Universidad Zaragoza
1542

Trabajo Fin de Grado

Servicio de Odometría Visual para Equipos Heterogéneos de Robots

Visual Odometry Service for Teams of Heterogeneous Robots

Autor

Daniel Guiral Moneva

Directores

Jesús Bermúdez Cameo | Rosario Aragüés Muñoz

Grado en Ingeniería Electrónica y Automática
Escuela de Ingeniería y Arquitectura, Universidad de Zaragoza, año 2024

Resumen

En este trabajo se propone una implementación alternativa de un servicio de odometría para robots a partir de la información adquirida a través de dos cámaras y el tratamiento de la información que nos brindan.

Se realiza el tratamiento de la imagen para determinar por técnicas de visión por computador y redes convolucionales la presencia o no de robots en escena y su localización. Además, se integra el algoritmo SORT para realizar seguimiento individual de cada robot en escena, permitiendo así el almacenamiento separado de la información referente al cambio de posición.

De acuerdo con la información de profundidad de la escena, se generan nubes de puntos 3D donde se filtran los elementos de la escena que no nos interesan y dejando únicamente los robots detectados individualmente.

A partir de la secuencia de escenas obtenidas y filtradas, se aplican técnicas como el algoritmo ICP para el cálculo de rotación y traslación en escena del robot en cuestión.

Finalmente, se obtienen resultados de la consecución de poses que se calculan y se realiza una comparativa con las mediciones reales de la escena.

Abstract

In this project, an alternative implementation of an odometry service for robots is proposed, based on the information acquired through two cameras and the processing of the data they provide.

Image processing is performed to determine, using computer vision techniques and convolutional networks, the presence or absence of robots in the scene and their location. Additionally, the SORT algorithm is integrated to track each robot individually in the scene, allowing for the separate storage of information related to position changes.

Based on the scene's depth information, 3D point clouds are generated where non-relevant elements are filtered out, leaving only the individually detected robots.

From the sequence of obtained and filtered scenes, techniques with the ICP algorithm are applied to calculate the rotation and translation of the robot in the scene.

Finally, the results of the calculated poses are obtained and compared with the actual measurements of the scene.

Agradecimientos

Agradecer a mis tutores, Jesús y Rosario, por su inestimable contribución durante el desarrollo de este trabajo.

A toda la comunidad universitaria de la Escuela, por su buen hacer durante mi formación en el grado.

A mis padres, Alfredo y Ana Mari, por su cariño y apoyo incondicional a lo largo de esta etapa académica.

Índice

Índice de figuras

Capítulo 1: Introducción

1.1 Motivación

En el contexto actual que vivimos de digitalización, muchos son los actores productivos dentro de la industria que están adoptando nuevas técnicas que permiten mejorar la productividad y la calidad de los productos resultantes. La fabricación aditiva, impresión 3D, el Internet de las Cosas (IoT), la automatización robótica son sólo una pequeña muestra de ello. Todos ellos recogidos bajo el paraguas del término industria 4.0, estamos ante lo que muchos conocen también como la Cuarta Revolución Industrial.

Otra de las disciplinas que despierta un gran interés y que puede marcar un antes y un después en la industria es la inteligencia artificial, la visión por computador y los distintos sistemas de percepción. Gracias al desarrollo de éstos, se pretende capacitar a las máquinas para que puedan "ver" y comprender el entorno captado mediante imágenes o vídeos, pudiéndose así mejorar la interacción de las mismas con el medio.

De forma paralela, la visión por computador puede servir también como complemento a otros sistemas automatizados ya implementados para garantizar el correcto funcionamiento de ellos en un entorno cambiante.

La detección de objetos, el seguimiento de movimientos, el reconocimiento de patrones, así como el análisis de la profundidad y distancia a los objetos son distintas aplicaciones que engloban el concepto de visión por computador y que acompañarán al lector a lo largo del presente texto.

Con todo esto, este Trabajo Final de Grado pretende realizar una implementación de estas técnicas de visión con las que reconocer un grupo de robots y analizar sus respectivas posiciones relativas; en aras de complementar dicha información con la que pudiera ser calculada de manera individual por los robots mediante otras técnicas como el uso de *encoders* en los motores, entre otros.

1.2 Objetivos

A través de este proyecto, se pretende obtener un sistema de odometría visual que permita la obtención de la posición y orientación de los robots a lo largo de la escena. Para ello, se determinan una serie de objetivos:

- Identificar los métodos más idóneos para la realización del sistema de odometría visual.
- Detección en escena de robots a través del uso de aprendizaje profundo (*deep learning*).
- Seguimiento del robot Sparki (*tracking*) a lo largo de la escena.
- Combinación de la imagen de profundidad y del seguimiento para obtención de la posición 3D.
- Evaluar el funcionamiento y efectividad de la implementación.

1.3 Metodología

Para el desarrollo de todos los aspectos que componen el proyecto final y resumidos en el apartado 1.2, haremos uso mayoritariamente del lenguaje de programación de Python, así como del entorno de desarrollo de Robot Operating System (ROS) y la interfaz de programación de Pyrealsense2.

Robot Operating System es una colección de bibliotecas y de distintas herramientas, mediante las cuales se logra simplificar el proceso de creación de Software para robots. En el caso que nos atañe, haremos uso de dicho entorno como elemento gestor del almacenamiento de la información que capturemos con ambas cámaras, así como para su posterior lectura.

Pyrealsense2 consiste en una interfaz mediante la cual conseguimos comunicarnos con el SDK de Intel RealSense haciendo uso de lenguaje Python, para la obtención de imágenes RGB, imágenes de profundidad, así como otros datos como la posición relativa desde el arranque de la captura de datos. De este modo, se logra simplificar la obtención de datos desde la cámara para su posterior tratamiento.

El proyecto ha sido dividido en diferentes fases para su realización:

Una primera fase, donde se trabaja en el enfoque del trabajo. Se realiza un primer contacto con las cámaras y lectura de documentación. Definición de objetivos y desarrollo de concepto de las ideas iniciales.

En una segunda fase, se entra en la fase de implementación. Se ponen en práctica los diferentes conceptos aprendidos en la fase anterior, así como de técnicas aprendidas a lo largo del grado en temas de programación.

En esta fase también se procede con la instalación del Kit de Desarrollo de Software (SDK) y trabajo con su interfaz de programación de Python, conocido como Pyrealsense2. Se hace una puesta a punto del robot Sparki, que utilizaremos a lo largo este proyecto.

Posteriormente se trabaja para realizar la adquisición de datos con las cámaras en el SDK con la intención de tener distintas escenas con las que trabajar. Para ello, se realiza la captura y almacenamiento de datos a través de Pyrealsense2 y ROS.

A partir de la obtención de escenas almacenadas se aprende a realizar la lectura de la información almacenada en archivos ROSBAG y procesamiento de las imágenes.

Se utilizan además técnicas de *deep learning* para el reconocimiento del robot Sparki. Es importante la utilización del algoritmo SORT con el que se realizan los seguimientos de robots. Se aplica igualmente el algoritmo ICP para calcular la posición relativa. Con todo ello, se ejecuta la composición de poses, para así obtener la odometría de cada robot.

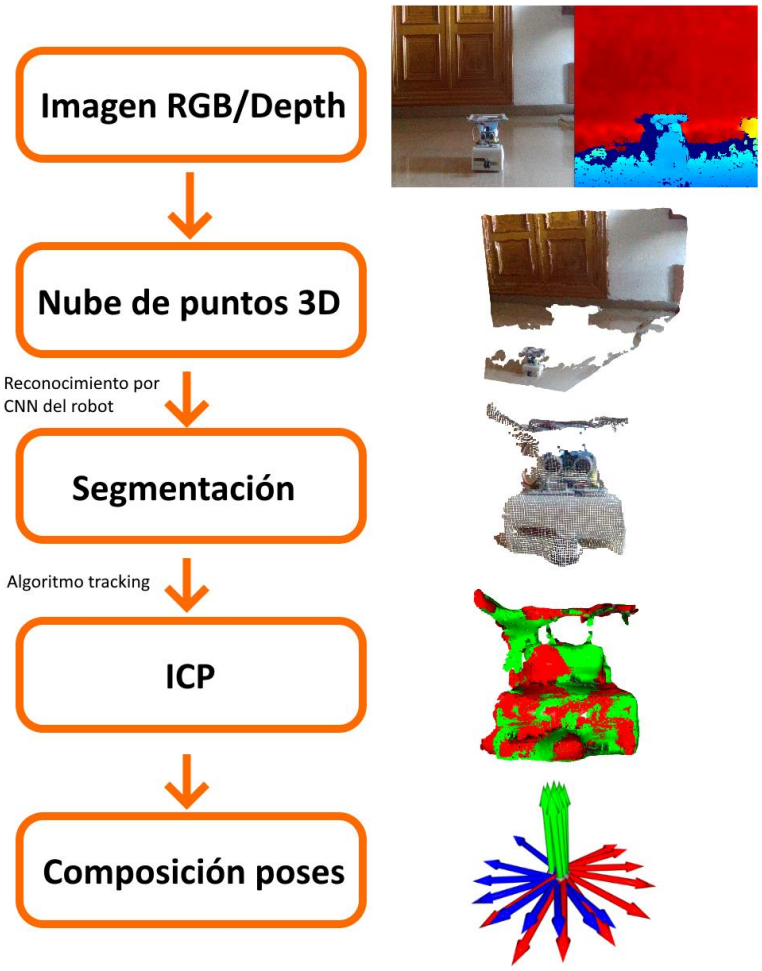

Figura 1: Diagrama de flujo con el desarrollo general del trabajo

Finalmente, en la fase tercera se recopilan los resultados obtenidos. Se hace un análisis de los mismos y se obtendrán las pertinentes conclusiones. En esta fase se terminaría la redacción de la memoria.

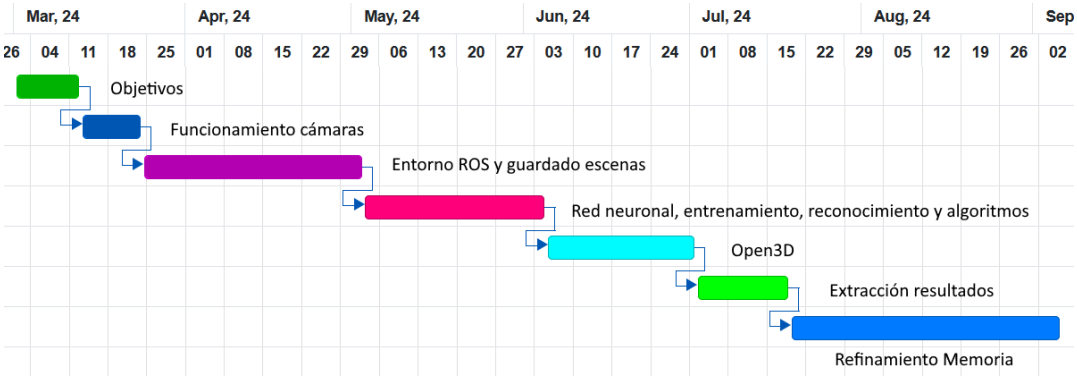

Figura 2: Cronograma realización trabajo final de grado

Para el desarrollo de este trabajo, se tomarán unas escenas de vídeo con las que se trabajará el desarrollo del sistema de odometría. Asimismo, se utilizarán para el desarrollo de esta memoria para extraer las conclusiones pertinentes.

1.4 Estructura de la memoria

En la presente sección se indica las diferentes partes en las que se encuentra dividido este documento:

Capítulo 1: Introducción → Se trata del capítulo actual. En éste, se detalla la motivación que lleva a realizar este proyecto, los objetivos, la metodología aplicada y la estructura de la memoria.

Capítulo 2: Recursos técnicos y software utilizado → En este capítulo se detallan los dispositivos utilizados, funcionalidades y características. Indica también el uso que se le otorga en este proyecto, delimitando así su finalidad. Se aprovecha asimismo para indicar el software utilizado para el desarrollo del proyecto, así como para su funcionamiento.

Capítulo 3: Conceptos teóricos → En el capítulo 3 se indican las bases en las que fundamentamos el presente trabajo, explicando al lector los conceptos teóricos que sustentan el funcionamiento del proyecto. De esta forma, se entra en el fondo de ciertos algoritmos explicando su funcionamiento y aplicación.

Capítulo 4: Sistema de odometría visual → En este apartado se explica de forma detallada el desarrollo de cada una de las funciones que componen nuestro programa. Se detallan así las necesidades a ir solventando y las soluciones propuestas ante ellas. Se explica también el uso de alguna librería, así como del funcionamiento interno de ellas.

Capítulo 5: Validación del método → Una vez alcanzado los hitos de este trabajo, se realiza un análisis del funcionamiento de cada algoritmo, analizando su rendimiento, entre otros.

Capítulo 6: Experimentos reales → Se presentan distintas escenas donde se evalúan las capacidades de la herramienta desarrollada en escenas grabadas.

Capítulo 7: Limitaciones → Se indica las limitaciones que presenta la aplicación.

Capítulo 8: Conclusiones y posibles adiciones futuras → En este último apartado se extraerán las pertinentes conclusiones a partir de los resultados analizados en capítulos anteriores. También se comentarán posibles adiciones futuras al sistema.

Se incluirán además los apartados de referencias, donde se especifica la literatura y fuentes consultadas para la realización de este proyecto, así como de los anexos.

Capítulo 2: Recursos técnicos y software utilizado

2.1 Recursos técnicos

2.1.1 Cámara Intel RealSense D435

Se trata de una cámara RGB-D desarrollada por Intel y destinada para su uso principalmente en aplicaciones de visión por computador, robótica y escaneo 3D, entre otros.

Figura 3: Cámara Intel RealSense D435

Este tipo de cámaras goza de popularidad en el ámbito los sistemas de percepción, al poder tomar capturas tanto de una imagen a color (RGB), así como de la captura de la profundidad (*Depth*) desde un mismo dispositivo, pudiendo así tener una percepción mucho más acertada del entorno. Otra de las aplicaciones en la que es más utilizada es la reconstrucción 3D.

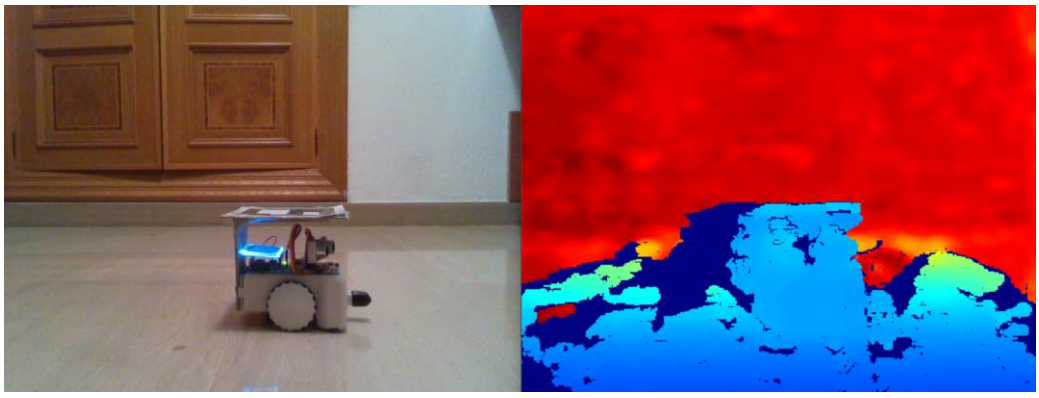

Figura 4: imagen RGB y depth obtenidas a través de la cámara D435

Con respecto a la captura de imágenes RGB, la cámara posee una resolución de hasta 1920 x 1080 a 30 imágenes por segundo.

Para la lectura de la profundidad, esta cámara utiliza la proyección de luz infrarroja estructurada sobre los objetos, permitiéndole así la lectura de distancias precisa. Tiene un rango de funcionamiento desde los 28 centímetros aproximadamente, hasta longitudes de 10 metros.

El conexionado de la cámara se realiza a través de USB 3.0, lo que facilita su compatibilidad con una gran variedad de sistemas.

2.1.2 Cámara Intel RealSense T265

Se trata de otro modelo de cámara de la misma serie Realsense. Esta cámara está diseñada para aplicaciones de localización en robot. Esto es consecuencia de llevar integrada un IMU (siglas del inglés, correspondientes con *Unidad de Medición Inercial*), mediante el cual se realiza la captura de la información referente al movimiento que tiene ésta misma, así como la de sus objetos anexos mientras se realizan la captura de datos. Por tanto, con ella podemos conocer el movimiento que realiza tanto en la rotación como en la traslación en el espacio tridimensional (6 grados de libertad).

Figura 5: Cámara Intel RealSense T265

Otra de las características de esta cámara son la inclusión de dos cámaras de ojo de pez. Se tratan de cámaras de gran angular que permiten capturar imágenes con un campo de visión amplio.

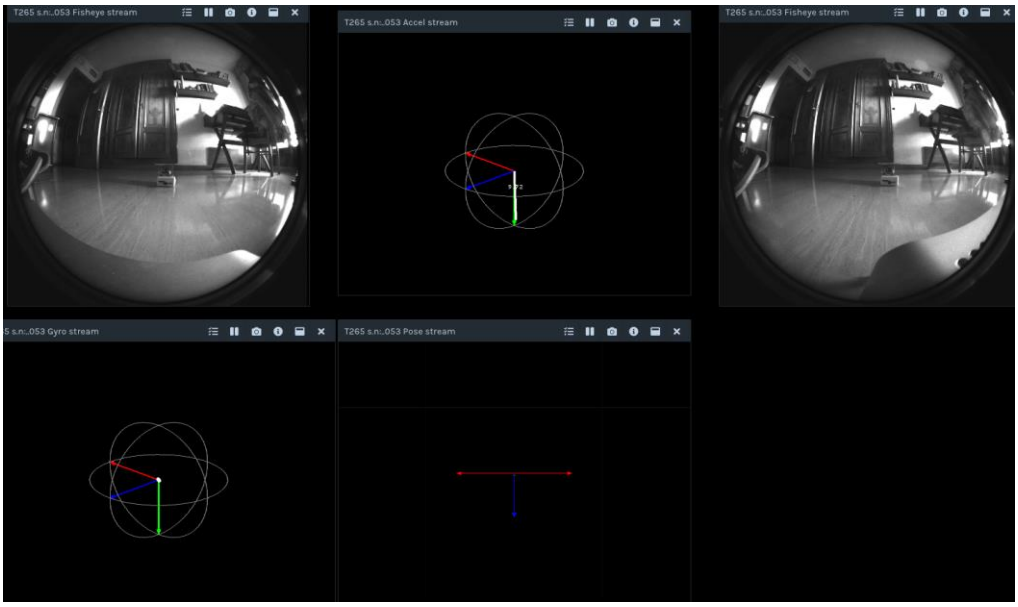

Figura 6: SDK de Intel RealSense, información de cámara con SLAM y cámaras de ojo de pez

De la misma forma que la cámara D435, la conexión a la misma se realiza mediante USB 3.0.

En nuestro caso, el propósito principal de la inclusión de la cámara Intel RealSense T265 es la de la localización y conocimiento del cambio de orientación y posición que puede llevar a cabo nuestro sistema de odometría durante la toma de datos.

2.1.3 Robot Sparki de ArcBotics

Sparki es un robot compacto educativo concebido para su uso en el aprendizaje de los conceptos de programación y robótica. Cuenta con una variedad de sensores y accionadores integrados, tales como el sensor de infrarrojos, sensor de ultrasonidos, motores de corriente continua y una pantalla LCD.

La programación de este dispositivo está basada en Arduino, plataforma que goza de una buena comunidad, lo que implica una buena cantidad de recursos y documentación, la mayoría de las veces libre y en abierto.

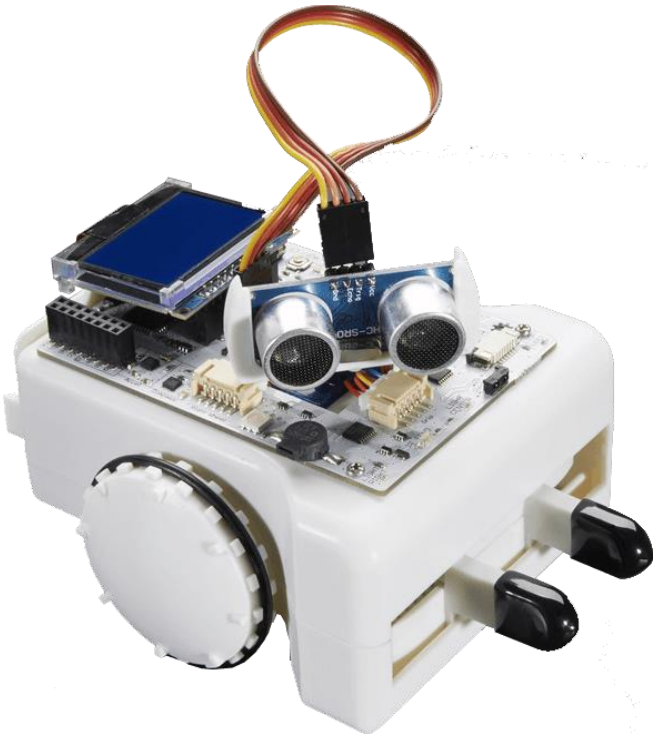

Figura 7: Robot educativo Sparki

Se trata de un robot móvil diferencial, lo que significa que está impulsado con dos ruedas, situadas cada una a un lateral del robot, en un mismo eje y siendo el giro independiente la una de la otra.

La alimentación de este robot se hace mediante pilas, y se incluye un mando para el control del robot mediante infrarrojos.

El propósito con este robot es el de ser controlado mediante dicho mando para la ejecución de pruebas y testeo del sistema de odometría.

2.2 Software utilizado

2.2.1 Entorno de Python

Como se ha comentado en la introducción, el desarrollo de la aplicación se llevará a cabo en leguaje Python. Para ello, utilizaremos la distribución de Python de Anaconda, ampliamente utilizada entre desarrolladores. La versión de Python instalada es la 3.8.3. Se recomienda la creación de un entorno virtual para la aplicación, si bien no es un requisito imprescindible.

2.2.2 SDK de Intel RealSense y Pyrealsense2

Se trata de un conjunto de bibliotecas que proporciona Intel a los creadores de aplicaciones que hacen uso de sus cámaras para que éstos puedan trabajar con ellas de manera más sencilla. De esta forma, la interacción con cualquiera de sus cámaras queda simplificada.

Se incluye también una aplicación donde visualizar directamente lo que observan las cámaras, así como la información sobre la profundidad, posición relativa en un espacio 3D etc. También permite guardar la información de una toma con cualquier cámara en formato ROSBAG para el posterior trabajo con la escena grabada.

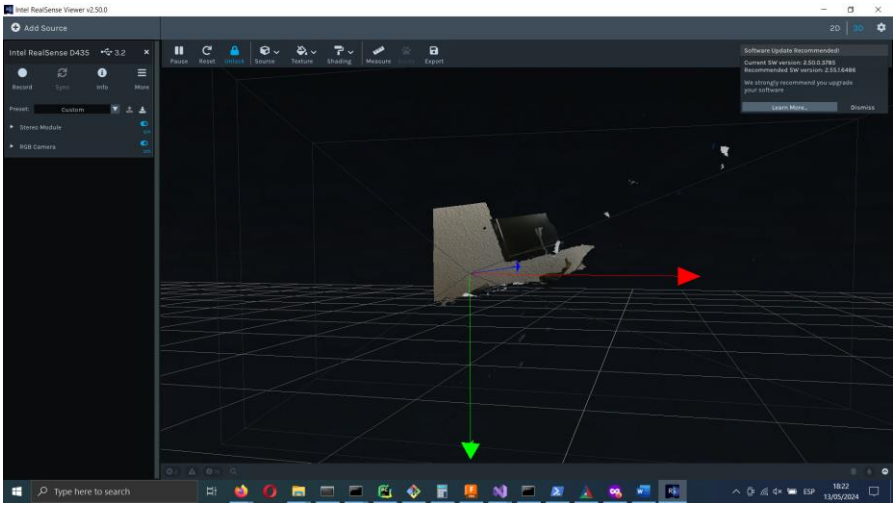

Figura 8: Interfaz visual del Intel RealSense Viewer

En nuestro caso, y para garantizar el funcionamiento de ambas cámaras con las que trabajaremos, escogemos la versión 2.50.0 del kit de desarrollo.

Como hemos comentado, el proyecto se quiere desarrollar en Python, para lo que necesitaremos una interfaz entre el SDK y nuestro entorno de Python. Para ello utilizaremos Pyrealsense2, sirviendo como puente de comunicación entre el código Python del proyecto y las bibliotecas del kit de desarrollo. De la misma forma, para garantizar el correcto funcionamiento de ambas cámaras se ha utilizado la versión 2.43.0.3018.

2.2.3 Robot Operating System

Se trata de un conjunto de herramientas y bibliotecas de código abierto creados para la simplificación del desarrollo de la programación y control de robots. Tiene como objetivo crear un estándar para la programación de robots.

Para este proyecto utilizaremos la versión 1.15.9 de ROS.

ROS se basa en la modularización por áreas de los distintos temas que forman un proyecto. Estos nodos se comunican entre sí a través de mensajes, basados en tópicos.

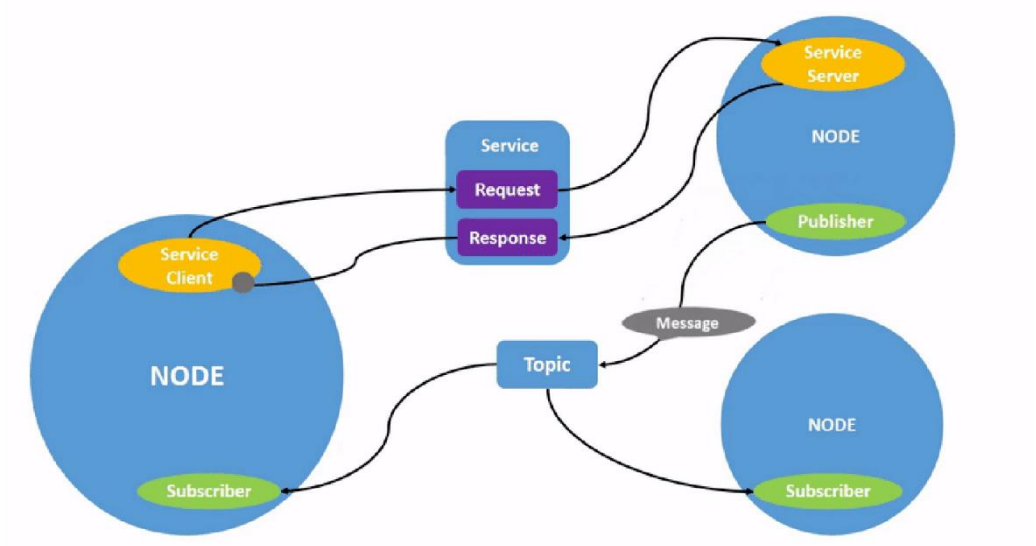

Figura 9: Representación de la modularización con la que trabaja ROS (Fuente: ROS Documentation)

Es importante destacar la posibilidad que ofrece ROS para instalar paquetes con código y configuraciones ya implementados por terceras personas, lo que puede llegar a simplificar parte del trabajo a realizar por parte del desarrollador.

En nuestro caso particular, el uso de ROS viene motivado para almacenar de manera ordenada la información que captamos de ambas cámaras Intel RealSense, así como para su pertinente lectura. El formato de archivo que se genera durante la grabación es conocido como ROSBAG y en él quedan registrados los fotogramas, información de la orientación, traslación de la cámara y demás.

2.2.4 PyTorch

Se trata de una biblioteca de código abierto desarrollada por Meta (Facebook) destinada al aprendizaje automático. Es ampliamente utilizada por desarrolladores para la implementación en modelos de aprendizaje profundo (*Deep Learning)*, dada su facilidad de uso mediante lenguaje Python.

Figura 10: Logotipo de la biblioteca PyTorch (Meta)

PyTorch está basado en Torch, conservando así la mayoría de las características de éste e incluyendo la interoperabilidad en Python con otras herramientas. Destaca por la posibilidad de uso de unas estructuras de datos llamadas tensores, estructura similar a la que proporciona Numpy, pero que además incluye el soporte con el que poder ejecutar las operaciones sobre la tarjeta gráfica (GPU). Esto se traduce en un aumento del rendimiento de cara al usuario.

Otra de las características más destacadas de PyTorch son los gráficos computacionales dinámicos. Permiten la construcción de modelos y redes neuronales de manera flexible y adaptable. De esta forma, se logra simplificar el desarrollo de nuevas ideas y modelos frente a nuevas posibles adaptaciones.

PyTorch incluye también Autograd, sistema por el cual se simplifica enormemente el cálculo de los gradientes de la función de pérdida. Éstos son utilizados en la retropropagación (con la llamada de *backward()* en Python) durante el proceso de entrenamiento de la red, indicando cómo cambiar cada parámetro para reducir el error. Por tanto, forma parte del cálculo en un proceso esencial por el cual se ajustan los pesos en las conexiones de la red neuronal.

Además de todo ello, existen varias bibliotecas construidas a partir de esta base, en las que se incluyen herramientas especializadas en distintos campos. TorchVision, especializada en el procesamiento de imágenes, proporciona varias utilidades como conjuntos de datos, modelos preentrenados y transformaciones

para el aumento y procesamiento de imágenes. Tenemos también TorchText, utilizado en el campo de lenguaje natural. Diseñado para el procesamiento de elementos de texto, incluye diversas utilidades al llevar intrínseco una variedad de conjuntos de datos (*datasets*), vocabulario, entre otros. Por último, hay que destacar también la existencia de TorchAudio, que proporciona herramientas de procesamiento de audio.

En el caso que atañe a este proyecto, se hará uso de TorchVision para cargar un modelo preentrenado y la transformación a tensores de imágenes, entre otros.

Capítulo 3: Conceptos teóricos

3.1 Recogida de datos de profundidad

La cámara Intel RealSense D435 tiene una serie de sensores utilizados para captar la información de la profundidad de la escena. Esta información es obtenida a través de la combinación de visión estéreo y de sensorización de infrarrojos.

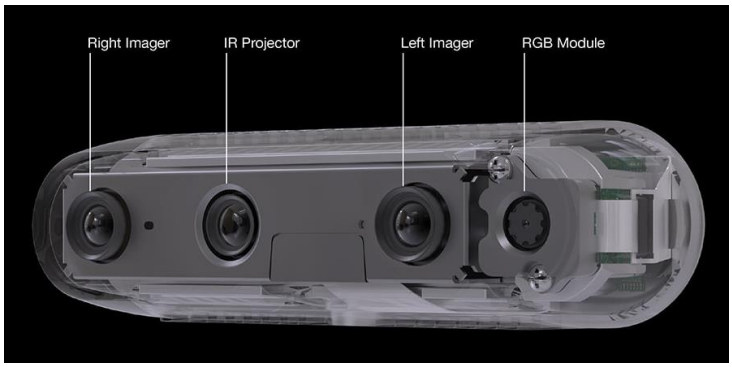

Figura 11: Composición interna de la cámara Intel RealSense D435 (Fuente:Intel)

La cámara incorpora dos cámaras de infrarrojos (Right Imager y Left Imager en la figura 11), posicionadas a cierta distancia respectivamente. Dicha distancia ocasiona que ambos objetivos capten dos instantáneas ligeramente distintas de la misma escena que enfocan. A través del proceso conocido como emparejamiento estéreo, el dispositivo calcula la diferencia de posición de los objetos de la escena entre instantáneas. Dicha información es utilizada para calcular la profundidad de los puntos en la escena. Este proceso está basado en los principios de triangulación en cada punto de la escena.

La fórmula con la que obtener la profundidad de un punto en la escena es:

$$Z = \frac{f\ B}{(x_l - x_r)} = \frac{f\ B}{disparidad}$$

Siendo f la distancia focal de las cámaras, B la distancia entre cámaras (también conocido como línea base). La disparidad es la diferencia entre puntos correspondientes en las imágenes en la horizontal.

Objetos que se encuentren próximos a los objetivos tendrán una disparidad alta, mientras que los objetos situados lejos, su disparidad será reducida.

Por otra parte, la cámara Intel RealSense D435 incluye un proyector de infrarrojos. Este se encuentra entre las cámaras IR que nos hemos referido para la triangulación, como se puede ver en la figura 11.

La función de este componente es la emisión de un patrón de luz infrarroja en la escena. Este patrón es utilizado para mejorar la calidad de la respuesta sobre todo en escenas donde la textura sea limitada y existan un gran número de superficies lisas y sin variaciones de color, entre otros factores.

De esta forma, el patrón proyectado añade una textura artificial a la escena con la que crear puntos de referencia adicionales que se utilizarán en el emparejamiento estéreo comentado.

3.2 Simple Online Real Time

Simple Online Real Time, conocido generalmente por sus iniciales (SORT), es un algoritmo que permite el seguimiento de personas y objetos en vídeos. Es popular su uso, dada su simplicidad y eficiencia, frente a otros algoritmos más complejos.

Una de sus principales características radica en su funcionamiento, basado en el filtro de Kalman y en el algoritmo húngaro. Por una parte, el filtro de Kalman sirve para realizar la predicción de la posición futura de los objetos detectados, a partir de la trayectoria y posición previa de los objetos detectados que se encuentran dentro del campo de visión de la cámara. De esta manera, si en algún fotograma en concreto el modelo no se llegase a detectar el objeto detectado en los fotogramas anteriores, se podría presuponer su trayectoria en el vídeo. Por otra parte, el algoritmo húngaro realiza emparejamientos de manera eficiente entre las detecciones de los objetos del último fotograma y las detecciones anteriores de fotogramas previos para determinar si se trata del mismo objeto. De esta forma, conseguimos que el objeto se contabilice únicamente en una ocasión, así como para garantizar que realizamos el tracking correctamente.

El funcionamiento de SORT es el siguiente: primero de todo, se hace uso de cualquier modelo de detección de objetos sobre un fotograma. De esta forma conoceremos la posición en coordenadas delimitantes de los objetos que detectamos. A continuación, se aplicaría el filtro de Kalman para predecir la posición de los objetos en el siguiente fotograma, basado en las posiciones y trayectorias anteriores de los mismos. Después, se realiza la asignación. Este proceso hace uso del algoritmo húngaro, de tal forma que podamos relacionar las detecciones de objetos del siguiente fotograma con los de fotogramas anteriores. Es lo que conocemos como *tracking* o seguimiento. El paso final consiste en la actualización del estado del filtro de Kalman para cada objeto. Se crearán y eliminarán referencias a las detecciones según las condiciones del vídeo.

SORT es utilizado por tanto en un amplio campo de aplicaciones, pudiendo ser utilizado en aplicaciones de vigilancia, robótica o incluso en vehículos autónomos.

A continuación, se detalla las principales características del filtro de Kalman y del algoritmo húngaro:

3.2.1 Filtro de Kalman

Se trata de un algoritmo que se basa en el modelo de espacio de estados en aras de estimar el estado futuro, así como la salida futura, a partir de medidas ruidosas o incompletas.

Dependiendo del retraso de las muestras recibidas de instantes anteriores, puede cumplir la función de estimador o simplemente la de filtro. Su principal ventaja es que puede calcular la estimación de estado en tiempo real, actualizando la estimación conforme se reciben nuevas medidas.

Desarrollado por Rudolf E. Kalman, es ampliamente utilizado en tareas de control y estimación de sistemas, así como en el procesamiento de señales.

La dinámica del proceso se sucede en dos fases diferenciadas. Suponiendo el caso de tiempo discreto, determinamos las siguientes ecuaciones:

la primera fase, denominada fase de predicción, es el paso donde se estima el estado en el instante siguiente.

$$\hat{x}_k = \mathrm{A}\hat{x}_{k-1} + \mathrm{Bu}_{k-1}$$

Siendo $\hat{x}_k$ los valores de estado en el instante *k*, u_{k-1} las entradas en el instante anterior *k-1*.

En esta fase, además, se calcula también la predicción de la covarianza del error:

$$P_k = AP_{k-1}A^T + Q$$

Siendo P_k la covarianza del error en el instante *k* (a posteriori) y *Q* es una matriz diagonal relativa al ruido del proceso.

Las matrices *A* y *B* corresponden con la matriz de función de transición de estado y la matriz de control, respectivamente.

En la segunda fase, llamada fase de actualización, el filtro de Kalman renueva los valores de la estimación del estado y de la covarianza del error.

Primeramente, se calcula la ganancia de Kalman K_k, que determina cómo se combinan la predicción y la medida observada. Se obtiene a partir de la siguiente ecuación:

$$K_k = P_k H^T (HP_k H^T + R_k)^{-1}$$

Donde *H* es la matriz de observación y *R* es una matriz diagonal relativa al ruido muestral.

A continuación, se realiza la actualización del estado $\hat{x}_k$, así como de la covarianza del error:

$$\hat{x}_k = \hat{x}_k + K_k(z - H\hat{x}_k)$$

$$P_k = (I - K_k H)P_{k-1}$$

Siendo *z* el vector de mediciones. Su obtención viene marcado por la siguiente fórmula:

$$z_k = H_k x_k + v_k$$

3.2.2 Algoritmo húngaro

Se trata de un método combinatorio que ayuda a revolver problemas de asignación. Desarrollado por Harold Kuhn en el año 1955, fue posteriormente mejorado por James Munkres en el año 1957.

Se utiliza para encontrar una asignación óptima de un conjunto de recursos a un conjunto de tareas, de manera que el coste total sea mínimo.

Se trabaja sobre una matriz de costes C_{ixj}, donde las filas corresponden con los agentes y las columnas las tareas. Cada uno de los valores en la matriz indica el coste de asignación de la tarea *j* al agente *i*.

El algoritmo trabaja realizando una serie de operaciones de tiempo polinomial O (n^3) que finaliza cuando la integridad de las filas tienen asignado un elemento de valor 0 que no ha sido asignado previamente a ninguna otra de las filas.

3.3 Redes neuronales convolucionales

Las redes neuronales convolucionales, conocidas también del inglés como CNN (*Convolutional Neural Network*), consisten en un tipo de red neuronal artificial especializadas en el procesamiento de imágenes, así como de sonido.

Tienen su inspiración en el funcionamiento del córtex visual humano, en aras de obtener un comportamiento similar al mismo. Como su nombre indica, un proceso fundamental radica en la operación matemática de la convolución. A través de la convolución, se logra una transformación de los datos de la imagen de forma que se resalten ciertas características de la misma.

Las CNN se componen de una serie de capas de nodos, llegando a estructurarse de la siguiente forma: una capa de entrada, una o más capas ocultas (pudiendo llegar a ser de hasta cientos) y una capa final. Cada una de ellas realiza operaciones de distinta índole, entre ellas la mencionada ya de

convolución, agrupación (Pooling) o de activación (ReLU, entre otras). Cada una tiene una función definida y de esta forma al pasar las imágenes por la red, se consigue extraer distintas características de las imágenes.

Al tratarse de una red a la que le aportaremos un conjunto de datos etiquetados durante el aprendizaje, se considerará como una red supervisada. El objetivo por tanto es minimizar el error entre las predicciones de la red y las salidas reales.

A continuación, se explica con mayor detalle las operaciones que pueden realizar cada una de las capas convolucionales mencionadas:

- Capas de convolución: se trata del proceso principal a destacar de este tipo de redes. En este tipo de operación se hace uso de unas pequeñas matrices numéricas que se desplazan a lo largo de los píxeles que conforman la imagen de entrada. En cada posición, se realiza la multiplicación entre los valores del filtro y los de los píxeles. Posteriormente, se hace la suma de productos resultantes de las operaciones anteriores.
- Capas de agrupación (*Pooling)*: son utilizadas con el propósito de reducir la dimensionalidad de las características extraídas. De esta manera, se logra reducir el tamaño de la imagen.

  Los tipos más comunes de agrupación son el Max Pooling y el Average Pooling. Consisten en generar la ventana de un tamaño definido y seleccionar el píxel de mayor valor, o bien el promedio de estos, respectivamente.
- Capas de activación: son las encargadas de introducir la no linealidad en la red convolucional. Esto es crucial ya que, sin esta no linealidad, una red neuronal profunda sería equivalente a una red neuronal con una sola capa de neuronas lineales, sin importar cuántas capas se añadiesen. La no linealidad permite que la red aprenda y modele relaciones complejas en los datos. Existen una serie de funciones de activación utilizadas comúnmente, tales como ReLU, la función sigmoide o la tangente hiperbólica, con las que se obtiene dicha no linealidad.

3.4 Algoritmo de punto iterativo más cercano (ICP)

Se trata de una técnica utilizada frecuentemente en los campos de visión por computador. Consiste en alinear dos nubes de puntos pertenecientes al mismo objeto, siendo en nuestro caso las nubes de puntos consecutivas detectadas entre dos fotogramas de la cámara.

Durante este proceso, se repite iterativamente el proceso de estimación de la transformación hasta alcanzar un criterio de convergencia que normalmente viene estipulado por el usuario, bien en número máximo de iteraciones o bien cuando la transformación sea mínima.

De esta forma, se obtiene la transformación que minimiza la distancia entre dos nubes de puntos.

Este algoritmo es muy útil para aplicaciones como la reconstrucción 3D, la navegación de robots, el seguimiento de objetos, y la extracción de la matriz de rotación y traslación, entre otros.

Existen varias maneras de aplicar el método. El primero es el conocido como punto a punto: en esta se calcula la función de error en función de las distancias entre puntos emparejados. La segunda aplicación del método es conocido como punto a plano. En este caso, se tiene en cuenta la orientación de la superficie modelada mediante el cálculo de la distancia entre un punto y el plano tangente en el punto correspondiente del modelo.

Para la implementación de estas iteraciones es pertinente la eliminación previa de puntos espurios que pueden corresponder a objetos de fondo, paredes, o el propio suelo.

3.5 Parámetros de la cámara

Un concepto importante a tener en cuenta para que las medidas que se obtengan en este proyecto sean lo más precisas posibles, es la utilización de los parámetros intrínsecos de calibración de la cámara. Dichos parámetros reflejan las modificaciones o distorsiones que se reflejan en la imagen final de resultas de adaptar la escena 3D que se fotografía para visualizarse sobre un plano 2D.

De esta forma, la utilización de estos datos para la transformación de la imagen permite interpretar de manera correcta el entorno, mejorando así las mediciones y exactitud de las imágenes tomadas para aplicaciones de visión por computador, como es en este caso.

A continuación, se indican alguna de las principales mejoras que conlleva el implementar dicha matriz sobre las imágenes tomadas:

- Se pretende corregir el error presente en las imágenes en la medición de distancias y ángulos.
- Se espera la corrección de la distorsión que es provocada por las lentes de las cámaras, en especial énfasis en los bordes de las imágenes finales.
- Para aplicaciones donde se escanean o reconstruyen escenas y objetos en 3D, es vital tener en cuenta dichos parámetros para garantizar un correcto mapeo.

De esta forma, se entiende que la matriz de calibración sirve para relacionar puntos en el espacio 3D (X, Y, Z) con sus correspondientes proyecciones en la imagen 2D en coordenadas homogéneas ($x_{cámara}$, $y_{cámara}$, w) capturada por la cámara.

La matriz intrínseca *M* viene descrita de la siguiente forma y sirve para relacionar ambas proyecciones:

$$M = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

Siendo f_x y f_y las distancias focales en píxeles en las direcciones x e y, respectivamente. c_x y c_y son las coordenadas del punto principal de la imagen.

Por tanto, la expresión que relaciona ambas representaciones queda de la siguiente forma:

$$\begin{pmatrix} x_{cámara} \\ y_{cámara} \\ w \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

Los parámetros que se han descrito vienen definidos por la construcción interna de la cámara y son estimables a partir del procedimiento de calibración de la cámara. En muchos casos también, los fabricantes aportan dichos valores para su consideración por parte de desarrolladores. Se describen más detalladamente los parámetros que lo definen:

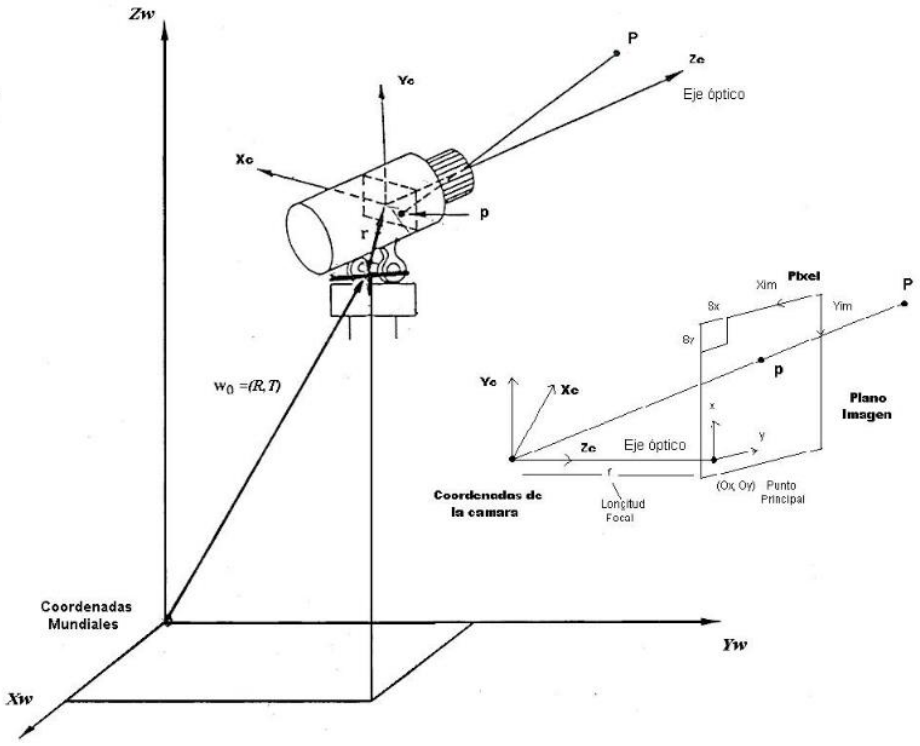

Figura 12: Modelo de cámara en proyección perspectiva [Fur et al, 1987]

- Distancia focal: se trata de la distancia entre el centro óptico de la lente de la cámara y el sensor o plano focal donde se proyecta la imagen. Dicha distancia se suele medir en milímetros (mm).

  Una distancia focal corta (del orden de los 10 a 30 mm) permiten un ángulo de visión amplio y en consecuencia se puede captar la mayoría de la escena. Por el contrario, una distancia focal larga (hasta los 300mm) tiene poco ángulo de visión y tiende a tener una mayor profundidad de campo. Suele tener su aplicación en teleobjetivos.

  En la matriz de calibración ya presentada, aparecen los términos f_x y f_y. Éstos son componentes que representan la distancia focal en términos de píxeles en ambas direcciones del plano, tanto en x como en y,

respectivamente. Se obtienen fácilmente de las siguientes expresiones:

$$f_x = \frac{f}{m_x} \qquad\qquad f_y = \frac{f}{m_y}$$

Siendo m_x y m_y los tamaños de los píxeles en cada una de las respectivas direcciones.

- Tamaño efectivo del pixel en las direcciones x e y: se tratan de los valores de m_x y m_y comentados. Éstos se pueden obtener mediante el cociente entre el tamaño de la imagen (longitud) en cada vértice, entre el número de píxeles de resolución en cada uno de los vértices. Suelen tener el mismo tamaño (píxel cuadrado), pero no es requisito obligatorio.
- Coordenadas del centro de la imagen (punto principal): corresponde con la intersección del eje óptico de la cámara con el plano de la imagen. Vienen denotados en la matriz de calibración como c_x y c_y. No tienen por qué coincidir con el centro de la imagen.
- Coeficiente de distorsión radial: se trata de un parámetro utilizado en la corrección de distorsiones surgidas por la naturaleza de la lente. En el proceso de adquisición de una fotografía, las líneas rectas pueden aparecer en la imagen como líneas curvas. Se trata de un fenómeno a evitar sobre todo en el análisis geométrico de la imagen.

Además de los parámetros intrínsecos de la cámara, han de considerarse también los parámetros extrínsecos. Éstos consisten en la identificación de la rotación y traslación de la cámara en referencia a unas coordenadas “mundo” que pueden venir identificadas para nuestro sistema.

De la misma forma que en cualquier sistema de referencia geométrico, la rotación y traslación vienen definidas a través de sus respectivos vectores que identifican dichas traslaciones de posiciones.

El vector rotación (R) se compone de una matriz 3x3, mientras que la traslación (T) se define a través de una matriz 3x1. Puede definirse por tanto la siguiente relación geométrica:

$$\begin{pmatrix} x_{cámara} \\ y_{cámara} \\ z_{cámara} \end{pmatrix} = R \begin{pmatrix} X_{mundo} \\ Y_{mundo} \\ Z_{mundo} \end{pmatrix} + T$$

3.6 RANSAC

RANSAC consiste en un algoritmo robusto e iterativo utilizado en la estimación de parámetros de un modelo matemático en un conjunto de datos que puede contener datos atípicos. Es muy utilizado en aprendizaje automático, así como en aplicaciones de visión por computador.

Su funcionamiento es simple: se toma un subconjunto aleatorio de datos de la muestra, con el que se intenta estimar un modelo. Éste puede ser el ajuste de una recta, un plano etc. A continuación, se realiza una evaluación de los datos comprobando cuántos puntos se ajustan al modelo estimado y cuántos se quedan fuera del mismo. Este proceso se realiza bien un número determinado de veces, donde se selecciona el mejor modelo, o bien se realiza el proceso hasta que se encuentra un modelo con un número aceptable de puntos acordes al modelo.

Se definen los conceptos de *inliers* y *outliers*: *Inliers* son los datos que se pueden atribuir y agrupar con el resto de puntos ajustados al modelo. Por otra parte, los *outliers* corresponden con los datos que quedan fuera del modelo y no se ajustan al mismo. Como puede existir cierto error a la hora de la recogida de los datos, el algoritmo contempla la posibilidad de considerar como *inlier* datos que puedan encontrarse dentro de un margen de tolerancia determinado.

Matemáticamente, existe una relación matemática mediante la cual se puede calcular el número de iteraciones *k* que se necesitan para obtener un subconjunto de datos formado únicamente por inliers es:

$$k = \frac{\log(1 - P)}{\log(1 - w^l)}$$

Siendo *w* la proporción de *inliers* en los datos y *l* el número de datos escogidos del conjunto para cada iteración. P se trata de una probabilidad deseada de obtener al menos un conjunto sin *outliers*

Para la demostración de su funcionamiento, se toma como referencia el resultado de aplicar RANSAC en el ajuste de un plano: a partir de la generación aleatoria de una serie de puntos sobre el plano $y = 3\ x_1 + 2\ x_2 + 1$, y añadiendo una serie de ruido a la serie, se ejecuta el algoritmo RANSAC así como el modelo lineal, permitiendo comprobar su idoneidad de uso en el caso que nos atañe.

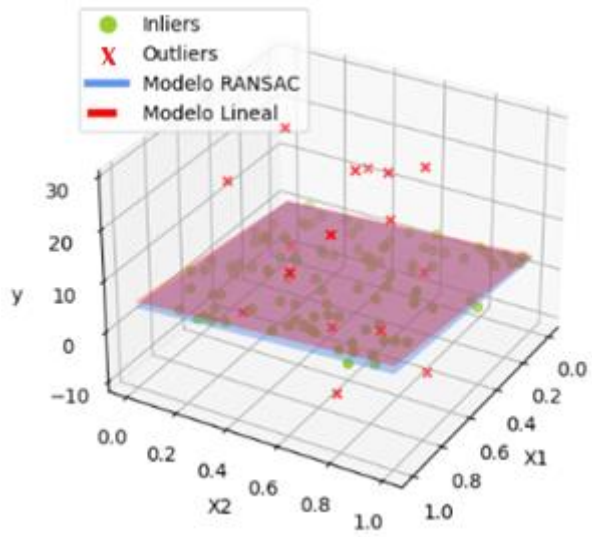

Figura 13: representación propia del funcionamiento del algoritmo RANSAC en la detección de planos

Los resultados que arrojan tanto el modelo lineal como RANSAC son similares para este caso, sin embargo, tal y como se ha mencionado, el proceso iterativo ejecutado en RANSAC suele ofrecer mejor tolerancia frente a los valores atípicos, ya que el modelo lineal se basa en la minimización de la suma de los cuadrados y esto hace que los puntos con grandes errores influyan indiscutiblemente sobre el resultado en este último.

Capítulo 4: Sistema de odometría visual

4.1 Adquisición de datos

El primer paso a realizar en el proceso de desarrollo de esta aplicación será la captura de datos. Para ello, se hará uso de las cámaras Intel RealSense explicadas con anterioridad.

Por una parte, se necesitarán los fotogramas RGB y de profundidad, ofrecidos por la cámara D435. Por otra parte, de la cámara T265 se obtendrá la posición actualizada tanto de la posición como la orientación detectada por el IMU interno de ésta. La toma de dichos datos se ejecutará a través de Pyrealsense y el entorno de desarrollo de Intel.

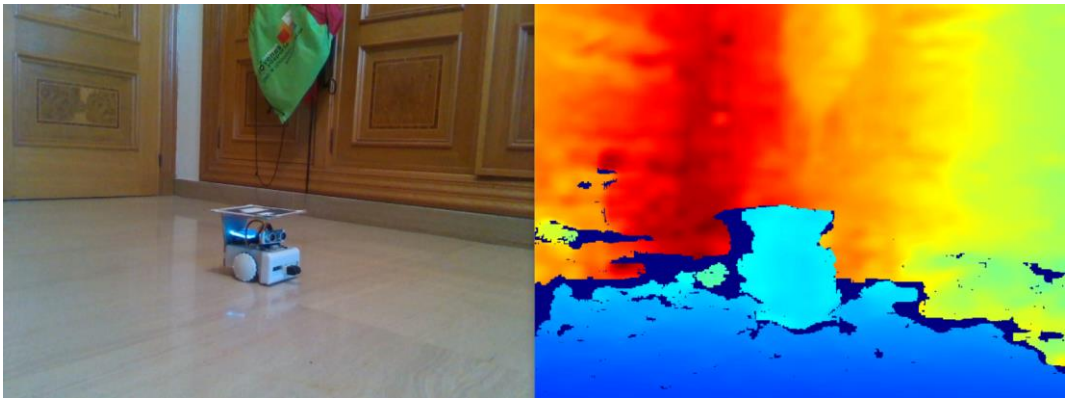

Figura 14: Instantánea RGB y profundidad en mapa de colores, respectivamente

Para el almacenamiento de dicha información, se ha decidido utilizar el ecosistema de ROS para garantizar el almacenamiento sencillo en tres temas o *topics* distintos (imagen color, imagen profundidad y posición de la cámara).

La información a adquirir consiste en unas escenas con los robots *Sparki* de forma que se pueda simular un sistema industrial multi-robot con el que trabajar.

De manera paralela, se ha trabajado para la obtención de imágenes requeridas para el entrenamiento de la red como se explica en el siguiente apartado.

4.2 Entrenamiento de la red

Para proseguir con la implementación de la aplicación, el siguiente paso consistiría en el reconocimiento del robot o robots que pudiera haber en escena.

En un sistema de percepción como es el del ser humano, años de evolución han permitido desarrollar una sección del cerebro denominada córtex visual. En éste, se recibe y procesa las imágenes capturadas desde los conos y bastones de la retina de nuestros ojos. El procesamiento de la información se ejecuta rápidamente, reconociendo bordes, formas, patrones… Finalmente, el cerebro ejecuta un vínculo entre dichas características con memorias y contextos previos, logrando así el reconocimiento del entorno.

De manera similar, se ha de entender el modelo de red convolucional a tratar: con el conocimiento adquirido por la CNN a través del entrenamiento que se le realice, ésta determinará la existencia o no del objeto que se le ha entrenado que encuentre en la imagen.

Para la realización de este proceso, se determina conveniente hacer uso de una red ya preentrenada. Son muchos sus beneficios: el ahorro de tiempo para la adquisición del reconocimiento de un nuevo objeto (en nuestro caso, el robot Sparki), su fácil adaptación a ello, la reducción de los datos de entrenamiento necesarios, entre otros. Estas redes suelen estar preentrenadas con grandes conjuntos de datos, tales como ImageNet.

En nuestro caso, como red preentrenada vamos a utilizar *fasterrcnn_resnet50_fpn*, que se trata de un modelo disponible en la librería de Torchvision. Faster R-CNN consiste en un marco de detección de objetos cuyo objetivo es delimitar y clasificar individualmente en la fotografía objetos a los que ha sido entrenado. ResNet-50 consiste en una arquitectura de red neuronal de 50 capas eficiente en términos de cómputo. Por otra parte, FPN son las iniciales de *Feature Pyramid Network* y tiene como objetivo la mejora de la capacidad de la red para resolver detecciones de objetos.

Se construirá un conjunto de datos propio del objeto a detectar (en este caso el robot Sparki). Para ello, se comienza tomando una grabación del robot desde

todas las perspectivas que puede ser visualizado. A partir de dicha grabación, se desarrolla y ejecuta un pequeño código en Python que transforma la toma de vídeo en una secuencia de imágenes correspondientes a cada uno de los fotogramas de dicha grabación. De esta sencilla forma, se ha logrado obtener más de 1000 imágenes del robot tomadas desde distintas perspectivas y bajo las condiciones que trabajaremos.

El siguiente paso consiste en ordenar todas esas imágenes de forma que puedan ser utilizadas para el entrenamiento de la red. Para ello, se hará uso de la plataforma en línea de Roboflow. A través de dicha plataforma, se pueden adjuntar el conjunto de fotografías obtenidas en el proceso anterior y fácilmente desde el aplicativo realizar las pertinentes anotaciones en cada uno de los fotogramas, de forma que se pueda indicar dónde se encuentra el objeto (y qué objeto) en cuestión en cada imagen. Este primer paso se trata de un procedimiento principalmente manual, a través del cual, en cada imagen, se añade el pertinente recuadro (en inglés, *bounding box*) que delimita dónde se encuentra un objeto en la imagen y la etiqueta bajo la cual llamaremos igualmente dicho objeto en el resto de las imágenes.

A continuación, en la figura 15 se muestra una captura del aplicativo, donde se observa las herramientas para la selección del área donde se encuentra el robot en una de las imágenes subidas. En el margen izquierdo, se puede apreciar que la clase que identifica al objeto recuadrado lleva la etiqueta robot. Si hubiese más objetos de interés distintos al robot, se recuadrarían y clasificarían bajo otra clase.

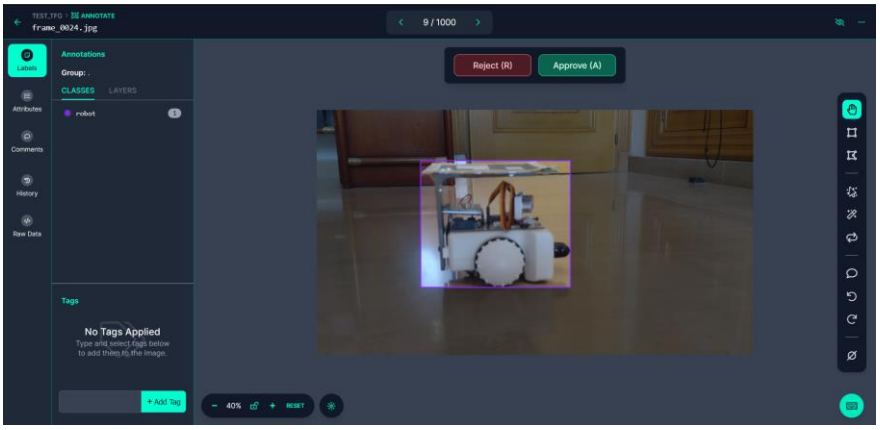

Figura 15: Interfaz de Roboflow con una imagen, anotando la posición del robot Sparki

El siguiente paso dentro del aplicativo de Roboflow va relacionado con la generación del conjunto de datos para el entrenamiento. Este proceso viene destacado por el aumento de imágenes (generalmente volteando la imagen horizontalmente para duplicar el número de imágenes fácilmente) para el *dataset,* así como de la generación del archivo JSON donde aparecen reflejados la información relativa a los objetos que aparecen en cada una de las imágenes. Las imágenes se clasifican en tres grupos siendo generalmente 70% a entrenamiento, 20% a validación y 10% de prueba/comprobación.

El porqué de diferenciar entre entrenamiento y validación viene principalmente marcado para asegurarse de que el modelo no está sobre ajustado y tiene capacidad de abstracción suficiente para detectar el robot en otras imágenes distintas a las de entrenamiento.

Una vez exportado el dataset, comienza el entrenamiento de la red. Como se ha indicado previamente, el entrenamiento se va a basar en una red preentrenada de Pytorch, concretamente *fasterrcnn_resnet50_fpn*. Al tratarse de una actividad que requiere una capacidad de cómputo alta, se utiliza Google Colab para este trabajo. En el script en Python a ejecutar en la plataforma de Colab, se adjunta a la nube el *dataset* y se cargan las librerías de Pytorch y TorchVision. Posteriormente, se invoca a la red preentrenada y se pone en modo entrenamiento con el comando model.train(). A través de la retropropagación del error y la actualización de los parámetros a partir de los gradientes calculados, se termina entrenando el modelo en cuestión. Este proceso conlleva un coste computacional elevado, lo que implica bastante tiempo de ejecución. Según la cantidad de imágenes del dataset y la disponibilidad de recursos para el cálculo, puede demorarse en el orden de horas.

El siguiente paso una vez obtenido el modelo entrenado es la verificación. Se trata de un paso crucial donde se le pasa al modelo entrenado imágenes que no ha visto durante el entrenamiento y a partir de los resultados que arroja se realizan evaluaciones en función de unas métricas y se evalúa su rendimiento. Los parámetros que se suelen analizar son principalmente la precisión, el *recall* o el F1-score. El análisis de dichos resultados ayuda a determinar si el modelo está bien entrenado dentro de dichos parámetros de funcionalidad.

En el capítulo 5 (resultados) se incluye información de estas métricas de forma que se pueda extraer un análisis de este modelo en cuestión.

4.3 Reconocimiento del robot

Una vez entrenada la red, se ha de verificar el funcionamiento correcto con las imágenes de prueba del dataset (también conocidas como las de *test*). Para ello se realiza y ejecuta un pequeño *script* Python donde comprobar dicho funcionamiento. En éste, se carga el modelo obtenido en el entrenamiento y explicado en el apartado anterior. Se inicializa el modelo indicándole que se entra en modo evaluación. A partir de ahora, con los pesos ajustados en el paso del entrenamiento, el modelo deberá indicar el posicionamiento del robot en la imagen. Para ello se le pasa una imagen a la red convolucional y se realizan las predicciones. Generalmente se realiza un proceso de filtrado a partir del umbral de certeza con el que reconoce los patrones que corresponden con el objeto. Si esa certeza es baja, se eliminan. Una vez filtradas las detecciones realizadas con menor certeza para dejar sólo las que superan el umbral, se procede a dibujar sobre la imagen el recuadro (o *Bounding* Box) que conforma el área en la imagen donde se encuentra el robot. Si hubiere varias detecciones que superaran el umbral de certeza, el script realizaría el mismo procedimiento con todos los robots presentes en la fotografía.

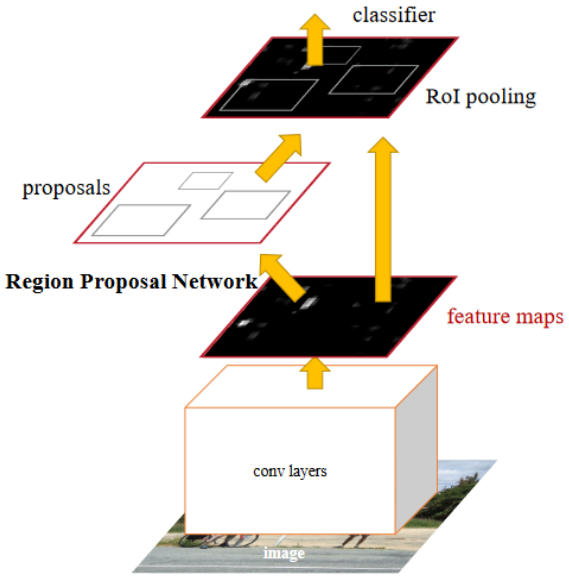

Figura 16: Funcionamiento esquemático de la red neuronal (Fuente: http://arxiv.org/pdf/1506.01497)

En la siguiente figura, se incluye una representación donde se comprueba que el funcionamiento es el esperable.

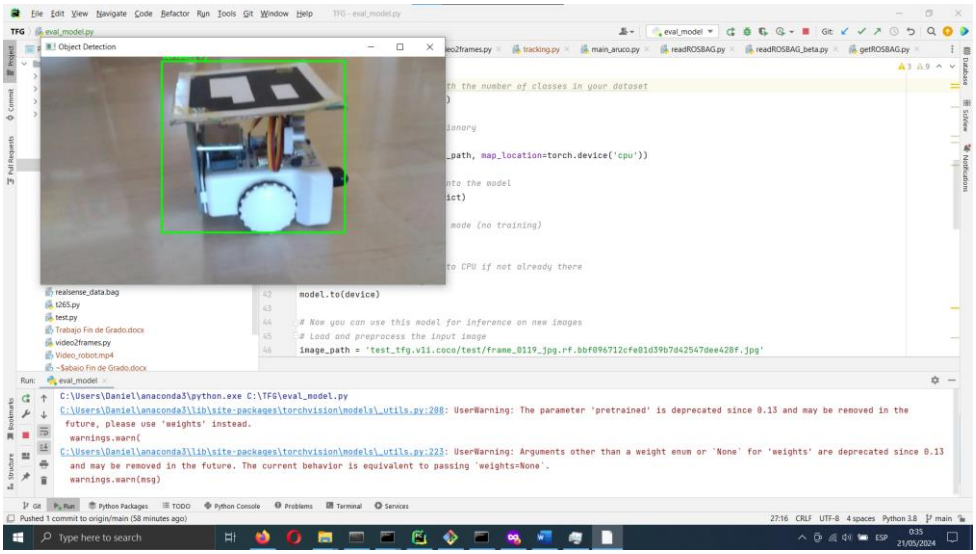

Figura 17: Captura de pantalla con la detección del robot Sparki sobre un fotograma de vídeo

El siguiente paso es integrarlo en nuestro aplicativo que se está construyendo. En este caso, en vez de aportarle la imagen directamente de las imágenes de prueba del conjunto de datos del entrenamiento, se irán aportando secuencialmente las imágenes RGB extraídas del archivo ROSBAG con el que se trabaja. Una vez se transforman las imágenes al formato necesario, se pasan las imágenes por la red convolucional, obteniendo así las predicciones pertinentes.

4.4 Aplicación de SORT

Para garantizar un seguimiento a lo largo de las secuencias de imágenes del robot o robots detectados, se propone el uso del algoritmo SORT. Como se ha explicado con más detalle en el capítulo anterior, SORT hace uso del filtro de Kalman y del algoritmo húngaro para mejorar la definición de las posiciones siguientes a través de estimaciones.

Llevado al campo práctico, existe una librería ya desarrollada por los autores del algoritmo mediante la cual se aplica dicho marco teórico a las detecciones calculadas por la red convolucional entrenada. Su uso es sencillo: primero se invoca a la biblioteca y se inicia el proceso creando un objeto *Sort()*.

A continuación, se entrega al algoritmo SORT los resultados de las detecciones de la imagen que han superado el umbral de certidumbre. De esta forma, relaciona la información que tiene de las detecciones pasadas de otros fotogramas con el actual, asociando un mismo identificador independiente a cada uno de los posibles robots que apareciesen en el trascurso de esa escena.

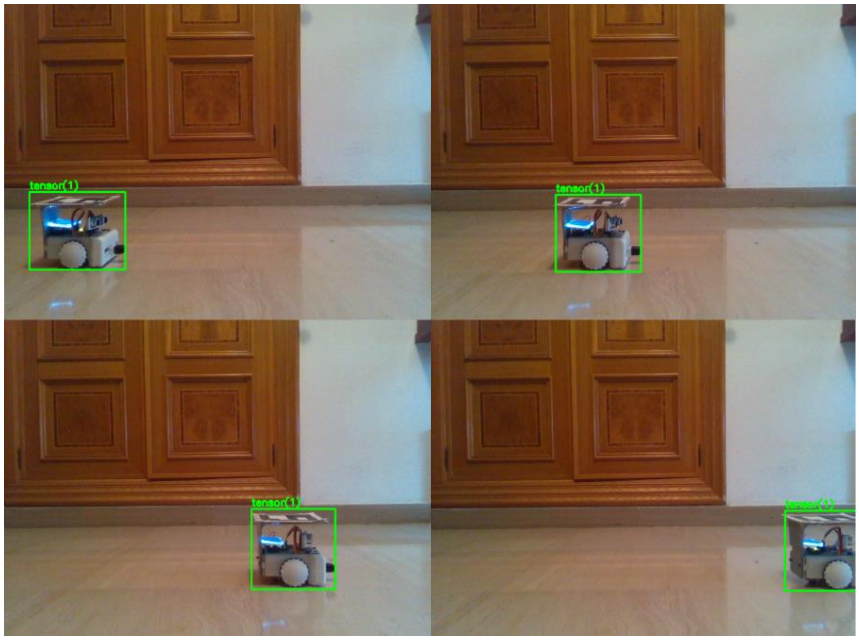

Figura 18: Distintos fotogramas RGB en los que se visualiza el tracking por medio del algoritmo SORT

4.5 Extracción de la pose

Una vez solventado el problema de detección y seguimiento de los robots en escena, el siguiente paso a realizar consiste en la obtención de la posición y orientación de cada uno de ellos, en referencia a la cámara Intel RealSense D435. Para ello, nos centraremos en la imagen de profundidad que realiza la cámara en el momento de captura de datos y que también se encuentra en el archivo ROSBAG generado.

Primeramente, se ha de transformar cada una de las imágenes de profundidad a partir de las que se trabaja en una nube de puntos. Para ello, se hace uso de la librería *Open3D*. Así, con los parámetros intrínsecos de la cámara, se logra la conversión correcta en un mapa de puntos 3D.

Es importante mencionar que el ámbito de trabajo en open3D, así como la pose que devuelve la cámara T265 están referenciados a unos ejes distintos a

los que trabaja la cámara D435 que nos da la imagen de profundidad. Por tanto, una vez obtenida la nube de puntos 3D habremos de realizar la transformación geométrica pertinente. Se cambiarán de sentido tanto el eje Y como el eje Z, según la matriz de transformación siguiente.

$$Transf = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

En la figura 19 se muestra el sistema de coordenadas de referencia de cada una de las cámaras y en open3D.

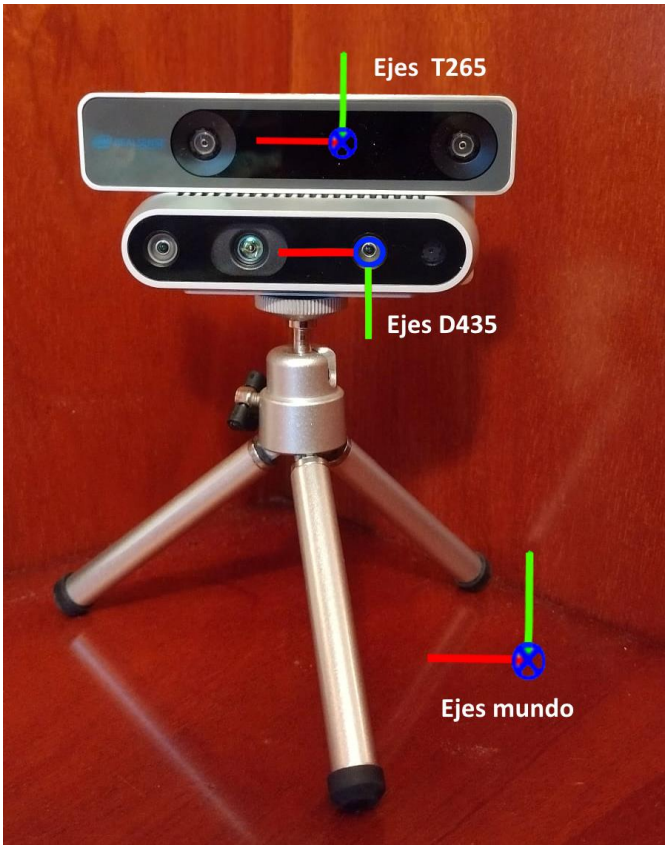

Figura 19: ejes de referencia de cada cámara y de las nubes de puntos 3D

El siguiente paso involucra el uso de la función de detección de planos, con el que se pretende eliminar el plano de suelo. Se utiliza la función *segment_plane* presente en la librería open3D, con la que a partir del método RANSAC elige subconjuntos aleatorios y ajustándolos al modelo de plano. Si no se ajustan correctamente, son descartados.

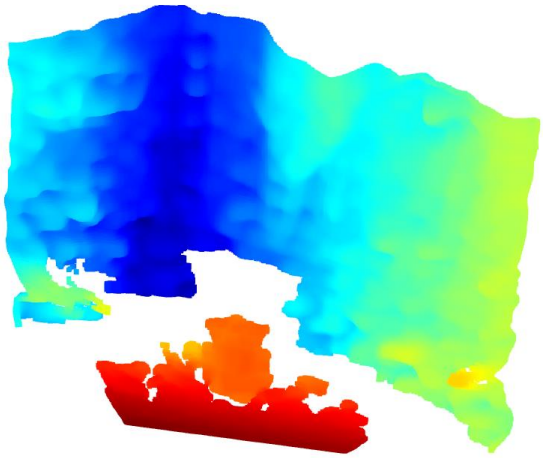

Figura 20: nube de puntos de escena a partir de fotograma original

El próximo paso implica la utilización de los *bounding boxes* resultantes para cada objeto robot detectado. Se llaman a las coordenadas que delimitan en la imagen al objeto y se procede a la eliminación del resto de puntos presentes en la escena. De esta manera, se tendrá únicamente la nube de puntos que corresponde con el robot presente en la escena. Si hubiere más robots detectados en escena, se repite el mismo procedimiento explicado para cada caso individual.

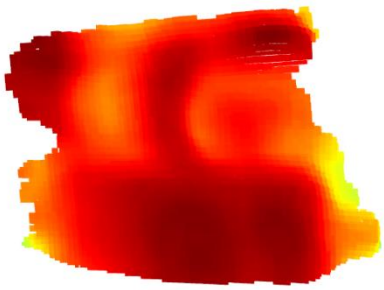

Figura 21: nube de puntos filtrada correspondiente al robot Sparki

Al haber eliminado los puntos que no corresponden con el objeto en cuestión, el siguiente paso a realizar es la aplicación del algoritmo ICP, mediante el que se realiza un proceso iterativo hasta llegar a encajar la nube de puntos que tenemos delimitada, con la obtenida en el fotograma anterior. Así, mediante el algoritmo, se pretende la extracción del cambio entre fotogramas de la posición y orientación del robot.

Este proceso, igualmente, es repetido con cada uno de los robots detectados en la escena para la extracción de los parámetros de localización individual.

En el fotograma en el que se detecta por primera ocasión dicho robot, al no tener un conjunto de puntos como referencia para aplicar ICP, se utiliza la nube de puntos filtrada para calcular el centroide, que es el punto que representa el centro geométrico de la forma tridimensional del robot. Este proceso es crucial para el seguimiento y la interacción precisa con el robot en el entorno tridimensional y servirá como referencia de posición en los ejes para las futuras transformaciones calculadas por el método iterativo.

En el apartado 5.2 el lector podrá comprobar el funcionamiento de dicho algoritmo ICP con dos nubes de puntos así como la obtención del centroide.

4.6 Composición de poses

El proceso final para la obtención de la posición de los robots y su trayectoria, se ha de añadir la posibilidad de que las cámaras se muevan a lo largo de la escena, simulando por tanto el desplazamiento de un robot ‘máster’. Por tanto, es importante obtener un punto de referencia respecto al cual queremos calcular nuestra posición. En términos de este trabajo se ha optado por tomar como referencia origen el momento de arranque de las tomas de vídeo. De todas formas, con la implementación efectuada, se podría llegar a considerar cualquier otro punto como origen, siempre y cuando conozcamos la matriz de transformación que correlacione el punto origen de la grabación con el punto de referencia que se quiere establecer como origen.

Se define una función en el programa Python que se utilizará para calcular las composiciones pertinentes. En este caso planteado, se tendrán que componer 3 de ellas:

- La primera, se trata de la traslación y rotación de la cámara Intel RealSense T265 respecto al punto de origen con el que queremos referenciar las posiciones y trayectorias del robot. Ésta, en nuestro caso y como se ha comentado, será el punto donde se encuentre la cámara al arranque de la grabación.

- La segunda, composición entre la posición de arranque de las cámaras y la trayectoria que realizan las cámaras acopladas.
- La tercera, la composición de rotación y traslación desde la cámara hasta el centroide de la nube de puntos del objeto detectado en su primera ocasión.
- Finalmente, la composición de los resultados obtenidos por ICP al comparar cada uno de los fotogramas con su anterior, componiendo el desplazamiento individual de cada robot.

Capítulo 5: Validación del método

En este apartado se pretende realizar un análisis pormenorizado del rendimiento de la red neuronal y los distintos algoritmos aplicados que integra el sistema, con el fin de poder evaluar su buen funcionamiento de manera individual.

5.1 Evaluación de la red neuronal

Para la realización de la evaluación de la red neuronal, existen una serie de parámetros y gráficas con los que considerar la calidad del entrenamiento que se ha realizado sobre la misma.

La primera característica a considerar en la CNN es determinar la existencia de sobreajuste o subajuste a raíz del entrenamiento de la misma. Estos términos son generalmente conocidos también como *overfitting* o *underfitting*, respectivamente y constituyen características a evitar en nuestra red neuronal.

Para ello, existe un punto del entrenamiento idóneo a alcanzar, en el cual la red neuronal es lo suficientemente correcta como para extraer conclusiones acertadas en imágenes no utilizadas durante el entrenamiento. En caso de que se siguiera entrenando una vez alcanzado dicho punto, la red perdería la capacidad de generalización, arrojando por tanto peores resultados en las futuras pruebas efectuadas.

Podemos extraer las pertinentes conclusiones del estado de entrenamiento de la red a partir de la comparación de los valores de pérdida (*loss*) y exactitud (*accuracy*) con las imágenes de entrenamiento y validación a lo largo de las épocas de entrenamiento. El resto de indicadores como la precisión (*precision*), sensibilidad (*recall*) y el valor F1 pueden ser igualmente indicativos de la calidad del entrenamiento en cada época.

Este fenómeno se manifiesta de forma más evidente en los valores de pérdida por épocas del conjunto de validación, donde se busca la minimización del valor. A partir de un momento del proceso de entrenamiento, se observa una

estabilización y posterior aumento del mismo, signo de que se ha alcanzado el punto idóneo donde dejar de entrenar.

El resto de métricas de *accuracy, precision, recall* y F1 son también indicativos del estado del entrenamiento y determinan el comportamiento de la red ante distintas imágenes del conjunto de validación.

Para solventar el problema de desconocimiento de cuándo dejar de entrenar la red, existe un método conocido como *early stopping* mediante el cual se toma como correcto el modelo ajustado en una época donde se minimiza el valor de pérdida con el conjunto de datos de validación y que es alcanzado previo al sobreajuste de la red (figura 22).

Se presentan a continuación en la figura 22, figura 23 y figura 24 distintas gráficas de las métricas obtenidas a partir del entrenamiento de la red para el propósito de detectar el robot Sparki:

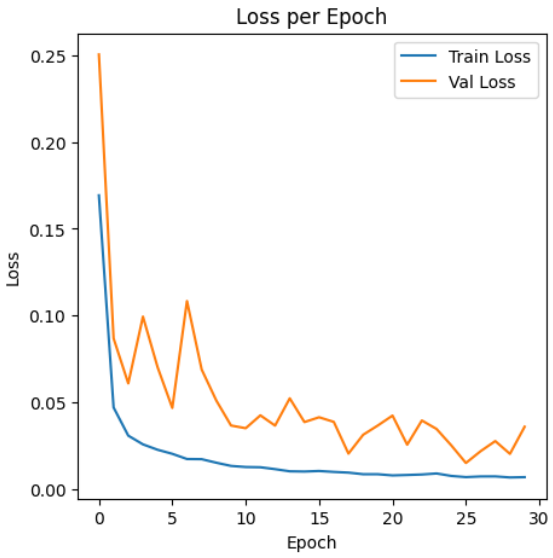

Figura 22: valores de pérdida por época

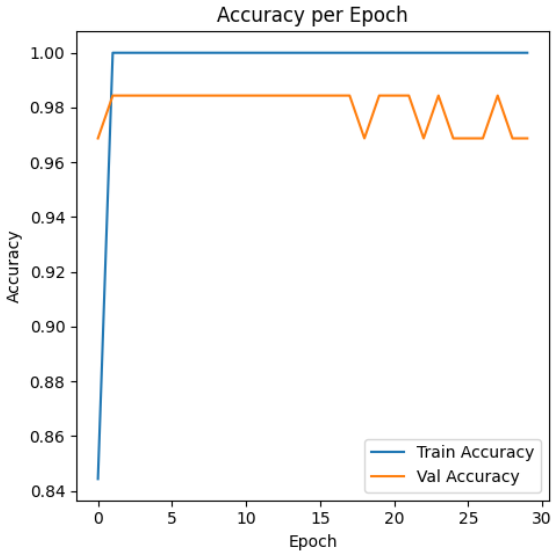

Figura 23: valores de exactitud por época

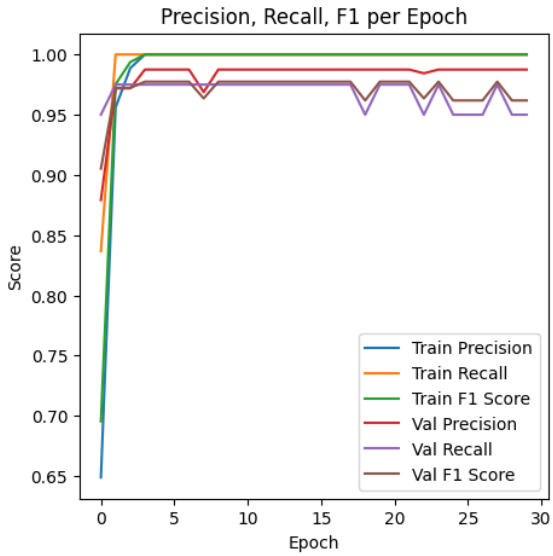

Figura 24: resto de valores calculados por época

El análisis de las métricas obtenidas del entrenamiento de la red a lo largo de varios ciclos o épocas, permiten determinar la existencia de sobreajuste en los ciclos finales. Este comportamiento se ajusta a las características esperables de prolongar más tiempo del necesario el proceso de entrenamiento.

Para la selección de un modelo que tenga capacidad suficiente de generalizar sin entrar en el sobreajuste, se utiliza el método ya explicado de *early stopping*, que realiza de manera automática el análisis de las métricas por época y selecciona la red en el momento de entrenamiento idóneo. Este proceso nos hace tomar como idóneo el modelo en la época 6.

En la siguiente tabla se recogen datos de distintas métricas de evaluación del comportamiento de la red en la época 6:

*Exactitud (Accuracy**)***	0.9844
Precisión (Precision)	0.9875
Sensibilidad (Recall)	0.9750
F1-Score	0.9774

Estos valores indican que el comportamiento de la red a la hora de la detección del robot Sparki es excelente. Los parámetros sugieren que el modelo es preciso en realizar detecciones correctas, manteniendo asimismo una buena sensibilidad en todo momento.

Hay que tener en consideración que el buen desempeño de este modelo se ve magnificado al tener un *dataset* limitado al área de trabajo donde se desempeña el proyecto. En futuribles situaciones en las que la detección del robot se tuviera que realizar en entornos complejos y con mayor número de objetos, detalles y luminosidad, entre otros, los resultados obtenidos, así como el desempeño de la red, podrían verse limitados.

5.2 Evaluación del sistema de obtención de pose

A lo largo de la obtención de posición y cambio de orientación del robot, se utilizan técnicas distintas que se combinan en aras de obtener el recorrido realizado en la toma de video.

Se analizan las implementaciones realizadas:

5.2.1 Obtención del centroide

En el momento de detección de un nuevo robot en escena, se hace uso de esa instantánea para la obtención de unas coordenadas relativas al punto origen. Se utiliza el conjunto de puntos resultante de filtrar aquellos puntos que no se corresponden con el robot para calcular el centroide.

Figura 25: Esfera verde representando el centroide de la superficie detectada del robot Sparki, comparación con la escena global

Como se puede observar en la figura, el punto representado por una esfera verde y que es resultante del cálculo del centroide, puede ser un buen punto de partida a partir del cual referenciar las posteriores composiciones. De este punto sólo se obtiene las traslaciones en los ejes de referencia.

El concatenamiento con el resto de *poses* obtenidas a lo largo de la secuencia, se realizará utilizando el método de ICP, explicado a continuación en el punto 5.2.2.

5.2.2 Resultados de implementación por método ICP

La implementación del ICP entre nubes de puntos de fotogramas consecutivos, referenciando de manera continua las nubes de puntos obtenidas respecto a la posición origen, resulta en la obtención satisfactoria del movimiento realizado por el robot.

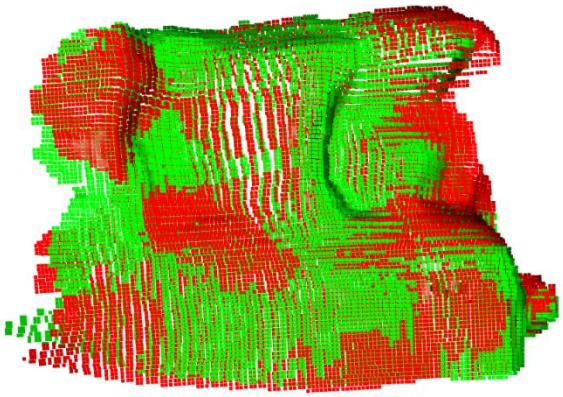

Figura 26 Aplicación del método ICP entre dos nubes de puntos, verde y rojo, respectivamente

En la escena de la figura, se observa el resultado de aplicar ICP entre dos escenas de dos fotogramas (siendo diferenciadas entre las nubes de puntos de color rojo y las de color rojo) consecutivos. La aplicación del algoritmo de iteración permite conocer la matriz de transformación que relaciona las dos nubes de puntos.

La concatenación en secuencia de éstas nos lleva a conocer la rotación y traslación total que ha realizado en la escena el robot. El resultado es obtenido en forma de matriz (*4x4*), al que se le concatenaría la primera matriz obtenida con el centroide del robot en la primera detección, que incluye la traslación desde la cámara a dicho punto.

Con toda la información almacenada, podríamos extraer de la secuencia del recorrido, así como la posición final respecto de la cámara.

Capítulo 6: Experimentos reales

A lo largo de este capítulo se pretende poner a prueba y evaluar la aplicación desarrollada a través una serie de casos de diferente índole. De esta forma se puede analizar su rendimiento a tenor de los resultados tanto de rotación como de traslación.

6.1 Caso 1: Desplazamiento lateral del robot

En este experimento, se ejecuta una traslación rectilínea a lo largo del eje X en sentido positivo. El robot realiza un recorrido hacia la derecha de la imagen de una distancia total de 60 centímetros. Como referencia a esta medida, se ha utilizado la anchura de las baldosas del escenario de pruebas, que son de 30 centímetros cada una, lo cual equivale a que el robot recorra la longitud de dos baldosas consecutivas. Las cámaras permanecen estáticas.

Este ejemplo resulta de gran utilidad para comprobar la precisión de las distintas escenas 3D generadas con coordenadas adecuadamente calibradas. Esto es debido a la necesidad de tener en consideración los parámetros intrínsecos de la cámara.

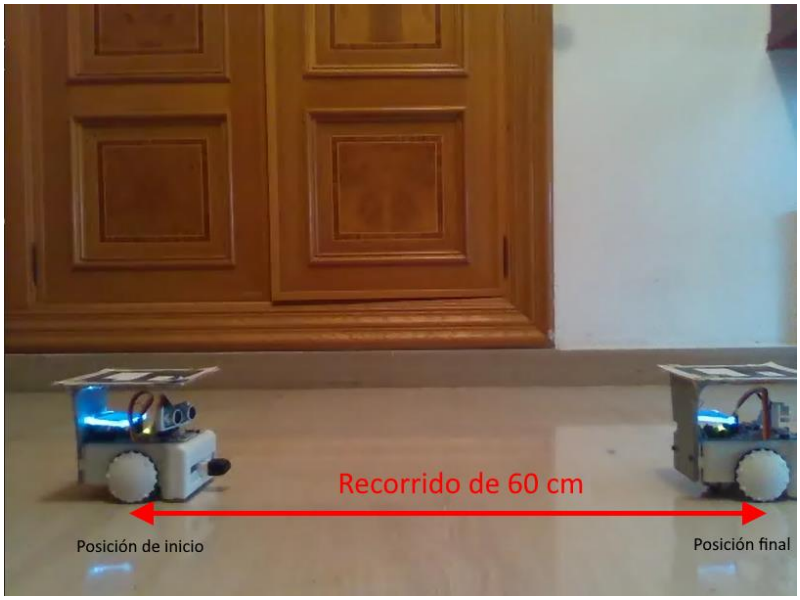

Figura 27: Primer ejercicio propuesto, recreación con robot en escena inicial y final

Para tener como referencia adicional, se sacarán las traslaciones (X, Y, Z) del robot tanto en la posición de inicio como en la posición final respecto de la cámara. Se tendrán en consideración los centroides para la obtención de éstas.

$$T_{inicial} = (X = -0.2767, \quad Y = -0.0712, \quad Z = -0.7683)\ [metros]$$

$$T_{final} = (X = 0.3126, \quad Y = -0.0451, \quad Z = -0.7417)\ [metros]$$

A través de esta primera aproximación, podemos tener otro marco comparativo de referencia con el que valorar el resultado que obtendremos con el sistema desarrollado. Éste, será equivalente a la medición física del espacio recorrido (60 centímetros).

La traslación ejecutada por el robot Sparki es posible calcularla con la distancia euclídea de ambas coordenadas de los centroides. Observamos como obtenemos un valor equivalente al ancho de las baldosas.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} = 0.5904\ metros$$

A continuación, se realiza el análisis equivalente poniendo a prueba el sistema desarrollado en este trabajo. Se realizará la composición de traslaciones calculadas entre el primer y último fotograma, haciendo tratamiento de las nubes de puntos, así como el uso del algoritmo ICP y los algoritmos de detección y seguimiento del robot.

Se obtiene el siguiente resultado en forma de matriz de transformación:

$$(R \mid t) = \begin{pmatrix} 0.9920 & -0.1112 & 0.0580 & 0.6104 \\ 0.1124 & 0.9935 & -0.0175 & 0.0302 \\ -0.0557 & 0.0238 & 0.9981 & 0.0362 \\ 0 & 0 & 0 & 1 \end{pmatrix}\ [metros]$$

Como se puede comprobar, la aplicación del método arroja también buenos resultados en este tipo de desplazamiento, obteniendo como matriz de rotación prácticamente la matriz identidad, lo que lleva a entender que no se ha registrado rotación apreciable fuera del ruido esperable en la lectura y análisis de datos. La distancia entre inicio y final la calculamos igualmente con la distancia euclídea

de las coordenadas de traslación, obteniendo como resultado 0.6122 metros de distancia recorrida.

Se realiza el mismo experimento recorriendo 60 centímetros, pero a mayor distancia de la cámara, con el propósito de comprobar la correcta adecuación de la escena 3D también en objetos más lejanos.

A través del análisis de las nubes de puntos en la posición inicial y final, se obtienen las siguientes coordenadas respecto de la cámara.

$$T_{inicial} = (X = -0.3142, \quad Y = -0.0477, \quad Z = -1.1744)\ [metros]$$

$$T_{final} = (X = 0.2828, \quad Y = -0.0237, \quad Z = -1.1869)\ [metros]$$

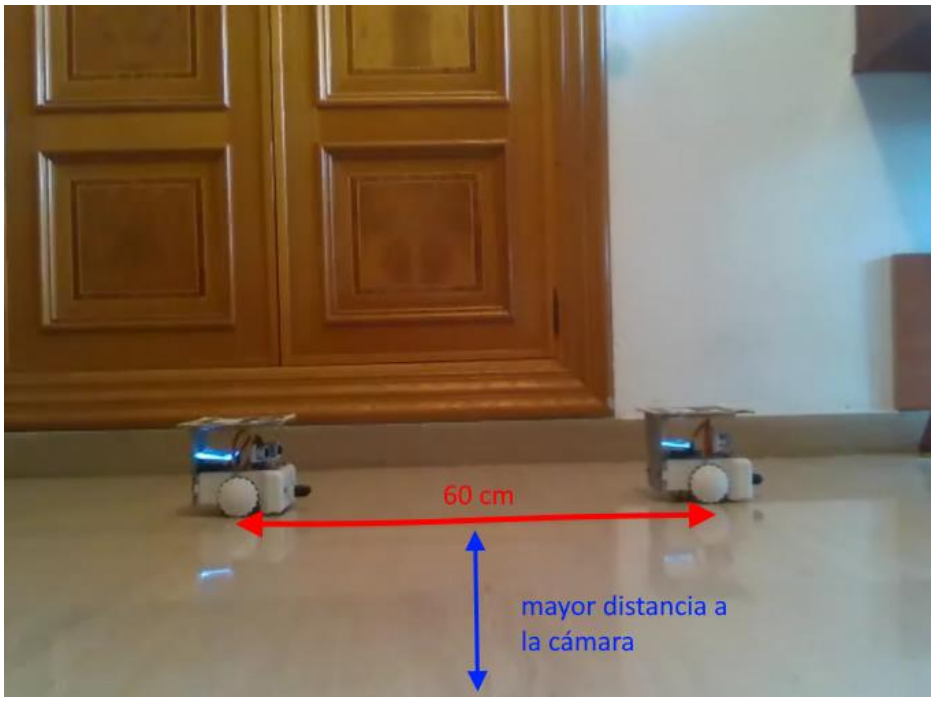

Figura 28: segundo ejercicio del caso propuesto, a mayor distancia de la cámara. Recreación con robot en posición inicial y final

La distancia euclídea determina una distancia de referencia a partir de los centroides entre las escenas inicial y final de 0.597 metros.

Centrándonos nuevamente en el análisis a través de la herramienta implementada en este trabajo, se obtiene la siguiente matriz de transformación entre la posición inicial y la final:

$$(R \mid t) = \begin{pmatrix} 0.8335 & 0.1926 & -0.5177 & 0.5739 \\ -0.2313 & 0.9728 & -0.0104 & -0.0190 \\ 0.5016 & 0.1285 & 0.8554 & 0.0744 \\ 0 & 0 & 0 & 1 \end{pmatrix} [metros]$$

La distancia euclídea determina una traslación de 0.579 metros, ajustándose bastante bien a ambas referencias tanto física (recorrido marcado por la distancia medida de las baldosas) como la referencia de las nubes de puntos. Se trataría de un error de alrededor de un 3%.

En cuanto a la matriz de rotación, se aprecia una distorsión mayor que en el primer ejemplo del caso, si bien es esperable al encontrarse más lejos de las cámaras y la calidad de detalles en la nube de puntos del robot disminuye.

La rotación calculada que ha sufrido el robot entre la posición final y la inicial se calcula de la siguiente forma:

$$Traza(R) = 1 + 2 * \cos(theta)$$

$$theta = \text{acos}\frac{Traza(R) - 1}{2} = 33.3º$$

En términos generales, comentar que se cumplen expectativas al no apreciar distorsiones manifiestas en las medidas obtenidas respecto a los valores reales, gracias a tener en consideración los parámetros intrínsecos de la cámara calibrados.

6.2 Caso 2: Rotación sobre el eje del robot

En este experimento realizarán rotaciones puras sobre el eje para comprobar el funcionamiento del algoritmo ICP, el cual resuelve las rotaciones y traslaciones a lo largo de la secuencia. Se mantienen igualmente las cámaras fijas.

Figura 29: Robot Sparki con las flechas de los sentidos de las rotaciones puras a aplicar

Como primer ejemplo, realizamos una rotación pura de 90º en sentido antihorario (flecha verde). El resultado se refleja en la siguiente matriz de transformación.

$$(R \mid t)(90º) = \begin{pmatrix} 0.0227 & 0.1294 & 0.9913 & -0.0039 \\ -0.0456 & 0.9906 & -0.1283 & 0.0039 \\ -0.9986 & -0.0423 & 0.0284 & 0.0126 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Analizando el resultado, se observa la no existencia de traslación, más allá del ruido esperable en este tipo de sistemas.

Respecto a la rotación, el ángulo percibido por el algoritmo según la información de la cámara se puede calcular de la siguiente forma:

$$theta = \arccos \frac{Traza(R) - 1}{2} = 88.8º$$

Este resultado es por tanto ligeramente menor al efectuado por el robot en la escena.

Como segundo ejemplo del caso de rotación pura, se propone el ejercicio de giro sobre el mismo eje en sentido horario (flecha roja) 180º.

El resultado obtenido viene recogido en la siguiente matriz:

$$Transf(180º) = \begin{pmatrix} -0.9935 & -0.0519 & -0.1011 & 0.0182 \\ -0.0633 & 0.9915 & 0.1134 & 0.0041 \\ 0.0943 & 0.1190 & -0.9883 & 0.0152 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Para este caso se observa igualmente que apenas existe desplazamiento en los ejes. Igualmente, calculo la rotación registrada en a partir del giro efectuado:

$$theta = \text{acos}\frac{Traza(R) - 1}{2} = 174.35º$$

En la figura mostrada a continuación se representa la rotación efectuada:

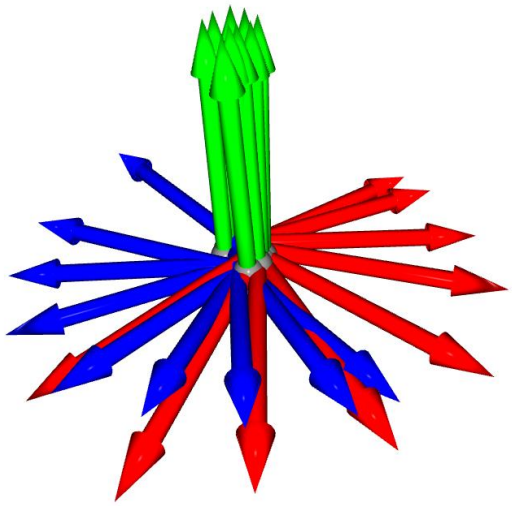

Figura 30: representación de giro del ejemplo de rotación pura de 180º

Como hemos podido observar en ambos casos, la aplicación de algoritmos iterativos como ICP no eximen de la aparición de pequeños errores que se pueden ir acumulando al resultado de la odometría. Este fenómeno es esperable al tratarse de un entorno real en el que no se dispone de una representación exacta del robot al girar, además de poder existir pequeños errores en la medida por parte de la cámara.

Se realiza como tercer ejemplo en modo comparación un giro de 180º en sentido horario igual que en el ejemplo 2, pero a mayor distancia de la cámara. En este caso sí que percibimos pérdidas mayores al no disponer de unas nubes de puntos con la misma definición que las tomadas más próximas a la cámara.

$$(R \mid t) = \begin{pmatrix} -0.0691 & -0.1480 & -0.9865 & 0.0063 \\ 0.0035 & 0.9888 & -0.1486 & 0.0053 \\ 0.9975 & -0.0137 & -0.0678 & 0.0037 \\ 0 & 0 & 0 & 1 \end{pmatrix} [metros]$$

Esto es equivalente a realizar un giro de 93,78º, valor muy dispar con los 180º que habríamos de detectar. La traslación permanece sin cambios reseñables.

Por tanto, queda reflejado nuevamente la idoneidad de tener las cámaras próximas al robot si no queremos perder precisión en la medida. A mayor distancia, mayor ruido en la medida.

6.3 Caso 3: Desplazamiento con traslaciones y rotaciones puras

Para este caso se propone la realización de un pequeño recorrido compuesto de translaciones y rotaciones puras. De esta manera podemos medir la efectividad del trabajo en un ejercicio compuesto de traslaciones laterales, traslaciones en el eje de la cámara y de rotaciones. Se mantienen igualmente las cámaras estáticas. En la figura siguiente se muestra el recorrido propuesto:

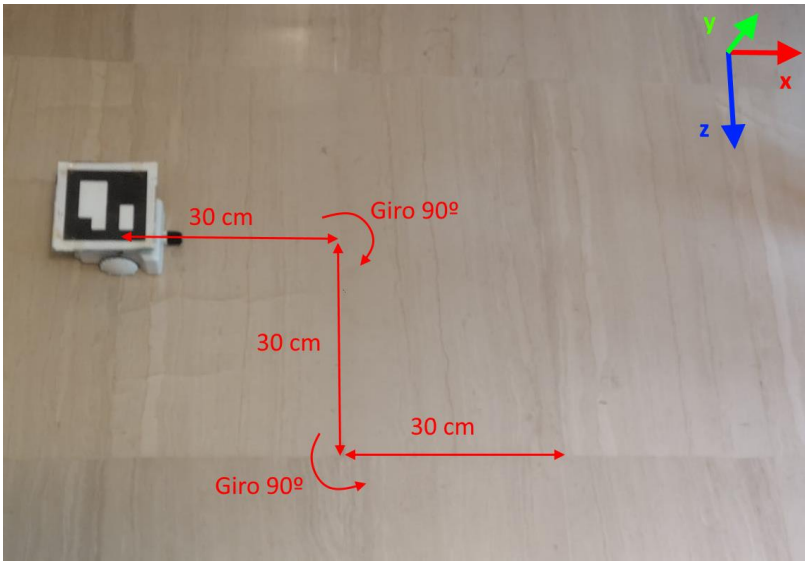

Figura 31: Indicaciones del ejercicio propuesto, indicando desde el origen la distancia a recorrer, sentido, así como las rotaciones que se realizarán

Como referencia, se obtienen nuevamente las posiciones en coordenadas de traslación del robot tanto en el primer como último fotograma:

$$T_{inicial} = (X = -0.3097, \quad Y = -0.0628, \quad Z = -1.0269)\ [metros]$$

$$T_{final} = (X = 0.2757, \quad Y = -0.0523, \quad Z = -0.7044)\ [metros]$$

Haciendo uso de la distancia euclídea calculamos la distancia rectilínea existente entre el origen y destino, siendo ésta de 0.6684 metros. Igualmente, la podemos calcular teóricamente como la norma del vector que une ambas posiciones, dando como resultado 0.6708 metros.

A continuación, se presenta la matriz de transformación obtenida a partir del trabajo desarrollado:

$$(R \mid t) = \begin{pmatrix} 0.9828 & 0.0115 & 0.1838 & 0.6338 \\ -0.1176 & 0.8071 & 0.5785 & 0.0346 \\ -0.1417 & -0.5902 & 0.79468 & 0.2819 \\ 0 & 0 & 0 & 1 \end{pmatrix} [metros]$$

A partir de las coordenadas de traslación calculamos la distancia euclídea que resulta de ejecutar la implementación, resultando en 0.694 metros. Se trata de un valor muy similar al calculado tanto por la distancia entre centroides como con la relación matemática.

Analizando más detalladamente el recorrido, se expone en la figura 32 el recorrido registrado y la posición y orientación del robot en distintos momentos de la toma.

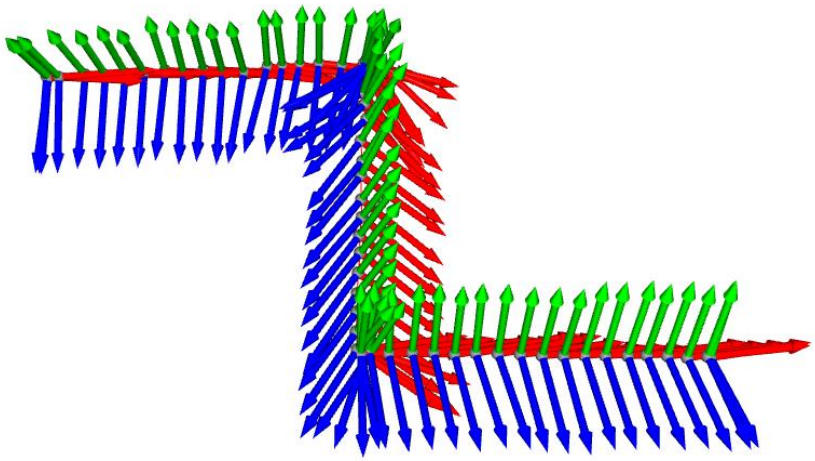

Figura 32: recorrido registrado y posición y orientación del robot

Se observa que el recorrido traslacional lo registra con bastante precisión, si bien al existir tantos procesos iterativos se aprecian mayores errores en las rotaciones. A destacar, los giros de 90º sobre el eje, que quedan reducidos tanto por la baja resolución en el giro del fondo como la pérdida de la verticalidad del eje Y (verde) en *frames* anteriores. La distancia entre puntos inicial y final del recorrido se registra con un ligero error, a vista del resultado de la distancia euclídea (0.694 metros calculados con el método frente a los 0.6708 metros de la norma, dato que hemos calculado en la página 49).

Con el fin de evaluar con mayor detalle la escena, se opta por segmentar el recorrido en partes diferenciadas para evaluar su funcionamiento en los distintos movimientos realizados. Se tendrá, por tanto, un primer desplazamiento rectilíneo sobre eje X positivo que llamaremos tramo 1. Seguidamente, se analizará el giro de 90º del robot en sentido horario. Le procederá el desplazamiento rectilíneo del robot sobre el eje Z en sentido positivo, que llamaremos tramo 2. A continuación se revisará el giro de 90º en sentido antihorario. Finalmente, se evaluará el tramo final que recorre de forma rectilínea entre este último giro y el final del caso, que lo designaremos como tramo 3.

Para ello, compondremos las matrices homogéneas de cada tramo resultando en las siguientes:

$$(R \mid t)(tramo\ 1) = \begin{pmatrix} 0.9673 & 0.1428 & -0.2091 & 0.3180 \\ -0.1081 & 0.9797 & 0.1688 & -0.0081 \\ 0.2290 & -0.1406 & 0.9631 & -0.0154 \\ 0 & 0 & 0 & 1 \end{pmatrix} [metros]$$

$$(R \mid t)(giro\ 90º) = \begin{pmatrix} 0.6817 & -0.0188 & -0.7313 & 0.0090 \\ -0.0696 & 0.9934 & -0.0905 & -0.0025 \\ 0.7282 & 0.1127 & 0.6759 & -0.0022 \\ 0 & 0 & 0 & 1 \end{pmatrix} [metros]$$

$$(R \mid t)(tramo\ 2) = \begin{pmatrix} 0.9056 & -0.0166 & 0.4236 & 0.2516 \\ -0.1439 & 0.9277 & 0.3442 & 0.0450 \\ -0.3987 & -0.3727 & 0.8378 & 0.1511 \\ 0 & 0 & 0 & 1 \end{pmatrix} [metros]$$

$$(R \mid t)(giro\ 90º) = \begin{pmatrix} 0.8347 & -0.1835 & 0.5190 & -0.0075 \\ 0.1663 & 0.9828 & 0.0800 & 0.0033 \\ -0.5248 & 0.0195 & 0.8509 & 0.0058 \\ 0 & 0 & 0 & 1 \end{pmatrix} [metros]$$

$$(R \mid t)(tramo\ 3) = \begin{pmatrix} 0.9999 & -0.0063 & -0.012 & 0.3074 \\ 0.0078 & 0.9911 & 0.1322 & 0.0547 \\ 0.0111 & -0.1323 & 0.9911 & 0.0605 \\ 0 & 0 & 0 & 1 \end{pmatrix} [metros]$$

A partir de estas matrices obtenidas, se estudia cada segmentación por separado del caso y analizamos el comportamiento tanto en rotación como traslación.

		Medida calculada	**Medida real**	**E_{Abs}**	**$E_{relativo}$(%)**
Tramo 1	| Trasl |	31.8 (cm)	30 (cm)	1.8 (cm)	6.1%
	| Rot |	17.25º	0º	17.25º	-
Giro 90º (sentido horario)	| Trasl |	0.9 (cm)	0 (cm)	0.9 (cm)	-
	| Rot |	47.93º	90º	42.06º	46.7%
Tramo 2	| Trasl |	29.42 (cm)	30 (cm)	0.58 (cm)	1.9%
	| Rot |	34.4º	0º	34.4º	-
Giro 90º (sentido antihorario)	| Trasl |	0.9 (cm)	0 (cm)	0.9 (cm)	-
	| Rot |	33.9º	90º	56.1º	62.3%

Tramo 3	| Trasl |	31 (cm)	30 (cm)	1 (cm)	3.3%
	| Rot |	7.65º	0º	7.65º	-

Los movimientos de rotación acontecidos son capturados con errores que bajo ciertas circunstancias pueden llegar a resultar elevados, mientras que la traslación no suele incurrir en errores reseñables para un propósito general.

Respecto a los tramos 1, 2 y 3, se considera de interés el error de traslación de las posiciones intermedias respecto a una trayectoria rectilínea entre el punto inicial y final. Para ello, vamos a tomar la referencia de las coordenadas en los ejes del primer y último fotograma de cada tramo, con los que definiremos una recta que las une. Con el resto de coordenadas de cada tramo, se calcularán las distancias entre cada punto y la recta. Posteriormente, se utilizarán dichas distancias en el cálculo de la desviación típica.

Para los tramos 1, 2 y 3, se obtienen respectivamente:

$$\sigma_{tramo\ 1} = \ 0.12\ centímetros$$

$$\sigma_{tramo\ 2} = \ 0.72\ centímetros$$

$$\sigma_{tramo\ 3} = \ 0.05\ centímetros$$

Se tratan de valores relativamente bajos, lo que nos lleva a concluir que las composiciones de matrices se realizan de manera regular y sin muchas oscilaciones. Comentar que el valor del tramo 2 ha resultado tener mayor desviación estándar respecto a los otros dos tramos, lo que lleva a pensar que el algoritmo por ICP puede ser algo más inestable cuando se realizan traslaciones por el eje Z. La distancia a la cámara es otro factor a considerar.

A continuación, se muestra en la figura 33 la distribución de distancias entre punto y recta calculados para cada tramo:

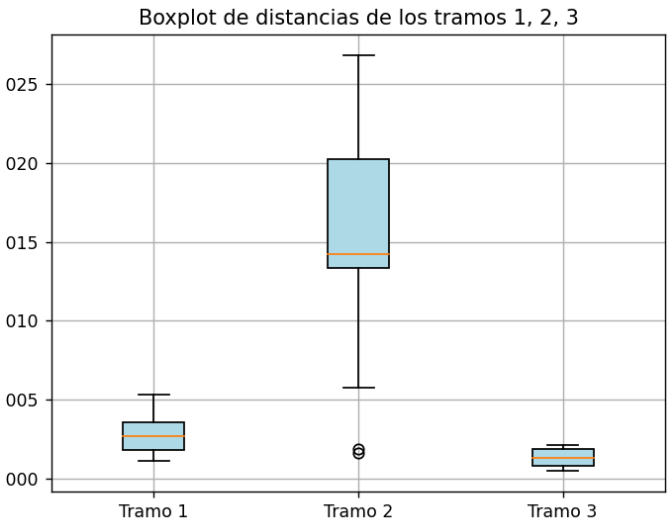

Figura 33: boxplot con distribución de distancias en tramo 1, tramo 2 y tramo 3

6.4 Caso 4: Desplazamiento con movimiento de cámaras

En este caso se plantea la realización de un movimiento rectilíneo con el robot mientras desplazamos las cámaras por la escena de forma paralela al desplazamiento del robot. Se opta por recorrer 30 centímetros en sentido negativo del eje X, dadas las limitaciones de movimiento de las cámaras al estar conectadas al ordenador.

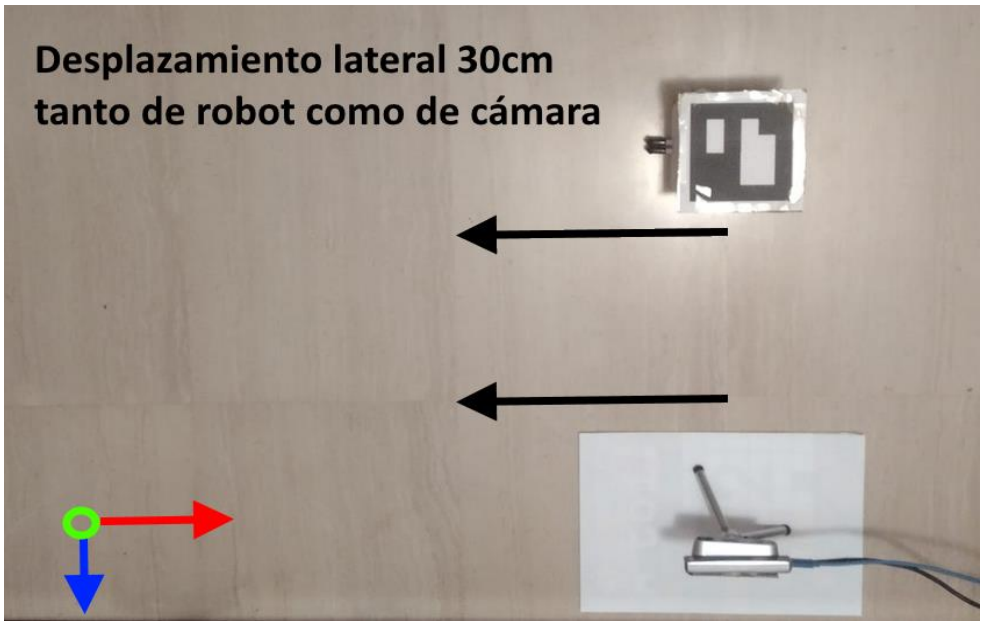

Figura 34: Desplazamiento paralelo de cámaras y robot Sparki

Con esta propuesta, se consigue analizar tanto la fiabilidad de los valores que aporta la cámara T265 en cuanto a su movimiento en la escena, como la medición que realiza el sistema del desplazamiento del robot con una referencia real.

Tras su ejecución, obtenemos la matriz de transformación del robot y la correspondiente a la pose propia de la cámara.

$$^{Abs}\,(R \mid t)_{Robot} = \begin{pmatrix} 0.9880 & 0.0314 & -0.1506 & -0.2603 \\ -0.0145 & 0.9936 & 0.1114 & -0.0213 \\ 0.1531 & -0.1079 & 0.9822 & -0.0380 \\ 0 & 0 & 0 & 1 \end{pmatrix} [metros]$$

$$^{Abs}(R \mid t)_{T265} = \begin{pmatrix} 0.9959 & -0.0055 & -0.0892 & -0.3137 \\ 0.0096 & 0.9989 & 0.0446 & -0.0024 \\ 0.0888 & -0.0452 & 0.9950 & -0.0399 \\ 0 & 0 & 0 & 1 \end{pmatrix} [metros]$$

Respecto a la matriz de transformación del robot, se aprecia como no se han efectuado rotaciones importantes alrededor de los ejes. Por tanto, ha mantenido con bastante solvencia un movimiento rectilíneo como el realizado por el robot en escena. En cuanto a la traslación, la distancia euclídea calculada a partir de los valores de traslación indica que ha recorrido una distancia de 26.3 centímetros a lo largo del eje X en sentido negativo casi en su totalidad, siendo un valor reseñable con un error de alrededor del 12%.

Analizando la representación del recorrido dibujando los ejes a lo largo del recorrido, no se aprecian alteraciones ni problemas derivados del uso del programa desarrollado. En ésta, el robot realiza un recorrido rectilíneo a lo largo del eje X en sentido negativo.

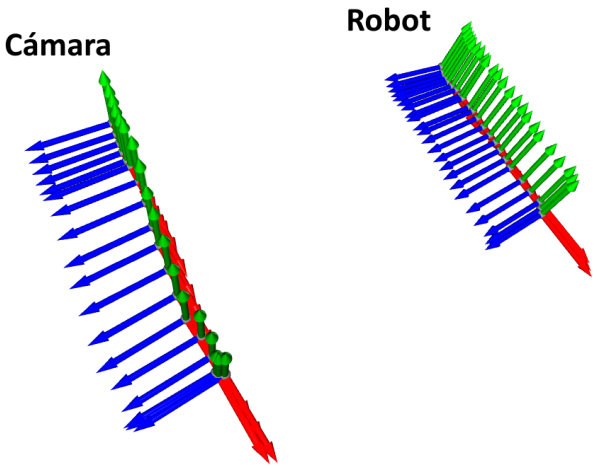

Figura 35: Vista lateral del recorrido realizado por el robot y las cámaras, paralelamente, a lo largo del eje X (rojo) en sentido opuesto a las flechas

Finalmente, en referencia a la matriz de transformación de la cámara T265, se observa una traslación sobre el eje X de 31.6 centímetros en sentido negativo, conforme lo previsto. De esta forma se ha podido comprobar y garantizar que ambas cámaras se encuentran calibradas y en las mismas unidades. Indicar también que no se aprecian rotaciones significativas, tal y como se ha descrito el ejemplo.

6.5 Discusión de los resultados de los diferentes casos

El análisis de las diferentes situaciones analizadas permite extraer varias conclusiones en líneas generales del funcionamiento de la aplicación de odometría visual.

Por una parte, existe una estrecha relación entre la fiabilidad del resultado propuesto y la proximidad del robot a la cámara, logrando mejores resultados en los casos que colocamos el robot en primer plano. Esto se debe a que, cuanto a más distancia de la cámara, la definición obtenida de la forma del robot por la cámara es de peor calidad, lo que dificulta al algoritmo ICP a la hora de iterar para lograr la transformación correcta.

Por otra parte, el registro del movimiento de traslación por parte del robot se realiza sin mayor problema en los casos analizados. En referencia a las rotaciones puras, existen situaciones donde el algoritmo no es capaz de

reconocer con precisión esos cambios de orientación del robot. Estos casos vienen derivados principalmente de tener al robot lejos de la cámara, como se ha comentado con anterioridad.

En aras de reunir los resultados obtenidos, en la siguiente tabla se reflejan las métricas obtenidas en cada uno de los casos, así como una comparación con las medidas reales.

		Medida calculada	**Medida real**	**E_{Abs}**	**$E_{relativo}$ (%)**
Caso 1: ejemplo 1 *(60cm rectilíneo, robot cerca de cámara)*	| Trasl |	61.22 (cm)	60 (cm)	1.22 (cm)	2.03%
	| Rot |	7.34º	0º	7.34º	-
Caso 1: ejemplo 2 *(60cm rectilíneo, robot lejos de cámara)*	| Trasl |	57.9 (cm)	60 (cm)	2.1 (cm)	3.5%
	| Rot |	33.3º	0º	33.3º	-
Caso 2: ejemplo 1 *(rotación pura 90º, robot cerca de cámara)*	| Trasl |	1.3 (cm)	0 (cm)	1.3 (cm)	-
	| Rot |	88.8º	90º	1.2º	1.35%
Caso 2: ejemplo 2 *(rotación pura 180º, robot cerca de cámara)*	| Trasl |	2.4 (cm)	0 (cm)	2.4 (cm)	-
	| Rot |	174.35º	180º	5.65º	3.13%

Caso 2: ejemplo 3 *(rotación 180º, robot lejos de cámara)*	| Trasl |	0.9 (cm)	0 (cm)	0.9 (cm)	-
	| Rot |	93.78º	180º	86.22º	47.9%
Caso 3 *(Escena compuesta de rotaciones y traslaciones)*	| Trasl |	69.4 (cm)	67.08 (cm)	2.32 (cm)	3.45%
	| Rot |	38.27º	0º	38.27º	-
Caso 4 *(Traslación con movimiento de cámara)*	| Trasl |	26.3 (cm)	30 (cm)	3.7 (cm)	12%
	| Rot |	10.91º	0º	10.91º	-

En el anexo I, el lector puede encontrar el desarrollo de más experimentos con su pertinente análisis individual.

Capítulo 7: Limitaciones

7.1 Superposición entre robots

En el caso de que dos o más robots se encuentren coincidentes a lo largo de un mismo eje de visión respecto de la cámara, las capacidades de la cámara para detectar con claridad las nubes de puntos de cada robot de forma independiente son limitadas.

Es por ello por lo que en dichas casuísticas donde exista conflicto, en la medida que la cámara sea capaz de detectar ambos robots superpuestos, la actualización de la posición y orientación de los mismos no sería efectuada y se mantendrán como referencias la ya calculadas en los momentos previos a la superposición.

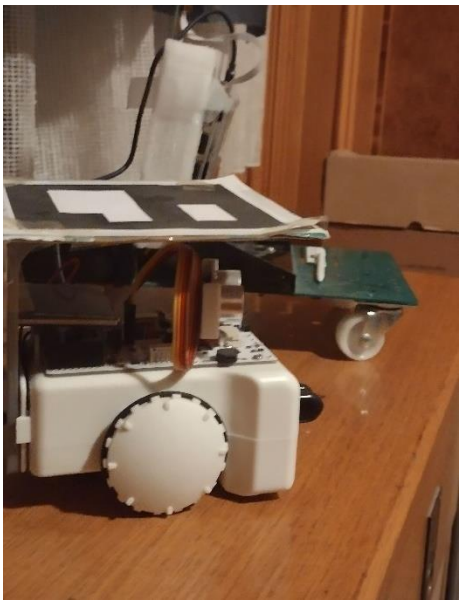

Figura 36: Superposición visual por parte del robot Sparki a una plataforma de otro robot

7.2 Pérdida de información por campo de visión

Al momento de instalar el sistema de cámaras en una posición (bien de forma externa en una posición estática o bien supeditada al movimiento que realizara un robot máster), existen circunstancias en las que los robots puedan quedar fuera del campo de visión de las cámaras. En este caso, habría perdida del tracking realizado al robot que se situara fuera de dicho campo de visión.

7.3 Reconocimiento de nuevos modelos de robots

La red neuronal con la que se trabaja ha sido entrenada a partir de una secuencia de imágenes correspondientes al robot Sparki. Este entrenamiento otorga a la red neuronal la capacidad de determinar si un objeto se trata de un robot o no.

Sin embargo, la ausencia de entrenamiento con más modelos de robots más allá de Sparki y la ausencia de potencial raciocinio humano de forma que pudiera llegar a englobar en el mismo concepto modelos que pueden llegar a ser tan dispares, obligan a ejercer un extra de precaución.

Figura 37: diferencias de diseño entre robots, a la izquierda robot Sparki, a la derecha robot BellaBot (Fuente imagen robot Bellabot: https://barcelonacolours.com)

7.4 Capacidad de cómputo para análisis en vivo

El trabajo con redes neuronales conlleva un coste computacional elevado. Dependiendo de las capacidades del equipo utilizado, el proceso puede llegar a volverse lento, ya que para manejar las complejas operaciones matemáticas involucradas es recomendable hacer uso de tarjetas gráficas (GPU).

Por tanto, en un contexto en el que se quisiera usar la implementación utilizando directamente como fuente la información que adquiere directamente

desde la cámara, habrá de estar provisto de un dispositivo con las especificaciones adecuadas.

7.5 Capacidad de reconocimiento personalizado

La implementación presentada realiza una detección a través de la imagen RGB de los posibles robots presentes en la escena. Sin embargo, y dado el mecanismo implementado, no se identifica individualmente los robots detectados. En cambio, se le asigna un número identificativo individual a cada robot según van apareciendo en escena.

Capítulo 8: Conclusiones y posibles adiciones futuras

El desarrollo de esta aplicación permite explorar un procedimiento alternativo para la obtención de la posición de distintos robots en un entorno controlado.

Se han alcanzado los objetivos propuestos inicialmente. Se ha implementado un método de grabación y almacenamiento de la información de las dos cámaras en un archivo ROSBAG, incluyéndose información relativa a la profundidad de la escena y la posición de las cámaras.

Se ha logrado realizar por técnicas de visión por computador y redes neuronales la detección del robot Sparki en diferentes escenas. Asimismo, se ha integrado un sistema de seguimiento del robot Sparki con la utilización del algoritmo SORT.

Se ha logrado también la generación de escenas 3D a partir de la información almacenada en ROSBAG y los parámetros de calibración de la cámara. También se ha realizado una filtración de puntos con el objetivo de quedar delimitados a los referentes al robot detectado. Se ha obtenido la rotación y traslación entre fotogramas con la aplicación del algoritmo ICP.

Finalmente, se presentan los resultados obtenidos de la consecución de poses calculadas y se comparan resultados con mediciones físicas en el mundo real, obteniendo una alta fiabilidad en las traslaciones del robot. En referencia a las rotaciones que realiza el robot, se aprecian dificultades en registrar la totalidad del giro practicado, principalmente en situaciones en las que el robot se encuentra lejos de la cámara. Este problema se ve reducido significativamente en situaciones donde el robot se encuentra próximo a la cámara, donde la cámara es capaz de definir con mayor precisión la figura del robot.

Se consideran un gran abanico de posibles adiciones futuras al trabajo realizado, entre las que destaca la implementación de un servidor donde se recoja y distribuya la información relativa a la pose de los robots en escena. De

esta forma, los robots podrían acceder a dicha información y servir como referencia de la odometría de los mismos.

Otra de las adiciones inminentes futuras es la ampliación del número de robots en escena y su correspondiente evaluación del sistema.

Asimismo, resultaría de interés la valoración de su idoneidad en el proceso de obtención de la posición de drones en espacios abiertos, donde opciones como los encoders dejan de tener utilidad y donde las alternativas se ven reducidas al geoposicionamiento por GPS.

Asimismo, la utilización de cámaras permite ampliar la posibilidad de usos a otros campos fuera del ámbito de cálculo de la odometría, posibilitando por ejemplo la detección de personas en lugares no autorizados o la presencia de fuego (detección visual de cualesquiera de ellos) según el tipo de instalaciones donde se le diera uso.

Referencias

[1] Intel(R) RealSense(TM), Intel RealSense SDK https://www.intelrealsense.com/sdk-2/

[2] Intel(R) RealSense(TM), Intel RealSense documentation https://dev.intelrealsense.com

[3] Arcbotics. (s.f.). *Sparki – The Easy Robot for Everyone*. http://arcbotics.com/products/sparki/

[4] Intel(R) RealSense(TM), PyRealSense2 project https://pypi.org/project/pyrealsense2/

[5] Willow Garage, Robot Operating System https://www.ros.org/

[6] Facebook's AI Research lab (FAIR), PyTorch https://www.pytorch.org/

[7] SOMBEKKE, Niels; VISSER, Arnoud. *Triangulation for depth estimation*. 2020. Tesis Doctoral. University of Amsterdam. https://staff.fnwi.uva.nl/a.visser/education/bachelorAI/Honours_Extension_Niels_Sombekke.pdf

[8] Bewley, A., Ge, Z., Ott, L., Ramos, F., & Upcroft, B. (2016, September). Simple online and realtime tracking. In *2016 IEEE international conference on image processing (ICIP)* (pp. 3464-3468). IEEE.

[9] Bewley, Alex. *SORT: Simple Online and Realtime Tracking*. GitHub, https://github.com/abewley/sort

[10] QUINTANILLA LÓPEZ-MANZANARES, Fernando. Aplicación de filtro de Kalman a la estimación de parámetros térmicos en transformadores de aceite. 2021. https://idus.us.es/bitstream/handle/11441/127690/TFG-3854-QUINTANILLA%20LOPEZ-MANZANARES.pdf?seque1nce=1&isAllowed=y

[11] CAMPANERO GARCÍA, David. Aplicación del Algoritmo Húngaro a la asignación de trabajos. pp 5-7 2018.https://oa.upm.es/51604/1/TFG_DAVID_CAMPANERO_GARCIA.pdf

[12] Rakshith Vasudev, Understanding and Calculating the number of Parameters in Convolution Neural Networks (CNNs) https://towardsdatascience.com/understanding-and-calculating-the-number-of-parameters-in-convolution-neural-networks-cnns-fc88790d530d

[13] DhanushKumar, Convolution and ReLU https://medium.com/@danushidk507/convolution-and-relu-fb69eb78dd0c

[14] BlanchR2 - Point Cloud Alignment in Open3D using the Iterative Closest Point (ICP) Algorithm https://medium.com/@BlanchR2/point-cloud-alignment-in-open3d-using-the-iterative-closest-point-icp-algorithm-22433693aa8a

[15] FERRER GUTIÉRREZ, Vicente. *Calibración de una camára por seguimiento de caracteristicas en una secuencia de imágenes*. 2004. Tesis de Maestría. https://ipicyt.repositorioinstitucional.mx/jspui/bitstream/1010/980/1/TMIPICYTF4C32004.pdf

[16] Ragon Ebker, Random Sample Consensus Explained 2023 https://www.baeldung.com/cs/ransac

[17] Pytorch (s.f.), documentación referente a la red *fasterrcnn_resnet50_fpn* https://pytorch.org/vision/main/models/generated/torchvision.models.detection.fasterrcnn_resnet50_fpn.html

[18] Roboflow. (s.f.). *Roboflow: Computer Vision Made Easy*. https://www.roboflow.com/

[19] OpenCV. (s.f.). *OpenCV Documentation*. https://docs.opencv.org/4.x/index.html

Anexos

Anexo I: Experimentos adicionales

En la presente sección se añaden ejemplos adicionales donde analizar el rendimiento del sistema implementado.

I.I Rotación 45º sobre eje del robot

Para este caso se realiza el análisis de una escena donde el robot Sparki realiza una rotación pura de 45º alrededor del eje Y en sentido antihorario. Las cámaras permanecen estáticas.

A partir de la implementación desarrollada, se extrae la siguiente matriz homogénea de transformación:

$$(R|t) = \begin{pmatrix} 0.7413 & -0.1567 & -0.6525 & -0.0008 \\ 0.1931 & 0.9810 & 0.0161 & -0.0009 \\ -0.6427 & 0.1140 & 0.7575 & -0.0056 \\ 0 & 0 & 0 & 1 \end{pmatrix} [metros]$$

Realizando el análisis de la composición, podemos determinar cómo ha permanecido estática la traslación, más allá del ruido que viene implícito en este tipo de sistemas. Respecto a la rotación, se calcula el ángulo girado a partir de la siguiente expresión:

$$theta = \text{acos}\frac{Traza(R) - 1}{2} = 42.39º$$

Para esta escena, el aplicativo devuelve el ángulo con un error relativo en torno al 5%. Como se ha podido comprobar en casos anteriores del capítulo 6, existe una incertidumbre mayor en la respuesta de la rotación comparado con la traslación. Para minimizar dicho error, se recomienda situar las cámaras próximas al robot al que se le estima su rotación.

En la figura incluida a continuación, se muestra el resultado de la composición de matrices a lo largo de la rotación que efectúa.

Figura 38: resultado de la composición consecutiva de matrices para el ejemplo de rotación de 45º

I.II Rotación 135º sobre eje del robot

Para completar los ejemplos de rotaciones puras, se opta por realizar una rotación pura sobre el eje vertical de 135º. La dinámica es equivalente al resto de ejemplos de rotaciones, donde se comenta el resultado de la matriz que componen cada una de las traslaciones detectadas entre fotogramas.

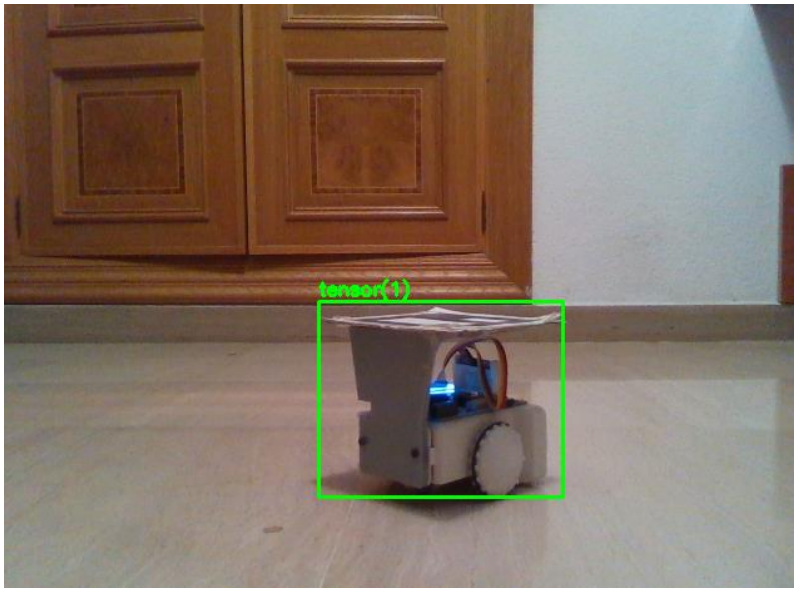

Figura 39: una de las escenas finales del ejemplo, con bounding box

Una vez ejecutado, se toma la respuesta de nuestro algoritmo en forma de matriz de transformación para su análisis:

$$(R|t) = \begin{pmatrix} -0.4219 & -0.1581 & 0.8927 & -0.0117 \\ 0.1738 & 0.9522 & 0.2508 & -0.0087 \\ -0.8898 & 0.2610 & -0.3742 & -0.0158 \\ 0 & 0 & 0 & 1 \end{pmatrix} [metros]$$

La rotación obtenida es de 114.83º, siendo éste un resultado menor al esperado (error en torno al 15%). En cuanto a la traslación, se detecta un pequeño registro de traslación de 0.016 metros.

En la figura 40 mostrada a continuación queda en evidencia la falta de unos grados para alcanzar los 135º:

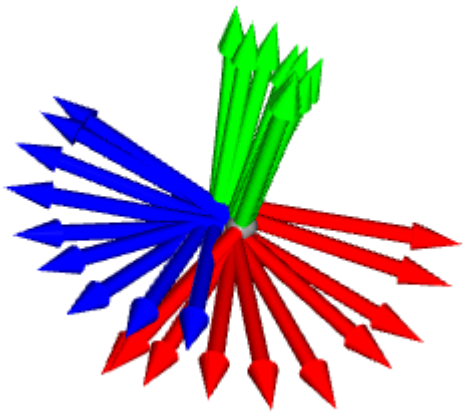

Figura 40:representación de la acumulación de rotación del ejemplo

I.III Traslación diagonal del robot sobre plano X-Z

En este experimento se realiza un movimiento rectilíneo a lo largo de la bisectriz de los ejes del plano X-Z. El desplazamiento se realiza en sentido negativo del eje X y positivo en Z. Se incluye a continuación una imagen de referencia para simplificación de la comprensión.

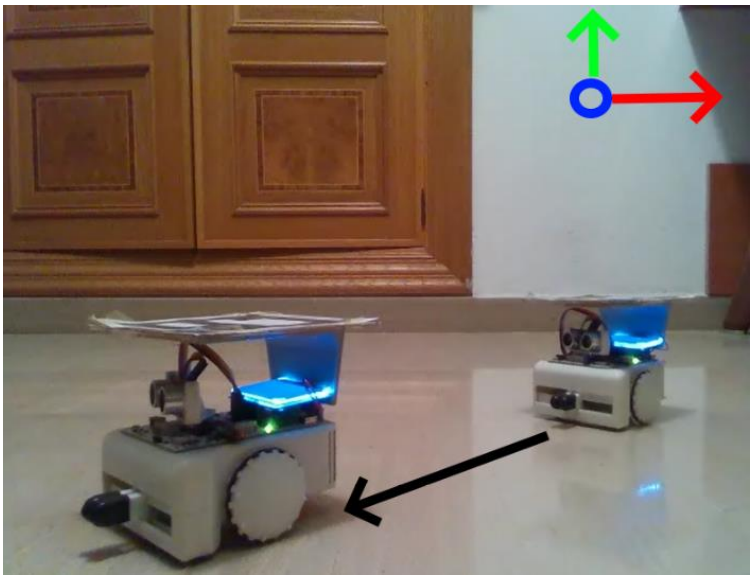

Figura 41: recorrido que realiza el robot Sparki en el ejemplo

Además, se obtiene la representación de la traslación que ha ido obteniendo, en consonancia con el movimiento descrito:

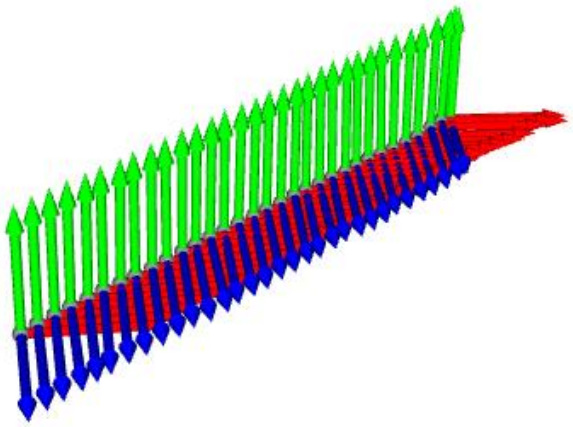

Figura 42: recorrido detectado a lo largo de la diagonal.

Igualmente, el programa nos devuelve el valor de la matriz de transformación entre el valor inicial y final.

$$(R|t) = \begin{pmatrix} 0.9852 & 0.0335 & 0.1675 & -0.3222 \\ -0.0313 & 0.9993 & -0.0156 & -0.0202 \\ -0.1679 & 0.0101 & 0.9857 & 0.2948 \\ 0 & 0 & 0 & 1 \end{pmatrix} [metros]$$

Primeramente, indicar que la traslación la ha compuesto en los ejes X negativo e Y positivo, conforme a lo esperado según la descripción del ejercicio. La norma devuelve un valor de traslación de 0.4375 metros.

Respecto a la rotación, se detecta una rotación leve de 10,1º, siendo una respuesta del sistema en consonancia con el resto de ejercicios, donde se van acumulando igualmente pequeños errores en la rotación.

Como referencia, se extrae además las coordenadas que refieren a los centroides tanto en el fotograma inicial y como en el final:

$$T_{inicial} = (X = 0.2114, \quad Y = -0.0347, \quad Z = -0.7268)\ [metros]$$

$$T_{final} = (X = -0.0974, \quad Y = -0.0493, \quad Z = -0.4281)\ [metros]$$

La distancia euclídea entre estas dos coordenadas son 0.4298 metros. Se trata de un valor muy próximo al calculado a través del trabajo desarrollado (0.4375 metros), lo que hace indicar el buen funcionamiento del algoritmo ICP en las traslaciones de esta índole.

Anexo II: Código implementado

Todo el código desarrollado y que ha sido utilizado para la realización de este trabajo se puede encontrar en el siguiente enlace de GitHub:

https://github.com/GM-Daniel/visual-odometry-robots