Framework unificado para navegación autónoma en TurtleBot3 Waffle Pi

Marcelo Contreras
Ing.Mecatrónica
UTEC
marcelo.contreras@utec.edu.pe

Cesar Guillen
Ing.Mecatrónica
UTEC
cesar.guillen@utec.edu.pe

Edward Lopez
Ing.Mecatrónica
UTEC
edward.lopez.t@utec.edu.pe

Mauricio Rivera
Ing.Electrónica
UTEC
mauricio.rivera@utec.edu.pe

Abstract

El presente trabajo corresponde al proyecto final del curso de Robótica Autónoma (UTEC) e introduce un framework unificado para navegación autónoma compuesto por módulos de percepción, localización, mapeo y planeamiento desarrollados en ROS para un uso efectivo y rápido. Esta implementación demuestra una navegación efectiva en el robot Turtlebot3 en escenarios cerrados cómo almacenes, laboratorios y hogares tanto en simulación (Gazebo) utilizando solo visión aun bajo la presencia de elementos dinámicos (personas) gracias a que dos módulos se encargan del mapeo, uno estático y otro dinámico, y al uso de dos planificadores de movimiento para considerar tanto el mapa precargado cómo la percepción local.

1 Introducción

La navegación representa el componente clave en los sistemas autónomos para poder desplazarse y interactuar con los entornos que la rodean. Esta compuesto por los siguientes módulos: percepción, localización, mapeo, planeamiento y control. Hasta la fecha, se han realizado avances importantes en cada módulo individualmente. No obstante, no son comunes las implementaciones completas de sistemas de navegación y las pocas conocidas forman parte de productos patentados lo cual no permite su libre uso y modificación. Por consiguiente, este trabajo busca utilizar algoritmos o paquetes del estado del arte en cada módulo y conectarlas dentro del entorno ROS para un rápido despliéguese en cualquier plataforma de prueba. Así mismo, este trabajo busca ofrecer robustez de navegación ante la presencia de objetos dinámicos al considerarlos dentro del mapeo y planeamiento. Los algoritmos elegidos son: ORB-SLAM 2 [7], Octomap [5], YOLOv8 [10], A* [1] y DWA [2]. Así mismo, esta solución considera un balance entre flexibilidad, eficiencia computacional y exactitud. Con el debido seguimiento de los tipos de mensajes de ROS utilizados, se puede reemplazar cada módulo por un algoritmo más actual. La plataforma de pruebas fue un robot móvil Turtlebot3 Waffle-PI equipado con una cámara realsense R2OO y dos entornos de simulación en Gazebo. Por otra parte, se hizo una implementación preliminar de una cámara con las mismas prestaciones (Astra Embedded S) para una implementación real enfocada en el mapeo del laboratorio L201 de UTEC. Esta plataforma de navegación permite que robot móviles puedan ser aplicados para casos cómo:

- Monitoreo de equipos industriales en lugares con presencia de personas.
- Desplazamiento y almacenamiento de cargas en almacenes (*warehouses*).
- Exploración de entornos con superficies lo suficientemente uniformes cómo túneles asfaltados.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

Las siguientes secciones están comprendidas por: Metodología, donde se explica en detalle el funcionamiento de cada módulo y su interacción; Resultados, presenta el entorno de pruebas, detalles de la implementación y resultados cualitativos; y Conclusiones.

2 Metodología

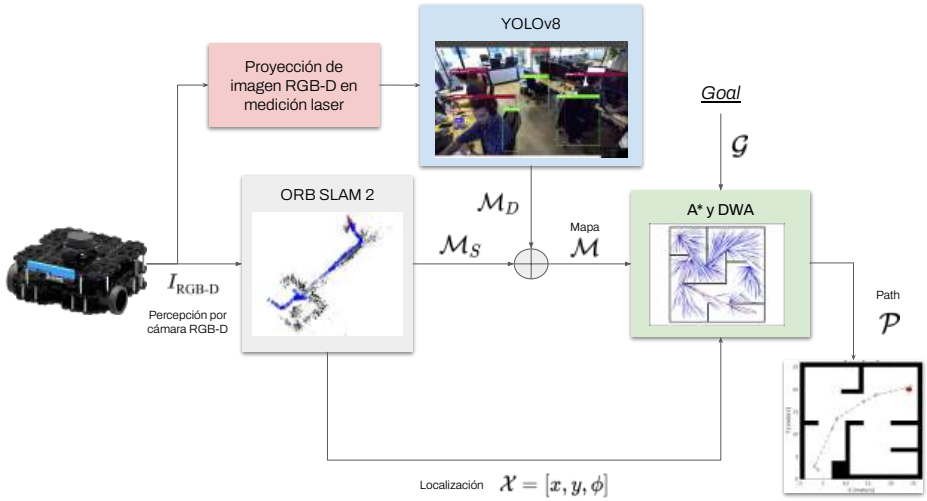


Figure 1: Esquemático del *framework* de navegación autónoma unificado

El programa requiere dos entradas: imágenes RGB-D y una posición deseada. Se ha optado por descartar cualquier medición de LiDAR en vista que este sensor suele aumenta el costo de las plataformas de evaluación y que muchas de sus cualidades pueden ser obtenidas mediante una única cámara RGB-D en entornos cerrados. Al inicio, la imagen $I_{\text{RGB-D}}$ es procesada por el algoritmo ORB-SLAM 2 que se encarga de generar un mapa estático del entorno $\mathcal{M}_S$ y proveer de localización $\mathcal{X}$ al módulo de planeamiento. En una rama paralela, se envía al componente RGB de la imagen $\mathcal{I}$ para detectar todas las personas en el campo de visión mediante YOLOv8 dentro de un *bounding box*. Esta región de interés es enmascarada en la imagen de profundidad para solo utilizar esa sección a la hora de proyectar la profundidad en mediciones de láser virtuales para reemplazar el LiDAR. Son estas mediciones independientes del mapa anterior las que denominados mapa dinámico. La planificación de movimiento recibe los dos mapas y la posición deseada $\mathcal{G}$ para trazar una serie de rutas mediante A* cómo planificador global y DWA cómo el local, el cual permite esquivar los obstáculos no mapeados. De este apartado, se envía el camino o *path* $\mathcal{P}$ al Turtlebot mediante comandos de velocidad.

2.1 Generación de mapa estático

El proceso de mapeo estático se consigue a través de la percepción de imágenes RGB-D (color y profundidad) en distintos instantes de tiempo. Se consideró ese tipo frente a otros opciones debido a dos motivos: 1) las cámaras monoculares y estéreo no consiguen generar nubes de puntos lo suficientemente densas para crear una grilla de ocupación 2D o 3D y 2) en entornos cerrados, las cámaras RGB-D proveen una precisión de profundidad mucho mayor que la triangulación en la configuración estéreo.

Para generar el mapa de los entornos, se debe conocer con precisión la ubicación del robot puesto que las mediciones considerarán cómo sistema de referencia el desplazamiento en cada instante. Así mismo, la localización necesita de un mapa de donde pueda extraer puntos de interés o *landmarks* que permite estimar y corregir su estimación de posición. En vista que los dos procesos son dependientes mutuamente, se ha considera una solución SLAM (*simultaneous localization and mapping*) del estado del arte para cámaras RGB-D.

2.2 ORB-SLAM 2

Propuesto por Raul Mur-Artal y Juan D. Tardos [7], ORB-SLAM 2 ha sido el *benchmark* para sistemas de odometría visual (VO) y SLAM visual puesto que al momento de su publicación obtuvo un balance prometedor entre su precisión de localización y su coste computacional debido al uso extensivo de paralelismo. Cómo se aprecia en la Fig.2, este algoritmo esta compuesto por 3 hilos o *threads*: *tracking*, *local mapping* y *loop closing*; Su *front-end* esta manejado por el hilo de *tracking* a través de *features* y descriptores ORB mientras que su *back-end* es el proceso de optimización por grafos conocido cómo *Bundle Adjustment*.

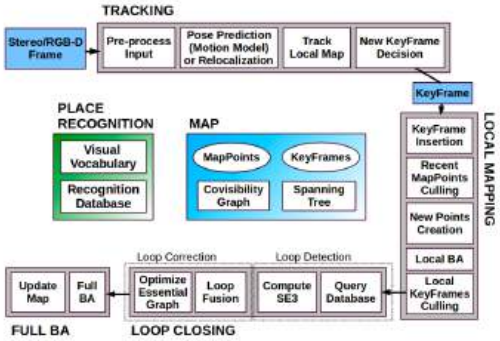


Figure 2: Esquema general de ORB-SLAM 2 [7] con sus hilos y subrutinas.

2.3 Procesamiento de mapa estático

Para conseguir la representación requerida para el planeamiento se necesita proyectar la nube de puntos tridimensional en una grilla de ocupación de la cual se pueda determinar los obstáculos. La nube de puntos generada por ORB-SLAM 2 de por sí no puede ser proyectada directamente puesto que presenta varios puntos atípicos aparte que puede considerar puntos en el suelo cómo obstáculos. Por ello se considera los siguientes procesamientos de nube de puntos:

- Un filtro pasaobjetos que rechaza cualquier punto que se encuentre en el rango $z \in [-0.05, 0.05]$ m. La estocacidad propia del SLAM ocasiona que algunos puntos estén por debajo de $z = 0$ a pesar que el robot nunca cambie su altura.
- Un filtro RANSAC utilizando un modelo de plano para eliminar cualquier área que corresponda al suelo restante cómo planos inclinados fuera del rango de z del primer punto.
- Un filtro estadístico basado *k-means* que detecta data espuria y alejada de nubes de puntos con centroides definidos

Esta etapa de procesamiento fue implementada utilizando la libreria PCL (*Point cloud library*) [9].

2.4 Octomap

Octomap [5] es una libreria escrita en C++ tomando en mente la eficiencia y robustez para la representación de nubes de puntos 3D consideración ocupación. Por una parte, emplea Octrees cómo representación volumétrica donde un punto en el espacio es un voxel y al determinarse que se encuentra ocupado, es dividido en ocho subvoxels. Tanto el voxel padre cómo el hijo se encuentran conectados mediante un grafos para su búsqueda rápida. Por otra parte, se considera que las mediciones de sensores cómo LiDAR o cámaras RGB-D pueden acarrear ruido, de manera que considera una estructura probabilística para determina si un voxel es ocupado uno mediante una probabilidad a priori definida por el usuario y la probabilidad de realizar un medición en base a todos los voxels anteriores. Después de calcular esta probabilidad, se utiliza un threshold de $p = 0.5$ para discriminar la ocupación final del voxel.

Para este trabajo, se utilizará la proyección ocupacional de los octress en un mapa conocido cómo grilla de ocupación compuesto por tres tipos de voxels: ocupados (negro), libre (blanco) y gris (no

explorado). Esta representación no ocupa mucho espacio y define claramente las área por donde el robot puede desplazarse.

2.5 Generación de mapa dinámico

2.5.1 Mediciones de láser virtuales

2.5.2 YOLOv8

Los humanos somos capaces de captar información visual y procesarla, pudiendo reconocer [11] y diferenciar entre objetos al instante [6]. A diferencia de nosotros, las computadoras tienen que desarrollar algoritmos con el fin de imitar esta capacidad humana para procesar imágenes y vídeos [6] [8].

Entre los modelos de red para cumplir esta labor se destaca a YOLO (You Only Look Once) por la gran popularidad que ha obtenido en los últimos años debido a su alta precisión [6]. La versión más reciente de YOLO es la versión 8 (YOLOv8) [10] la cual se destaca por obtener mayores valores de COCO mAP, en comparación con las otras versiones de la familia YOLO, haciendo uso de una menor cantidad de parámetros y menor latencia (Figura). Esta versión es capaz de realizar clasificación, detección, segmentación, seguimiento y pose estimation [10]. Para el presente trabajo se hará uso de esta versión para realizar detección personas con el objetivo de obtener los puntos de las esquinas de los bounding boxes formados alrededor de las personas detectadas. Además existen diferentes modelos de esta versión como YOLOv8n, YOLOv8s, YOLOv8m, YOLOv8l y YOLOv8x (Figura) [10]; nosotros haremos uso del primer modelo debido a que estimados que el ambiente en el que será probado no requerirá de un gran mPA para realizar el trabajo de forma esperada, además que todo el programa del robot estará corriendo en el CPU, por lo que se da prioridad al uso bajo de recursos.

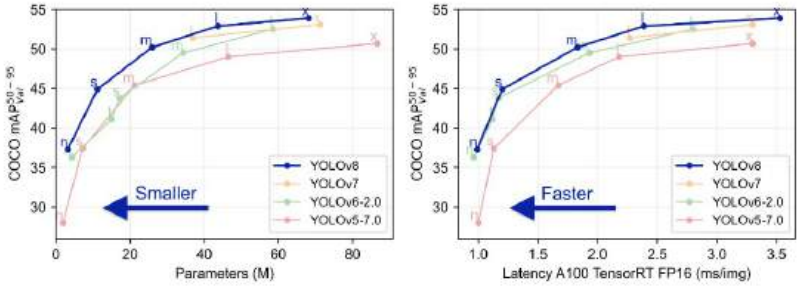


Figure 3: Comparación del rendimiento de distintas versiones de YOLO.

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Figure 4: Modelos de YOLOv8.

Lo que se desea con la incorporación de la detección de personas es añadir un mapa de costo dinámico al mapa de costo estático existente. La detección de las personas se hará haciendo uso de los frames dados por la cámara RGB, se realizará el procesamiento de la imagen y de detectarse una persona, se utilizarán las coordenadas de las esquinas del bounding box (en el plano de la imagen) para hacer

una máscara la cual enmascarará los frames entregados por la cámara de profundidad RGB-D por donde cualquier valor fuera del bounding box será 0 (muy lejano) (Figura) y todo valor de obtenido por la cámara de profundidad será considerado como un costo en el mapa. De esta forma es posible trabajar los costos dinámico y estático independientemente sin hacer generar redundancia en los costos obtenidos.

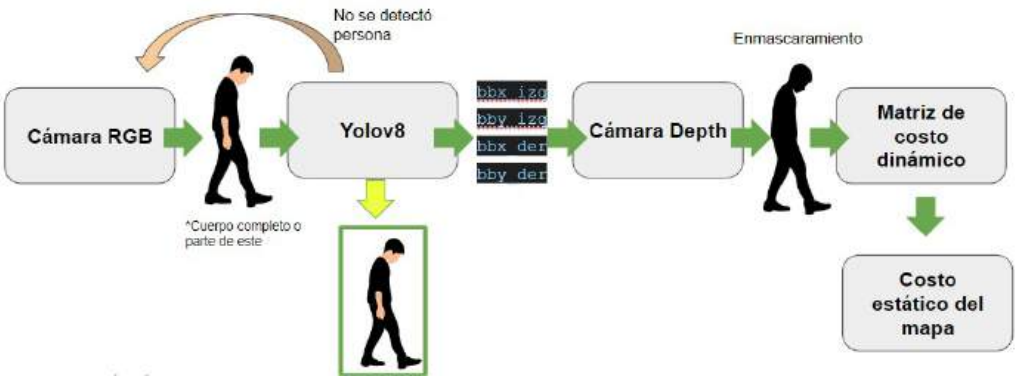


Figure 5: Pipeline de la obtención del Mapa de costo dinámico.

2.6 ROS Motion Planning

ROS Motion Planning es un paquete para ROS Noetic que incluye, en términos generales, la localización, planeación y navegación de AGVs / AMRs mediante Lidar, principalmente para las versiones turtlebot waffle, waffle-pi y burger. El paquete ofrece una amplia cantidad de algoritmos de planificación de movimiento como A*, RRT, dijkstra, D*, DWA, PID, etc. Además, el entorno brinda mucha versatilidad al modificar los mapas, mundos, algoritmos y parámetros de los planeadores. Los principales algoritmos de planeación usados en este framework son el A* para la planeación global y DWA para la planeación local.

2.6.1 Algoritmo A*

El algoritmo para la planificación de movimiento que se ha escogido entre los disponibles ha sido A-estrella (A*). Tal como es expuesto por Hart, Nilsson y Raphael [4], un algoritmo de búsqueda debería encontrar las nodos óptimos para expandir su árbol de ubicaciones a las que el robot puede moverse. Con tal objetivo, se emplea una función de admisibilidad con la que se habilita la visita a los potenciales nodos para que viaje el robot y escoge la opción óptima de acuerdo a sus restricciones.

2.6.2 Dynamic Window Approach (DWA)

Para el planeamiento local del robot se emplea una ventana dinámica cuyo funcionamiento es explicado por Fox, Burgard y Thrun [3]. Los autores explican que la dinámica del robot es trabajada directamente en el espacio de velocidades. Restringe el comando del robot a aquellos valores que son alcanzables en intervalos cortos de tiempo para las aceleraciones que han sido limitadas. Todo esto se realiza sobre el establecimiento de trayectorias circulares determinadas por pares de velocidad traslacional y rotacional.

3 Resultados

3.1 Simulación

3.1.1 Mapa estático

En los entornos de simulación escogidos son Turtlebot3 House y Amazon Small House (Fig .6), ejecutuados dentro de Gazebo. Para su elección, se consideraron ambientes que tengan superficies texturizadas, evitando tener colores planos, para facilitar la detección de características ORB. Así

mismo, estos deben ser entornos cerrados debido a que las mediciones de profundidad por imágenes RGB-D son confiables dentro de un rango de $\sim$ 10 m.

(a) *Turtlebot house* (b) *Amazon house*

Figure 6: Entornos de simulación *house* y *Amazon house* en Gazebo

Para el análisis de resultados, se seguirá el mismo orden del esquemático general visto en la Fig. 2 empezando por el mapeo con ORB-SLAM 2. En Fig. 7, se aprecia que algoritmo considera puntos de superficies altamente texturizadas cómo la alfombra o el rostro de una persona y bordes que se encuentran bien definidos por el gradiente de color en la vencindad del borde mismo.

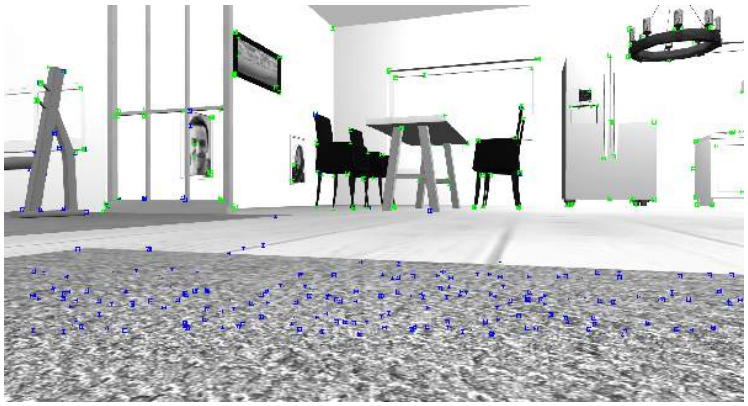


Figure 7: Visualización de ORB SLAM 2 donde los puntos verdes son los obtenidos en tiempo real y los azules son los almacenados por el mapa para localización aislada

Ambos entornos fueron recorridos en su totalidad y consideró revistar la posición inicial 3 veces para que tanto la trayectoria cómo el mapa fueran corregidos mediante *loop closing* para mayor fiabilidad. El mapa generado por el algoritmo de SLAM visual se puede ver en la Fig. 8. Si bien se observa que las paredes y esquinas que limitan el entorno tiene una densidad de puntos aceptable, hay presencia notable de ruido entorno a estas regiones de la nube que se muestra dentro de círculos verdes. Así mismo, los puntos de la alfombra también han sido mapeados y si no son filtrados, serán proyectados por Octomap en la grilla de ocupación cuando en realidad es un entorno libre. Para lidiar con ello, se aplicó el procesamiento de puntos por PCL explicado en la sección de metodología con excepción del filtro RANSAC puesto que este es activado por defecto en Octomap.

A forma de comprobación que el mapeo represente a escala el entorno de su alrededor, se realizó una comparación entre la nube de puntos generada por ORB-SLAM 2 mediciones del LiDAR integrado con el Turtlebot3. En 9 se aprecia que hay una alta similitud entre las mediciones láser de las paredes y su respectiva nube de puntos.

Después del filtrado, la nube de puntos son convertidas a Voxels mediante el Octomap considerando una resolución de 0.05, aparte de aplicar un filtro RANSAC para remover el suelo. Esta nueva representación es proyectada en el plano $z = 0$ y se consigue el mapa de ocupación ya que los Voxels

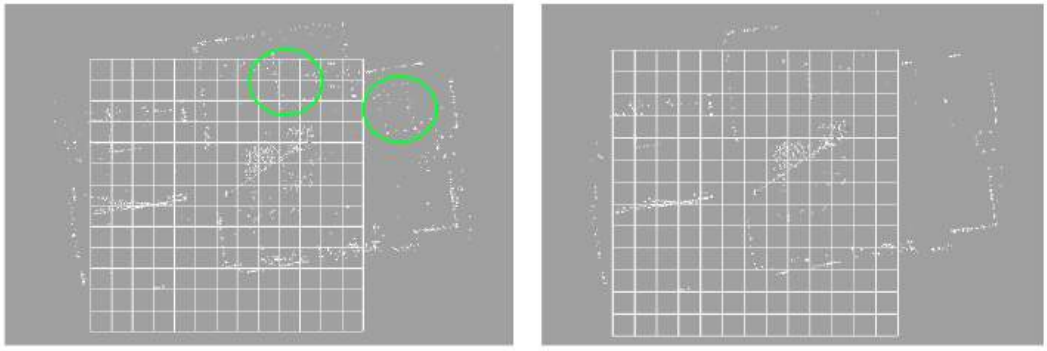
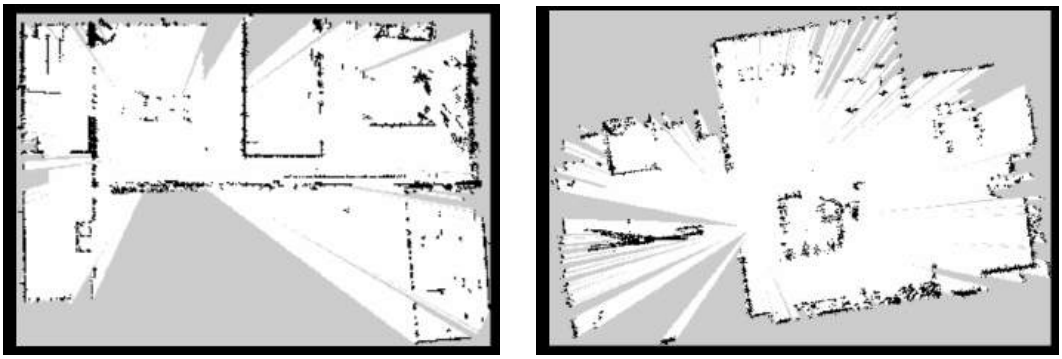


Figure 8: Comparación pre (izquierda) y post procesamiento (derecha) de la nube de puntos del mapa por ORB-SLAM 2

Figure 9: Comparación de localización y escala de nube de puntos de ORB-SLAM 2 y LiDAR

ya crea su grafo de búsquedas considerando ocupación. Los resultados de mapeo en grilla se observan en 10.



(a) Turtlebot house (b) Amazon house

Figure 10: Mapas de ocupación de entornos de simulación mediante Octomap

3.1.2 Mapa de costo dinámico

Para esta prueba se hizo uso del mundo Amazon house y se incluyó personas. Se teleoperó el turtlebot hasta llegar a estar al frente de una persona. Una vez colocado, fue posible observar las piernas de la persona, con ello el robot indicó que una persona había sido detectada mostrando el correcto funcionamiento del modelo YOLOv8. En la Figura a se muestra la imagen de la cámara de profundidad en la cual se encuentran una persona y una mesa; en la Figura b se puede observar la imagen dada por la cámara de profundidad después del enmascaramiento, en esta solo se puede ver un recuadro con la persona adentro de este (notar que no se incluye la mesa dentro de este frame); y

en la Figura c se puede apreciar la inclusión del mapa de costo dinámico, resaltando que a pesar de que tanto la persona como la mesa son visibles por la cámara de profundidad, solo la persona genera un costo a partir de esta cámara (costo dinámico), mientras que la mesa solo posee el costo estático mas no el dinámico.

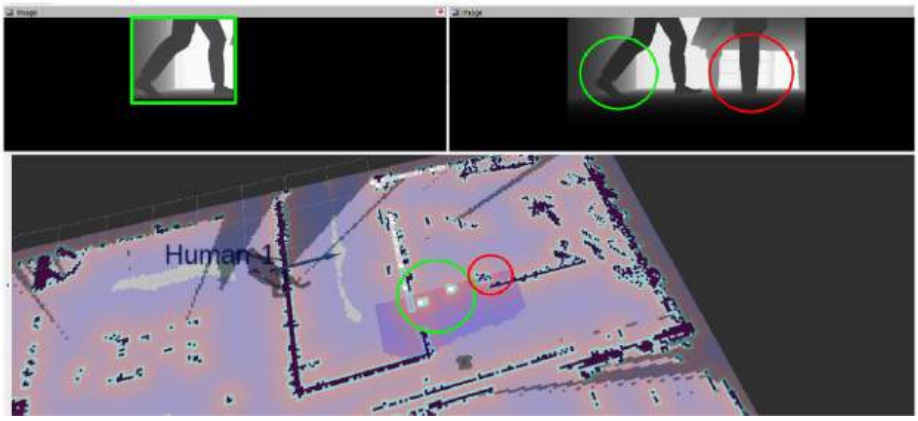
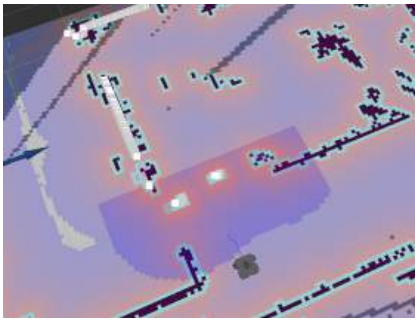
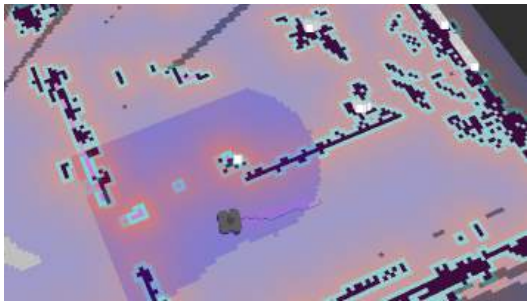


Figure 11: Resultados de la prueba del Mapa de costo dinámico en simulación.

3.1.3 Planificación



(a) Planeamiento con coste dinámico de persona



(b) Planeamiento despues de perder detección de persona

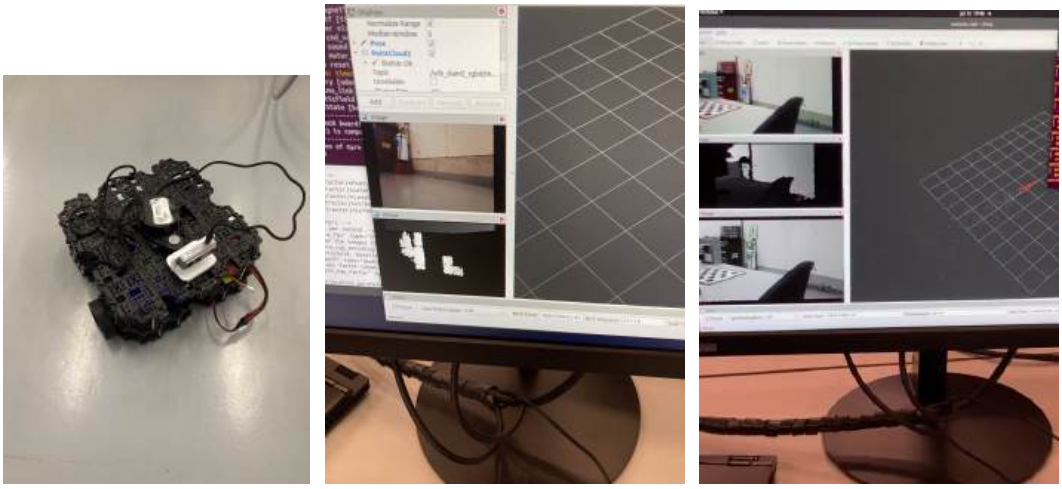
Figure 12: Resultados de planeamiento con mapa completo (estático y dinámico)

3.2 Entorno real

$$K = \begin{bmatrix} 1734.154 & 0 & 1004.148 \\ 0 & 1742.898 & 499.833 \\ 0 & 0 & 1 \end{bmatrix}, p = [-0.024 \quad -0.091 \quad -0.005 \quad 0.01 \quad 0] \quad (1)$$

4 Conclusiones

- Se diseñó un marco de trabajo que unifica percepción, localización, mapeo y planificación para conseguir navegación autónoma.
- Fue aprovechada las capacidades de una cámara RGB-D para reemplazar el sensado por LiDAR en todos las etapas del pipeline.
- Se consiguió incorporar elementos dinámicos en el mapa estático y que fueran considerados en la planificación.
- Trabajos futuros: La implementación completa del módulo, extensión utilizando ORB-SLAM 3 con calibración extrínseca IMU-Cámara.



(a) Instalación de cámara Astra Embedded S en Turtlebot

(b) Visualización de imagen RGB-D despues de montar la cámara

(c) ORB-SLAM 2 mapeando usando imágenes de la cámara Astra

Figure 13: Implementación preliminar

References

[1] Daniel Foead, Alifio Ghifari, Marchel Budi Kusuma, Novita Hanafiah, and Eric Gunawan. A Systematic Literature Review of A* Pathfinding. *Procedia Computer Science*, 179:507–514, 1 2021.

[2] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics Automation Magazine*, 4(1):23–33, 1997.

[3] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[4] 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.

[5] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: an efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2 2013.

[6] Muhammad Hussain. Yolo-v1 to yolo-v8, the rise of yolo and its complementary nature toward digital manufacturing and industrial defect detection. *Machines*, 11(7):677, 2023.

[7] Raul Mur-Artal and Juan D. Tardos. ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, oct 2017.

[8] Mike O Ojo and Azlan Zahid. Deep learning in controlled environment agriculture: A review of recent advancements, challenges and prospects. *Sensors*, 22(20):7965, 2022.

[9] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[10] Ultralytics. GitHub - ultralytics/ultralytics: NEW - YOLOv8 in PyTorch > ONNX > CoreML > TFLite.

[11] Bin Zhang, Changqin Quan, and Fuji Ren. Study on cnn in the recognition of emotion in audio and images. In *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*, pages 1–5, 2016.