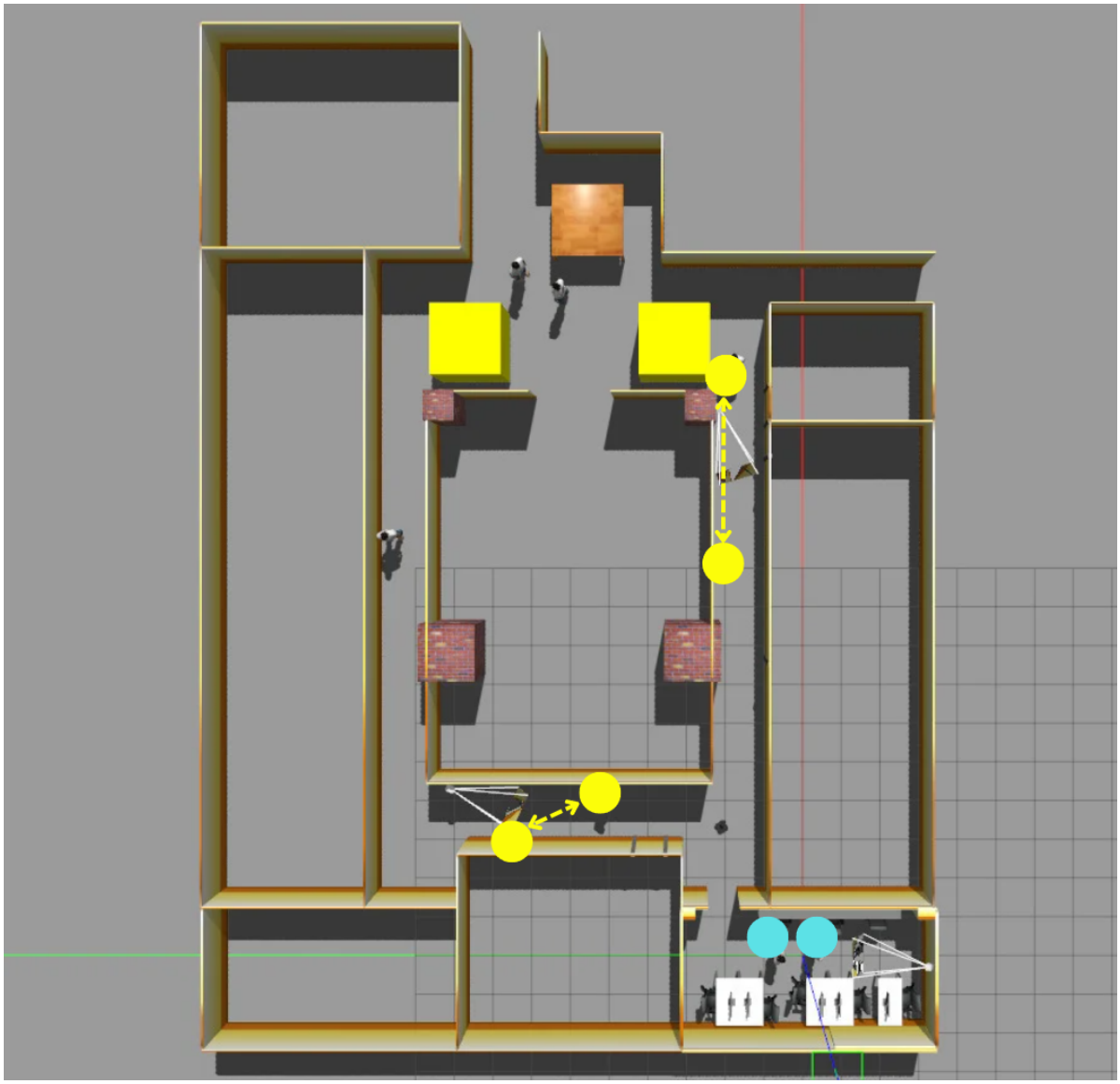

Figura 4.5: Obstáculos prueba mapa real L.

Tabla 4.11: Resultados de sistemas de navegación en ambientes de prueba. (Para facilitar la visualización se marcan en negrita solo los resultados donde el sistema de apoyo visual presenta mejoras respecto al sistema base).

Resultados Pruebas				
Tipo Obstáculo	**Mapa**	**Algoritmo**	**SR**	**Tiempo Nav Promedio [s]**
Fijo	Sim S	Nav Base	0.85	34.38
		Nav + Apoyo Visual	0.85	46.55
	Sim M	Nav Base	0.75	91.36
		Nav + Apoyo Visual	0.75	103.48
	Sim L	Nav Base	0.60	211.82
		Nav + Apoyo Visual	0.60	281.76
	Real L	Nav Base	0.20	467.31
		Nav + Apoyo Visual	**0.25**	**465.86**
Movil	Sim S	Nav Base	0.60	40.08
		Nav + Apoyo Visual	**0.65**	39.90
	Sim M	Nav Base	0.60	100.58
		Nav + Apoyo Visual	0.60	103.93
	Sim L	Nav Base	0.40	245.17
		Nav + Apoyo Visual	**0.50**	259.51
	Real L	Nav Base	0.10	467.31
		Nav + Apoyo Visual	**0.20**	**459.64**

4.5.3. Experimentos

En la Tabla 4.11, se presentan los resultados de las pruebas de navegación en diferentes ambientes, tanto simulados como reales. Los resultados están organizados por tipo de obstáculo (fijo o móvil) y se comparan los algoritmos de navegación base (Nav Base) y navegación con apoyo visual (Nav + Apoyo Visual).

Para los obstáculos fijos en los ambientes simulados (Sim S, Sim M, Sim L), la tasa de éxito (SR) permanece constante al comparar ambos algoritmos, mostrando un SR de 0.85, 0.75 y 0.60 respectivamente. Sin embargo, el tiempo promedio de navegación aumenta significativamente al utilizar el apoyo visual, debido al margen de error en la estimación de la pose, lo que provoca una mayor precaución en el robot. En el ambiente real (Real L), se observa una mejora en la tasa de éxito (de 0.20 a 0.25) y una leve reducción en el tiempo promedio de navegación, lo que indica que el apoyo visual ayuda a corregir problemas de localización que suelen presentarse en entornos reales donde las mediciones de sensores tienen más problemas.

En el caso de obstáculos móviles, se aprecia una mejora en la tasa de éxito en los escenarios Sim S (de 0.60 a 0.65) y Real L (de 0.10 a 0.20) cuando se utiliza el apoyo visual. Esto demuestra que el sistema es efectivo para detectar y evitar obstáculos en movimiento, aumentando así la capacidad del robot para adaptarse a cambios dinámicos en su entorno. Los tiempos de navegación en los escenarios Sim S y Real L también muestran ligeras mejoras, lo que sugiere que el apoyo visual contribuye a una navegación más eficiente al prever y evitar colisiones con obstáculos móviles. La tasa de éxito en el escenario real muestra una mejora

(de 0.10 a 0.20), pasando de tener éxito en 2 de 20 casos a 4 de 20. A pesar de que duplica la cantidad de experimentos exitosos, sigue siendo un resultado bajo tanto con como sin el sistema de apoyo visual. Esto evidencia los problemas que el robot enfrenta en la tarea de navegación dentro del escenario real en comparación con los casos simulados.

Dado que el mapa simulado L corresponde a una representación en Gazebo del mapa real, se puede hacer una comparación válida en ambos casos, mostrando que para el mismo mapa el robot enfrenta dificultades considerablemente mayores para su localización y navegación en el ambiente real. Durante los experimentos, se observó a través de las herramientas de ROS que el robot constantemente tenía problemas de localización en el ambiente de pruebas. Uno de los problemas era que los pasillos no tenían texturas diferenciables por el LiDAR. A pesar de que el sistema visual corregía la posición en algunos momentos, el robot seguía enfrentando problemas de localización. Además, se presentaron problemas con el LiDAR debido a paredes de vidrio y materiales que el sensor no podía detectar, lo que incluso dificultó el proceso de mapeo del espacio. Esto afectó las pruebas de este trabajo, pero también evidenció la necesidad de contar con nuevas fuentes de información que complementen los sensores del robot durante la ejecución de tareas.

En general, se observa que para los obstáculos fijos, el sistema de apoyo visual no contribuye a mejorar la tasa de éxito (SR) y, en algunos casos, impacta negativamente en el tiempo promedio de navegación, aumentándolo. Esto se debe a que, al estar los obstáculos quietos, el sistema no proporciona información adicional útil, ya que el robot obtiene la misma información de los obstáculos cuando estos están dentro del alcance de sus sensores. El error en la estimación de la pose de los obstáculos aumenta la zona de colisión, disminuyendo las rutas posibles para el robot, lo que resulta en un aumento del tiempo de navegación. Sin embargo, el robot tiende a mantener una mayor distancia de los obstáculos.

Para los obstáculos móviles, el sistema de apoyo visual sí contribuye a mejorar la tasa de éxito. Esto se debe a que el sistema puede detectar el movimiento de los obstáculos antes y delimitar las zonas de colisión en las áreas de "patrullaje"de las personas. Estas zonas, por donde circulan las personas, evitan que el robot llegue a un punto de colisión y que ingrese en una zona del mapa de costos de la cual no puede salir. En cuanto al tiempo de navegación, el sistema no tiene un impacto significativo. Aunque evita obstáculos móviles, tiende a realizar rutas más largas.

Los resultados en el simulador y en el escenario real muestran que el sistema es capaz de mejorar la percepción del entorno por parte del robot. El sistema tiene un impacto positivo mayor en el caso real para la localización, debido a los mayores problemas de medición de los sensores. La navegación solo se ve beneficiada en los casos de obstáculos móviles, ya que el sistema de apoyo logra delimitar las zonas de movimiento de los obstáculos, evitando que el robot llegue a puntos sin solución en el mapa de costos. Sin embargo, los resultados son insuficientes para resolver el problema con tasas aceptables, siendo solo una mejora parcial para el sistema de navegación. Con el fin de identificar las debilidades del sistema, se plantea como trabajo a futuro una serie de nuevos experimentos para caracterizar mejor el desempeño de cada submódulo del sistema completo. Entre estos experimentos se propone trabajar con el sistema de transferencia de conocimiento, realizando pruebas en más entornos, con una mayor cantidad de clases y con variaciones de cámaras y escenarios, para identificar fortalezas y debilidades. Respecto al sistema de apoyo a la navegación, se plantea realizar experimentos

que midan las métricas asociadas a la localización del robot, dado que una parte del sistema de apoyo visual impacta directamente esta tarea y se observó que en ambientes reales el robot presenta numerosos problemas. Adicionalmente, se explorará si existe un método que pueda ayudar a estimar la orientación del robot para completar la referencia para el sistema de localización. En el trabajo de Bultmann et al. [7], se utiliza la detección de keypoints para este fin, pero empleando computación externa mediante una Nvidia Jetson Xavier NX. Se podría trabajar en una solución orientada a esos modelos, pero una versión capaz de funcionar dentro del robot. En cuanto al sistema de navegación, se propone medir el impacto de utilizar el sistema visual en la búsqueda de objetivos. Por ejemplo, si es necesario buscar a una persona en particular en un espacio, el robot explorará diferentes lugares con una heurística hasta encontrar a la persona. El sistema visual podría indicar rápidamente la ubicación de la persona u objeto deseado, o descartar ubicaciones donde no está presente, optimizando así el tiempo de búsqueda.

Capítulo 5

Conclusión

En esta tesis se ha planteado y validado que el uso de información visual de cámaras en el entorno de trabajo de un robot puede mejorar sus tareas de detección de objetos, localización y desplazamiento. Se propuso la integración de cámaras a través de la red local por WiFi y un modelo liviano de detección de objetos basado en aprendizaje profundo, junto con regresiones para la estimación de la posición del robot y obstáculos. Los experimentos demostraron que el sistema puede ser ejecutado por el robot en su CPU sin necesidad de computación externa, logrando detectar la posición del robot y dos tipos de obstáculos con buena precisión. La estimación de la posición de los objetos detectados tiene un margen de error de centímetros, lo que se considera suficientemente preciso para la toma de decisiones del robot. Además, el sistema se implementa en tiempo real en el sistema de localización y navegación del robot a través de la integración de los módulos del sistema en ROS.

Adicionalmente, esta tesis presenta un método de ajuste automático para modelos de bajo consumo, optimizándolos para su ejecución en hardware limitado y facilitando su uso en nuevos entornos sin requerir grandes tiempos de calibración ni esfuerzo humano. Este método de auto-calibración y destilación de conocimiento de un modelo profesor a un modelo estudiante liviano permite mejorar considerablemente el desempeño del modelo objetivo en la vista de una o más cámaras fijas, aprovechando que es un entorno controlado que requiere menos generalización. Como resultado, el sistema logra acercarse a la precisión del modelo profesor con una velocidad diez veces superior. Se detectó que el desempeño del sistema decae mientras más diversas sean las imágenes involucradas en la calibración, es decir, mayor cantidad de cámaras y escenarios. Esto quedó demostrado en un caso donde con cuatro cámaras de zonas distintas la precisión no mejoró. Las pruebas muestran que este proceso de calibración puede demorar entre 10 y 60 minutos y puede realizarse en un PC con GPU convencional.

Los resultados finales de la combinación de módulos en este trabajo de tesis no muestran una mejora considerable en la tasa de éxito (SR) y tiempo de navegación para casos con obstáculos inmóviles, aunque las trayectorias son más seguras. En el caso de obstáculos móviles, el sistema mejora modestamente la tasa de éxito de navegación y mantiene los tiempos de navegación relativamente similares. Las métricas de navegación en general muestran una mayor tasa de éxito en el simulador, con una pequeña mejora cuando se usa el sistema

de apoyo visual. En el entorno real, las métricas de navegación son bajas, con tasas de éxito insuficientes, pero es donde el sistema de apoyo visual muestra un mayor aporte en la mejora de la localización del robot, debido a que en el mundo real los sensores presentan mayor error en sus mediciones.

En conclusión, los objetivos de la tesis se cumplieron parcialmente. Por un lado, se comprobó que el uso de cámaras en la red WiFi del edificio donde el robot realiza sus labores puede generar nueva información útil para el robot. El sistema de apoyo visual es capaz de ejecutarse dentro del robot sin necesidad de computación externa y sin comprometer otros procesos del robot. Sin embargo, el caso de uso planteado para mejorar la tasa de éxito y los tiempos de navegación solo mostró resultados positivos en la tasa de éxito, sin una mejora significativa en los tiempos de navegación. El sistema de ajuste automático mostró un gran desempeño, siendo una opción interesante para distintos casos de uso en algoritmos de visión computacional basados en aprendizaje profundo aplicados a cámaras fijas.

Este trabajo demuestra que existe una gran oportunidad en la combinación de la imagen de las cámaras CCTV del edificio y la información de los sensores del robot, estableciendo una base sólida para nuevas investigaciones y desarrollos. Esta combinación puede representar un aporte significativo al campo de la robótica de servicio.

Como trabajo futuro, se plantea la integración de nuevos sistemas de navegación que podrían tener un mejor desempeño aprovechando las cámaras como nuevas fuentes de información. Por ejemplo, sistemas de navegación que combinen el uso de aprendizaje reforzado usando esta información para mejorar problemas de observabilidad parcial. Respecto al método de calibración usando destilamiento de conocimiento, se propone a futuro caracterizar mejor en qué casos presenta mejor desempeño, realizando pruebas en diversos entornos, analizando el límite de clases a abordar y la cantidad de variabilidad en las cámaras, dado que se observó que esto es importante al momento de aplicar el sistema. Otra parte del trabajo a futuro que se propone es trabajar con el sistema de localización aisladamente, midiendo métricas asociadas a esta tarea con el fin de caracterizar mejor las fortalezas y debilidades del sistema visual. También se plantea trabajar la estimación de la orientación del robot revisando trabajos relacionados y opciones capaces de ser ejecutadas dentro del robot, manteniendo el objetivo de este trabajo de no utilizar computación externa.

Además, este trabajo se limitó a validar la hipótesis con una cantidad acotada de objetos a detectar. Se plantea que las cámaras pueden aportar nueva información útil para las tareas de navegación o toma de decisiones de alto nivel del robot cuando se solicitan tareas complejas de búsqueda. Esta información también puede ser útil en una capa de alto nivel del robot, donde se debe buscar un objetivo cuya posición en el mapa no se conoce. Por ejemplo, en la búsqueda de una persona en particular, el uso de la nueva información visual de las cámaras puede colaborar con heurísticas, descartando algunos lugares donde no hay probabilidad de encontrar al objetivo, seleccionando los lugares de mayor probabilidad o directamente detectándolo e indicando su posición. Esto podría reducir significativamente el tiempo de navegación, disminuyendo la cantidad de objetivos de navegación que el robot debe alcanzar en la búsqueda de su objetivo.

También se plantea como una extensión de este trabajo la implementación de nuevos modelos de detección de clases e instancias de objetos. El uso de nuevos modelos capaces

de detectar diferentes tipos de objetos no abordados en este trabajo podría proporcionar al robot una mayor cantidad de información fuera de su campo sensorial, mejorando la toma de decisiones en las tareas evaluadas o nuevas tareas. Finalmente, otro punto interesante a evaluar a futuro es la búsqueda de nuevos modelos profesor y estudiante. Teniendo definido el hardware y la tarea a realizar, se pueden usar estrategias de búsqueda de arquitectura (NAS) o compresión de arquitectura, optimizando los modelos en función de la precisión y la velocidad en su tarea final.

Bibliografía

[1] Raul Mur-Artal, JMM Montiel, and JD Tardós. Orb-slam: a versatile and accurate monocular slam system. In *IEEE transactions on robotics*, volume 31, pages 1147–1163. IEEE, 2015.

[2] Raul Mur-Artal and Juan D Tardos. Orb-slam2: an open-source slam system for monocular, stereo, and rgb-d cameras. In *IEEE Transactions on Robotics*, volume 33, pages 1255–1262. IEEE, 2017.

[3] Carlos Campos, Richard Elvira, Juan J Gómez, JM M Montiel, and Juan D Tardós. Orb-slam3: An accurate open-source library for visual, visual-inertial and multi-map slam. In *IEEE Transactions on Robotics*, volume 37, pages 1874–1890. IEEE, 2021.

[4] Zhefan Xu, Xiaoyang Zhan, Yumeng Xiu, Christopher Suzuki, and Kenji Shimada. Onboard dynamic-object detection and tracking for autonomous robot navigation with rgb-d camera. *IEEE Robotics and Automation Letters*, 9(1):651–658, 2024.

[5] Kiran Jot Singh, Divneet Kapoor, Khushal Thakur, Anshul Sharma, and Xiao-Zhi Gao. Computer-vision based object detection and recognition for service robot in indoor environment. *Computers, Materials Continua*, 72:197–213, 02 2022.

[6] Imran Ahmed, Sadia Din, Gwanggil Jeon, Francesco Piccialli, and Giancarlo Fortino. Towards collaborative robotics in top view surveillance: A framework for multiple object tracking by detection using deep learning. *IEEE/CAA Journal of Automatica Sinica*, 7:1043–1058, 2020.

[7] Simon Bultmann, Raphael Memmesheimer, and Sven Behnke. External camera-based mobile robot pose estimation for collaborative perception with smart edge sensors. 06 2023.

[8] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[9] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *ArXiv*, abs/1804.02767, 2018.

[10] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *ArXiv*, abs/2004.10934, 2020.

[11] Chuyin Li, Lu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, L. Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, Yiduo Li, Bo Zhang, Yufei Liang, Linyuan Zhou, Xiaoming Xu, Xiangxiang Chu, Xiaoming Wei, and Xiaolin Wei. Yolov6: A single-stage object detection framework for industrial applications. *ArXiv*, abs/2209.02976, 2022.

[12] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7464–7475, 2023.

[13] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *ArXiv*, abs/1503.02531, 2015.

[14] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. Improved knowledge distillation via teacher assistant. In *AAAI Conference on Artificial Intelligence*, 2019.

[15] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics & Automation Magazine*, 13(2):99–110, 2006.

[16] Sebastian Thrun. Particle filters in robotics. In *Conference on Uncertainty in Artificial Intelligence*, 2002.

[17] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[18] Sven Koenig and Maxim Likhachev. D* lite. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 476–483, 2002.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25:1097–1105, 2012.

[20] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[21] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[22] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. pages 770–778, 2016.

[23] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[24] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[25] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[26] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In *Computer Vision – ECCV 2016*, pages 21–37. Springer International Publishing, 2016.

[27] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. pages 2999–3007, 2017.

[28] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 734–750. Springer, 2018.

[29] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. Centernet: Keypoint triplets for object detection. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6568–6577, 2019.

[30] Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

[31] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: a survey. *J. Mach. Learn. Res.*, 20(1):1997–2017, jan 2019.

[32] Colin White, Mahmoud Safari, Rhea Sanjay Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. Neural architecture search: Insights from 1000 papers. *ArXiv*, abs/2301.08727, 2023.

[33] Anubhav Ashok, Nicholas Rhinehart, Fares Beainy, and Kris M. Kitani. N2n learning: Network to network compression via policy gradient reinforcement learning. 2018.

[34] Yihui He and Song Han. ADC: automated deep compression and acceleration with reinforcement learning. *CoRR*, abs/1802.03494, 2018.

[35] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *ArXiv*, abs/1905.11946, 2019.

[36] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection. pages 10778–10787, 2020.

[37] RangiLyu. Nanodet-plus: Super fast and high accuracy lightweight anchor-free object detection model. `https://github.com/RangiLyu/nanodet`, 2021.

[38] Guanghua Yu, Qinyao Chang, Wenyu Lv, Chang Xu, Cheng Cui, Wei Ji, Qingqing Dang, Kaipeng Deng, Guanzhong Wang, Yuning Du, Baohua Lai, Qiwen Liu, Xiaoguang Hu, Dianhai Yu, and Yanjun Ma. Pp-picodet: A better real-time object detector on mobile devices. *CoRR*, abs/2111.00902, 2021.

[39] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[40] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525, 2017.

[41] Chien-Yao Wang, I-Hau Yeh, and Hongpeng Liao. Yolov9: Learning what you want to learn using programmable gradient information. *ArXiv*, abs/2402.13616, 2024.

[42] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. ROS: an open-source Robot Operating System. page 6.

[43] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022.

[44] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, 2004.

[45] Young-Hee Lee, Chen Zhu, Thomas Wiedemann, Emanuel Staudinger, Siwei Zhang, and Christoph Günther. Covor-slam: Cooperative slam using visual odometry and ranges for multi-robot systems. *ArXiv*, abs/2311.12580, 2023.

[46] Ros noetic ninjemys. `http://wiki.ros.org/noetic`, 2020. Accessed: 2024-07-21.

[47] Gazebo 11. `http://gazebosim.org/`, 2020. Accessed: 2024-07-21.

[48] ROBOTIS. *TurtleBot3 Manual*, 2017. Accessed: 2024-07-21.

[49] Gary Bradski. The opencv library. In *Dr. Dobb's Journal of Software Tools*, 2000.

[50] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. pages 740–755, 2014.

[51] Open Source Robotics Foundation. amcl - ros wiki. `http://wiki.ros.org/amcl`, 2013. Accessed: 2024-07-21.

[52] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[53] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[54] Cyrill Stachniss. *Robotic Mapping and Exploration*, volume 55. Springer, 2009.

[55] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[56] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–10, 2017.

[57] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2815–2823, Los Alamitos, CA, USA, jun 2019. IEEE Computer Society.

[58] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 815–832, Cham, 2018. Springer International Publishing.

[59] Andrew Howard, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, Yukun Zhu, Ruoming Pang, Hartwig Adam, and Quoc Le. Searching for mobilenetv3. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1314–1324, 2019.