SLAM Y GROUNDTRUTH USANDO MARCADORES

DESARROLLO DE METODOLOGÍA SLAM USANDO MARCADORES TANTO PARA EL GROUDTRUTH COMO PARA EL SLAM DE UN ROBOT TIPO TURTLEBOT

Autor:

JULIÁN ESCOBAR ACEVEDO

Tutor – Coautor:

DAVINSON CASTAÑO CANO

UNIVERSIDAD EAFIT

ESCUELA DE INGENIERÍA

DEPARTAMENTO DE MAESTRÍA EN INGENIERÍA

MEDELLÍN, ANTIOQUIA, COLOMBIA

2024

CONTENIDO

Pag.

ILUSTRACIONES

Pag.

ECUACIONES

Pag.

TABLAS

Pag.

1 INTRODUCCIÓN

En el campo de la robótica la construcción de un mapa es una tarea fundamental, por permitir conocer puntos de referencia, ubicar el robot en el espacio y reubicarlo para disminuir los errores de estimación al volver a desplazarse por las áreas ya conocidas [1]. La tecnología de localización y mapeo simultánea (SLAM por sus siglas en inglés) soluciona el problema de adquirir el ambiente inicialmente desconocido en el cual se desempeña un robot y a su vez calcular la posición de sí mismo y de los obstáculos en el espacio, por medio del procesamiento de las señales proporcionadas por sensores [2]. Esta tecnología fue propuesta por los investigadores Smith R y Cheeseman P en 1986 [3]. La información debe poderse utilizar por los robots para reducir la incertidumbre de la ubicación de este y de su entorno, en variables como la posición y la orientación, para poder lograr tareas determinadas [3]. Además, la auto localización ayuda a los robots con tareas autónomas como ir de un punto a otro evitando obstáculos y bordeándolos, como también una relocalización robusta aumentando así la interacción humano-robot y su autonomía [4]. Otro dominio del área de desarrollos SLAM, es la realidad aumentada (AR) que se aplica para procesos de rehabilitación médica, producción industrial, comercio electrónico, educación, cultura, entre otros [5].

Los algoritmos de SLAM, en general, se componen de 3 partes, inicialización, seguimiento y mapeo. En la inicialización, se define el sistema de coordenadas global 0,0,0 o punto de partida inicial del mapa a generar y la identificación de los primeros elementos que se usarán como referencia para comenzar con el mapeo y rastreo. El seguimiento es la capacidad de calcular la pose en el sistema global de referencia a partir de la información disponible con respecto al ambiente en un momento determinado. El mapeo, es la actualización del mapa a partir de la información captada por los sensores, en relación con el medio en el que se encuentra. El seguimiento y mapeo se realizan en paralelo después de la inicialización [6].

Es importante tener en cuenta para los métodos SLAM, la implementación de optimización de la información, debido a la presencia de errores de medición constantes e inherentes de los sensores que se utilizan, por consecuencia de las limitaciones físicas. Uno de los métodos más importantes es el ciclo de cierre o cierre de bucle. El cierre de bucle consiste en detectar patrones que fueron capturados previamente, en la misma ubicación, pero en momentos diferentes. Con la información de cierre de bucle, es posible estimar el error acumulado y reorganizar el mapa teniendo en cuenta el error.

Actualmente existen varios tipos de SLAM, entre ellos los principales son el LiDAR-SLAM el cual utiliza como fundamento el sensor LiDAR (*Light Detection and Ranging* por sus siglas en inglés) para la detección de entorno y obstáculos y el Visual-SLAM, que utiliza sensores visuales como las cámaras [7].

LiDAR-SLAM es un dispositivo que utiliza laser como sensores y que por lo general gira 360° sobre su eje, emitiendo el láser al objeto y capturando el tiempo de viaje del haz de luz y poder obtener diferentes distancias entre el robot y lo que lo rodea realizando así el mapa del entorno [8]. LiDAR está clasificado en los métodos de nube de puntos y de cuadrículas, capaces de generar mapas de cuadrícula 2D y 3D [9]. Los sensores LiDAR 2D son hardware de escaneo de una sola línea. Estos, son sensores comúnmente usados para SLAM, por su alta precisión de mapeo, bajo costo de procesamiento de datos (en comparación con un LiDAR 3D) y alta robustez al desenfoque producido por el movimiento, además no es alterado por la iluminación [10].

Un sistema LiDAR efectivo consiste en un *front-end* para la obtención de los datos y un *back-end* para realizar la optimización y procesamiento de la data recibida para generar el mapa. Múltiples algoritmos se han utilizado para el *front-end*, como para el *back-end*. Entre ellos, ICP (*Iterative Closest Point*) [11], NDT (*Normalized Distribution Transform*) [12] y CSM (*Correlative Scan Matcher*) [13] para el *front-end* y iSAM [14], iSAM2 [15] y g2o [16] para el *back-end*.

A pesar de las bondades del LiDAR, se pueden presentar problemas en algunos entornos complicados, como lo son escenarios con componentes similares y pocas geometrías redundantes o diferenciadoras, lo que degrada en alto nivel el rendimiento de la asociación de los datos [17] y por lo tanto la generación del mapa y la auto localización [18]. El otro escenario son los entornos a gran escala, con altos tiempos y movimientos para el mapeo, donde se repiten las problemáticas del primer entorno.

Visual-SLAM es comúnmente usado en robots móviles debido al bajo coste, ligereza, bajo consumo de energía y amplia gama de los sensores tipo cámara que se pueden utilizar [19]. Cómo sensor puede utilizarse cualquier cámara digital con conexión USB, Wifi o Bluetooth, desde una Panda WebCam genérica de 3Mpx de no más de 10 USD, pasando por las cámaras integradas en cualquier celular o computador con tecnología igual o superior a 3G, hasta cámaras de última tecnología como una Tenveo VA3000 con precios superiores a los 500 USD.

Los tipos de cámara que comúnmente se usan para Visual SLAM son las monoculares, RGB, omnidireccionales (captura de imágenes simultáneas en múltiples direcciones), estéreo (dos cámaras RGB calibradas) o RGB-D (captura, además de imágenes RGB, información de profundidad del píxel) [20].

Sin embargo, Visual-SLAM es sensible a texturas, luces y geometrías [21], además de que requiere de calibración fotométrica y es computacionalmente costoso [22], ya que para generar el mapa, se debe procesar la imagen que recibe el sensor, encontrar puntos clave (referencias geométricas como vértices y aristas) y posicionarlos de acuerdo con las características del sensor, como el punto focal, pixeles, distorsión, rectificación y proyección. Teniendo que realizar estos cálculos para cada imagen generada en el tiempo de acuerdo con la capacidad del sensor en FPS (*Frames Per Second* por sus siglas en inglés). Por ejemplo, el Visual-SLAM basado en cámaras RGB-D tiene susceptibilidad al cambio de iluminación y solo funciona en interiores [23].

Del Visual SLAM salen otros modelos que usan el mismo principio, pero usando diferentes algoritmos y tipos de procesamiento de las imágenes. Algunos son ORB-SLAM, OpenVSLAM, Ground-SLAM [24] y el SLAM basado en marcadores. Los primeros 3 métodos se basan en métodos de identificación de puntos clave [25], que detectan características distintivas del entorno. Sin embargo, la coincidencia de puntos clave o *key points* (KP) tiene características limitadas en cuanto a escala del mapa y entornos con texturas repetitivas, lo que en algunos casos lo hace incapaz de identificar una escena desde varios puntos de vista [26]. SLAM basado en marcadores, no cuenta con estas limitaciones, pues se basa en un algoritmo de reconocimiento y localización de marcadores (códigos QR) planos, donde cada uno es un cuadro negro y un código binario interno para su identificación. Un solo marcador proporciona cuatro puntos que corresponden a las esquinas y se pueden localizar con precisión de subpíxeles para obtener una estimación precisa de la pose de la cámara [27]. En un entorno grande se puede disponer de varios marcadores con el fin de que el robot pueda desplazarse y se siga obteniendo la posición de este con respecto al mapa.

El objetivo final de desarrollar todos estos métodos SLAM es implementarlos en la navegación autónoma de los robots. En la universidad EAFIT, sede Medellín, se tiene planteado implementar un robot con capacidad de implementación de SLAM para ayudar al equipo de la bodega logística a trasladar materiales desde el punto de almacenamiento, hasta el punto de entrega. Para esto, se diseñó a la medida un robot.

El robot de la universidad EAFIT, fue diseñado con un diámetro de 50cm, altura de 30cm, capacidad de carga de 80kg, 4 ruedas con 2 servomotores independientes acoplados a 2 de las ruedas, cámara web Logitech C270 y sensor LiDAR. Como este cuenta con una circunferencia de 50 cm y los pasillos del laboratorio de logística tienen un ancho de 60 cm, se tiene una holgura de ±5 cm. Estas características involucran que el robot necesite de una gran precisión y exactitud a la hora de mapear y de localizarse, para así no colisionar con algún obstáculo.

También, se cuenta con otro robot para la investigación tipo TurtleBot. Los robots tipo TurtleBot, son plataformas móviles generalmente de geometría circular, con 2 ruedas con servomotores independientes para cada rueda. Cuenta con acoples para accesorios como sensores y actuadores.

Para obtener la precisión que permita localizar el robot en el rango de holgura de los +- 5cm, es que se le implementa el LiDAR y el Visual-SLAM en paralelo, ya que se cuenta con ambos sensores implementados en el TurtleBot. El uso de más de 1 método de SLAM es comúnmente usado en la navegación autónoma para obtener mejores resultados en el mapeo y seguimiento. Por ejemplo, la combinación de RGB-D SLAM con odometría de las ruedas. Este método consiste en fusionar el SLAM basado en los sensores RGBD y la geometría de las llantas que ayuda a resolver los problemas de relocalización en los SLAM basados únicamente en visión en entornos repetitivos [28]. Lograron pasar de un error relativo de 2.7% (puramente con Visual-SLAM RGB-D) a 0.41% (combinación de ambos métodos). Otra combinación de métodos es CVI-SLAM, colaboración Visual-SLAM y la adición de un sensor inercial con el cual se pueden obtener las aceleraciones y velocidades angulares, pudiendo así ayudar a la localización y relocalización de cada robot [29] [30]. Con esta combinación de métodos, se pasó de un error cuadrático medio de 0.615% (puramente con Visual-SLAM tipo ORB-SLAM) a 0.526% (combinación de ambos métodos). Por último, caso, se presenta Uco-SLAM, una combinación entre Visual-SLAM basado en marcadores y ORB-SLAM. Como la desventaja común de ORB-SLAM, es que los mapas generados no cuentan con una escala definida, fallan cuando se da un movimiento de rotación puro, requieren de diversidad en texturas para crear los KP y la relocalización falla cuando hay patrones repetitivos lo que los hace inútiles para la navegación autónoma, se utiliza Visual-SLAM basado en marcadores, debido a que los marcadores eliminan el problema de relocalización al contar con un respectivo ID, eliminando así la ambigüedad en los entornos repetitivos y los falsos cierres de bucles. Elimina tanto el problema de las texturas como el de la escalabilidad, pues el marcador frente a temas de iluminación sigue siendo el mismo y se conoce su tamaño real [31]. Para este caso, se pasa de una precisión de 89.15% (puramente con Visual-SLAM tipo ORB-SLAM) a 99.83% (combinación de ambos métodos).

Como uno de los métodos combinados es LiDAR y se sabe que presenta problemas en escenarios con componentes similares y pocas geometrías redundantes o diferenciadoras, se usa el Visual-SLAM basado en marcadores, que como se evidencia en Uco-SLAM, se eliminan estos problemas de ambigüedad.

La fusión de LiDAR y Visual-SLAM basado en marcadores se da gracias a la implementación de un algoritmo *Extended Kalman Filter*. Para entender que es un *Extended Kalman Filter* (EKF), primero hay que comprender el algoritmo de *Kalman Filter*, este es un modelo matemático lineal, utilizado en el control automático y procesamiento de señales. En este se tiene en cuenta el ruido de las señales para estimar la incertidumbre y dar un estado estimado real de la posición [32]. Ahora, el EKF es una extensión del *Kalman Filter*, pero este, en lugar de usar un modelo lineal para el sistema, utiliza una aproximación lineal del modelo para calcular las estimaciones. En otras palabras, EKF se utiliza para sistemas no lineales y *Kalman Filter* para sistemas lineales [33].

EKF se utiliza en SLAM, para estimar la ubicación del robot y construir el mapa a partir de las mediciones de los sensores. Una forma de aplicarlo es usar EKF en dos etapas, la primera consiste en determinar el modelo no lineal de movimiento del robot junto con las mediciones de los sensores. La segunda, una vez definido el modelo de movimiento, se puede utilizar aproximaciones lineales del modelo, para dar la estimación de ubicación del robot y actualizar la construcción del mapa [34], de acuerdo con la Ilustración 1.

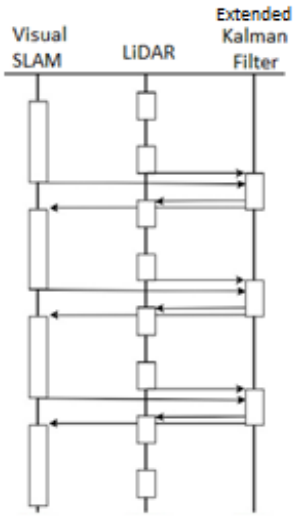


Ilustración 1-Integración de EKF a Visual SLAM y LiDAR

Elaboración propia

Al conocer los conceptos de SLAM, su uso, tipos de SLAM, el proyecto de la universidad EAFIT y lo que se ha implementado y cómo se ha implementado, se presenta el estudio de esta tesis de investigación, la cual se enfoca cuantitativamente en la navegación autónoma de un robot implementando Visual-SLAM basado en marcadores y LiDAR. Investigaciones pasadas han creado mapas utilizando marcadores, como la realizada por Rafael Muñoz-Salinas [35] en 2018, donde implementó marcadores del tipo ArUco a todo lo largo de una habitación para detectarlos con una cámara, ubicarlos globalmente en el espacio. Así, lograron obtener la posición de la cámara globalmente sin importar si estaba en movimiento, rotando o ambas, siempre y cuando estuviera visualizando al menos 1 marcador. Pero, aunque estas investigaciones se enfocaron en la localización y relocalización, solo utilizan imágenes para generar los mapas y no un robot en movimiento. Tampoco entran en el área de la navegación autónoma.

Por lo tanto, el objetivo de este estudio es determinar la mejor forma de implementar Visual-SLAM basado en marcadores en un robot en movimiento tipo TurtleBot de un punto A hacia un punto B, en un mapa generado únicamente por marcadores tipo ArUco, usando una cámara monocular, que comparado con un *groundtruth* (movimiento/ubicación real del robot) se obtenga la mejor precisión en centímetros de una trayectoria. Como objetivo segundario, se tiene el encontrar la diferencia en centímetros del mapa real comparado con el mapa creado a partir de marcadores tipo ArUco.

La obligación de que la cámara esté todo el tiempo observando marcadores para su localización, nos hace plantear la hipótesis de que en algún momento de desconexión o perdida de fps, se debe implementar un algoritmo que permita reubicar al robot luego de que vuelva a recuperar la señal y pueda seguir con su trayectoria. Adicionalmente esperamos encontrar una diferencia con respecto a la posición real del robot con respecto a la programada en la navegación autónoma, debido a los problemas de ambigüedad y fallas en la precisión del método Visual-SLAM. Todo esto con el fin de conocer si es posible usar mapeado y navegación autónoma con Visual-SLAM basado en marcadores para el robot de la universidad EAFIT-Medellín y obtener una tolerancia de posición de ± 5cm.

2 PLANTEAMIENTO DEL PROBLEMA

Como problemática, se plantea la siguiente pregunta a resolver ¿Cuál es la mejor forma de implementar Visual-SLAM basado en marcadores ArUco para un robot en movimiento tipo TurtleBot de 50 cm de diámetro, donde se obtenga la mejor precisión de localización con respecto a un *groundtruth* ?

3 OBJETIVOS

3.1 GENERAL

Estudiar la precisión del Visual-SLAM basado en ArUco como método de localización y de generación de *groundtruth* y determinar si puede ser usado para un robot tipo TurtleBot de 50 cm de diámetro, donde pueda realizar actividades de navegación autónoma con precisión de ± 5cm.

3.2 ESPECÍFICOS

1) Encontrar donde ubicar el sensor usado para el SLAM en el robot tipo TurtleBot.
2) Determinar cuál es la mejor ubicación en el espacio disponible (para la investigación) para el sensor utilizado para el *groundtruth*.
3) Analizar la precisión del mapa generado en comparación con el real.
4) Definir los mejores métodos para comparar los datos del SLAM y el *groundtruth*.
5) Comparar los valores de SLAM con el *Groundtruth*.
6) Analizar los resultados de la comparación, para verificar la viabilidad de implementar este método para el modelo de TurtleBot.

4 ESTADO DEL ARTE

Al analizar distintos artículos de investigación referentes a SLAM visual basado en marcadores, se identifica los siguientes puntos clave al tener en cuenta en el desarrollo del proyecto, los cuales son las diferentes implementaciones de *groundtruth*, el uso de ROS como aplicación implementada para el SLAM, los diferentes tipos de marcadores que existen y el uso específico de ArUco como marcador para SLAM. A continuación, se presenta el estado del arte para cada segmento.

4.1 GROUNDTRUTH

El *Groundtruth*, o verdad de referencia, desempeña un papel fundamental en la evaluación y validación de los sistemas SLAM. Se refiere a la ubicación y orientación precisa del robot en un entorno conocido. Para establecer un *Groundtruth* confiable, se han empleado diversas técnicas, como sistemas de seguimiento por GPS de alta precisión en exteriores, sistemas de medición de posición en interiores basados en tecnologías como LIDAR o cámaras de seguimiento óptico. La elección del método de *Groundtruth* depende en gran medida del entorno y los objetivos específicos de la aplicación.

Los siguientes son algunos de los métodos que hablan de como implementaron diferentes modelos de *groundtruth*:

4.1.1 Groundtruth Evaluation of Large Urban 6D SLAM [36]

Los algoritmos para resolver el problema de localización y mapeo simultáneo (SLAM) en robótica móvil son fundamentales para la investigación. Aunque los métodos de SLAM producen mapas coherentes, estos pueden ser incorrectos, requiriendo experimentos de referencia [37]. Algoritmos populares usan estimaciones de posición 3DoF, pero la tendencia actual es hacia el 6D SLAM [37]. La precisión en ambientes reales implica considerar 6 grados de libertad. Este artículo evalúa algoritmos de mapeo autónomo en un entorno urbano grande. Mapas generados por algoritmos en línea y fuera de línea se comparan con estimaciones de posición basadas en odometría, giroscopios y GPS. El *Groundtruth* es proporcionado por una Localización de Monte Carlo (MCL) usando mapas de referencia precisos.

En el estado del arte para mapeo, los métodos probabilísticos integran modelos de movimiento y percepción, permitiendo localizar al robot y mapear. Los lazos cerrados son esenciales para modelar áreas previamente visitadas y asegurar la coherencia topológica. Thrun [38] y otros presentan enfoques como el filtro de Kalman extendido y FastSLAM. Aunque es posible extender estos métodos a 6D, las estrategias para reducir el costo computacional y la extracción confiable de características aún no son suficientes para ser implementadas de manera confiable. Con esta investigación, se busca mejorar la precisión y eficiencia de escaneo rotando o inclinando el escáner.

Este artículo introduce una nueva técnica de evaluación para algoritmos de SLAM basada en los resultados finales y una posición de referencia obtenida independientemente del algoritmo probado. Se utiliza una técnica que se basa en mapas topográficos altamente precisos obtenidos de oficinas de registros en una urbe en Alemania.

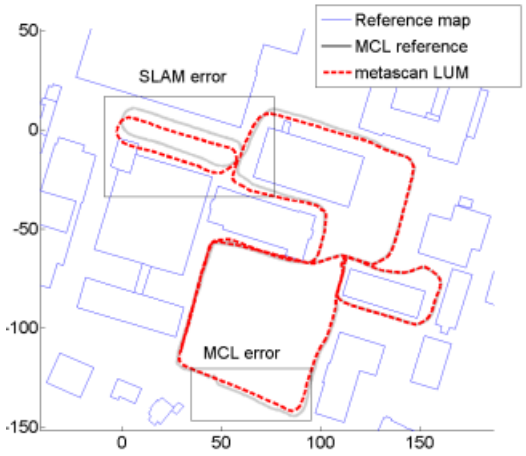


Ilustración 2-Trayectorias MCL y LUM

Tomada de: Groundtruth Evaluation of Large Urban 6D SLAM

MCL y el algoritmo SLAM bajo prueba usan los mismos datos, aunque los resultados y las posiciones de referencia no son completamente independientes. Los algoritmos de localización y mapeo funcionan de manera diferente, eliminando errores acumulativos.

Se usa una técnica de GPS, odometría y escaneo láser 3D, donde se aplica el algoritmo *Iterative Closest Points* (ICP) para la transformación [39]. Se implementa el método Virtual 2D *Scans* para emparejar los datos del escáner 3D con el mapa de referencia 2D en la Localización de Monte Carlo (MCL) [40]. Esta técnica proporciona una forma de generar posiciones de referencia confiables en entornos interiores y exteriores urbanos.

Se verifican condiciones cruciales: suficientes puntos de referencia en datos del sensor [41], condiciones numéricas del filtro de partículas [42] y evaluación global de resultados. Las posiciones MCL y SLAM se comparan en sistemas de coordenadas globales [37]. Algunas métricas de desempeño objetivo incluyen distancia euclidiana, orientación, y desviación estándar. El experimento se realizó en la Universidad *Leibniz Universitat Hannover* con 924 escaneos 3D y posiciones GPS.

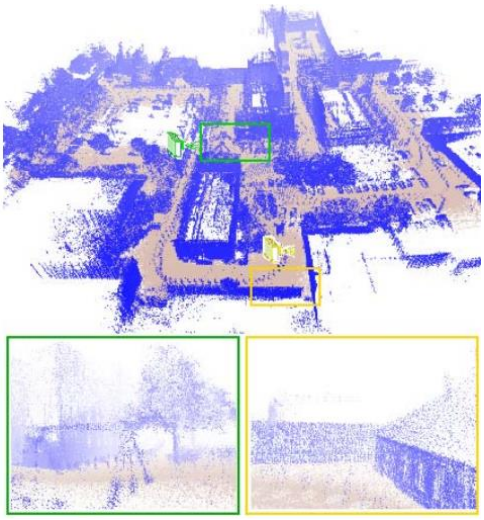


Ilustración 3-Modelo escaneos MCL

Tomada de: Groundtruth Evaluation of Large Urban 6D SLAM

Los resultados se validan con mapas topográficos altamente precisos. La comparación de resultados entre Monte Carlo Localization (MCL) y el método implementando LiDAR para SLAM y uso de ICP, destaca el rendimiento superior de la implementación de ICP. También se evalúan los métodos LUM, que mejoran el cierre de bucles. Los resultados fueron los siguientes:

ORIENTATION ERRORS [DEG]

method	σ_θ	$e_{\theta,max}$
Odometry	77.2	256.6
OdometryGyro	15.1	56.7
GPS	27.3	171.0
pairwise ICP	6.3	17.7
metascan ICP	2.4	11.8
pairwise LUM	5.2	22.8
metascan LUM	4.3	21.2

POSITION ERRORS [M].

method	σ	e_{max}
Odometry	55.1	261.2
OdometryGyro	64.7	250.1
GPS	5.8	95.1
pairwise ICP	5.2	21.8
metascan ICP	1.6	6.6
pairwise LUM	4.9	17.0
metascan LUM	3.8	13.8

Ilustración 4-Valores de error en la orientación y posición MCL

Tomada de: Groundtruth Evaluation of Large Urban 6D SLAM

4.1.2 A Benchmark for Multi-Modal LiDAR SLAM with Groundtruth in GNSS-Denied Environments [43]

Los sensores LiDAR se utilizan en una variedad de aplicaciones, desde los coches autónomos hasta la topografía forestal. Los LiDAR's giratorios de alta resolución permiten una gran conciencia de los entornos circundantes, pero son caros. Los nuevos LiDAR's de estado sólido son más baratos y proporcionan un rango de detección significativamente mayor.

Este artículo presenta una referencia que compara diferentes modalidades de LiDAR (giratorio, de estado sólido) en entornos diversos. Para permitir una comparación más precisa y justa, los autores introdujeron un nuevo método para la generación de la verdad del terreno en espacios interiores más grandes.

Los autores también evaluaron diez algoritmos SLAM de última generación basados en filtros. Los resultados indican las limitaciones de los algoritmos SLAM actuales y las posibles direcciones de investigación futura.

Los algoritmos 3D LiDAR SLAM se han estudiado mucho como un componente crucial de los sistemas robóticos y autónomos. Los principales tipos de algoritmos 3D LiDAR SLAM son *LiDAR-only*, *loosely-coupled* o *tightly coupled* con datos IMU.

Los algoritmos *LiDAR-only* son limitados por su alta susceptibilidad a paisajes sin características. Los algoritmos *tightly coupled* son más precisos y flexibles, y han demostrado ser más eficaces que los *LiDAR-only* y *loosely coupled*.

Los algoritmos 3D LiDAR SLAM se han aplicado a una variedad de sensores, incluidos los LiDAR's giratorios y los LiDAR's de estado sólido. Los LiDAR's de estado sólido tienen un campo de visión más pequeño y un muestreo irregular, pero son más baratos y ligeros que los LiDAR's giratorios.

En este artículo, se presenta una plataforma de recolección de datos que se monta en un vehículo móvil con ruedas para adaptarse a entornos variables. La plataforma se mueve manualmente o tele-operada, excepto en el entorno forestal, donde se lleva a mano.

Los autores presentan una plataforma de recolección de datos que contiene varios sensores LiDAR, desde LiDAR's giratorios tradicionales con diferentes resoluciones, hasta nuevos LiDAR's de estado

sólido con patrones de escaneo no repetitivos. También se incluye una cámara LiDAR y una cámara estereoscópica de ojo de pez.

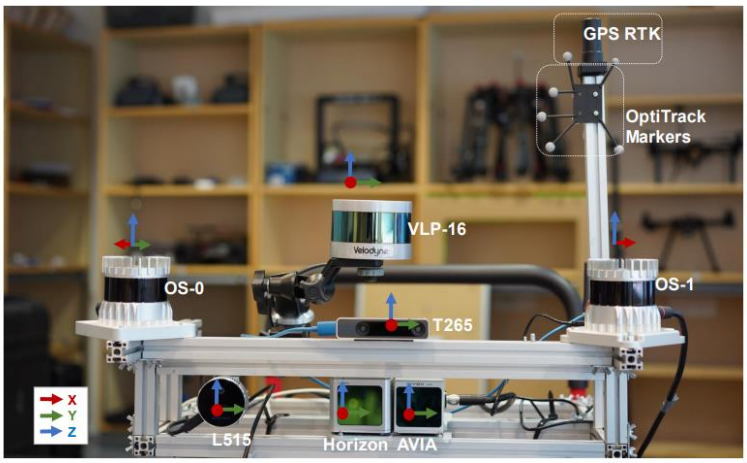


Ilustración 5-Ubicación de sensores LiDAR

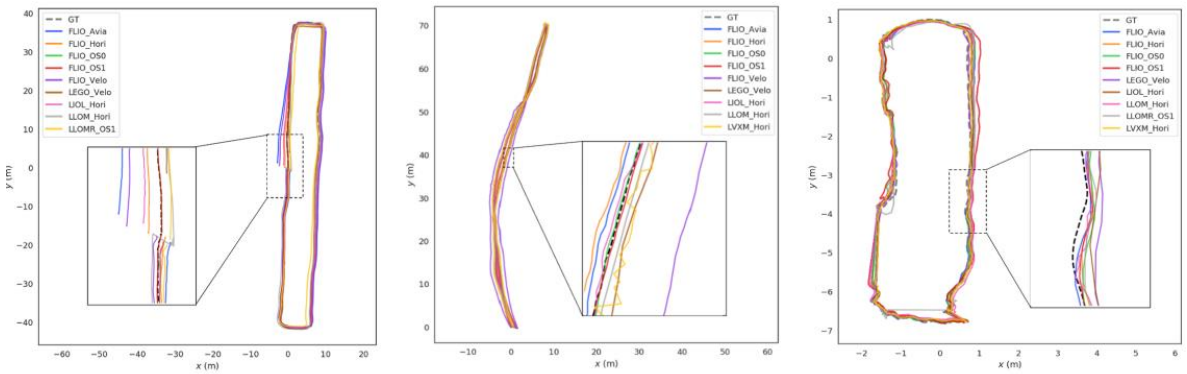
Tomada de: A Benchmark for Multi-Modal LiDAR SLAM with Groundtruth in GNSS-Denied Environments

Para proporcionar una verdad del terreno precisa para grandes entornos interiores y exteriores, los autores proponen un marco de generación de mapa de tierra basado en LiDAR de estado sólido asistido por SLAM. El marco consiste en tres pasos:

- Recolección de sub-mapas de nube de puntos sin distorsión a partir de los datos de los sensores LiDAR.
- Correspondencia y fusión de las sub-mapas en un mapa global mediante ICP.
- Eliminación de ruido del mapa generado para obtener un mapa de tierra denso y de alta definición.

Los autores validan el marco de generación de mapa de tierra evaluando su precisión en dos secuencias de datos de interiores. Los resultados muestran que el marco puede generar mapas de tierra precisos con una densidad de puntos de hasta 240.000 puntos por sub-mapa.

En este estudio, se evaluaron algoritmos populares de SLAM 3D LiDAR en múltiples secuencias de datos de varios escenarios, incluidos entornos interiores, exteriores y forestales. Los resultados mostraron que el algoritmo FAST-LIO con láser giratorio de alta resolución tiene el rendimiento más robusto en entornos interiores y exteriores, incluso en escenarios desafiantes como pasillos largos. Los sistemas de SLAM basados en láser sólido funcionan bien en entornos exteriores, pero son menos precisos en entornos interiores debido a la menor resolución de los sensores.



(a) trajectory comparison of sequence *Indoor10* (b) trajectory comparison of sequence *Road03* (c) trajectory comparison of sequence *Forest01*

Ilustración 6-Diferentes trayectorias calculadas por LiDAR

Tomada de: A Benchmark for Multi-Modal LiDAR SLAM with Groundtruth in GNSS-Denied Environments

En este estudio, se evaluaron 5 algoritmos de SLAM de código abierto en términos de odometría de LiDAR, consumo de energía y precisión. Los experimentos se realizaron en 9 secuencias en 2 plataformas de computación. Se encontró que el algoritmo FLIO basado en láser giratorio tiene un buen rendimiento con bajo consumo de energía en entornos interiores y exteriores. Sin embargo, en el entorno forestal, el algoritmo LIOL basado en láser sólido tiene el mejor rendimiento en términos de precisión y calidad de mapeo.

Sequence	FLIO_OS0	FLIO_OS1	FLIO_Velo	FLIO_Avia	FLIO_Hori	LLOM_Hori	LLOMR_OS1	LIOL_Hori	LVXM_Hori	LEGO_Velo
Indoor06	**0.015 / 0.006**	0.032 / 0.011	N/A	0.205 / 0.093	0.895 / 0.447	N/A	0.882 / 0.326	N/A	N/A	0.312 / 0.048
Indoor07	**0.022 / 0.007**	0.025 / 0.013	0.072 / 0.031	N/A	N/A	N/A	N/A	N/A	N/A	0.301/0.081
Indoor08	0.048 / 0.030	**0.042 / 0.018**	0.093 / 0.043	N/A	N/A	N/A	N/A	N/A	N/A	0.361 / 0.100
Indoor09	**0.188 / 0.099**	N/A	0.472 / 0.220	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Indoor10	0.197 / 0.072	**0.189 / 0.074**	0.698 / 0.474	0.968 / 0.685	0.322 / 0.172	1.122 / 0.404	1.713 / 0.300	0.641 / 0.469	N/A	0.930 / 0.901
Indoor11	0.584 / 0.080	**0.105 / 0.041**	0.911 / 0.565	0.196 / 0.098	0.854 / 0.916	0.1.097 / 0.0.45	1.509 / 0.379	N/A	N/A	N/A
Road03	0.123 / 0.032	**0.095 / 0.037**	1.001 / 0.512	0.211 / 0.033	0.351 / 0.043	0.603 / 0.195	N/A	0.103 / 0.058	0.706 / 0.396	0.2464 / 0.063
Forest01	0.138 / 0.054	0.146 / 0.087	N/A	0.142 / 0.074	0.125 / 0.062	0.116 / 0.053	0.218 / 0.110	**0.054 / 0.033**	0.083 / 0.041	0.064 / 0.032
Forest02	0.127 / 0.065	**0.121 / 0.069**	N/A	0.211 / 0.077	0.348 / 0.077	0.612 / 0.198	N/A	0.125 / 0.073	0.727 / 0.414	0.275 / 0.077

Ilustración 7-Tabla de comparación de resultados con cada LiDAR

Tomada de: A Benchmark for Multi-Modal LiDAR SLAM with Groundtruth in GNSS-Denied Environments

4.1.3 High-accuracy Fiducial Markers for Groundtruth [44]

Los algoritmos de odometría visual (VO) se utilizan para estimar la posición y orientación de un vehículo a partir de imágenes de su entorno. Sin embargo, la evaluación de estos algoritmos requiere una base de referencia precisa, que es difícil de obtener en entornos exteriores.

Los conjuntos de datos de VO al aire libre suelen utilizar un sistema de navegación inercial (INS) de alta gama como base de referencia. Sin embargo, estos INS son pesados y consumen mucha energía, lo que los hace poco prácticos para su uso en vehículos aéreos no tripulados (UAV) pequeños.

Este artículo propone un sistema que utiliza marcadores de alta precisión para proporcionar una base de referencia para la evaluación de VO en UAV. Los marcadores se pueden colocar a lo largo de la ruta de vuelo del UAV con poses medidas con precisión.

Este sistema tiene varias ventajas sobre los métodos tradicionales de base de referencia. Es más ligero y consume menos energía que un INS de alta gama, y es más preciso que la odometría láser.

Los marcadores cuadrados se han utilizado en el campo de la realidad aumentada durante muchos años. ARToolkit es quizás el más conocido. Los pasos básicos del algoritmo son el umbralización de la imagen, la búsqueda de regiones con forma de cuadrilátero y el uso de las esquinas para calcular una homografía.

Los marcadores cuadrados tienen muchos inconvenientes, por lo que se han propuesto varios enfoques, como ARTag, ArUco y AprilTag. Estos enfoques se centran en la robustez a la oclusión y la rotación.

Los marcadores se utilizan para estimar la pose de un objeto en una imagen. La precisión de la estimación de la pose se ve afectada por varios factores, como la distorsión geométrica, el ruido y la distancia al marcador.

La distorsión geométrica se introduce durante la impresión, la captura de imágenes y el procesamiento de imágenes. El ruido se introduce durante la captura de imágenes y el procesamiento de imágenes. La distancia al marcador afecta a la precisión de la estimación de la pose, ya que la distancia afecta a la cantidad de distorsión geométrica y ruido.

Para mejorar la precisión de la estimación de la pose, los sistemas de marcadores deben reducir la distorsión geométrica, el ruido y la sensibilidad a la distancia.

RUNE-Tag es un marcador fiducial de alta precisión que utiliza anillos concéntricos de puntos. Tiene una excelente resistencia a la oclusión y es resistente al ruido, el desenfoque e la iluminación desigual.

El concepto de marcador estructurado es de [45], que utiliza un marcador unidimensional con un patrón de sinusoide para la estimación de la distancia.

El marcador propuesto utiliza un patrón de sinusoide radial, como se muestra en la Ilustración 8.

Ilustración 8-Diseño de RUNE-Tag

Tomada de: High-accuracy Fiducial Markers for Ground

Al estimar la pose del marcador, se utiliza cada píxel del marcador, a diferencia de los marcadores tradicionales que solo utilizan esquinas o bordes.

El marcador se detecta primero en una imagen encontrando su contorno. Luego, se realiza una estimación aproximada de la pose utilizando las ubicaciones de las esquinas. La estimación aproximada de la pose se refina luego utilizando una serie de optimizaciones no lineales. El proceso

de optimización compara la imagen de la cámara con una representación del marcador en la pose estimada actual.

La principal limitación de este proceso es que la superficie de error debe ser lo suficientemente suave y convexa para que el proceso de optimización converja a un resultado útil. Dependiendo de los parámetros que se optimicen, la estimación inicial puede tener que estar cerca de los valores reales.

En la implementación actual, tx y ty (siendo Tx la posición en x y Ty la posición en y) se refinan en el primer paso utilizando solo la porción central del marcador. Este paso no es tan sensible al error en los otros parámetros, el error en θ y φ tiene un mayor efecto mientras estén más lejos del centro. A continuación, se refinan θ y φ, y finalmente tz (siendo Tz la posición en z).

El marcador propuesto utiliza un patrón de sinusoide radial. Se comparó con un marcador ArUco y un marcador AprilTag. El marcador propuesto mostró una disminución del doble en la desviación estándar de la posición del marcador en comparación con el marcador ArUco. El marcador AprilTag no funcionó bien en este experimento.

El patrón de sinusoide se eligió porque se pensó que los métodos de dominio de frecuencia podrían usarse para estimar la pose del marcador. Sin embargo, estos métodos no dieron buenos resultados.

El marcador propuesto no permite identificar marcadores individualmente. Sin embargo, esto no es estrictamente necesario para su uso como referencia. Una solución más sencilla para otras aplicaciones podría ser simplemente usar estos marcadores en conjunto con marcadores estándar.

4.1.4 Map Comparison of LiDAR-based 2D SLAM Algorithms Using Precise Groundtruth [46]

Este artículo presenta un método para evaluar mapas SLAM con precisión micrométrica utilizando el láser *tracker* FARO como referencia. Se detallan configuraciones de software y hardware, el sistema del robot y la metodología de experimentos. Se realiza un análisis comparativo de mapas generados por diferentes algoritmos SLAM y se concluye con futuras direcciones de investigación.

El algoritmo Gmapping 1 es un enfoque SLAM basado en láser que utiliza un filtro de partículas Rao-Blackwellizado [47]. Este enfoque supera problemas típicos de los filtros de partículas, como la complejidad computacional y la reducción de precisión debido a la eliminación de partículas durante el remuestreo. Gmapping mejora la localización del robot al integrar observaciones sensoriales recientes con el modelo de movimiento de odometría, reduciendo la incertidumbre.

Por otro lado, Cartographer 2 es un sistema de SLAM en tiempo real en 2D y 3D con múltiples configuraciones de sensores [48]. Utiliza un enfoque de mapeo de cuadrícula que evita la acumulación de errores durante iteraciones largas. Realiza un emparejamiento de escaneo iterativo con sub-mapas y comprueba automáticamente el cierre de bucle. Cartographer se basa en *Ceres scan matching* [49] para encontrar poses de escaneo más precisas.

Hector SLAM 3 es un sistema 2D SLAM que combina el emparejamiento robusto de escaneo láser y la navegación 3D con un sistema de sensores inerciales mediante un filtro de Kalman Extendido (EKF). Está diseñado para la computación a bordo de la posición en tiempo real del robot en 6 grados de libertad (6DOF) con mapeo 2D basado en LiDAR de alta velocidad de actualización [50].

Los experimentos se llevaron a cabo con el robot móvil de tracción diferencial llamado Plato equipado con un ordenador Jetson TX1, sensores de choque, sonares, módulo IMU Troyka, LiDAR 2D Hokuyo urg-04lx-ug01, LiDAR 3D Velodyne VLP-16 y codificadores de rueda para calcular la odometría de las ruedas. El ordenador a bordo del robot utiliza Ubuntu 16.04 con ROS Kinetic.

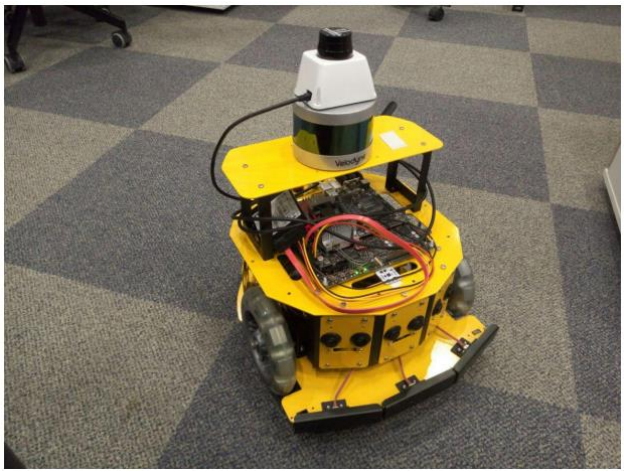


Ilustración 9-Diseño de robot implementado para SLAM LiDAR-based

Tomada de: Map Comparison of LiDAR-based 2D SLAM Algorithms Using Precise Groundtruth

Se utilizó un FARO *Laser Tracker* 4 para crear el mapa de referencia. Este sistema portátil permite mediciones 3D de alta precisión con una precisión de hasta 0.015 mm.

Para los experimentos en interiores, se eligió un aula en la que se reorganizaron las mesas para crear un circuito para la navegación del robot móvil. El mapa de referencia se creó construyendo planos de las paredes y el suelo a partir de puntos obtenidos con el FARO *Laser tracker*. Se realizaron cortes horizontales a la altura del LiDAR en la plataforma del robot móvil para obtener puntos 2D que se conectaron para crear el mapa.

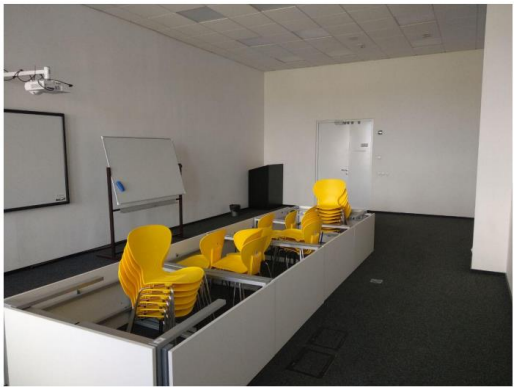
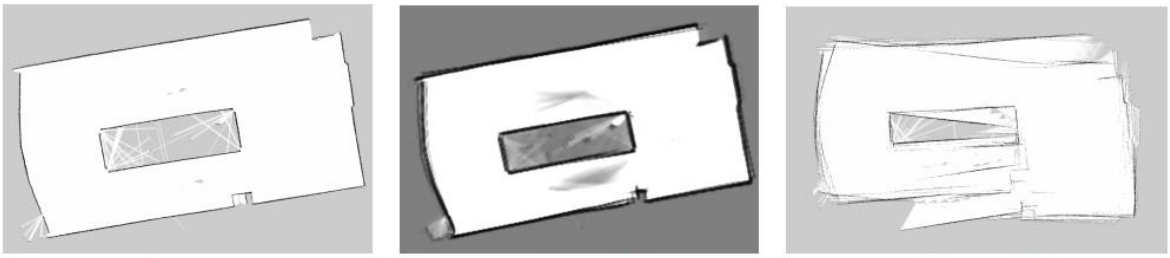


Ilustración 10-Entorno de mapa para LiDAR-based

Tomada de: Map Comparison of LiDAR-based 2D SLAM Algorithms Using Precise Groundtruth

Se registraron archivos ROS bag para poner a los algoritmos de SLAM en igualdad de condiciones. Estos archivos capturan datos de los temas ROS con muestreo temporal adecuado y se utilizaron en los algoritmos SLAM fuera de línea para obtener mapas en forma de Cuadrícula de Ocupación. La Cuadrícula de Ocupación representa un mapa en 2D en el que cada celda puede estar ocupada, libre o desconocida.



(a) Gmapping (b) Cartographer (c) Hector SLAM

Ilustración 11-Resultados de mapa para LiDAR-based

Tomada de: Map Comparison of LiDAR-based 2D SLAM Algorithms Using Precise Groundtruth

Los mapas de ocupación de los mapas guardados se modificaron para tener solo dos significados: ocupado y libre, simplificando la comparación. La alineación se realizó utilizando el algoritmo ICP (*Iterative Closest Point*) [51], calculando la métrica de distancia promedio al vecino más cercano (ADNN) como la suma de todas las distancias dividida por el número de celdas ocupadas. Los experimentos se llevaron a cabo en diferentes escenarios de movimiento del robot, y los resultados de la alineación se muestran en la siguiente tabla.

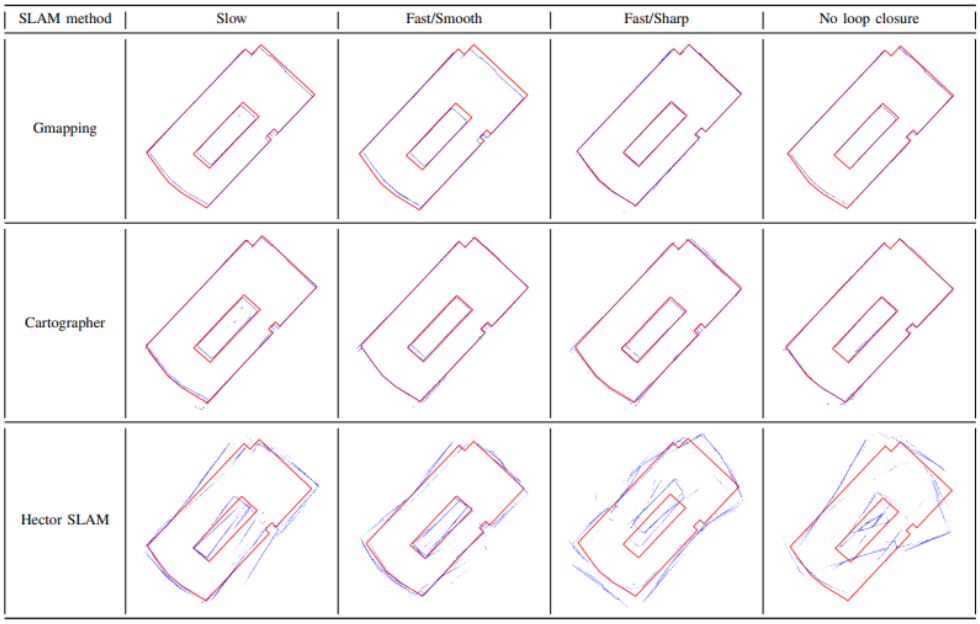


Ilustración 12-Tabla de comparación de resultados de LiDAR-based

Tomada de: Map Comparison of LiDAR-based 2D SLAM Algorithms Using Precise Groundtruth

Los resultados indican que Google Cartographer construye mapas con el menor error en la mayoría de los escenarios en comparación con el mapa de referencia. Gmapping también proporciona resultados aceptables en la construcción de mapas 2D, incluso sin cierre de bucle. Por otro lado, Hector SLAM, al utilizar solo datos LIDAR y carecer de opción explícita para el cierre de bucle, ofrece resultados menos precisos. En general, Google Cartographer se destaca como uno de los mejores algoritmos para generar mapas 2D con LIDAR en un robot móvil.

4.1.5 On construction of a reliable groundtruth for evaluation of visual slam algorithms [52]

Para evaluar la precisión de SLAM visual, se pueden usar conjuntos de datos públicos, pero a menudo es necesario crear uno propio debido a las diferencias en entorno y equipamiento. Usualmente, los conjuntos de datos públicos tienen una precisión de centímetros proporcionada por sistemas externos como GPS diferencial [53] o sistemas de captura de movimiento [54]. Sin embargo, estos sistemas son costosos. Proponemos usar el conjunto de datos para crear la verdad fundamental mediante el procesamiento fuera de línea con el algoritmo SLAM. Esto mejora la confiabilidad y la precisión del *groundtruth*. Evaluamos esta hipótesis comparando RGB-D SLAM con el sistema de captura de movimiento WhyCon [55] en un entorno desafiante. Nuestra aproximación se enfoca en la precisión de localización.

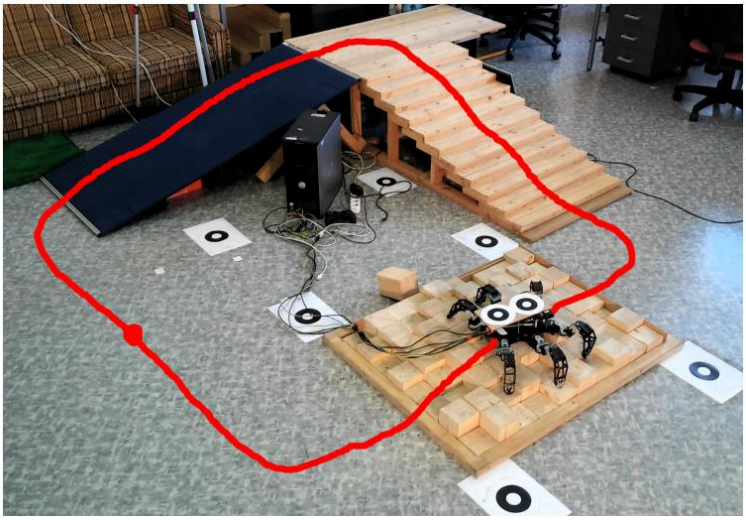


Ilustración 13-Mapa y trayectoria de "Reliable groundtruth"

Tomada de: On construction of a reliable groundtruth for evaluation of visual slam algorithms

Se propone utilizar un conjunto de datos capturado y procesarlo sin conexión con un sistema SLAM. Esta estimación de trayectoria resultante se utiliza como *groundtruth* para compararla con las trayectorias generadas en línea. Esto es útil cuando no se dispone de una localización externa precisa como *groundtruth*.

Las métricas de evaluación utilizadas son ATE (error absoluto de trayectoria) y RPE (error de posición relativa), como se presenta en [56]. ATE mide el error absoluto de todas las posiciones estimadas de 6 grados de libertad (6-DOF), dividiéndose en ATEt (distancias euclidianas) y ATEφ (error absoluto de orientaciones estimadas). RPE mide la deriva de la trayectoria estimada y se calcula con un intervalo de tiempo fijo (Δ = 1 en la evaluación). Puede dividirse en RPEt (error traslacional relativo) y RPEφ (error rotacional relativo).

Se diseñó un experimento práctico para evaluar la precisión de la localización utilizando el procesamiento sin conexión de un conjunto de datos SLAM visual capturado por un robot hexápodo en terreno complejo, con un *groundtruth* proporcionado por un sistema de localización externo. Se enfocó en la evaluación con una cámara de luz estructurada y el algoritmo RGB-D SLAM basado en [57], debido a su bajo costo y la información métrica que proporciona. Para obtener un *groundtruth* confiable. El conjunto de datos experimental se capturó con la cámara RGB-D Asus Xtion Pro y se registraron imágenes en color y mediciones de profundidad a 10 fotogramas por segundo.

El sistema RGB-D SLAM extrae características de la imagen RGB y utiliza la información de profundidad para encontrar una transformación rígida entre el fotograma actual y fotogramas

previamente mapeados. Se añade la pose estimada al mapa (grafo de poses) y se refina utilizando el algoritmo de optimización de grafo g2o [58]. Este proceso es especialmente beneficioso en caso de cierres de bucle grandes.

El entorno experimental incluyó obstáculos como escaleras, rampas y bloques de madera de diferentes alturas para simular desafíos que los métodos SLAM pueden enfrentar. El robot hexápodo se guio a lo largo de una trayectoria cuadrada de unos 9 metros de longitud. Los desafíos incluyeron giros rápidos en las esquinas, rotaciones angulares forzadas y movimientos de cámara impredecibles debido a la locomoción del robot hexápodo.

Se capturaron y analizaron cinco trayectorias para evaluar la precisión de localización. El *groundtruth* de las primeras tres trayectorias solo contiene información traslacional. El procesamiento tomó más de 4 horas por trayectoria. Los resultados se resumen en la Tabla y las trayectorias se muestran en la Figura.

	Frame-by-frame processing					Online processing				
Trial	ATE_t [cm]	ATE_ϕ [rad]	RPE_t [cm]	RPE_ϕ [rad]	End dist. [cm]	ATE_t [cm]	ATE_ϕ [rad]	RPE_t [cm]	RPE_ϕ [rad]	End dist. [cm]
No. 1	4.00	–	0.92	–	7.11	9.60	–	1.13	–	3.5
No. 2	4.32	–	0.54	–	12.54	8.13	–	0.56	–	8.93
No. 3	3.68	–	0.40	–	5.21	22.02	–	0.87	–	33.15
No. 4	9.97	0.08	1.21	0.01	9.13	12.50	0.09	1.31	0.01	7.12
No. 5	11.44	0.06	0.80	0.01	15.96	15.64	0.06	0.94	0.02	27.55

Ilustración 14-Tabla de resultados para "reliable groundtruth"

Tomada de: On construction of a reliable groundtruth for evaluation of visual slam algorithms

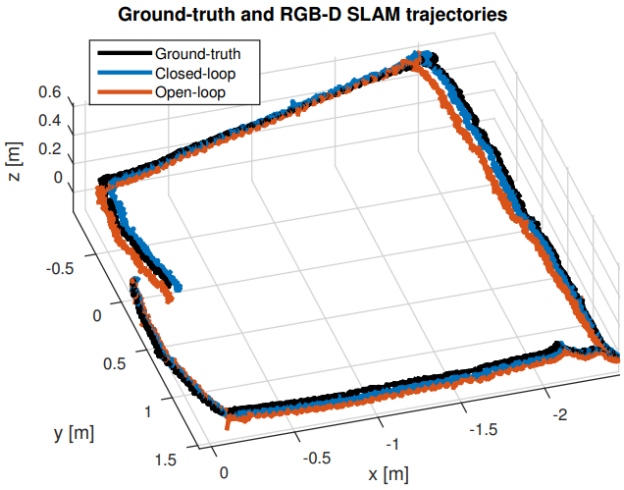


Ilustración 15-Diagramas de trayectorias calculadas para "reliable groundtruth"

Tomada de: On construction of a reliable groundtruth for evaluation of visual slam algorithms

La Tabla muestra que el ATEt promedio, utilizando el procesamiento de fotograma a fotograma, es aproximadamente el 1% de la longitud de la trayectoria y el RPEt promedio es inferior a 1 cm. Los valores de ATEt y RPEt son siempre más altos para el procesamiento en línea. Sin embargo, las

diferencias en ATEφ y RPEφ son despreciables a pesar de las pequeñas imperfecciones en la estimación de la orientación. Los resultados también indican que el procesamiento de fotograma a fotograma proporciona el mejor resultado obtenible por un método SLAM dado.

En conclusión, se propone un enfoque para obtener una verdad fundamental confiable en ausencia de sistemas de localización externa costosos. Este enfoque es aplicable a diferentes métodos SLAM y plataformas, y puede ser útil cuando no se dispone de sistemas de localización externa adecuados.

4.2 USO DE ROS EN LA APLICACIÓN

Robot Operating System (ROS) ha emergido como una plataforma popular para el desarrollo de aplicaciones de robótica debido a su modularidad y flexibilidad. La comunidad de ROS ha creado una amplia gama de paquetes y herramientas que facilitan la implementación de SLAM y la integración de sensores en sistemas robóticos. Esto lo convierte en una elección natural para aplicaciones de SLAM visual basado en marcadores, ya que proporciona un marco de trabajo sólido para el desarrollo y la experimentación.

Los siguientes artículos habla sobre porqué se implementa ROS para realizar SLAM y robótica, así como algunas de sus características:

4.2.1 ROS: an open-source Robot Operating System [59]

Este artículo presenta el *framework* ROS (*Robot Operating System*), diseñado para abordar los desafíos en el desarrollo de software para robots en crecimiento. ROS enfatiza la investigación en robótica a gran escala y la integración de software. Su arquitectura de procesos distribuidos y conectividad *peer-to-peer* permite un amplio uso de robots en red. Además, se destacan los siguientes objetivos filosóficos de ROS: ser *peer-to-peer*, basado en herramientas, multilingüe, liviano y de código abierto [60]. Aunque ROS se diseñó para robots de servicio, su arquitectura general es aplicable más allá de ese dominio [61].

La preferencia por lenguajes de programación en el desarrollo de código varía según factores como el tiempo de programación, la facilidad de depuración, la sintaxis y la eficiencia en tiempo de ejecución. ROS se ha diseñado para ser neutral en cuanto a lenguaje, admitiendo actualmente C++, Python, Octave y LISP. Para admitir diferentes lenguajes, se utiliza un lenguaje de definición de interfaz (IDL) que describe los mensajes enviados entre módulos. Los generadores de código crean implementaciones nativas en cada lenguaje compatible.

ROS utiliza una arquitectura de *microkernel* con numerosas herramientas pequeñas para construir y ejecutar componentes de ROS en lugar de un entorno monolítico. Esto mejora la estabilidad y la gestión de la complejidad. Se fomenta el desarrollo de controladores y algoritmos en bibliotecas independientes sin dependencias de ROS, lo que facilita la reutilización y las pruebas unitarias. ROS también reutiliza código de otros proyectos de código abierto, como Player, OpenCV y OpenRAVE, integrándolo de manera modular y minimizando el parcheo.

ROS es una plataforma de código abierto fundamental para la robótica. Su licencia BSD permite proyectos comerciales y no comerciales. La comunicación se basa en nodos, mensajes, temas y servicios. Los nodos son procesos que intercambian información mediante listas de datos estructurados. Los servicios son transacciones síncronas. Ofrece herramientas para el registro y reproducción de datos sensoriales, simplificando la investigación en percepción robótica. Además, brinda una interfaz de registro elegante llamada rosconsole para monitorear sistemas distribuidos.

El artículo aborda el uso de ROS en la investigación de robótica y su organización en paquetes. Se destaca el uso de roslaunch para facilitar el lanzamiento de sistemas de navegación y cómo ROS

fomenta la colaboración mediante la organización en paquetes. También se menciona la utilidad de rospack para gestionar dependencias y la flexibilidad en la estructura de los paquetes. Además, se describe cómo ROS permite la observación en tiempo real de datos del sistema, con herramientas como rviz y rostopic para visualizar y filtrar datos.

El sistema tf de ROS simplifica y unifica el tratamiento de marcos espaciales. Construye un árbol de transformaciones dinámicas que relaciona todos los marcos de referencia en el sistema. A medida que fluye información desde los diversos subsistemas del robot (codificadores de articulaciones, algoritmos de localización, etc.), el sistema tf puede generar transformaciones entre nodos en el árbol, calculando las transformaciones necesarias. Esto facilita tareas como generar nubes de puntos en un marco estacionario a partir de escaneos recibidos por un sensor en un robot en movimiento.

4.2.2 Enabling Codesharing in Rescue Simulation with USARSim/ROS [62]

Este texto destaca la importancia de utilizar plataformas de código abierto como ROS [63] para el desarrollo de sistemas de control robótico avanzados sin empezar desde cero. Se enfatiza que esto es esencial para acelerar el desarrollo de robots inteligentes y hábiles, como parte de la misión RoboCup.

Se menciona que ROS permite a los desarrolladores compartir módulos y centrarse en mejorar aplicaciones específicas. Además, se resalta la necesidad de simuladores precisos que incluyan modelos de ruido para sensores, como se discute en [64]. El simulador *Unified System for Automation and Robot Simulation* (USARSim), basado en *Unreal Developers Kit* (UDK), se presenta como una plataforma de simulación confiable utilizada por la comunidad de robótica.

USARSim se describe como un sistema de simulación basado en física de alta fidelidad que utiliza el motor de física PhysX y renderización 3D de alta calidad para crear entornos realistas. También se menciona que USARSim incluye modelos de entorno, robots comerciales y experimentales, y modelos de sensores.

Además, se destaca que ROS ofrece una abstracción para la configuración de hardware y software robótico complejo, fomenta la colaboración y proporciona herramientas para desarrolladores. Ejemplos de aplicaciones de ROS en la industria y la academia se mencionan, y se enfatiza su enfoque de código abierto y la disponibilidad de su código fuente para la comunidad.

Los parámetros de configuración se basan en la creación de robots en USARSim y en la creación de un árbol de transformaciones tf en ROS. La información de sensores se incluye automáticamente en el árbol de transformaciones. El control de vehículos en USARSim varía según el tipo de robot, pero se mapea automáticamente desde comandos de ROS en el tópico cmd_vel.

Diversos equipos [65] están desarrollando software de navegación de alto nivel al combinar algoritmos existentes de ROS con algoritmos de sus institutos. Estos equipos realizan la localización y mapeo simultáneos para generar mapas en línea y planificar rutas (por ejemplo, con el algoritmo RRT) para reemplazar la tele operación con navegación autónoma.

Este artículo presenta un nuevo paquete de ROS que permite la interfaz fluida entre USARSim y ROS. El paquete facilita la detección automática de robots y sensores, y genera los temas estándar de ROS que se esperarían de una plataforma física.

4.2.3 Multi Sensor Fusion for Navigation and Mapping in Autonomous Vehicles: Accurate Localization in Urban Environments [66]

El artículo subraya la trascendental importancia de la localización precisa en vehículos autónomos, especialmente en entornos urbanos densos, donde la localización se convierte en un aspecto crítico [67]. Se hace hincapié en la necesidad de emplear múltiples sensores para facilitar la navegación autónoma, destacando la riqueza de información proporcionada por datos visuales [68] y la precisión de las mediciones LiDAR para describir objetos desde una perspectiva geométrica [69].

El artículo también señala el auge en el desarrollo de vehículos autónomos para uso civil en la última década, liderado por empresas como Google desde 2009 [70], con la participación de numerosos actores de la industria, startups, y académicos. Este avance promete alterar la industria logística a nivel mundial, con camiones y barcos autónomos ya en etapas avanzadas de desarrollo y programados para operar en los próximos cinco o diez años [71]. Sin embargo, persisten desafíos tecnológicos y legales en la entrega de última milla [72].

El artículo se enfoca en el estudio y la comparación de diferentes métodos de localización para un pequeño robot de reparto en entornos urbanos densos. En estos escenarios, se utiliza un mapa existente del entorno operativo para lograr una localización más precisa mediante la correspondencia en tiempo real entre los escaneos y el mapa. Los autores analizan varias técnicas de localización, incluida la fusión de sensores, y proponen una estrategia para reconstruir áreas de un mapa local cuando los datos están dañados o el entorno ha sufrido modificaciones significativas.

En cuanto a la revisión relacionada, se destaca la tecnología de navegación autónoma a través de mapeo 3D con LiDAR's, que ha ganado popularidad en la última década debido a su alta precisión en la medición de rangos en comparación con otros sensores. Se menciona el uso de LiDAR para odometría y mapeo [73], con énfasis en la importancia de la estimación precisa de la posición en mapeo 3D. Se mencionan soluciones que combinan LiDAR y cámaras monocromáticas para mejorar la estimación de movimiento propio [74]. Además, se discuten técnicas de coincidencia de mapas, como *Normal Distributions Transform* (NDT), que permiten una localización con precisión de centímetros en entornos urbanos. Se citan mejoras al algoritmo NDT, como DENDT con *Differential Evolution* y se describe un conjunto de datos proporcionado por *JD Discovery Global Digitalization Challenge*, que incluye información GNSS, datos LiDAR, acelerómetros y giroscopios crudos, y velocímetros de ruedas. ROS se utiliza para procesar los datos. Se emplean cinco enfoques de localización que combinan diferentes sensores.

Se presenta la odometría/LiDAR como alternativa a la odometría/visual. Esta técnica extrae características de datos LiDAR y las compara en cuadros consecutivos, basándose en aspectos geométricos. Proporciona alta precisión en espacios abiertos con objetos diferenciados.

En los resultados experimentales, se aplicaron cinco enfoques al conjunto de datos proporcionado, mostrando errores de localización traslacional y rotacional. NDT++ mostró un error estable y mínimo en posición y rotación. NDT+ sin datos inerciales mostró errores mayores, pero con correcciones basadas en GNSS.

Para una comparación más detallada, se muestra la variabilidad del error de localización en gráficos y tablas. La IMU independiente tiene errores significativamente mayores. Los errores de translación se muestran con detalles sobre las coordenadas x e y. La GNSS muestra un desplazamiento negativo en x debido a condiciones atmosféricas o de entorno, que puede corregirse en tiempo real.

LOCALIZATION ERROR MEAN AND STADARD DEVIATION

	$\mu_{rot.}$	σ_{rot}	μ_x	σ_x	μ_y	σ_y
GNSS	1.52	0.79	-0.46	0.22	10^{-4}	0.18
IMU	-1.24	0.62	n/a	n/a	N/A	N/A
LOAM	-0.50	1.88	0.40	0.49	0.10	0.49
NDT+	0.03	0.87	-0.02	1.51	-0.05	1.13
NDT++	10^{-3}	0.31	-0.01	0.10	-0.05	0.10

Ilustración 16-Tabla de resultados para "Multi sensor fusion"

Tomada de: Multi Sensor Fusion for Navigation and Mapping in Autonomous Vehicles: Accurate Localization in Urban Environments

La distribución de errores GNSS se analiza en un histograma, revelando una simetría estrecha en los errores x e y. LiDAR con mapa 3D proporciona la mayor precisión en la localización, pero es esencial combinar datos de otros sensores para un enfoque más robusto y estable. GNSS e IMU son cruciales para la precisión y la estabilidad del algoritmo.

En resumen, la localización precisa en áreas urbanas densas es esencial para abordar la entrega autónoma de última milla. Se han explorado enfoques de localización en entornos urbanos utilizando datos LiDAR 3D complementados con GNSS e IMU. La coincidencia de escaneos 3D es el mejor enfoque cuando se combina con IMU y GNSS.

4.2.4 Comparison of ROS-based Visual SLAM methods in homogeneous indoor environment [75]

Este artículo presenta una comparación cualitativa de los métodos basados en sensores visuales monoculares, estéreo y RGB-D, verificada con *groundtruth*. Se utilizó un vehículo terrestre no tripulado (UGV) equipado con sensores y una plataforma de cómputo para realizar experimentos.

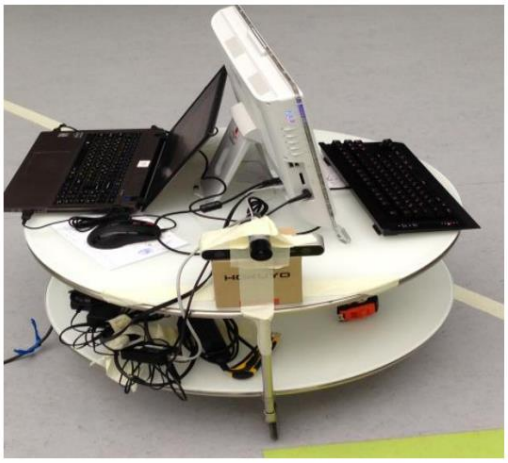


Ilustración 17-Robot implementado en "Comparison o ROS-based Visual SLAM"

Tomada de: Comparison of ROS-based Visual SLAM methods in homogeneous indoor environment

Se realizaron pruebas de trayectoria en un entorno típico de oficina para verificar algoritmos de SLAM en ROS [76]. Se utilizó una LIDAR 2D y el paquete Hector SLAM, que construye un mapa 2D basado

en escaneos de LIDAR en tiempo real. Se verificaron las trayectorias de otros algoritmos de SLAM (ORB-SLAM y DPPTAM) con respecto a la LIDAR.

ORB-SLAM es una biblioteca de SLAM en tiempo real basada en características para cámaras monoculares, estéreo y RGB-D. Utiliza el algoritmo de *Bundle Adjustment* para posicionar características en 3D, calculando así una trayectoria de cámara y recuperando una escena 3D dispersa. Este algoritmo utiliza el detector de características ORB (*Oriented FAST and Rotated* BRIEF), lo que le permite funcionar en tiempo real. ORB-SLAM implementa en ROS un nodo que procesa transmisiones monoculares en vivo, además de proporcionar cálculos necesarios y una interfaz gráfica.

DPPTAM, por otro lado, es uno de los métodos de SLAM visual más nuevos y directos en tiempo real. Estima una reconstrucción 3D densa de la escena y guarda la trayectoria como una secuencia de puntos en una nube de puntos. Se basa en la suposición de que regiones homogéneas y monocromáticas pertenecen a áreas aproximadamente planas. La implementación de DPPTAM en ROS es un nodo en tiempo real que no cuenta con una interfaz gráfica. Para la visualización de características se utiliza RViz y para nubes de puntos MeshLab.

Tras las pruebas, se obtuvo la trayectoria de ORB-SLAM en valores relativos y escalados, lo que reveló discrepancias con la trayectoria marcada. La discrepancia entre las trayectorias indica problemas de odometría visual con DPPTAM, ya que no coincide con la trayectoria marcada. Estos resultados ofrecen una evaluación crítica de la eficacia de estos métodos en la navegación en interiores.

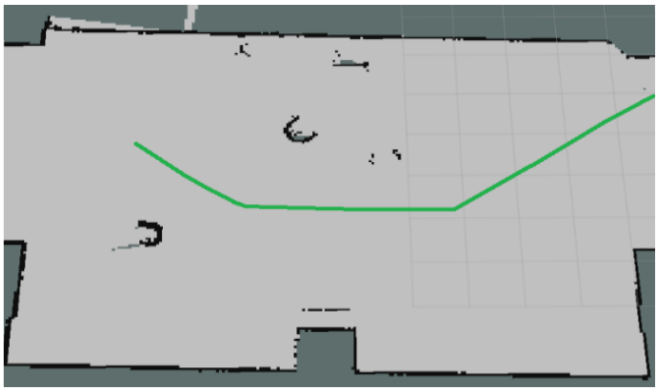


Ilustración 18-Mapa obtenido en ROS para "Comparison o ROS-based Visual SLAM"

Tomada de: Comparison of ROS-based Visual SLAM methods in homogeneous indoor environment

Se utilizaron herramientas y software de Stereolabs en conjunto con la cámara ZED para mapeo 3D en tiempo real y generación de escenas en forma de nube de puntos. La integración con ROS se logró a través del paquete zed-ros-wrapper. Se registró una nube de puntos en formato SVO y se procesó con la herramienta ZEDfu para crear un mapa denso que incluye obstáculos importantes como sillas y mesas.

El algoritmo *Real-Time Appearance-Based Mapping* (RTABMap) se basa en el reconocimiento de cierres para crear un mapa visual basado en grafos. La integración con ROS se realizó a través del paquete RTAB-Map ros pkg. RTABMap proporciona una interfaz gráfica que visualiza la detección de cierres, la odometría visual y la nube de puntos.

Se observó que la odometría visual basada en la cámara ZED tenía una discrepancia máxima de aproximadamente 10 cm con respecto a la trayectoria marcada. Además, se compararon otros

métodos de SLAM, como ORB-SLAM, DPPTAM y ZEDfu, que crean mapas tanto densos como dispersos, cada uno con sus ventajas y desventajas en la detección de objetos y obstáculos.

SLAM method	Sensors	Average deviation, m	Maximal deviation, m
Hector SLAM	2D LIDAR	0.11	0.18
ORB-SLAM	Monocular Camera	0.19	0.43
DPPTAM	Monocular Camera	2.05	4.26
ZEDfu	Stereo ZED camera	0.14	0.32
RTAB-Map	Kinect 2.0 depth sensor	0.42	0.67

Ilustración 19-Tabla de resultados para "Comparison o ROS-based Visual SLAM"

Tomada de: Comparison of ROS-based Visual SLAM methods in homogeneous indoor environment

Finalmente, se mencionó la utilidad de OctoMap para traducir una nube de puntos densa en un mapa de obstáculos cualitativo, lo que podría ser valioso para la navegación de robots en espacios cerrados. Estos resultados destacan la versatilidad de las soluciones de mapeo y odometría visual en entornos robóticos, especialmente cuando se integran con ROS.

Las recomendaciones que se dan en el artículo son:

1) Utilizar ORB-SLAM si se necesita alto rendimiento y el entorno no tiene objetos monocromáticos planos.

2) Usar DPPTAM si se requiere un mapa de área densa o si el entorno carece de suficientes características visuales y el hardware es lo suficientemente potente.

3) Las cámaras estéreo o los sensores RGB-D dan buenos resultados y proporcionan mapas y localización en valores absolutos.

4) Para mapas de gran profundidad, es preferible usar una cámara estéreo.

5) Si hay paredes monocromáticas grandes, vidrios tintados o espejos, un sensor RGB-D es una mejor opción. Los planes futuros incluyen la implementación de otros métodos de SLAM visual y odometría, con un enfoque en mejorar la calidad de la nube de puntos densa y realizar pruebas en entornos más complejos.

4.2.5 Embedded Mobile ROS Platform for SLAM Application with RGB-D Cameras [77]

El artículo utilizó una cámara RGB-D para resolver la creación de mapas y la navegación, con metodología SLAM. Este proyecto se enfoca en implementar un vehículo autónomo todo terreno para ampliar el conocimiento. Los vehículos todo terreno suelen ser más robustos y confiables que los vehículos de carretera, pero requieren un alto nivel de entrenamiento del conductor debido a la navegación en terrenos difíciles.

El sistema utiliza un algoritmo SLAM para crear un mapa del entorno, con el enfoque en la extracción y coincidencia de características. Se emplea ORB-SLAM2 para la navegación en 3D debido a su compatibilidad con ROS y su fiabilidad gracias a la coincidencia de características ORB. El sistema se divide en tres subsistemas principales: Computación y Cognición, Movilidad y Actuación, y Comunicación y Control, que trabajan juntos para la navegación autónoma.

Se presenta un prototipo de plataforma robótica construida sobre el marco de ROS. Utiliza el algoritmo ORB-SLAM2 y una cámara RGB-D RealSense 435Di con IMU para la cognición y la computación. La comunicación y el control se realizan mediante módulos Wi-Fi y Bluetooth. La computación se realiza en una Jetson Nano, que también aloja la aplicación de servidor para ROS. Se establece una conexión entre Jetson TX2 y ESP32 a través de Wi-Fi para controlar el vehículo. Se incorpora una cámara de profundidad RealSense D435i para la navegación autónoma en ROS, con capacidades de sincronización y calibración, alta velocidad de cuadro y un rango de detección de 1-15 metros.

Los algoritmos de SLAM en ROS utilizan Transform Frames (TF) para representar datos espaciales. Se genera un mapa base en el inicio y se proyectan los datos subsiguientes en él. La ubicación relativa se extrae a través de datos de IMU y puntos de referencia previos. Se realizaron experimentos con una cámara RealSense en ROS para la navegación autónoma. Hubo optimizaciones de rendimiento y soporte para múltiples máquinas en ROS. Se resolvió un problema de recursos insuficientes al conectar Jetson Nano a una computadora de trabajo a través de Ethernet. Se utilizó ORB-SLAM2 para el mapeo sin IMU, pero su inflexibilidad y falta de documentación dificultaron su implementación. En resumen, se logró desarrollar un robot todo terreno autónomo para procesamiento de imágenes y generación de trayectorias de navegación.

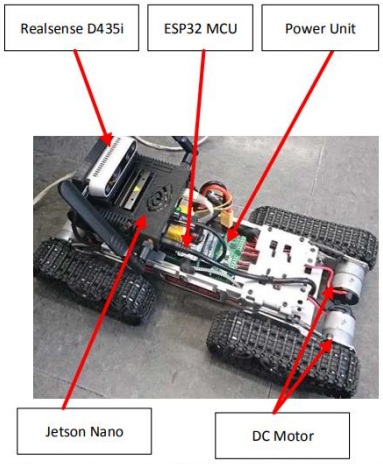


Figure 3. Assembled prototype platform

Ilustración 20-Robot implementado para "Embedded Mobile ROS Platform for SLAM"

Tomada de: Embedded Mobile ROS Platform for SLAM Application with RGB-D Cameras

4.3 TIPOS DE MARCADORES

Existen varios tipos de marcadores utilizados en SLAM visual, cada uno con sus propias ventajas y desventajas. Uno de los tipos más populares y ampliamente utilizados es el marcador ArUco (*Aumented Reality University of Cordoba*). Estos marcadores son patrones bidimensionales que se colocan en el entorno y se utilizan para que la cámara del robot los detecte y calcule su posición y orientación en relación con el marcador. Los marcadores ArUco son especialmente atractivos debido a su facilidad de detección y su capacidad para proporcionar información precisa de pose en tiempo real.

Los siguientes son algunos de los artículos de investigación que hablan de como implementaron diferentes tipos de marcadores:

4.3.1 A Multi-ring Color Fiducial System and A Rule-Based Detection Method for Scalable Fiducial-tracking Augmented Reality [78]

En realidad virtual, se exploran mundos generados por computadora sin necesidad de movimientos físicos. La realidad aumentada (RA), se alinea la posición del usuario en el mundo real con el virtual, siendo esencial para aplicaciones a gran escala. La detección de marcadores es clave en RA. Los sistemas actuales usan marcadores de un solo tamaño, limitando su rango. En este artículo se propone un concepto de sistema de múltiples anillos y tamaños de marcadores, aumentando la escalabilidad. Se desarrolla un método robusto de detección que funciona en diversas condiciones de iluminación. Esto puede ser de gran utilidad en aplicaciones a gran escala. Se utiliza un prototipo de vehículo terrestre no tripulado equipado con sensores y un sistema de cómputo para experimentos.

Se han desarrollado sistemas de RA con seguimiento de marcadores [79], pero la mayoría utiliza marcadores circulares sólidos o concéntricos [80], o esquinas de grandes rectángulos [81], limitando su rango a áreas pequeñas. Varios sistemas basados en visión enfrentan problemas con cambios en la iluminación. El sistema propuesto introduce un algoritmo de detección basado en reglas para superar problemas de iluminación en la detección de marcadores. Además, se introduce un sistema de marcadores de anillos múltiples y tamaños variados, que aumenta la escalabilidad del seguimiento.

Las dimensiones detectables de los marcadores se definen en función de su proximidad a la cámara y su tamaño. Niveles bajos contienen marcadores pequeños cercanos a la cámara, mientras que niveles altos contienen marcadores grandes más alejados. Los marcadores deben estar distribuidos sin patrones regulares y son esenciales para determinar la posición de la cámara [82] [83]. Se busca optimizar el rendimiento del sistema y el tamaño de los marcadores a través del parámetro c. El algoritmo de detección de marcadores busca minimizar el tiempo de procesamiento y el tamaño de búsqueda, influyendo en la eficiencia y el tamaño óptimo.

Se aplica un enfoque de múltiples resoluciones para detectar marcadores en imágenes, dividido en detección gruesa y fina. La detección gruesa encuentra posibles regiones, mientras que la fina detecta marcadores. Luego, se aplican pruebas de forma y color. Ambas etapas comparten asignación de valores de membresía y creación de segmentos de línea, diferenciándose en el número de líneas muestreadas y cómo se agrupan. Se basa en reglas y algoritmos difusos. Se asigna un valor de membresía a cada píxel de las líneas muestreadas y se crean segmentos de línea que se agrupan en regiones potenciales. La región con mayor prioridad se selecciona para la detección fina, generando posibles marcadores.

Se desarrolla una función de membresía utilizando lógica difusa para la detección de bordes en marcadores. A diferencia de las técnicas convencionales que requieren valores de umbral y conocimiento previo sobre la imagen, esta función de membresía busca la posición óptima de los bordes en las regiones de transición. Estas reglas se derivan de las características distintivas de los marcadores, como la distancia mínima entre dos marcadores, el tamaño mínimo y máximo de un marcador y su forma. Además, se parte del conocimiento de que los marcadores se sitúan sobre fondos sólidos, lo que implica que los bordes de interés se encuentran entre dos regiones homogéneas en la imagen.

La función de membresía consta de dos componentes, m1 y m2. m1 indica el grado de cercanía al valor de intensidad mediano entre dos regiones. m2 refleja el grado de proximidad al borde ideal. Así, la función de membresía m se emplea para encontrar una posición de borde cercana al borde ideal.

Se aplican estas reglas para recopilar píxeles de cada línea de exploración horizontal que las cumpla. Estos píxeles se agrupan en segmentos de línea conectados sin atravesar píxeles que no cumplan con las reglas. Luego, se selecciona un píxel de cada grupo con el valor de membresía más alto,

indicando la mejor ubicación para un borde. Estos píxeles seleccionados se conectan para formar segmentos de línea.

Cada marcador tiene un color sólido, lo que permite localizar su posición agrupando segmentos de línea del mismo color. Sin embargo, la presencia de ruido en las imágenes introduce un desafío de medida de similitud. Para superar esto, se propone una medida de similitud basada en teoría de probabilidad y utiliza información local en los segmentos de línea. Se crea una función de densidad de probabilidad uniforme para cada segmento de línea, y se considera que dos segmentos de línea tienen el mismo color cuando las funciones de densidad de probabilidad uniforme se superponen. Esto es posible debido a que la similitud de color se verifica cuando dos segmentos de línea están adyacentes y los marcadores y su fondo tienen colores diferentes.

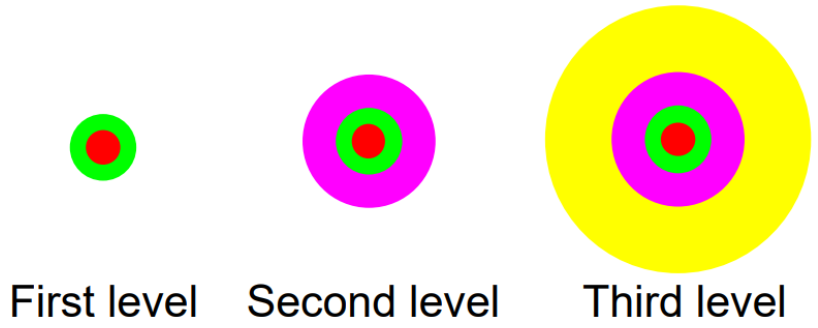


Ilustración 21-Tipo de marcador propuesto en "Multi-ring Color Fiducial System"

Tomada de: A Multi-ring Color Fiducial System and A Rule-Based Detection Method for Scalable Fiducial-tracking Augmented Reality

En resumen, este enfoque simplificado mejora la eficiencia del algoritmo de detección de bordes en marcadores, eliminando la necesidad de umbrales y proporcionando robustez contra variaciones de iluminación y ruido en las imágenes.

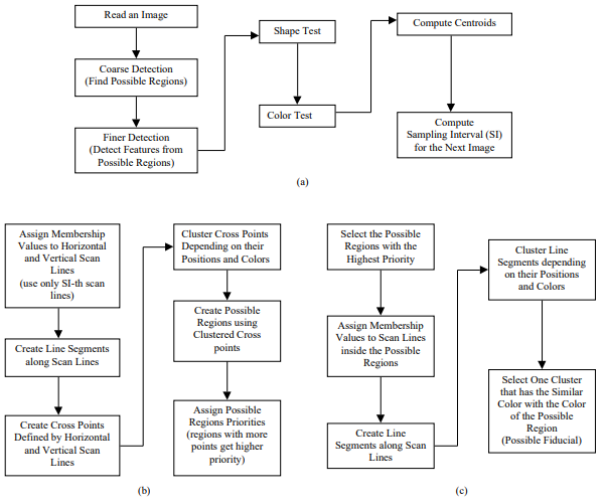


Ilustración 22-Flujo metodología para "Multi-ring Color Fiducial System"

Tomada de: A Multi-ring Color Fiducial System and A Rule-Based Detection Method for Scalable Fiducial-tracking Augmented Reality

La implementación se realiza en un sistema gráfico SGI Indy con una cámara de video SONY DXC-151A. Se emplea un conjunto de marcadores de tres niveles con seis colores y tamaños escalables.

El rango de detección total varía de 1.7' a 15.4'. Es importante destacar que el rendimiento del sistema depende del número y tamaño de los marcadores presentes en una imagen, y no utiliza predicciones de posición. El método de detección de marcadores se compara con otro método basado en gradientes, demostrando su robustez bajo diferentes condiciones de iluminación y apertura de cámara.

4.3.2 Analysis and Improvements in AprilTag Based State Estimation [84]

En entornos interiores, los marcadores son comunes para la localización debido a su facilidad de uso [85]. En entornos exteriores, el GPS se utiliza ampliamente, pero a menudo no proporciona la precisión requerida [86]. Este estudio se enfoca en abordar las limitaciones de AprilTag, identificando la orientación de la cámara como una fuente clave de error. Propone soluciones que incluyen correcciones geométricas y un modelo probabilístico de errores. Estas técnicas mejoran significativamente la precisión y la precisión en una variedad de situaciones.

Un sistema fiducial visual emplea información codificada en 2D en una etiqueta para determinar la posición y orientación de un marcador ante la cámara, siendo de gran utilidad en aplicaciones de realidad aumentada [87]. Se ha observado que AprilTag supera a sus predecesores en términos de tasa de detección, distancia Hamming entre códigos, escala y precisión angular [88]. No obstante, aún existen desafíos relacionados con su precisión y la distancia entre la cámara y la etiqueta. Pese a estas limitaciones, debido a su robustez y precisión. En investigaciones recientes, se ha demostrado que AprilTag es robusto ante pequeñas obstrucciones [89]. Una ventaja significativa de AprilTag es su bajo costo y su capacidad para servir como solución de localización en aplicaciones de realidad aumentada y robótica que solo requieren una cámara monocular y una etiqueta impresa en papel [90].

En aplicaciones de robótica, se ha empleado para el seguimiento de vehículos terrestres y aéreos no tripulados [91]. Además, se ha utilizado en la calibración de sistemas de múltiples cámaras y láser 2D, lo que permite una estimación mejorada de la posición y orientación. AprilTag también se ha utilizado para evaluar con precisión sistemas de localización y algoritmos de odometría visual [92].

Sin embargo, es importante destacar que la precisión de AprilTag puede verse afectada por varios factores, como el ángulo de visión, la distancia y la rotación de la cámara alrededor de su eje. Aunque se ha mejorado su precisión en entornos interiores mediante la fusión de datos de múltiples etiquetas y técnicas de fusión de datos como el filtro de Kalman, la aplicabilidad en entornos exteriores más amplios sigue siendo un desafío pendiente [93].

En el contexto de la detección de etiquetas para la Realidad Aumentada, AprilTag utiliza marcadores 2D con códigos para una detección rápida y robusta. Proporciona una única pose 6-DOF relativa al marco de referencia de la cámara. Los resultados muestran que la precisión y precisión de AprilTag están en el rango de centímetros. Se realiza una medición de errores para evaluar la posición de la cámara respecto a AprilTag. Se seleccionan 9 puntos nominales para abordar la incertidumbre en las mediciones debido a ángulos de visión y distancias variables.

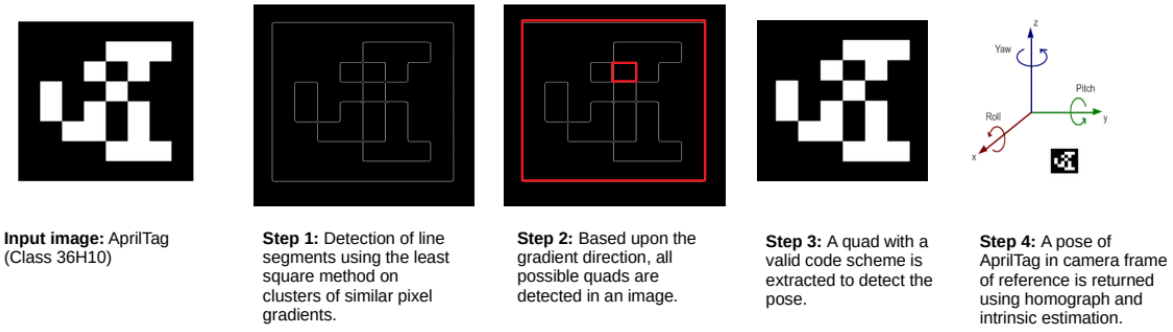


Ilustración 23-Metodología implementada para "Analysis and Improvements in AprilTag"

Tomada de: Analysis and Improvements in AprilTag Based State Estimation

En el contexto de AprilTag, la precisión se maximiza cuando la cámara apunta hacia el centro del marcador o cuando el eje z de la cámara se alinea con el centro del marcador. Se observa una mayor inexactitud cuando la cámara se desplaza hacia los lados. También se evidencia que la inexactitud aumenta a medida que la cámara se aleja del marcador en el eje z.

Cuando se fija la cámara de manera que su eje z no apunte al centro de AprilTag, se obtienen mediciones inexactas en términos de precisión, sin importar la ubicación de la cámara respecto al tag. Esto se demuestra empíricamente al variar el ángulo de giro de la cámara.

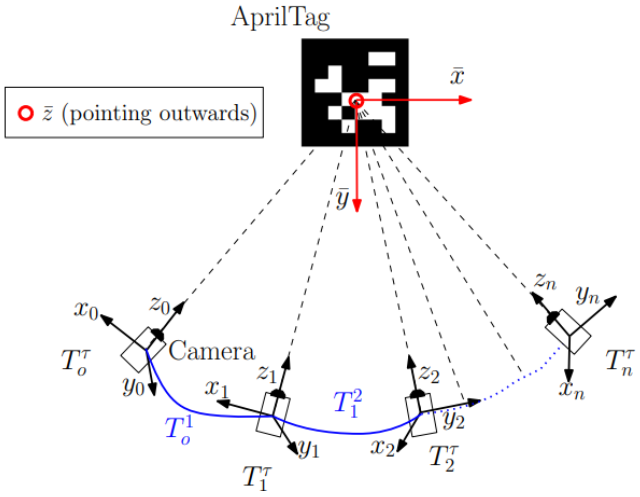


Ilustración 24-Ubicaciones de cámara con respecto a AprilTag

Tomada de: Analysis and Improvements in AprilTag Based State Estimation

El análisis de los datos revela que la inexactitud se incrementa significativamente con la distancia desde el tag en el eje z y a medida que la cámara se desplaza desde el frente hacia los lados. La inconsistencia en el marco de referencia debido al ángulo de giro de la cámara es una fuente importante de inexactitud, ya que AprilTag genera un nuevo marco de referencia para cada ángulo de giro.

Nominal Reference Points		Motion Capture (MoCap) Readings				
x_r(cm)	y_r(cm)	N	$\mu_{\bar{x}}$(cm)	$\mu_{\bar{y}}$(cm)	$\sigma^2_{\bar{x}}$(cm^2)	$\sigma^2_{\bar{y}}$(cm^2)
0	30	217	0.7062	30.3437	0.003110	0.003120
6	30	195	6.9367	29.5193	0.004890	0.000590
−6	30	193	−3.9230	30.1210	0.045800	0.003080
0	50	204	2.9902	50.0462	0.09500	0.014700
15	50	202	17.7118	49.6042	0.009490	0.002650
−15	50	205	−12.0469	50.8171	0.000168	0.000242
0	70	217	3.2683	70.0810	0.002380	0.000267
20	70	199	23.3681	69.4768	0.000210	0.000197
−20	70	212	−16.8097	70.9505	0.037100	0.001090

Ilustración 25-Tabla de resultados para "Analysis and Improvements in AprilTag"

Tomada de: Analysis and Improvements in AprilTag Based State Estimation

En resumen, se identifican limitaciones en la precisión de AprilTag en función de la distancia desde el tag, el ángulo de visualización de la cámara y el ángulo de giro de la cámara, lo que sugiere la necesidad de mejoras en la implementación actual.

Basado en el análisis de medición del sistema AprilTag, se proponen técnicas de mejora. Se introduce una corrección pasiva para la consistencia del marco. Esta técnica, denominada "Corrección de Eje de Giro Suave (SYAC)", ajusta la geometría del sistema para alinear el eje z de la cámara con el centro de la etiqueta. También se propone una corrección activa con un gimbal de eje de giro para alinear en tiempo real el eje z de la cámara. La combinación de ambas técnicas mejora significativamente la precisión de AprilTag. Se presenta un análisis comparativo de diferentes enfoques, mostrando una disminución sustancial en el error en comparación con AprilTag sin corrección. Las técnicas propuestas aumentan la precisión y la consistencia de AprilTag.

Ground-Truth (cm)		Error in Mean(μ) Using Different Approaches for AprilTag. (cm)									
Motion Capture System (MoCap)		Raw AprilTag Readings (Camera Pointing Towards Tag's Center)		Raw AprilTag Readings (Camera Pointing Away from Tag's Center)		Applying Soft Yaw Angle Correction (SYAC) on Raw AprilTag Readings		Applying Active Correction with Yaw Axis Gimbal on Raw AprilTag Readings		Applying (SYAC + Active Yaw Axis Gimbal Correction) on Raw AprilTag Readings	
$\bar{x}$	$\bar{y}$	$\lvert\bar{x}-\mu_{\bar{x}}\rvert$	$\lvert\bar{y}-\mu_{\bar{y}}\rvert$	$\lvert\bar{x}-\mu_{\bar{x}}\rvert$	$\lvert\bar{y}-\mu_{\bar{y}}\rvert$	$\lvert\bar{x}-\mu_{\bar{x}}\rvert$	$\lvert\bar{y}-\mu_{\bar{y}}\rvert$	$\lvert\bar{x}-\mu_{\bar{x}}\rvert$	$\lvert\bar{y}-\mu_{\bar{y}}\rvert$	$\lvert\bar{x}-\mu_{\bar{x}}\rvert$	$\lvert\bar{y}-\mu_{\bar{y}}\rvert$
0.642	31.00	0.884	0.855	1.758	0.855	1.223	0.389	3.3989	0.693	2.135	0.330
6.71	29.82	0.3061	0.037	1.250	0.405	0.286	0.075	0.8652	0.251	0.049	0.045
−5.89	29.85	0.0898	0.578	0.519	0.172	0.392	0.690	0.7491	0.030	0.325	0.615
2.017	50.02	1.2599	0.0618	3.367	0.868	1.824	1.298	4.958	0.139	4.143	0.141
16.90	51.13	0.0141	1.709	3.011	2.244	0.415	1.534	0.934	1.455	0.045	1.918
−14.42	49.63	2.4985	0.228	1.150	1.325	1.853	0.567	1.192	0.145	0.429	0.545
−2.10	68.90	3.408	1.126	2.071	0.511	3.955	2.940	3.401	1.107	2.283	1.386
22.70	71.16	0.3426	2.088	0.721	3.738	2.182	0.573	1.685	1.317	0.765	1.789
−21.10	71.23	0.6559	0.432	3.343	2.596	5.352	0.873	0.685	0.285	0.259	0.518

Ilustración 26-Tabla de resultados comparados con los diferentes modelos en "Analysis and Improvements in AprilTag"

Tomada de: Analysis and Improvements in AprilTag Based State Estimation

Se logran mejoras significativas en la precisión y precisión de AprilTag, a costa de un ligero aumento en el tiempo de ejecución.

La precisión del sistema se demuestra en una trayectoria cercana a la verdad en experimentos al aire libre.

4.3.3 Comparing ARTag and ARToolkit Plus Fiducial Marker Systems [94]

Este artículo compara dos sistemas de marcadores, ARTag y ARToolkit Plus, adecuados para sistemas de visión robustos. Ambos utilizan marcadores cuadrados en una superficie plana, con dos etapas de detección: detección de características únicas y verificación/identificación.

ARTag mejora ARToolkit al usar un enfoque basado en bordes y codificación digital para reducir falsos positivos. Utiliza técnicas de corrección de errores para una detección más confiable. ARToolkit Plus, inspirado en ARTag, se centra en la etapa de verificación/identificación y comparte similitudes en la apariencia de los marcadores con ARTag.

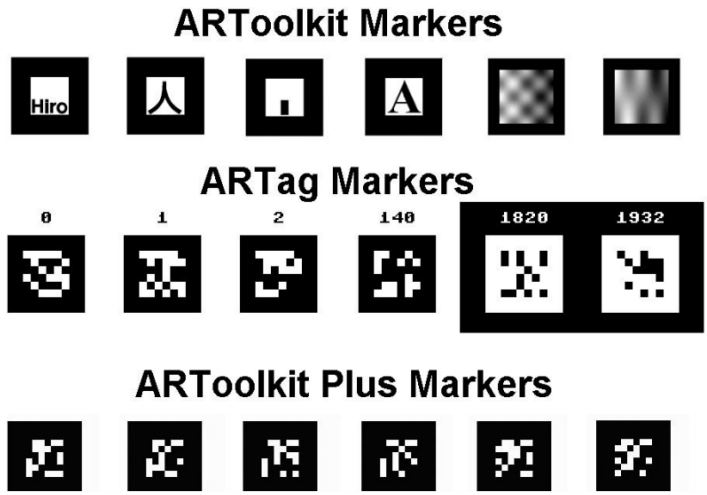


Ilustración 27-Diferentes librerías a comparar en "Comparing ARTag and ARToolkit "

Tomada de: Comparing ARTag and ARToolkit Plus Fiducial Marker Systems

Este artículo se enfoca en comparar ARToolkit Plus y ARTag, demostrando que ARTag es más robusto como sistema de marcador fiducial.

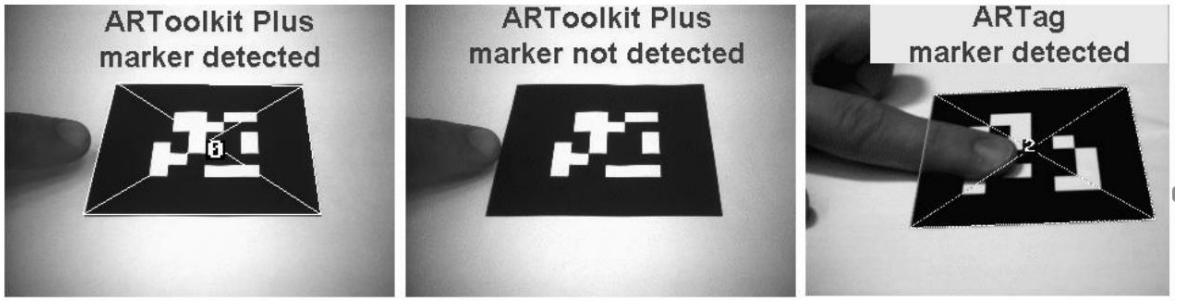


Ilustración 28-Robustéz de ARTag

Tomada de: Comparing ARTag and ARToolkit Plus Fiducial Marker Systems

Los sistemas de marcadores consisten en patrones impresos y algoritmos de visión por computadora para identificarlos en una imagen. Estos sistemas suelen emplear un proceso de dos pasos: primero, localizar una característica única y luego verificar/identificar. Muchos usan un límite cuadrilateral como característica única.

En la etapa de la detección de características únicas, puede verse afectada la tasa de falsos negativos en ARTag y ARToolkit Plus. Ambos sistemas requieren bordes cuadriláteros completos y son sensibles a la variación de iluminación. La detección de cuadriláteros en ARToolkit Plus también falla con variaciones de iluminación. ARTag muestra ventajas en la detección en condiciones de

iluminación variable. Se pueden realizar múltiples llamadas a la función de detección de ARToolkit Plus y combinar los resultados, pero no resuelve los problemas de variación de irradiación en un solo marcador.

En resumen, la detección de características únicas y la variación de iluminación son desafíos clave en los sistemas de marcadores como ARTag y ARToolkit Plus.

ARTag demuestra tener histogramas de Hamming más favorables en comparación con ARToolkit Plus, lo que se traduce en una reducción sustancial de la confusión inter-marcadores. Esta diferencia se origina en cómo codifican sus contenidos digitales. ARToolkit Plus, por ejemplo, simplemente repite el número de identificación (ID) cuatro veces para crear un código de 36 bits y luego aplica una operación XOR con una máscara fija. En contraste, ARTag utiliza la convolución digital del ID con códigos de verificación y corrección de errores específicamente seleccionados para optimizar el histograma de distancia de Hamming. Esto se traduce en una menor probabilidad de confusión entre marcadores, especialmente cuando no se rotan.

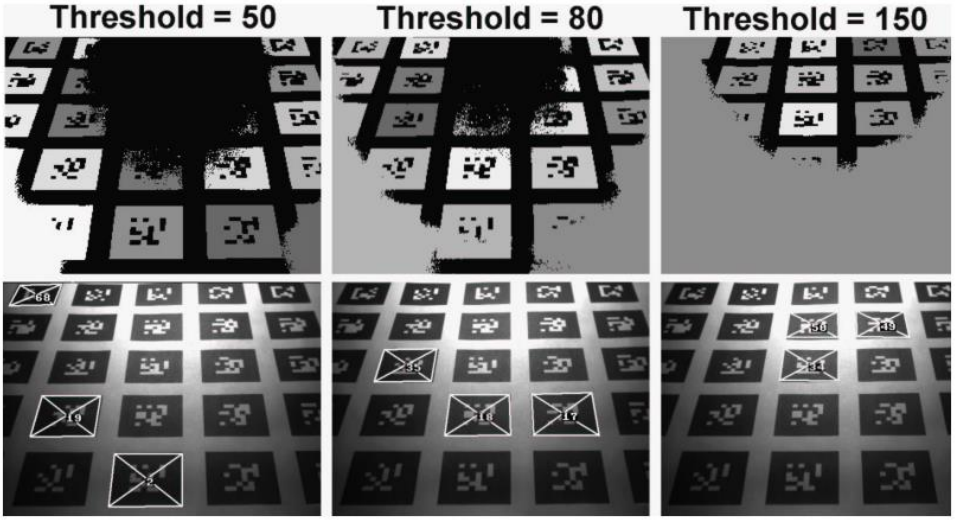


Ilustración 29-Detecciones para diferentes Threshold "Comparing ARTag and ARToolkit "

Tomada de: Comparing ARTag and ARToolkit Plus Fiducial Marker Systems

Además, se menciona que ARTag posee una ventaja en la detección de características únicas y presenta una tasa de falsos negativos más baja debido a su esquema de corrección de errores. Esto significa que ARTag puede funcionar de manera más efectiva en condiciones no ideales y en imágenes ruidosas.

En resumen, se destaca que ARTag presenta un diseño más efectivo para reducir la confusión inter-marcadores y mejorar la detección en condiciones adversas, al aprovechar al máximo su esquema de corrección de errores y su detección basada en bordes para identificar características únicas en comparación con el método de escala de grises empleado por ARToolkit Plus.

4.3.4 Experimental Comparison of Fiducial Markers for Pose Estimation [95]

Este estudio compara cuatro paquetes de marcadores: ARTag, AprilTag, ArUco y STag, ampliamente utilizados en aplicaciones UAS y de código abierto. Se evalúan tasas de detección y precisión de localización desde diversas distancias y orientaciones, junto con el rendimiento computacional en tres sistemas ampliamente utilizados en plataformas autónomas. Además, se presenta una implementación ROS para el marcador STag.

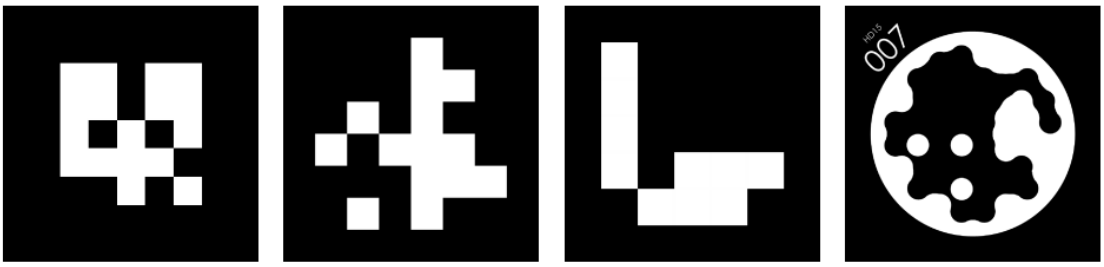


Fig. 1: The markers used in this work: (a) ARTag [13], [14], (b) AprilTag [15], [16], (c) ArUco [17], and (d) STag [10].

Ilustración 30-Diferentes marcadores usados en "Experimental Comparison of Fiducial Markers"

Tomada de: Experimental Comparison of Fiducial Markers for Pose Estimation

Se han mencionado cuatro marcadores utilizados en aplicaciones de UAS y robótica, cada uno con sus características específicas. El ARTag se basa en ARToolkit pero utiliza la teoría de codificación digital para crear patrones internos únicos, lo que mejora la fiabilidad de detección. AprilTag, basado en ARTag, ofrece mejoras adicionales, como un algoritmo de segmentación de imagen basado en grafos y un nuevo sistema de codificación para una mayor robustez contra la oclusión y la deformación. ArUco permite a los usuarios crear bibliotecas personalizadas de marcadores para una mayor eficiencia y menor tiempo de procesamiento, lo que lo hace adecuado para aplicaciones específicas. STag se centra en la estabilidad de las mediciones de estimación de posición y utiliza patrones circulares en el centro de los marcadores, lo que mejora la precisión y la consistencia de las mediciones. Estos marcadores se han utilizado en diversas aplicaciones, como la localización de robots, la identificación y seguimiento de objetos, y el aterrizaje autónomo de UAS en plataformas en movimiento, demostrando su versatilidad en la investigación y desarrollo de UAS y robótica.

Se realizaron pruebas de evaluación en cuatro paquetes de software en tres computadoras diferentes utilizadas comúnmente en aplicaciones de Sistemas Aéreos No Tripulados (UAS). Las computadoras incluyeron una Raspberry Pi 3B+, una NVidia Jetson TX2 y un Intel NUC (NUC5i7RYH). Además, se utilizaron dos cámaras: una Logitech C270 Webcam y un módulo de cámara Raspberry Pi versión 2 (piCam). Las pruebas abarcaron diversas distancias, ángulos y condiciones de iluminación para evaluar la precisión y la tasa de detección de cada paquete.

Ilustración 31-Posiciones de los marcadores en "Experimental Comparison of Fiducial Markers"

Tomada de: Experimental Comparison of Fiducial Markers for Pose Estimation

Los resultados revelaron diferencias significativas en la detección y la precisión según la distancia y el tipo de cámara utilizada. AprilTag sobresalió en términos de tasa de detección, mientras que STag demostró una mayor precisión en general. Sin embargo, se observó una disminución en la precisión a medida que aumentaba la distancia en todos los paquetes evaluados.

Camera	Angle	Light	ARTag			AprilTag			ArUco			STag		
			mean	std	rate	mean	std	rate	mean	std	rate	mean	std	rate
Logitech	0°	normal	2.831	0.158	45.833	2.450	0.068	99.539	2.386	0.176	99.552	3.317	0.347	92.825
		shadow	3.132	0.102	45.045	3.417	0.039	100.000	2.012	0.053	99.545	3.380	0.092	94.444
	10°	normal	0.625	0.047	46.083	0.884	0.168	99.552	1.773	0.115	99.548	1.118	0.135	91.855
		shadow	2.039	0.081	45.455	1.386	0.136	99.541	1.401	0.131	99.539	1.421	0.329	93.953
	20°	normal	2.326	0.055	46.296	1.517	0.080	99.545	2.219	0.178	100.000	2.319	0.115	92.793
		shadow	3.129	0.089	45.045	2.299	0.068	99.545	2.744	0.074	99.543	2.971	0.089	93.088
	30°	normal	1.162	0.068	46.330	1.325	0.066	99.541	1.379	0.122	100.00	1.565	0.077	91.818
		shadow	1.527	0.125	45.662	1.310	0.057	99.537	1.455	0.105	99.539	1.728	0.302	94.907
	40°	normal	0.214	0.050	46.083	0.641	0.024	99.091	0.203	0.025	99.550	0.590	0.070	92.991
		shadow	0.564	0.063	45.662	0.456	0.031	99.087	0.910	0.050	99.548	0.605	0.186	96.789
	50°	normal	0.141	0.022	44.595	0.365	0.015	99.543	0.459	0.046	99.087	0.244	0.033	94.064
		shadow	0.156	0.021	45.662	0.022	0.024	99.528	0.162	0.046	99.537	0.047	0.204	95.909
	60°	normal	0.346	0.030	44.395	0.126	0.009	99.543	0.239	0.011	100.00	0.422	0.019	97.727
		shadow	0.473	0.020	45.455	0.264	0.016	100.00	0.188	0.018	100.00	0.194	0.558	95.045
	70°	normal	0.738	0.039	45.662	0.476	0.044	99.083	0.779	0.015	99.545	0.699	0.064	99.541
		shadow	0.716	0.009	45.872	0.399	0.012	99.103	0.607	0.013	99.087	0.097	0.282	96.330
	80°	normal	1.270	0.039	41.176	1.052	0.089	100.00	1.238	0.046	99.087	1.258	0.042	98.190
		shadow	1.143	0.042	45.045	0.874	0.009	99.552	0.976	0.010	100.00	0.602	0.235	97.273
Pi camera	0°	normal	1.883	0.110	50.249	0.649	0.267	99.010	0.602	0.604	99.010	0.627	0.363	99.502
		shadow	0.010	0.147	49.751	0.288	0.611	99.502	4.863	0.435	99.502	0.411	0.419	99.502
	10°	normal	1.532	0.200	49.751	0.476	0.092	97.887	1.024	0.178	99.502	0.571	1.902	99.502
		shadow	1.521	0.100	49.451	0.323	0.222	99.502	4.201	0.041	99.502	1.801	0.874	99.502
	20°	normal	0.062	0.052	49.254	0.028	0.054	99.502	0.138	0.080	99.502	0.088	0.168	99.502
		shadow	0.487	0.063	49.751	0.561	0.152	99.502	0.278	0.111	99.502	0.600	0.184	98.701
	30°	normal	0.256	0.039	49.367	0.035	0.055	99.502	0.765	0.141	99.502	0.479	0.158	100.000
		shadow	0.294	0.036	49.751	0.288	0.036	99.502	0.255	0.067	99.502	0.134	0.097	99.502
	40°	normal	0.092	0.038	49.254	0.034	0.016	98.658	0.296	0.031	99.502	0.457	0.387	100.000
		shadow	0.148	0.034	49.751	0.026	0.075	99.005	0.502	0.016	99.502	0.497	0.025	99.502
	50°	normal	0.440	0.059	48.795	0.519	0.025	99.502	0.134	0.018	99.502	0.080	0.032	99.502
		shadow	0.210	0.018	49.751	0.025	0.020	99.502	0.207	0.016	99.502	0.434	0.020	100.000
	60°	normal	0.163	0.015	49.254	0.022	0.016	97.887	0.528	0.033	99.502	0.026	0.048	99.502
		shadow	0.477	0.015	49.412	0.066	0.019	99.502	0.193	0.010	99.005	0.172	0.034	97.279
	70°	normal	0.131	0.015	49.751	0.137	0.011	97.902	0.299	0.007	100.000	0.322	0.048	98.026
		shadow	0.071	0.031	49.751	0.175	0.031	99.502	0.342	0.009	99.005	0.184	0.152	98.990
	80°	normal	0.335	0.019	49.020	0.235	0.012	99.502	0.518	0.004	99.502	0.277	0.169	36.634
		shadow	0.169	0.097	49.505	0.010	0.025	100.00	0.153	0.016	99.502	0.257	0.073	94.527

Ilustración 32-Resultados para ángulos en "Experimental Comparison of Fiducial Markers"

Tomada de: Experimental Comparison of Fiducial Markers for Pose Estimation

Los resultados resaltan que AprilTag mostró un rendimiento general sólido, logrando altas tasas de detección en la mayoría de las configuraciones. Sin embargo, STag se destacó como el más preciso en las mediciones de posición, mientras que ArUco también presentó resultados consistentes y competitivos.

Camera	Distance (cm)	Light	ARTag			AprilTag			ArUco			STag		
			mean	std	rate	mean	std	rate	mean	std	rate	mean	std	rate
Logitech	75	normal	2.262	0.046	42.899	3.850	0.017	96.914	2.067	0.021	97.072	1.863	0.024	93.577
		shadow	2.204	0.024	43.130	3.900	0.021	96.997	2.034	0.027	95.976	1.859	0.017	93.090
	100	normal	2.183	0.017	43.699	4.721	0.021	96.402	2.204	0.028	95.926	1.671	0.067	89.325
		shadow	2.224	0.051	44.260	4.723	0.021	96.163	2.184	0.028	96.030	1.716	0.007	91.958
	150	normal	2.484	0.442	43.375	5.981	0.073	96.644	2.242	0.112	96.165	1.450	0.249	95.562
		shadow	1.909	0.212	43.634	6.052	0.106	97.068	2.443	0.315	96.071	1.425	0.108	95.007
	175	normal	2.668	0.344	43.546	6.559	0.146	96.074	2.315	0.229	96.103	1.012	0.107	96.112
		shadow	ND	ND	0.0	6.639	0.121	96.554	2.620	0.197	95.994	1.131	0.059	96.036
	200	normal	3.155	0.220	43.611	7.236	0.191	96.012	3.050	0.315	96.083	0.648	0.257	96.511
		shadow	ND	ND	0.0	7.273	0.251	96.473	3.615	0.200	96.473	0.680	0.192	96.054
Pi camera	75	normal	1.431	0.007	47.544	2.142	0.012	94.704	1.284	0.011	94.693	0.737	0.077	90.606
		shadow	1.549	0.015	47.118	2.306	0.031	94.444	1.353	0.020	94.829	0.764	0.057	94.636
	100	normal	1.954	0.084	49.766	2.541	0.031	94.429	1.562	0.026	94.595	0.842	0.023	98.448
		shadow	2.214	0.053	47.349	2.785	0.042	95.333	1.813	0.100	93.417	0.993	0.024	95.333
	150	normal	2.617	0.280	47.313	3.292	0.232	95.333	3.632	0.047	94.051	0.475	0.757	75.805
		shadow	3.184	0.226	47.447	3.596	0.082	94.865	3.748	0.078	94.123	0.495	0.588	94.624
	175	normal	1.768	0.192	47.292	3.410	0.340	94.787	3.827	0.612	94.917	0.093	1.041	93.279
		shadow	3.824	0.071	46.891	4.081	0.235	94.366	4.838	0.696	93.001	0.398	0.540	93.361
	200	normal	2.697	0.289	47.627	3.510	0.212	94.429	3.780	1.361	95.067	1.542	0.127	95.410
		shadow	4.403	0.441	47.024	4.877	0.290	94.648	6.151	0.875	95.087	1.194	0.240	91.589

Ilustración 33-Resultados para distancias en "Experimental Comparison of Fiducial Markers"

Tomada de: Experimental Comparison of Fiducial Markers for Pose Estimation

Un hallazgo interesante fue que las imágenes de baja resolución con alto contraste tendían a producir errores más bajos en las mediciones. Aunque las imágenes de alta resolución mejoraron la tasa de detección en la mayoría de los casos, no necesariamente llevaron a mediciones más precisas.

En términos de eficiencia computacional, AprilTag se destacó como el paquete más exigente en recursos, requiriendo una cantidad significativa de CPU y memoria. STag tuvo el segundo mayor uso de CPU, pero un uso de memoria bajo. ARTag y ArUco resultaron ser las opciones más eficientes desde el punto de vista computacional.

Es importante destacar que la implementación de ROS utilizada en el estudio difiere del algoritmo original propuesto para encontrar la pose de la cámara a partir de las esquinas detectadas del marcador. Esto puede haber afectado la estabilidad general de los resultados.

En resumen, la elección del paquete de marcadores depende de las necesidades específicas de la aplicación y la plataforma de hardware utilizada. ArUco se presenta como la elección más adecuada para plataformas de bajo consumo de energía como la Raspberry Pi, mientras que, en computadoras más potentes, todos los paquetes, excepto ARTag, son opciones viables.

4.3.5 ChromaTag: A Colored Marker and Fast Detection Algorithm [96]

Este artículo introduce ChromaTag, un marcador fiduciario de colores y su respectivo algoritmo de detección. ChromaTag se destaca al lograr una detección rápida y precisa gracias a su diseño y algoritmo de detección innovadores.

Ilustración 34-Diseño de marcador "ChromaTag"

Tomada de: ChromaTag: A Colored Marker and Fast Detection Algorithm

En contraste con los marcadores tradicionales que suelen emplear bordes de alto contraste (blanco y negro) para la detección inicial, ChromaTag aprovecha el espacio de color LAB. El canal A detecta cambios significativos entre rojo y verde, lo que es poco común en escenas naturales, reduciendo así las detecciones falsas iniciales. Para la localización precisa, utiliza el canal L de alta resolución. El canal B se emplea para codificar la identificación del marcador.

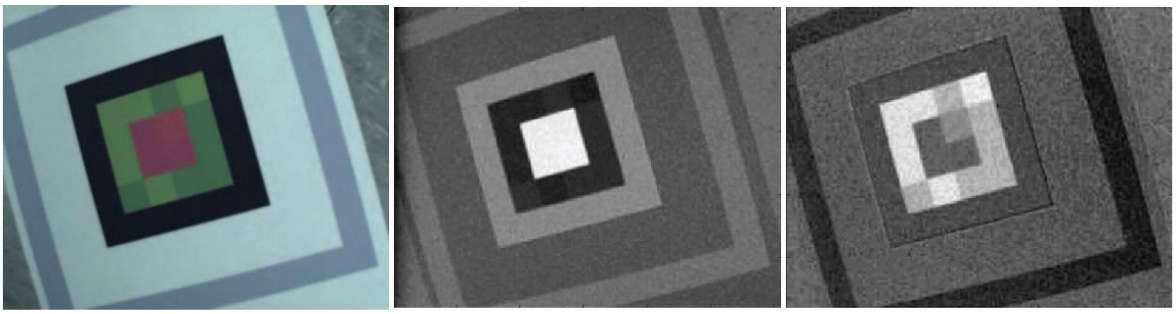


Ilustración 35-Visiones según canal en "ChromaTag"

Tomada de: ChromaTag: A Colored Marker and Fast Detection Algorithm

El algoritmo de detección de ChromaTag detecta inicialmente marcadores, construye polígonos en sus bordes, los simplifica a cuadriláteros y decodifica la identificación. La robustez ante variaciones de iluminación se logra mediante el uso de diferencias de cromaticidad y luminancia en todo el proceso de detección y localización, lo que resulta en una velocidad de más de 700 cuadros por segundo. El proceso de detección implica encontrar posibles ubicaciones de etiquetas, escanearlas para identificar los bordes y construir un polígono inicial que representa la etiqueta. Luego, este polígono se expande para abarcar completamente los bordes de la etiqueta, y se ajusta a un cuadrilátero que define la forma final de la etiqueta. Finalmente, se descifra la información de la etiqueta y se verifica su correspondencia en una base de datos precomputada.

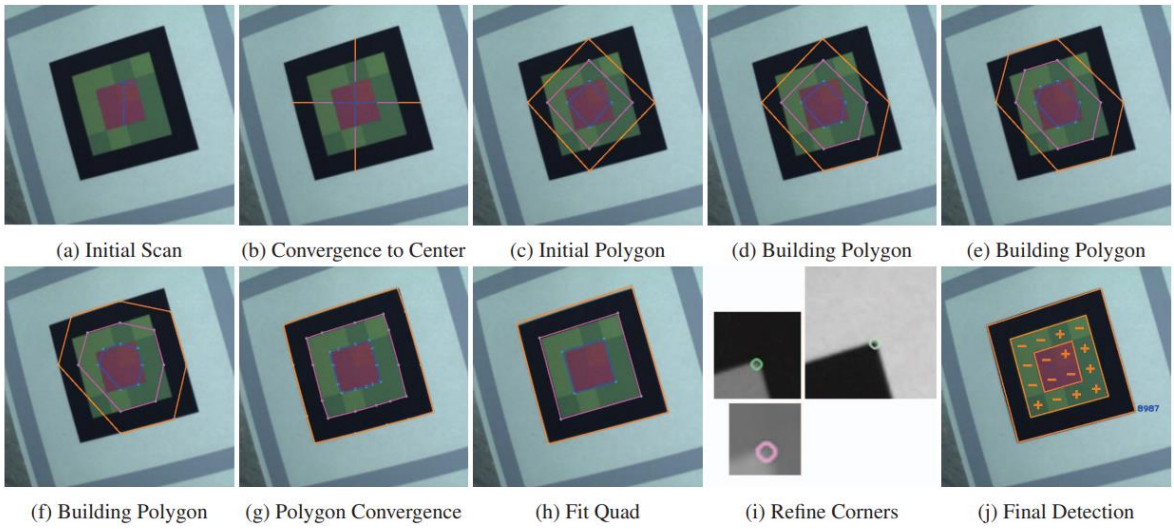


Ilustración 36-Metodología para detectar "ChromaTag"

Tomada de: ChromaTag: A Colored Marker and Fast Detection Algorithm

El artículo compara ChromaTag con una serie de marcadores fiduciarios existentes CCtag, RuneTag y AprilTag y demuestra que logra velocidades de detección significativamente más rápidas sin sacrificar la precisión. Las imágenes en color se capturaron a 30 fps con una resolución de 752 x 480 utilizando una cámara Matrix Vision mvBluefox-200wc. Cabe destacar que la cámara estaba montada en un servo motor que rotaba continuamente en el plano entre 0 y 180 grados, lo que permitió capturar datos desde diferentes ángulos de vista.

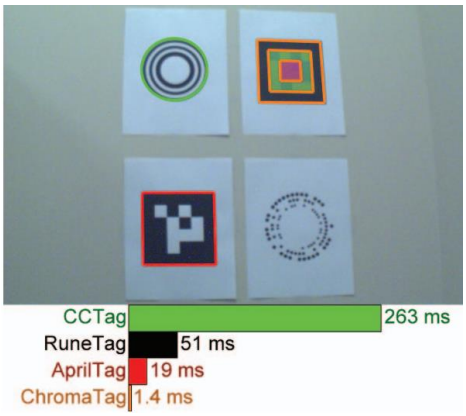


Ilustración 37-Velocidad de detección para diferentes marcadores en "ChromaTag"

Tomada de: ChromaTag: A Colored Marker and Fast Detection Algorithm

Este artículo subraya la importancia de contar con marcadores fiduciarios eficientes en aplicaciones en tiempo real, lo que lo convierte en una contribución valiosa para la investigación en este campo.

El artículo introduce ChromaTag, un sistema que emplea una estrategia de configuración de colores basada en el espacio de color LAB para lograr una detección precisa de etiquetas. Este enfoque aprovecha la alta gradiente entre colores opuestos en cada canal, como se muestra en la Ilustración 38, donde el canal A representa colores opuestos de rojo y verde, y el canal B refleja colores opuestos de azul y amarillo. El diseño de ChromaTag se basa en estas propiedades, con un centro rojo rodeado por un anillo verde que presenta una gran diferencia en el canal A, lo que lo hace altamente distinguible en la mayoría de las escenas naturales.

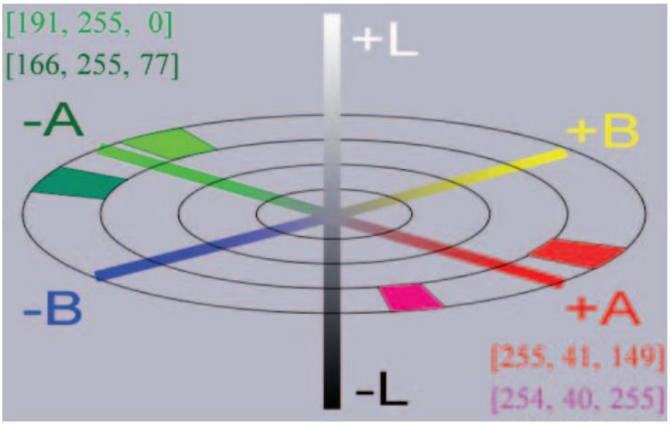


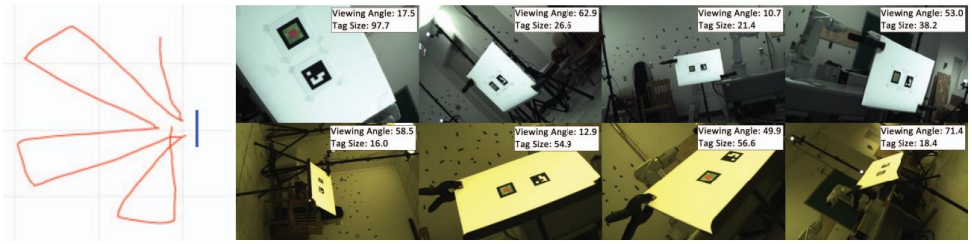
Ilustración 38-Canal LAB de colores para "ChromaTag"

Tomada de: ChromaTag: A Colored Marker and Fast Detection Algorithm

Además, ChromaTag codifica su información binaria en el canal B, que tiene un impacto limitado en la intensidad del canal A. Esto permite adaptar umbrales por etiqueta para abordar variaciones debidas a la iluminación o la impresión. Se ha demostrado que el espacio de color LAB es más resistente a las variaciones de impresión y luz en comparación con otros espacios de color como YUV.

La localización precisa es esencial para decodificar la etiqueta y recuperar la pose de la cámara. ChromaTag incorpora rectángulos concéntricos en blanco y negro en su diseño para proporcionar bordes de alto contraste y alta resolución, lo que facilita la localización precisa de las esquinas.

En cada conjunto de datos, ChromaTag se colocó junto a uno de los marcadores de comparación en una superficie plana, y se registraron los datos mientras la cámara exploraba la escena. La posición de la cámara durante la recolección de datos se capturó mediante un sistema de captura de movimiento. Este proceso se repitió dos veces, considerando dos condiciones de iluminación: balance de blancos (WB) y sin balance de blancos (NWB). La Ilustración 39 proporciona una representación visual de la trayectoria de captura y algunas imágenes de muestra.



(a) Data Collection Path (b) Example Data Under Two Lighting Conditions

Ilustración 39-Diferentes ubicaciones para detectar "ChromaTag"

Tomada de: ChromaTag: A Colored Marker and Fast Detection Algorithm

Uno de los logros más destacados es que ChromaTag supera significativamente a sus competidores en términos de velocidad de detección. La Tabla muestra los tiempos de cálculo para cada marcador fiducial, y ChromaTag logra un promedio de 926.4 FPS, superando a AprilTag, CCTag y RuneTag por un margen significativo.

	Average Frames Per Second		
	Total	>0 Detections	0 Detections
ChromaTag	926.4	709.2	2616.1
AprilTag	56.1	56.3	49.0
CCTag	10.0	6.5	18.5
RuneTag	41.9	2.4	71.3

Ilustración 40-Tabla de marcadores detectados por tiempo "ChromaTag"

Tomada de: ChromaTag: A Colored Marker and Fast Detection Algorithm

Además de su velocidad, ChromaTag también destaca por su alta precisión. En comparación con AprilTag, ChromaTag alcanza una precisión del 96%, mientras que AprilTag solo logra un 44%. Este aumento en la precisión es especialmente notable dado que ambos utilizan la familia 16H5. ChromaTag logra rechazar muchos falsos positivos iniciales que AprilTag identifica como tags.

		Chroma	April	Chroma	CCTag
Frames	WB	10266		11238	
	NWB	10303		10891	
Precision	WB	96.9	46.0	96.3	100.0
	NWB	95.7	42.9	95.7	99.9
Recall	WB	64.0	96.4	64.5	45.7
	NWB	67.9	98.2	66.1	46.3

Ilustración 41-Resultados comparación en "ChromaTag"

Tomada de: ChromaTag: A Colored Marker and Fast Detection Algorithm

Otra ventaja de ChromaTag es su robustez ante la variación de color. Los resultados en la Tabla muestran que ChromaTag mantiene un alto nivel de precisión y recuperación tanto en conjuntos de datos con balance de blancos como sin él, a pesar de las diferencias significativas en color. Esto se

debe a la forma en que ChromaTag utiliza las diferencias de píxeles LAB para la detección, lo que garantiza una detección sólida incluso en condiciones de iluminación y color variables.

En resumen, ChromaTag ofrece una detección rápida y precisa, lo que lo convierte en una opción sólida para una amplia gama de aplicaciones, especialmente cuando se requiere una detección rápida y precisa en condiciones de variación de color.

4.4 USO DE MARCADORES ARUCO EN SLAM

El uso de marcadores ArUco en aplicaciones de SLAM implica la detección y el seguimiento de estos marcadores por parte de la cámara del robot. Cuando se detecta un marcador ArUco en una imagen, se puede calcular su pose con respecto a la cámara. Al combinar esta información de pose con la información de otros marcadores y sensores, como odometría, se puede realizar una estimación precisa de la ubicación y orientación del robot en tiempo real.

4.4.1 Mobile robot localization in industrial environments using a ring of cameras and ArUco markers [97]

Este artículo presenta un método de localización para robots tipo plataforma en la inspección y reparación de láminas de acero. Se enfoca en entornos interiores sin suficientes puntos de referencia, abordando la localización mediante técnicas de SLAM y marcadores ArUco [98]. Se utiliza un anillo de 8 cámaras en el robot y marcadores artificiales en el entorno.

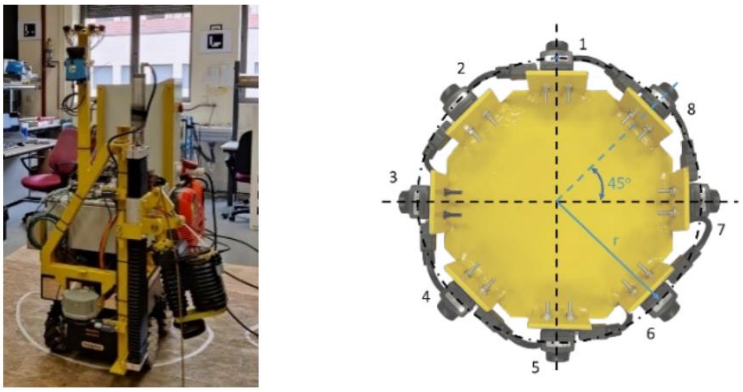


Ilustración 42-Modelo de robot para "Mobile robot localization in industrial environments"

Tomada de: Mobile robot localization in industrial environments using a ring of cameras and ArUco markers

El sistema de marcadores ArUco [99] se compone de marcadores y un algoritmo para su detección e identificación. Cada marcador ArUco es un cuadro con un borde negro y una matriz binaria interna que define un identificador. La detección se realiza con el paquete ArUco_detect [100]. Para localizar al robot, se mapean los marcadores en un marco de referencia común y marco de referencia global. El algoritmo de mapeo se divide en pasos: detectar marcadores, seleccionar uno como sistema de referencia, estimar poses y aplicar transformaciones. Para localizar al robot, se utilizan ocho poses estimadas y se fusionan. Se emplea un enfoque de selección de estimaciones con el menor error entre ocho cámaras en cada instante. Las estimaciones cercanas y con distancias Euclidianas pequeñas se promedian de forma ponderada. Se aplica un filtro Kalman lineal para optimizar la pose.

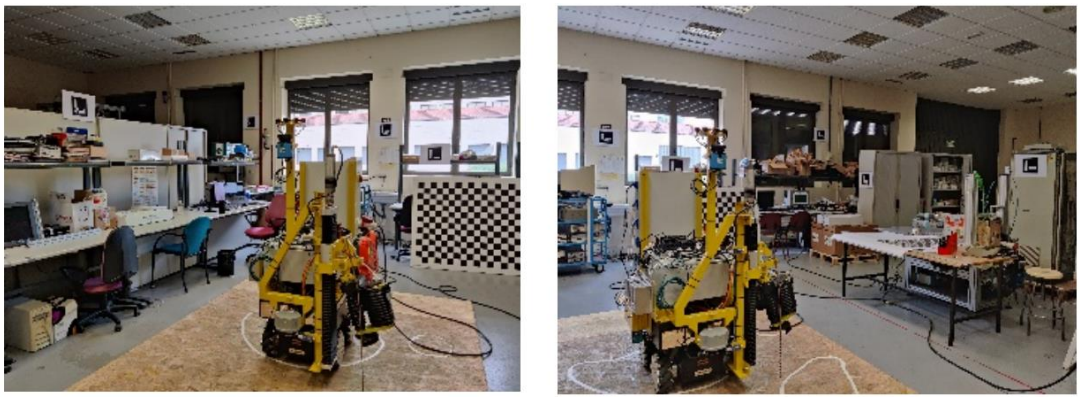


Ilustración 43-Entorno de trabajo para "Mobile robot localization in industrial environments"

Tomada de: Mobile robot localization in industrial environments using a ring of cameras and ArUco markers

Las mediciones consisten en posición (x, y) y orientación (ϕ). Tras la predicción, se realiza la corrección con las observaciones, mejorando la estimación. El robot se probó en un entorno con marcadores ArUco y se evaluó su localización en comparación con el paquete gmapping. Los errores absolutos fueron pequeños, con una mediana de 3.10 cm en la posición y un error promedio de 1.901º en la orientación.

1			2			3		
X error (cm)	Y error (cm)	Euclidean error(cm)	X error (cm)	Y error (cm)	Euclidean error(cm)	X error (cm)	Y error (cm)	Euclidean error(cm)
2.015	4.969	5.363	1.255	1.480	1.941	0.499	3.132	3.172
4			5			6		
X error (cm)	Y error (cm)	Euclidean error(cm)	X error (cm)	Y error (cm)	Euclidean error(cm)	X error (cm)	Y error (cm)	Euclidean error(cm)
2.521	1.812	3.104	0.302	0.221	0.374	2.870	0.609	2.932
7			8			9		
X error (cm)	Y error (cm)	Euclidean error(cm)	X error (cm)	Y error (cm)	Euclidean error(cm)	X error (cm)	Y error (cm)	Euclidean error(cm)
7.425	1.556	7.587	0.257	0.652	0.701	2.01	2.061	2.881
10			11			12		
X error (cm)	Y error (cm)	Euclidean error(cm)	X error (cm)	Y error (cm)	Euclidean error(cm)	X error (cm)	Y error (cm)	Euclidean error(cm)
6.611	2.452	7.051	16.135	6.152	17.268	3.899	2.478	4.619
13			14			15		
X error (cm)	Y error (cm)	Euclidean error(cm)	X error (cm)	Y error (cm)	Euclidean error(cm)	X error (cm)	Y error (cm)	Euclidean error(cm)
0.144	1.468	1.475	1.346	0.461	1.423	11.463	5.827	12.859

Ilustración 44-Tabla resultados para cada cámara "Mobile robot localization in industrial environments"

Tomada de: Mobile robot localization in industrial environments using a ring of cameras and ArUco markers

Se propuso un sistema de localización de robots móviles utilizando un anillo de 8 cámaras y marcadores ArUco fijados en el entorno. Se desarrolló un algoritmo de mapeo para localizar los marcadores con respecto a un sistema de referencia común. Los experimentos validaron la efectividad de los algoritmos propuestos. Este sistema de localización demostró resultados equiparables a un algoritmo SLAM con sensores costosos. La versatilidad del método basado en marcadores lo hace adecuado para entornos diversos. La precisión depende del marcador y su posición relativa a las cámaras, que puede mejorarse con un anillo de cámaras y optimización de mapas.

4.4.2 A Robust Planar Marker-Based Visual SLAM [101]

En el artículo se aborda la fusión de sensores para SLAM con enfoque en sistemas visuales-inerciales como VINS [102], ORB-SLAM3 [103], y Smart Markers [104], pero se destaca la necesidad de parámetros cámara con sensores IMU precisos. Se propone utilizar marcadores planos como ArUco [105] o AprilTag [88] para obtener poses de cámara, a pesar de ambigüedades [106]. Los resultados se comparan en precisión y velocidad con otros modelos como UcoSLAM [31], SPM-SLAM [107] y TagSLAM [108].

El artículo presenta una mejora en el sistema SLAM basado en marcadores, el cual es un sistema monocular que se utiliza para estimar las poses de una cámara y crear mapas de marcadores en tiempo real con alta precisión. La mejora se enfoca en abordar la ambigüedad en las poses de los marcadores, un desafío común en este tipo de sistemas.

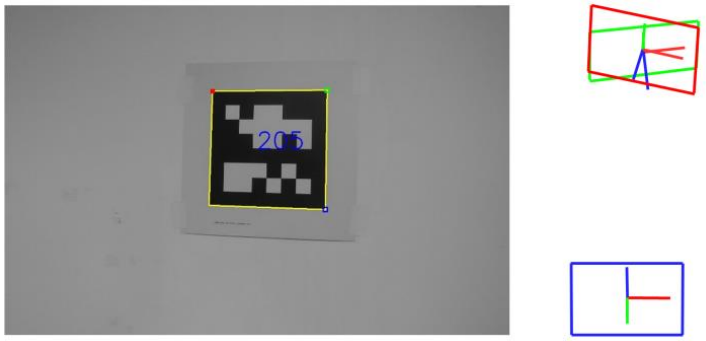


Ilustración 45-Tipo de marcador para "Robust Planar Marker-Based Visual SLAM"

Tomada de: A Robust Planar Marker-Based Visual SLAM

Se usan los siguientes sistemas de referencia para realizar SLAM: el sistema de referencia del mundo (wrs), el sistema de referencia de la cámara (crs) y el sistema de referencia del marcador (mrs). Estas transformaciones se representan mediante matrices 4x4 en el grupo de transformaciones rígidas en el espacio 3D.

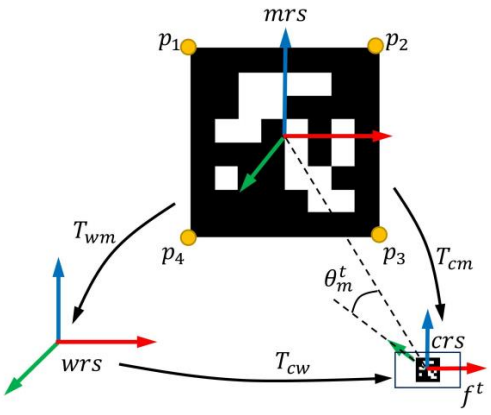


Ilustración 46-Transformaciones para calcular posición "Robust Planar Marker-Based Visual SLAM"

Tomada de: A Robust Planar Marker-Based Visual SLAM

Para solucionar el problema de ambigüedad de las poses de los marcadores se implementa el método IPPE y se calcula la relación de error de re-proyección entre estas soluciones. Además, se

presenta el cálculo del ángulo de vista de un marcador en un fotograma y se discute cómo la ambigüedad solo afecta a las componentes de rotación de la pose.

El artículo también revisa los algoritmos clave de SPM-SLAM, que son esenciales para las mejoras propuestas y sus etapas clave, que incluyen la localización de la cámara en cada fotograma, la detección de nuevos marcadores, la optimización de fotogramas clave y marcadores, y la corrección de la deriva acumulativa utilizando la optimización de grafo de poses.

En la sección sobre el proceso operativo del sistema, se resaltan las mejoras propuestas en cajas verdes. Se describe cómo el sistema inicializa el sistema de referencia del mundo, agrega marcadores y fotogramas clave al mapa, y se mencionan las mejoras en la robustez durante la inicialización en términos de movimientos de rotación.

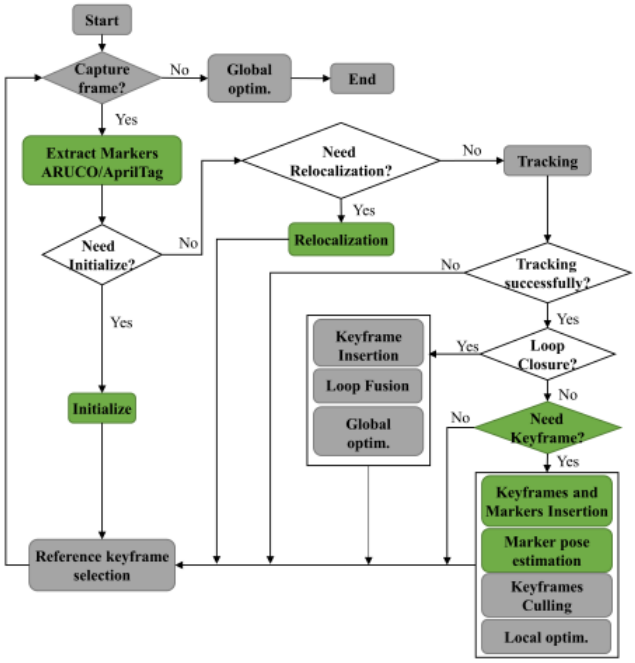


Ilustración 47-Flujo de información para "Robust Planar Marker-Based Visual SLAM"

Tomada de: A Robust Planar Marker-Based Visual SLAM

En cuanto a la inicialización de fotogramas, se establecen condiciones para el éxito de la inicialización de un solo fotograma y de dos fotogramas. Cuando se estima con éxito la pose de un fotograma Ttcw en el proceso de seguimiento, se decide si el fotograma puede considerarse como una clave y ser insertado en el mapa. Esto se basa en tres condiciones:

1. Si se detecta al menos un nuevo marcador en un fotograma, se convierte en clave y se agrega al mapa.

2. Si la distancia entre el fotograma actual ft y el fotograma clave de referencia fw es mayor que τb, se agrega al mapa.

3. Si el ángulo entre ft y fw es mayor que ∆θ, también se agrega al mapa. Esta última condición mejora la robustez del sistema.

Cuando se agrega un nuevo marcador al mapa, se estima rápidamente su posición Twm a partir de un conjunto de fotogramas clave que lo observan. Se selecciona la mejor pose Twm minimizando el error entre el marcador m y los marcadores observados (M2M) en los fotogramas clave.

Los resultados mostraron que el nuevo método superó consistentemente a SPM-SLAM, TagSLAM y UcoSLAM en términos de precisión. Además, se analizó la velocidad promedio del sistema, y se observó que el nuevo método fue considerablemente más rápido que TagSLAM y UcoSLAM, lo que es crucial para aplicaciones en tiempo real.

Table 1. Absolute trajectory errors on the SPM dataset.

Dataset	Length [m]	SPM-SLAM	TagSLAM	UcoSLAM	Ours
		ATE [m]			
sequence 01	19.5	0.060	0.430	0.084	**0.059**
sequence 02	23.3	0.046	0.054	0.079	**0.045**
sequence 03	23.1	0.055	0.233	0.058	**0.054**
sequence 04	32.8	0.014	0.027	0.037	**0.013**
sequence 05	26.9	0.017	0.023	0.070	**0.016**
sequence 06	33.2	0.017	0.026	0.028	**0.016**
sequence 07	18.9	**0.048**	0.246	0.094	0.050
sequence 08	26.1	0.064	0.066	0.131	**0.063**

Ilustración 48-Resultados para los métodos en "Robust Planar Marker-Based Visual SLAM"

Tomada de: A Robust Planar Marker-Based Visual SLAM

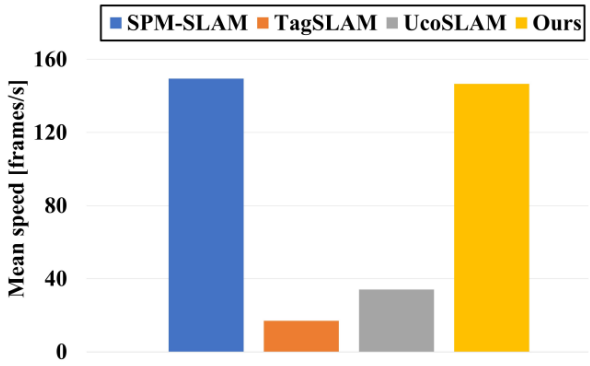


Ilustración 49-Cuadros por segundo por método "Robust Planar Marker-Based Visual SLAM"

Tomada de: A Robust Planar Marker-Based Visual SLAM

4.4.3 Mapping and localization from planar markers [35]

Este artículo se centra en la estimación de la pose de la cámara en aplicaciones como navegación de robots y realidad aumentada. Se comparan dos enfoques principales: *Structure from Motion* (SfM) y SLAM, que se basan en *key points* [109]. Se destaca la limitación de *key points* en cuanto a la invarianza a escala, rotación y cambios de vista, lo que dificulta la identificación de escenas desde diferentes perspectivas.

A pesar de la predominancia de enfoques basados en puntos clave, se ha abordado escasamente el mapeo y la localización a gran escala con marcadores planos. Algunos trabajos, como el de Hyon e Young [110], han utilizado filtros de Kalman extendidos para estimar la pose del robot en entornos con marcadores, pero no abordan la optimización de las ubicaciones de los marcadores ni la ambigüedad. Otros enfoques, como el de Klopschitz y Schmalstieg, dependen de SfM (*Structure from Motion*) para estimar la ubicación de los marcadores, lo que puede no ser siempre efectivo. En cuanto a técnicas de *Structure from Motion*, se utilizan combinaciones de imágenes y *key points* para calcular la posición relativa de las cámaras. Estos enfoques abordan desafíos importantes en SLAM

visual y marcadores, pero se necesita una investigación más profunda en mapeo a gran escala y métodos de resolución de ambigüedad.

En contraste, los marcadores planos cuadrados se diseñan para ser detectables desde una amplia gama de ubicaciones [111]. Se utilizan tanto un borde negro externo fácilmente detectable como un código binario interno para identificación. Los marcadores se componen de marcadores válidos y algoritmos de detección e identificación. Los marcadores planos cuadrados son comunes y ofrecen ventajas, ya que un solo marcador proporciona cuatro puntos de correspondencia para la estimación de la pose de la cámara. La pose de una cámara con respecto a cuatro puntos no lineales y coplanares tiene una ambigüedad de rotación.

Este estudio aborda el mapeo y la localización de marcadores planos en el contexto de SLAM visual. Se formula el problema como una minimización del error de re-proyección de las esquinas de los marcadores en múltiples fotogramas, optimizando eficientemente con el algoritmo Levenberg–Marquardt (LM) [112]. La reducción del número de variables se logra al optimizar conjuntamente las cuatro esquinas de cada marcador, asegurando la distancia real entre ellos. Se emplea un grafo de poses iniciales basado en la detección de marcadores para evitar mínimos locales, y se corrigen errores acumulativos a lo largo de los ciclos del grafo [113].

Además, se propone un método para estimar las poses de los fotogramas, incluso en presencia de soluciones ambiguas debido al problema de ambigüedad. La optimización implica los parámetros de las poses de marcadores y fotogramas, así como los parámetros intrínsecos de la cámara. Se utiliza el algoritmo LM para encontrar el mínimo del error de re-proyección, aprovechando matrices dispersas para acelerar el cálculo.

El artículo destaca la importancia de proporcionar una buena estimación inicial, especialmente si el problema tiene varios mínimos locales. Para los parámetros de la cámara, se propone una calibración offline [114] como estimación inicial.

Se utilizan transformaciones γ para mover puntos entre sistemas de referencia de marcadores y se crea un grafo de poses iniciales. La calidad de las transformaciones varía debido al ruido y otros factores, por lo que se busca encontrar las mejores poses relativas para crear un grafo de poses iniciales. Luego, se crea un grafo de poses dirigidas, donde los nodos representan marcadores y los bordes representan las poses relativas. Se utiliza un algoritmo para encontrar un camino mínimo y obtener estimaciones de poses en un sistema de referencia común.

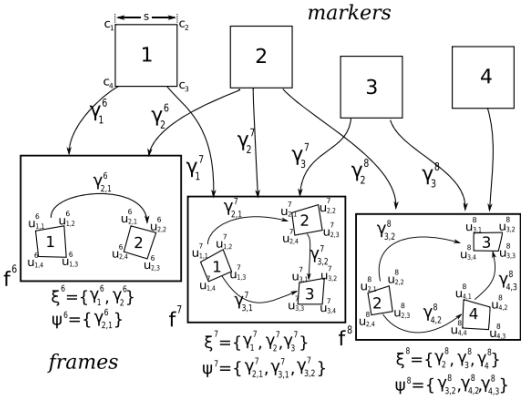


Ilustración 50-Transformaciones entre cuadros "Mapping and localization from planar markers"

Tomada de: Mapping and localization from planar markers

Se encuentra errores en las poses relativas, por lo que se emplea la técnica de propagar errores a lo largo de los ciclos del grafo [115], conocida como "*motion averaging*". En la primera etapa, se

eliminan conexiones atípicas del grafo G, evitando que corrompan la optimización. Las conexiones se eliminan basándose en un intervalo de confianza del 99% en la media. Luego, se propagan los errores a lo largo de los ciclos de G.

Los ciclos del grafo deben ser consistentes y minimizar la distancia ponderada entre las poses nuevas y antiguas. La optimización de las matrices de rotación se lleva a cabo para cada ciclo y se descompone en partes fraccionadas. Se distribuyen los errores a lo largo de los nodos de manera independiente en cada ciclo, promediando las estimaciones de rotación para los bordes que aparecen en múltiples ciclos.

Este proceso aborda la corrección de errores en las poses relativas del grafo, lo que es esencial para la estimación precisa de la ubicación de los robots basada en SLAM visual.

Este estudio presenta siete experimentos que validan un enfoque de localización basado en marcadores. Utilizaron una computadora Intel i7 sin GPU, y la detección de marcadores se realizó con la librería ArUco. Se comparan resultados del método planteado con VisualSFM y OpenMVG. El último experimento los comparó con LSD SLAM y ORB-SLAM. El último demostró la capacidad del método con un número mínimo de imágenes.

El primer experimento consistió en utilizar 32 marcadores de 1.2 cm en una caja. Los resultados fueron:

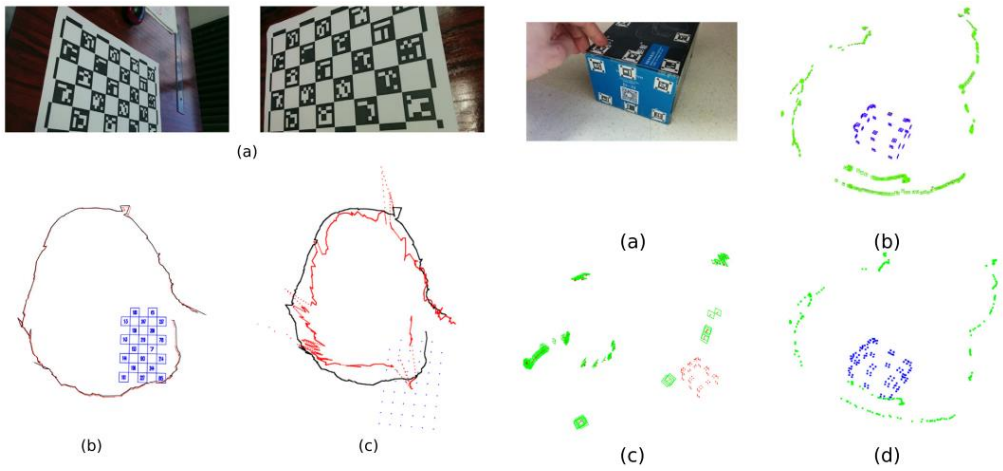


Ilustración 51-Detección de marcadores en "Mapping and localization from planar markers"

Tomada de: Mapping and localization from planar markers

Method	Comp. time	ACE	ATE
Ours	12 s	0.48 mm	4.32 mm
VisualSFM	123 s	0.64 mm	0.11 m
OpenMVG	1211 s	0.45 mm	2.24 mm

Ilustración 52-Resultados para diferentes metodologías para "Mapping and localization from planar markers"

Tomada de: Mapping and localization from planar markers

- Tiempo de procesamiento: 12 s.
- Medidas ACE (Error Absoluto de Esquina) y ATE (Error Absoluto de Trayectoria) se utilizaron para evaluar la precisión.

- El método logró una reconstrucción precisa en comparación con otros métodos.
- VisualSFM tardó 123 s, mientras que OpenMVG tuvo la mejor precisión, pero un tiempo de procesamiento mucho mayor.
- El método logró una reconstrucción precisa en 12 s, pero no había datos de referencia.
- VisualSFM encontró solo 104 de los 128 puntos y mostró errores más grandes.
- OpenMVG logró encontrar todos los marcadores, pero su tiempo de procesamiento fue mucho mayor.

El experimento de comparación con los métodos LSD-SLAM y ORB-SLAM2 se hizo en una habitación usando una cámara PtGrey FLEA3. La posición real de la cámara se registró con un sistema de captura de movimiento Optitrack. Se realizaron 3 secuencias de video y se le aplicó cada método de SLAM. Las posiciones de inicio y fin del video se encontraban separadas por una distancia de 2m.

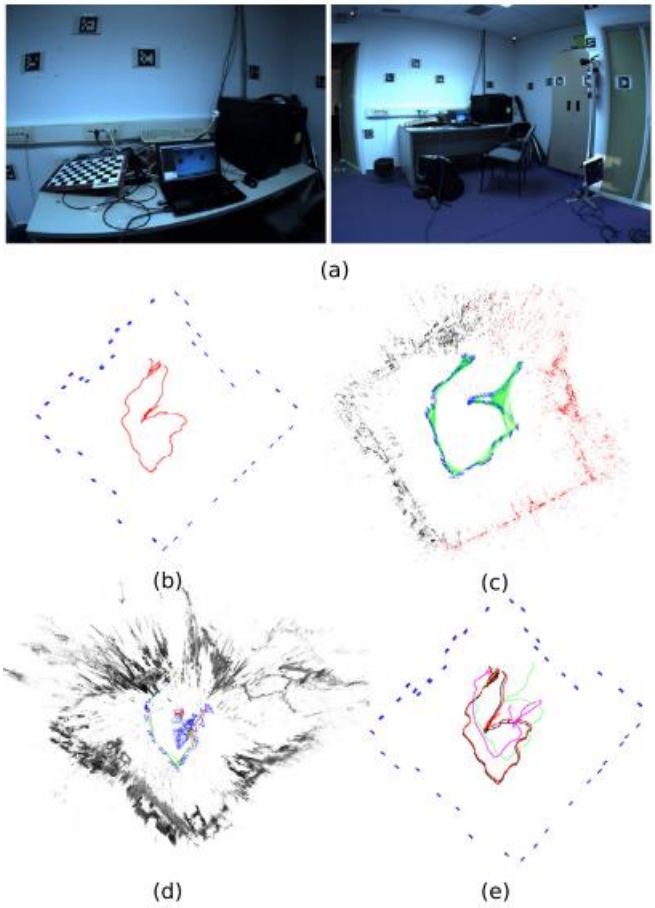


Ilustración 53-Diferentes trayectorias por metodología "Mapping and localization from planar markers"

Tomada de: Mapping and localization from planar markers

El método supera a ambos en términos de ATE con respecto a la posición real. Se muestran los resultados en la Ilustración 54.

Sequence	Ours	LSD-SLAM	ORB-SLAM2
SLAM-Seq 1	**0.0447** m	0.440 m	0.231 m
SLAM-Seq 2	**0.0433** m	0.117 m	0.913 m
SLAM-Seq 3	**0.0694** m	0.652 m	0.314 m

Ilustración 54-Resultados en diferentes secuencias para "Mapping and localization from planar markers"

Tomada de: Mapping and localization from planar markers

Este estudio presenta un enfoque innovador para la cartografía y localización mediante marcadores planos cuadrados. Se recopilan observaciones de marcadores en una secuencia de video y se crea un grafo de poses iniciales. Luego, se refina el grafo distribuyendo errores de rotación y traslación. Se obtiene una estimación inicial de las poses de los marcos utilizando posibles ambigüedades. Finalmente, se refinan todas las poses mediante una optimización de Levenberg-Marquardt para reducir el error de re-proyección. El método se compara con técnicas de Estructura desde el Movimiento (SfM) y SLAM basadas en puntos clave, demostrando un rendimiento superior en la obtención de mapas y resultados de localización bajo una variedad de condiciones.

4.4.4 An Improvement on ArUco Marker for Pose Tracking Using Kalman Filter [116]

Este artículo se centra en el seguimiento de objetos mediante técnicas de visión por computadora, con un enfoque en el uso de marcadores ArUco para estimar la posición del objeto y resolver problemas de ocultamiento y ruido. Los marcadores fiduciales, como ArUco, se utilizan para una detección precisa y rápida. El seguimiento se mejora mediante un filtro de Kalman, permitiendo la estimación durante el ocultamiento y suavizando la posición resultante. Además, se destaca la aplicabilidad en dispositivos móviles debido a bajos requisitos de cómputo.

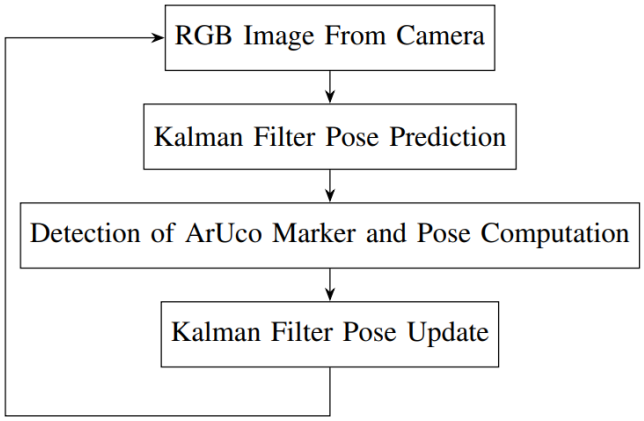


Ilustración 55-Flujo de información para "An Improvement on ArUco Marker for Pose Tracking"

Tomada de: An Improvement on ArUco Marker for Pose Tracking Using Kalman Filter

Se ha desarrollado una plataforma de pruebas para demostrar los efectos de la aplicación del filtro Kalman en el sistema propuesto. La plataforma utiliza un motor paso a paso para controlar la posición del marcador ArUco, permitiendo movimientos precisos. Una cámara web captura el movimiento del marcador, cuyas imágenes son procesadas por el sistema. Para probar la robustez del algoritmo, se oculta el marcador en algunas partes de la secuencia de imágenes.

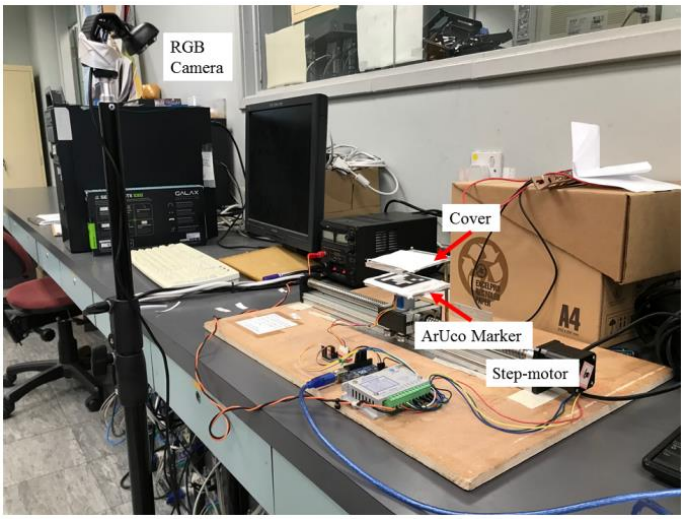


Ilustración 56-Entorno de trabajo para "An Improvement on ArUco Marker for Pose Tracking"

Tomada de: An Improvement on ArUco Marker for Pose Tracking Using Kalman Filter

En un experimento, se colocó el marcador ArUco en una plataforma y se movió con un motor paso a paso, ocultándolo con una cubierta en ciertos momentos. La Ilustración 57 muestra los resultados, donde el filtro Kalman permitió estimar la posición incluso cuando el marcador estaba oculto, demostrando su capacidad para manejar la oclusión y brindar una estimación óptima.

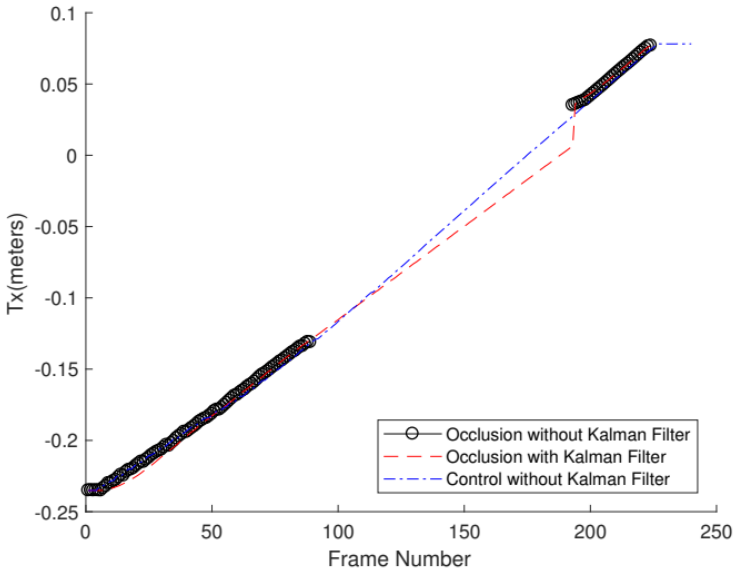


Ilustración 57-Resultados para "An Improvement on ArUco Marker for Pose Tracking" con oclusión

Tomada de: An Improvement on ArUco Marker for Pose Tracking Using Kalman Filter

Otro experimento realizado en la investigación se enfoca cuando hay presencia de ruido en las imágenes o vibración de los marcadores debido al movimiento. En estos casos, la precisión de la pose obtenida por el marcador ArUco se ve afectada. Para mejorarla, se aplicó el filtro Kalman. La Ilustración 58 muestra que sin el filtro Kalman, la pose se ve afectada por el ruido, lo cual es indeseable en aplicaciones como la realidad virtual o mapeo. Con el filtro Kalman, la pose se mantiene estable.

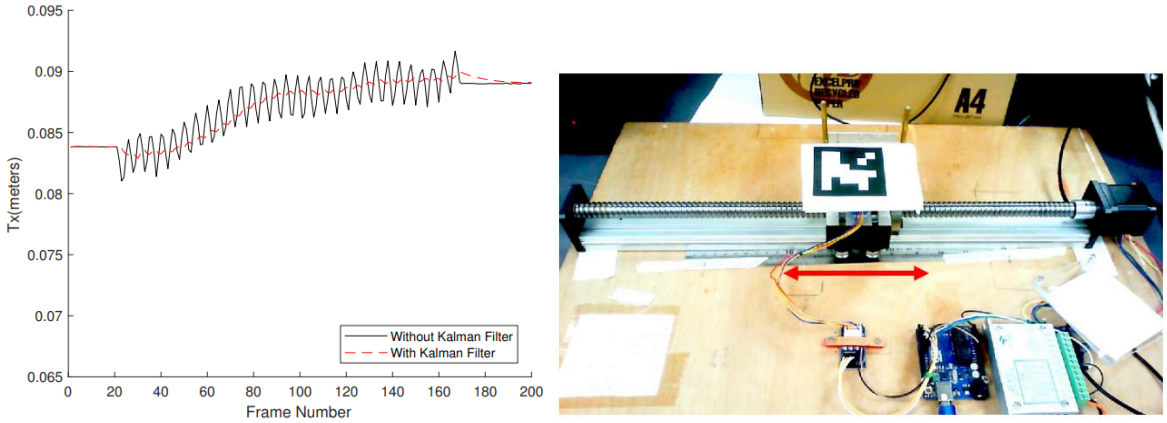


Ilustración 58-Resultados para "An Improvement on ArUco Marker for Pose Tracking" en vibración

Tomada de: An Improvement on ArUco Marker for Pose Tracking Using Kalman Filter

Los resultados experimentales demuestran la efectividad de la aplicación del filtro Kalman en la mejora de los resultados del marcador ArUco.

4.4.5 Uco-SLAM [31]

Los visual-SLAM como ORB y LDSO son SLAM de uso común pero cuentan con varias desventajas. Entre ella es que los mapas generados no cuentan con una escala definida, fallan cuando se da un movimiento de rotación puro, requieren de diversidad en texturas para crear los KP y la re-localización falla cuando hay patrones repetitivos lo que los hace inútiles para la navegación autónoma. Debido a esta situación se crea el Uco-SLAM. Este método es la combinación entre marcadores y KP (obtenidos de un modelo ORB modificado).

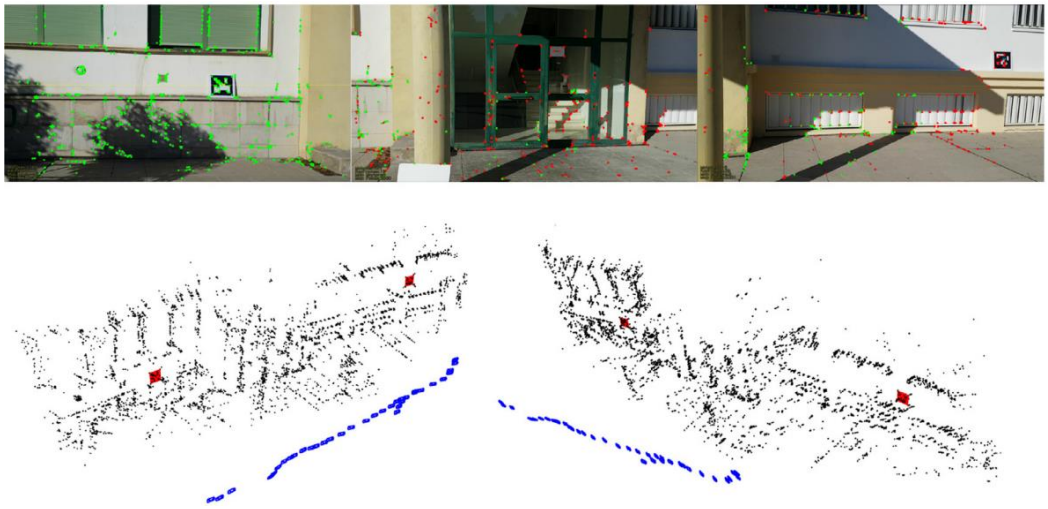


Ilustración 59-Combinación marcadores y ORB en "Uco-SLAM"

Tomada de: Uco-SLAM

Los marcadores eliminan el problema de re-localización, ya que cada marcador cuenta con un respectivo ID eliminando así la ambigüedad en los entornos repetitivos y los falsos cierres de bucles. Elimina tanto el problema de las texturas como el de la escalabilidad, pues el marcador frente a temas de iluminación sigue siendo el mismo y se conoce su tamaño real. Los colores son negro y

blanco, ya que hay un alto contraste entre ellos. Este contraste lo visualiza la cámara y así pueden determinar su ubicación, orientación e ID. Si luego de un determinado tiempo, se vuelve a visualizar un ID que haya sido reconocido, se procede a hacer un cierre de bucle.

Para detectar el marcador y su posición, se deben identificar las 4 esquinas del cuadro negro. Cuando se detecta un marcador, los nodos de las esquinas indican en que *key frame* (KF o marco de referencia) se encuentra el marcador y en que coordenadas de pixeles. Los marcos no obtienen su posición hasta que sean vistos en al menos 2 fotogramas más (3 en total), así se puede triangular su posición. Se debe evitar la “ambigüedad de pose planar” que se da cuando el marcador no se encuentra completamente plano.

El método propuesto para inicializar el mapa es obtener el primer par de fotogramas (F0 y F1). En estos se buscan dos enfoques, el primero busca los primeros KP cercanos y luego se busca el primer marcador (así si no se encuentran KP, se puede inicializar buscando el marcador). Si falla tanto al reconocer los KP como los marcadores, se toma un fotograma nuevo (F2) y se realiza la misma comparación entre F0 y F2. Si nuevamente no tiene éxito, se cambia F0 por F actual, hasta que se pueda inicializar el mapeo.

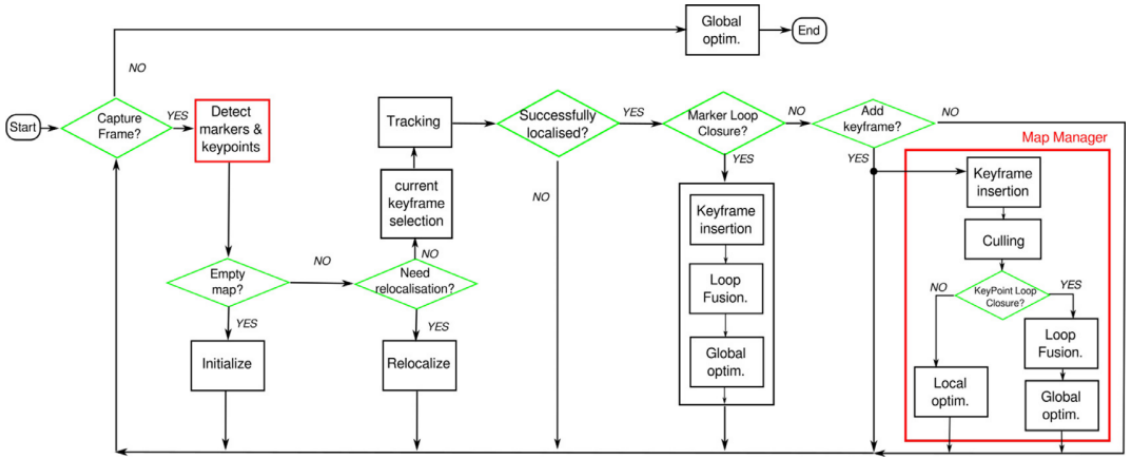


Ilustración 60-Flujo de información para "Uco-SLAM"

Tomada de: Uco-SLAM

Una vez inicializado el mapeo y ubicación inicial, el sistema intenta estimar la posición de la cámara usando la posición anterior como punto de partida. Para esto se usan los KP observados en el fotograma anterior, ya que es muy probable que estos se repitan en el fotograma actual. Estas coincidencias, junto con la estimación de la posición del marcador observado, proporcionan una estimación de la posición actual.

Se utiliza una estrategia de “supervivencia” para tener eficiencia en los KP. Esta indica que para agregar un KP, este debe estar visible en al menos 2/3 de los siguientes fotogramas, hasta que se agreguen 2 KF. También se usa para mejorar el rendimiento de KF. Para este caso, no se crea otro KF hasta que los KP compartidos sean menor a un % definido.

Para validar la efectividad de Uco, se compara con ORB y LDSO. El experimento usa las bases de datos Kitti, Euroc-MAV, TUM y SPM. Para saber si un método es mejor que otro, proponen un método diferente de comparación. En este método se evalúa el error de trayectoria absoluto (ATE) solo en los marcadores rastreados por ambos métodos. La metodología proporciona un valor Sρ(a , b) entre -1 y 1, donde A es 1 SLAM y B el otro. Si el resultado es 0 significa que el ATE comparado entre A y B son iguales, si el valor es 1 significa que A es mejor que B. Se calcula como el ATE del método A comparado con el ATE del método B, en los fotogramas seguidos por ambos métodos.

Las secuencias de las bases de datos se corren 2 veces en cada SLAM. La primera vez es para realizar el mapeo y la segunda para la localización. Como ORB trabaja en paralelo y no es secuencia, se modifica el método para que se pueda comparar con los otros, se le da el nombre ORB-SLAM2. En la tabla 7, la columna S1p(a,b) muestra la comparación de los errores obtenidos durante la primera ejecución, la columna S2p(a,b) muestra los resultados después de la optimización global y la columna P es el valor de confidencia usado en el método de comparación para evitar que diferencias infinitesimales entre el ATE de A y de B sean consideradas como relevantes (threshold). En esta parte del experimento solo se aplica KP para poder comparar los métodos.

P	A	B	S1p(a,b)	S2p(a,b)
0.01	LDSO	ORB-SLAM2	-0.23	-0.34
0.01	LDSO	UcoSLAM	-0.22	-0.37
0.01	ORB-SLAM2	UcoSLAM	-0.01	-0.14
0.05	LDSO	ORB-SLAM2	-0.37	-0.34
0.05	LDSO	UcoSLAM	-0.34	-0.37
0.05	ORB-SLAM2	UcoSLAM	-0.01	-0.12
0.1	LDSO	ORB-SLAM2	-0.37	-0.3
0.1	LDSO	UcoSLAM	-0.36	-0.4
0.1	ORB-SLAM2	UcoSLAM	0	-0.12
0.25	LDSO	ORB-SLAM2	-0.36	-0.3
0.25	LDSO	UcoSLAM	-0.35	-0.37
0.25	ORB-SLAM2	UcoSLAM	0	-0.1

Ilustración 61-Valores por modelo con valor de confidencia en "Uco-SLAM"

Tomada de: Uco-SLAM

De los resultados de la tabla 7, se tiene que LDSO es el peor método y ORB comparado con Uco son muy similares los resultados, siendo Uco un poco mejor en la segunda ejecución de secuencia. Esto debido a que la metodología implementada en Uco genera más KF que ORB. Como Uco es el método que mejor desempeño tuvo, se implementa el método comparando Uco solo con KP, con solo marcadores y fusionados (KP con marcadores). El experimento utiliza 8 videos. La tabla 8 muestra los resultados de ATE y % trayecto.

	Uco (KP)		Uco (M)		Uco (KP+M)	
	ATE	%trck	ATE	%trck	ATE	%trck
Video 1	0.601	64.5	0.065	98.4	0.057	100
Video 2	0.057	99.8	0.048	98.4	0.051	99.8
Video 3	0.108	99.8	0.054	98.1	0.053	99.8
Video 4	0.01	99.8	0.013	99	0.008	99.8
Video 5	0.015	99.5	0.014	98.8	0.011	99.7
Video 6	0.687	50	0.014	98.6	0.011	99.7
Video 7	1.28	100	0.053	98.7	0.05	100
Video 8	0.052	99.8	0.062	99.2	0.071	99.8

Ilustración 62-Resultados para cada método en "Uco-SLAM"

Tomada de: Uco-SLAM

El promedio de %std es 89.15% para Uco (KP), 98.65% para Uco(M) y 99.83% para Uco(KP+M). Como Uco (KP+M) tiene el mejor resultado, se demuestra que la combinación de marcadores y KP es más efectiva que cualquiera de los métodos por individual.

5 METODOLOGÍA

A continuación, se presenta en detalle la metodología empleada en este estudio. Describiendo cómo se llevará a cabo la investigación y cómo se abordarán los objetivos y preguntas de investigación. En esta sección, se proporcionarán explicaciones detalladas de los procedimientos, enfoques y herramientas utilizados para lograr los objetivos del proyecto. La metodología se divide en varias etapas clave, desde la configuración del entorno de trabajo hasta la evaluación de la precisión de la posición del robot en comparación con un *groundtruth*. Cada paso se abordará minuciosamente para ofrecer una comprensión completa de la estrategia de investigación empleada en este estudio.

5.1 SELECCIÓN DE GROUNDTRUTH

El *groundtruth* cumple un papel muy importante en el contexto de proyectos que implementan metodologías SLAM, ya que permite ser un comparativo con respecto al rendimiento de los algoritmos que se implementan. Se compara los datos de salida del algoritmo para determinar su precisión y exactitud, así como para determinar cuál algoritmo es el más adecuado de acuerdo con el enfoque de cada proyecto. También sirve para ajustar los parámetros específicos de los modelos para así poder mejorarlos en términos de rendimiento, que pueden llevar al descubrimiento de nuevos modelos y algoritmos.

Por estos motivos es importante destacar la correcta selección y generación del *groundtruth*. Los datos de verdad terreno deben ser recopilados de manera precisa y con instrumentación adecuada, lo que puede ser costoso no solo en la adquisición de equipos de alta calidad, sino también en costo computacional. Además, es crucial que los datos de verdad terreno sean representativos del entorno y las condiciones en las que se desplegará el sistema SLAM, de lo contrario, las evaluaciones y las pruebas pueden no reflejar con precisión el rendimiento real del sistema.

Debido a las características de nuestro proyecto, se decide implementar los marcadores tipo ArUco como método de *groundtruth* sustentando esta decisión en las investigaciones High-accuracy Fiducial Markers for Groundtruth y On construction of a reliable groundtruth for evaluation of visual slam algorithms , donde en el título Estado del Arte se exponen los resultados y cómo se puede implementar la metodología para el *groundtruth* con marcadores.

En el contexto de nuestra investigación, adaptamos la metodología propuesta a nuestro proyecto mediante la ubicación de dos marcadores tipo ArUco. El primero es el marcador con identificador 1 de la librería de ArUco, este se sitúa en la parte superior de la plataforma del robot Kobuki, ubicando el centro del marcador aproximadamente en el centro del robot como se muestra en la Ilustración 63.

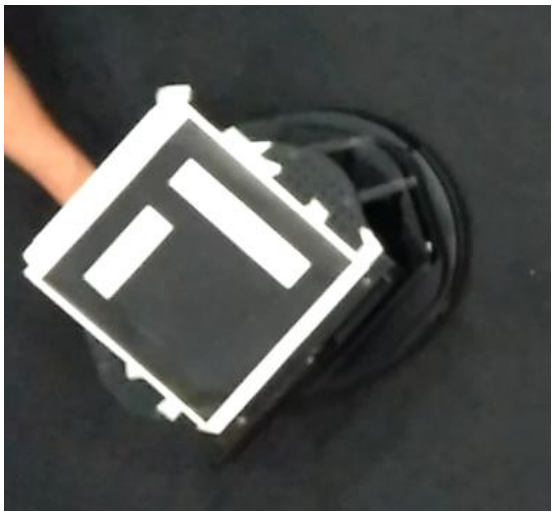


Ilustración 63-Ubicación de marcador en Kobuki para groundtruth

Elaboración propia

El segundo marcador, de dimensiones idénticas, se posiciona en un punto fijo dentro de la habitación. Elegimos estratégicamente ubicar este segundo marcador sobre la mesa central de la habitación. Esta elección se justifica por la incapacidad del Kobuki para pasar por encima del marcador y ocultarlo. Además, esta ubicación garantiza que el marcador sea visible desde prácticamente cualquier posición en la habitación.

El marcador ubicado en el centro de la mesa está etiquetado como el marcador 0 en la biblioteca de ArUco. Se encuentra a una distancia aproximada de 205 mm en el eje X y 83 mm en el eje Y, medidos desde la esquina superior derecha de la misma mesa central. Además, este marcador está orientado con su propio sistema de coordenadas, como se representa en la Ilustración 64.

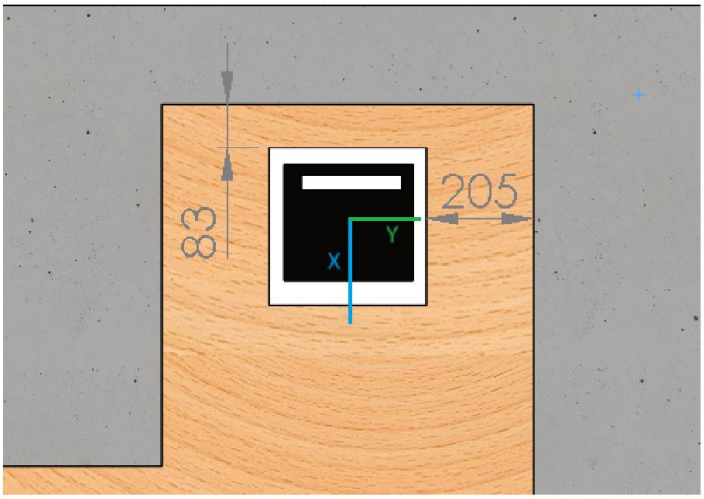


Ilustración 64-Ubicación de marcador fijo para groundtruth

Elaboración propia

Para mantener un sistema de coordenadas de referencia global constante, el sensor de cámara designado como MotoG5 debe mantener una vista constante del marcador con identificador 0. La distancia relativa entre el sensor y el marcador en los ejes X, Y y Z debe mantenerse constante en

todo momento. Es por esta razón que se ha elegido la posición previamente mencionada como la referencia para el sistema de coordenadas global.

Se utiliza la librería kobuki_real_world_pos desarrollada y explicada en el título ArUco mapping implementado en ROS, para por medio de la cámara, se pueda encontrar el SGR (Sistema Global de Referencia) asignado al marcador con identificado 0 y se calcule la posición global de la cámara. En el mismo frame, se identifica el SLK (Sistema Local del Kobuki), este se traslada al SLC (Sistema Local de la Cámara), para posteriormente trasladarlo al SGR, cómo se muestra en la Ilustración 65.

Se emplea la librería denominada kobuki_real_world_pos cuyo desarrollo, explicación y funcionamiento han sido detallados en el título ArUco mapping implementado en ROS. Esta librería se utiliza con el propósito de emplear la cámara para localizar el Sistema Global de Referencia (SGR) asociado al marcador con identificación 0, y a partir de esta información, calcular la posición global de la cámara.

En el mismo cuadro de referencia, también se identifica el Sistema Local del Kobuki (SLK). Luego, se realiza una transformación para llevarlo al Sistema Local de la Cámara (SLC). Posteriormente, se lleva a cabo otra transformación para situarlo en el Sistema Global de Referencia (SGR), tal como se muestra en la Ilustración 65.

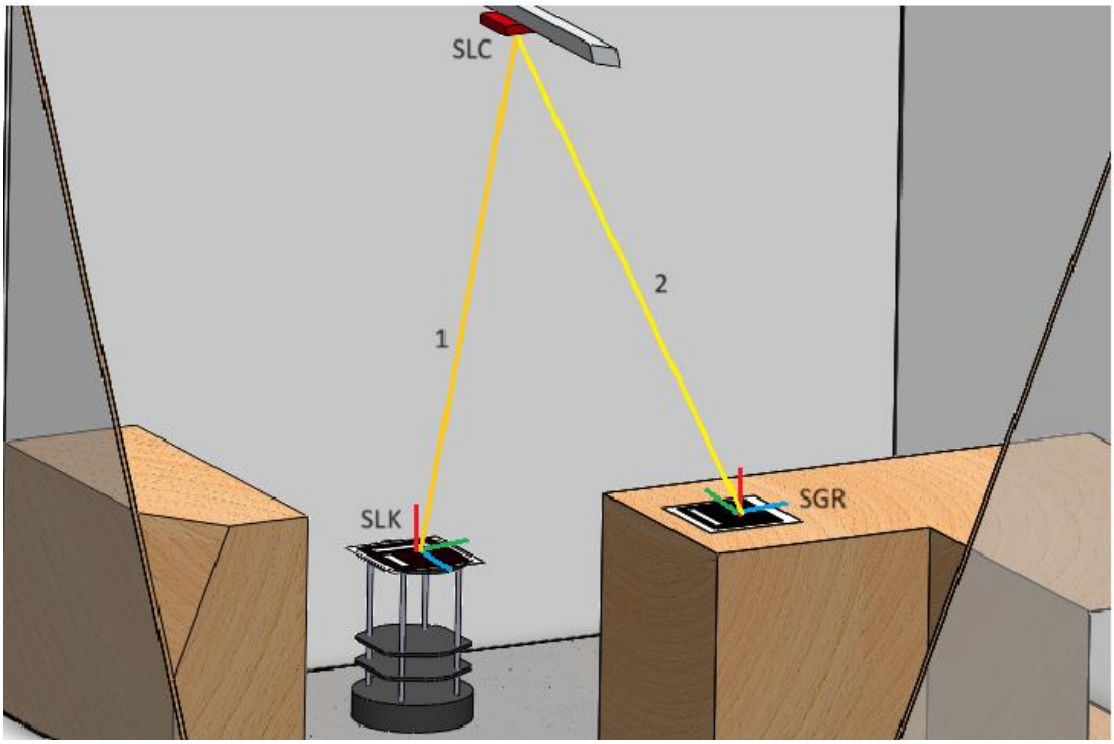


Ilustración 65-Tranformaciones para sistema global groundtruth

Elaboración propia

Es relevante destacar que, en todo este proceso, se registra y almacena la información de coordenadas en el espacio-tiempo en el que se capturaron los datos. De esta manera, se obtienen los datos necesarios para generar el *groundtruth*, que se utiliza para evaluar y validar el desempeño del algoritmo en el marco de esta investigación de maestría.

5.2 SELECCIÓN DE MARCADORES

La implementación de marcadores tipo ArUco para este proyecto presenta diversas ventajas significativas las cuales son el motivo de su selección como tipo de marcador para implementar en el proyecto. En primer lugar, la facilidad de detección de los marcadores ArUco se destaca como un atributo clave, permitiendo una identificación rápida y precisa en tiempo real. Este aspecto es respaldado por su capacidad para proporcionar información precisa de pose, lo cual es esencial para el desarrollo de la investigación.

Es importante destacar que los marcadores ArUco cuentan con una amplia documentación y han sido implementados exitosamente en una variedad de proyectos, como se evidencia en investigaciones previas, por ejemplo, en Mobile robot localization in industrial environments using a ring of cameras and ArUco markers y Mapping and localization from planar markers , presentados en el estado del arte de la tesis. Además, la versatilidad de ArUco se refleja en su capacidad para integrarse con otros métodos de SLAM, como se ilustra en trabajos como An Improvement on ArUco Marker for Pose Tracking Using Kalman Filter y Uco-SLAM , así como en A Robust Planar Marker-Based Visual SLAM , también presentados en el estado del arte de la tesis.

En comparación con otros marcadores, como Apriltag, ArUco presenta ventajas notables. Apriltag, según el artículo Analysis and Improvements in AprilTag Based State Estimation , exhibe limitaciones en términos de precisión, especialmente en función de la distancia, el ángulo de visión y giro de la cámara. Además, en Experimental Comparison of Fiducial Markers for Pose Estimation , aunque Apriltag muestra resultados sólidos, ArUco demuestra un rendimiento comparable en términos de distancias y ángulos de cámara.

Aunque Chromatag se destaca en ChromaTag: A Colored Marker and Fast Detection Algorithm , su comparación con ArUco se encuentra ausente en la literatura revisada. Además, según nuestro conocimiento, los marcadores Chromatag no cuentan con una biblioteca compatible con el entorno de ROS, representando una limitación importante en términos de integración y flexibilidad.

Otros factores cruciales que respaldan la elección de ArUco incluyen la disponibilidad de una biblioteca publicada para ROS Melodic y Noetic, accesible en https://wiki.ros.org/ArUco_ros. Esta biblioteca, detallada en la sección de Librería ArUco, facilita la implementación y el desarrollo en el entorno de ROS.

Finalmente, la accesibilidad y el bajo costo de los marcadores ArUco son aspectos destacados, ya que solo requieren una impresora con buena calidad de impresión, preferiblemente láser, y papel de color blanco y tinta de un solo color (negro). La posibilidad de elegir el tamaño según las condiciones específicas de iluminación y los espacios disponibles hace que la implementación de ArUco sea una opción práctica y económicamente eficiente para el proyecto de maestría.

5.3 COMPUTADOR, USO DE ROS, INSTALACIÓN, PAQUETES

El sistema computacional empleado en este proyecto es una computadora ASUS equipada con un procesador Intel Core i5 de 8ª generación, 8 GB de RAM y una unidad gráfica Intel Graphics. Este dispositivo opera bajo el sistema operativo Ubuntu 18.04 y está proporcionado por la Universidad EAFIT, la cual nos otorgó acceso para la ejecución de la investigación. Este equipo está equipado con el entorno ROS Melodic, esencial para la adquisición de datos, la visualización de resultados, la instalación de bibliotecas, el establecimiento de conexiones y la implementación del algoritmo de SLAM.

La elección de ROS Melodic sobre ROS Noetic, a pesar de que ambos admiten la librería de ArUco, se basa en consideraciones clave. Ambos entornos están fundamentados en Python, un lenguaje de programación en el cual el autor de la tesis posee habilidades avanzadas. Esta familiaridad facilita cualquier ajuste en el entorno, la creación de nuevas bibliotecas y otros procesos. Además, ROS

Melodic es una versión de Soporte a Largo Plazo (LTS), lo que implica que recibirá actualizaciones de mantenimiento y soporte durante un periodo más prolongado.

Una distinción relevante es que ROS Melodic está diseñado para operar en Ubuntu 18.04 LTS, mientras que ROS Noetic está diseñado para Ubuntu 20.04 LTS. La elección de Ubuntu 18.04 se basa en la prevalencia de documentación, videos y proyectos relacionados con ArUco en este entorno hasta el momento de la realización del proyecto. Se destaca que en Ubuntu 20.04 aún persisten errores en la librería debido a sus nuevas actualizaciones e integraciones, problemas que ya han sido solventados en la versión 18.04.

5.3.1 Configuración de ROS

Explicado el razonamiento detrás de la elección de ROS Melodic, a continuación se detallan los pasos para su instalación. El primer paso, desde la consola de Ubuntu, implica la ejecución del siguiente script para configurar el sistema para aceptar el software de ROS:

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

Seguido de esto, se instala la librería de Curl para configurar las claves de acceso al git de ROS:

1. sudo apt install curl # if you haven't already installed curl
2. curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -

Antes de la instalación de ROS Melodic, es crucial asegurarse de que el sistema esté actualizado a su última versión mediante el script:

- sudo apt update

Con el sistema actualizado, se procede a la instalación de ROS Melodic. Para este proyecto, se recomienda instalar todas las librerías y herramientas que ofrece ROS Melodic mediante el siguiente script:

```
sudo apt install ros-melodic-desktop-full
```

Con ROS Melodic instalado, se debe configurar el entorno de trabajo. Siguiendo la recomendación de ROS, se añade el entorno al Bash de Ubuntu para que se incluya automáticamente en cada nueva consola lanzada, evitando configuraciones manuales. Esto se logra mediante el siguiente script:

```
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc

source ~/.bashrc
```

Hasta este punto, ROS Melodic con sus librerías y herramientas están instalados y configurados en el Bash de Ubuntu. El siguiente paso consiste en la instalación de las dependencias de ROS para construir paquetes. Los paquetes son entornos de trabajo independientes en los cuales se pueden manejar diferentes herramientas y librerías. Para realizar esta tarea, se ejecuta el siguiente script:

```
sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build-ess
ential
```

Finalmente, para ejecutar el entorno de trabajo de ROS, se debe dirigir a la ubicación del entorno de trabajo desde la consola (más adelante se detalla cómo crear el entorno de trabajo) y ejecutar el siguiente comando:

```
sudo rosdep init
rosdep update
```

Establecimiento de Entornos de Trabajo y Utilización de Paquetes en ROS:

La instauración de un entorno de trabajo en ROS y la estructuración del código en paquetes representan un enfoque altamente beneficioso, con notables ventajas en términos de modularidad, reusabilidad, gestión de dependencias, comunicación entre nodos y herramientas de desarrollo. Estas ventajas convergen para propiciar un desarrollo más eficiente y colaborativo de sistemas robóticos, lo cual es crucial en el contexto de aplicaciones avanzadas y complejas.

La relevancia de crear un entorno de trabajo y sus respectivos paquetes radica en su capacidad para optimizar la organización y la eficacia del desarrollo. Para iniciar este proceso, se requiere navegar a través de la consola hasta el directorio deseado para el entorno de trabajo y ejecutar el siguiente script:

```
mkdir -p ~/nombre_entorno/src #crear carpetas
cd ~/nombre_entorno/ #moverse a la carpeta
catkin_make #crear el entorno de trabajo
```

Estos comandos no solo generan el entorno de trabajo con el nombre designado, sino que también crean las carpetas de Build y Devel, donde se almacenan los archivos de configuración correspondientes a los paquetes y al entorno de trabajo, respectivamente. Posteriormente, se activar el entorno de trabajo utilizando el siguiente script:

```
source devel/setup.bash
```

Con el entorno de trabajo establecido y activado (siempre debe ser activado para correr cualquier código en ROS del paquete), el siguiente paso implica la creación de paquetes dentro de este entorno. Esta práctica modulariza el proyecto, permitiendo segmentar diferentes funcionalidades en distintos paquetes. Por ejemplo, se puede diseñar un paquete para utilizar ArUco y determinar su posición con relación a un sistema coordenado global, mientras que otro paquete puede centrarse únicamente en el desplazamiento del Kobuki.

Para agregar la librería de ArUco como dependencia del paquete, primero debe ser instalada en el entorno de Ubuntu. Para eso se ejecuta el siguiente script:

```
sudo apt-get install ros-melodic-ArUco-ros
```

La creación de paquetes se realiza desde la carpeta “src” del entorno de trabajo mediante la ejecución del siguiente script:

```
catkin_create_pkg nombre_paquete dependencia1 dependencia2
```

Este comando, al ser ejecutado, establece un paquete con el nombre especificado y define las dependencias necesarias, como std_msgs, rospy, roscpp, entre otras. Para conocer más sobre ROS y la forma de trabajar en él, visitar la página https://wiki.ros.org/ROS/Tutorials.

5.4 ENTORNO DE TRABAJO DEL PROYECTO

Al comprender de manera integral la configuración de entornos de trabajo mediante la implementación de paquetes y los beneficios inherentes a este proceso, se procederá a una exposición detallada de la estructura del entorno de trabajo diseñado específicamente para el proyecto de investigación en cuestión. En esta sección, nos enfocaremos exclusivamente en la arquitectura del entorno de trabajo y en la descripción de los paquetes pertinentes, así como en su aplicación concreta. La estructura del entorno de trabajo se presenta en la Ilustración 66:

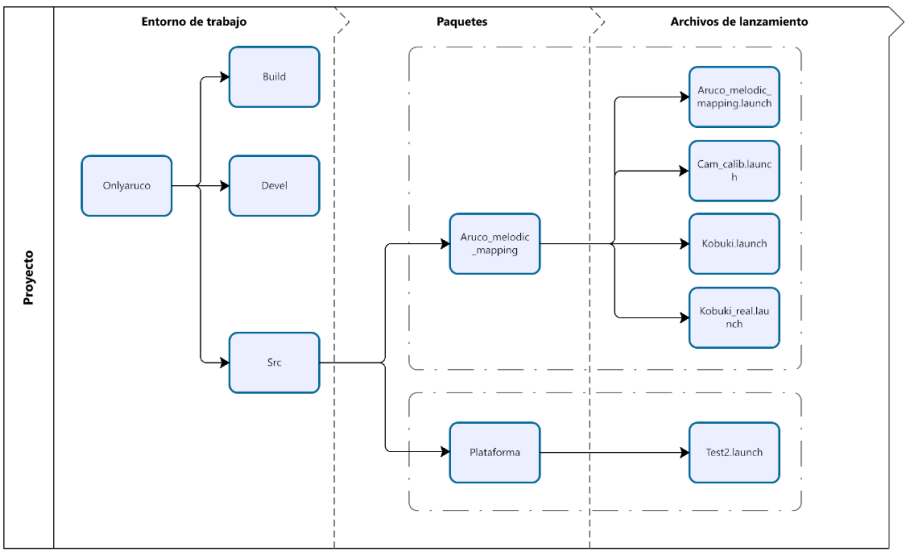


Ilustración 66-Estructura de documentos paquete OnlyArUco

Elaboración propia

El funcionamiento de cada paquete se basa en los archivos de lanzamiento que contienen, ya que estos son los que ejecutan los códigos con los algoritmos, configura sus variables e indica las librerías externas que se deben utilizar para el funcionamiento. Se debe recordar siempre entrar en el paquete creado en ROS y activarlo para que estos archivos ".launch" puedan ejecutarse. A continuación, se explica que hace cada archivo de ejecución de cada paquete.

5.4.1 Paquete ArUco_melodic_mapping

5.4.1.1 ArUco_melodic_mapping.launch

Este archivo desempeña un papel crucial en nuestra investigación, ya que se emplea para iniciar la librería de ArUco y ejecutar el algoritmo diseñado para localizar marcadores en el sistema coordenado global, lo que contribuye a la creación del mapa. Su función principal es establecer la conexión entre el sensor celular Xiaomi Note 8 y ROS, al tiempo que abre la aplicación Rviz para visualizar las ubicaciones de los marcadores identificados. Se proporciona una explicación más detallada en la sección ArUco mapping implementado en ROS.

5.4.1.2 Cam_calib.launch

Este archivo facilita la conexión del teléfono Xiaomi Note 8 con ROS para llevar a cabo la calibración de la cámara mediante la utilización de la librería camera_calibration en el entorno de ROS. La importancia de la calibración de la cámara, así como las instrucciones detalladas sobre cómo llevar

a cabo este proceso, se encuentran explicadas de manera exhaustiva en la sección titulada Calibración de cámaras usando ROS.

5.4.1.3 Kobuki.launch

Este documento facilita la recopilación de datos relativos a la ubicación (X, Y, Z) y las rotaciones del robot Kobuki. Estos datos son adquiridos mediante el sensor celular Xiaomi Note 8 y se registran en relación con el entorno circundante. La información recopilada se almacena para su posterior procesamiento y representación gráfica en Excel, aunque esta última fase debe llevarse a cabo manualmente. Para obtener detalles adicionales sobre el proceso de extracción de la información y su disponibilidad para su uso, se recomienda consultar la sección titulada Experimento y extracción de resultados.

5.4.1.4 Kobuki_real.launch

Este documento facilita la recopilación de datos de *groundtruth* del robot Kobuki en cuanto a la ubicación (X, Y, Z) y las rotaciones. Estos datos son adquiridos mediante el sensor celular MotoG5. La información recopilada se almacena para su posterior procesamiento y representación gráfica en Excel, aunque esta última fase debe llevarse a cabo manualmente. Para obtener detalles adicionales sobre el proceso de extracción de la información y su disponibilidad para su uso, se recomienda consultar la sección titulada Experimento y extracción de resultados.

5.4.2 Paquete Plataforma

5.4.2.1 Test2.launch

Este archivo se emplea para iniciar Rviz, una herramienta que presenta un modelo tridimensional del robot en un mapa predefinido. Además, se detalla el tipo de robot que se visualizará en Rviz y activa el paquete de movimiento del robot. Este paquete facilita el control de la velocidad lineal y rotacional del robot mediante el teclado del ordenador conectado al Kobuki, lo que posibilita la navegación del robot desde un punto fijo.

En el proceso de creación de nuevos archivos con extensión ".launch", es imperativo considerar la necesidad de conferir al archivo la capacidad de ejecutarse.

Este procedimiento debe realizarse mediante la siguiente secuencia de acciones: en primer lugar, se debe dirigir al directorio que alberga el archivo en cuestión. Una vez allí, se procederá a efectuar un clic derecho sobre el archivo, posteriormente se seleccionará la opción "Propiedades", dentro de la pestaña "Permisos", se activará la casilla correspondiente a "Permitir ejecutar el archivo como un programa". Asimismo, en la sección designada como "Propietario", se elegirá la configuración "Lectura y escritura".

Ilustración 67-Volver ejecutable un archivo

Elaboración propia

5.5 SENSORES A UTILIZAR

En un inicio, se optó por utilizar una cámara Logitech C270 como sensor para la captura de imágenes en el proceso de SLAM. Sin embargo, al poner a prueba este enfoque con marcadores, se revelaron deficiencias notables en cuanto a la calidad de la luz y de las imágenes. La cámara tenía dificultades para detectar con claridad los marcadores, y con frecuencia confundía sus identificadores. Con base en estas limitaciones, se tomó la decisión de emplear un teléfono móvil Xiaomi Redmi Note 8 como sensor en el robot Kobuki, y un Motorola Moto G5 como sensor para la implementación del *groundtruth*, que se explica con detalle en el título selección de groundtruth. Estos teléfonos móviles presentan las siguientes especificaciones relacionadas con sus cámaras:

Xiaomi Redmi Note 8:

- Cámara principal: 48 megapíxeles

- Tipo de enfoque: Enfoque automático por detección de fase (PDAF)

- Apertura: f/1.79

Motorola Moto G5:

- Cámara principal: 13 megapíxeles

- Tipo de enfoque: Enfoque automático por detección de fase (PDAF)

- Apertura: f/2.0

La elección de utilizar teléfonos móviles como sensores presenta ventajas y desventajas en la ejecución del proyecto. Uno de los principales beneficios radica en la capacidad de implementar cámaras con una resolución sustancialmente mayor y una mayor cantidad de megapíxeles en comparación con la Logitech. Además, ambas cámaras cuentan con funciones de enfoque automático, lo que permite capturar de manera más precisa los cuadros de imágenes a medida que el escenario experimenta cambios en la iluminación y movimiento.

Sin embargo, existen desafíos significativos que acompañan esta elección. Por un lado, no es posible conectar directamente la cámara de un teléfono móvil al computador mediante un cable. Hasta nuestro conocimiento, no existe una librería que permita integrar la cámara de un celular con ROS por medio de cable y que permita procesar los frames de manera eficiente. Tras una búsqueda exhaustiva, se identificó una solución en forma de una aplicación llamada DroidCam, la cual puede instalarse en el celular y tiene una librería compatible con Ubuntu y ROS. Los detalles de su instalación, conexión y funcionamiento se explican en el título Droidcam. No obstante, la limitación aquí es que la conexión debe realizarse a través de una red Wi-Fi, lo que puede resultar en una velocidad de transmisión de datos hasta diez veces más lenta que una conexión por cable. Esta ralentización puede generar problemas en la recepción de las imágenes y su procesamiento en tiempo real, lo que podría impactar negativamente en la ejecución del proyecto.

Para la ubicación del dispositivo Xiaomi, se ha desarrollado un soporte impreso en 3D que se integra a la parte superior del robot Kobuki mediante la utilización de tornillos. Este soporte ha sido diseñado específicamente para permitir la inserción del celular a una altura que sitúa la cámara principal del dispositivo a unos 40 centímetros aproximadamente del suelo.

Una consideración clave en este proceso ha sido la alineación precisa de la cámara principal del celular con el centro del robot. Esto se realiza con el propósito de evitar la pérdida de imágenes que podría surgir debido a movimientos bruscos del robot durante su desplazamiento. La Ilustración 68

proporciona una representación visual de la configuración del ensamblaje del celular en el robot Kobuki.

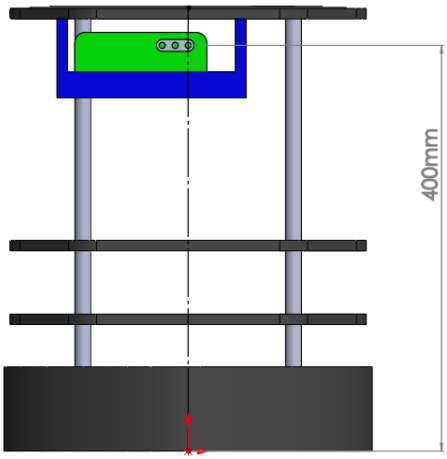


Ilustración 68-Ubicación Xiaomi en Kobuki

Elaboración propia

Este enfoque de ubicación del sensor ofrece diversas ventajas que deben ser destacadas. Por un lado, permite una posición elevada del sensor, lo que puede ser beneficioso para captar un campo de visión más amplio y obtener datos más completos. Además, la alineación cuidadosa con el centro del robot asegura que las imágenes capturadas sean consistentes y útiles para las tareas de mapeo y localización.

Sin embargo, no podemos pasar por alto las desventajas de esta configuración. La elevación del sensor puede introducir cierto grado de inestabilidad, especialmente en terrenos irregulares o en situaciones donde el robot experimenta movimientos bruscos. Además, el proceso de ensamblaje y alineación precisa puede ser laborioso y requerir ajustes periódicos para mantener su eficacia.

Para optimizar la localización del dispositivo Moto G5 en el contexto de este proyecto, es esencial identificar una ubicación estratégica que ofrezca un amplio campo de visión, al tiempo que se consideran las especificaciones de la cámara del dispositivo. Con este objetivo en mente, se ha planteado una solución consistente en la utilización de una viga de aluminio de sección transversal cuadrada de 5 cm x 5 cm y un espesor de 1 mm. Esta viga se instala en la parte superior de los paneles de la oficina, elevándose a una altura aproximada de 2.2 metros y atravesándola de lado a lado, asegurándola firmemente en los laterales para evitar movimientos no deseados.

Para adaptar el dispositivo Moto G5 a esta estructura, se ha diseñado e impreso en 3D una carcasa a la medida. Esta carcasa facilita la unión del dispositivo a la viga y garantiza que el celular se coloque en una posición en la que la cámara apunte perpendicularmente hacia el suelo. Esta configuración de la cámara se encuentra claramente ilustrada en la Ilustración 69.

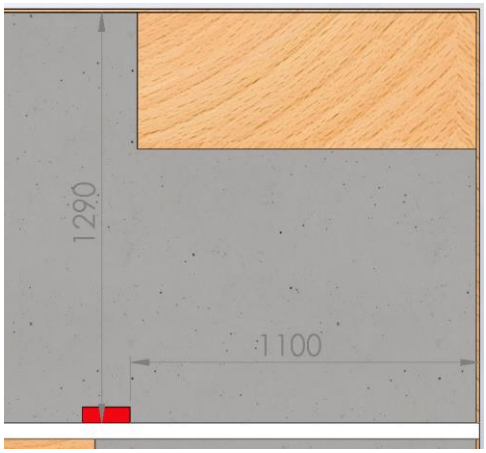


Ilustración 69-Ubicación de MotoG5 en oficina

Elaboración propia

Gracias a esta disposición y al ángulo de visión del sensor de la cámara, se logra un diámetro de visión de aproximadamente 2.7 metros, lo cual se aprecia en detalle en la Ilustración 70. Dentro de este rango, se llevará a cabo la recolección de datos para el proyecto, lo que resulta en un espacio útil para el desplazamiento del robot de aproximadamente 4.27 metros cuadrados.

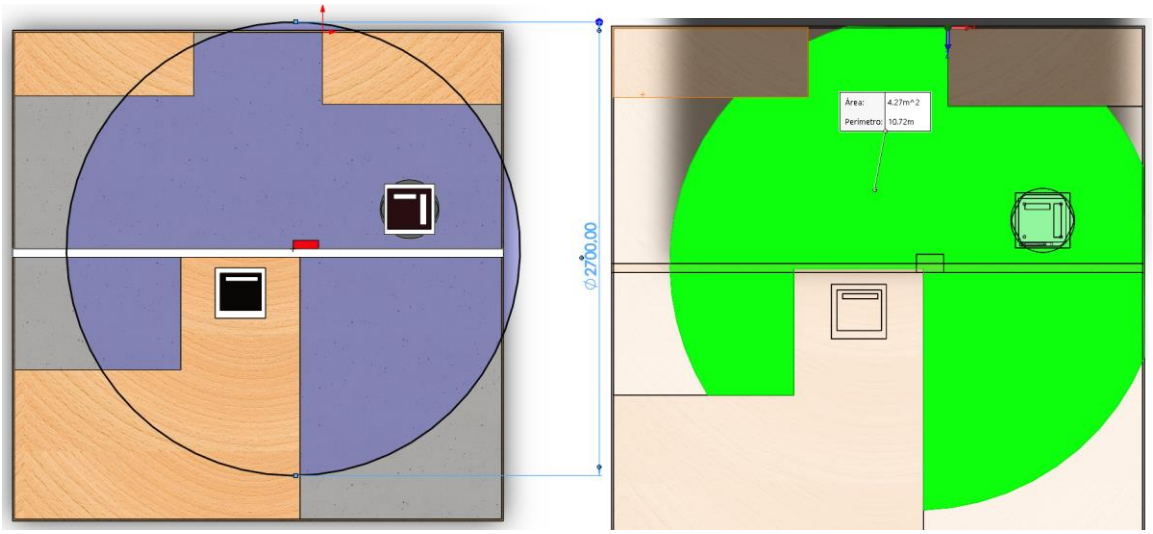


Ilustración 70-Área de visualización MotoG5

Elaboración propia

Esta elección de ubicación y configuración presenta ventajas significativas, como un campo de visión extendido y una distribución óptima para la captura de datos.

5.6 ROBOT A IMPLEMENTAR

En el contexto del presente proyecto de investigación de maestría, se hace uso de un recurso invaluable proporcionado por la Universidad EAFIT: un robot del tipo TurtleBot 2 de la reconocida compañía Kobuki. Es relevante destacar que el TurtleBot 2 es considerado como uno de los robots de código abierto más asequibles y populares en todo el mundo, especialmente destinado a fines

educativos y de investigación. Esta elección estratégica se traduce en múltiples ventajas que influyen directamente en el éxito de esta investigación.

Ilustración 71-Vistas principales Kobuki

Tomada de: Clearpath Robotics

Un elemento distintivo de este robot es su estrecha afinidad con el entorno de trabajo. Fue concebido en estrecha colaboración con los mismos desarrolladores originales de ROS en su versión "Melodic". En el paquete de envío del TurtleBot 2, se suministra un completo instructivo que detalla la creación y el uso de entornos de trabajo, paquetes e instalaciones. Además, proporciona una guía paso a paso para la configuración y ejecución del sistema. Esta relación es de suma importancia, ya que ROS la plataforma que servirá como base para la implementación de los algoritmos y la recopilación de datos, como se detalla en profundidad en la sección titulada Configuración de ROS.

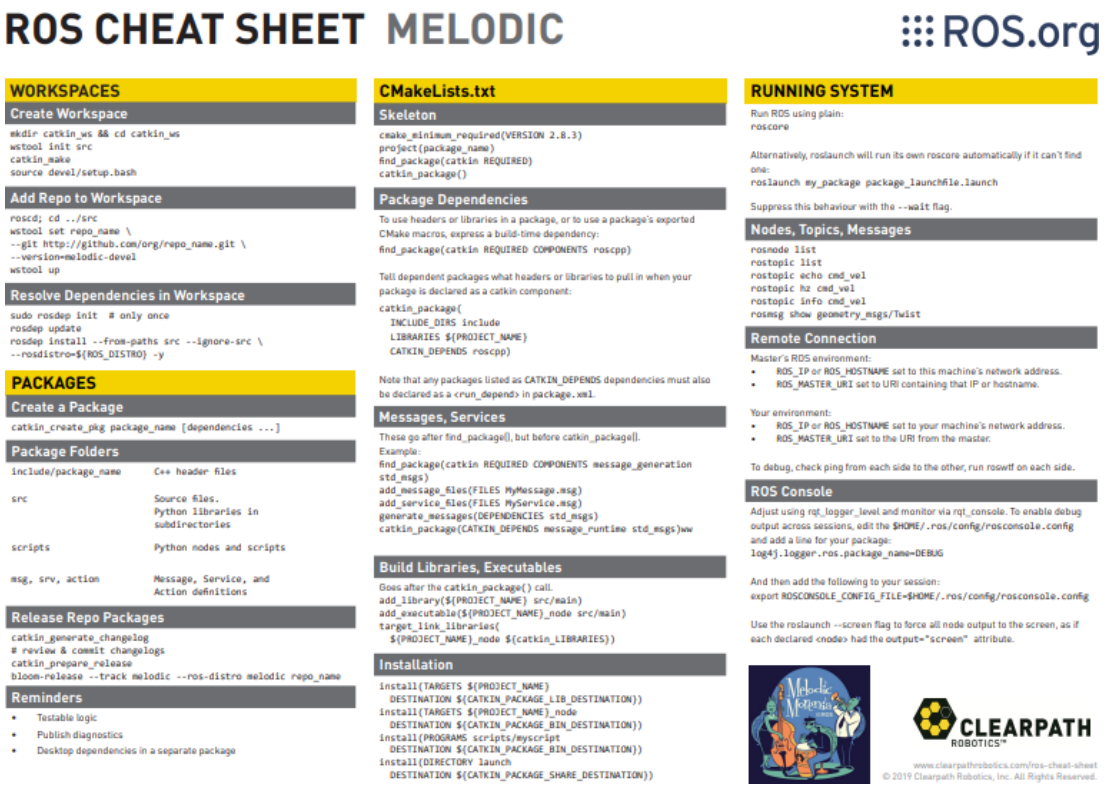


Ilustración 72-Ficha técnica Kobuki

Tomada de: Clearpath Robotics

En términos de sus especificaciones técnicas, el TurtleBot 2 presenta dimensiones de 354 mm de diámetro, una altura de 420 mm y un peso de 6.3 kg (sin contar sensores externos). Su velocidad

máxima de 0.65 m/s y una velocidad de rotación de 180°/s le otorgan la agilidad necesaria para desenvolverse en el entorno de oficina, tal como se detalla en la sección Lugar de experimento. La autonomía de la batería, con capacidad de 4400 miliamperios, oscila entre 0.8 y 4 horas sin necesidad de recarga, lo que proporciona un amplio margen de tiempo para la realización de múltiples experimentos, cada uno de 5 minutos de duración como mínimo.

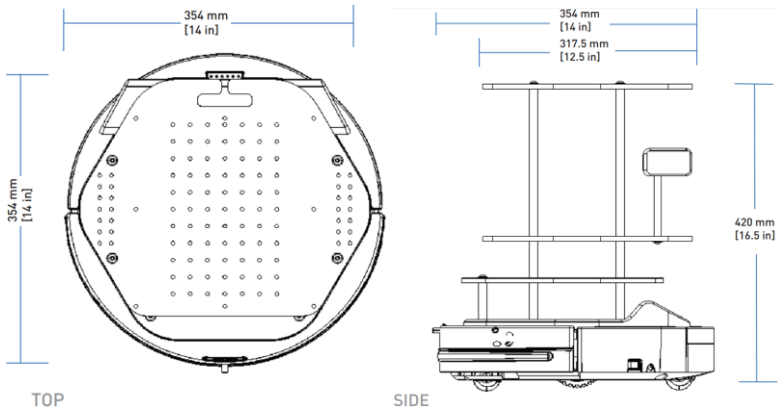


Ilustración 73-Dimensiones Kobuki

Tomada de: Clearpath Robotics

La unidad central de procesamiento del TurtleBot 2 está equipada con una memoria RAM de 4GB, un procesador Intel Core i3-4010U y una memoria interna de 500GB (Para más información del Turtlebot2 de Kobuki visitar la URL: https://clearpathrobotics.com/turtlebot-2-open-source-robot/). Además, cuenta con puertos USB que desempeñan un papel fundamental en la instalación de drivers para los servomotores responsables del movimiento y rotación del robot, así como en la integración de paquetes de ROS que permiten su control desde el programa principal, estableciendo una conexión eficiente mediante estos puertos USB.

Las características previamente descritas se traducen en un enfoque más robusto y exitoso para alcanzar los objetivos de esta investigación.

5.7 LIBRERÍA ARUCO

En esta sección, detallaremos las características fundamentales de la librería de ArUco en ROS Melodic, centrándonos en la información que proporciona esta librería y su relevancia para el algoritmo que se desarrollará, como se expone en la sección ArUco mapping implementado en ROS.

Cuando se ejecuta cualquiera de los archivos con extensión ".launch" (previamente explicados en la sección Entorno de trabajo del proyecto), se activa la librería de ArUco en ROS Melodic. La librería, al ser activada, publica un nodo denominado "MarkerArray". Es posible visualizar todos los nodos activos en ROS mediante el comando en la consola (asegurándose de tener el paquete activo):

```
Rostopic list
```

En nuestro caso muestra el nodo con el siguiente nombre “ArUco_msgs/MarkerArray”, cómo se muestra en la Ilustración 74 de la consola:

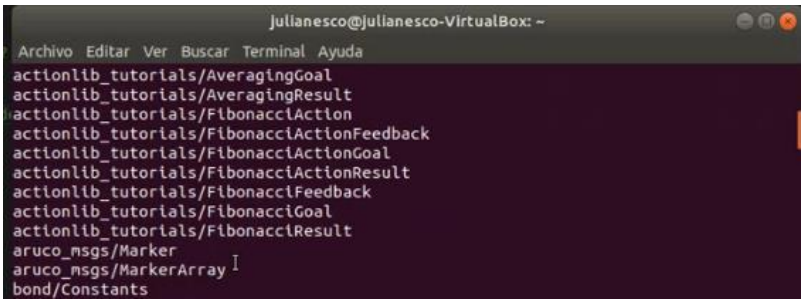


Ilustración 74-Nodos publicados por ROS

Elaboración propia

No se emplea el nodo "ArUco_msgs/Marker" debido a que solo muestra información del marcador en la posición 1 de la lista, sin considerar la posible detección de múltiples marcadores en la imagen.

Para mostrar la información que contiene cada nodo, se puede utilizar el comando en la terminal:

```
Rosmsg show nombre_del_nodo
```

En el caso de “ArUco_msgs/MarkerArray”, se publica una lista que contiene la información de cada marcador que detectado en la imagen. La información que publica es la siguiente para cada marcador encontrado:

```
aruco_msgs/Marker[] markers
  std_msgs/Header header
    uint32 seq
    time stamp
    string frame_id
  uint32 id
  geometry_msgs/PoseWithCovariance pose
    geometry_msgs/Pose pose
      geometry_msgs/Point position
        float64 x
        float64 y
        float64 z
      geometry_msgs/Quaternion orientation
        float64 x
        float64 y
        float64 z
        float64 w
    float64[36] covariance
  float64 confidence
```

Ilustración 75-Estructura nodo markers en ROS

Elaboración propia

Para el proyecto de investigación, se usarán los valores de Stamp (para conocer el tiempo en que se detectó el marcador), ID (describe el ID del marcador del que se está mostrando la información) y todos los valores de “geomatry_msgs/pose/point” (valores de posición X, Y y Z del marcador) y “geomatry_msgs/pose/Orientación” (valores de orientación X, Y, Z y W) para realizar análisis y transformaciones subsiguientes.

Es relevante destacar que la documentación de la librería especifica que los valores de posición se refieren a la posición del marcador con respecto al sistema de coordenadas de la cámara [35], y su unidad de medida es en metros, cómo se muestra en la Ilustración 76.

Estimating the pose of markers

We will assume at this point that no ambiguity is present in the captured images.

Using the following code, you'll detect the markers in a image along with the pose of each marker wrt to the camera.

```
#include <opencv2/highgui.hpp>
#include "aruco.h"
using namespace std;
int main(int argc,char **argv)
{
   cv::Mat im=cv::imread(argv[1]);
   aruco::CameraParameters camera;
   camera.readFromXMLFile(argv[2]);
   aruco::MarkerDetector Detector;

Detector.setDictionary("ARUCO_MIP_36h12");
```

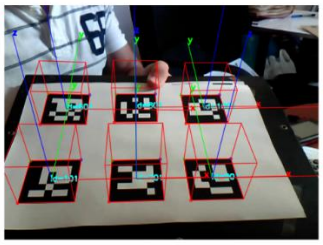


Ilustración 76-Referencia documentación librería ArUco

Tomada de: Mapping and localization from planar markers documentation

Para visualizar la imagen transmitida desde el dispositivo móvil mediante Droidcam y con la librería de ArUco implementada, se puede emplear el siguiente comando en la consola:

```
Rosrun rqt_gui rqt_gui
```

El comando rosrun rqt_gui rqt_gui se utiliza para ejecutar la interfaz gráfica de usuario (GUI) de RQT (ROS Qt GUI Toolkit) en ROS. RQT es una herramienta que proporciona una interfaz gráfica para la visualización y la interacción con diversas herramientas y plugins de ROS.

Una vez que la interfaz gráfica está activa, se debe cambiar el nodo que publica el archivo ".launch" desde el Dropdown donde está ubicado el mouse en la Ilustración 77.

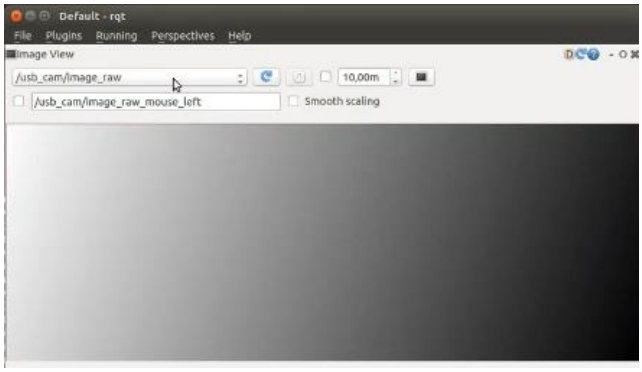


Ilustración 77-RQT seleccionar nodo

Elaboración propia

Cuando se visualiza la cámara con la librería aplicada, los marcadores tipo ArUco se destacan en el marco de la Ilustración 78 con un borde azul, mostrando el sistema de coordenadas en el origen del marcador y su ID. La representación visual del eje Y en verde, el eje X en azul, el eje Z en rojo, y el ID en amarillo se presenta en la Ilustración 78 correspondiente.

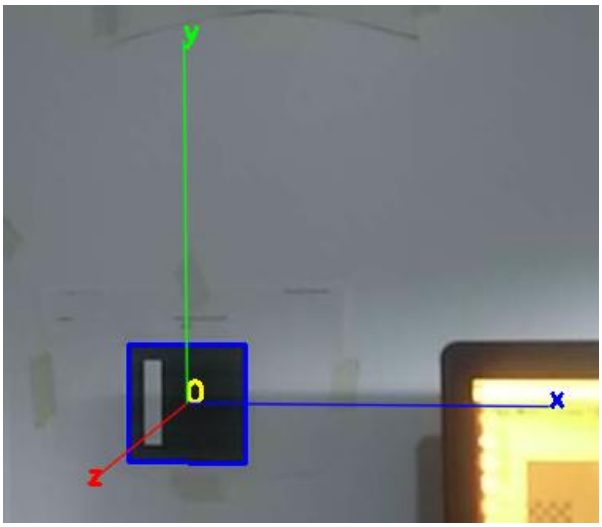


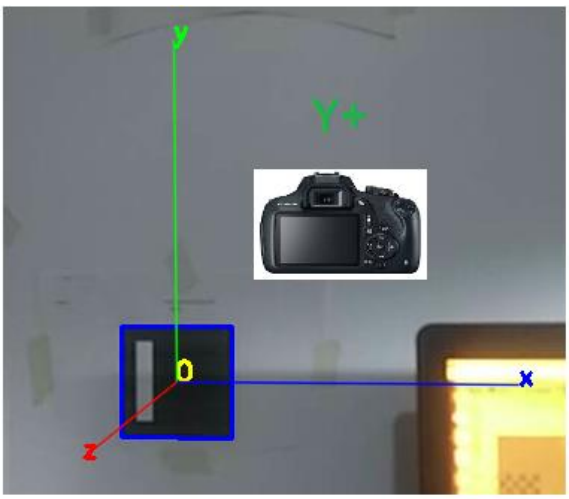
Ilustración 78-Sistema coordenado en marcador

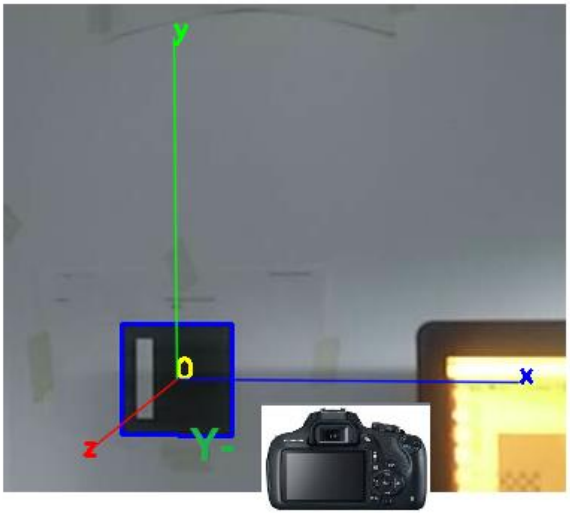
Elaboración propia

Al comprender el sistema de coordenadas del marcador, se lleva a cabo un experimento para validar los signos y datos proporcionados por "ArUco_msgs/MarkerArray" en función de la posición de la cámara respecto al sistema de coordenadas del marcador.

El experimento consiste en posicionar la cámara en el eje Y positivo del sistema de coordenadas del marcador, verificar el signo de los datos publicados por el nodo, y repetir este proceso para el eje Y negativo. Este procedimiento se replica para los ejes X y Z. Este análisis permite comprender y determinar la orientación del sistema de coordenadas de la cámara según la implementación de la librería de ArUco.

A continuación, se presentan las representaciones visuales de la posición de la cámara con respecto al sistema coordenado del marcador (los valores de Y+, Y-, X+ y X- son los signos de los resultados):





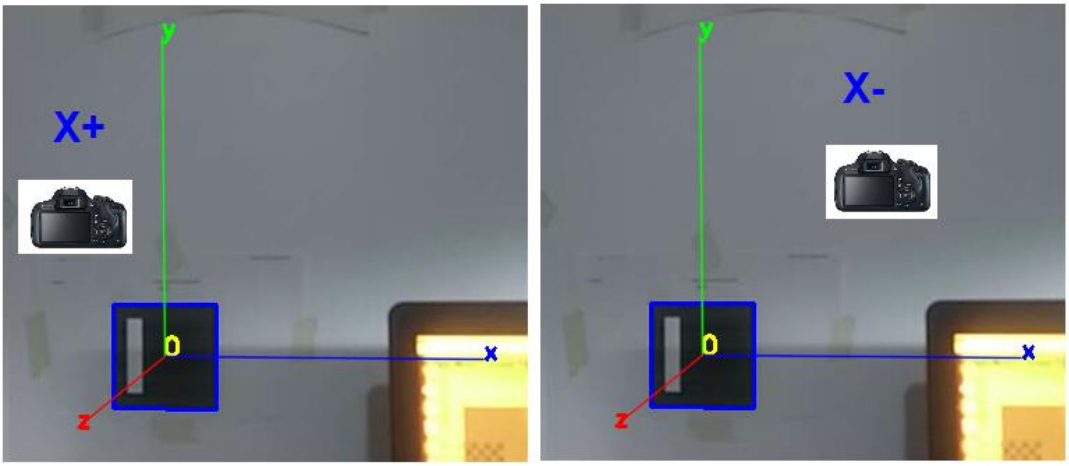


Ilustración 79-Ubicación de la cámara y signo de resultado ArUco

Elaboración propia

Los resultados obtenidos son los siguientes:

- Z: Aumenta cuando la cámara se aleja y disminuye cuando se acerca, siempre manteniendo una unidad de medida positiva, independientemente de la ubicación de la cámara.
- Y: Cuando la cámara está debajo del eje X (-Y del sistema de coordenadas del marcador), se obtiene un valor negativo (mayor negatividad cuanto más abajo). En contraste, cuando la cámara está sobre el eje X (+Y del sistema de coordenadas del marcador), se obtiene un valor positivo (mayor positividad cuanto más arriba).
- X: Cuando la cámara está a la izquierda del eje Y (-X del sistema de coordenadas del marcador), se obtiene un valor positivo (mayor positividad cuanto más a la izquierda). Por otro lado, cuando la cámara está a la derecha del eje Y (+X del sistema de coordenadas del marcador), se obtiene un valor negativo (mayor negatividad cuanto más a la derecha).

Con los resultados del experimento, se define el eje coordenado de la cámara, como se muestra en la Ilustración 80 (eje coordenado con colores de ROS):

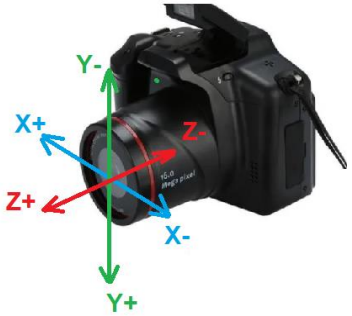


Ilustración 80-Sistema coordenado de cámara

Elaboración propia

Es importante mencionar que, antes de realizar cualquier transformación, se observa que los valores publicados para Y deben invertirse para ajustar el sistema de coordenadas de la cámara a la ubicación real del marcador. Este ajuste es crucial para garantizar la consistencia en las interpretaciones de las coordenadas y la precisión en las futuras transformaciones.

5.8 ARUCOS A IMPLEMENTAR EN EL ENTORNO

La selección de los marcadores ArUco para la implementación en el proyecto se basa en la elección específica de los IDs 1, 3, 4, 5, 7, 8, 9, 11, 13, 14, 15, 16, 18, 19, 20, 22, 23, 24, 25, 26, 27, 29, 30, 32, 34, 35, 36, 37, 38, 60, 61, 62, 63, 65, 67, 69, 71, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83 y 85, totalizando 49 marcadores que fueron estratégicamente distribuidos alrededor de la oficina.

La selección de estos marcadores se llevó a cabo mediante una elección aleatoria de los IDs dentro del rango de 1 a 100. Estos fueron posicionados a una altura aproximada de 40 cm desde el piso de la oficina hasta el borde inferior del marcador, como se visualiza en la Ilustración 81.

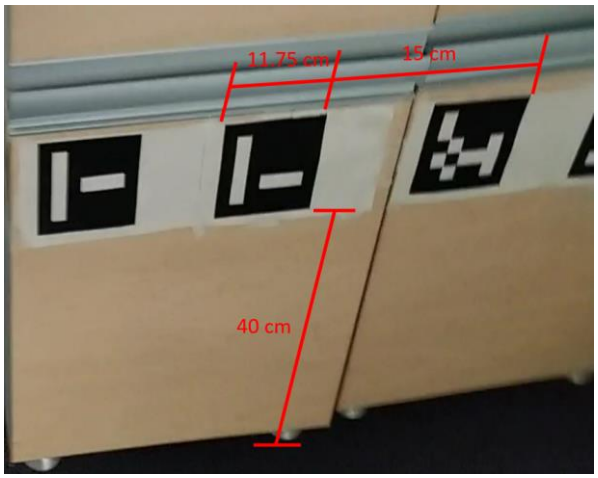


Ilustración 81-Ubicación de marcadores

Elaboración propia

La disposición espacial de los marcadores se diseñó con una separación entre centros de aproximadamente 15 cm, abarcando todo el perímetro útil de la oficina. Para adherirlos a las paredes de la oficina, se utilizaron cintas de enmascarar, siguiendo una distribución aleatoria que se representa en la Ilustración 82.

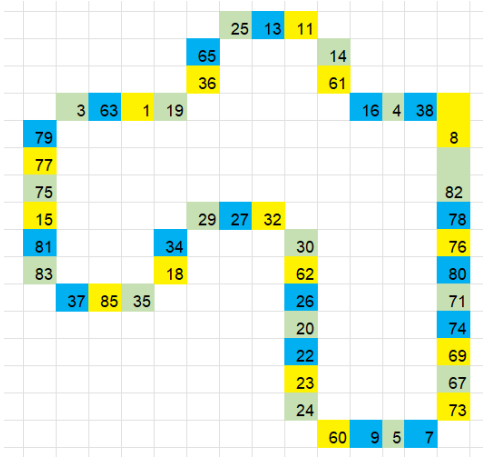


Ilustración 82-Ubicación de ID en oficina

Elaboración propia

Todos los marcadores comparten la misma medida de 11.75 cm y presentan un borde blanco de al menos 1 cm de ancho. Estos fueron impresos utilizando una impresora Multifuncional HP 1200w Neverstop Laser Blanco, empleando papel reciclado de la marca Reprograf.

Los Marcadores ArUco se imprimieron y generaron por medio de la página https: // chev.me / ArUcogen /. Se debe tener en cuenta que en “Dictionay” se debe seleccionar “Original ArUco”, en “Marker ID” se debe poner el ID a imprimir y en “Markersize,mm” se debe poner el valor de ancho y largo del ArUco.

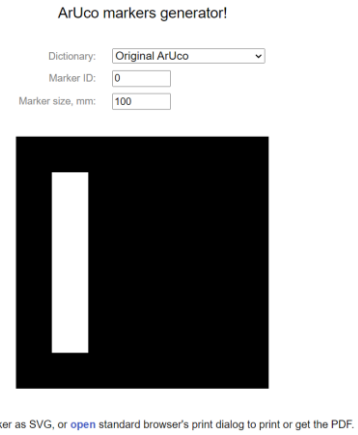


Ilustración 83-Librería de descarga de marcadores ArUco

Tomada de: https: // chev.me / ArUcogen

La elección de la medida de 11.75 cm para los marcadores se fundamenta en el estudio Experimental Comparison of Fiducial Markers for Pose Estimation . En este experimento, se propone una relación entre la distancia de la cámara y la medida del marcador, que oscila entre 4.2 y 11. Considerando esta relación, la medida seleccionada de 11.75 cm proporciona un rango de trabajo que abarca desde 50 cm hasta 1.3 m. En el contexto de la oficina, esta elección asegura que en ningún punto el robot se encuentre a una distancia superior a 1.3 m de un marcador, optimizando así la eficacia del sistema en este entorno específico.

5.9 LUGAR DE EXPERIMENTO

El espacio designado para la ejecución del proyecto de investigación se ubica en la oficina 24, situada en el bloque 19, piso 2 de la Universidad EAFIT. Cabe resaltar la generosidad de la institución al ceder amablemente este espacio para la realización de la investigación. La configuración de la oficina se caracteriza por la presencia de módulos desarmables fabricados en aluminio, los cuales están provistos de vidrio opaco. La entrada cuenta con una puerta en vidrio.

En cuanto a la iluminación, la oficina cuenta con cuatro luces LED en el techo, cada una emitiendo 900 lúmenes, proporcionando así una iluminación adecuada para el desarrollo de las actividades relacionadas con el proyecto.

La disposición del mobiliario incluye dos estanterías tipo biblioteca, siendo ubicadas en las esquinas superior izquierda y derecha que modifican el espacio para tener más obstáculos y una geometría no tan cuadrada. También cuenta con una mesa en forma de L, ubicada en el centro de la oficina, cambiando aún más la geometría disponible para el desplazamiento del robot y asemejándose al entorno real en el que tendrá que trabajar, teniendo así un área libre de 5.5 m2 aproximadamente y un perímetro para la ubicación de los marcadores de aproximadamente 13 m, información que se evidencia en la Ilustración 84.

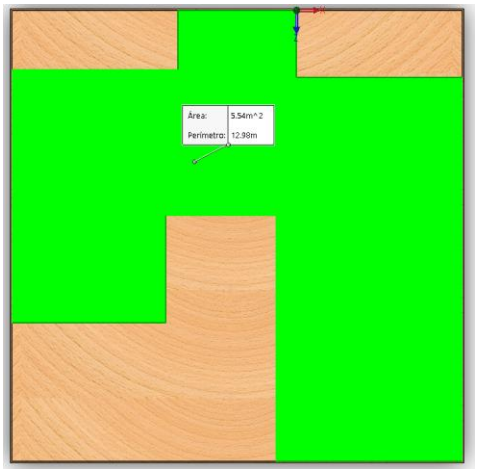


Ilustración 84-Área disponible en oficina

Elaboración propia

Asimismo, el piso de la oficina está revestido con alfombra, lo que puede perjudicar el correcto movimiento del Kobuki, dando vibración en el movimiento por el desnivel que puede dar la textura de la alfombra.

Resulta pertinente destacar que la disposición y medidas exactas de la oficina se encuentran detalladas en la Ilustración 85 (altura de 2.2 mts), proporcionando una representación en vista superior precisa de la distribución del espacio de trabajo. Este entorno brinda un escenario propicio para llevar a cabo investigaciones de manera efectiva, ofreciendo obstáculos, rincones y pasillos con medidas similares a las que tendría el robot en su entorno real.

Ilustración 85-Dimensiones reales de oficina

Elaboración propia

5.10 DROIDCAM

En la sección dedicada a la utilización de sensores definidos como cámaras, como se expone en el apartado Sensores a utilizar, es indispensable acceder a la información captada por las cámaras de

los dispositivos móviles desde ROS sin depender de una conexión por cable. Para cumplir con este requisito, se recurre a la aplicación DroidCam, una herramienta que posibilita a los usuarios convertir sus dispositivos Android en cámaras web para sus ordenadores. Esta aplicación presenta la ventaja de contar con librerías diseñadas para integrarse con entornos Linux y con ROS, lo que la convierte en una elección ideal para la implementación del proyecto en cuestión.

La instalación de DroidCam se lleva a cabo a través de la Play Store del celular. Una vez instalada y abierta, la aplicación proporciona una dirección IP web, a través de la cual se puede acceder a la información captada por la cámara del dispositivo móvil. La aplicación siempre debe estar abierta en el celular cada vez que se vaya a ejecutar algún archivo ".launch" que necesite de la Ilustración 86 proporcionada por el celular.

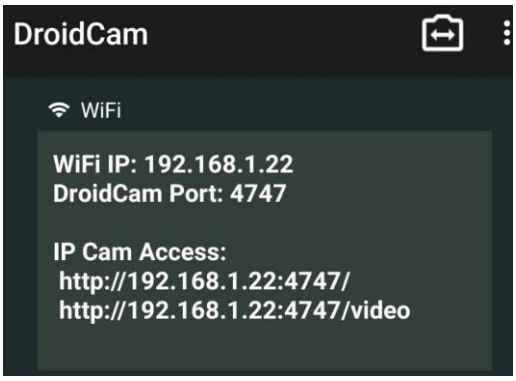


Ilustración 86-Interfaz de Droidcam

Elaboración propia

Para visualizar la información capturada por la cámara del celular y transmitida por DroidCam en un entorno Ubuntu, se requiere la instalación del paquete ROS video_stream_opencv de OpenCV. Este paquete incluye un nodo destinado a publicar un flujo de video en temas de imágenes de ROS desde una IP. Detalles adicionales sobre este paquete pueden encontrarse en la URL: https://wiki.ros.org/video_stream_opencv. Para instalar el paquete en Ubuntu, se ejecuta el siguiente comando en la consola:

```
sudo apt-get install ros-melodic-video-stream-opencv
```

Con el objetivo de simplificar el lanzamiento de la cámara con ROS, se puede emplear un archivo de lanzamiento (launch file). El siguiente ejemplo ilustra un archivo de lanzamiento que ejecuta el paquete video_stream_opencv, publica un nodo de ROS y es adecuado para su utilización en el programa:

```
<launch>
  <node pkg="video_stream_opencv" type="video_stream" name="camera_r">
    <param name="video_stream_provider" value="http://192.168.117.32:4747/video"/>
    <param name="fps" value="30.0" />
    <param name="camera_info_url" value="file://$(find ArUco_melodic_mapping)/src/MotoG5/ost.yaml" />
    <remap from="/image_raw" to="/camera_image_r" />
  </node>
```

```
</launch>
```

Explicación del archivo para comprender qué hace y qué publica:

<launch>:

Indica el inicio del archivo de lanzamiento.

<node pkg="video_stream_opencv" type="video_stream" name="camera_r">:

Este bloque lanza un nodo del paquete video_stream_opencv con el tipo video_stream y le asigna el nombre "camera_r".

<param name="video_stream_provider" value="http://192.168.117.32:4747/video"/>:

Este parámetro especifica la fuente del flujo de video. En este caso, la fuente es una URL que apunta a un video en streaming en la dirección http://192.168.117.32:4747/video. Esta dirección debe ser la que publica Droidcam.

<param name="fps" value="30.0" />:

Este parámetro establece la tasa de fotogramas por segundo del video en 30.0.

<param name = "camera_info_url" value="file://$ (find ArUco melodic mapping) / src / MotoG5 / ost.yaml" />:

Este parámetro especifica la URL del archivo que contiene la información de la cámara. En este caso, la URL es de un archivo YAML llamado "ost.yaml" ubicado en la ruta $(find ArUco_melodic_mapping)/src/MotoG5/. Estos parámetros ayudan a video_stream_opencv a rectificar la imagen proporcionada por Droidcam según los parámetros de calibración de la cámara. La calibración de la cámara se explica en la sección Calibración de cámaras usando ROS.

<remap from="/image_raw" to="/camera_image_r" />:

Este comando realiza un remapeo del nombre del tema. Cambia el nombre del tema de salida de "/image_raw" a "/camera_image_r". En este caso "/camera_image_r" es donde video_stream_opencv publica la información captada por la cámara del celular y publicada por Droidcam y es donde nos podemos conectar para utilizar el video y procesarlo como aplicarle la librería de ArUco.

</node>:

Cierra la etiqueta del nodo.

</launch>:

Indica el final del archivo de lanzamiento.

Este archivo ".launch" se implementa en todos los paquetes del proyecto en los que se tenga que captar información de las cámaras de los celulares.

5.11 CALIBRACIÓN DE CÁMARAS USANDO ROS

La calibración de la cámara constituye un paso crítico al emplear una cámara como sensor en sistemas basados en ROS, y esto se debe a dos razones fundamentales en el contexto de nuestro proyecto:

Precisión de la medición: La calibración de la cámara ayuda a corregir las distorsiones geométricas y ópticas presentes en las imágenes capturadas por la cámara. Esto asegura que las mediciones realizadas en el mundo real a partir de las imágenes sean precisas y confiables. Sin calibración, las mediciones pueden contener errores debido a distorsiones, lo que afectaría la precisión de las tareas que dependen de la información visual.

Mejora de la calidad de las imágenes: La calibración también puede mejorar la calidad general de las imágenes al corregir problemas como la distorsión radial y tangencial. Esto es especialmente importante cuando se utilizan cámaras para aplicaciones de visión por computadora, donde la precisión y la calidad de la imagen son fundamentales.

Conscientes de la importancia de la calibración de cámaras, los desarrolladores de ROS han creado una librería que utiliza la tecnología de OpenCV. Esta librería extrae características clave, como el ancho y alto en píxeles de la imagen, así como la matriz de distorsión, rectificación y proyección de la cámara.

A continuación, se detalla el proceso paso a paso para instalar estas librerías y calibrar la cámara:

5.11.1 Instalación de Paquetes Necesarios

Se comienza instalando el paquete "usb_cam" de ROS y "uvcdynctrl" para desactivar el enfoque automático en la versión Melodic de ROS. Esto se logra ejecutando el siguiente script en la consola:

```
sudo apt-get install ros-melodic-usb-cam uvcdynctrl
```

5.11.2 Activación del Entorno ROS

Posteriormente, se activa el entorno de ROS con el siguiente script:

```
roscore
```

5.11.3 Activación de Droidcam y Identificación de Dispositivo

Se activa el archivo Launch de Droidcam para acceder a la imagen del dispositivo. Luego, se identifica el dispositivo correspondiente a la cámara del celular ejecutando el siguiente script:

```
Ls -ltrh /dev/video*
```

Esta acción muestra todos los dispositivos identificados para video por Ubuntu, cómo se muestra en la Ilustración 87:

Ilustración 87-Dispositivos de video vinculados

Elaboración propia

Uno de estos dispositivos corresponde a la cámara del celular. Por prueba y error se debe identificar cual es el de la cámara del celular, reemplazando la parte del código de “/video#” de los siguientes scripts.

Paso siguiente, desactivar el enfoque automático de la cámara (si la cámara admite el enfoque automático) por medio del script:

```
uvcdynctrl --device=/dev/video# --set='Focus, Auto' 0
```

Verificar que el enfoque automático esté desactivado:

```
uvcdynctrl --device=/dev/video# --get='Focus, Auto'
```

5.11.4 Conexión con "camera_calibration"

Con el nodo que publica las imágenes de la cámara, se conecta "camera_calibration" utilizando el siguiente script, ajustando los parámetros de “image:=” y “camera:=” según el archivo .launch de Droidcam. También se debe modificar el parámetro de 0.02517 por el valor real de la medida en metros de los cuadros luego de ser impreso (para imprimir el ajedrez, ir a la URL https://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration?action=AttachFile&do=view&target=check-108.pdf):

```
Rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.02517 image:=/camera_r/camera_image_r camera:=/camera_r --no-service-check
```

Al ejecutar este código, aparece la siguiente ventana con la imagen de la cámara del celular, tres botones (*Calibrate, Save y Commit*) y 4 barras (X, Y, Size y Skew).

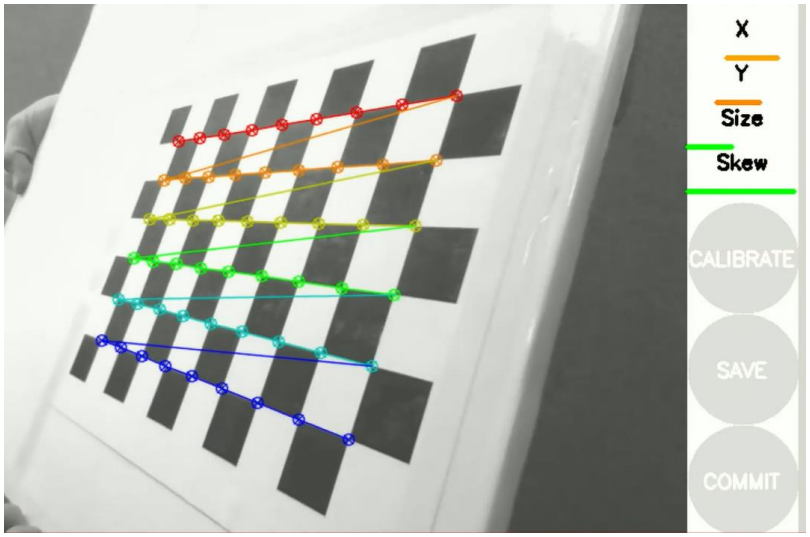


Ilustración 88-Visualización de calibración de cámara

Elaboración propia

5.11.5 Calibración de la Cámara

Es crucial asegurar que la imagen del ajedrez esté en una superficie plana y fija. Se debe mover la cámara en diferentes direcciones hasta que las 4 barras de ajuste queden verdes. Después de obtener suficientes imágenes, se hace clic en "Calibrate" y luego en "Save". Si se hace clic en "Commit", se copian los datos de calibración a:

```
/home/<nombre_de_usuario>/.ros/camera_info/ost.yaml
```

Se puede extraer el archivo ost.yaml y guardarlo en la ruta deseada. Al elegir una ruta, es esencial actualizar los archivos ".launch" con la ubicación del archivo de calibración. En nuestro caso, para el celular MotoG5, se guardó en el Paquete ArUco_melodic_mapping, en la ruta interna /src/MotoG5/ost.yaml.

5.12 ARUCO MAPPING IMPLEMENTADO EN ROS

En el marco de este proyecto de investigación, el primer desafío radica en la implementación efectiva de la librería ArUco en ROS Melodic con el propósito de llevar a cabo SLAM y generar un mapa detallado del entorno de trabajo. Para abordar esta tarea, se ha desarrollado el Paquete ArUco_melodic_mapping. Esta sección de la tesis se centrará en proporcionar una comprensión detallada de la implementación de dicho paquete, así como de los códigos que se han diseñado como parte integral de este proceso.

5.12.1 Documentos

En cuanto a la exploración de los documentos presentes en el Paquete ArUco_melodic_mapping, previamente hemos abordado los archivos ejecutables en la sección Entorno de trabajo del proyecto. Ahora, nuestra atención se dirigirá hacia la carpeta "SRC", donde se encuentran archivos cruciales para el funcionamiento y la eficacia del proyecto.

A continuación, se presenta una descripción detallada de las carpetas contenidas en "SRC" y sus respectivos archivos:

5.12.1.1 mapa_oficina

Esta carpeta contiene los archivos del mapa creado obtenido en A_laser en formato JSON y PGM ("*Portable Gray Map*", es un formato de archivo de imagen que se utiliza para almacenar imágenes en escala de grises) y las características del mapa en formato YAML ("YAML *Ain't a Markup Language*," es un formato de serialización de datos que utiliza una sintaxis legible por humanos. YAML se utiliza comúnmente para la configuración de aplicaciones, archivos de manifiesto y datos estructurados.). También, contiene el archivo en JSON markers_world_pos el cual tiene la información de ID, posición y orientación para cada marcador con respecto al sistema coordenado global.

5.12.1.2 MotoG5

Esta carpeta contiene la serie de imágenes capturadas al momento de hacer la calibración de la cámara para el celular MotoG5 y su archivo "Ost" en formato YAML, que almacena información detallada sobre la calibración.

5.12.1.3 XiaomiNote8

Esta carpeta contiene la serie de imágenes capturadas al momento de hacer la calibración de la cámara para el celular Xiaomi Note 8 y su archivo "Ost" en formato YAML, que almacena información detallada sobre la calibración.

5.12.2 Lista de archivos JSON

5.12.2.1 initial_marker_pos
Es un archivo que contiene el ID, posición y orientación del marcador definido como estático para la ubicación del Kobuki en el *groundtruth*. Este archivo es creado manualmente.

5.12.2.2 map_occupancy_grid
Es el mismo archivo JSON de la carpeta mapa_oficina que contiene la información de cuales recuadros de la malla del mapa son grises, negros o blancos. Está copiado porque el que está en la carpeta es meramente informativo y este se utiliza en el código.

5.12.2.3 markers_world_pos
Es el mismo archivo JSON de la carpeta mapa_oficina que contiene la información de ID, posición y orientación de los marcadores con respecto al sistema global de coordenadas. Está copiado porque el que está en la carpeta es meramente informativo y este se utiliza en el código.

5.12.3 Lista de archivos Python

5.12.3.1 a_mapping
Este código en Python transforma la información del nodo de ArUco ROS Melodic en las posiciones y orientación de los marcadores ubicados en la oficina con respecto al sistema coordenado global y lo publica en formato JSON con el nombre markers_world_pos. El código se explica a detalle en A_mapping.

5.12.3.2 a_laser
Este código en Python genera el mapa en malla para ROS, identificando los marcadores que aparecen en la imagen, reconoce su posición con respecto al sistema coordenado global y asigna malla ocupada en la ubicación del marcador y entre el marcador y la cámara asigna malla libre. Guarda la malla en formato JSON con el nombre map_occupancy_grid. El código se explica a detalle en A_laser.

5.12.3.3 functions_to_slam
Es un Código en python que se utiliza como librería, con las funciones principales para el funcionamiento de a_laser y a_mapping para realizar el SLAM. Este código se explica a detalle en Funtions_to_slam.

5.12.3.4 kobuki_real_world_pos
Este Código en Python captura la información de la ubicación X e Y correspondiente al *groundtruth* del Kobuki. Genera un código en JSON con nombre “real”, con la ubicación y el tiempo en el que se capturó la información. Este código se explica a detalle en kobuki_real_world_pos.

5.12.3.5 kobuki_world_pos
Este Código en Python captura la información de la ubicación X e Y correspondiente a la calculada por el Kobuki. Genera un código en JSON con nombre “calculate”, con la ubicación y el tiempo en el que se capturó la información. Este código se explica a detalle en kobuki_world_pos.

5.12.4 Lista de archivos Rviz

Ambos archivos, "ejercicio_4.rviz" y "ejercicio_real.rviz", se utilizan para especificar las características a visualizar en Rviz. La distinción entre "ejercicio_real.rviz" y "ejercicio_4.rviz" radica en el uso de los tópicos "/ArUco_r/result" y "/ArUco/result", respectivamente. Kobuki_real.launch utiliza "ejercicio_real.rviz", mientras que Kobuki.launch y ArUco_melodic_mapping.launch utilizan "ejercicio_4.rviz".

5.13 FUNCIONES Y CÓDIGOS

En esta sección explicaremos cada uno de los códigos cómo están diseñados, que funcionalidades tienen, su matemática, que datos de entrada y de salida tienen y cómo transforman la información.

5.13.1 Funtions_to_slam

Empezamos por el código del archivo en Python Funtions_to_slam ya que este es el que se utiliza como librería para todos los otros códigos de A_laser, A_mapping, kobuki_real_world_pos y kobuki_world_pos.

Esta librería contiene 3 clases PositionToCalc, GridMapping y FuncToMapping.

5.13.1.1 PositionToCalc

Esta clase contiene las funciones necesarias para calculas las posiciones del Kobuki con respecto al sistema coordinado global, sea para el *groundtruth* como para el SLAM. El desglose de funciones para esta clase se presenta a continuación:

5.13.1.1.1 to_the_beginning

Entradas:

Posición actual en X e Y, actual_pos.

Posición inicial en X e Y, first_pos.

Transformaciones:

Al usarse para el *groundtruth* como para SLAM hay que buscar un punto 0 de ubicación, así las rutas de movimiento pueden compararse entre ellas, cómo se muestra en la Ilustración 89:

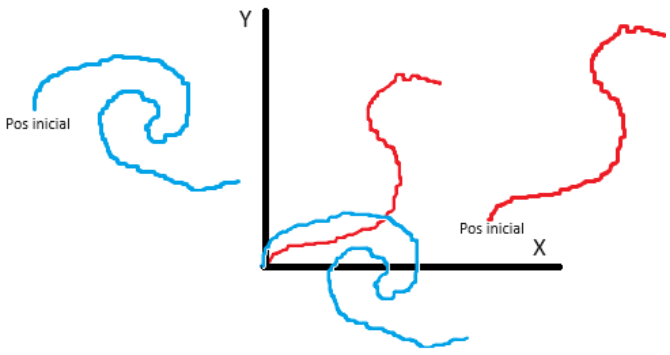


Ilustración 89-Traslación de trayectorias

Elaboración propia

Para realizar esta traslación, se resta a la posición X actual e Y actual, la correspondiente posición X inicial e Y inicial.

Salidas:

Posición punto 0 en X e Y, pos.

5.13.1.1.2 pos_in_time

Entradas:

Posición actual X e Y, actual_pos.

Nombre del archivo, name.

Tiempo en el que fue capturado el dato, time.

Posición inicial X e Y, first_pos.

Directorio donde guardar el archivo JSON, dir.

Transformaciones:

Aplica to_the_beginning y guarda un archive JSON que contiene la información del tiempo en el que se tomó el dato y el resultado de to_the_beginning con el nombre proporcionado en la ruta seleccionada.

Salidas:

Archivo en JSON con nombre y ubicación definidos en la función.

5.13.1.2 to_marker_type

Entradas:

Listado de información de marcadores en formato del proyecto, array.

Transformaciones:

Para guardar la información de la librería de ArUco en ROS Melodic, se define el siguiente formato, una lista con la siguiente información:

```
("id",-1),
("posx",0.0),
("posy",0.0),
("posz",0.0),
("quatx",0.0),
("quatx",0.0),
("quaty",0.0),
("quatz",0.0),
("quatw",0.0),
("visible", False),
("parent_frame", " "),
("child_frame", " "),
("count", 0),
```

("flag", False)

Esta función extrae la información de la lista entregada para cada marcador y lo guarda en formato legible para Rviz.

Salidas:

Listado de todos los marcadores entregados en la entrada con formato legible para Rviz, mark.

5.13.1.3 del_false_id

Entradas:

Listado de información de marcadores en formato del proyecto, array.

Transformaciones:

Se tiene un listado global de nombre false_id_marker, el cual contiene todos los id reales que se ubicaron en el mapa. La función identifica todos los id que se pasaron en la lista de entrada y elimina los que no estén en la lista false_id_marker. Se pueden presentar marcadores falsos al momento de mover el robot, ya que genera desenfoque y perturbación en la imagen, lo que puede hacer que la librería detecte marcadores que no existan.

Salidas:

Lista de marcadores filtrado por solo los que tienen ID conocido en el proyecto, array.

5.13.1.4 sum_vect

Entradas:

Listado de marcadores, Vect.

Información para el marcador 1, ArUco1.

Posición del marcador en la lista que se va a comparar con el marcador 1, i.

Índice del marcador con la posición más cercana con respecto a la cámara, index.

Transformaciones:

Por medio de las posiciones X, Y y Z de los marcadores, se encuentra la hipotenusa más pequeña. Si ArUco1 no tiene datos, lo primero que hace es asignarle información del marcador en la posición i en la lista vect. Si ArUco1 tiene dato, compara con un ArUco2. Aplica la función de la hipotenusa para encontrar la distancia más pequeña:

$$Hipotenusa = \sqrt{X^2 + Y^2 + Z^2}$$

Ecuación 1-Hipotenusa

Salidas:

Información de marcador con la menor hipotenusa, ArUco1.

Índice del marcador en la lista entregada, index.

5.13.1.5 ArUco_sort

Entradas:

Listado de marcadores encontrados por la librería ArUco ROS Melodic, array.

Transformaciones:

Utiliza el algoritmo de ordenamiento por mezcla para guardar en una lista en la posición 0 al marcador que está más a la derecha (X mayor) y así sucesivamente hasta ubicar en la posición final de la lista el marcador ubicado más a la izquierda (X menor).

Aquí está una explicación paso a paso del algoritmo:

División: Se divide la lista no ordenada en mitades sucesivas hasta que cada sublista tenga un solo elemento. Esto se hace recursivamente hasta que no se puede dividir más.

Ordenación: Se combinan dos sublistas adyacentes para formar una sublista ordenada más grande. Este proceso de combinación se realiza de manera recursiva hasta que toda la lista esté ordenada.

Combinación (*Merge*): Durante la combinación, los elementos de las dos sublistas ordenadas se comparan y se colocan en orden en una nueva lista. Este proceso se repite hasta que se haya formado una única lista ordenada.

Salidas:

Lista organizada de mayor a menor para los valores de X, array.

5.13.1.6 pos_calc

Entradas:

Lista de “Hijo” con información de posición y orientación, child.

Índice en la lista de “Hijo”, c.

Lista de “Padre” con información de posición y orientación, parent.

Índice en la lista de “Padre”, p.

Transformaciones:

Imaginemos que tenemos tres sistemas de coordenadas: A, B y C. Queremos determinar la posición del sistema de coordenadas A con respecto al sistema C, pero la única información que tenemos es la transformación desde A hasta B y desde B hasta C.

En este escenario, utilizamos un proceso secuencial de transformaciones para calcular la posición final de A con respecto a C. Primero, aplicamos la transformación que nos lleva de A a B, y luego aplicamos la transformación que nos lleva de B a C.

Este proceso de composición de transformaciones nos permite construir la transformación total que va desde A hasta C. En términos matemáticos, si denotamos la transformación de A a B como T_AB y la transformación de B a C como T_BC, entonces la transformación total de A a C, denotada como T_AC, se obtiene multiplicando estas dos transformaciones:

$$T_{AC} = T_{BC} \times T_{AB}$$

Ecuación 2-Transformación de AB a AC

De esta manera, logramos encontrar la posición del sistema de coordenadas A con respecto al sistema C a través de la secuencia de transformaciones desde A hasta B y luego desde B hasta C. Muy importante que la multiplicación se haga en el orden de la ecuación.

En nuestro proyecto siempre se llama matriz de transformación “Hijo” a T_AB y “Padre” a T_BC.

Existen 5 transformaciones posibles para nuestro proyecto. Las matrices se obtienen utilizando la función matriz_tf. A continuación, se detalla cada una:

5.13.1.6.1 Posiciones de los marcadores con respecto a la cámara

La librería ArUco ROS Melodic proporciona los valores asociados a las posiciones de los marcadores. En este contexto, esos valores corresponden a la matriz normal hija, derivada de la ubicación del marcador respecto a la cámara. Es esencial señalar que, en este escenario específico, la matriz padre se define como una matriz identidad 4x4. Esto se debe a que la matriz identidad representa un sistema coordenado en sí mismo; en este caso, representa la ubicación de la cámara con respecto a sí misma.

Dado que la multiplicación de cualquier matriz por la identidad resulta en la misma matriz, no se tiene en cuenta esta matriz padre en el análisis. En otras palabras, su inclusión no afecta los cálculos, ya que la matriz identidad actúa como un elemento neutro en la operación de multiplicación. Esta relación se visualiza claramente en la Ilustración 90:

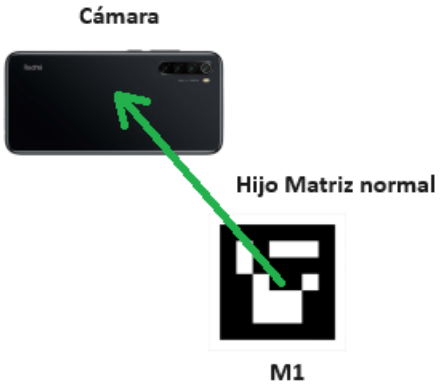


Ilustración 90-Transformación de marcador a cámara

Elaboración propia

5.13.1.6.2 Posición de la cámara con respecto al marcador escogido para el sistema global de coordenadas (MSGC)

En este caso, la transformación hija es de la cámara y es representada por una matriz inversa, debido a que se conoce la ubicación del MSGC con respecto a la cámara, pero no la de la cámara con respecto al MSGC. Para el caso de la matriz padre, aplica el mismo concepto de Posiciones de los marcadores con respecto a la cámara Esta relación se visualiza claramente en la Ilustración 91:

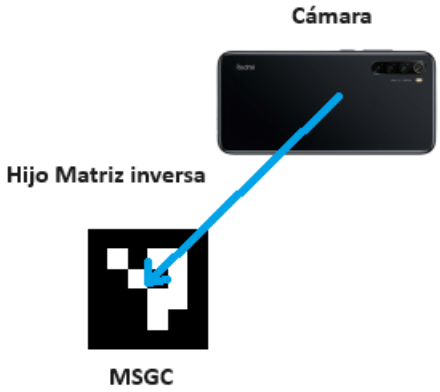


Ilustración 91-Transformación de cámara con respecto a marcador sistema global

Elaboración propia

5.13.1.6.3 Posición de un marcador con respecto al MSGC y el MSGC es visible en la imagen

En este contexto, la transformación hija implica el uso de un marcador cuya posición es desconocida, representada por una matriz normal. Por otro lado, la matriz padre se extiende desde la cámara hasta el MSGC y se presenta como una matriz inversa. La relación entre ambas transformaciones se muestra en la Ilustración 92.

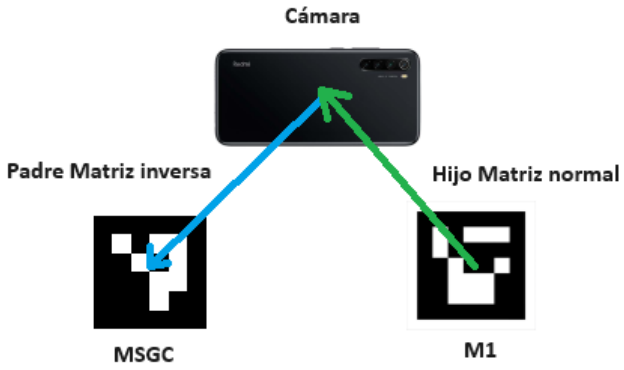


Ilustración 92-Transformación de marcador a marcador sistema global

Elaboración propia

5.13.1.6.4 Posición de la cámara con respecto a MSGC y MSGC no es visible

En esta situación, es imprescindible que, en caso de que MSGC no sea visible, al menos haya un marcador (llamémoslo M2) cuya posición con respecto a MSGC se haya calculado previamente. En este escenario específico, la transformación hija describe la relación entre la cámara y el marcador M2, representada por una matriz inversa. Por otro lado, la transformación padre representa la relación entre el marcador M2 y MSGC, utilizando una matriz normal. Esta conexión se muestra en la Ilustración 93:

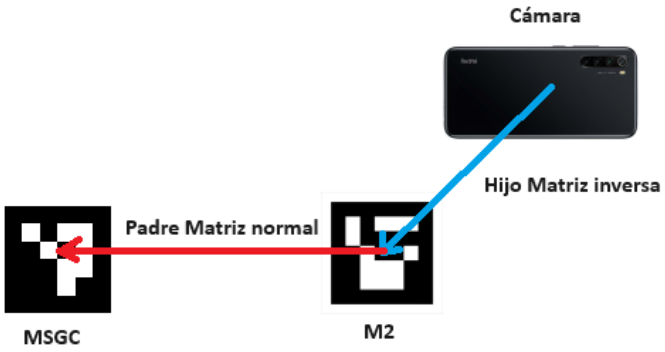


Ilustración 93-Transformación de cámara a sistema global

Elaboración propia

5.13.1.6.5 Posición de un marcador con respecto al MSGC y el MSGC NO es visible en la imagen

En este caso, el primer paso es encontrar la posición de la cámara con respecto al MSGC, cómo se explica en Posición de la cámara con respecto a MSGC y MSGC no es visible Esta será la transformación padre y se representará por una matriz normal. Luego, la transformación hija será la del marcador con respecto a la cámara y se representa por una matriz normal. Esta conexión se muestra en la Ilustración 94:

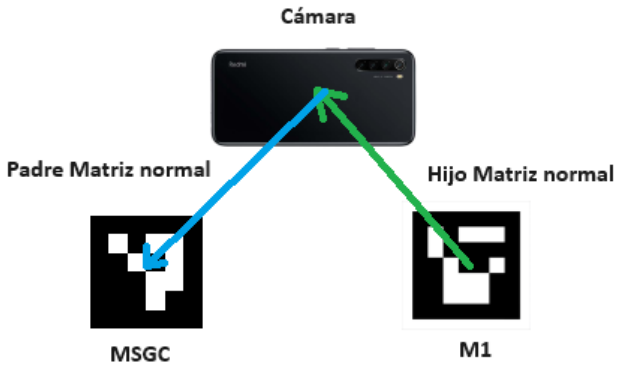


Ilustración 94-Transformación marcador a marcador calculado

Elaboración propia

La función pos_calc, se encarga de definir cual transformación aplicar y devuelve los resultados de posición y orientación en el formato del proyecto explicado en to_marker_type, para pasar los datos de matriz a valores, se usa la función quat_trans_from_matriz.

Salidas:

La lista que entró como child, con los datos de posición y orientación modificados de acuerdo con la transformación, child.

5.13.1.7 matriz_tf

Entradas:

Lista de marcadores con información de posición y orientación, data.

Índice en data de marcador a encontrar la matriz de transformación, i.

Indicador de si se necesita la matriz normal o inversa, flag.

Transformaciones:

la función toma datos sobre la posición y orientación de un marcador, con la función de Numpy np.array, crea una matriz de transformación homogénea y, según el valor de flag, devuelve la matriz de transformación normal (si flag es False) o la matriz de transformación inversa (si flag es True).

La matriz de transformación normal permite localizar un sistema O respecto a otro M, para nuestro caso es la ubicación del marcador con respecto al sistema coordenado de la cámara. Se usa la función de la librería de ROS fromTranslationRotation, que recibe la ubicación X, Y y Z y la orientación X, Y, Z y W para crear la matriz 4x4. Tiene la siguiente estructura matricial:

$$Matriz\ normal = \begin{bmatrix} R0 & R1 & R2 & Tx \\ R3 & R4 & R5 & Ty \\ R6 & R7 & R8 & Tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ecuación 3-Estructura matriz transformación

Donde desde R0 a R8 representa la matriz 3x3 de rotación y de Tx a Tz el vector traslación (los valores de posición de X, Y y Z del marcador).

La matriz de rotación se puede calcular con las siguientes fórmulas, donde Q representa a la orientación:

$$R0 = 1 - (2 \times Qy^2) - (2 \times Qz^2)$$

$$R1 = (2 \times Qx \times Qy) - (2 \times Qz \times Qw)$$

$$R2 = (2 \times Qx \times Qz) + (2 \times Qy \times Qw)$$

$$R3 = (2 \times Qx \times Qy) + (2 \times Qz \times Qw)$$

$$R4 = 1 - (2 \times Qx^2) - (2 \times Qz^2)$$

$$R5 = (2 \times Qy \times Qz) - (2 \times Qx \times Qw)$$

$$R6 = (2 \times Qx \times Qz) - (2 \times Qy \times Qw)$$

$$R7 = (2 \times Qz \times Qy) + (2 \times Qx \times Qw)$$

$$R8 = 1 - (2 \times Qx^2) - (2 \times Qy^2)$$

Ecuación 4-Fórmulas de rotación

La matriz de transformación inversa sirve para conocer la localización de M con respecto a O, en nuestro caso, la ubicación de la cámara con respecto al marcador. Es una matriz 4x4 que se calcula con la siguiente fórmula:

$$Matriz\ inversa = \begin{bmatrix} Rotación^{Transpuesta} & -Rotación^{Transpuesta} \times Translación \\ 0 & 1 \end{bmatrix}$$

Ecuación 5-Matriz de transformación inversa

Donde la transpuesta de la matriz se obtiene al aplicar la función de Numpy ".linalg.inv".

Salidas:

Matriz inversa si flag es True, matrizinver.

Matriz normal si flag es False, matriz.

5.13.1.8 quat_trans_from_matriz

Entradas:

Matriz de transformación de marcador, matriz.

Transformaciones:

Utiliza las funciones "transformations.orientaciónn_from_matrix" y "transformations. translation_from_matrix" de ROS para extraer la información de la orientación y posición respectivamente desde una matriz de transformación.

Salidas:

Vector de posiciones X,Y y Z, translation.

Vector de orientación Qx, Qy,Qz y Qw, quat.

5.13.1.9 pass_info

Entradas:

Lista donde guardar la información en el nuevo formato, new_data.

Lista donde extraer la información para guardar en el nuevo formato, old_data.

Ubicación en la lista new_data, donde guardar la información nueva, inew.

Ubicación en la lista old_data, donde extraer la información, iold.

Transformaciones:

Extrae la información en la ubicación iold de la lista old_data en formato marker.msg y la guarda en la ubicación inew de la lista new_data con el formato del proyecto explicado en to_marker_type. No asigna el valor del id, cuando detecta que el identificador en la que va a ingresar la información es el de la cámara.

Salidas:

Lista con la información en el formato del proyecto en la ubicación especificada, new_data.

5.13.1.10 on_plane

Entradas:

Listado de marcadores y/o cámara con la información de orientación y posiciones, array.

Transformaciones:

En el marco del proyecto, se lleva a cabo un experimento en un plano 2D con el propósito de posicionar el kobuki y los marcadores. Esta decisión se fundamenta en la ubicación del sensor MotoG5 y la necesidad de agilizar los cálculos al eliminar la información del eje Z, que carece de relevancia dado que el área de trabajo del proyecto es completamente plana, sin inclinaciones o deformaciones en el suelo. La función desarrollada para este propósito recibe datos de ubicación y orientación en un espacio 3D, transformándolos posteriormente a un espacio 2D con referencia a los ejes X e Y.

El procedimiento de la función implica recorrer la lista de información en el formato del proyecto. Para cada posición, se extraen los valores de los ángulos con respecto a los ejes X, Y y Z. La obtención de estos ángulos se realiza mediante la función de ROS "transformations.euler_from_orientaciónn". Esta función toma los valores de orientación y los convierte en ángulos Yaw (respecto a Z), Pitch (respecto a Y) y Roll (respecto a X).

Dado que la representación es en 2D con relación a X e Y, los valores de Pitch y Roll se descartan, siendo relevante únicamente la rotación Yaw. En consecuencia, se extraen los valores de quaternio considerando un Pitch y Roll de 0, y se utiliza la función de ROS "transformations.orientaciónn_from_euler" para obtener la orientación a partir de los ángulos.

Adicionalmente, se ajusta el valor de Z a 0 para que la ubicación quede en el plano X e Y. Este proceso garantiza una representación clara y ordenada de la información, simplificando la manipulación de datos en un entorno 2D coherente con las necesidades del proyecto.

Salidas:

Lista de información sobre el plano X e Y y rotación en Z, array.

5.13.1.11 dist_markers

Entradas:

Posición del marcador 1 en X, x1.

Posición del marcador 2 en X, x2.

Posición del marcador 1 en Y, y1.

Posición del marcador 2 en Y, y2.

Posición del marcador 1 en Z, z1.

Posición del marcador 2 en Z, z2.

Transformaciones:

Esta función se diferencia de sum_vect, ya que su objetivo es calcular la hipotenusa entre dos marcadores. Aunque implementa la misma fórmula que sum_vect, es necesario considerar los signos de cada posición para determinar si se deben sumar o restar al calcular los valores de los catetos X, Y y Z. En este proceso, se utiliza la función sum_rest.

Salidas:

Distancia de hipotenusa entre los marcadores, distance_markers.

5.13.1.12 sum_rest

Entradas:

Valor posición eje marcador 1, a.

Valor posición eje marcador 2, b.

Transformaciones:

La función tiene como objetivo determinar la variación de posición entre dos marcadores, marcador 1 y marcador 2, a lo largo de un eje específico. La operación depende de los signos asociados a las posiciones de ambos marcadores. Si las posiciones tienen el mismo signo, se obtiene el delta mediante una resta. En cambio, si las posiciones tienen signos opuestos, se realiza una suma.

Ejemplo mismo signo para X, donde el punto rojo representa al marcador 1 y el azul al marcador 2:

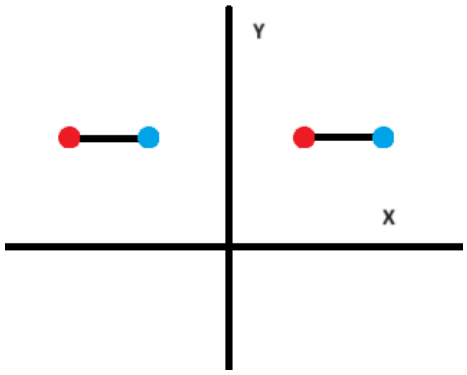


Ilustración 95- Posición para mismo signo para eje X

Elaboración propia

Ejemplo mismo signo para Y, donde el punto rojo representa al marcador 1 y el azul al marcador 2:

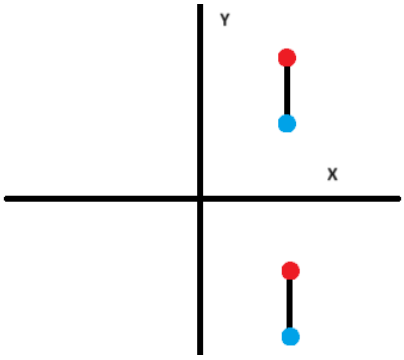


Ilustración 96-Posición para mismo signo en Y

Elaboración propia

Salidas:

El resultado de la suma o resta de las posiciones en el mismo eje, result.

5.13.1.13 publish_tf

Entradas:

Listado de información de marcadores y/o cámara en formato proyecto, data.

Posición del marcador y/o cámara a publicar la ubicación en Rviz, i.

Transformación padre a trasladar, parent_frame.

Transformaciones:

En Rvis, publica la posición y las orientaciones de la información en la ubicación especificada en la lista. Para ello, utiliza la función ROS sendTransform y nómbrala como "Odom". La transformación de referencia padre es "map", que se define como el sistema de coordenadas global del proyecto.

Salidas:

Publicación en Rviz de la posición de Odom con respecto a map.

5.13.2 GridMapping

Esta clase tiene como objetivo principal la generación del mapa de la oficina, integrando información sobre la ubicación global de los marcadores y presentándolo en un formato legible para Rviz. A continuación, se detallan las funciones clave de esta clase:

5.13.2.1 Bresenham

Entradas:

Posición en X de la cámara con respecto al sistema coordenado global, x0.

Posición en Y de la cámara con respecto al sistema coordenado global, y0.

Posición en X del marcador con respecto al sistema coordenado global, x1.

Posición en Y del marcador con respecto al sistema coordenado global, y1.

Transformaciones:

El algoritmo de Bresenham es un algoritmo de trazado de líneas que calcula eficientemente los píxeles que deben ser iluminados para dibujar una línea entre dos puntos en una cuadrícula discreta. Bresenham utiliza solo operaciones de suma y resta, evitando multiplicaciones y divisiones, lo que lo hace eficiente en términos de velocidad de cálculo.

Aquí está el desglose del código:

1. x_points e y_points son listas que almacenarán las coordenadas X e Y de los píxeles en la línea, respectivamente.

2. dx y dy calculan las diferencias en las coordenadas X e Y entre los dos puntos. Es importante destacar que X e Y deben ser valores enteros ubicados en la cuadrícula.

3. sx y sy son factores de signo que indican la dirección en la que se está moviendo en los ejes X e Y. Se establecen en 1 si el punto final es mayor que el punto inicial en la dirección correspondiente, de lo contrario, se establecen en -1.

4. error inicializa la variable de decisión para el algoritmo de Bresenham.

5. x e y se inicializan con las coordenadas del punto inicial x0, y0.

6. Se utiliza un bucle *while* para iterar hasta que se llega al punto final x1, y1. Dentro del bucle, se actualizan las listas x_points e y_points con las coordenadas actuales.

7. Se verifica si se ha alcanzado el punto final. Si es así, se rompe el bucle.

8. Se calcula e2, que es el doble del error actual.

9. Se verifica si e2 es mayor o igual a dy. Si es así, se actualiza el error y la coordenada X, y se avanza en la dirección X.

10. Se verifica si e2 es menor o igual a dx. Si es así, se actualiza el error y la coordenada Y, y se avanza en la dirección Y.

11. Finalmente, la función devuelve las listas x_points e y_points que contienen las coordenadas de los píxeles que forman la línea.

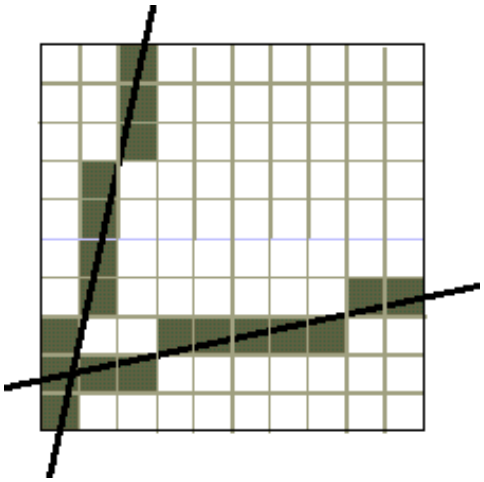


Ilustración 97-Representación gráfica de Bresenham

Elaboración propia

Salidas:

Listado de posiciones en X que deben estar marcados en el mapa, x_points.

Listado de posiciones en Y que deben estar marcados en el mapa, y_points.

5.13.2.2 to_ij

Entradas:

Posición en X del marcador o cámara en el sistema coordenado global, x.

Posición en Y del marcador o cámara en el sistema coordenado global, y.

Resolución de las cuadrículas del mapa, map_resolution.

Transformaciones:

Para la cuadrícula del mapa, es necesario que los valores de X e Y sean números enteros. La función en cuestión utiliza las coordenadas decimales del marcador o cámara en los ejes X e Y. Estos valores se dividen por la resolución del mapa y se redondean hacia abajo al número entero más cercano.

Salidas:

Valor entero de posición en X en el mapa, i.

Valor entero de posición en Y en el mapa, j.

5.13.2.3 to_ij_global

Entradas:

Listado de ubicaciones en unidades enteras de X o Y de marcadores o valor X o Y de la posición de la cámara, occ.

Ubicación en unidad entera en X o Y del mapa con respecto al sistema coordenado, map_center.

Resolución de los cuadrantes del mapa, map_resolution.

Transformaciones:

En ocasiones, podría ocurrir que, debido a descuidos al ingresar los parámetros en el archivo a_laser, el centro del mapa no quede alineado con el centro del sistema de coordenadas global. La función en cuestión utiliza las coordenadas (expresadas como valores enteros) con respecto al sistema de coordenadas global para un único eje (ya sea X o Y) y las traslada al sistema de coordenadas del mapa. Cuando el centro del mapa coincide con el centro del sistema global de coordenadas, los valores permanecen inalterados.

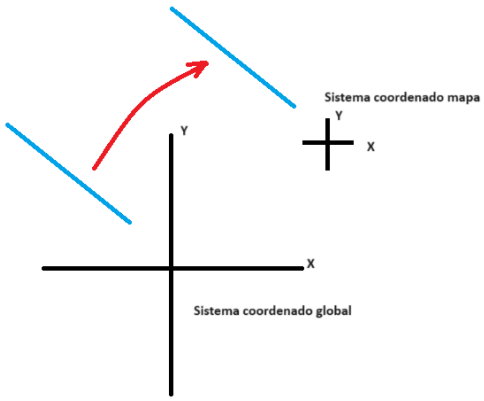


Ilustración 98-Traslación de trayectoria sistema global a sistema mapa

Elaboración propia

Esta función ofrece dos opciones: la opción de lista, que se utiliza cuando hay múltiples valores por ubicar en el mapa, y la opción simple, que se emplea cuando solo hay un valor a ubicar en el mapa.

Salidas:

Listado o valor de ubicaciones con valores enteros en el sistema coordenado del mapa, occ_global.

5.13.2.4 create_map

Entradas:

Vector de cuadrículas ocupadas, libres y neutras del mapa, data.

Transformaciones:

La función siguiente crea un mensaje legible denominado "OccupancyGrid()" diseñado para Rviz, el cual refleja el mapa generado. Este mensaje debe contener información crucial, como el nombre del

mapa, su resolución, ancho y largo en términos de cuadrículas, posición en coordenadas X e Y en relación con el sistema de coordenadas global, así como el vector de datos que indica cuáles cuadrículas están ocupadas, cuáles están libres y cuáles son neutras. En última instancia, la función pública este mensaje con el propósito de visualizarlo en Rviz.

Salidas:

Publicación del mensaje con la información del mapa en formato legible para ser visualizado en Rviz.

5.13.2.5 camera_tf
Entradas:

Información de ubicación y orientación de la cámara con respecto al sistema coordenado global, tf_c.

Transformaciones:

Se extrae la información de posición y orientación en formato marker.msg y se publica en un formato legible para Rviz mediante la función "TransformBroadcaster()". Además, se incluyen las transformaciones de base_footprint y base_link en la publicación. Estas transformaciones son necesarias al utilizar el paquete "Plataforma" para dirigir el movimiento del robot desde la terminal. Es importante destacar que estas transformaciones se encuentran en la misma posición que la cámara con respecto al sistema de coordenadas global.

Salidas:

Publicación en formato legible para Rviz de las ubicaciones y orientaciones de la cámara, base_footprint y base_link.

5.13.3 FuncToMapping
Esta clase contiene las funciones específicamente para identificar y agregar esquinas entre los marcadores, según ciertos criterios de distancia y ángulo. A continuación, se detallan las funciones clave de esta clase:

5.13.3.1 Euler
Entradas:

Información de orientación y posición del marcador con respecto al sistema coordenado global, m.

Transformaciones:

Con Qx, Qy, Qz y Qw extrae los ángulos Yaw (respecto a Z), Pitch (respecto a Y) y Roll (respecto a X) con la función de ROS “transformations.euler_from_quaternion”.

Salidas:

Vector con los ángulos del marcador con respecto al sistema coordenado global, euler.

5.13.3.2 corner_point
Entradas:

Información de orientación y posición del marcador con respecto al sistema coordenado global, mark.

Transformaciones:

La función toma un marcador como entrada y utiliza su posición en el plano XY y su orientación (ángulo de Euler con respecto al eje Z. Para esto utiliza la función Euler) para calcular los coeficientes m y b de la ecuación de la recta asociada al marcador.

La ecuación de la recta en el plano XY es de la forma:

$$Y = M \times X + B$$

Ecuación 6-Ecuación de la recta

Los valores m y b se calculan de la siguiente manera:

m: Se calcula usando el tangente del ángulo de Euler del marcador con respecto al eje Z.

b: Se calcula usando la posición del marcador en el plano XY y la pendiente m.

$$B = Y - M \times X$$

Ecuación 7-Ecuación de la recta despejada para B

Estos coeficientes representan la recta que pasa a través del marcador en el plano XY. La función devuelve estos coeficientes como resultado.

Salidas:

Coeficiente representante de la pendiente de la ecuación de la recta para el marcador, m.

Coeficiente representante de la ordenada al origen de la ecuación de la recta para el marcador, b.

5.13.3.3 angle_markers

Entradas:

Información de orientación y posición del marcador 1 con respecto al sistema coordenado global, m1.

Información de orientación y posición del marcador 2 con respecto al sistema coordenado global, m2.

Ángulo mínimo permitido entre marcadores, min_angle_allowed.

Ángulo máximo permitido entre marcadores, max_angle_allowed.

Transformaciones:

Esta función tiene como propósito determinar si el ángulo formado entre dos marcadores con respecto al eje Z está dentro del rango especificado por el ángulo mínimo y máximo. Para comprenderla mejor, es crucial familiarizarse con cómo se presenta la información de los ángulos y cómo se transmite en el contexto del proyecto.

La información sobre los ángulos de los marcadores se extrae de la biblioteca ArUco ROS Melodic. Esta biblioteca proporciona los ángulos en términos de quaternios. Al aplicar la función Euler a estos quaternios, se obtienen los ángulos en radianes. Es importante tener en cuenta que los ángulos resultantes se limitan al rango de 0 a π, y se les asigna un signo positivo o negativo según el cuadrante en el plano XY en el que se ubiquen. La Ilustración 99 muestra cómo se asigna el signo del ángulo según el cuadrante:

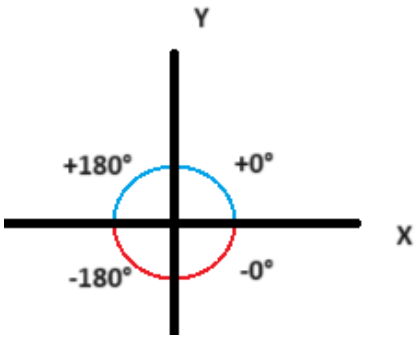


Ilustración 99-Signos de quaternio con respecto al cuadrante

Elaboración propia

Para comprender cómo se asigna el valor del ángulo en función de la rotación de un marcador, se utiliza una representación gráfica en un sistema coordenado XY. En esta representación, se incluyen elementos clave: una línea gris que simboliza al marcador desde una vista superior, una flecha roja que indica la dirección desde la cual es captado por la cámara, un punto verde que marca la ubicación de la cámara, y una curva azul que representa la medición del ángulo entre el eje X del sistema coordenado y el marcador.

A continuación, se detallan las cuatro posibilidades:

1. Cuando el marcador es detectado por la cámara en el cuadrante de la X y la Y positivos:

En esta situación, se tiene un ángulo negativo entre 0 y -π/2.

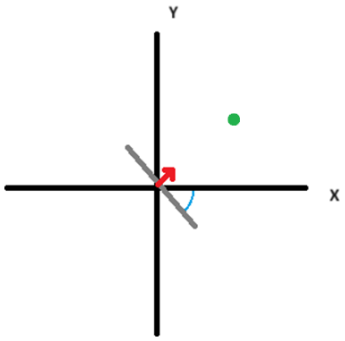


Ilustración 100-Valores quaternio para cámara en cuadrante 1

Elaboración propia

2. Cuando el marcador es detectado por la cámara en el cuadrante de la X negativa y la Y positiva:

En esta situación, se tiene un ángulo negativo entre 0 y π/2.

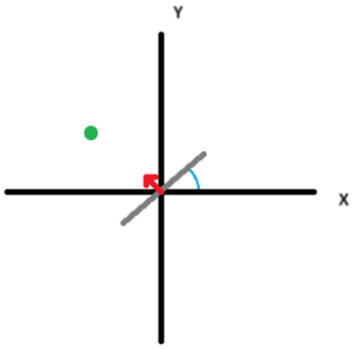


Ilustración 101-Valores quaternio para cámara en cuadrante 2

Elaboración propia

3. Cuando el marcador es detectado por la cámara en el cuadrante de la X y la Y negativas:

En esta situación, se tiene un ángulo negativo entre π/2 y-π.

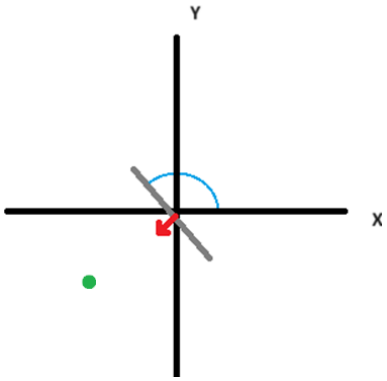


Ilustración 102-Valores quaternio para cámara en cuadrante 3

Elaboración propia

4. Cuando el marcador es detectado por la cámara en el cuadrante de la X positiva y la Y negativa:

En esta situación, se tiene un ángulo negativo entre -π/2 y -π.

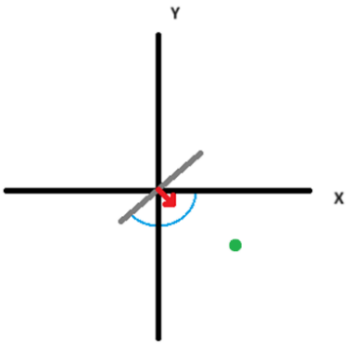


Ilustración 103-Valores quaternio para cámara en cuadrante 4

Elaboración propia

Ahora que conocemos cómo se interpreta y entrega la información de los ángulos de los marcadores con respecto al sistema global, hay que conocer qué ángulos se pueden generar entre 2 marcadores. Solo hay 2 posibilidades, esquinas internas y externas. El cálculo realizado por esta función siempre es entre 0 a 180° de apertura. A continuación, se detallan los tipos de esquinas, representándose la apertura calculada por la función mediante el arco azul:

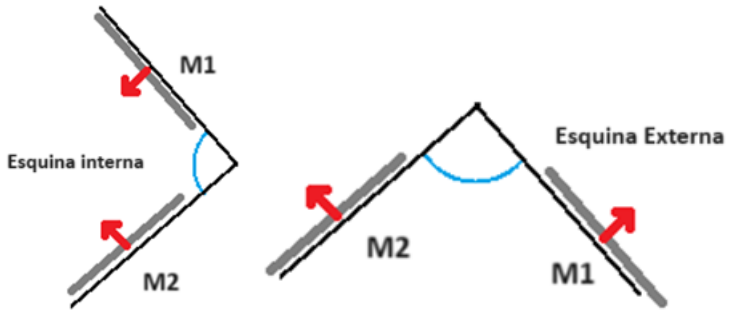


Ilustración 104-Ángulos formados por esquinas

Elaboración propia

Para calcular el ángulo de apertura primero es importante conocer qué signo tienen los ángulos de los marcadores con respecto al eje Z del sistema coordenado.

Si los signos de los ángulos de M1 y M2 son iguales y el ángulo de M1 en valor absoluto es mayor al ángulo en valor absoluto de M2, se usa la siguiente fórmula:

$$Apertura = \left|\left|Ángulo\ M2\right| + \pi - \left|Ángulo\ M1\right|\right|$$

Ecuación 8-Fórmula apertura para M1 y M2 signos iguales, pero M1 es mayor

Si los signos de los ángulos de M1 y M2 son iguales y el ángulo de M1 en valor absoluto es menor al ángulo en valor absoluto de M2, se usa la siguiente fórmula:

$$Apertura = \left|\left|Ángulo\ M1\right| + \pi - \left|Ángulo\ M2\right|\right|$$

Ecuación 9-Fórmula apertura para M1 y M2 signos iguales, pero M1 es menor

Si los signos de los ángulos de M1 y M2 no son iguales, se usa la siguiente fórmula:

$$Apertura = \left|\pi - \left|Ángulo\ M1\right| - \left|Ángulo\ M2\right|\right|$$

Ecuación 10-Fórmula apertura para M1 y M2 signos diferentes

Finalmente, verifica si el ángulo de apertura se encuentra en el rango de min_angle_allowed y max_angle_allowed.

Salidas:

True si el ángulo de apertura está en rango del ángulo máximo y mínimo.

False si el ángulo de apertura no está en rango del ángulo máximo y mínimo.

5.13.3.4 add_marker

Entradas:

Listado de información con orientación y posición de los marcadores con respecto al sistema coordenado global, array.

Posición en la lista del marcador y la distancia a evaluar, i.

Listado de distancias entre marcadores, dist.

Identificador único para el nuevo marcador a crear, id_corner.

Transformaciones:

La función tiene como objetivo principal la creación de un marcador cuando se identifica un ángulo de apertura entre dos marcadores, el cual debe estar dentro de un rango predefinido de ángulos mínimo y máximo. Este indicador señala la presencia de una esquina entre los dos marcadores, que posteriormente se utilizará en la generación del mapa.

Para iniciar, la función verifica si la distancia entre los marcadores cumple con el requisito máximo establecido. Si la distancia es mayor, existe la posibilidad de que no se haya identificado un marcador entre los dos en evaluación, ya que se espera que los marcadores estén a una distancia promedio de 15 cm, como se especifica en la sección Arucos a implementar en el entorno.

Si la distancia es válida, la función utiliza la función corner_point para extraer los coeficientes M y B correspondientes al marcador 1 y al marcador 2. Esto posibilita el cálculo del punto de intersección entre las funciones de ambos marcadores. Ambas funciones son ecuaciones lineales no paralelas, como se muestra en la Ilustración 105, donde la línea azul representa la función lineal asociada al marcador 1, la línea roja a la del marcador 2, y el punto negro indica la intersección de ambas funciones:

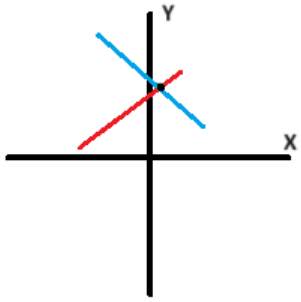


Ilustración 105-Intersección 2 ecuaciones lineales

Elaboración propia

Dado que el punto negro es aquel en el que ambas funciones son iguales respecto a las coordenadas X e Y, se utiliza la fórmula:

$$M1 \times X + B1 = M2 \times X + B2$$

Ecuación 11-Intersección entre ecuaciones lineales

Despejando X, obtenemos:

$$X = \frac{B2 - B1}{M1 - M2}$$

Ecuación 12-Despejar X para intersección

Para calcular Y, se puede utilizar cualquiera de las dos funciones lineales después de obtener el valor de X. En este caso, se utiliza la función lineal del marcador 1:

$$Y = M1 \times X + B1$$

Ecuación 13-Ecuación de la recta para valor Y de intersección

Finalmente, la función agrega a array el nuevo marcador identificado como esquina.

Salidas:

Listado de marcadores con el nuevo marcador tipo esquina agregado, solo con datos de posición X e Y con respecto al sistema coordenado global, array.

Listado de distancia entre marcadores con la distancia del marcador tipo esquina en True, dist.

5.13.3.5 get_values

Entradas:

Listado de información con la ubicación y orientación de los marcadores y la cámara con respecto al sistema coordenado global, marker_array.

Rango permitido de separación entre marcadores, max_dist_markers.

Ángulo mínimo permitido entre marcadores, min_angle_allowed.

Ángulo máximo permitido entre marcadores, max_angle_allowed.

Identificador inicial para los marcadores creados como esquinas, id_corner.

Transformaciones:

Es la función principal de la clase FuncToMapping y consiste en unificar todas las demás funciones para generar las esquinas del mapa.

En primer lugar, se realiza la identificación de los datos asociados a la cámara en el array de marcadores (marker_array). Estos datos se pueden reconocer debido a que el valor en frame_id es "/camera". Una vez localizada la cámara, se almacena en una variable aparte y se elimina del listado, ya que la cámara no influye en la creación de esquinas para el mapa.

A continuación, se verifica si hay más de un marcador en la lista marker_array, ya que, si solo hay un marcador, no existe la posibilidad de tener esquinas.

En caso de que haya más de un marcador, se utiliza la función dist_markers para identificar la distancia entre los marcadores y se guarda el valor en la variable dist. Este valor se compara con la distancia máxima permitida, y si es menor, se añade el valor verdadero a una lista llamada dist_list; de lo contrario, se añade un valor falso. En esta lista, la primera posición contiene la distancia entre M1 y M2, la segunda posición la distancia entre M2 y M3, y así sucesivamente, como se muestra en la Ilustración 106:

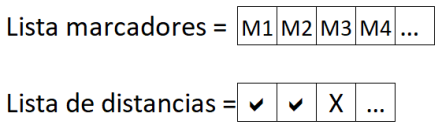


Ilustración 106-Ejemplo formato lista marcadores y distancias

Elaboración propia

Luego, utiliza la función angle_markers para identificar si el ángulo de apertura entre el marcador n y n+1 está en el rango de ángulo mínimo y máximo, de ser afirmativo, guarda en una lista la ubicación en marker_array del marcador n.

Finalmente, utiliza la función add_marker para agregar un marcador tipo esquina entre el marcador en la ubicación n y n+1.

Salidas:

Listado de marcadores, con los marcadores tipo esquina agregados, marker_array.

Listado de si la distancia entre el marcador n y n+1 está o no en el rango, dist_list.

Datos de posición y orientación de la cámara con respecto al sistema coordenado global, cam.

5.13.4 A_mapping

Esta función se ejecuta a través del archivo ArUco_melodic_mapping.launch con la línea <node pkg="ArUco_melodic_mapping" type="a_mapping.py" name="a_mapping" output="screen" required="true"/> sin comentar y la línea <!--<node pkg="ArUco_melodic_mapping" type="a_laser.py" name="map_from_json" output="screen" required="true" />--> comentada. El propósito de este código es definir un marcador como el marcador de referencia para el sistema coordenado global para luego calcular las posiciones y orientaciones de los otros marcadores con respecto a ese sistema coordenado.

Paso a paso del código:

Entradas:

Información de nodo /ArUco_r/markers con la información en formato marker.msg de todos los marcadores visibles por la cámara Xiaomi Note 8.

Transformaciones:

1. Obtenemos la información de orientación y posición de los marcadores visibles con respecto a la cámara.
2. Organizamos las posiciones de los marcadores del de más a la derecha hacia el más a la izquierda con la función ArUco_sort, para tener un mejor orden en la lista, ya que la librería de ArUco entrega el listado del menor ID al mayor UD.
3. Eliminamos los marcadores falsos que no existen en el entorno del experimento con la función del_false_id.
4. Verificamos si luego de eliminar los marcadores falsos, si todavía queda algún marcador en la lista. También, verificamos ya se definió cual era el marcador para el sistema de referencia Global.
5. Si no se ha definido, utilizamos la función de ROS transformations.quaternion_from_euler para definir la orientación del marcador con Roll (ángulo con respecto a X) de 90°, Yaw (ángulo con respecto a Z) de 0° y Pitch (ángulo con respecto a Y) de 0°. Así logramos que la posición y orientación del marcador global sea como el de la Ilustración 107, lo que permite tener un sistema coordenado global paralelo y alineado a los ejes principales de Rviz.

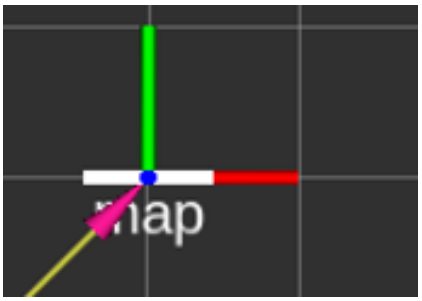


Ilustración 107-Dirección sistema coordenado global

Elaboración propia

Si no se realiza el paso de definir la ubicación del marcador global, los marcadores no se ubicarían de manera perpendicular al plano XY, sino que estarían perpendiculares al plano XZ. A continuación, se muestra la Ilustración 108 que muestra cómo quedan ubicados los marcadores de forma perpendicular tanto al plano XY como al plano XZ.

Plano XZ:

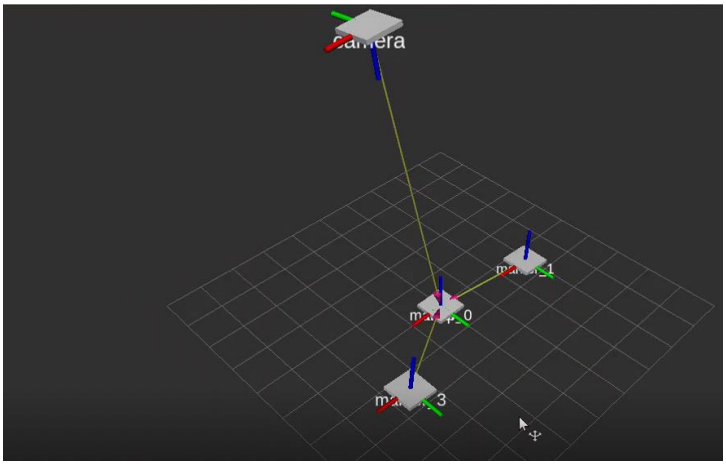


Ilustración 108-Vista marcadores en plano XZ

Elaboración propia

Plano XY:

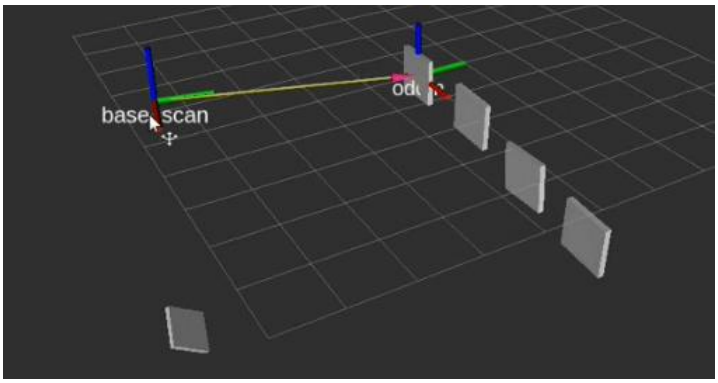


Ilustración 109-Vista marcadores en plano XY

Elaboración propia

6. Con el marcador global ubicado, revisamos si los siguientes marcadores de la lista ya habían sido identificados o si es la primera vez.
7. Si es la primera vez en ser identificados, almacena la información de ubicación con respecto a la cámara.
8. Para que un marcador sea considerado con posición válida, debe haber sido detectado al menos N veces seguidas. En este proyecto, N se ha definido como 4. Esta elección se fundamenta en la necesidad de recopilar varios datos consecutivos para realizar un promedio ponderado y calcular así la posición del marcador de manera precisa.
   La razón detrás de requerir que las detecciones sean consecutivas radica en minimizar el desplazamiento del Kobuki. Dado que la frecuencia de imágenes es de 30 fps (frames por segundo), con un valor de N igual a 4, el lapso entre la primera y la cuarta imagen es de tan solo 0.13 segundos. Esta cercanía temporal es esencial para garantizar que el desplazamiento del robot entre detecciones sea despreciable.
   En caso de que el marcador no sea visible en las N imágenes subsiguientes por alguna razón, el código descarta los datos recopilados hasta ese momento y reinicia el conteo desde cero. El proceso se repite hasta que el marcador sea detectado N veces consecutivas, asegurando así la robustez y confiabilidad en el cálculo de la posición.
9. En la detección del marcador, la posición relativa al sistema coordenado global se calcula mediante la función pos_calc. El proceso comienza con la identificación del marcador global en la imagen. Si este marcador es visible, se emplea el método Posición de un marcador con respecto al MSGC y el MSGC es visible en la imagen En caso de no encontrar el marcador global, se requiere al menos un marcador con posición global válida. En este escenario, se sigue el método Posición de la cámara con respecto a MSGC y MSGC no es visible, seguido por el método Posición de un marcador con respecto al MSGC y el MSGC NO es visible en la imagen.
10. Cuando un marcador cuenta con sus cuatro posiciones globales, se realiza una comparación entre ellas para confirmar que la disparidad no sea considerable. En este proyecto, se ha establecido un umbral de 0.002 metros para las posiciones y de 0.02° para la orientación. Este enfoque busca prevenir errores en el cálculo de la posición y orientación que podrían surgir debido a movimientos abruptos del Kobuki o a la pérdida o disminución de la señal.
11. Si la mitad o más de las posiciones presentan una disparidad mayor a lo establecido, se descartan las posiciones y se empieza nuevamente desde el paso 7 para ese marcador.
12. Si más del 50% de las posiciones cumplen con una disparidad menor a la establecida, se promedian los valores de posición y orientación globales para el marcador. Este promedio se considera como una posición válida en el sistema global de referencia y se almacena en la lista de los marcadores calculados.
    La metodología gana importancia en experimentos que incluyen el cálculo de posición y la extracción de N posiciones sucesivas. A continuación, se presenta la posición obtenida tanto para un único cálculo como para cuatro. Es relevante destacar que los marcadores estaban ubicados en una pared, por lo que debían estar alineados colinealmente entre sí.

N = 1:

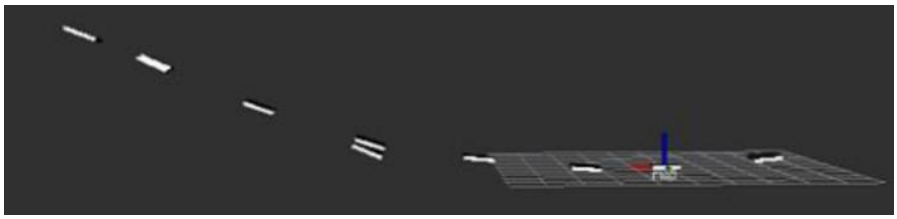


Ilustración 110-Posiciones de marcadores encontrados con N = 1

Elaboración propia

N = 4:

Ilustración 111-Posiciones de marcadores encontrados con N = 4

Elaboración propia

Se eligió N igual a 4, ya que al aumentar este valor no se observaban mejoras significativas en la posición publicada. Sin embargo, el tiempo de procesamiento aumentaba exponencialmente, pasando de 0.15 segundos con 4 cálculos a 0.8 segundos con 5.

13. Los marcadores visibles con posición global válida se utilizan para calcular las posiciones de los marcadores sin posición válida. No se vuelve a calcular la posición de los marcadores ya válidos.
14. Se publica en Rviz un marcador global, junto con todos los marcadores válidos y la posición de la cámara, para cada imagen detectada. Esto permite revisar visualmente la detección de todos los marcadores, identificando posibles faltantes. La información publicada incluye el ID, la forma, la posición y la orientación en relación con el sistema global de coordenadas.
15. Cuando se detiene el código, la información se guarda en formato JSON bajo el nombre "markers_world_pos". Este archivo contiene los datos de cada marcador válido, incluyendo el ID, la posición y la orientación en quaternios con respecto al sistema de coordenadas global. Es importante destacar que la posición y orientación proporcionadas no son las planares con respecto al plano XY.

Salidas:

Publicación de la posición y orientación de la cámara con respecto al sistema coordenado global.

Publicación de la posición y orientación de los marcadores válidos con respecto al sistema coordenado global.

Archivo JSON con información de posición y orientación global de los marcadores válidos.

5.13.5 A_laser

Esta función se ejecuta a través del archivo ArUco_melodic_mapping.launch con la línea <node pkg="ArUco_melodic_mapping" type="a_laser.py" name="map_from_json" output="screen" required="true" /> sin comentar y la línea <!--<node pkg="ArUco_melodic_mapping"

type="a_mapping.py" name="a_mapping" output="screen" required="true"/>--> comentada. El propósito de este código es crear el mapa por medio de la ubicación global de todos los marcadores.

Paso a paso del código:

Entradas:

Información de nodo /ArUco_r/markers con la información en formato marker.msg de todos los marcadores visibles por la cámara Xiaomi Note 8.

Transformaciones:

5.13.5.1 Función Open_json

1. Para obtener las posiciones de todos los marcadores en el entorno con respecto al sistema coordenado global, es necesario extraer la información del archivo JSON denominado markers_world_pos. Es crucial ejecutar previamente el código A_mapping antes de ejecutar este fragmento de código.
2. Luego, extraemos todos los ID de los marcadores que son visibles en la imagen. Esto nos ayuda a saber qué información se debe extraer de markers_world_pos.
3. Extraemos la información de los ID que son visibles en la imagen y los almacenamos en una variable.
4. Usamos la función sum_vect de PositionToCalc para encontrar el marcador visible más cercano a la cámara para calcular la posición de la cámara con respecto al sistema coordenado global.
5. Paso seguido convierte la información del marcador más cercano al formato de datos del proyecto, con la función pass_info de PositionToCalc.
6. Calcula la posición del Kobuki con respecto al marcador más cercano usando la función pos_calc de PositionToCalc, con la metodología explicada en Posición de la cámara con respecto a MSGC y MSGC no es visible.
7. Una vez calculada la posición del Kobuki, procedemos a pasarla al plano XY usando PositionToCalc con su función on_plane.
8. También, pasamos la información de los marcadores visibles al plano XY (recordemos que markers_world_pos guarda toda la información, no la planar) con la función on_plane de PositionToCalc.
9. Agregamos al listado de marcadores, la información de la cámara.
10. Convertimos toda la información del listado a formato de datos del proyecto con la función to_marker_type de PositionToCalc.

5.13.5.2 Función ArUco_map

1. Organizamos de derecha a izquierda el listado de marcadores visibles con la función ArUco_sort de PositionToCalc.
2. Eliminamos los marcadores que realmente no existen con la función del_false_id de PositionToCalc.
3. Extraemos la información de la ubicación global de los marcadores y la cámara en el plano XY con la función Función Open_json y la almacenamos en una variable.
4. Verificamos que el listado tenga más de 2 conjuntos de datos. Mínimo debe haber 2 marcadores y la cámara. Sino no puede realizar cálculos.
5. Usamos la función get_values de FuncToMapping para identificar si entre los marcadores se encuentra alguna esquina y de ser así, que agregue un marcador tipo esquina.
6. Al ejecutar el código por primera vez, se configura una variable que indique la realización exitosa del cálculo inicial del mapa. Los valores de posición de la cámara en la imagen actual (n) se almacenan en una variable para su comparación con los de la siguiente imagen (n+1).
   Es crucial destacar que este procedimiento es esencial debido a la posibilidad de que movimientos abruptos del Kobuki generen un "salto" en su ubicación. Por lo tanto, se realiza

una verificación para asegurarse de que la diferencia de ubicación entre n+1 y n no supere los 100 cm. Esto es especialmente relevante para el proyecto, considerando que el robot no puede desplazarse a una velocidad superior a 2.5 km/h.
Si ya se había calculado la posición de la cámara, extrae la distancia entre la posición en la imagen n, con la posición en la imagen n+1. Para esto usa la función dist_markers de PositionToCalc. Si lo supera, no realiza el mapa.
Si la distancia es menor, procede a recorrer la lista para crear la línea de borde ocupado (línea negra en la Ilustración 112) entre marcadores y la línea libre (línea blanca en la Ilustración 112) entre los marcadores y la cámara cómo se evidencia en la Ilustración 112:

Ilustración 112-Vista de mapa generado en Rviz

Elaboración propia

7. Para crear la línea negra, primero debe extraer la posición en X e Y de los marcadores n y n+1 y almacenarlos en variables.
8. Para crear un mapa se necesita de datos enteros para representar a las coordenadas de la cuadrícula. Para esto usamos la función to_ij de GridMapping.
9. Paso seguido es utilizar la función Bresenham de GridMapping, para conocer las coordenadas de la línea que se genera entre el marcador n y el n+1.
10. Los datos obtenidos por Bresenham los pasamos al sistema coordenado del mapa usando la función to_ij_global de GridMapping, porque puede que el centro del mapa no esté en el punto 0,0 del sistema coordenado global (esto puede suceder por un error al ingresar de manera manual las dimensiones y centroide del mapa). La línea creada por en esta etapa corresponde a la línea roja de la Ilustración 113:

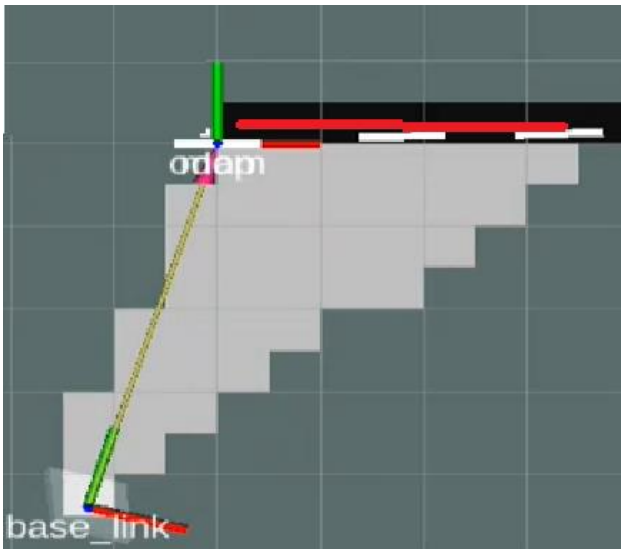


Ilustración 113-Cuadrantes ocupados en mapa

Elaboración propia

11. Se hace el mismo proceso de volver los datos enteros y el traslado al sistema coordenado del mapa para la información de la cámara.
12. Luego, se recorre cada una de las coordenadas de la línea negra y se les asigna el valor de 100, el cual indica que es una coordenada negra u ocupada.
13. Después de eso, se utiliza la función Bresenham de GridMapping para trazar las líneas libres entre las cuadrículas ocupadas y la cámara. La representación visual muestra las coordenadas libres identificadas mediante Bresenham en líneas azules, mientras que los puntos rojos indican cada cuadrícula ocupada:

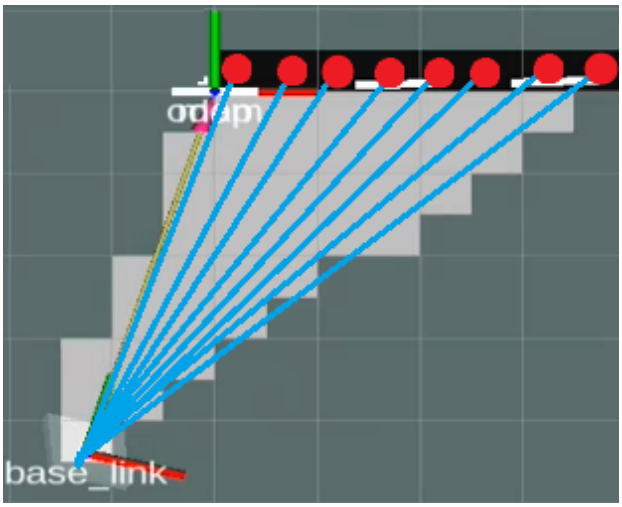


Ilustración 114-Aplicación de Bresenham para crear cuadrantes libres

Elaboración propia

14. Cuando se obtienen las coordenadas de cada cuadrícula libre, se asigna a cada una el valor de 0 para representar un espacio disponible.
15. Finalmente, se utiliza la función create_map de GridMapping para crear el mapa en el formato legible para Rviz y se publica la posición de la cámara en Rviz con la función camera_tf de GridMapping.

5.13.5.3 Función Save_map

Esta función guarda el mapa creado en Función_ArUco_map en formato JSON con nombre "map_from_json_occupancy_grid.json".

Salidas:

JSON de coordenadas en el mapa con los valores de libre (0), ocupado (100) o neutro (1) para cada una de las cuadrículas, map_from_json_occupancy_grid.json.

Publicación de la posición y orientación en el plano XY global para la cámara.

Publicación de mapa generado en formato legible para Rviz.

5.13.6 kobuki_real_world_pos

Esta función se ejecuta a través del archivo Kobuki_real.launch con el propósito de captar la información de posición en X e Y del Kobuki en relación con el *groundtruth*. Como se detalla en la selección de groundtruth, se utiliza un marcador fijo y otro sobre el Kobuki para rastrear su ubicación respecto al primero. El código emplea funciones de la librería Funtions_to_slam para identificar el marcador fijo, calcular la posición del marcador del Kobuki con respecto al fijo y almacenar los datos de posición X e Y junto con la hora de detección.

Paso a paso del código:

Entradas:

Información de nodo /ArUco_r/markers con la información en formato marker.msg de todos los marcadores visibles por la cámara MotoG5.

Transformaciones:

1. Extrae la información del JSON initial_marker_pos, el cual indica cual es el marcador fijo y su ubicación con respecto al sistema coordenado global para el *groundtruth*.
2. Identificamos si hay más de 1 marcador en la lista de marcadores identificados por el MotoG5. Si solo hay 1 marcador quiere decir que la cámara del MotoG5 no está visualizando al Kobuki, ya que, por su ubicación, siempre visualiza al marcador fijo, cómo se explica en Sensores a utilizar. Si no se ven ambos marcadores, no calcula nada.
3. Se verifica si se ha calculado la ubicación del MotoG5 con respecto al marcador fijo.
    1.3.1. Si no se ha calculado, procede a revisar si existe el archivo "real.json" (porque puede que se haya realizado antes un experimento que haya dejado el archivo en la carpeta), si existe lo elimina para que no afecte los nuevos datos que se van a recopilar.
    1.3.2. Paso seguido convierte la información del marcador fijo encontrado por la librería ArUco ROS Melodic al formato de datos del proyecto, con la función pass_info de PositionToCalc.
    1.3.3. Calcula la posición de la cámara aplicando la función pos_calc de PositionToCalc, donde aplica la matriz inversa para encontrar la posición de la cámara.
    1.3.4. Configura la variable para que no vuelva a calcular la posición de la cámara.
4. Si la posición ya está calculada, convierte la información del marcador ubicado en el Kobuki al formato del proyecto, con la función pass_info de PositionToCalc.
5. Calcula la posición del marcador del Kobuki con respecto al marcador fijo usando la función pos_calc de PositionToCalc, con la metodología explicada en Posición de un marcador con respecto al MSGC y el MSGC es visible en la imagen.

6. Una vez calculada la posición del Kobuki, procedemos a pasarla al plano XY usando PositionToCalc con su función on_plane.
7. Si es la primera vez que se calcula la posición del Kobuki con respecto al marcador fijo, crea el archivo “real.json” para almacenar la información. Configura la variable para que se identifique que ya se calculó la primera posición.
8. Finalmente, utiliza la función pos_in_time de PositionToCalc para agregar la información de ubicación del Kobuki con respecto al marcador fijo al archivo “real.json” como histórico, no borra los datos anteriores.
9. Realiza todas estas verificaciones y procesos hasta que se apague el código manualmente, indicando así la finalización del experimento.

Salidas:

Archivo de histórico de ubicaciones X e Y del Kobuki con respecto al marcador fijo, con su respectivo tiempo en el que se calculó la ubicación.

5.13.7 kobuki_world_pos

La función que se ejecuta a través del archivo Kobuki.launch tiene como objetivo captar la información de posición en las coordenadas X e Y del Kobuki con respecto al mapa. Para lograr esto, se emplean marcadores a lo largo del perímetro del entorno experimental, según se detalla en la sección Arucos a implementar en el entorno. A través de las funciones A_mapping y A_laser, se determina la posición de todos los marcadores en relación con un sistema de coordenadas global, generando así el mapa correspondiente, para que posteriormente, el Kobuki sigua una serie de trayectorias para ubicarse, extrayendo la información del cálculo de ubicación.

El código utiliza funciones de la biblioteca Funtions_to_slam para identificar los marcadores visibles por el Xiaomi Note 8. Luego, se determina la ubicación de estos marcadores en el sistema de coordenadas global, lo que permite calcular la posición del Kobuki en dicho sistema. Este proceso facilita la comparación con el *groundtruth*, permitiendo evaluar la precisión de la ubicación del Kobuki.

Paso a paso del código:

Entradas:

Información de nodo /ArUco_r/markers con la información en formato marker.msg de todos los marcadores visibles por la cámara Xiaomi Note 8.

Transformaciones:

1. Para obtener las posiciones de todos los marcadores en el entorno con respecto al sistema coordenado global, es necesario extraer la información del archivo JSON denominado markers_world_pos. Es crucial ejecutar previamente el código A_mapping y posteriormente el código A_laser antes de ejecutar este fragmento de código.
2. Identificamos las distancias de todos los marcadores visibles con respecto a la cámara usando la función dist_markers de PositionToCalc.
3. Organizamos el listado de distancias de menor a mayor.
4. Con el listado de distancias ordenado, organizamos la lista de marcadores de acuerdo con su respectiva distancia. Esto nos permite obtener el marcador más cercano a la cámara, siendo el más idóneo para realizar el cálculo de la cámara con respecto al sistema coordenado global. Este marcador cercano es probablemente el que presenta menos problemas de incertidumbre y otras falencias asociadas a la

implementación de marcadores tipo ArUco, las cuales se detallan en el artículo Uso de Marcadores ArUco en SLAM.

5. Paso seguido convierte la información del marcador más cercano al formato de datos del proyecto, con la función pass_info de PositionToCalc.
6. Buscamos en markers_world_pos la posición del marcador más cercano con respecto al sistema coordenado global y lo guardamos en una variable.
7. Calcula la posición del Kobuki con respecto al marcador más cercano usando la función pos_calc de PositionToCalc, con la metodología explicada en Posición de la cámara con respecto a MSGC y MSGC no es visible.
8. Una vez calculada la posición del Kobuki, procedemos a pasarla al plano XY usando PositionToCalc con su función on_plane.
9. Al calcular la posición inicial del Kobuki en relación con el sistema de coordenadas globales por primera vez, se procede a la creación del archivo "calculate.json" para almacenar la información correspondiente. Luego, se configura una variable que indique la realización exitosa del cálculo inicial. Los valores de posición en la imagen actual (n) se almacenan en una variable para su comparación con los de la siguiente imagen (n+1).

   Es crucial destacar que este procedimiento es esencial debido a la posibilidad de que movimientos abruptos del Kobuki generen un "salto" en su ubicación. Por lo tanto, se realiza una verificación para asegurarse de que la diferencia de ubicación entre n+1 y n no supere los 50 cm. Esto es especialmente relevante para el proyecto, considerando que el robot no puede desplazarse a una velocidad superior a 2.5 km/h.
10. Si ya se había calculado la posición del Kobuki, extrae la distancia entre la posición en la imagen n, con la posición en la imagen n+1. Para esto usa la función dist_markers de PositionToCalc.
11. Si la distancia es menor a los 50 cm propuestos, utiliza la función pos_in_time de PositionToCalc para agregar la información de ubicación del Kobuki con respecto al sistema coordenado global al archivo “calculare.json” como histórico, no borra los datos anteriores.
12. Realiza todas estas verificaciones y procesos hasta que se apague el código manualmente, indicando así la finalización del experimento.

Salidas:

Archivo de histórico de ubicaciones X e Y del Kobuki con respecto al sistema coordenado global, con su respectivo tiempo en el que se calculó la ubicación.

5.14 CONFIGURACIÓN DE RVIZ

En el contexto de ROS, RViz es una herramienta de visualización 3D que se utiliza para visualizar datos de sensores, información del robot, y otros elementos en un entorno tridimensional.

Aquí hay algunas características clave de RViz y cómo se utiliza en ROS Melodic:

1. Visualización de Modelos 3D: RViz permite visualizar modelos 3D de robots, sensores y otros objetos. Puedes cargar modelos URDF (*Unified Robot Description Format*) para representar visualmente el robot y sus componentes.

2. Visualización de Datos de Sensores: RViz puede mostrar datos en tiempo real de varios sensores, como cámaras, láseres y radares. Esto es útil para verificar la salida de los sensores y asegurarse de que estén funcionando correctamente.

3. Planificación de Trayectorias: Puedes planificar y visualizar trayectorias para el robot en RViz. Esto es útil para verificar la planificación de movimiento y asegurarse de que el robot siga la trayectoria deseada.

5.Configuración Interactiva: RViz permite la configuración interactiva de visualizaciones y parámetros, lo que facilita ajustar la presentación de datos según las necesidades específicas del usuario.

En el contexto del proyecto, se utiliza Rviz para publicar la ubicación de marcadores con forma de cuadrado en un sistema global de referencia, los sistemas coordenados nativos de cada marcador y de la cámara y el mapa generado por el SLAM.

Para configurar Rviz se puede hacer de dos formas: generando un archivo de configuración ".rviz" o de manera manual. En la subsección Lista de archivos Rviz de la sección ArUco mapping implementado en ROS se explican los archivos que configuran Rviz para el entorno del proyecto. Sin embargo, es importante conocer cómo manipularlo manualmente.

Rviz puede ejecutarse desde la consola realizando los siguientes pasos:

1. Ejecutar en 1 consola el script que activa el entorno de ROS:

```
Roscore
```

2. Ejecutar en otra consola el script que abre la herramienta:

```
Rosrun rviz rviz
```

Para visualizar una publicación en Rviz, primero se debe considerar el tipo de publicación que se quiere visualizar. Al abrir Rviz y presionar el botón "Add", se despliega el listado disponible de tipos de publicación, como se evidencia en la Ilustración 115:

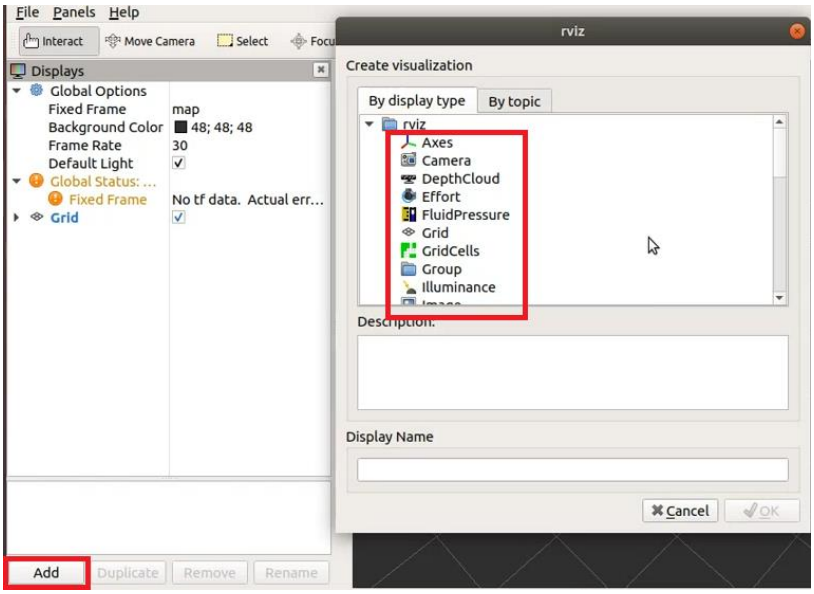


Ilustración 115-Agregar visualizaciones para Rviz

Elaboración propia

En nuestro proyecto, las publicaciones de códigos se clasifican en tres tipos principales: Map, TF y Marker.

5.14.1 Map

Se trata de un conjunto de cuadrículas de colores que constituyen un mapa. Estas cuadrículas pueden ser de tres colores distintos:

- Blanco, indicando que la cuadrícula se considera libre.
- Negro, cuando la cuadrícula se considera un obstáculo.
- Gris, cuando no se tiene certeza sobre si la cuadrícula es libre u obstáculo.

La Ilustración 116 es una representación de una publicación de un mapa en Rviz:

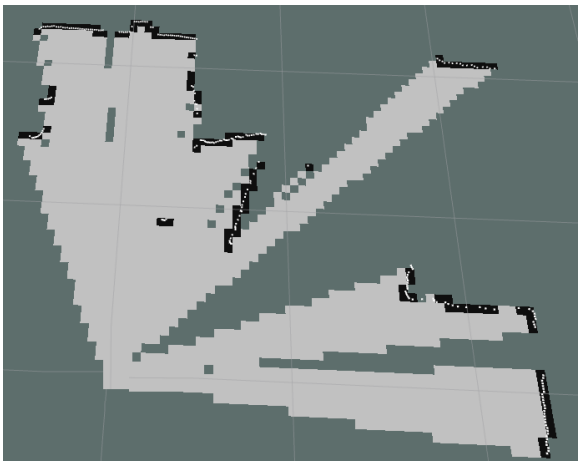


Ilustración 116-Ejemplo de mapa en Rviz

Elaboración propia

Para visualizar un mapa en Rviz, selecciona el tipo "Map" de la lista. El mapa aparecerá en el menú izquierdo; despliégalo y elige la opción "Topic". Selecciona el nombre de la publicación que está transmitiendo la información del mapa.

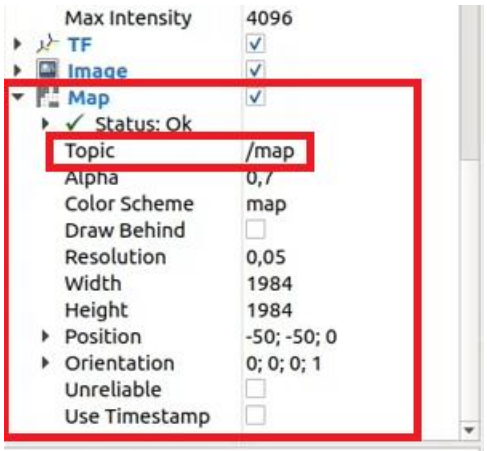


Ilustración 117-Editar Topic mapa en Rviz

Elaboración propia

En el proyecto, el único código que publica un Topic del tipo map se encuentra en el archivo a_laser. Esta publicación se realiza a través de la función create_map de GridMapping, utilizando la siguiente línea de código:

map_pub = rospy.Publisher(map_frame, OccupancyGrid, queue_size=100)

En esta línea, map_frame es el nombre de la publicación que se debe configurar como Topic en Rviz. Para este proyecto en particular, se ha decidido nombrarlo "/map".

5.14.2 TF

Este tipo de publicación se refiere a los sistemas coordenados nativos utilizados por los marcadores y la cámara.

Se representan con un nombre, un cilindro rojo que representa el eje X, un cilindro verde que representa el eje Y y un cilindro azul que representa el eje Z.

La Ilustración 118 muestra la representación de un TF en Rviz:

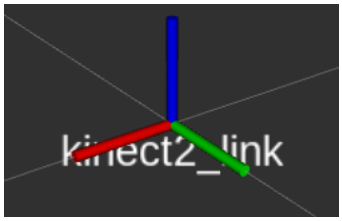


Ilustración 118-Ejemplo de TF en Rviz

Elaboración propia

Para visualizar un mapa en Rviz, selecciona el tipo "TF" de la lista. El mapa aparecerá en el menú izquierdo; despliégalo y habilita el checkbox “Show Names”, “Show Axes”, “Show Arrows” y en el Dropdown de “Frames” habilita el checkbox “All Enabled”. Así se mostrará automáticamente todos los TF que están siendo publicados en el momento.

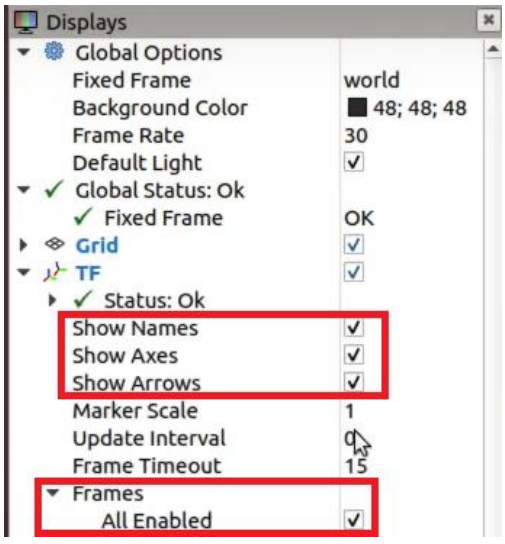


Ilustración 119-Editar TF en Rviz

Elaboración propia

En el Proyecto, los códigos A_laser, A_mapping, kobuki_real_world_pos y kobuki_world_pos publican los tópicos tipo TF por medio de las funciones camera_tf de GridMapping y publish_tf de PositionToCalc. Estas funciones utilizan la siguiente línea de código para publicar los TF:

tl.sendTransform((px,py,pz), (rx,ry,rz,w), time, name, parent_frame)

El “parent_frame” es el nombre del sistema coordenado global, para el caso de este proyecto se nombra “world”. Esta información se puede validar en el menú de la izquierda de Rviz en la sección de “Global Options” en “Fixed Frame”. (px,py,pz) son las posiciones X, Y y Z del TF con respecto al “parent_frame”. (rx,ry,rz,w) son los quaternios del TF con respecto al “parent_frame”. Time es el tiempo en el que se publica el dato y name es el nombre que aparece asociado al TF.

5.14.3 Marker

Representa los cubos que simbolizan a los marcadores en nuestro proyecto. A esta geometría se le puede especificar el largo, ancho, alto y color. La Ilustración 120 hace referencia a la geometría seleccionada para los marcadores del proyecto:

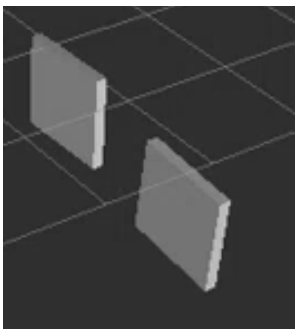


Ilustración 120-Ejemplo marcador en Rviz

Elaboración propia

Para visualizar un mapa en Rviz, selecciona el tipo "Marker" de la lista. El mapa aparecerá en el menú izquierdo; despliégalo y elige la opción "Topic". Selecciona el nombre de la publicación que está transmitiendo la información de los marcadores.

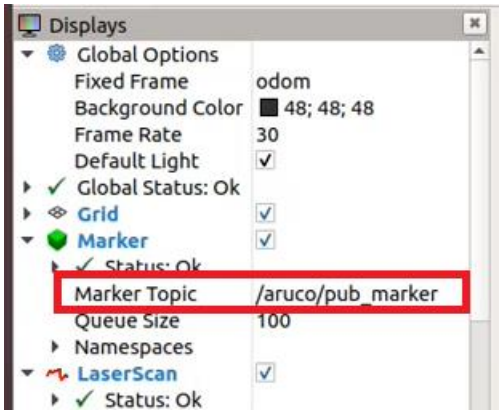


Ilustración 121-Editar Topic marker en Rviz

Elaboración propia

En el proyecto, el único código que publica un Topic del tipo marker se encuentra en el archivo a_mapping. Esta publicación se realiza a través de la siguiente línea de código:

pub=rospy.Publisher("/ArUco/pub_marker", Marker, queue_size=100)

En esta línea, /ArUco/pub_marker es el nombre de la publicación que se debe configurar como Topic en Rviz.

Después de configurar manualmente el Rviz, es posible guardar las configuraciones en un archivo con extensión ".rviz". Para hacerlo, accede al menú de Rviz en la esquina superior izquierda, selecciona la opción "File", elige "Save" y guarda el archivo en la ubicación y con el nombre que prefiera el usuario.

6 EXPERIMENTO Y EXTRACCIÓN DE RESULTADOS

6.1 EXPERIMENTO

Habiendo explicado todas las metodologías necesarias para la comprensión del experimento, presentamos en esta sección el paso a paso definido para realizar el experimento.

1. Imprimir los marcadores tipo ArUco definidos para el SLAM en Arucos a implementar en el entorno, recortarlos y ubicarlos en el entorno con cinta. Verificar que la superficie esté limpia de grasa y polvo para que tenga buena adherencia.
2. Imprimir los marcadores tipo ArUco definidos para el *groundtruth* en selección de groundtruth, siendo el ID 1 el del Kobuki y el ID 0 el fijo. Recortarlos y ubicarlos en el entorno con cinta. Verificar que la superficie esté limpia de grasa y polvo para que tenga buena adherencia.
3. Crear, instalar y activar los paquetes, librerías y programas explicados en Computador, uso de ros, instalación, paquetes.
4. Configurar los archivos ".launch" y ".py" para que sean ejecutables como se explica en Entorno de trabajo del proyecto.
5. Instalar Droidcam en el Xiaomi Note 8 y el MotoG5 y configurar los archivos ".launch" cómo se explica en Droidcam.
6. Calibrar el Xiomi Note 8 y el MotoG5 según se explica en Calibración de cámaras usando ROS y guardar los archivos en la ruta definida en el archivo ".launch".
7. Ubicar los celulares de acuerdo con la sección Sensores a utilizar en el Kobuki y la oficina.
8. Con todo organizado, el primer código que se debe activar es el a_mapping. **Tener en cuenta la configuración necesaria explicada en la sección ArUco_mapping implementado en ROS para la correcta ejecución del código**. Desde la consola usamos el siguiente script para ejecutar el código:

```
Roslaunch nombre_paquete nombre_archivo_.launch
```

   Donde, nombre_paquete es ArUco_melodic_mapping y nombre archivo es ArUco_melodic_mapping.launch. Hay que recordar que Droidcam debe estar abierto en el celular antes de ejecutar el código.
9. Se ubica al Kobuki a 30cm de la pared y con la cámara en dirección de la flecha azul de la Ilustración 122. En el experimento el marcador global fue el de ID 13. Realiza el desplazamiento del Kobuki siguiendo la trayectoria roja de la Ilustración 122, manteniendo la cámara paralela al perímetro. Puedes desplazar el Kobuki utilizando el paquete definido en Paquete Plataforma o empujándolo manualmente mientras está apagado.

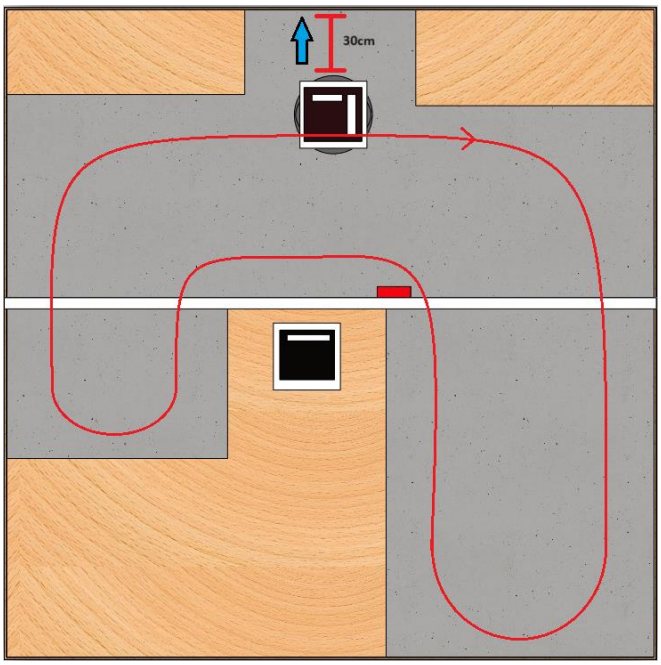


Ilustración 122-Trayectoria Kobuki para crear mapa

Elaboración propia

10. Se verifica en Rviz que se estén ubicando en el mapa todos los marcadores. Si en algún punto, no aparece un marcador, se debe devolver el Kobuki a una posición donde se vea el marcador faltante y al menos 1 marcador válido.
11. Una vez realizado el recorrido rojo y se visualicen todos los marcadores, detener el código desde la consola presionando Control + C. Automáticamente se genera el documento markers_world_pos.
12. Una recomendación eficaz consiste en examinar las distancias entre los marcadores del archivo markers_world_pos para asegurar que concuerden con las mediciones reales tomadas con el flexómetro, dentro de una tolerancia específica. Esto permite confirmar la exactitud del cálculo de las posiciones. La sección Resultados muestra los valores de RMSE, MSE y MAE para dicha comparación.
13. Con la ejecución de esta parte del código, habríamos completado los requisitos necesarios para llevar a cabo el SLAM completo en el proyecto. En este punto, el Kobuki adquiere la capacidad de ubicarse y generar el mapa de manera simultánea.
14. Con la información global de todos los marcadores guardada, se debe activar el código a_laser, para realizar el SLAM. **Tener en cuenta la configuración necesaria explicada en la sección A_laser para la correcta ejecución del código**. Desde la consola usamos el siguiente script para ejecutar el código:

```
Roslaunch nombre_paquete nombre_archivo_.launch
```

    Donde, nombre_paquete es ArUco_melodic_mapping y nombre archivo es ArUco_melodic_mapping.launch. Hay que recordar que Droidcam debe estar abierto en el celular antes de ejecutar el código.
15. En la ejecución de esta fase del experimento, la cámara no necesita estar instalada en el Kobuki. Puedes sostenerla con las manos y desplazarte alrededor de la oficina para mapear el entorno. Al abrir la pestaña de Rviz y visualizar la imagen captada por la cámara, inicia la identificación de cada marcador. Asegúrate de que la línea negra que conecta los

marcadores forme el contorno perimetral, y que aparezcan líneas blancas entre la cámara y los marcadores. Repite este proceso hasta completar el perímetro de la oficina en línea negra y asegúrate de que el interior quede completamente en blanco. Rviz muestra la ubicación de la cámara lo que ayuda a guiar al usuario para generar el mapa.

16. Una vez se tengan todas las cuadrículas del mapa con su respectivo color, detener el código desde la consola presionando Control + C. Automáticamente se genera el documento map_occupancy_grid, el archivo PGM y el YAML del mapa.
17. Para evaluar la efectividad y precisión de la metodología en comparación con el *groundtruth*, se llevaron a cabo 9 pruebas de ruta con trayectorias aleatorias. Durante estas pruebas, se registró la información de ubicación del Kobuki y el tiempo en que se obtuvo cada dato. La comparación se realizó entre los datos obtenidos mediante SLAM y el *groundtruth*. Para realizar esta evaluación, fue necesario preparar ambas cámaras y ejecutar simultáneamente dos códigos: kobuki_real_world_pos y kobuki_world_pos.
18. Antes de ejecutar los códigos, es necesario posicionar el Kobuki de manera visible para el MotoG5. Durante la ejecución de la prueba, asegúrese de que el Kobuki sea constantemente visible para el MotoG5. Esto implica supervisar la pestaña de Rviz, donde se proyecta la imagen del MotoG5, y ajustar manualmente la posición del Kobuki. En caso de que el Kobuki no sea visible en algún momento, es crucial apagar el código y reiniciar el proceso.
19. Para activar ambos códigos simultáneamente, asegúrate de tener disponibles dos consolas con los paquetes correspondientes activados. Inicia ejecutando el script (1) en una consola para activar la recopilación de datos destinada al *groundtruth*. Posteriormente, en la otra consola, ejecuta el script (2) para llevar a cabo la recopilación de datos desde el Kobuki.

```
(1)Roslaunch ArUco_melodic_mapping kobuki_real.launch

(2)Roslaunch ArUco_melodic_mapping kobuki.launch
```

Con ambos códigos corriendo, se puede empezar a mover el Kobuki.

20. El usuario tiene la flexibilidad de determinar la duración del tiempo y la forma de la trayectoria en un sistema específico. Durante las pruebas realizadas con el Kobuki, se observó que, en términos generales, las pruebas tuvieron una duración promedio de 1 a 3 minutos. La trayectoria, por otro lado, se ajustaba de manera manual a medida que el Kobuki se movía.
21. Para terminar de recopilar datos, detener el código desde la consola presionando Control + C en ambas consolas, siendo el código del Kobuki el primero en apagar y luego el *groundtruth*. Automáticamente se genera el documento “real.json” para el *groundtruth* y “calculate.json” para el Kobuki.
22. Para mejor manipulación de los datos, se puede usar la página https://products.aspose.app/cells/conversion/json-to-xlsx. que convierte un documento en formato JSON a Excel.
23. Después de transformar los datos a Excel, se debe abrir la información correspondiente a "calculate". En este paso, se verifica el momento en el que el Kobuki comienza a moverse para evitar la presencia de datos en 0, dado que existe un período entre la activación del código y el inicio del movimiento del Kobuki. Una vez identificado este intervalo, se procede a eliminar los datos anteriores.
24. Con el tiempo definido, se accede a los datos de "real" y se busca dicho tiempo, permitiendo una tolerancia de 0.5 segundos. Posteriormente, se eliminan los datos anteriores a este punto, asegurando que ambos conjuntos de datos comiencen en el mismo intervalo de tiempo.
25. Este proceso también se aplica a los últimos datos, ya que se presenta un lapso cuando el Kobuki se detiene y los códigos se apagan. De esta manera, se evita la comparación de datos innecesarios y se logra una mayor coherencia en la información analizada.

26. Con los datos organizados, se aplica la metodología de RMSE, MSE y MAE para encontrar la diferencia entre el *groundtruth* y los datos desde el Kobuki. RMSE, MSE y MAE debe aplicarse para los datos tanto en X como en Y, para el mismo periodo de tiempo. Si no se cuenta con datos en el mismo tiempo, se debe encontrar el dato equivalente, haciendo una interpolación, siendo los datos del *groundtruth* los fijos y los datos del Kobuki a los que hay que realizarles la interpolación. También se calculan las distancias entre las posiciones para el correspondiente tiempo. Este proceso se explica en la sección Resultados.

6.2 ROOT MEAN SQUARE ERROR (RMSE) - ERROR CUADRÁTICO MEDIO

El RMSE es una métrica comúnmente utilizada para medir la diferencia entre los valores predichos y los valores reales en un conjunto de datos. Es particularmente útil en problemas de regresión, donde estás tratando de predecir un valor continuo.

6.2.1 Fórmula

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}\left(Y_{Kobuki\ i} - Y_{groundtruth\ i}\right)^2}{n}}$$

Ecuación 14-Fórmula RMSE

- n es el número total de observaciones.

- Y_kobuki es el valor predicho para la observación i.

- Y_groundtruth es el valor real para la observación i.

6.2.2 Significado y Utilización

1. Calidad de las Predicciones:

- Cuanto menor sea el RMSE, mejor se ajustan las predicciones al conjunto de datos real.
- El RMSE penaliza de manera significativa los errores más grandes, ya que se elevan al cuadrado.

2. Interpretación:

- La unidad de medida del RMSE es la misma que la del conjunto de datos (por ejemplo, si estás prediciendo datos en metros, la unidad resultante sería metros).

3. Resultados:

- Un RMSE de 0 indica que las predicciones son perfectas.
- Cuanto mayor sea el RMSE, mayor es la dispersión de los errores, lo que sugiere que las predicciones pueden no ajustarse bien a los datos reales.

4. Sensibilidad a Outliers:

- Dado que los errores se elevan al cuadrado, el RMSE es sensible a valores atípicos. Grandes errores tendrán un impacto más significativo.

6.3 MEAN SQUARED ERROR (MSE) - ERROR CUADRÁTICO MEDIO

El MSE es una métrica ampliamente utilizada para evaluar la calidad de las predicciones en un conjunto de datos, especialmente en problemas de regresión. A continuación, se presenta una descripción similar a la proporcionada para el RMSE:

6.3.1 Fórmula

$$MSE = \frac{\sum_{i=1}^{n}\left(Y_{Kobuki\ i} - Y_{groundtruth\ i}\right)^2}{n}$$

Ecuación 15-Fórmula MSE

- n es el número total de observaciones.

- Y_kobuki_i es el valor predicho para la observación i.

- Y_groundtruth_i es el valor real para la observación i.

6.3.2 Significado y Utilización

1. Calidad de las Predicciones:

- Cuanto menor sea el MSE, mejor se ajustan las predicciones al conjunto de datos real.
- El MSE penaliza de manera significativa los errores más grandes, ya que se elevan al cuadrado.

2. Interpretación:

- La unidad de medida del MSE es el cuadrado de la unidad original del conjunto de datos. Por ejemplo, si estás prediciendo datos en metros, la unidad resultante sería metros al cuadrado.

3. Resultados:

- Un MSE de 0 indica que las predicciones son perfectas.
- Cuanto mayor sea el MSE, mayor es la dispersión de los errores, indicando que las predicciones pueden no ajustarse bien a los datos reales.

4. Sensibilidad a Outliers:

- Dado que los errores se elevan al cuadrado, el MSE es sensible a valores atípicos. Grandes errores tendrán un impacto más significativo.

6.4 MEAN ABSOLUTE ERROR (MAE) - ERROR ABSOLUTO MEDIO

El MAE es otra métrica común para evaluar la precisión de las predicciones. A continuación, se presenta una descripción similar:

6.4.1 Fórmula

$$MAE = \frac{\sum_{i=1}^{n}\left|Y_{Kobuki\ i} - Y_{groundtruth\ i}\right|}{n}$$

Ecuación 16-Fórmula MAE

- n es el número total de observaciones.

- Y_kobuki_i es el valor predicho para la observación i.

- Y_groundtruth_i es el valor real para la observación i.

6.4.2 Significado y Utilización

1. Calidad de las Predicciones:

- Cuanto menor sea el MAE, mejor se ajustan las predicciones al conjunto de datos real.
- A diferencia del MSE, el MAE no penaliza de manera significativa los errores más grandes, ya que no se elevan al cuadrado.

2. Interpretación:

- La unidad de medida del MAE es la misma que la del conjunto de datos original (por ejemplo, si estás prediciendo datos en metros, la unidad resultante sería metros).

3. Resultados:

- Un MAE de 0 indica que las predicciones son perfectas.
- Cuanto mayor sea el MAE, mayor es la magnitud promedio de los errores, indicando una menor precisión en las predicciones.

4. Sensibilidad a Outliers:

- El MAE es menos sensible a valores atípicos en comparación con el MSE, ya que no eleva los errores al cuadrado.

6.5 RESULTADOS

6.5.1 Ubicación de marcadores

Los resultados iniciales del experimento incluyen las ubicaciones de los marcadores con respecto al sistema coordenado global. Para validar la precisión de estos cálculos, se recomienda verificar si las distancias entre los marcadores son similares a las medidas reales mencionadas en la sección Lugar de experimento.

La evaluación de los resultados implica la selección al azar de 2 marcadores "paralelos" entre sí.

Se extraen sus valores de ubicación con respecto al sistema coordenado global en el eje perpendicular y se comparan con los valores reales medidos con un flexómetro. El método utilizado para determinar las distancias entre los marcadores es el mismo que se emplea en las funciones sum_rest y dist_markers.

La Ilustración 123 es ilustrativa para saber cómo se definió la distancia.

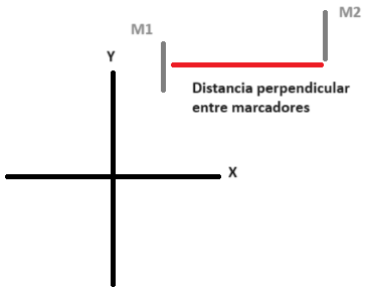


Ilustración 123-Referencia de distancia perpendicular entre marcadores

Elaboración propia

En total, se tomaron 14 medidas, seleccionando 2 de cada medida para asegurar que el primer dato no fuera precisamente el exacto de la medida real debido a la elección aleatoria. Se presenta la siguiente tabla, donde la primera columna muestra los ID de los marcadores medidos, la segunda columna muestra las distancias entre esos marcadores, y la tercera columna presenta los datos medidos con el flexómetro, expresados en metros.

Marcadores	Distancia entre marcadores (m)	Distancia Real (m)
63 y 85	1,598	1,6
1 y 35	1,589	1,6
81 y 34	1,060	1
83 y 18	1,062	1
82 y 77	2,908	2,9
78 y 15	2,845	2,9
5 y 4	2,524	2,52
9 y 16	2,531	2,52
71 y 26	1,212	1,21
69 y 22	1,214	1,21
13 y 60	2,950	2,95
11 y 9	2,928	2,95
14 y 65	0,782	0,77
36 y 61	0,777	0,77

Tabla 1-Resultados de distancias entre marcadores comparado con las reales

Elaboración propia

Aplicando a los datos reales y medidos las metodologías de MAE, RMSE y MSE se obtienen los siguientes resultados:

MAE = 0,0186

MSE = 0,00821

RMSE = 0,0287

Los resultados de los métodos implementados indican una precisa determinación de las ubicaciones de los marcadores. Las mediciones se sitúan en un rango de 0.7 a 3 unidades de metro, y los valores de MAE, RMSE y MSE muestran hasta 24 veces menor discrepancia en comparación con el dato mínimo de la tabla y el RMSE que representa el valor máximo entre los métodos evaluados. Al ser metodologías dependientes de la escala de las variables, se pueden considerar pequeños los valores de las metodologías, por lo tanto, se pueden considerar precisas las ubicaciones de los marcadores.

6.5.2 Mapa

Con los resultados de las posiciones calculadas de los marcadores con respecto al sistema global verificadas y aprobadas, se procede a ejecutar el paso 14 de la sección Experimento y extracción de resultados. El resultado de la ejecución se muestra a continuación:

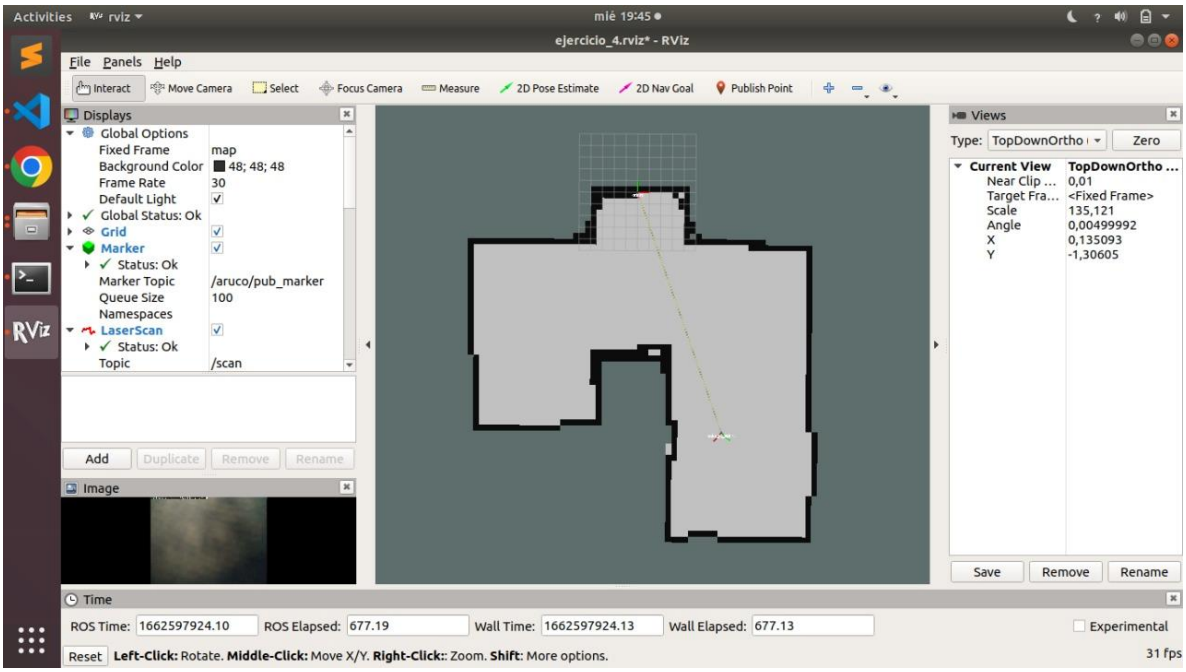


Ilustración 124-Resultado de mapa generado para el proyecto

Elaboración propia

El mapa generado es consistente en cuanto a perímetro, zonas libres y zonas desconocidas que se presenta en la sección Lugar de experimento.

6.5.3 Pruebas

Se llevan a cabo un total de 9 pruebas de ruta para el Kobuki, enumeradas desde 0. En cada prueba, se representan gráficamente las posiciones del *groundtruth* en azul y las posiciones del Kobuki en naranja. El eje X indica la posición en el eje horizontal, mientras que el eje Y indica la posición en el eje vertical. Las gráficas muestran datos interpolados después de la eliminación de la información innecesaria, como se detalla en el punto 23 de la sección Experimento y extracción de resultados.

A continuación, se muestran las trayectorias correspondientes a cada una de las pruebas.

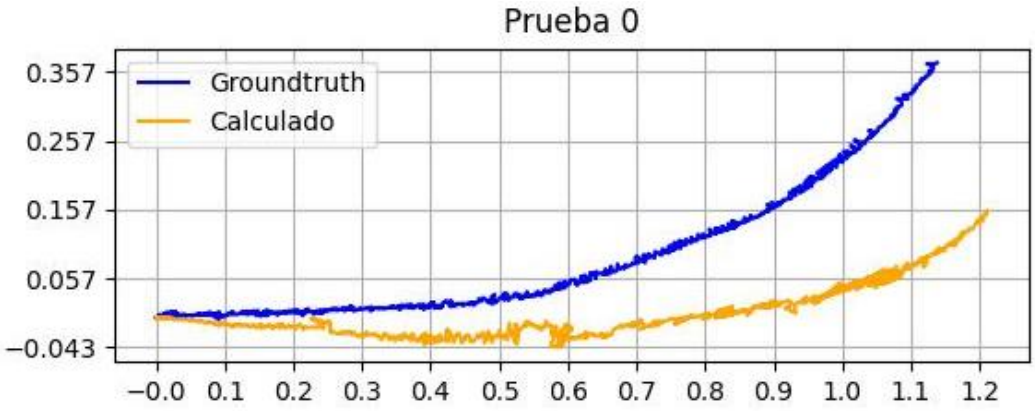


Ilustración 125-Resultados posiciones globales Kobuki y groundtruth Prueba 0

Elaboración propia

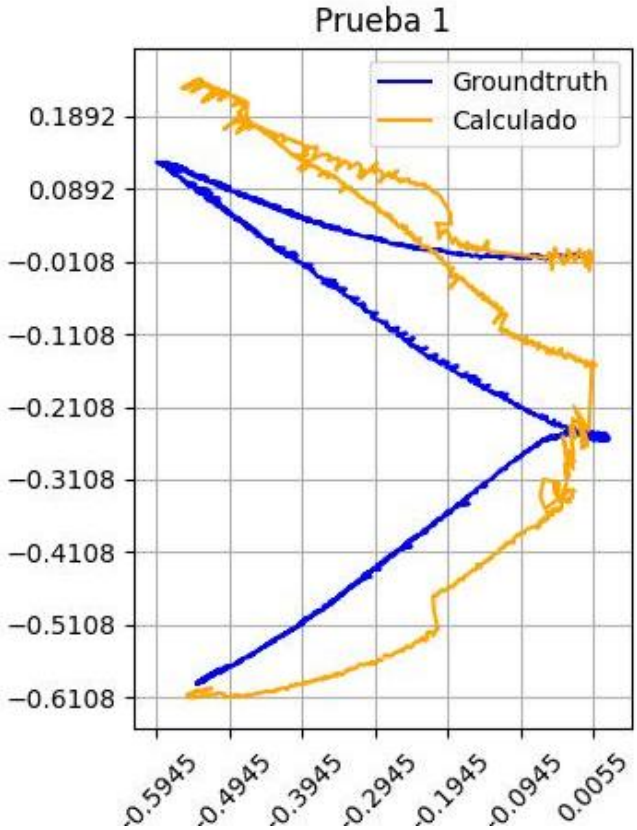


Ilustración 126-Resultados posiciones globales Kobuki y groundtruth Prueba 1

Elaboración propia

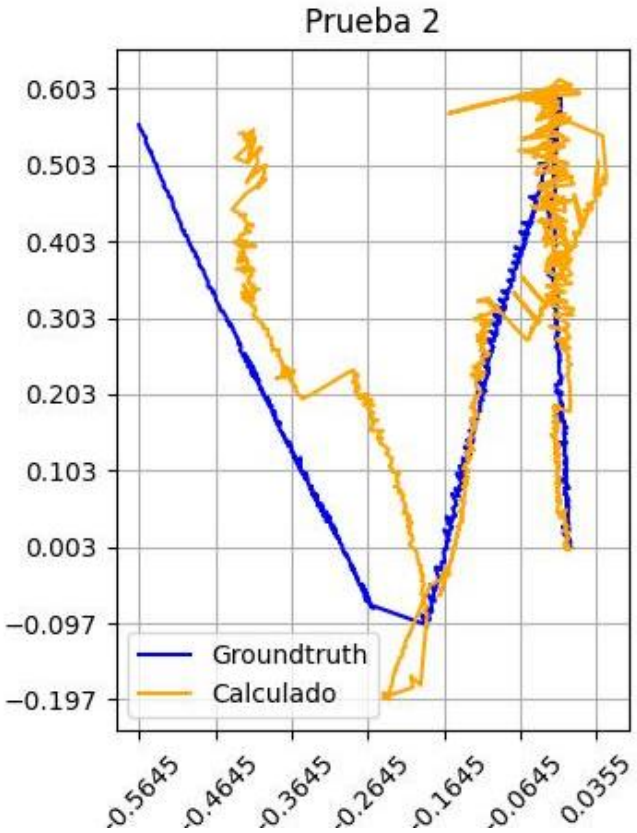


Ilustración 127-Resultados posiciones globales Kobuki y groundtruth Prueba 2

Elaboración propia

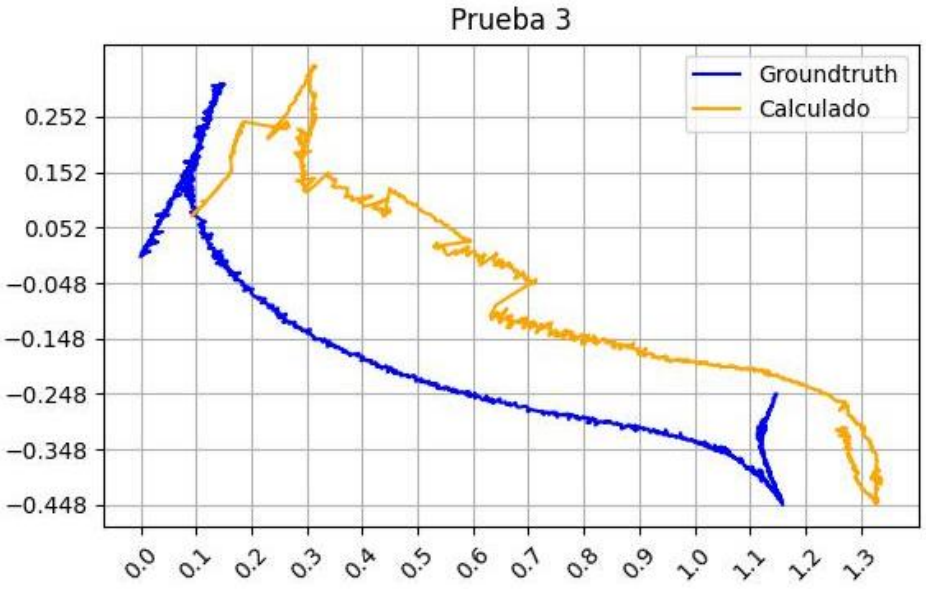


Ilustración 128-Resultados posiciones globales Kobuki y groundtruth Prueba 3

Elaboración propia

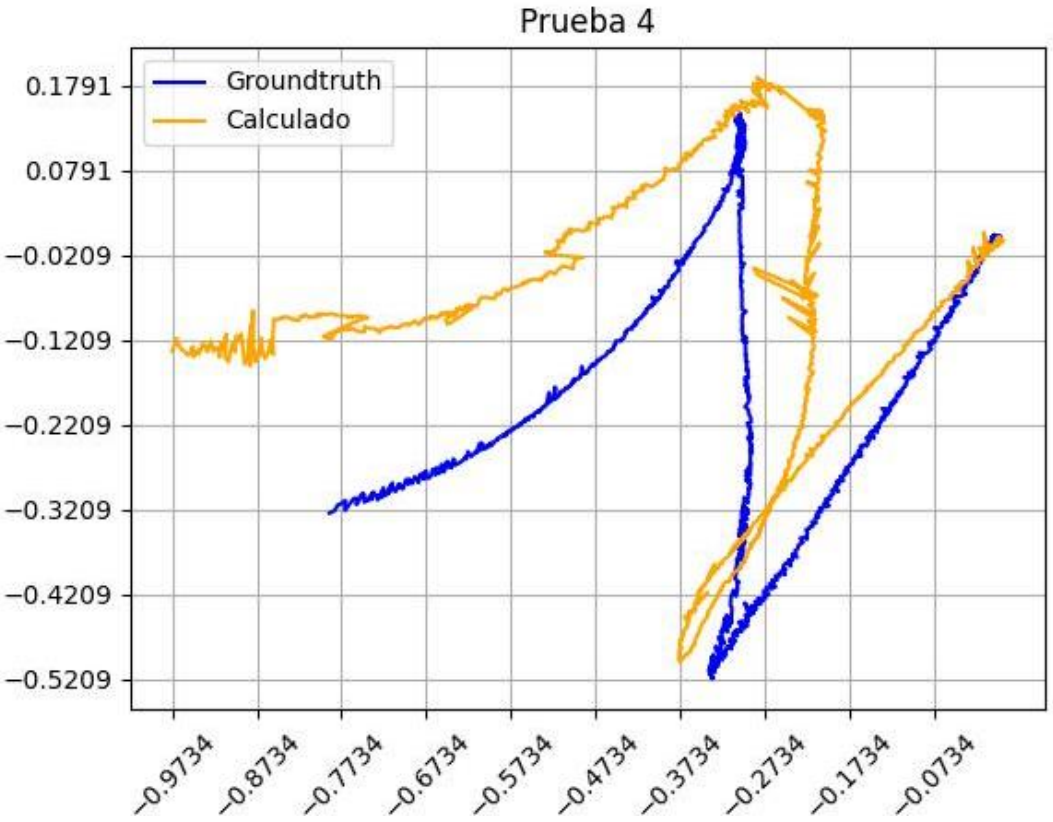


Ilustración 129-Resultados posiciones globales Kobuki y groundtruth Prueba 4

Elaboración propia

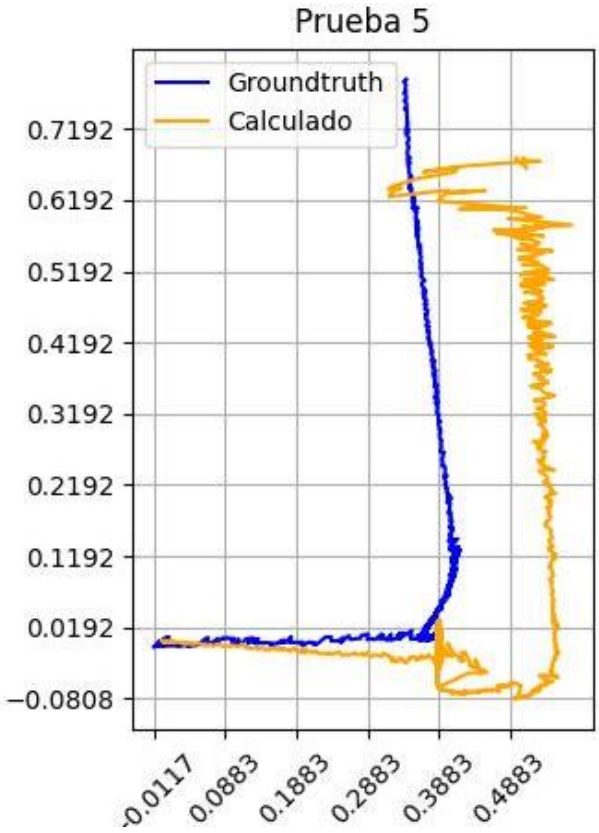


Ilustración 130-Resultados posiciones globales Kobuki y groundtruth Prueba 5

Elaboración propia

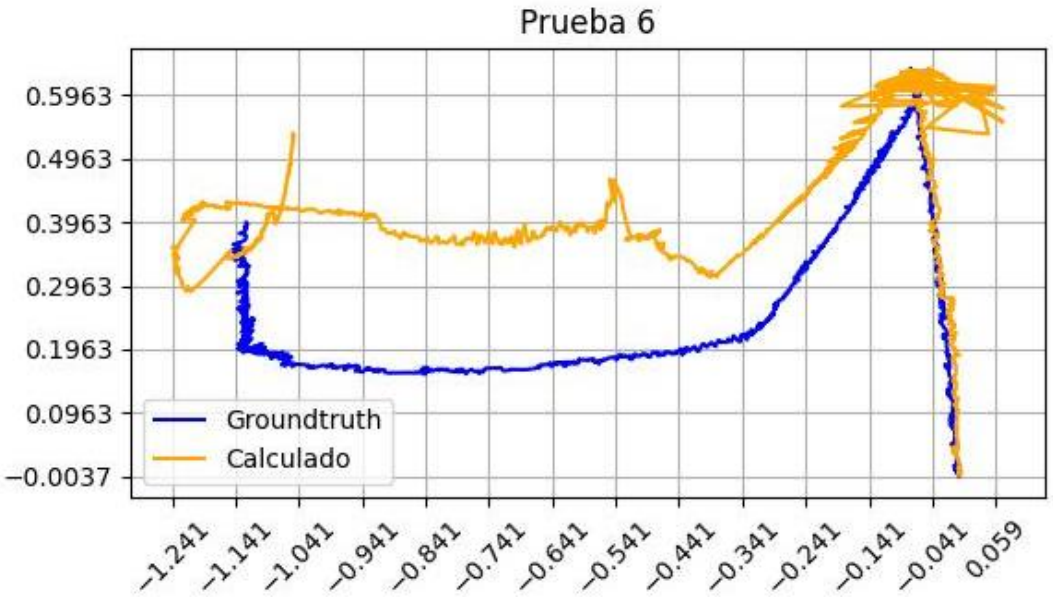


Ilustración 131-Resultados posiciones globales Kobuki y groundtruth Prueba 6

Elaboración propia

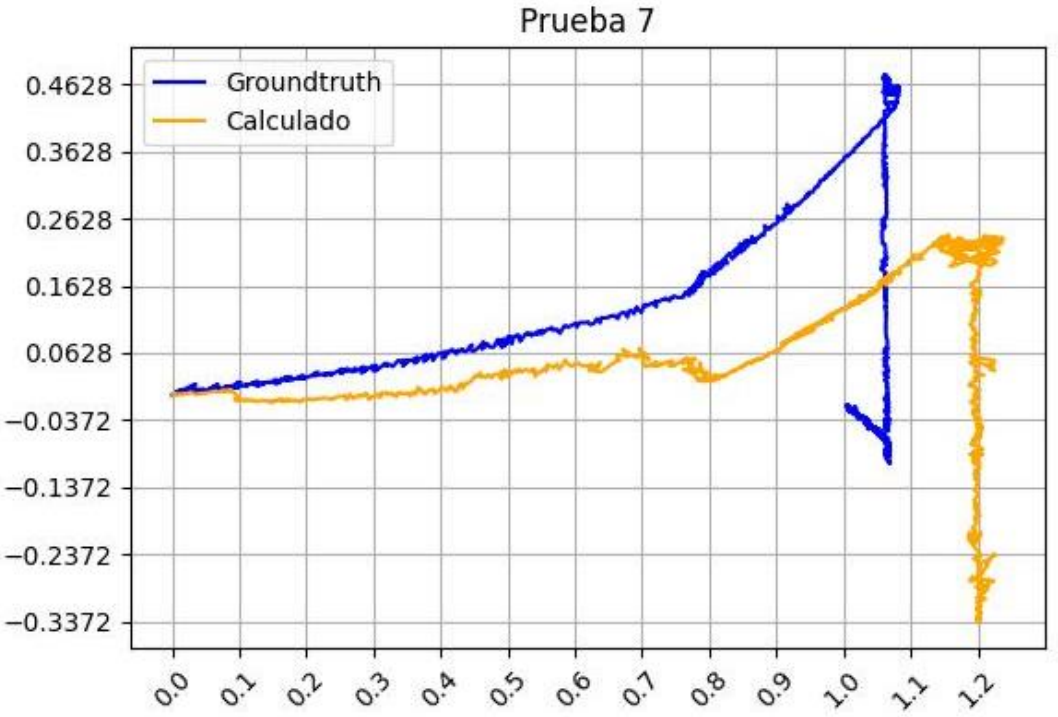


Ilustración 132-Resultados posiciones globales Kobuki y groundtruth Prueba 7

Elaboración propia

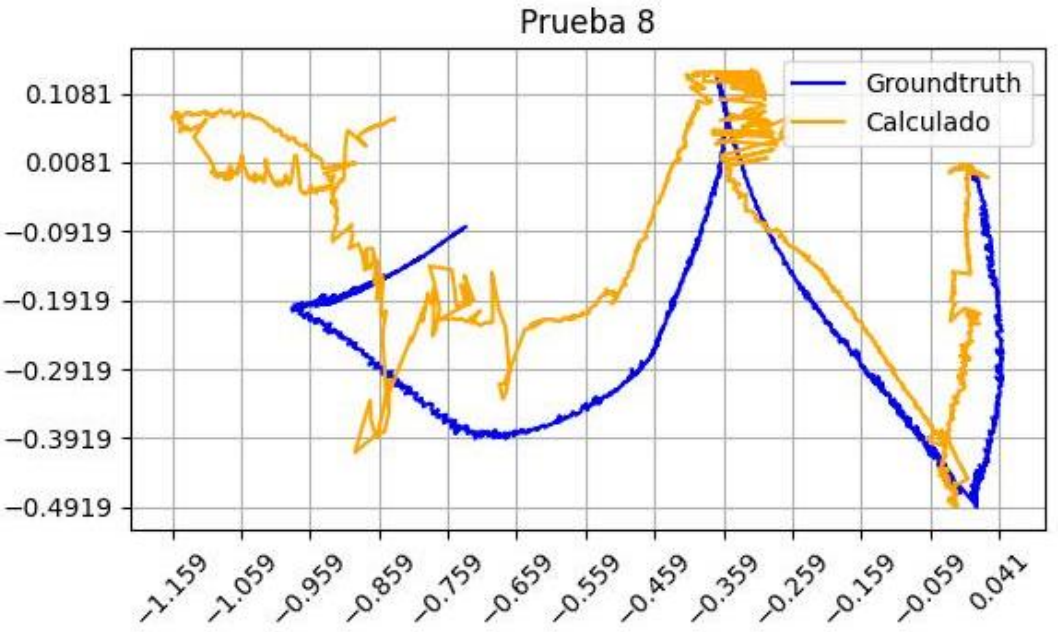


Ilustración 133-Resultados posiciones globales Kobuki y groundtruth Prueba 8

Elaboración propia

Para calcular los valores de RMSE, MSE y MAE, es necesario llevar a cabo una interpolación que permita comparar los datos en el mismo periodo de tiempo. En este proceso de interpolación, se establecen como datos fijos los valores correspondientes al *groundtruth*. La interpolación se lleva a cabo tanto para los datos en el eje X como para los datos en el eje Y, utilizando las siguientes fórmulas:

$$Fracción\ de\ tiempo = \frac{Tiempo_{real} - Tiempo\ más\ cercano\ inferior_{Kobuki}}{Tiempo\ más\ cercano\ superior_{Kobuki} - Tiempo\ más\ cercano\ inferior_{Kobuki}}$$

Ecuación 17-Interpolación fracción de tiempo

$$\begin{aligned} &Valor\ interpolado \\ &\quad = Valor\ más\ cercano\ inferior_{Kobuki} \\ &\quad + Fracción\ de\ tiempo \\ &\quad \times (Valor\ más\ cercano\ superior_{Kobuki} - Valor\ más\ cercano\ inferior_{Kobuki}) \end{aligned}$$

Luego de realizar la interpolación a cada uno de los datos del Kobuki, se aplican las fórmulas correspondientes para RMSE, MSE y MAE. La siguiente tabla contiene los resultados de cada una de las metodologías, para los valores de X e Y de cada prueba.

Número de prueba	MSE X	MSE Y	RMSE X	RMSE Y	MAE X	MAE Y
Prueba 0	0.000433	0.0106	0.0208	0.103	0.0127	0.0792
Prueba 1	0.00116	0.00968	0.034	0.0984	0.0271	0.0889
Prueba 2	0.00283	0.00122	0.053	0.0349	0.0382	0.0247
Prueba 3	0.0421	0.00716	0.205	0.0846	0.198	0.0755
Prueba 4	0.00687	0.00353	0.0829	0.0594	0.0664	0.0374
Prueba 5	0.0128	0.0165	0.113	0.128	0.091	0.108
Prueba 6	0.0059	0.0198	0.0771	0.1407	0.0613	0.105
Prueba 7	0.0101	0.0293	0.1	0.171	0.0772	0.15
Prueba 8	0.0101	0.0184	0.1007	0.136	0.082	0.084

Tabla 2-Valores RMSE, MSE, MAE para cada prueba VS groundtruth

Elaboración propia

Las metodologías analizadas en este estudio son sensibles a la escala de las variables. Al comparar los valores de RMSE, MSE y MAE, observamos que varían entre 0.000433 en MSE X de la prueba 0 y 0.205 en RMSE X de la prueba 3. Al considerar el valor máximo aceptado de tolerancia de ubicación del Kobuki con respecto al *groundtruth* (definido para el proyecto como +-0.05 m), se evidencia que estos valores son significativamente altos. De hecho, pueden ser hasta 4 veces el valor máximo aceptado.

Esta disparidad indica una baja relación entre lo calculado por el Kobuki y el *groundtruth*. En consecuencia, se concluye que no se puede confiar en la metodología de SLAM implementada en el Kobuki. Esta teoría se ve respaldada por los datos extraídos de la distancia entre las posiciones calculadas por el Kobuki y las del *groundtruth* en metros. Estos datos se presentan a continuación:

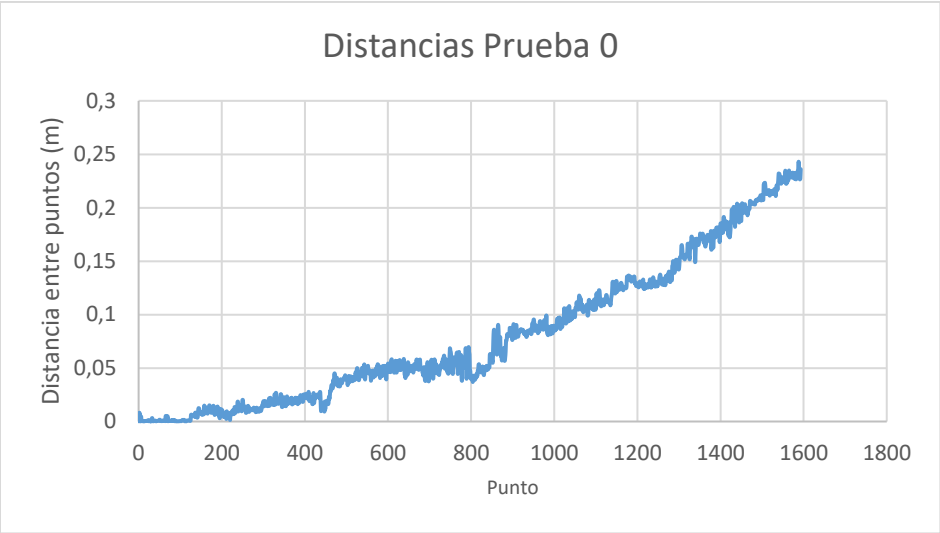


Ilustración 134-Distancia entre puntos de Kobuki equivalentes con groundtruth Prueba 0

Elaboración propia

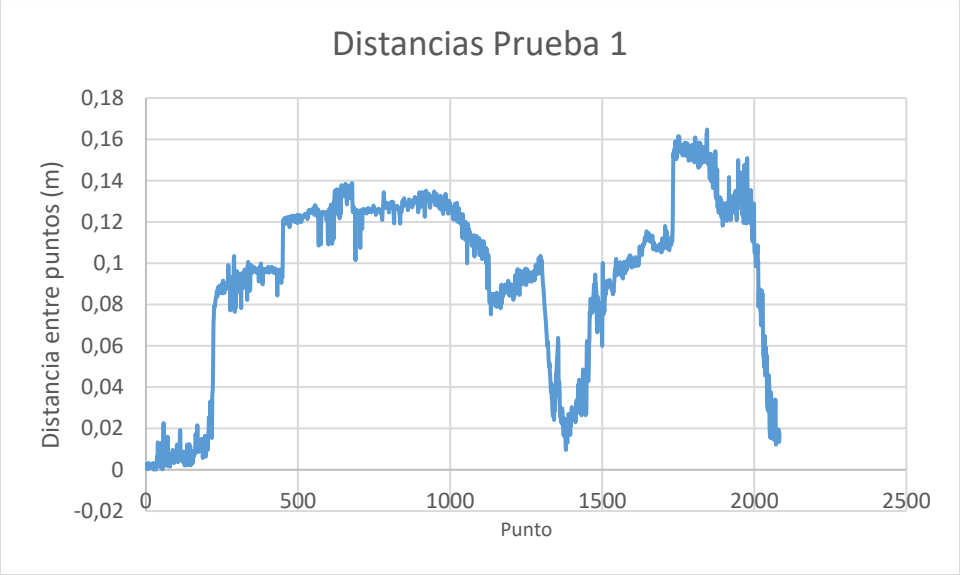


Ilustración 135-Distancia entre puntos de Kobuki equivalentes con groundtruth Prueba 1

Elaboración propia

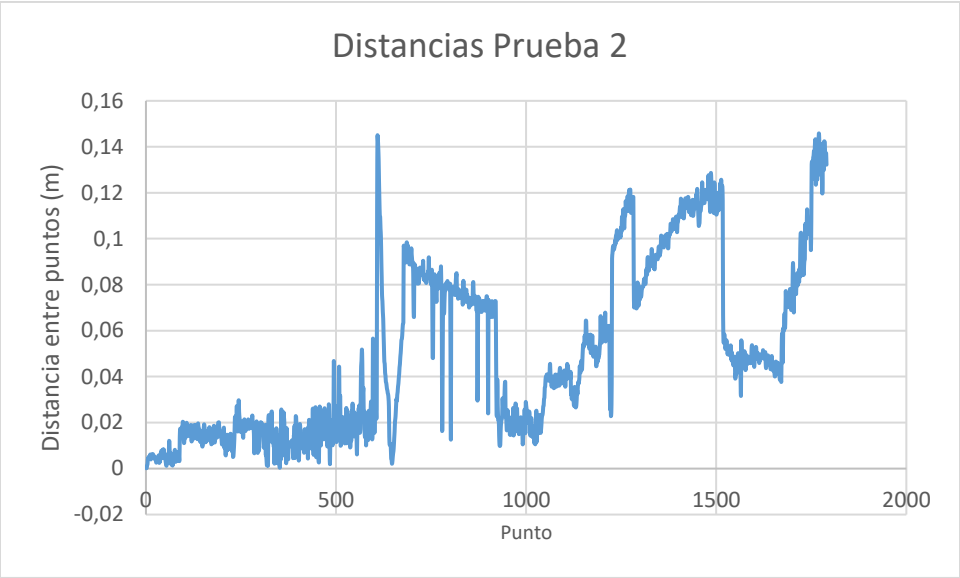


Ilustración 136-Distancia entre puntos de Kobuki equivalentes con groundtruth Prueba 2

Elaboración propia

Ilustración 137-Distancia entre puntos de Kobuki equivalentes con groundtruth Prueba 3

Elaboración propia

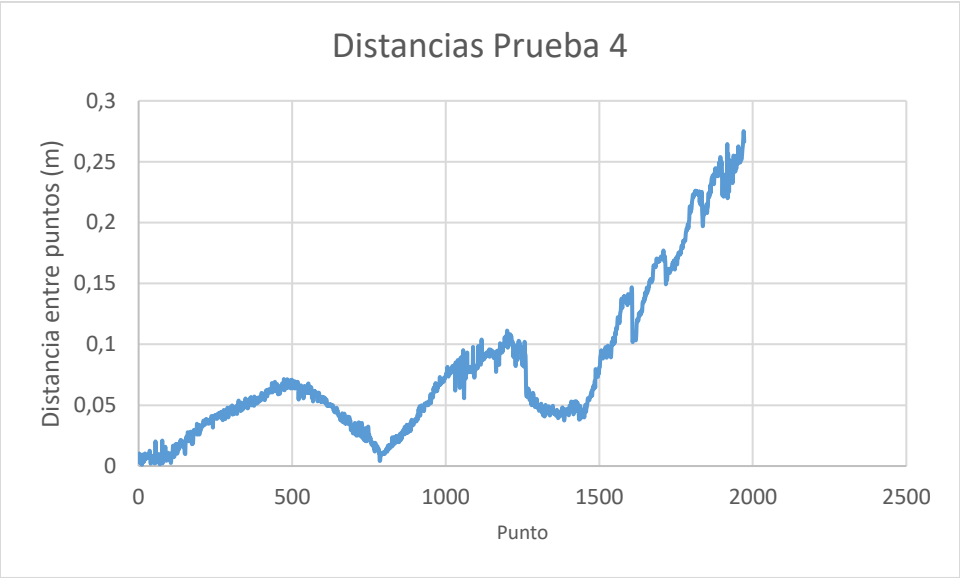


Ilustración 138-Distancia entre puntos de Kobuki equivalentes con groundtruth Prueba 4

Elaboración propia

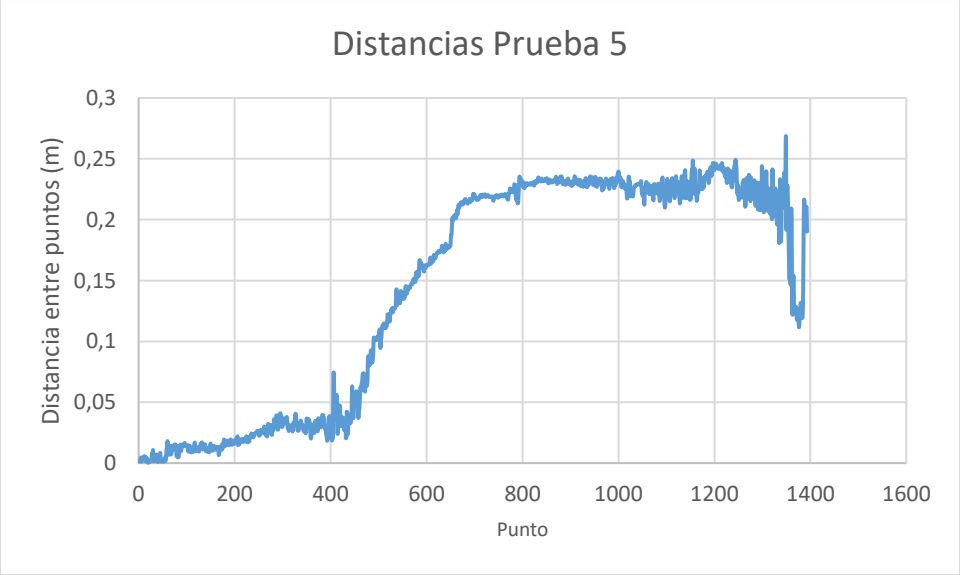


Ilustración 139-Distancia entre puntos de Kobuki equivalentes con groundtruth Prueba 5

Elaboración propia

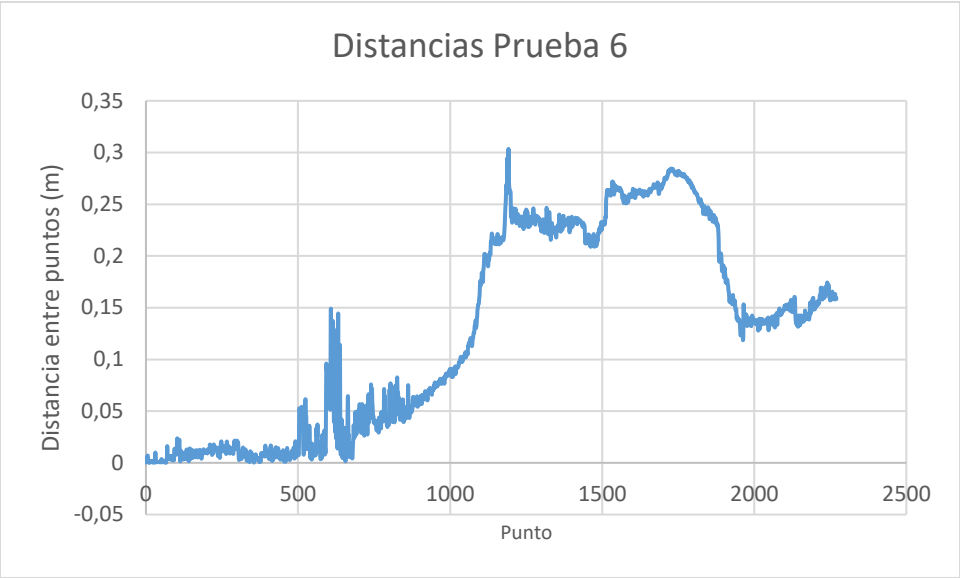


Ilustración 140-Distancia entre puntos de Kobuki equivalentes con groundtruth Prueba 6

Elaboración propia

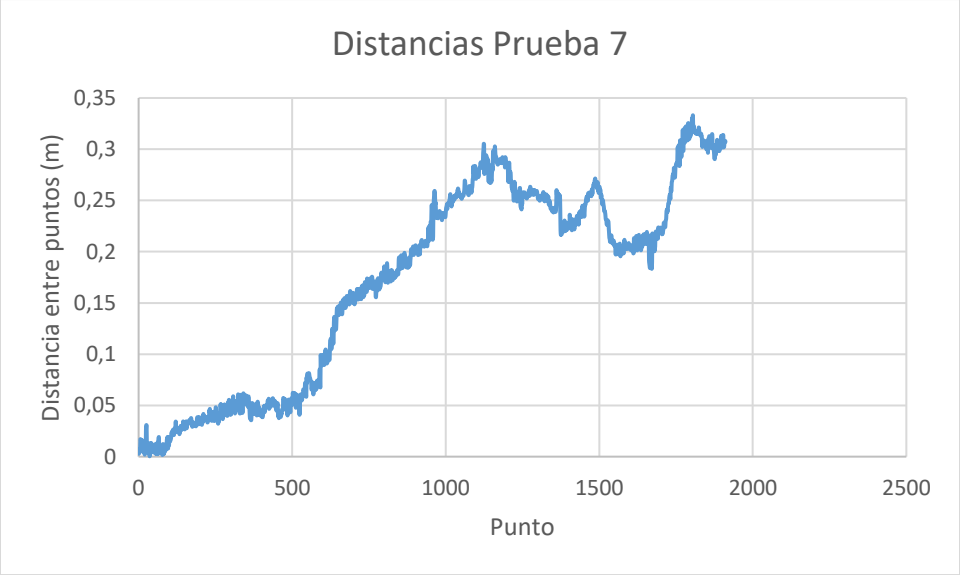


Ilustración 141-Distancia entre puntos de Kobuki equivalentes con groundtruth Prueba 7

Elaboración propia

Ilustración 142-Distancia entre puntos de Kobuki equivalentes con groundtruth Prueba 8

Elaboración propia

Se obtiene el promedio de distancias en metros entre el punto calculado por el Kobuki y su correspondiente punto calculado por el *groundtruth* en cada prueba. Además, se registra el valor máximo en metros.

Número de prueba	Promedio (m)	Máximo (m)
Prueba 0	0,08083667	0,24303528
Prueba 1	0,09506818	0,16471904
Prueba 2	0,05113129	0,14584658
Prueba 3	0,21404642	0,37408912
Prueba 4	0,07954541	0,27514063
Prueba 5	0,14388205	0,26873236
Prueba 6	0,12686733	0,30353415
Prueba 7	0,17178957	0,33312422
Prueba 8	0,1237195	0,40804735

Tabla 3-Valores promedio y máximo de ubicación de Kobuki respecto a groundtruth

Elaboración propia

Estos resultados fortalecen la conclusión de los resultados extraídos por los métodos RMSE, MSE y MAE. En términos generales, para todas las pruebas realizadas, se observa que el valor promedio de la posición del Kobuki con respecto al *groundtruth* supera la tolerancia máxima aceptada. Además, en relación con las desviaciones entre las posiciones, se evidencia que la diferencia de distancias puede llegar hasta 40 cm o incluso más en casos donde la duración del experimento se extiende. Esto indica la presencia de un error acumulativo exponencial en ciertos escenarios de prueba.

7 CONCLUSIONES

La presente investigación se sumerge en el mundo de la *Simultaneous Localization and Mapping* (SLAM), explorando la viabilidad del enfoque Visual-SLAM basado en marcadores ArUco en el contexto de la navegación autónoma de un robot TurtleBot, usando el mismo el mismo método de SLAM para el *groundtruth*, pero implementado de formas diferentes. A lo largo de este estudio, se han abordado objetivos específicos que van desde la selección de ubicaciones estratégicas para el sensor de SLAM hasta la comparación exhaustiva de los resultados obtenidos con un *groundtruth* basado en el mismo método. A medida que nos sumergimos en las conclusiones, es crucial reflexionar sobre los descubrimientos clave que han surgido de esta investigación y evaluar su impacto en el ámbito en el que se quiere implementar en la universidad EAFIT.

De los resultados obtenidos por las pruebas, se presentan las siguientes conclusiones:

- La implementación exitosa de los métodos SLAM y *groundtruth* en los sensores Xiaomi Redmi Note 8 y Moto G5 se logró gracias a la cuidadosa elección de la ubicación. El sensor en el Kobuki fue posicionado de manera que permitiera capturar una imagen con la identificación simultánea de más de un marcador, sin obstruir la visión del marcador para el *groundtruth*. La ubicación estratégica del sensor para el *groundtruth* facilitó la localización del Kobuki en la máxima área posible dentro del entorno de trabajo.
- Al analizar los resultados obtenidos para el mapa del entorno de trabajo y compararlos con los valores reales medidos, se puede concluir que el método implementado en la sección Experimento y extracción de resultados y los algoritmos implementados en ArUco mapping implementado en ROS fueron los correctos. Esto se debe a la buena precisión de la ubicación de los marcadores con respecto al sistema coordenado global.
- Implementar la metodología de interpolación es esencial para obtener ubicaciones en los mismos períodos de tiempo. Esto facilita la aplicación de las metodologías de error de manera adecuada, evitando posibles interpretaciones incorrectas de los datos y asegurando la obtención de resultados coherentes. Además, es crucial tener en cuenta la metodología de RMSE, ya que esta es más sensible a variaciones altas en los valores, proporcionando así un mejor punto de vista para procesos que requieren precisión.
- El análisis de los datos obtenidos a través del Kobuki y el *groundtruth* revela que la incorporación de un sensor tipo celular para la localización en el Kobuki no es factible. Esto se debe a que no cumple con la tolerancia máxima permitida por el proyecto. Varias razones respaldan esta conclusión:
    - **Baja señal de conexión wifi:** La presencia de una señal débil de conexión wifi impide el cálculo rápido de la gran cantidad de datos necesarios. Este inconveniente resulta en saltos de tiempo significativos entre la captura de una imagen y otra, generando imprecisiones en la posición calculada.
    - **Irregularidades del piso:** Posiblemente, la existencia de irregularidades en el terreno provoca movimientos abruptos en el Kobuki. Estos movimientos bruscos distorsionan las imágenes capturadas, generando representaciones incorrectas y, por ende, cálculos de posición inexactos.
    - **Implementación simultánea de ambos métodos en un solo computador:** La ejecución simultánea de los dos métodos en un mismo equipo limita tanto la capacidad de cálculo como la recepción de información. Esta limitación afecta negativamente la precisión y confiabilidad de la localización.
    - **Algoritmos:** El código del algoritmo kobuki_world_pos que determina la posición del Kobuki en relación con el sistema de coordenadas global difiere del utilizado para calcular las posiciones de los marcadores respecto al mismo sistema. Aunque se lograron resultados precisos al generar el mapa, la precisión en el cálculo de la

posición del Kobuki fue insatisfactoria. Por lo tanto, se sugiere seguir una metodología similar a la implementada en a_mapping.

Esta metodología implica la identificación de un marcador en más de una imagen consecutiva, permitiendo así considerar diversas perspectivas. Posteriormente, se realiza un promedio de las posiciones calculadas y se eliminan aquellas que se desvíen significativamente de la media, conforme se explica en detalle en la sección correspondiente de A_mapping.

8 TRABAJOS FUTUROS

La investigación concluyó proporcionando valiosas percepciones sobre la viabilidad y eficacia de los métodos desarrollados para SLAM basado en marcadores ArUco, en el marco del proyecto a implementar en la Universidad EAFIT. A lo largo del estudio, nos propusimos comparar el rendimiento de nuestro sistema de Visual-SLAM con un *groundtruth* basado en el mismo método, con el objetivo final de determinar su aplicabilidad para actividades de navegación autónoma con una precisión de ±5 cm.

En esta fase de la tesis, presentamos recomendaciones derivadas de nuestras observaciones y resultados experimentales:

- **Optimización del Código:**

Durante la ejecución del proyecto, desconocíamos algunas librerías de Python y el manejo de bases de datos. Ejemplo de ello es la estructura para almacenar los marcadores en formato lista en lugar de diccionario, lo cual hubiera facilitado la búsqueda de datos específicos por ID. Además, no se utilizó pandas para realizar promedios ponderados, sino que se crearon funciones. La reestructuración del código con buenas prácticas puede mejorar los resultados y reducir el tiempo de ejecución.

- **Configuración de Computadores:**

Se sugiere implementar dos computadores por separado, uno para analizar los datos del *groundtruth* y otro para analizar los datos del SLAM. En caso de no contar con dos computadores, se puede grabar videos desde ambos sensores y luego diseñar un código que utilice el video como entrada para la librería de ArUco en ROS Melodic, facilitando así la implementación de los algoritmos para *groundtruth* y SLAM.

- **Conexión por Cable para Sensores:**

Se recomienda utilizar sensores que puedan conectarse vía cable al computador en lugar de señal wifi. Esto permitirá una transferencia de datos ágil y en tiempo real, mejorando la eficiencia del sistema.

- **Implementación de Metodología A_mapping:**

Se sugiere implementar la metodología A_mapping en kobuki_world_pos, ya que los buenos resultados obtenidos con esta metodología podrían traducirse en una mayor precisión y un cálculo de SLAM mejorado.

- **Entorno de Trabajo para Kobuki:**

Es recomendable contar con un entorno de trabajo para el Kobuki que no tenga un piso de alfombra, sino de concreto. Esto evitará imperfecciones en el piso que generen saltos impredecibles del robot, evitando la pérdida de información y una mala generación de los cálculos de posición.

- **Uso de Metodología del Groundtruth para Localización del Kobuki:**

Se puede sugerir el uso de la metodología implementada para calcular las posiciones del *groundtruth* como un método de localización para el Kobuki. Esto se debe a la baja variación de los datos y la estabilidad presentada, eliminando posibles errores por desniveles en la superficie de desplazamiento del Kobuki. La cámara se mantiene estable, ofreciendo mayor estabilidad en los datos en comparación con el método implementado en el Kobuki, sustentado por la observación de los gráficos en la sección de Resultados.

9 BIBLIOGRAFÍA

[1] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid y J. Leonard, «Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age,» 2016.

[2] A. M. Barros, M. Michel, Y. Moline, G. Corre y F. Carrel, «A Comprehensive Survey of Visual SLAM Algorithms,» 2022.

[3] R. Smith y P. Cheeseman, «On the Representation and Estimation of Spatial Uncertainty,» 1987.

[4] R. Tian, Y. Zhang, Y. Feng, L. Yang, Z. Cao, S. Coleman y D. Kerr, «Accurate and Robust Object-oriented SLAM with 3D Quadric Landmark Construction in Outdoor Environment,» 2021.

[5] L. Jinyu, Y. Bangbang, C. Danpeng, W. Nan, Z. Guofeng y B. Hujun, «Survey and evaluation of monocular visual-inertial SLAM algorithms for augmented reality. Virtual Real.,» 2019.

[6] T. Taketomi, H. Uchiyama y a. S. Ikeda, «Visual slam algorithms: a survey from 2010 to 2016,» 2017.

[7] H. Ismail, R. Roy, L.-J. Sheu, W.-H. Chieng y L.-C. Tang, «Exploration-Based SLAM (e-SLAM) for the Indoor Mobile Robot Using Lidar,» 2022.

[8] Q. Zoe, Q. Sun, L. Chen, B. Nie y Q. Li, «A comparative analysis of LiDAR SLAM-based indoor navigation for autonomous vehicles.,» 2020.

[9] J. Li, J. Zhao, Y. Kang, X. He, C. Ye y L. Sun, «DL-SLAM: Direct 2.5D LiDAR SLAM for Autonomous Driving,» 2020.

[10] R. Ren, H. Fu y M. Wu, «Large-scale outdoor SLAM based on 2D Lidar. Electronics,» 2019.

[11] S. Rusinkiewicz y M. Levoy, «Efficient variants of the ICP algorithm,» 2001.

[12] P. Biber y W. Strasser, «The normal distributions transform: a new approach to laser scan matching.,» 2003.

[13] E. Olson, «Real-time correlative scan matching.,» 2009.

[14] M. Kaess, A. Ranganathan y F. Dellaert, «Incremental Smoothing and Mapping,» 2008.

[15] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard y F. Dellaert, «Incremental smoothing and mapping with fluid relinearization and incremental variable reordering,» 2011.

[16] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige y W. Burgard, «A general framework for graph optimization,» 2011.

[17] C. Wang y e. al., «Semantic line framework-based indoor building modeling using backpacked laser scanning point cloud,» 2018.

[18] Z. Wang, Y. Chen, Y. Mei, K. Yang y B. Cai, «IMU-assisted 2D SLAM method for low-texture and dynamic environments,» 2018.

[19] Zhang, S. Zhao, D. An, J. Liu, H. Wang, Y. Feng, D. Li y R. Zhao, «Visual SLAM for underwater vehicles,» 2022.

[20] J. P. Mucheroni, «A mapping of visual SLAM algorithms and their applications in augmented reality,» 2020.

[21] T. Qin, P. Li y a. S. Shen, «Vins-mono: A robust and versatile monocular visual-inertial state estimator,» 2018.

[22] J. Engel, V. Koltun y a. D. Cremers, «Direct sparse odometry,» 2017.

[23] T. Schops, T. Sattler y a. M. Pollefeys, «Bad slam: Bundle adjusted direct rgb-d slam,» 2019.

[24] J. Rodriguez y a. D. Castano-Cano, «SimSLAM 2D: A Simulation Framework for Testing and Benchmarking of two-dimensional Visual-SLAM Methods,» 2019.

[25] Rosten y T. Drummond, «Machine learning for high-speed corner detection, in: Computer Vision,» 2006.

[26] E. Rublee, V. Rabaud, K. Konolige y G. Bradski, «ORB: an efficient alternative to SIFT or SURF,» 2011.

[27] J. Wang y E. Olson, «AprilTag 2: efficient and robust fiducial detection,» 2016.

[28] A. Ligocki y A. Jel´ınek, «Fusing the RGBD SLAM with Wheel Odometry,» 2021.

[29] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart y a. P. Furgale, «Keyframe-based visual–inertial odometry using nonlinear optimization,» 2015.

[30] R. Tard´os, Mur-Artal y a. J. D., «Visual-inertial monocular SLAM with map reuse,» 2017.

[31] R. Muñoz-Salinas y R. Medina-Carnicer, «UcoSLAM: Simultaneous localization and mapping by fusion of keypoints and squared planar markers,» 2019.

[32] M. Montemerlo y S. Thrun, «Simultaneous Localization and Mapping with Unknown Data Association Using FastSLAM,» 2002.

[33] Coelho, M. &. Ahmed, K. &. Bousson y Kouamana, «Survey of Nonlinear State Estimation Based on Kalman Filtering,» 2017.

[34] S. Yavuz, Z. Kurt y M. S. Biçer, «Algoritmas Simultaneous Localization and Mapping Using Extended Kalman Filter,» 2009.

[35] e. al. y R. Muñoz-Salinas, «Mapping and localization from planar markers,» 2018.

[36] O. Wulf, A. N. uchter, J. Hertzberg y a. B. Wagner, «Ground Truth Evaluation of Large Urban 6D SLAM,» 2007.

[37] A. Nuchter, H. Surmann, K. Lingemann y J. Hertzberg, «6D SLAM with an Application in Autonomous Mine Mapping,» 2004.

[38] Nebel y G. L. a. B., «Exploring Artificial Intelligence in the New Millenium,» 2002.

[39] O. Wulf, C. Brenneke y a. B. Wagner, «Colored 2D Maps for Robot Navigation with 3D Sensor Data,» 2004.

[40] O. Wulf, K. O. Arras, H. I. Christensen y a. B. Wagner, «2D Mapping of Cluttered Indoor Environments by Means of 3D Perception,» 2004.

[41] R. Triebel, P. Pfaff y a. W. Burgard, «Multi-Level Surface Maps for Outdoor Terrain Mapping and Loop Closing,» 2006.

[42] D. Fox, S. Thrun, W. Burgard y a. F. Dellaert, «Particle filters for mobile robot localization,» 2001.

[43] H. S. §, L. Qingqing§, Y. Xianjia, J. P. n. Queralta, Z. Zou y T. Westerlund, «A Benchmark for Multi-Modal Lidar SLAM with Ground Truth in GNSS-Denied Environments,» 2023.

[44] M. J. Edwards, M. P. Hayes y R. D. Green, «High-accuracy Fiducial Markers for Ground Truth,» 2016.

[45] A. Roberts, W. N. Browne y a. C. Hollitt, «Accurate Marker Based Distance Measurement with Single Camera,» 2015.

[46] R. Yagfarov, M. Ivanou y a. I. Afanasyev, «Map Comparison of Lidar-based 2D SLAM Algorithms Using Precise Ground Truth,» 2018.

[47] G. Grisetti, C. Stachniss y a. W. Burgard, «Improved techniques for grid mapping with rao-blackwellized particle filters,» 2007.

[48] W. Hess, D. Kohler, H. Rapp y a. D. Andor, «Real-time loop closure in 2d lidar slam,» 2016.

[49] S. Agarwal, K. Mierle y e. al., «Ceres solver,» 2012.

[50] S. Kohlbrecher, O. V. Stryk, J. Meyer y a. U. Klingauf, «A flexible and scalable slam system with full 3d motion estimation,» 2011.

[51] F. Pomerleau, F. Colas, R. Siegwart y e. al., «A review of point cloud registration algorithms for mobile robotics,» 2015.

[52] J. Bayer, Petr Čížek y J. Faigl, «ON CONSTRUCTION OF A RELIABLE GROUND TRUTH FOR EVALUATION OF VISUAL SLAM ALGORITHMS,» 2016.

[53] M. Menze y A. Geiger., «Object Scene Flow for Autonomous Vehicles,» 2015.

[54] J. Sturm, N. Engelhard, F. Endres y e. al, «A benchmark for the evaluation of RGB-D SLAM systems,» 2012.

[55] T. Krajník, M. Nitsche, J. Faigl y e. al, «A practical multirobot localization system,» 2014.

[56] J. Sturm, N. Engelhard y e. a. F. Endres, «A benchmark for the evaluation of RGB-D SLAM systems,» 2012.

[57] F. Endres, J. Hess, N. Engelhard y e. al., «An evaluation of the RGB-D SLAM system,» 2012.

[58] R. Kümmerle, G. Grisetti, H. Strasdat y e. al, «A general framework for graph optimization,» 2011.

[59] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler y Andrew, «ROS: an open-source Robot Operating System,» 2007.

[60] R. Scheutz y a. M., «Development environments for autonomous mobile robots: A survey,» vol. 22, nº 2, 2007.

[61] M. Quigley, E. Berger y a. A. Y. Ng, «STAIR: Hardware and Software Architecture,» 2007.

[62] Z. Kootbally, S. Balakirsky y a. A. Visser, «Enabling Codesharing in Rescue Simulation with USARSim/ROS,» 2013.

[63] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs y R. Wheeler, «Ros: An open-source robot operating system,» 2009.

[64] J. Wang, M. Lewis, S. Hughes, M. Koes y S. Carpin, «Validating USARSim for use in HRI Research,» 2005.

[65] S. Taleghani, M. Shayesteh, S. Samizade, F. Sistani, S. Hashemi, A. Hashemi y J. Najafi, «Mrl team description paper for virtual robots competition,» 2013.

[66] L. Qingqing, J. P. n. Queralta, T. N. Gia, Z. Zou y a. T. Westerlund, «Multi Sensor Fusion for Navigation and Mapping in Autonomous Vehicles: Accurate Localization in Urban Environments,» 2019.

[67] e. al. y N. Akai, «Robust localization using 3d ndt scan matching with experimentally determined uncertainty and road marker matching,» 2017.

[68] e. al. y S. Se, «Vision-based global localization and mapping for mobile robots,» vol. 21, nº 3, 2005.

[69] e. al. y C. Premebida, «High-resolution lidar-based depth mapping using bilateral filter,» 2016.

[70] M. John., «Google cars drive themselves,» *The New York Times,* 2010.

[71] H. Flamig., «Autonomous vehicles and autonomous driving in freight ¨ transport,» 2016.

[72] e. al. y N. Boysen, «Scheduling last-mile deliveries with truck-based autonomous robots,» 2018.

[73] e. al. y J. Zhang, «Loam: Lidar odometry and mapping in real-time,» 2014.

[74] e. al. y J. Zhang, «Visual-lidar odometry and mapping: Low-drift, robust, and fast,» 2015.

[75] Afanasyev, I. Z. Ibragimov y a. I. M., «Comparison of ROS-based Visual SLAM methods in homogeneous indoor environment,» 2018.

[76] S. Kohlbrecher, J. Meyer, O. v. Stryk y a. U. Klingauf, «A flexible and scalable SLAM system with full 3D motion estimation,» 2011.

[77] A. Newman, G. Yang, B. Wang, D. Arnold y a. J. Saniie, «Embedded Mobile ROS Platform for SLAM Application with RGB-D Cameras,» 2020.

[78] Neumann, Y. Cho y J. L. Ulrich, «1.1.1 A Multi-ring Color Fiducial System and A Rule-Based Detection Method for Scalable Fiducial-tracking Augmented Reality,» 2014.

[79] J. P. Mellor, «Enhanced Reality Visualization in a Surgical Environment,» 1995.

[80] A. State, G. Hirota, D. T. Chen, B. Garrett y a. M. Livingston, «Superior Augmented Reality Registration by Integrating Landmark Tracking and Magnetic Tracking,» 1996.

[81] K. Vallino, N. Kutulakos y a. J., «Affine Object Representations for Calibration-free Augmented Reality,» 1996.

[82] S. Linnainmaa, D. Harwood y a. L. S. Davis, «Pose Determination of a Three-Dimensional Object Using Triangle Pairs,» vol. 10, nº 5, 1988.

[83] R. Horaud, B. Conio y a. O. Leboulleux, «An Analytic Solution for the Perspective 4-Point Problem,» 1989.

[84] S. MuhammadAbbas, SalmanAslam, K. Berns y a. A. Muhammad, «Analysis and Improvements in AprilTag Based State Estimation,» 2019.

[85] M. Fiala, «Comparing ARTag and ARToolkit Plus fiducial marker systems,» 2005.

[86] G. Reina, A. Vargas, K. Nagatani y K. Yoshida, «Adaptive kalman filtering for gps-based mobile robot localization,» 2007.

[87] C. Owen, F. Xiao y P. Middlin, «What is the best fiducial?,» 2002.

[88] E. Olson, «AprilTag: A robust and flexible visual fiducial system,» 2011.

[89] A. Sagitov, K. Shabalina, R. Lavrenov y E. Magid, «Comparing fiducial marker systems in the presence of occlusion,» 2017.

[90] C. Feng y V. Kamat, «Augmented reality markers as spatial indices for indoor mobile AECFM applications,» 2012.

[91] H. Jiaxin, G. Yanning, F. Zhen y G. Yuqing, «Vision-based autonomous landing of unmanned aerial vehicles,» 2017.

[92] E. Ramirez, «An Experimental Study of Mobile Device Localization,» 2015.

[93] N. Kayhani, A. Heins, W. Zhao, M. Nahangi, B. McCabe y A. Schoelligb, « Improved Tag-based Indoor Localization of UAVs Using Extended Kalman Filter,» nº 36, 2019.

[94] M. Fiala, «Comparing ARTag and ARToolkit Plus Fiducial Marker Systems,» 2005.

[95] M. Kalaitzakis, S. Carroll, A. Ambrosi, C. Whitehead y N. Vitzilaios, «Experimental Comparison of Fiducial Markers for Pose Estimation,» 2020.

[96] J. DeGol, T. Bretl y D. Hoiem, «ChromaTag: A Colored Marker and Fast Detection Algorithm,» 2017.

[97] S. Roos-Hoefgeest, I. A. Garcia y R. C. Gonzalez, «Mobile robot localization in industrial environments using a ring of cameras and ArUco markers,» 2021.

[98] P. Javierre, B. P. Alvarado y a. P. d. l. Puente, «Particle Filter Localization Using Visual Markers Based Omnidirectional Vision and a Laser Sensor,» 2019.

[99] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas y a. M. J. Marín-Jiménez, «Automatic generation and detection of highly reliable fiducial markers under occlusion,» vol. 47, nº 6, 2014.

[100] Wiki y a. -. ROS, 27 11 2018. [En línea]. Available: http://wiki.ros.org/aruco_detect. [Último acceso: 2023].

[101] Z. Wang, Z. Zhang, W. Zhu, X. Hu, H. Deng, G. He y a. X. Kang, «ARobust Planar Marker-Based Visual SLAM,» 2022.

[102] T. Qin, P. Li y S. Shen, «A Robust and Versatile Monocular Visual-Inertial State Estimator,» 2018.

[103] C. Campos, R. Elvira, J. Rodríguez, J. Montiel y J. Tardós, «ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM,» 2021.

[104] L. Ortiz-Fernandez, E. Cabrera-Avila, B. da Silva y L. Gonçalves, «Smart Artificial Markers for Accurate Visual Mapping and Localization,» 2021.

[105] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas y M. Marín-Jiménez, «Automatic generation and detection of highly reliable fiducial markers under occlusion,» 2014.

[106] D. Oberkampf, D. DeMenthon y L. Davis, «Iterative pose estimation using coplanar feature points,» 1996.

[107] R. Munoz-Salinas, M. Marin-Jimenez y R. Medina-Carnicer, «SPM-SLAM: Simultaneous localization and mapping with squared planar markers,» 2019.

[108] B. Pfrommer y K. Daniilidis, «Tagslam: Robust slam with fiducial markers,» 2019.

[109] H. Bay, T. Tuytelaars y L. V. Gool, «SURF: speeded up robust features,» vol. 3951, 2006.

[110] H. Lim y Y. Lee, «Real-time single camera slam using fiducial markers,» 2009.

[111] H. Kato y M. Billinghurst, «Marker tracking and hmd calibration for a video-based augmented reality conferencing system,» 1999.

[112] K. Madsen, H. Nielsen y O. Tingleff, «Methods for Non-Linear Least Squares Problem,» vol. 2, 2004.

[113] G. Sharp, S. Lee y D.K, «Multiview registration of 3d scenes by minimizing error between coordinate frames,» 2004.

[114] Z. Zhang, «A flexible new technique for camera calibration,» 2000.

[115] G. Sharp, S. Lee y D. Wehe, «Multiview registration of 3d scenes by minimizing error between coordinate frames,» 2004.

[116] H. C. Kam, Y. K. Yu y K. H. Wong, «An Improvement on ArUco Marker for Pose Tracking Using Kalman Filter,» 2018.