Universidad Nacional de
Educación a Distancia

Escuela Técnica Superior de
Ingeniería Informática

Universidad
Complutense de Madrid

Facultad de Informática

Creación de mapas fotográficos 3D del interior de edificaciones mediante el uso de drones

Alumno: **Efrén Albisu Iso**

Directores: **José Sánchez Moreno y David Moreno Salinas**

Trabajo de Fin de Máster

Máster Universitario en Ingeniería de Sistemas y Control

09/06/2025

Máster en Ingeniería
de Sistemas y Control

Creación de mapas fotográficos 3D del interior de edificaciones mediante el uso de drones

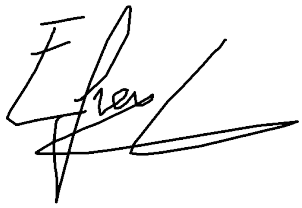
Proyecto específico propuesto por un profesor

Alumno: **Efrén Albisu Iso**

Directores: **José Sánchez Moreno y David Moreno Salinas**

Autorización

Autorizamos a la Universidad Complutense y a la UNED a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firmado: 

Resumen

Este Trabajo de Fin de Máster aborda la aplicación de técnicas de localización y mapeo simultáneo visual (VSLAM) en vehículos aéreos no tripulados (UAVs), con el objetivo de evaluar su viabilidad para la navegación autónoma en entornos sin GPS. Para ello, se ha partido de un dron comercial de bajo coste con cámara incorporada, se ha calibrado la cámara y se ha implementado un sistema VSLAM utilizando las imágenes tomadas por el dron. El sistema ha sido evaluado mediante la generación de mapas 3D del entorno y la localización de imágenes tomadas por el dispositivo, demostrando resultados satisfactorios en términos de precisión, tanto en la creación del mapa como en la localización. Este trabajo concluye con una reflexión sobre las posibles mejoras del sistema y las líneas de investigación futura en el campo de la navegación visual autónoma en UAVs de bajo coste.

Palabras clave: Crazyflie, dron, ORB, SLAM, VSLAM, puntos clave, *feature tracking*, triangulación.

Abstract

This Master's Thesis addresses the application of Visual Simultaneous Localization and Mapping (VSLAM) techniques in Unmanned Aerial Vehicles (UAVs), with the aim of evaluating their feasibility for autonomous navigation in GPS-denied environments. To this end, a low-cost commercial drone with an integrated camera was used as the starting point, the camera was calibrated, and a VSLAM system was implemented using the images captured by the drone. The system was evaluated through the generation of 3D maps of the environment and the localization of images taken by the device, showing satisfactory results in terms of accuracy in both map creation and localization. The work concludes with a reflection on possible improvements to the system and future research directions in the field of visual autonomous navigation in low-cost UAVs.

Keywords: Crazyflie, drone, ORB, SLAM, VSLAM, keyframes, feature tracking, triangulation.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

1.1. Qué es SLAM

El SLAM viene del inglés “Simultaneous Location And Mapping”, que traducido significa Localización y Mapeo Simultáneos. Es un método utilizado en robots, vehículos autónomos o cualquier tipo de dispositivo con sensores que permite crear un mapa y localizar el robot, vehículo o dispositivo en el mapa al mismo tiempo. Los algoritmos de SLAM permiten que el dispositivo se sitúe en una posición desconocida dentro de un sistema también desconocido y construir de forma incremental un mapa del entorno mientras simultáneamente determina su posición utilizando el mapa.

El SLAM ha sido un área de intensa investigación técnica durante las últimas décadas. Los avances significativos en la velocidad de procesamiento computacional, junto con la creciente disponibilidad de sensores de bajo coste como las cámaras, han permitido que los algoritmos de SLAM pasen del ámbito teórico a aplicaciones prácticas en una amplia variedad de campos.

Los algoritmos de SLAM son útiles en muchas otras aplicaciones, tales como controlar la navegación de una flota de robots móviles para organizar estanterías en un almacén, estacionar un automóvil autónomo en una plaza vacía, o entregar un paquete por dron en un entorno desconocido, entre otras.

1.2. Cómo funciona SLAM

El SLAM es un proceso en el que un robot móvil puede construir un mapa de su entorno y al mismo tiempo usar ese mapa para saber su localización. En SLAM, la trayectoria del robot y las posiciones de los puntos de referencia o hitos están estimadas en línea sin la necesidad de un

conocimiento previo de la localización. En el problema del SLAM se considera un robot moviéndose a través de un entorno realizando observaciones de un número de puntos característicos desconocidos mediante un sensor. La Figura 1 ilustra la estructura del problema del SLAM.

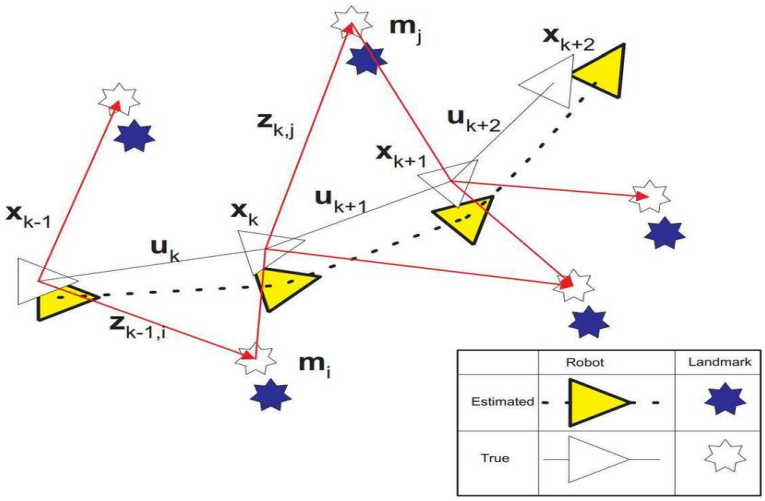


Figura 1. Ilustración del problema de SLAM. Fuente [1].

Para el instante ***k*** las siguientes variables están definidas:

$\boldsymbol{X_k}$: Vector de estado describiendo la posición y la orientación del vehículo.

$\mathbf{U_k}$: Vector de control aplicado en el instante 1−k para llevar al vehículo desde su posición anterior hasta la actual $\mathbf{k_x}$ en el instante **k** .

$\mathbf{m_i}$: Vector que describe la localización del hito i-ésimo, cuya localización se asume a través del tiempo invariante.

$\mathbf{Z_{k,i}}$: Observación tomada desde el robot de la localización del i-ésimo hito en el instante k . Cuando haya múltiples observaciones de diferentes hitos en el mismo instante o cuando un hito específico no sea relevante la observación se escribirá para simplificar como $\mathbf{Z_k}$

En términos generales, SLAM utiliza dos tipos de componentes tecnológicos. El primer tipo es el procesamiento de señales de sensores, incluido el procesamiento frontal, que depende

en gran medida de los sensores utilizados. El segundo tipo es la optimización de grafos de pose, incluido el procesamiento posterior, que es independiente de los sensores. La Figura 2 resume los principales componentes técnicos necesarios para aplicar las técnicas de SLAM.

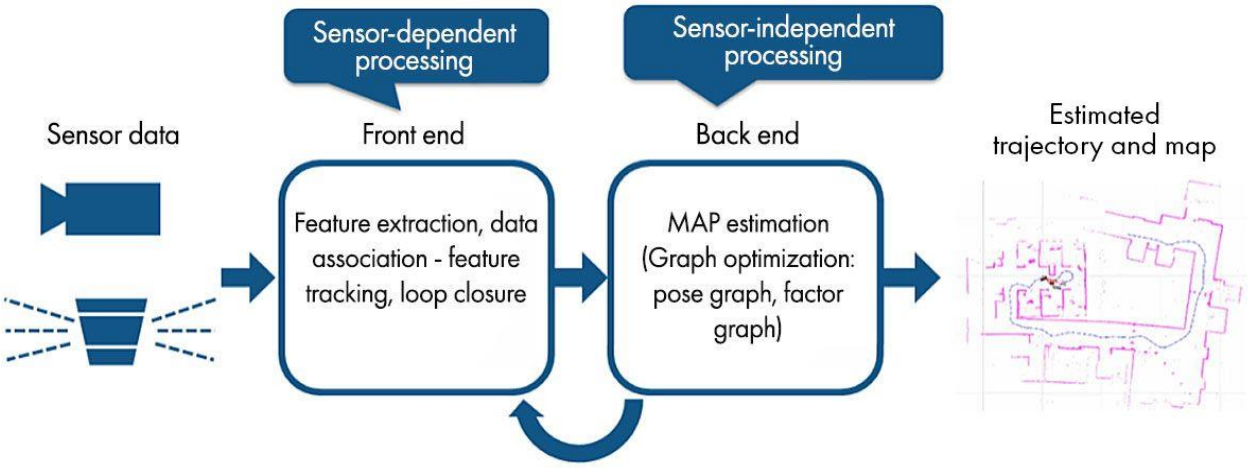


Figura 2. Resumen de la estructura de las tecnicas SLAM. Fuente [2].

1.3. Métodos de SLAM

Dependiendo del tipo de sensores usados para obtener información del entorno, los métodos de SLAM se pueden diferenciar en dos grandes grupos. Los sistemas que usan Lidar y los sistemas que usan cámaras. Además, también se han ido añadiendo nuevos sensores para mejorar y complementar la información obtenida por los métodos anteriores, estos sensores pueden ser IMU (unidad de medida inercial) o GPS (sistema de posicionamiento global).

1.3.1. SLAM Lidar

En este método se usa la información obtenida por un LiDAR (Light Detection and Ranging), para la medición de las distancias entre nuestro dispositivo y el entorno. Es un método muy preciso, incluso en condiciones de poca luz o de noche y se utiliza en aplicaciones donde se

requiere una alta velocidad en la adquisición de los datos, como por ejemplo la conducción de vehículos autónomos. La principal desventaja de esta tecnología es el precio.

A su vez, según el tipo de sensor empleado, el método se puede dividir en dos grupos: SLAM Lidar 2D y 3D. El método 2D hace escaneos es un solo plano, mientras que el 3D hace un escaneo tridimensional. La Figura 3 muestra un Lidar 3D comercial, fabricado por Feima Robotics [3].

Figura 3. Lidar 3D, modelo SLAM100, fabricado por Feima Robotics. Fuente [3].

1.3.2. SLAM Visual

Como su nombre indica, SLAM visual (o VSLAM) utiliza imágenes captadas mediante cámaras y otros sensores de imagen para obtener información del entorno y estimar el movimiento del dispositivo. Es un método que se caracteriza porque se puede implementar a bajo coste con cámaras relativamente económicas, pero que depende de que el entorno tenga iluminación y texturas adecuadas.

Dependiendo del tipo de cámara o cámaras utilizadas (Figura 4), se puede clasificar en

SLAM monocular, SLAM estéreo y SLAM RGB-D. El SLAM monocular es un tipo de algoritmo de SLAM en el que VSLAM utiliza una sola cámara como único sensor, lo que dificulta definir la profundidad. El sistema SLAM estéreo utiliza dos cámaras y por lo tanto se puede estimar la profundidad directamente. Finalmente, el SLAM RGB-D utiliza cámaras con sensor de profundidad, permitiendo que la profundidad sea una medida y no una estimación, aunque son más caros.

Figura 4. Diferentes ejemplos de cámaras para SLAM visual. A la derecha una cámara monocular, en el centro una cámara estéreo [4], y a la derecha una cámara RGB-D [5].

Por otro lado, los métodos VSLAM se pueden clasificar en dos grupos dependiendo del procesado que se hace con las imágenes: métodos dispersos (también llamados métodos basados en características) y los métodos directos (también conocido como métodos densos). Los métodos dispersos procesan las imágenes para obtener solo algunos puntos de interés (*keypoints*). Son métodos rápidos a nivel de cálculo computacional y no necesitan mucha memoria. En cuanto a los métodos densos, utilizan para el procesado toda la información de las imágenes, es decir, cada pixel de la imagen es utilizado para la creación del SLAM. Con estos métodos, se pueden conseguir mapas del entorno muy detallados, pero la carga computacional y la memoria utilizada son enormes. La Figura 5 muestra ejemplos de los diferentes resultados que

se pueden obtener al aplicar métodos dispersos o directos.

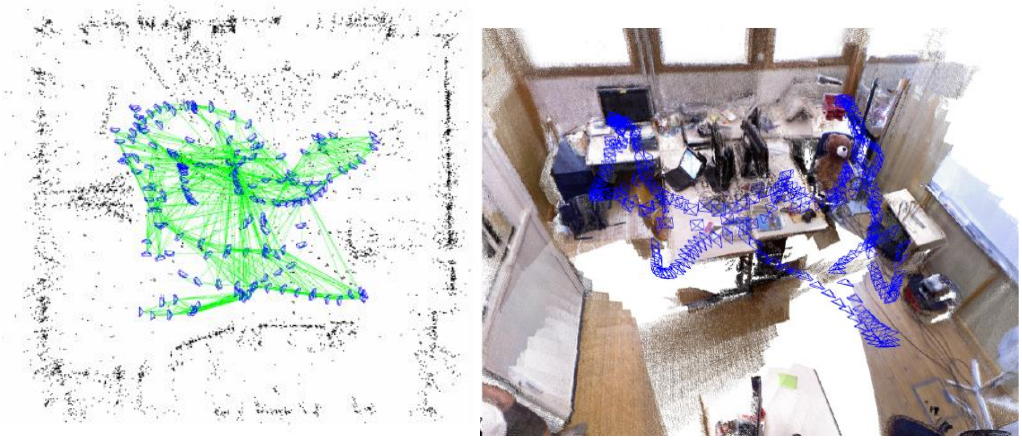


Figura 5. Ejemplos de SLAM visual disperso (izquierda) y directo (derecha). Fuente [6].

En este proyecto nos vamos a centrar en el SLAM visual, y más en concreto en el monocular SLAM basado en métodos dispersos.

1.3.3. SLAM multisensor

El SLAM multisensor es un tipo de algoritmo de SLAM que utiliza diversos sensores, como cámaras, IMU (unidades de medición inercial), GPS, LiDAR y radar, entre otros, para mejorar la precisión y solidez de los algoritmos de SLAM. Dado que aprovecha las ventajas y supera las limitaciones de diferentes sensores, el SLAM multisensor logra alcanzar un rendimiento superior. Si bien las cámaras proporcionan datos visuales detallados, son menos eficaces en situaciones de poca luz o alta velocidad. Por su parte, LiDAR funciona de forma sistemática en diversas condiciones de iluminación, pero puede tener dificultades con ciertas texturas. El SLAM multisensor ofrece una solución más fiable que un sensor único, ya que integra datos de diversas fuentes. La principal desventaja es la complejidad para integrar todos los sensores y utilizarlos de

forma coordinada y el precio asociado a medida que se añaden más sensores al sistema.

1.4. Evolución del SLAM visual

El primer trabajo publicado relacionado con el SLAM visual data de 2007. Desde entonces, se han logrado importantes avances en el desarrollo de esta técnica. La Figura 6 presenta un resumen de los hitos más relevantes en la evolución del SLAM visual desde 2007 hasta la actualidad.

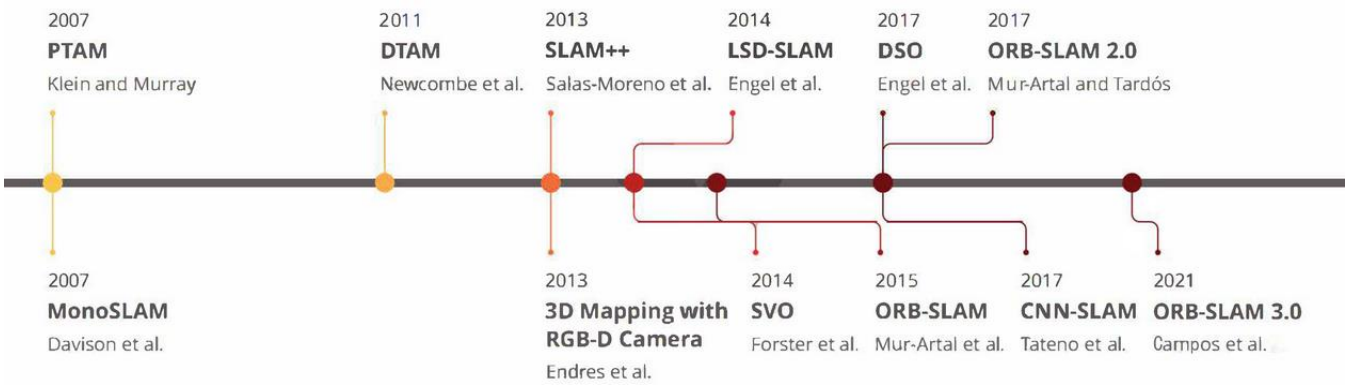


Figura 6. Principales proyectos relacionados con SLAM visual. Fuente [7].

El proyecto pionero en SLAM visual fue presentado en 2007 por Andrew Davison en su artículo sobre MonoSLAM [8], en el que se introdujo el primer algoritmo capaz de realizar SLAM en tiempo real utilizando únicamente imágenes de una cámara. MonoSLAM, acrónimo de *monocular SLAM*, se caracteriza por emplear una única cámara como sensor, en contraste con los sistemas estéreo que requieren múltiples dispositivos. En aquel momento, el sistema necesitaba una GPU (*Graphics Processing Unit*) para poder ejecutarse y funcionaba principalmente como una prueba de concepto, ya que presentaba limitaciones significativas: consumía rápidamente la memoria disponible y solo era capaz mapear entornos reducidos, como

habitaciones. Tras la publicación y ante el creciente interés de la comunidad, Davison puso el código a disposición pública a través de su sitio web.

De forma paralela al desarrollo de MonoSLAM, surgió PTAM (*Parallel Tracking and Mapping*) [9], cuya publicación tuvo lugar en el mismo año, apenas unos meses más tarde. Curiosamente, aunque MonoSLAM recibió mayor reconocimiento inicial por haber sido el primero en ser publicado, la mayoría de los sistemas modernos de SLAM visual han adoptado una arquitectura basada en los principios introducidos por PTAM. Mientras que el enfoque de MonoSLAM ha sido prácticamente descartado, PTAM marcó un cambio de paradigma. Este enfoque que PTAM introdujo, y que años más tarde se consolidó en la comunidad, es lo que hoy se conoce como SLAM visual indirecto.

En el 2011, el mismo laboratorio creador de MonoSLAM presentó *DTAM: Dense Tracking and Mapping in Real-Time*, el primer sistema directo que no se concentra en puntos sino que utiliza la imagen completa, denominando por oposición SLAM visual indirecto al resto de los sistemas.

En 2014 la Universidad Técnica de Múnich (TUM) publicó LSD-SLAM (*Large Scale Direct SLAM*) [10], iniciando la categoría SLAM visual semidirecto y marcando un hito histórico: fue el primer sistema capaz de ejecutarse en tiempo real en una PC sin GPU y de los primeros en publicar su código fuente como código abierto, lo que facilitó la instalación y prueba a miles de aficionados que popularizaron las posibilidades de SLAM visual. A partir del desarrollo de LSD-SLAM, los sistemas de SLAM visual comenzaron a orientarse hacia la ejecución en tiempo real sin necesidad de una GPU, utilizando únicamente un ordenador personal. Además, se consolidó la práctica de publicar el código fuente en abierto, con el objetivo de fomentar la colaboración de la

comunidad científica en la depuración, validación y mejora de los algoritmos.

ORB-SLAM (2015) [11] ha sido uno de los sistemas de SLAM visual más influyentes, y sus aportaciones técnicas marcaron un antes y un después en el desarrollo de algoritmos robustos y eficientes. Publicado como código abierto [12], se consolidó rápidamente como referente el estado del arte. La principal novedad de ORB-SLAM es que integra todos los componentes necesarios para formar un algoritmo de SLAM robusto. Tanto los algoritmos de *frontend* como de *backend* están diseñados de manera eficiente, priorizando la precisión sin comprometer el rendimiento en tiempo real.

ORB-SLAM utiliza ORB para la extracción de características, pero no todos los sistemas de SLAM que emplean ORB como extractor de características pueden considerarse ORB-SLAM. Es decir, aunque un sistema use ORB, eso no significa que sea una implementación de ORB-SLAM.

A continuación, se enumeran las principales contribuciones de ORB-SLAM:

- Uso de características ORB para todas las tareas: Utiliza las mismas características ORB para el seguimiento, mapeo, relocalización y cierre de bucles, lo que asegura eficiencia, simplicidad y fiabilidad.
- Operación en tiempo real en grandes entornos: Enfoca el seguimiento y el mapeo en un área local de covisibilidad mediante un grafo de covisibilidad, lo que hace que el rendimiento sea independiente del tamaño total del mapa.
- Cierre de bucles en tiempo real: Optimiza un grafo de poses llamado *Essential Gr*aph, construido a partir de un árbol generador mínimo, enlaces de cierre de bucle y conexiones fuertes del grafo de covisibilidad.
- Relocalización de cámara en tiempo real: Ofrece una gran invariancia ante cambios de

perspectiva e iluminación, lo que permite la recuperación tras fallos de seguimiento y favorece la reutilización de mapas.

- Procedimiento de inicialización robusto: Inicializa el mapa de forma automática y robusta, seleccionando entre modelos adecuados tanto para escenas planas como no planas.

- Supervivencia del más apto para puntos de mapa y *keyframes*: Adopta una política de generación generosa pero eliminación estricta, mejorando la robustez del seguimiento y soportando una operación continua al eliminar *keyframes* redundantes.

Además, ORB-SLAM dio origen a dos extensiones posteriores:

- ORB-SLAM2 (2017) [12]: Introduce la creación y gestión de múltiples mapas. Extiende las capacidades originales para trabajar con cámaras RGB-D y monoculares, permitiendo un mapeo y relocalización eficientes en distintos entornos. También está publicado como código abierto [13].

- ORB-SLAM3 (2021) [14]: Integra mediciones inerciales (IMU) junto con los datos visuales para mejorar la robustez y la precisión. También soporta sistemas de cámaras estéreo y estéreo-inerciales, ofreciendo así mayor versatilidad y precisión en tareas de mapeo y localización 3D. También está publicado como código abierto [15].

1.5. Motivación

En los últimos años, los vehículos aéreos no tripulados (UAVs o drones) han ganado protagonismo en múltiples sectores, desde la inspección industrial y la agricultura de precisión, hasta la exploración de entornos inaccesibles y la asistencia en situaciones de emergencia. Una de las limitaciones clave para la autonomía de estos dispositivos es su capacidad para localizarse y mapear su entorno en tiempo real, especialmente en entornos donde no es posible disponer de señal GPS o en los que las condiciones estructurales varían dinámicamente.

En este contexto, las técnicas de Simultaneous Localization and Mapping (SLAM), y en particular las basadas en visión (VSLAM), se han consolidado como una solución prometedora para dotar a los drones de capacidades de navegación autónoma más avanzadas y fiables. VSLAM permite estimar simultáneamente la trayectoria del dron y construir un mapa del entorno utilizando exclusivamente información visual obtenida por cámaras, lo que lo convierte en una alternativa atractiva por su bajo coste, versatilidad y escalabilidad.

La motivación principal de este trabajo radica en la necesidad de explorar, analizar y optimizar el uso de algoritmos de VSLAM en plataformas aéreas comerciales de bajo coste.

1.6. Propuesta y objetivos

El principal objetivo de este trabajo es demostrar que se pueden aplicar técnicas VSLAM a las imágenes grabadas por el dron comercial Crazyflie 2.0 para crear modelos 3D del entorno. Este modelo 3D será un mapa del entorno de nuestro dispositivo y además el dispositivo será capaz de localizarse dentro de este mapa 3D.

En nuestro caso nos vamos a centrar en entornos controlados de interior donde no hay cambios a lo largo de la grabación, es decir es una escena estática. Por otro lado, debido a la propia configuración del dron comercial, solo disponemos de una cámara, por lo tanto todo el proyecto se centrará en técnicas de VSLAM monocular.

Dentro de las técnicas de SLAM visual, para el desarrollo de nuestro proyecto nos centraremos en las técnicas dispersas, basadas en características. Son técnicas que proporcionan menos detalle del entorno pero son más robustas. En este trabajo hemos usado como referencia las técnicas y estructura aplicada en el proyecto publicado por la ORB-SLAM [10].

1.7. Estructura del documento

El presente Trabajo de Final de Máster se estructura en cuatro capítulos, distribuidos de la siguiente manera:

- Capítulo 1: Introducción. Se presenta el contexto general del proyecto, centrado en la localización y mapeo simultáneo visual (VSLAM) aplicado a drones. Se introducen los fundamentos del SLAM, su funcionamiento básico y la evolución histórica de esta tecnología
- Capítulo 2: Materiales y métodos. Se detallan los recursos empleados para el desarrollo del proyecto, incluyendo el entorno de desarrollo, las características técnicas del dron utilizado y el proceso de calibración de la cámara. Además, se describen las técnicas aplicadas, así como el flujo de trabajo seguido durante la implementación del sistema.
- Capítulo 3: Resultados y discusión. Se presentan los resultados obtenidos a partir de la ejecución del sistema, incluyendo la generación de mapas tridimensionales y la localización visual a partir de imágenes. Se analiza el comportamiento del sistema y se discuten los aspectos más relevantes de su rendimiento.
- Capítulo 4: Conclusiones y líneas futuras. Se resumen los principales hallazgos del proyecto, se evalúa el grado de cumplimiento de los objetivos y se proponen posibles mejoras y líneas de trabajo futuro para continuar con el desarrollo

Capítulo 2. Materiales y métodos

En esta sección se explica el entorno de desarrollo utilizado en este proyecto, incluyendo los detalles técnicos del ordenador utilizado para implementar las técnicas VSLAM, el sistema operativo utilizado, los lenguajes de programación y librerías utilizadas. Por otro lado, también se profundiza en los detalles de la plataforma comercial Crazyflie, que es el drone y accesorios, que se van a emplear para hacer la grabación y envío de las imágenes. También se explica cómo se ha calibrado la cámara utilizada por el drone. Finalmente, se explican los métodos y cómo se han estructurado los mismos para poder aplicar VSLAM monocular a las imágenes obtenidas por el drone

2.1. Entorno de desarrollo

Para el desarrollo de este proyecto se ha utilizado un ordenador portátil con las siguientes características:

- Intel Core i3 processor 350M (2.26GHz, 3MB L3 cache)
- 4 GB RAM
- 1 TB SSD

A nivel de sistema operativo, este ordenador tiene un sistema operativo dual:

- Linux Mint 21.3 Virginia
- Windows 10

Se han utilizado ambos sistemas operativos para algunas de las fases de configuración del dron comercial. Sin embargo, debido a problemas de rendimiento, solo las fases iniciales de configuración del dron se han hecho en Windows 10. En general, el rendimiento de todas las herramientas proporcionadas por el fabricante del dron comercial utilizado en este proyecto

funcionan mejor en Linux.

Los lenguajes de programación utilizados en este proyecto son Python y Matlab, empleando las siguientes librerías:

- Python 3.8, con las librerías OpenCV, numpy, cfclient, cflib
- Matlab 2022b y la librería Computer Vision Toolbox

Python y sus librerías se han utilizado para realizar la lectura y guardado de las imágenes enviadas por el dron vía WiFi. Matlab y su librería se han utilizado para la implementación del conjunto de algoritmos necesarios para aplicar VSLAM a las imágenes enviadas por el dron y para el procesamiento de los datos.

2.2. Plataforma de desarrollo Crazyflie 2.x

La plataforma de desarrollo Crazyflie 2.x, es una plataforma de desarrollo de código abierto creada por la empresa Bitcraze [16]. Esta plataforma está diseñada para la investigación y prototipado rápido en robótica aérea. Se trata de un micro dron ligero y modular, que permite el control preciso y la integración de sensores gracias a su arquitectura abierta y soporte para múltiples entornos de programación. A su vez, la plataforma incluye aplicaciones, librerías para controlar el dron y también para el resto de accesorios hardware.

En las siguientes secciones se detallan cuáles son los componentes de la plataforma Crazyflie 2.x que se han utilizado en este proyecto.

A nivel de desarrollo y vuelo del dron, el propio fabricante proporciona una máquina virtual donde incluye todos los programas necesarios para el desarrollo y el vuelo. Sin embargo, también es posible hacer una instalación completa del entorno, aproximación que se va a seguir en este trabajo. Como referencia para la preparación del entorno de trabajo se ha tenido en

cuenta la información proporcionada en la web del fabricante pero también el Trabajo Final de Máster realizado por Juan Miguel Sánchez García con el título “Creación de mapas fotográficos 3D del interior de edificaciones mediante el uso de drones“ (Febrero-2024) [17]. Los principales pasos son los siguientes: configurar la Crazyradio 2.0, instalar la aplicación cfclient, actualizar las versiones de firmware, configurar el PC para guardar las imágenes enviadas por el dron y, finalmente, se incluye una sección sobre las diferentes formas de volar el dron.

2.2.1. Hardware dron Crazyflie 2.0

El quadricoptero utilizado en este proyecto es el modelo Crazyflie 2.0 (Figura 7). Este modelo forma parte de la plataforma de desarrollo Crazyflie 2.x. El firmware es de código abierto y la flexibilidad de la plataforma la hace ideal para investigación, educación u otras aplicaciones donde la apertura y el control total sean importantes. Otro punto a remarcar de la plataforma es su tamaño y peso reducido. En el momento de escribir esta memoria, el modelo usado para la misma se ha descatalogado, pero dentro de la familia de nano quadricopteros se pueden encontrar otros modelos recientemente presentados y con características muy similares (Tabla 1).

Figura 7. Crazyflie 2.0. Fuente [18].

Modelo	Crazyflie 2.0 [18]	Crazyflie 2.1 [19]	Crazyflie 2.1+ [20]	Crazyflie 2.1 Brushless [21]
Estado	Discontinuado	Discontinuado	Activo	Activo
Fecha lanzamiento	2014	2019	2024	2025
Tiempo de vuelo	~7 minutos	~7 minutos	~7 minutos	~10 minutos
Peso	27gr	27gr	27gr	27gr
Motores	Motores con escobillas	Motores con escobillas	Motores con escobillas	Motores sin escobillas
IMU y sensores	MPU-9250 (IMU 9 ejes) LPS25H (presión)	BMI088 (IMU 6 ejes) BMP388 (presión)	BMI088 (IMU 6 ejes) BMP388 (presión)	BMI088 (IMU 6 ejes), BMP388 (presión)
Observaciones	Es el primer dron desarrollado dentro del entorno Crazyflie 2.x	Sensores mejorados, con detección automática de *decks*, más robusto que el 2.0	Batería y hélices actualizadas para mejorar los tiempos de vuelo	Ligeramente mayor peso, pero con las mismas dimensiones. Los motores sin escobillas le permiten mayor peso de carga

Tabla 1 Resumen de la familia de drones Crazyflie 2.x.

Todos los modelos son compatibles con la mayoría de los *decks* de expansión de Bitcraze (excepto el LED-ring en el modelo *Brushless*) y ofrecen conectividad vía Bluetooth LE y radio de 2.4 GHz. Además, permiten actualizaciones de firmware inalámbricas y son compatibles con sistemas operativos como Windows, macOS y Linux.

2.2.2. Especificaciones técnicas Crazyflie 2.0

Todos los detalles sobre los productos de Crazyflie se pueden encontrar en la web del fabricante, pero a continuación están resumidos los detalles técnicos principales del Crazyflie 2.0 [22].

Especificaciones mecánicas:

- Peso al despegue: 27g.
- Dimensiones: 92x92x92mm.

Especificaciones de los microcontroladores:

- MCU principal STM32F405 (Cortex-M4, 168MHZ, 192kb SRAM, 1Mb flash).
- MCU nRF51822 para control de radio y potencia (Cortex-M0, 32Mhz, 16kb SRAM, 128kb flash).
- Conector micro-USB.
- Cargador LiPo de 100mA, 500mA y 980mA.
- Interfaz USB alta velocidad.
- Capacidad USB OTG.
- 8KB EEPROM.

Especificaciones de la IMU:

- 3 ejes acelerómetros / giróscopo (BMI088).
- Sensor de presión de alta precisión (BMP388).

Especificaciones de vuelo:

- Tiempo de vuelo con la batería: 7 minutos.
- Tiempo de carga de la batería: 40 minutos.
- Máxima carga útil: 15g.

Especificaciones de la transmisión de radio:

- Banda de radio de 2.4GHz ISM.
- Amplificador de alcance de 20dBm, alcance máximo a 1 km (línea de visión).
- Soporte bluetooth para clientes iOS y Android.

- Antena dual para chip y conector U.FL.

Especificación de los conectores de expansión:

- VCC 3.0V (máx. 100mA).
- GND.
- VCOM (unregulated VBAT o VUSB, max 1A).
- VUSB (entrada y salida).
- I2C (4000kHz).
- SPI (Interfaz de Periféricos Serie).
- 2 x UART (Transmisor Receptor Asíncrono Universal).
- 4 x GPIO/CS para SPI.
- 1 bus cableado.
- 2 x GPIO conectado a nRF51.

2.2.3. Accesorios hardware

Para llevar a cabo el proyecto, además del dron, se han utilizado otros accesorios hardware dentro del eco-sistema Crazyflie 2.x. En concreto, se han utilizado dos *expansion decks* [22] y el adaptador CrazyRadio 2.0 [23]. Los *expansion decks* mejoran el dron Crazyflie 2.0 y en este proyecto se ha utilizado la AI-deck 1.1 y el Flow deck v2.

El adaptador AI-deck 1.1 (Figura 8) expande las capacidades de computación del Crazyflie 2.0, permitiendo comunicaciones inalámbricas, cálculos complejos basados en algoritmos de inteligencia artificial y navegación plenamente autónoma. Asimismo, incorpora una cámara, que será utilizada para la captura de imágenes para nuestro VSLAM monocular.

Figura 8. AI-deck 1.1. Fuente [24].

El adaptador Flow deck v2 (Figura 9) permite detectar los movimientos en cualquier dirección. Consta de dos sensores: uno para medir la distancia al suelo y otro para detectar movimientos mediante flujo óptico.

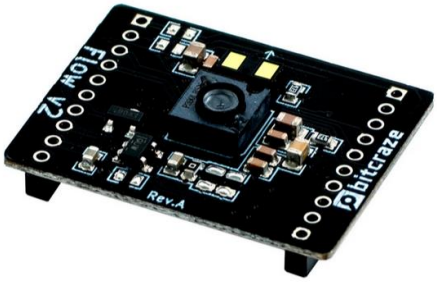


Figura 9 Flow deck v2. Fuente [25].

El adaptador Crazyradio 2.0 (Figura 10) es un dispositivo USB de radio de largo alcance. Permite la ejecución de código en el dron desde el entorno de programación y la carga de código en el AI-deck.

Figura 10. CrazyRadio 2.0. Fuente [23].

2.2.4. Configurar CrazyRadio 2.0

Como ya se ha comentado anteriormente, la CrazyRadio 2.0 se usa para cargar código en el AI-deck y para poder controlar el vuelo del dron usando la aplicación *cfclient*. En este apartado, se explican los pasos necesarios para configurar la Crazyradio 2.0 en Windows y en Linux.

Para Windows 10, es necesario conectar el Crazyradio 2.0 al USB del PC y seguir las indicaciones [26]. Una vez completada la instalación, se puede confirmar que se han instalado correctamente revisando en el administrador de dispositivos (Figura 11).

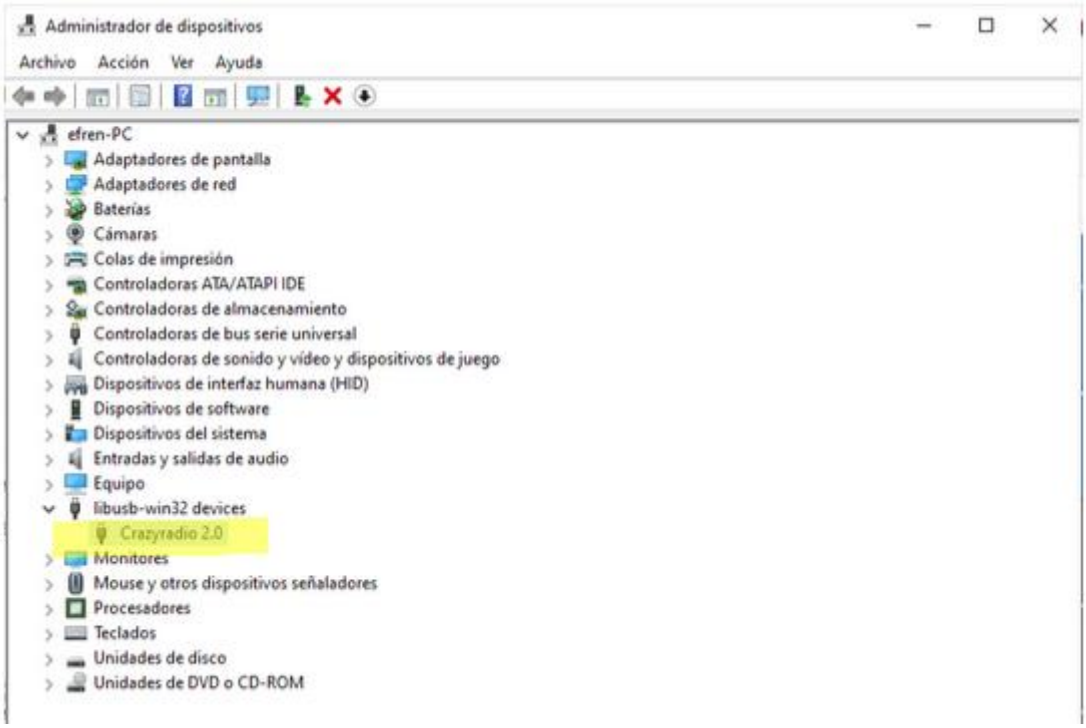


Figura 11. Administrador de dispositivos Windows 10 con CrazyRadio 2.0 correctamente instalada.

Para Linux, el proceso es más sencillo y directamente al conectar el Crazyradio 2.0 al USB, se instalarán los drivers necesarios. Solo es necesario proporcionar permisos *udev* [27].

2.2.5. Configurar el cfclient

La aplicación *cfclient* (CrazyFlie client) es una herramienta basada en Python y proporcionada dentro de la plataforma de Crazyflie 2.x. Tiene una interfaz gráfica que permite al usuario *flashear* el firmware del Crazyflie, controlar el vuelo y cambiar parámetros.

La aplicación *cfclient* se ha instalado para los dos sistemas operativos (Windows 10 y Linux Mint). En esta sección se resumen los pasos para completar la instalación. Después de hacerla funcionar en los dos sistemas operativos, el rendimiento con Windows 10 era bastante lento y la interacción con la aplicación en Windows 10 no era rápida. Debido a los problemas de rendimiento, se ha decidido seguir todo el proyecto con Linux Mint.

Para hacer la instalación de Linux Mint se recomienda seguir las últimas instrucciones publicadas por el fabricantes [28], aquí se listan los pasos seguidos en este proyecto:

1. Instalar el sistema de control de versiones, gestor de paquetes de Python y la librería para el framework Qt:

   sudo apt install git python3-pip libxcb-xinerama0 libxcb-cursor0

   pip3 install --upgrade pip

2. Instalar el cliente *cfclient*. Para evitar problemas de compatibilidades entre las distintas versiones de librerías usadas en el desarrollo del proyecto, es conveniente instalar el cliente en un entorno virtual de Python:

   $ mkdir cflient

   $ cd cfclient

   ~cfclient$ python3 -m venv env

   ~cfclient$ source env/bin/actívate

   ~cfclient$ pip install cflib

~cfclient$ pip install cfclient

3. Desde un terminal ejecutar el comando siguiente para lanzar el programa cliente

~cfclient$ cfclient

Si la instalación ha sido correcta, se mostrara el menú principal de *cfclient*, como se muestra en la Figura 12.

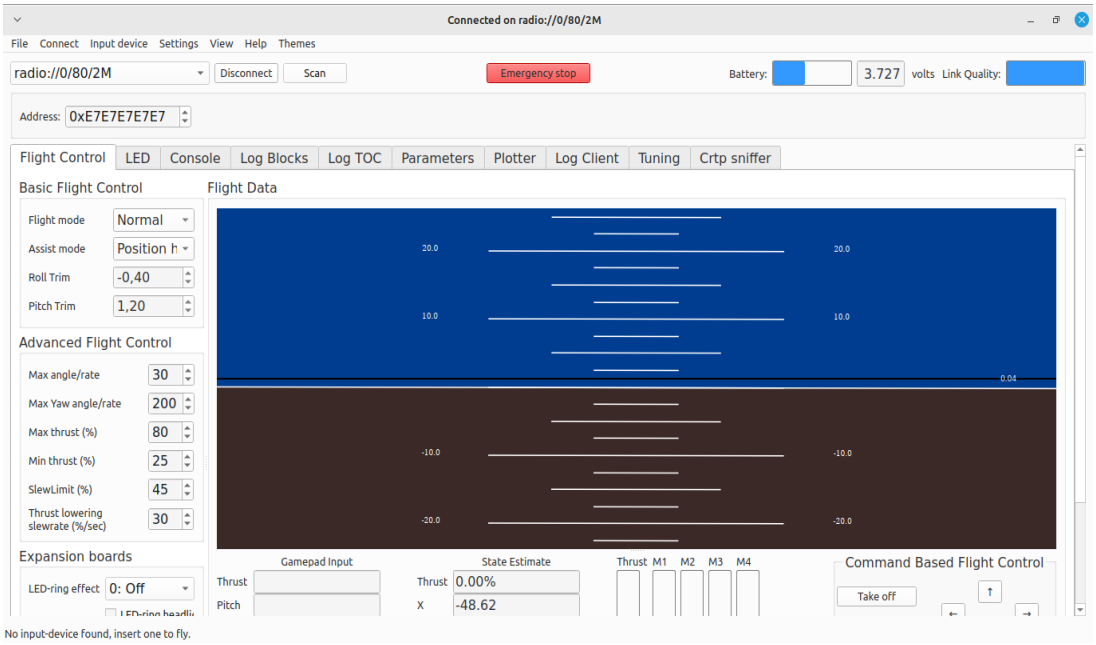


Figura 12. Menú principal de la aplicación *cfclient*.

2.2.6. Configurar dron Crazyflie 2.1

El dron utilizado en el proyecto ya ha sido utilizado anteriormente y, por tanto, no es la primera vez que se pone en marcha. En este caso, solo hemos tenido que actualizar el firmware de los procesadores STM32 y nRF51 del Crazyflie y el procesador ESP de la AI-deck a la última versión, configurar el punto de acceso wifi de la AI-deck y cargar el código para el envío de imágenes desde la AI-deck por el puerto wifi. El código del firmware está en constante desarrollo, por lo que es recomendable instalar siempre las últimas versiones.

NOTA: Si es la primera vez que se pone en marcha la AI-deck, es necesario actualizar a la

última versión del *boatloader* para poder hacer actualizaciones del firmware usando la CrazyRadio. Se recomienda seguir las instrucciones del fabricante [29] y el trabajo de Juan Miguel [17]. El *boatloader* que trae de fábrica no permite cargar código en la AI-deck a través de la Crazyradio y por lo tanto, será necesario utilizar un adaptador (JTAG) para conectarnos por cable directamente a la AI-deck. Este punto no aplica en este proyecto, ya que el dron empleado ya había sido utilizado en otros proyectos previamente.

A continuación se detallan los pasos necesarios para realizar la actualización del firmware de los procesadores STM32 y nRF51 del Crazyflie 2.1; y el ESP de la AI-deck:

- Conectar el adaptador Crazyradio 2.0 al PC.
- Asegurarse de que el único módulo conectado al Crazyflie es el AI-deck.
- Abrir la última versión del cfclient.
- Ir a la opción '*Connect*'->'*bootloader*' (Figura 13)
- Detectar la URI del crazyflie, en este proyecto es radio://0/80/2M.
- Pulsar el botón '*Connect*'.
- Seleccionar el último firmware disponible. La última versión probada es:

  2024.02-firmware-cf2-2024.02.zip

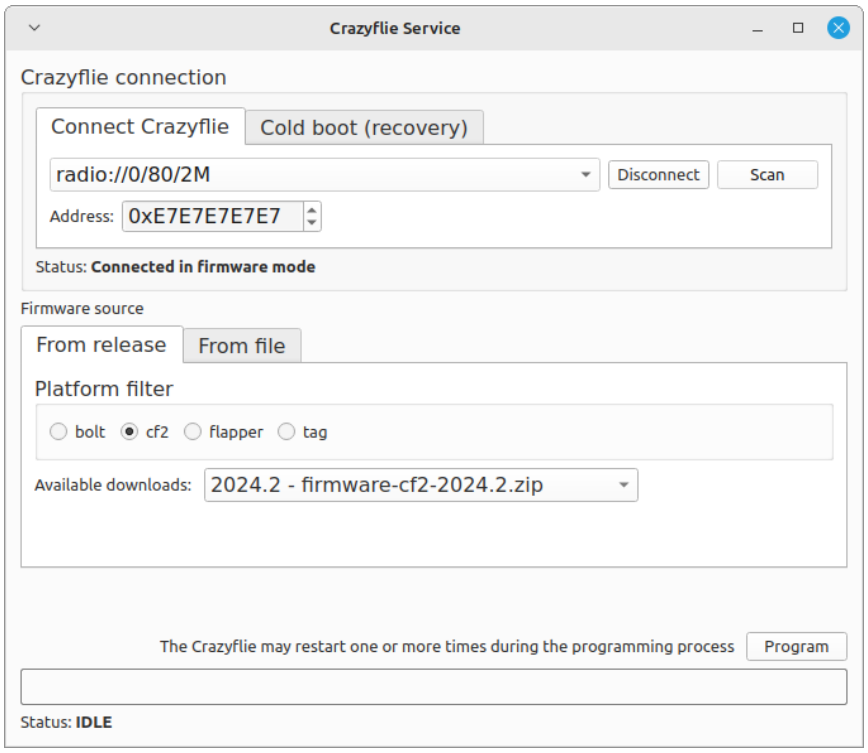


Figura 13. Menú del *cfclient* para actualizar el firmware de Crazyflie 2.x.

- Pulsar el botón '*Program*'.
- Una vez que el estado del proceso es Idle y el Crazyflie está desconectado,
- conectarlo nuevamente para comprobar la salida en el registro de la consola del *cfclient* y que el LED1 parpadea a 2 Hz.

Por otro lado, al actualizar el *firmware* del Crazyflie se pierde cualquier configuración previa del punto de acceso wifi y el código de la AI-deck. Hay varias formas de habilitar el envío de imágenes por wifi (como se explican en [17]), pero en nuestro caso seguiremos el ejemplo *WiFi Video Streamer* proporcionado por Bitcraze [30] y habilitaremos el punto de acceso wifi a través del firmware del Crazyflie.

Los siguientes pasos explican cómo habilitar el WiFI de la AI-deck como punto de acceso:

- Instalar un toolchain.

  $ sudo apt-get install make gcc-arm-none-eabi

- Clonar el repositorio del firmware.

  $ git clone – recursive https://github.com/bitcraze/crazyflie-firmware.git

- Configurar la plataforma que vamos a usar (Crazyflie 2.0).

  $ cd crazyflie-firmware

  ~crazyflie-firmware$ make cf2_defconfig

  ~crazyflie-firmware$ make -j 12

- Cargar el nuevo binario en la memoria flash.

  ~/crazyflie-firmware$ make cload

- Configurar la compilación usando el sistema KBuild.

  $ sudo apt install build-essential libncurses5-dev

  $ ~crazyflie-firmware$ make menuconfig

- Marcar la opción "*Support AI-deck*" del menú "*Expansion deck*".

- Establecer el wifi como punto de acceso.

  Expansion deck configuration → Support the AI dec → WiFi setup at startup (Act as Access Point) → Act as Access Point

  Definimos el nombre del punto de acceso como "AIDECK-AP"

- Establecer unas credenciales para la red wifi que se va a crear.

  Expansion deck configuration → Support the AI deck → Credentials for access-point

  Dejamos el punto de acceso sin contraseña.

- Cargar el firmware.

  $ make

  $ cfloader flash build\cf2.bin stm32-fw -w radio://0/80/2M/E7E7E7E7E7

Actualizar el código de la AIdeck para que envíe imágenes por wifi:

- Clonar el repositorio con los ejemplos proporcionados por Bitcraze.

  $ git clone – recursive https://github.com/bitcraze/aideck-gap8-examples

- Localizar el ejemplo proporcionado por Bitcraze para el envío de imágenes desde la AI-deck (Figura 14).

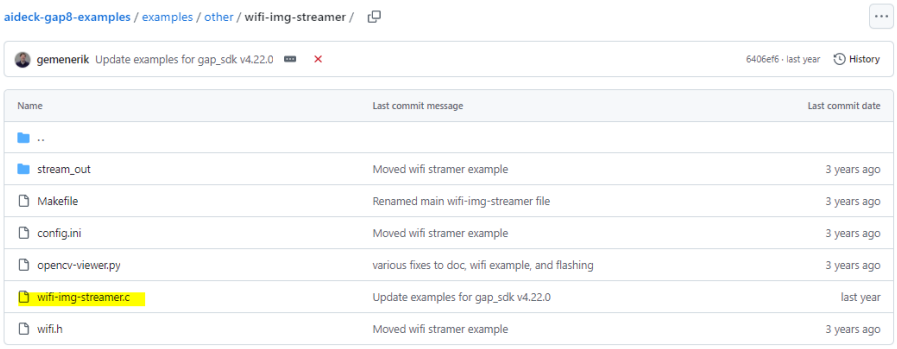


Figura 14. Estructura del contenido del repositorio “aideck-gap8-examples”.

- Modificar el código de la aplicación “*wifi-img-streamer.c”* para cambiar el formato de las imágenes y la resolución (Figura 15).

wifi-img-streamer.c | opencv-viewer.py | nuevo 2

```
33
34  #include "cpx.h"
35  #include "wifi.h"
36
37  #define IMG_ORIENTATION 0x0101
38  #define CAM_WIDTH 324
39  #define CAM_HEIGHT 324  // 244
40
41  static pi_task_t task1;
42  static unsigned char *imgBuff;
43  static struct pi_device camera;
44  static pi_buffer_t buffer;
45
46  static EventGroupHandle_t evGroup;
47  #define CAPTURE_DONE_BIT (1 << 0)
48
49  // Performance menasuring variables
50  static uint32_t start = 0;
51  static uint32_t captureTime = 0;
52  static uint32_t transferTime = 0;
53  static uint32_t encodingTime = 0;
54  // #define OUTPUT_PROFILING_DATA
55
56  static int open_pi_camera_himax(struct pi_device *device)
57  {
58    struct pi_himax_conf cam_conf;
59
60    pi_himax_conf_init(&cam_conf);
61
62    //cam_conf.format = PI_CAMERA_QVGA;
63    cam_conf.format = PI_CAMERA_VGA;
64
65    pi_open_from_conf(device, &cam_conf);
66    if (pi_camera_open(device))
67      return -1;
68
69    // rotate image
70    pi_camera_control(device, PI_CAMERA_CMD_START, 0);
```

Figura 15. Código fuente de *wifi-img-streamer.c.*

- Compilar el código usando *docker*.

  $ cd aideck-gap8-examples

  $ docker run --rm -v ${PWD}:/module aideck-with-autotiler tools/build/make-example examples/other/wifi-img-streamer image

- Actualizar el firmware de la AI-deck con el código "*wifi-img-streamer.c*" modificado

  $ cfloader flash examples/other/wifi-img-streamer/BUILD/GAP8_V2/GCC_RISCV_FREERTOS/target.board.devices.flash.img deck-bcAI:gap8-fw -w radio://0/80/2M

En este punto la AI-deck ya está configurada para emitir imágenes por el puerto de acceso wifi. En el siguiente apartado se explica cómo configurar el ordenador para leer y guardar estas imágenes.

2.2.7. Configurar PC para guardar imágenes del Crazyflie 2.0

En este apartado se explican los pasos necesarios para guardar las imágenes emitidas por la AI-deck en nuestro PC mientras el dron realiza un vuelo.

- Conectarse al nuevo punto de acceso WiFi creado por el Crazyflie: AI-DECK-AP.

- Como en el paso anterior hemos cambiado la resolución de las imágenes enviadas por la AI-deck, tenemos que cambiar la resolución en el *script* para abrir y guardar las imágenes en nuestro PC (Figura 16).

wifi-img-streamer.c | opencv-viewer.py | nuevo 2

```
#print("Magic is good")
#print("Resolution is {}x{} with depth of {} byte(s)".format(width, height, depth))
#print("Image format is {}".format(format))
#print("Image size is {} bytes".format(size))

# Now we start rx the image, this will be split up in packages of some size
imgStream = bytearray()

while len(imgStream) < size:
    packetInfoRaw = rx_bytes(4)
    [length, dst, src] = struct.unpack('<HBB', packetInfoRaw)
    #print("Chunk size is {} ({:02X}->{:02X})".format(length, src, dst))
    chunk = rx_bytes(length - 2)
    imgStream.extend(chunk)

count = count + 1
meanTimePerImage = (time.time()-start) / count
print("meanTimePerImage {}".format(meanTimePerImage))
print("inverse meanTimePerImage {}".format(1/meanTimePerImage))

if format == 0:
    bayer_img = np.frombuffer(imgStream, dtype=np.uint8)
    bayer_img.shape = (324, 324)
    color_img = cv2.cvtColor(bayer_img, cv2.COLOR_BayerBG2BGRA)
    cv2.imshow('Raw', bayer_img)
    cv2.imshow('Color', color_img)
    if args.save:
        cv2.imwrite(f"stream_out/raw/img_{count:06d}.png", bayer_img)
        cv2.imwrite(f"stream_out/debayer/img_{count:06d}.png", color_img)
    cv2.waitKey(1)
else:
    with open("img.jpeg", "wb") as f:
        f.write(imgStream)
    nparr = np.frombuffer(imgStream, np.uint8)
    decoded = cv2.imdecode(nparr,cv2.IMREAD_UNCHANGED)
    cv2.imshow('JPEG', decoded)
    cv2.waitKey(1)
```

Figura 16. Código fuente *opencv-viewer.py*.

- Instalar el visor opencv-python.

  $ pip install opencv-python

- Iniciar el visor, guardando localmente las imágenes:.

$ cd aideck-gap8-examples/examples/other/wifi-img-streamer

$ python3 opencv-viewer.py -- save

De esta forma, se empiezan a guardar imágenes cada ~0.40 segundos en el PC (Figura 17).

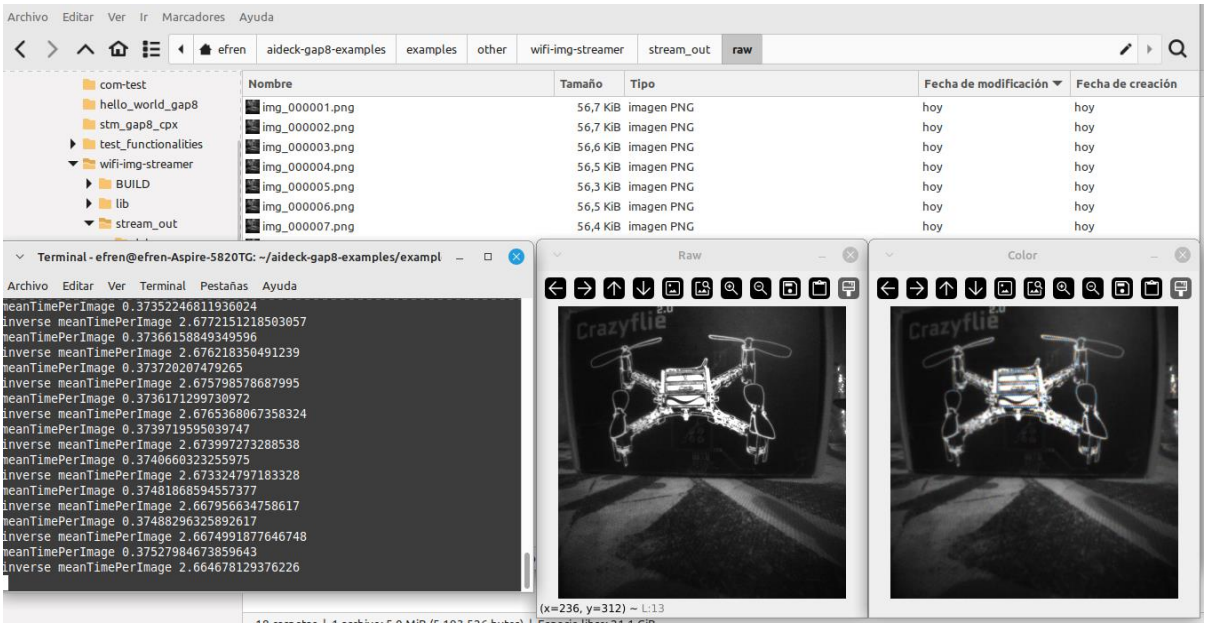


Figura 17 Ejemplo de captura de imágenes con Crazyflie 2.0 desde PC

Posteriormente, estas imágenes serán usadas en los algoritmos de VSLAM. La Figura 18 muestra un ejemplo de imagen emitida por la AI-deck y guardada en el PC localmente. Puede observarse como la resolución de la imagen recibida es de 324x324 píxeles.

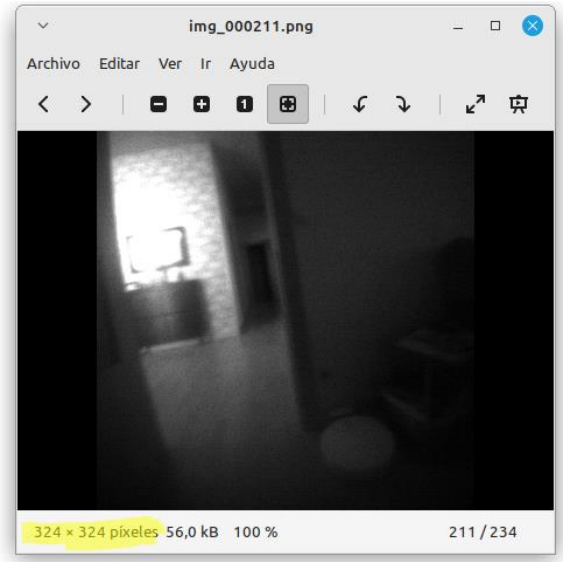


Figura 18. Ejemplo imagen tomada por la cámara del Crazyflie 2.0.

2.2.8. Control de vuelo del dron

El dron Crazyflie 2.0 tiene varias formas de ser controlado (o volado). En este apartado se resumen los métodos principales.

- Desde el PC usando el software **Crazyflie cfclient**:
  - Conectar Crazyradio 2.0 al USB del PC.
  - Abrir la aplicación cfclient.
  - Se puede controlar el dron directamente del teclado o conectar un gamepad (*joystick* tipo Xbox o PlayStation).

- Desde un móvil Android:
  - Instalar la aplicación Crazyflie Client en el móvil Android.
  - https://play.google.com/store/apps/details?id=se.bitcraze.crazyfliecontrol2&hl=es_419&pli=1.
  - Conectarse a través de la aplicación móvil al *bluetooth* del dron.
  - Utilizar los controles táctiles del móvil para realizar los vuelos.

- Desde el PC usando scripts de programación de la trayectoria en Python:
  - Conectar Crazyradio 2.0 al USB del PC
  - Instalar la librería club en nuestro PC
  - Más detalles se pueden encontrar en la web del fabricante [31]

Con cualquiera de estas formas se consigue recibir imágenes en el PC mientras el dron realiza un vuelo, ya sea planificado o a tiempo real controlado directamente desde el PC o el móvil.

2.3. Calibrar cámara

En este proyecto, para calibrar la cámara, se ha usado la propia librería de matlab "Camera Calibration" [32] de la toolbox "Computer Vision". También es posible calibrar la cámara desde Python utilizando la librería de OpenCV [33]

La calibración de la cámara es el proceso de estimar los parámetros de la cámara utilizando imágenes que contienen un patrón de calibración. Los parámetros incluyen las características intrínsecas de la cámara, los coeficientes de distorsión y las características extrínsecas de la cámara. Se utilizan estos parámetros de la cámara para eliminar los efectos de distorsión de la lente de una imagen, medir objetos planos, reconstruir escenas en 3D a partir de múltiples cámaras y realizar otras aplicaciones de visión por ordenador.

En este proyecto solo se ha tenido en cuenta los parámetros intrínsecos; los coeficientes de distorsión se han considerado nulos. Por otro lado las características extrínsecas de la calibración solo tienen sentido para cámaras estéreo.

El patrón utilizado para la calibración es un patrón de tablero de ajedrez (Figura 19). Este patrón consiste en cuadrados blancos y negros de tamaño igual alternados. Las esquinas de los cuadros que se encuentran dentro del patrón se utilizan como puntos de control. Estas esquinas pueden ser detectadas automáticamente en la imagen de calibración 2-D mediante un algoritmo de detección de esquinas. Al asumir que el punto de la esquina inferior derecha del cuadro superior izquierdo del tablero de ajedrez es el origen, también podemos determinar sus coordenadas 3-D en el mundo utilizando el tamaño de los cuadros del tablero de ajedrez.

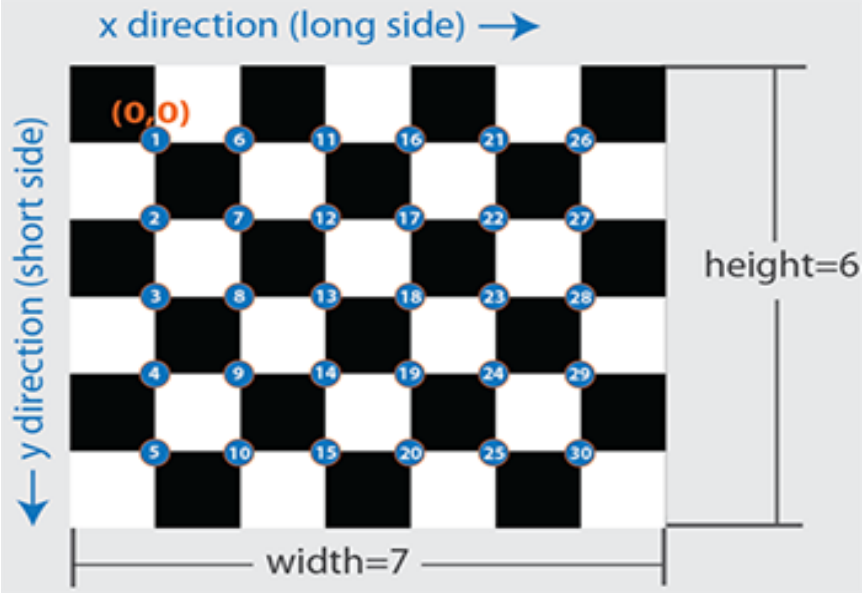


Figura 19. Ejemplo patrón "tablero de ajedrez" para la calibración dela cámara. Fuente [34].

En este proyecto, el patrón utilizado es de dimensiones 10x7 con una tamaño de 28mm de lado de cada cuadrado. Una vez realizada la calibración, se obtienen la matriz de calibración, que tendrá la forma siguiente:

$$K = \begin{pmatrix} fx & s & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{pmatrix}$$

Dónde:

fx, fy distancia focal

cx, cy : centro óptico, normalmente cercano al centro de la imagen

s: *skew*, representa el ángulo entre los ejes X e Y en el sensor, en nuestro caso asumimos que es cero.

2.4. Técnicas aplicadas en el VSLAM Monocular

Como se ha comentado en las secciones anteriores, las técnicas VSLAM están en constante evolución. Dependiendo de los recursos computacionales y la necesidad de trabajar en tiempo real; las técnicas utilizadas en el algoritmo pueden cambiar. En esta sección se muestra un resumen con las principales técnicas aplicadas en este proyecto.

2.4.1. Extracción de puntos clave

Para la mayoría de los sistemas de Visual SLAM (VSLAM) se necesitan dos componentes:

1. **Detector de puntos clave** – decide *dónde* mirar.
2. **Descriptor** – codifica *cómo* es cada punto para poder compararlo entre imágenes.

La Tabla 2 muestra un resumen de las combinaciones más utilizadas para VSLAM.

Detector	Descriptor	Invariante a la rotación	Escalado automático	Precisión	Relocalización	Eficiencia
Harris	Patch	No	No	Alta	--	Alta
Shi-Tomasi	Patch	No	No	Alta	--	Alta
SIFT	SIFT	Si	Si	Baja	Alta	Baja
SURF	SURF	Si	Si	Baja	Alta	Baja
FAST	BRIEF	No	No	Media	Media	Alta
ORB	ORB	Si	No	Media	Media	Alta

Tabla 2 Comparación entre diferentes métodos de detectores de puntos clave y descriptores. Fuente [6]

En nuestro proyecto se ha utilizado ORB (Oriented FAST and Rotated BRIEF) [35], que es un método combinado que incluye tanto el detector como el descriptor. Como detector usa FAST para encontrar esquinas y añade una estimación de orientación (mediante momentos de

intensidad) para que los puntos sean invariantes a la rotación. El cálculo del descriptor proporciona una cadena binaria de 256 bits, usa una versión rotada de BRIEF (Binary Robust Independent Elementary Features) adaptada según la orientación del punto clave. De esta manera, logra que el descriptor también sea invariante a la rotación. Son extremadamente rápidos de calcular y emparejar, y presentan una buena invariancia respecto al punto de vista.

En todo el flujo de trabajo se usarán los mismos puntos clave y descriptores, tanto a la hora de buscar emparejamientos como en el momento de cerrar el lazo.

La Figura 20 muestra en la izquierda una imagen original tomada por el dron Crazyflie 2.0 y la derecha, esta misma imagen superpuesta con los puntos ORB detectados.

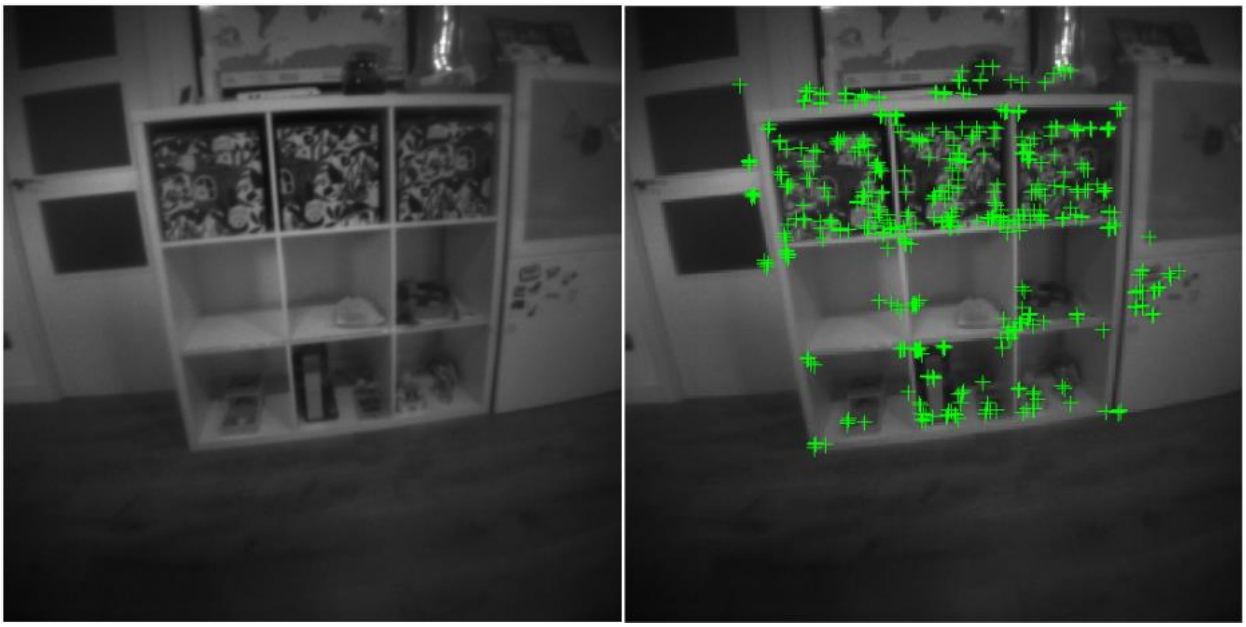


Figura 20. Ejemplos de puntos OBR sobre imágenes tomadas por el Crazyflie 2.0.

En Matlab podemos usar la siguiente función para extraer puntos característicos y descriptores: *extractfeatures* [36].

2.4.2. Emparejamiento de puntos clave

El emparejamiento en el proceso de VSLAM significa encontrar correspondencias entre puntos clave (*keypoints*) detectados en distintas imágenes tomadas por la cámara (o cámaras) a lo largo del tiempo. Hay distintos métodos para hacer este emparejamiento, pero en este proyecto se han emparejado por distancia.

Se comparan los descriptores de dos imágenes con una métrica de distancia y se emparejan los más cercanos.

Existen diferentes formas de calcular la distancia entre descriptores, como la suma de diferencias absolutas (*Sum of Absolute Difference*s, SAD), la suma de diferencias al cuadrado (*Sum of squared differences*, SSD) o la distancia de Hamming. En el caso de descriptores binarios, como los generados por el algoritmo ORB, la distancia de Hamming es mucho más eficiente computacionalmente que las distancias euclidianas tradicionales. Por este motivo, en este proyecto se ha utilizado la distancia de Hamming para comparar descriptores, ya que estos están representados como vectores de bits. La distancia de *Hamming* se implementa con una simple operación *XOR* y conteo de bits en 1. A continuación, se muestra un ejemplo:

Descriptor A: 11010110

Descriptor B: 10011110

$1 \neq 1 \rightarrow 0$

$1 \neq 0 \rightarrow 1$

$0 \neq 0 \rightarrow 0$

$1 \neq 1 \rightarrow 0$

$0 \neq 1 \rightarrow 1$

$1 \neq 1 \rightarrow 0$

$1 \neq 1 \rightarrow 0$

$0 \neq 0 \rightarrow 0$

Distancia de Hamming = 2

La Figura 21 muestra un ejemplo de imágenes tras aplicar emparejamiento:

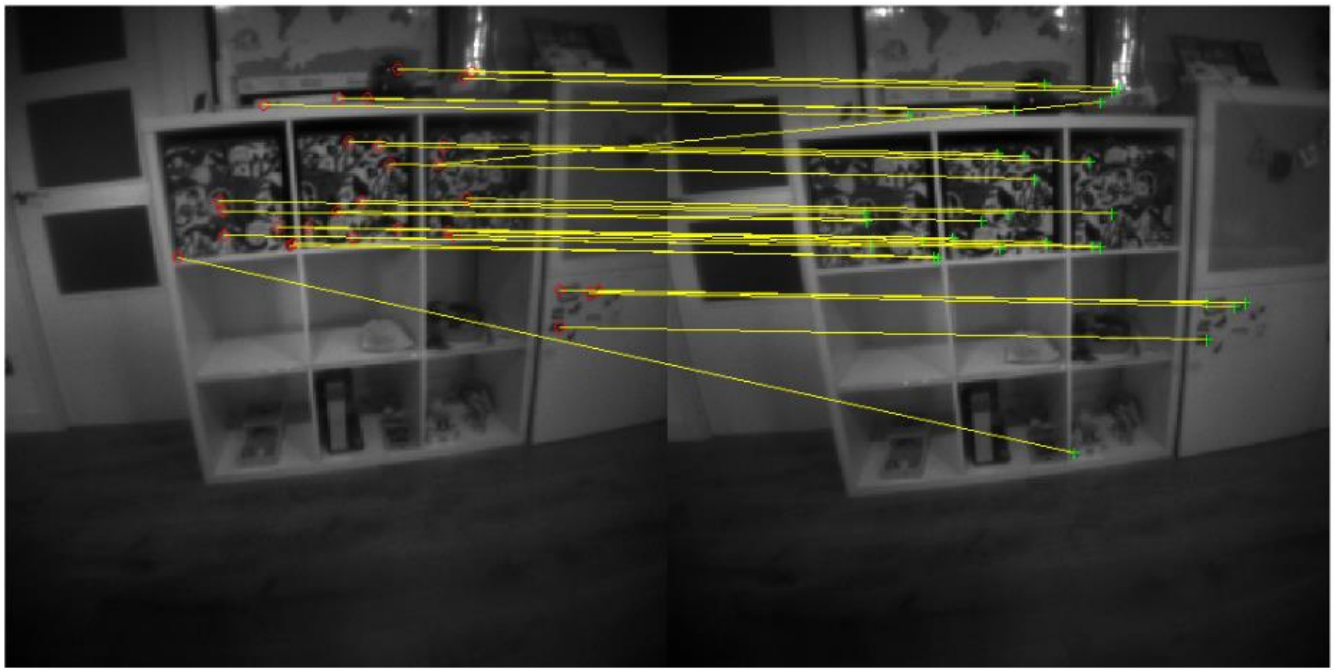


Figura 21. Ejemplo emparejamiento entre dos imágenes.

Se puede observar como algunos de los emparejamientos son incorrectos, pero es normal debido a que algunos puntos pueden ser erróneos por oclusiones, ruido, repetición. Para mejorar el emparejamiento y evitar estos puntos erróneos se tienen que aplicar otros filtrados posteriores, en nuestro caso aplicaremos RANSAC para obtener un subconjunto coherente que maximice la consistencia geométrica. En Matlab podemos usar la siguiente función para buscar los emparejamientos *matchfeatures* [37].

2.4.3. Estimación de la posición y orientación de la cámara

La posición y orientación de la cámara nos sirve para saber dónde está la cámara y hacia dónde apunta en cada instante. La posición es la ubicación del centro óptico de la cámara en el mundo y se expresa como un vector 3D:

$$t = (x, y, z)$$

La orientación nos dice cómo está rotada la cámara respecto al sistema de referencia del mundo. Se puede expresar como una matriz de rotación, con cuaterniones, o con ángulos de euler. En las aplicaciones VSLAM, lo normal es usar matrices de rotación o cuaterniones. En este proyecto se han usado matrices de rotación

$$R \in \mathbb{R}^{3x3} \qquad R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$$

Cuando tenemos reunida la información de la posición y la orientación de la cámara, hablamos de la pose de la cámara. La pose es la transformación rígida (T) de los puntos de la cámara a los puntos de la referencia absoluta del mundo y viceversa. Permite transformar un punto 3D de la referencia del mundo a coordenadas de la cámara (o al revés).

$$T \in \mathbb{R}^{4x4} \qquad T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$$

Para estimar la pose de la cámara, se pueden emplear principalmente dos enfoques, dependiendo de la información disponible y de la fase de ejecución del algoritmo de VSLAM. Si se encuentra en una fase de inicialización del algoritmo donde no se dispone de un mapa previo, se emplean dos imágenes 2D para triangular entre los emparejamientos de las imágenes. Por otro lado, si ya se dispone de información del entorno, por ejemplo, en una fase de localización, se llevará a cabo una estimación entre puntos 3D conocidos y puntos 2D de la imagen a localizar. El proceso para cada técnica a continuación.

Cálculo de la pose usando dos imágenes 2D:

- Se detectan y emparejan puntos clave entre dos imágenes.
- Se estima la matriz esencial (E) usando correspondencias 2D-2D y aplicando filtrado por RANSAC.
- A partir de la matriz esencial y la calibración de la cámara, se recupera la pose relativa (R, t) entre las dos vistas.
- El problema en este punto es que no hay una solución única, hay cuatro posibles soluciones. De las cuatro soluciones posibles, se elige la correcta triangulando puntos y verificando cuál de las soluciones cumple que los puntos están enfrente de ambas cámaras.

Cálculo de la pose usando correspondencia entre 3D y 2D:

- Se parte de un conjunto de puntos 3D ya conocidos del mapa.
- Se detectan puntos clave en una nueva imagen y se asocian a los puntos 3D.
- Se resuelve el problema de Perspective-n-Point (PnP) [38], filtrando con RANSAC para ser robusto a errores de correspondencia.

En Matlab podemos usar las siguientes funciones:

- *estimateEssentialMatrix* [39] para calcular la matriz esencial.
- *estrelpose* [40] para calcular la pose a partir de la matriz esencial.
- *estworldpose* [41] para calcular la pose entre información 3D y 2D.

2.4.4. *Bundle adjustment*

Bundle Adjustment (BA) [42] es la técnica utilizada para optimizar las posiciones de un conjunto de puntos P y las poses de un conjunto de cámaras C, minimizando el error de reproyección; es la optimización principal realizada en el SLAM visual moderno basado en características.

$$\underset{\{X_j, T_i\}}{\arg min} = \sum_{i=1}^{n_{imagenes}} \sum_{j=1}^{n_{puntos}} \left\| x_{ij} - \pi(T_i, X_j) \right\|^2$$

Dónde:

x_{ij}: Punto 2D observado del punto 3D j en la imagen i

$\pi(T_i, X_j)$: Proyección del punto 3D X_j usando la pose T_i

T_i: pose de la cámara

X_j: Posición del punto 3D

Para conseguir reducir el error de reproyección se utiliza un algoritmo de minimización no lineal robusto que modifica los valores estimados de los parámetros conjuntamente. En Matlab podemos usar la función *bundleAdjustment* [43]

2.4.5. Técnicas para cerrar el lazo

Las técnicas de cierre de lazo (*loop closure*) se usan para detectar si la cámara ha regresado a un lugar previamente visitado. Esto es fundamental para corregir el error acumulado (*drift*) en la estimación de la trayectoria y del mapa. Durante el movimiento, errores pequeños en la estimación de la pose se acumulan. Si no se corrigen, el mapa y la trayectoria se vuelven inconsistentes. El cierre de lazo detecta estas oportunidades para reidentificar lugares conocidos

y reajustar todas las poses de cámara mediante optimización global (*bundle adjustment*).

Para aplicaciones en tiempo real, como en ORB-SLAM, es muy recomendable aplicar técnicas como Bag of Wrods (BoW) [44]. Esta técnica es muy empleada para aplicaciones en tiempo real por su bajo coste computacional. Con esta técnica se entrena un diccionario visual (vocabulario) a partir de muchos descriptores (como ORB). Después, cada descriptor de una imagen se asigna a la palabra más cercana del diccionario. Las imágenes se representan como un vector de frecuencias de palabras (BoW vector) y, finalmente, se comparan estos vectores. Es un método muy eficiente para comparar muchas imágenes.

Otro método que se puede aplicar es la comparación directa entre descriptores. Es un método más costoso computacionalmente, pero bastante robusto. Este es el método que se ha usado en este proyecto. Para minimizar el coste computacional, el emparejamiento directo para cerrar el lazo no se aplica de forma continua, sino que se aplica cada cierto número de imágenes y solo se tienen en cuenta si la detección se produce entre imágenes no consecutivas.

2.5. Flujo de trabajo VSLAM Monocular

Como se ha mencionado en el capítulo 1, las técnicas de VSLAM están en constante evolución y mejora. La elección de una u otra técnica depende en gran medida de la aplicación específica y de los recursos de cálculo disponibles. En este proyecto, se ha tomado como referencia la solución propuesta por ORB-SLAM [10]. En este proyecto se ha optado por realizar la implementación completa en Matlab, debido a su facilidad para el prototipado y la visualización intermedia de resultados. Se han tenido en cuenta las referencias proporcionadas por Matlab [45] para implementar las técnicas VSLAM monoculares.

La primera parte de la implementación es la definición de todos los parámetros y entradas necesarias para la ejecución del algoritmo. En esta parte es donde además se define si se va a trabajar en modo “creación mapa” o “localización”.

En el modo de creación de mapa, se utilizaran imágenes previamente generadas para crear un mapa 3D del entorno. Esta sección a su vez se subdivide en las siguientes partes:

1. Inicialización del modelo
2. Seguimiento (*Tracking*)
3. Actualizar mapa
4. Cierre de lazo

Finalmente, tanto si estamos en modo creación de mapa o localización, se mostrarán los resultados obtenidos. La Figura 22 muestra la secuencia de las diferentes partes en las que está dividida la implementación.

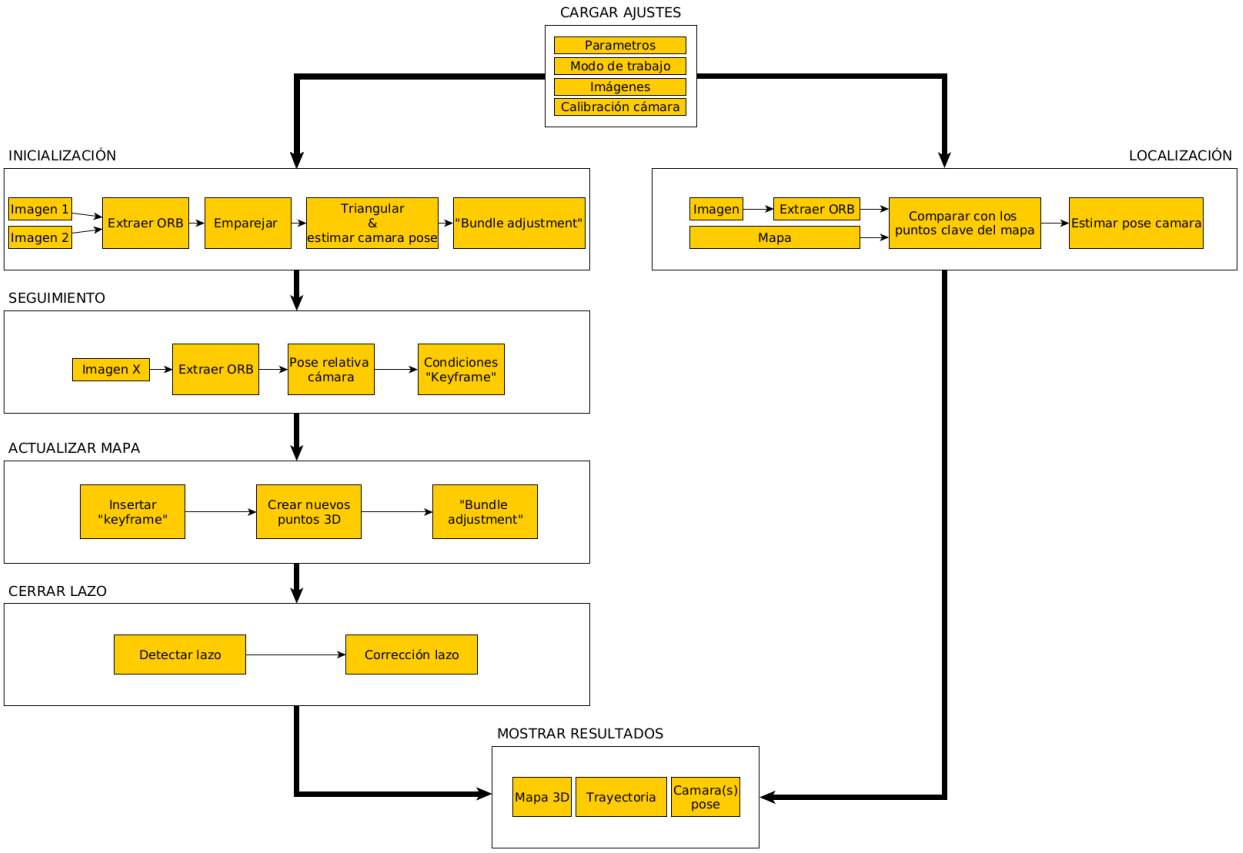


Figura 22. Secuencia de la implementación de VSLAM.

2.5.1. Cargar ajustes

En este bloque se definen los ajustes que la herramienta necesita para poder ejecutarse. Los principales puntos que se tienen que definir son los siguientes:

- Directorio con las imágenes.
- Calibración de la cámara.
- Número de puntos ORB que se van a detectar en cada imagen.
- Criterios para los puntos ORB (factor de escala y número de niveles de descomposición).
- Si se quiere ejecutar el modo creación de mapa o solo en localización.

Además, en esta sección también se hacen algunas comprobaciones básicas, como que la

calibración sea compatible con las imágenes que se van a usar

2.5.2. Inicializar modelo

El objetivo de la inicialización del mapa es calcular la pose relativa entre dos fotogramas para triangular un conjunto inicial de puntos del mapa. Los pasos dentro de la inicialización son los siguientes:

- Cargar las dos primeras imágenes.
- Corregir las imágenes teniendo en cuenta los parámetros intrínsecos de la cámara.
- Extraer los puntos característicos ORB en ambas imágenes y buscar los emparejamientos.
- Si hay suficientes emparejamientos seguir los siguientes pasos; si no, volver a seleccionar una nueva imagen.
- Calcular la matriz esencial aplicando RANSAC para obtener la pose de la cámara.
- Finalmente, se aplica *bundle adjustment* para refinar la reconstrucción inicial.

2.5.3. Seguimiento (*Tracking*)

En este paso se leen todas las imágenes disponibles en bucle y se aplica un proceso muy similar al que se ha aplicado en la inicialización, aunque con los siguientes matices:

- Cargar la siguiente imagen.

- Corregir la imagen teniendo en cuenta los parámetros intrínsecos de la cámara.

- Buscar los emparejamientos con la imagen anterior.

- De los emparejamientos encontrados, utilizar la posición en el mapa de los puntos para estimar la posición y orientación de la cámara en la nueva imagen.

- Decidir si se considera un *keyframe*. En esta implementación el criterio para considerar

como *keyframe* es muy conservador y solo se pide que tenga como mínimo 50 puntos de seguimiento emparejados.

2.5.4. Actualizar modelo

Si en el paso anterior se ha definido que la imagen actual cumple las condiciones para ser un *keyframe*, en este paso se usará esta imagen para añadir información al mapa 3D. Se añadirá la información relativa a la posición y orientación de la cámara de la imagen actual y, posteriormente, se obtendrán nuevos puntos por triangulación para ser añadidos en nuestro modelo. Finalmente, también se aplicará *bundle adjustment* para reducir el error de reproyección de los puntos añadidos.

2.5.5. Cierre de lazo (*loop closure*)

En el cálculo iterativo de la posición relativa de la cámara, se añade un *drift,* que se acumula en cada iteración. Por lo tanto, es necesario este bloque para corregir este *drift*. Este bloque utiliza la imagen actual e intenta detectar y cerrar el lazo. Para detectar si ha habido un lazo cerrado, se compara la información de la imagen actual con la información almacenada de las imágenes anteriores. Se considera que hay un lazo cerrado si se encuentra que la imagen actual tiene un número de puntos de interés similares a alguna imagen anterior que no es consecutiva. Una vez confirmado que se ha detectado un lazo cerrado, se añade esta nueva conexión en el mapa 3D y se optimiza el conjunto de todos los puntos 3D y poses de las cámaras

2.5.6. Localización

La localización solo se puede ejecutar cuando ya existe un modelo VSLAM previamente calculado. En este proceso, primero se carga el mapa guardado, incluyendo los puntos 3D y sus descriptores. Luego, para cada nueva imagen de entrada, corrige la imagen teniendo en cuenta

los parámetros intrínsecos de la cámara, se detectan y extraen características con ORB, y se emparejan con las del mapa de referencia. Finalmente, se estima la orientación y posición de la cámara usando las correspondencias entre las características de la imagen y los puntos 3D del mapa.

2.5.7. Mostrar resultados

En esta sección, se muestran los resultados obtenidos por el algoritmo VSLAM. En el modo "creación del mapa", se mostrará el mapa 3D final, la trayectoria del dispositivo y las posiciones/orientaciones de la cámara en cada instante. Por otro lado, en el modo "localización", se mostrará el mapa 3D junto con la posición de nuestro dispositivo de las imágenes que queremos localizar. También se muestran otros datos de interés, como pueden ser la evolución del número de *keyframes* creados, el número de puntos ORB obtenidos en cada imagen, el número de emparejamientos o el número de puntos emparejados después de aplicar RANSAC. Estos datos son interesantes para poder analizarlos en caso de que el algoritmo pierda el seguimiento ya sea por falta de emparejamientos o por problemas a la hora de extraer puntos característicos de las imágenes utilizadas.

Capítulo 3. Resultados y discusión

En esta sección, se exponen los resultados alcanzados en este proyecto. Hay una primera sección donde se presentan los resultados de la calibración de la cámara utilizada en el dron. Las siguientes secciones muestran los resultados al aplicar las técnicas de VSLAM a diferentes conjuntos de imágenes grabados con el dron.

3.1. Calibración de la cámara

Para calibrar la cámara se ha utilizado la librería de Matlab "Camera Calibration" [34] de la *toolbox* "Computer Vision". La Figura 23 muestra en la izquierda un ejemplo de una de las imágenes originales tomada por el dron y a la derecha la misma imagen postprocesada por la aplicación de "Camera Calibration". Se han utilizado un total de 28 imágenes

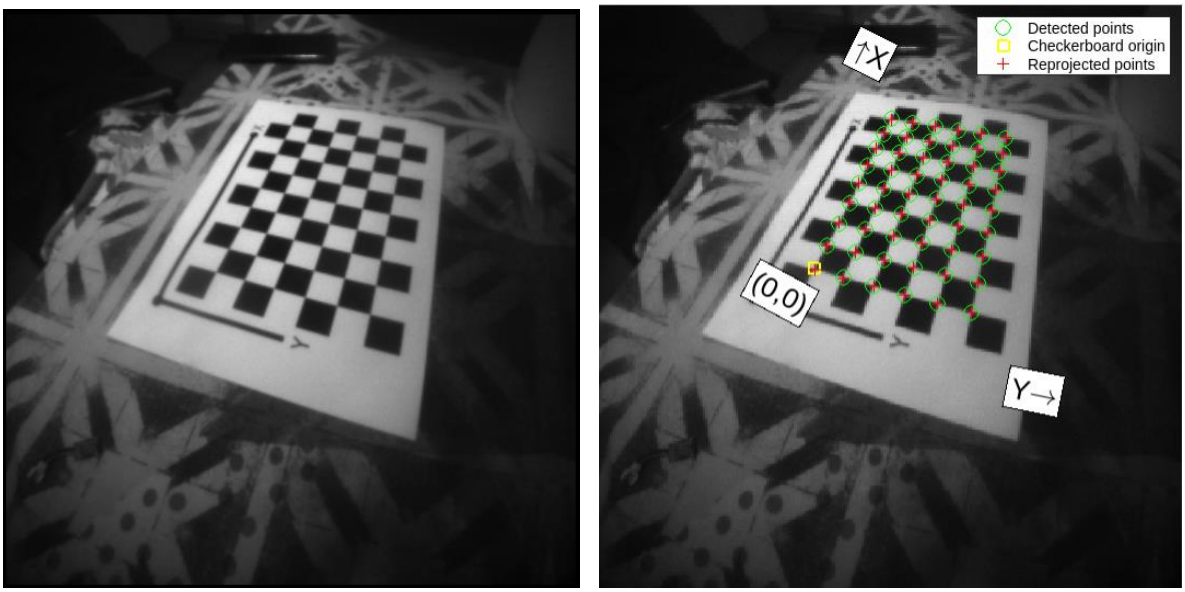


Figura 23. Ejemplo imágenes usadas en la calibracion y procesado en la toolbox de Matlab.

Una vez acabo el postprocesado las imágenes, se obtienen los siguientes resultados con

la calibración de la cámara:

Longitud focal:
f1 = 194.54
f2 = 195.26

Puntos principales:
p1= 165.34
p2 = 112.50

Matriz intrínseca = $\begin{pmatrix} f1 & 0 & p1 \\ 0 & f2 & p2 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 194.54 & 0 & 165.34 \\ 0 & 195.26 & 112.50 \\ 0 & 0 & 1 \end{pmatrix}$

Estos parámetros serán utilizados en el resto del proyecto.

La Figura 24 muestra la posición de la cámara en cada una de las imágenes respecto al patrón usado para la calibración.

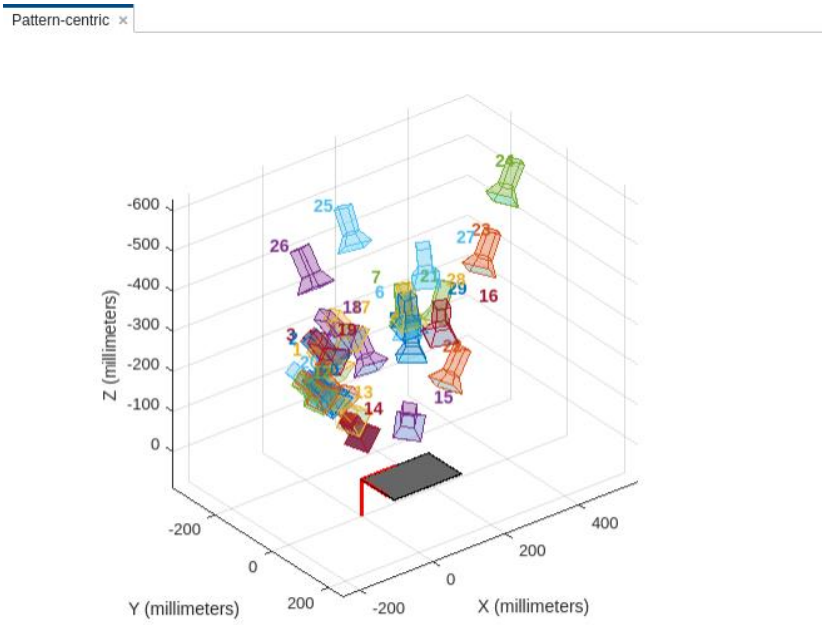


Figura 24. Resutados obtenidos en Matlab al calibrar la camara.

3.2. VSLAM modo creación mapa

En esta sección, se incluyen los resultados obtenidos en la creación del mapa 3D usando imágenes grabadas desde el dron Crazyflie 2.0. El total de imágenes incluidas en este set es de 419 imágenes.

La inicialización se ha realizado utilizando las dos primeras imágenes (Figura 25, Figura 26), en las cuales ya se cumplen los criterios necesarios para garantizar una buena inicialización del modelo 3D.

Figura 25. Imagen inicial con 500 puntos ORB detectados.

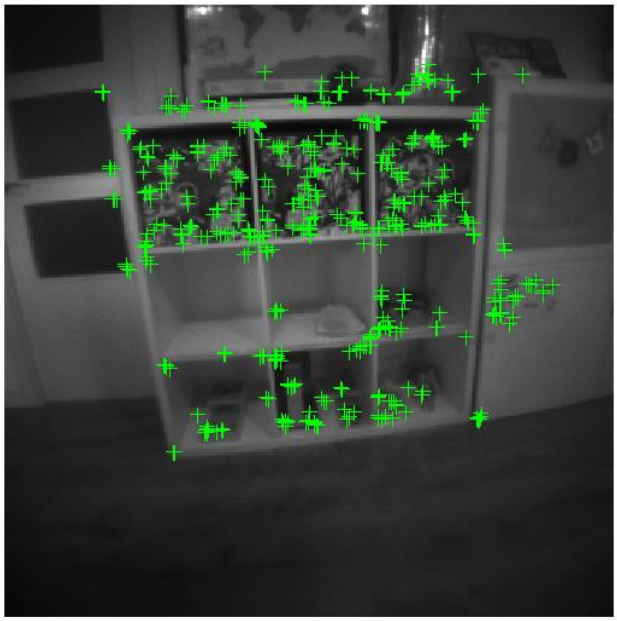


Figura 26 Segunda imagen con los 500 puntos ORB detectados

En las dos imágenes se detectan 500 puntos ORB y al hacer el *match* se obtienen 273 puntos válidos. La Figura 27 muestra el solapamiento de las dos imágenes utilizadas para la inicialización, donde los "+" son los puntos de interés de la imagen 1 y los "O" son los puntos de interés correspondientes de la imagen 2.

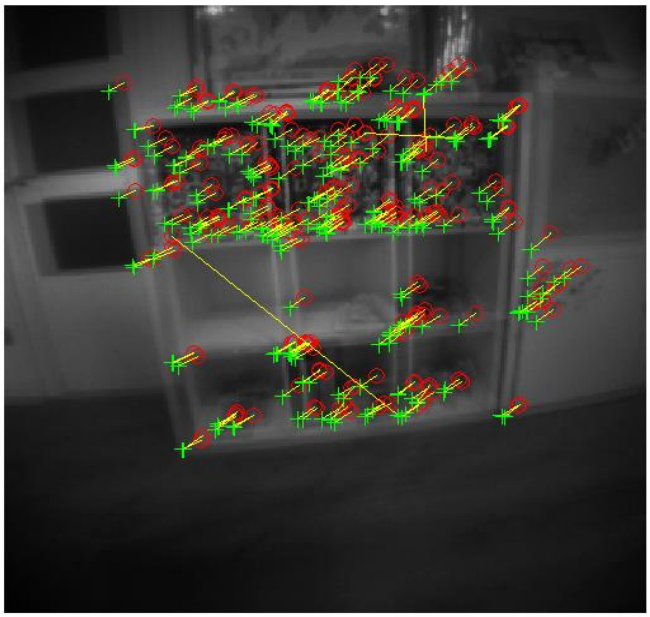


Figura 27. Emparejamiento durante la inicialización con 273 puntos encontrados.

Se puede apreciar como algunos puntos se emparejan incorrectamente entre las imágenes, es necesario aplicar el filtrado por RANSAC con el objetivo de mantener solo los puntos válidos para hacer la triangulación inicial. La Figura 28 muestra los puntos de interés una vez aplicado el filtrado RANSAC.

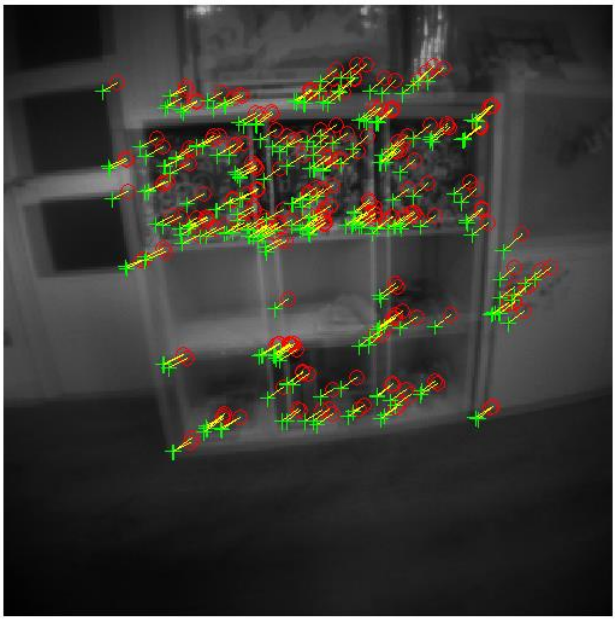


Figura 28. Emparejamiento durante la inicialización aplicando RANSAC, con 254 puntos encontrados.

Una vez aplicado el filtrado RANSAC, el número de puntos se reduce a 254. Con estos puntos ya podemos hacer la triangulación inicial y calcular la posición relativa de la cámara entre las dos imágenes, a su vez permitirá calcular la posición de los puntos de interés en el modelo 3D, formando el primer set de puntos de nuestro mapa 3D. Las Figura 29, Figura 30 y Figura 31 muestran el mapa creado en la fase de inicialización en vista cenital, vista frontal y en 3D, respectivamente.

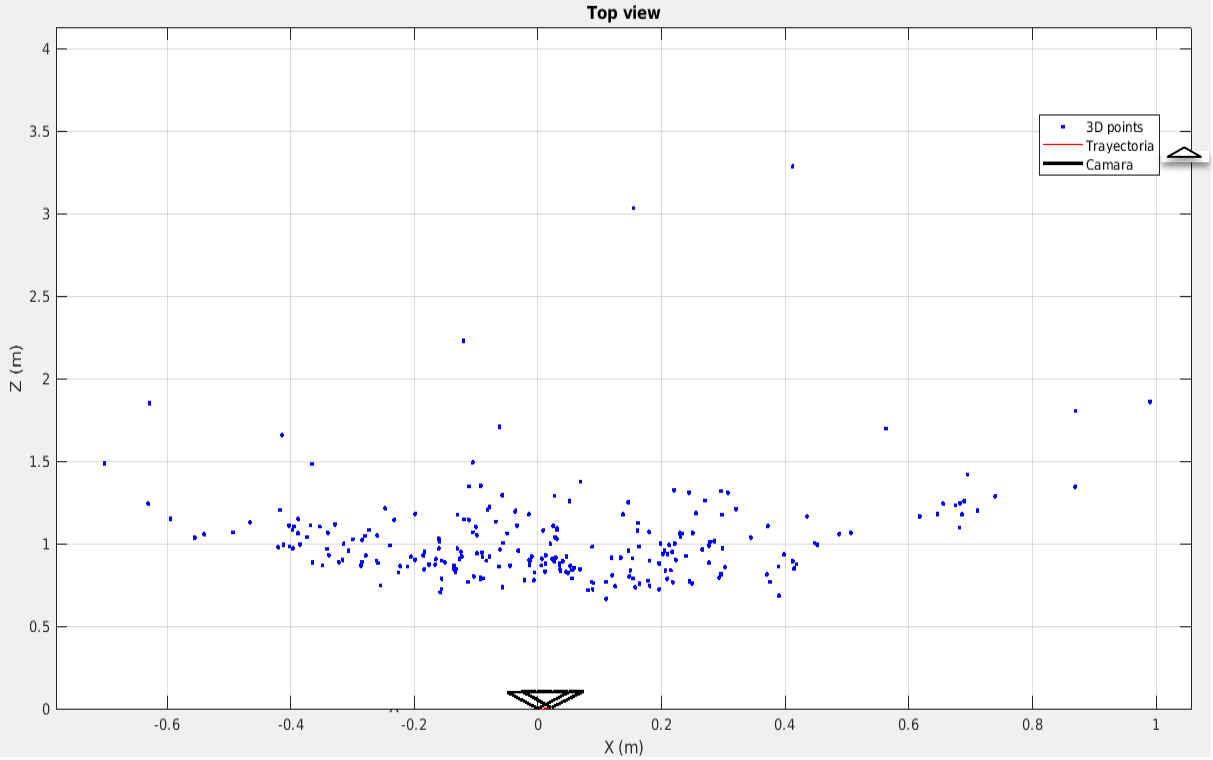


Figura 29. Nube de puntos inicial del mapa 3D, vista cenital.

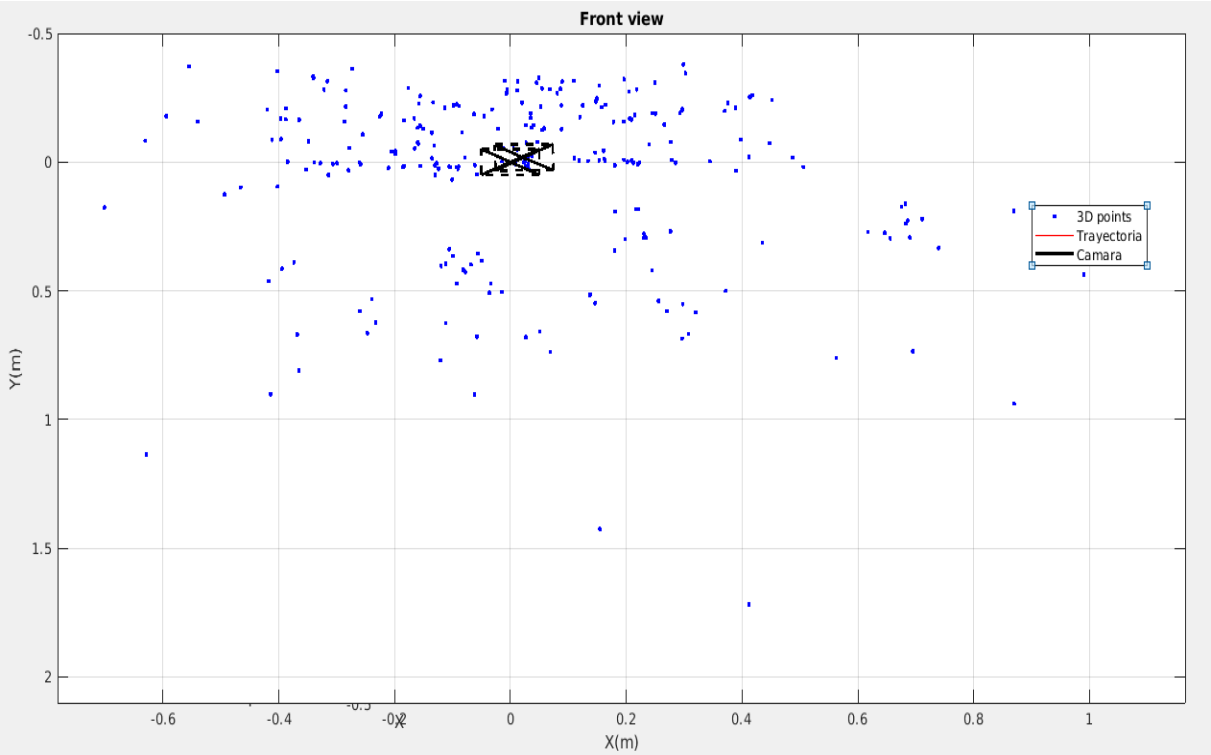


Figura 30. Nube de puntos inicial del mapa 3D, vista frontal.

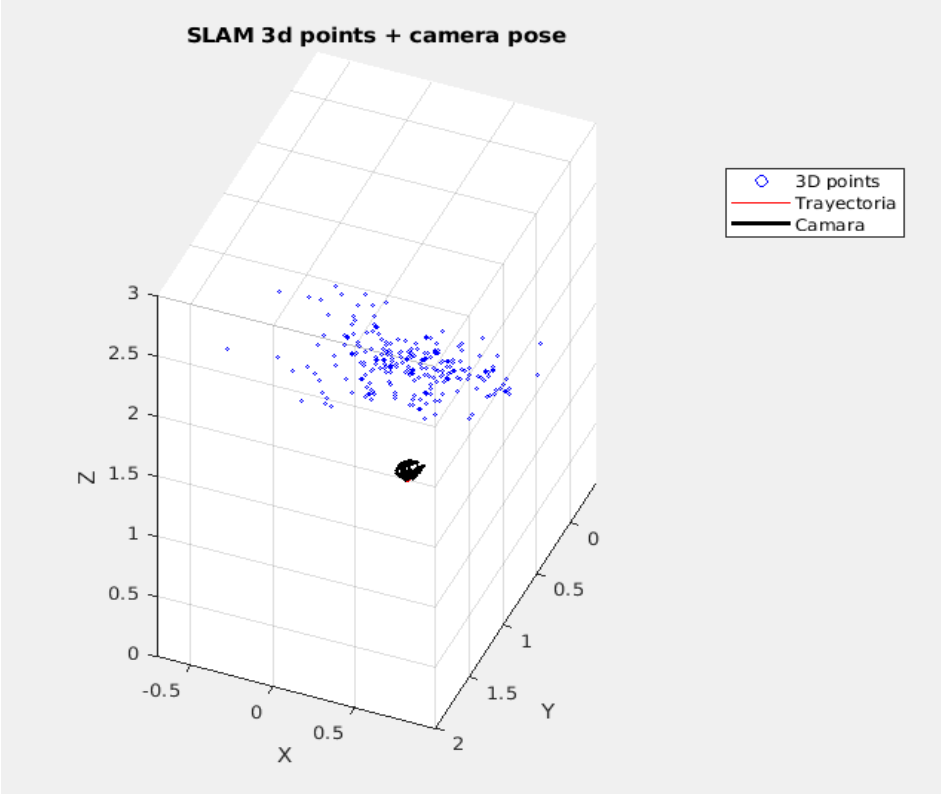


Figura 31. Nube de puntos inicial del mapa 3D.

En este punto ya tenemos el modelo inicializado y podemos seguir con el análisis del resto de imágenes. Una vez completado el análisis de todo el conjunto de imágenes se obtienen los resultados siguientes sin aplicar las técnicas del "cierre del lazo". Las Figura 32, Figura 33 y Figura 34 muestran el mapa actualizado en vista cenital, vista frontal y en 3D, respectivamente.

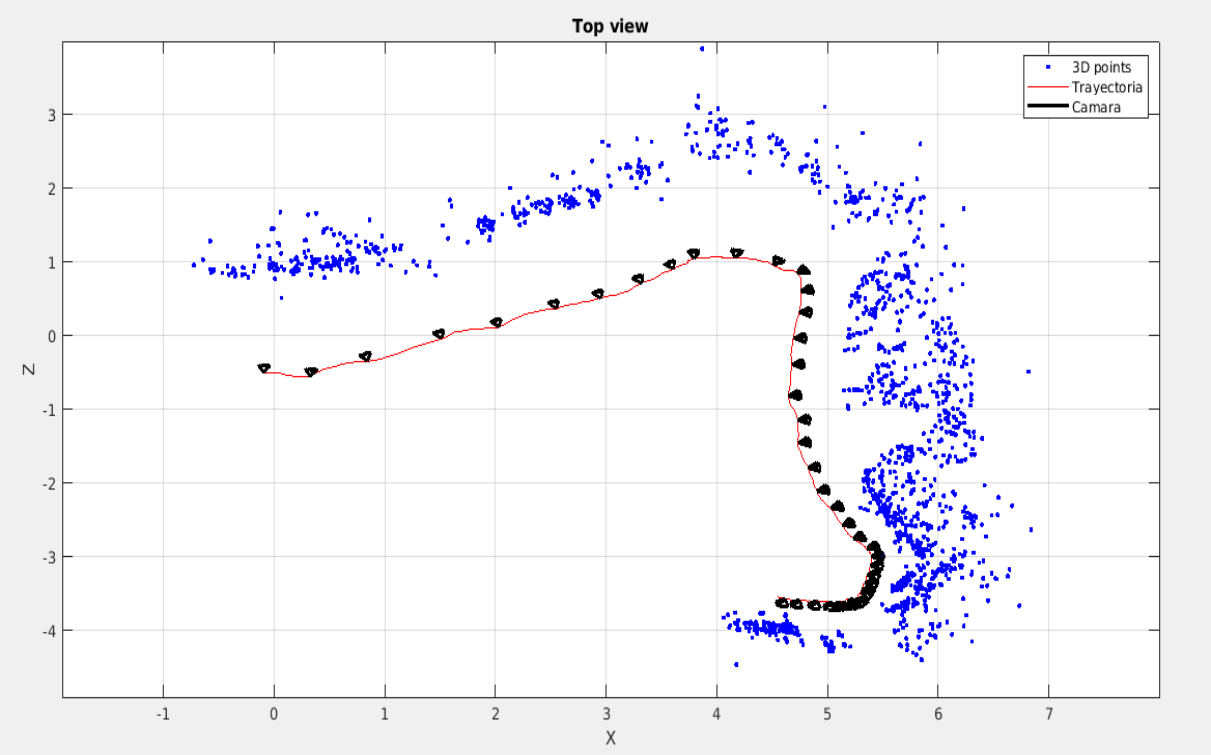


Figura 32. Mapa 3D actualizado, vista cenital.

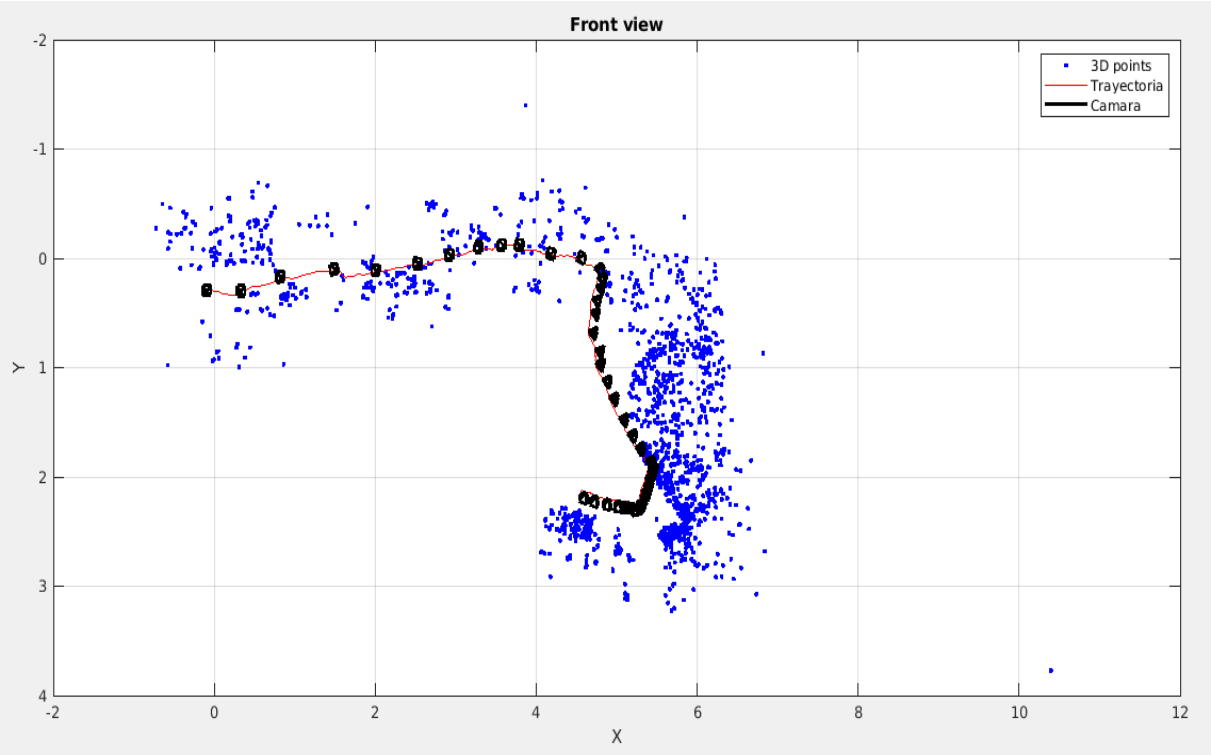


Figura 33. Mapa 3D actualizado, vista frontal.

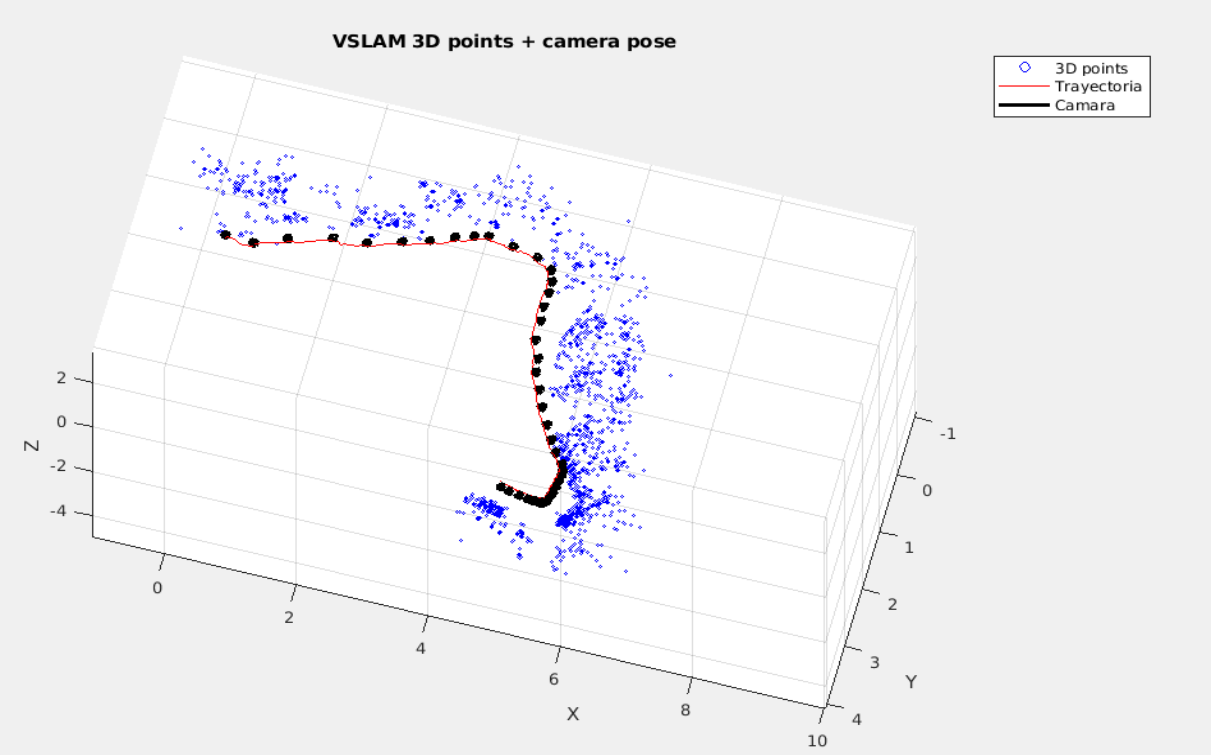


Figura 34. Mapa 3D.

Se observa claramente cómo, entre el punto inicial y final, hay un *offset* o *drift*, esto es algo esperado ya que en cada iteración se calcula el cambio relativo de cámara entre cada imagen y por lo tanto hay un error que se va acumulando.

A continuación, se aplican las técnicas para cerrar el lazo. Las Figura 35, Figura 36 y Figura 37 muestran el mapa tras la aplicación de *loop closure* en vista cenital, vista frontal y en 3D, respectivamente.

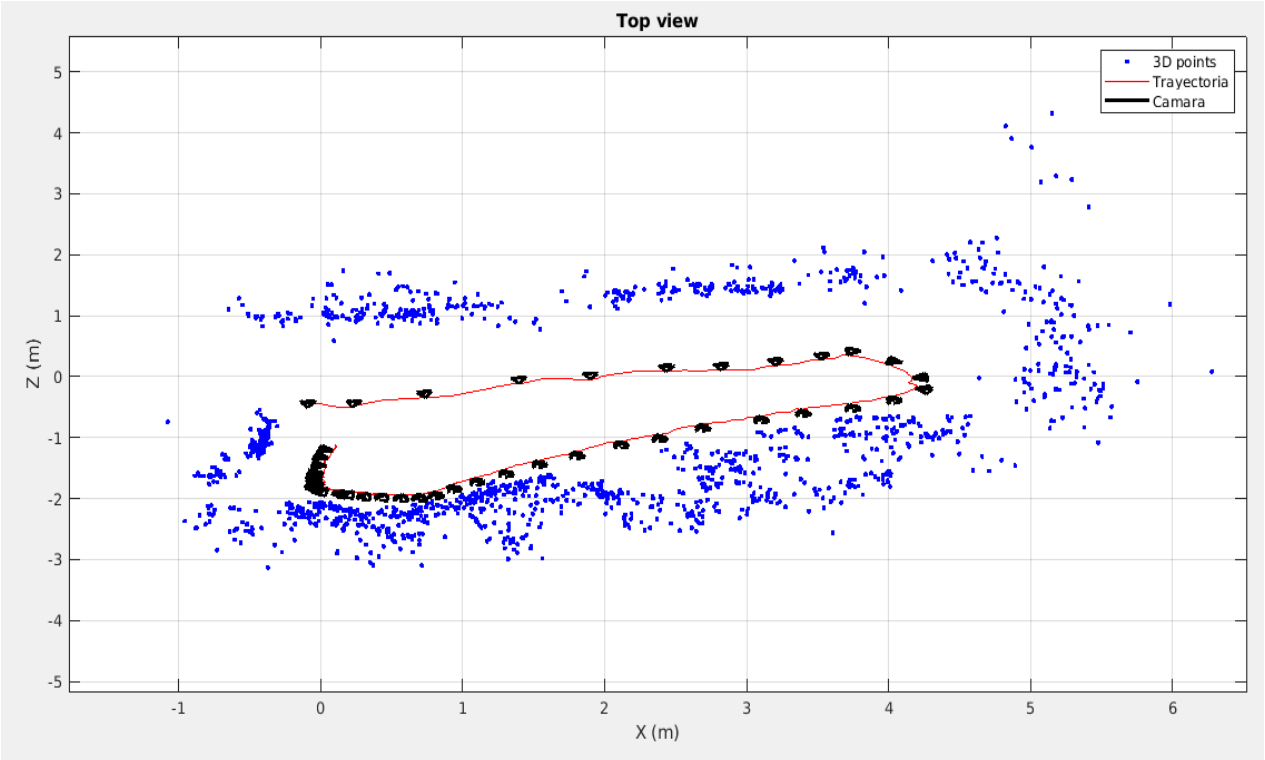


Figura 35. Mapa 3D aplicando *loop closure*, vista cenital.

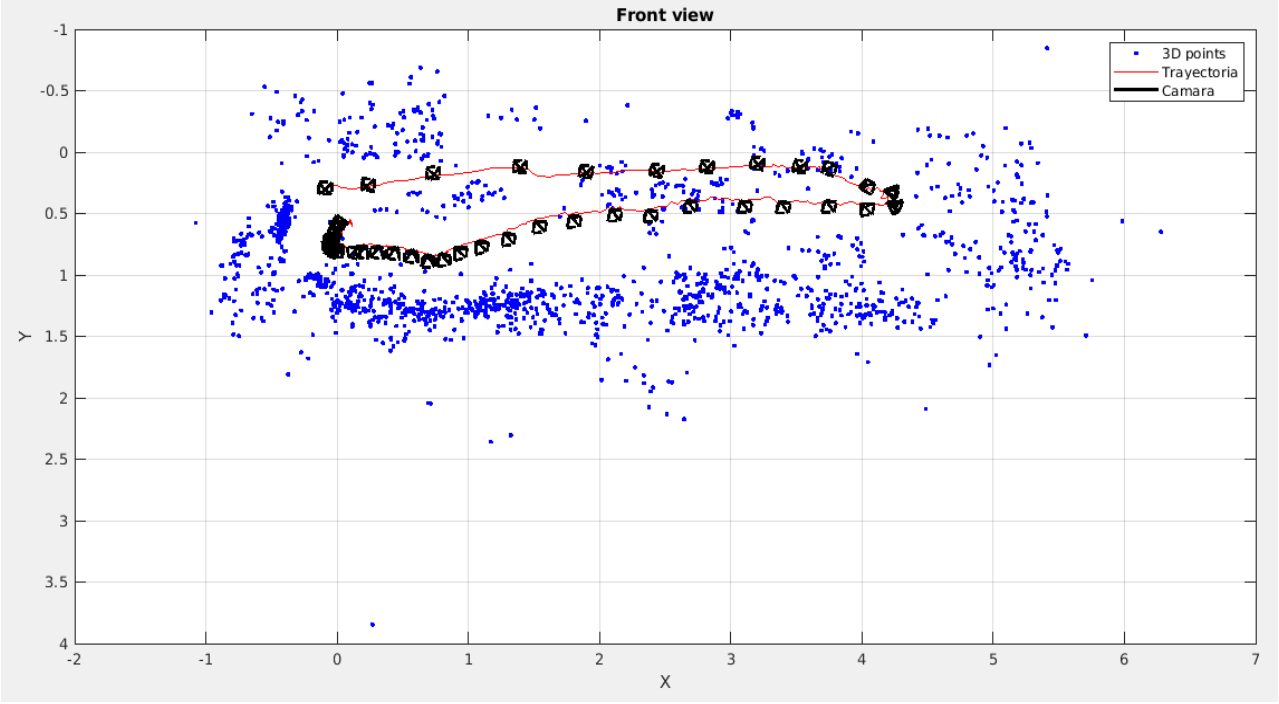


Figura 36. Mapa 3D aplicando *loop closure*, vista frontal.

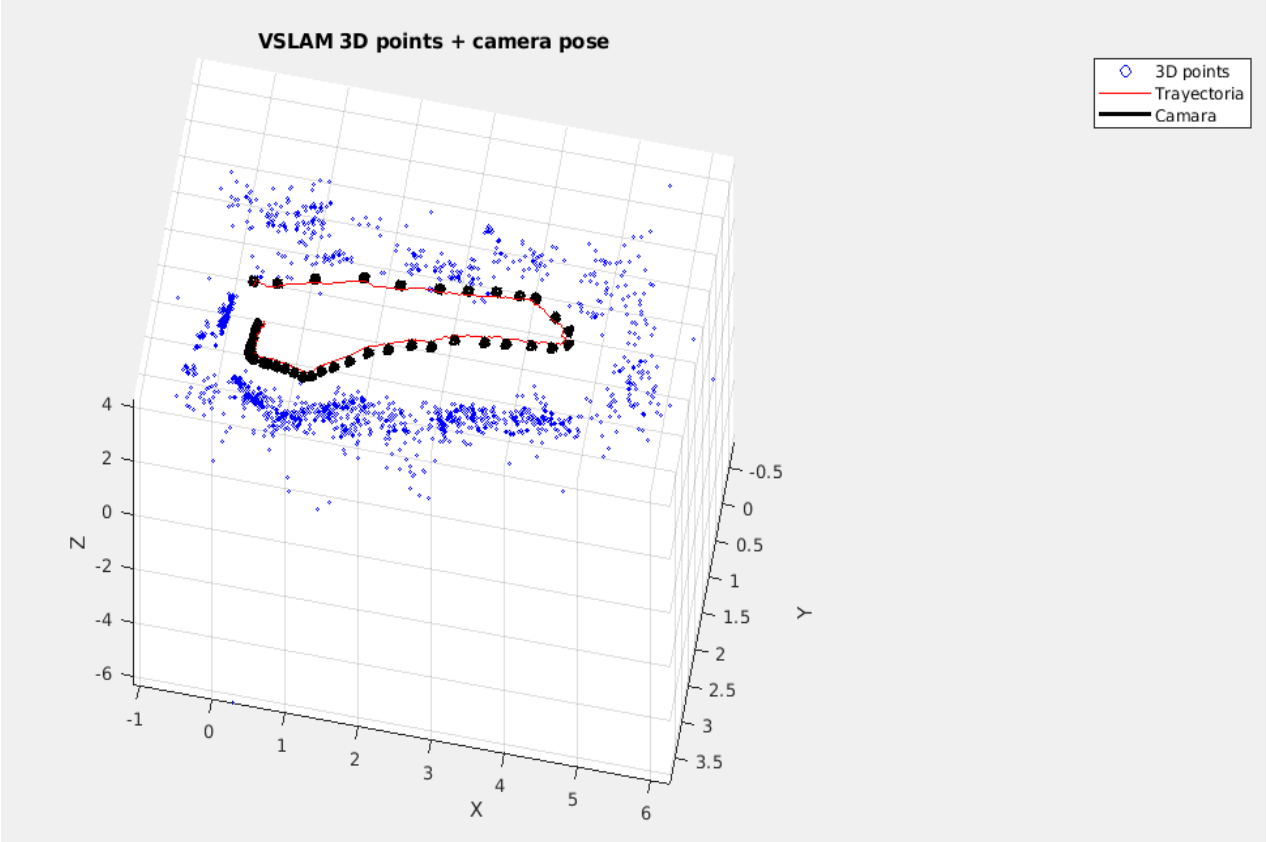


Figura 37. Mapa 3D aplicando *loop closure*.

Al aplicar las técnicas de cerrar lazo se ve como se mejora mucho la trayectoria del dron, la posición inicial y final prácticamente están en el mismo punto en el mapa 3D creado. Después de aplicar el cerrado de lazo, se siguen viendo unas pequeñas desviaciones entre la posición inicial y final, esto puede ser debido al número de puntos que se han utilizado para cerrar el lazo.

En la Figura 38 se ha comparado el mapa creado por el dron, con el tamaño real de la habitación. Para hacer este ejercicio, se han superpuesto la nube de puntos del mapa 3D desde una vista cenital, con el rectángulo (sombreado en azul) que limita la superficie real de la habitación (aprox. 6 x 4 m).

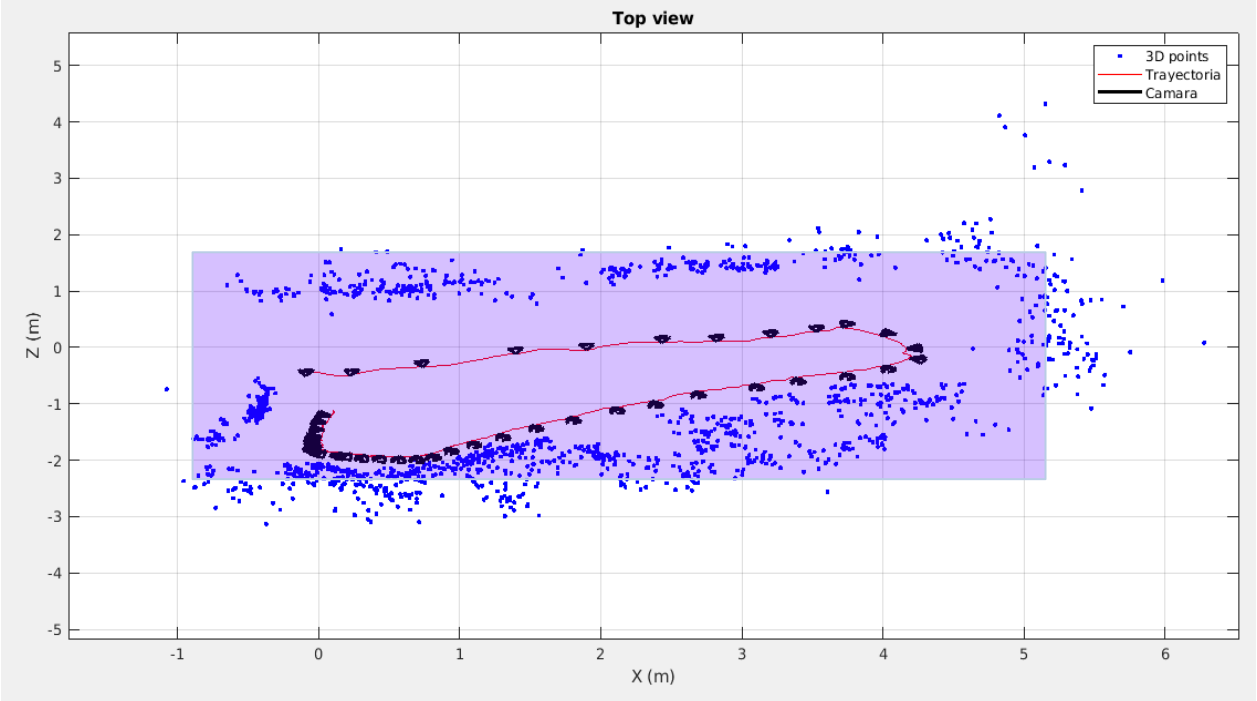


Figura 38. Mapa 3D aplicando *loop closure*, vista frontal incluyendo área real de la habitación.

Se puede observar que algunos puntos están fuera del rectángulo azul, pero en general, la mayoría de los puntos están dentro del área de la habitación. Hay que tener en cuenta que el factor escala y profundidad al aplicar VSLAM monocular son una estimación.

3.3. VSLAM modo localización

En este apartado, se incluyen los resultados al ejecutar el programa en modo "localización". Se parte de un modelo creado anteriormente y el objetivo es determinar la posición del dispositivo usando imágenes tomadas por el mismo. Se muestran los resultados obtenidos al localizar dos imágenes (Figura 39 y Figura 47) obtenidas por el dron.

Figura 39. Imagen 1 usada en la localización.

De esta imagen se obtienen los puntos ORB, en nuestro caso está definido un máximo de 500 puntos por imagen. La Figura 40 muestra la imagen original superpuesta con los puntos ORB obtenidos. En esta imagen se han podido obtener el máximo de puntos.

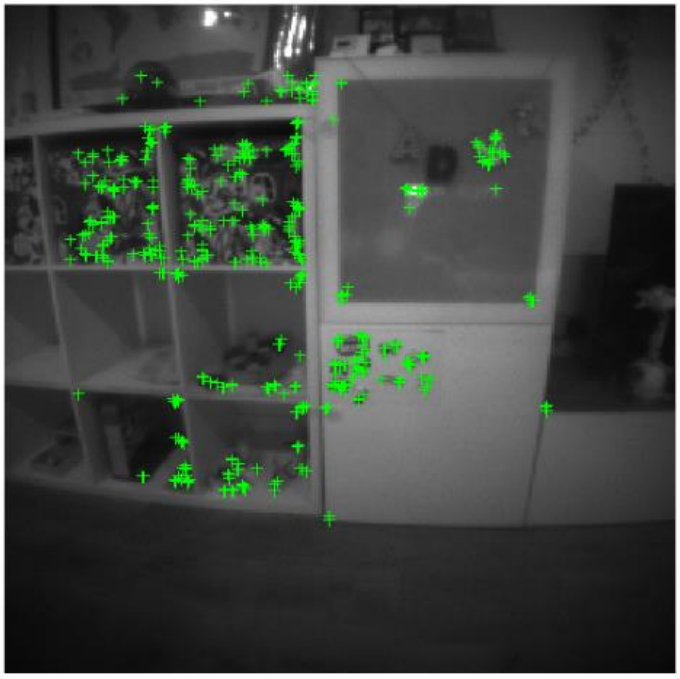


Figura 40. Imagen 1 con los puntos ORB.

Lo siguiente es buscar con cuál de las imágenes de nuestro modelo tiene mayor número de emparejamientos. Para ello, comparamos la información de los descriptores de nuestro modelo con los de la imagen a localizar. La Figura 41 muestra el número de emparejamiento en el eje “y” y el identificador de la imagen del modelo en el eje “x”.

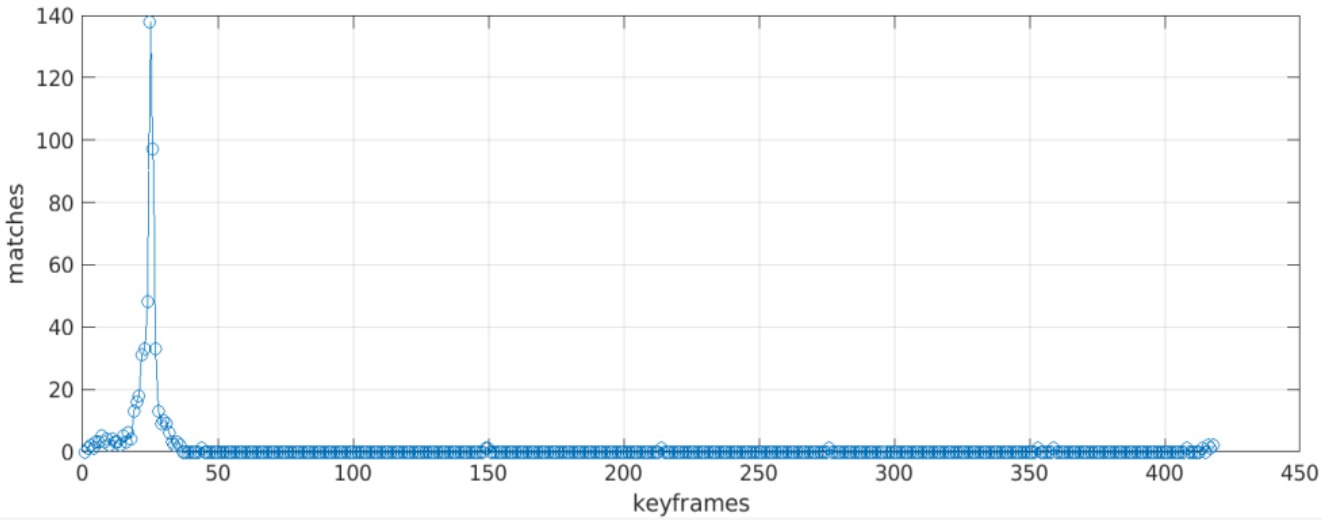


Figura 41. Número de emparejamiento entre los descriptores del modelo y la imagen 1.

Se observa que hay una zona donde los emparejamientos aumentan. Esto quiere decir que nuestro dron está situado cerca de estas posiciones. Para obtener la pose del dron utilizaremos la imagen que tiene mayor número de emparejamientos con la imagen a localizar; en este caso, 138 puntos. La Figura 42 muestra la imagen original con los 138 puntos.

Figura 42. Imagen 1 con los puntos ORB obtenidos en el emparejamiento.

De los 138 puntos que se consideran emparejados, nos quedamos solo con los puntos que tienen un punto asociado en el mapa 3D de nuestro modelo. La Figura 43 muestra la imagen original con los puntos emparejados que solo existen en el modelo 3D, 125 puntos.

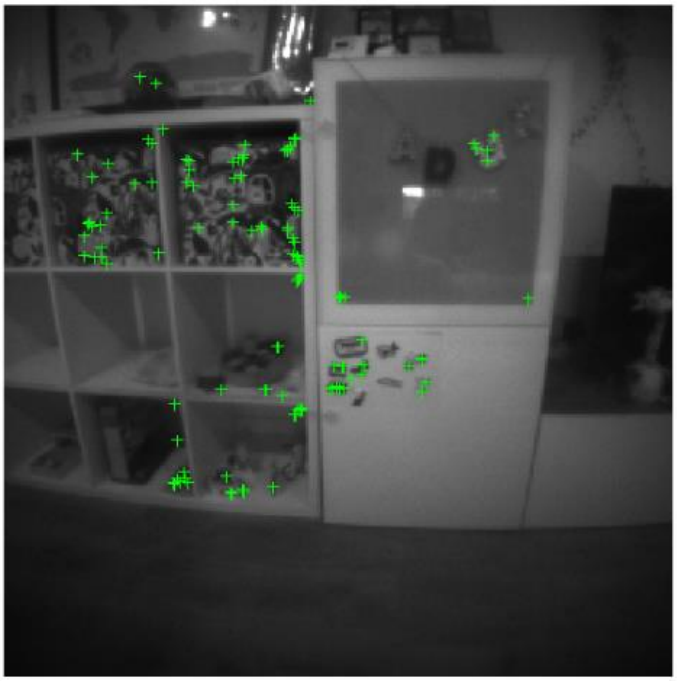


Figura 43. Imagen 1 con los puntos ORB obtenidos en el emparejamiento que existen en el mapa 3D.

Y, finalmente, aplicamos triangulación y RANSAC para obtener la pose de nuestro dron. El número final de puntos válidos se reduce hasta 100 "*inliers*", pero son suficientes para calcular la pose de forma correcta.

Figura 44. Imagen 1 con los *inliers* (100).

Los resultados de la triangulación entre los puntos 3D de nuestro modelo y los puntos emparejados de la imagen a localizar en 2D, nos dan la pose de la cámara. La Figura 45 muestra el resultado de la localización con respecto al mapa previamente creado.

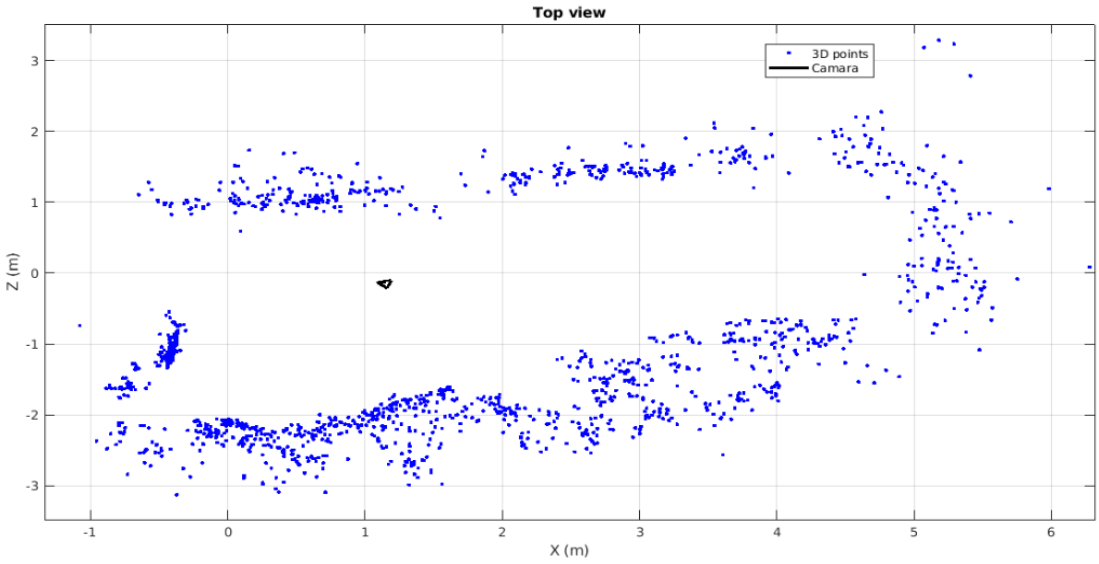


Figura 45. Localización del dron usando la imagen 1 en el mapa 3D, vista cenital.

La Figura 46 muestra un zoom en la zona donde se encuentra el dron.

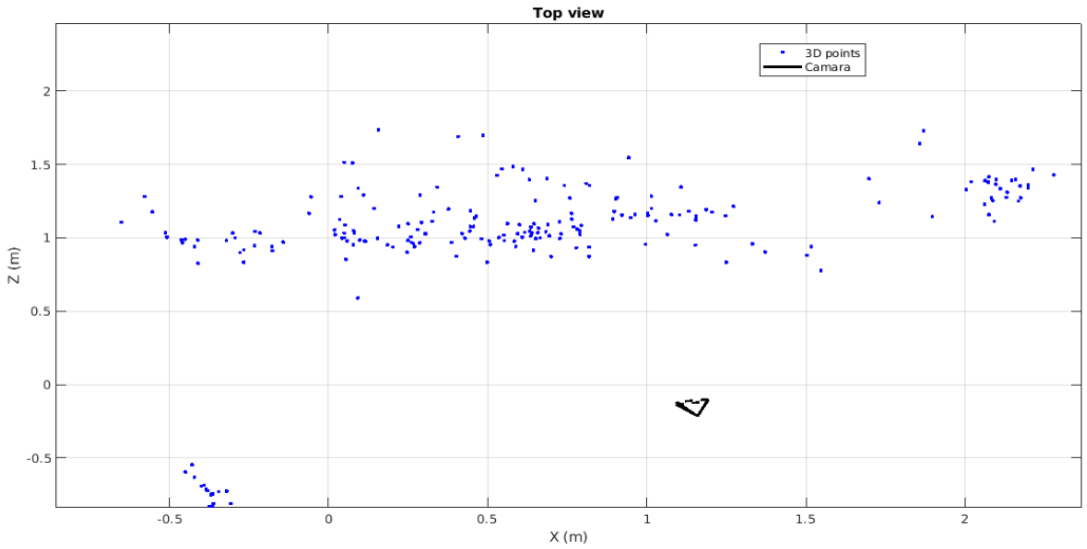


Figura 46. Zoom en la localización del dron usando la imagen 1 en el mapa 3D, vista cenital.

La posición calculada por el algoritmo es coherente con la posición real del dron.

El mismo proceso se ha aplicado a una segunda imagen (Figura 47), definida como "Imagen 2".

Figura 47. Imagen 2 usada en la localización.

Obtenemos los puntos ORB, en este caso también se pueden obtener el máximo de 500 puntos. La Figura 48 muestra la Imagen 2 con todos los puntos OBR obtenidos.

Figura 48. Imagen 2 con todos los puntos ORB.

Se compara la información de los descriptores de nuestro modelo con los de la imagen a localizar. La Figura 49 muestra el número de emparejamientos en el eje “y” y el identificador de la imagen del modelo en el eje “x”. Se obtiene un número máximo de emparejamientos de 170 puntos.

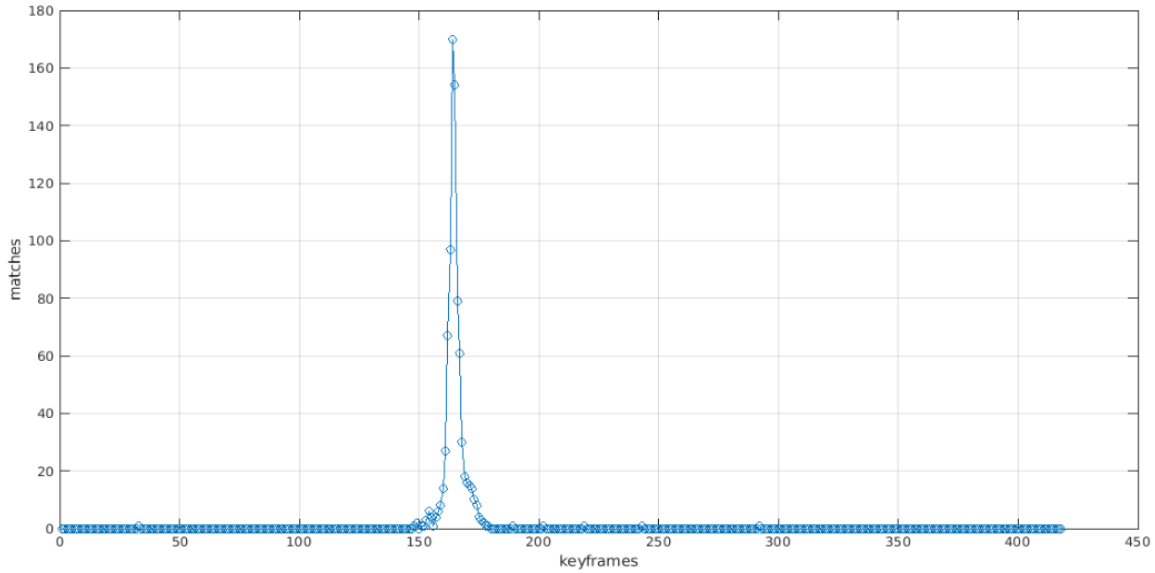


Figura 49. Número de emparejamiento entre los descriptores del modelo y la imagen 2.

Para obtener la pose del dron, utilizaremos la imagen que tiene mayor número de emparejamientos con la imagen a localizar, es este caso 170 puntos (Figura 50).

Figura 50. Imagen 2 con los puntos ORB obtenidos en el emparejamiento.

De los 170 puntos que se consideran emparejados, nos quedamos solo con los puntos que tienen un punto asociado en el mapa 3D de nuestro modelo. La Figura 51 muestra la imagen

original con los puntos emparejados que solo existen en el modelo 3D, 140 puntos.

Figura 51. Imagen 2 con los puntos ORB obtenidos en el emparejamiento que existen en el mapa 3D.

Al aplicar filtrado por RANSAC, nos quedamos con 65 puntos válidos para hacer la triangulación entre los puntos 3D de nuestro mapa y los puntos 2D de la imagen que queremos localizar (Figura 52).

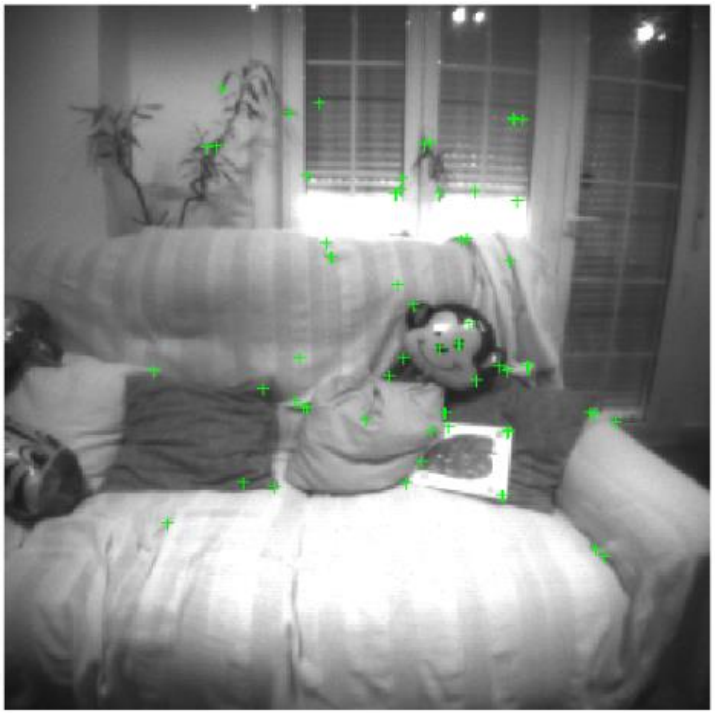


Figura 52. Imagen 2 con los *inliers* (65 puntos).

El resultado de la triangulación nos da la pose del dron. La Figura 53 muestra la posición y orientación de nuestro dron respeto al mapa 3D.

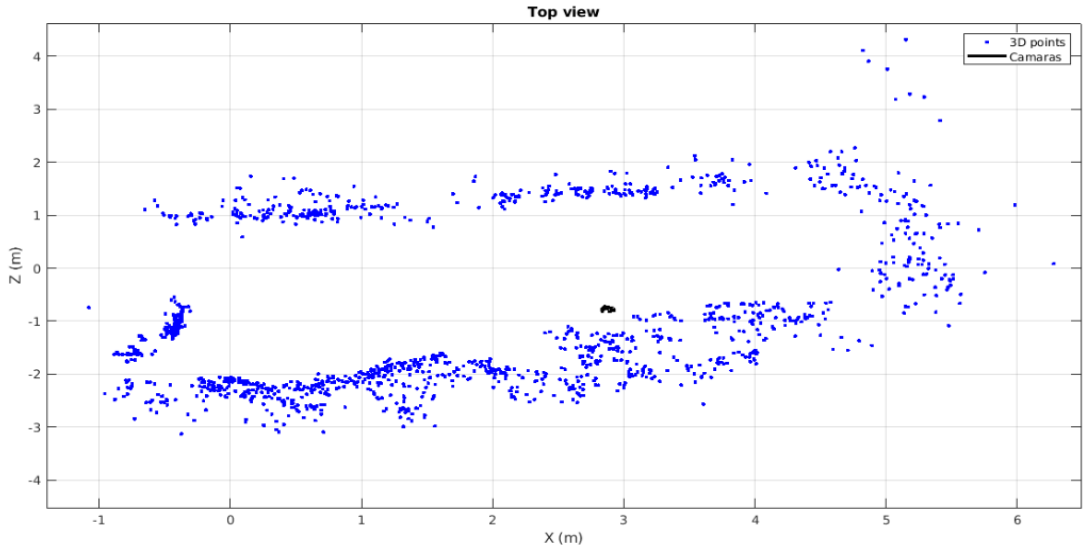


Figura 53. Localización del dron usando la imagen 2 en el mapa 3D, vista cenital.

La Figura 54 muestra un zoom para ver mejor la pose del dron en el mapa 3D.

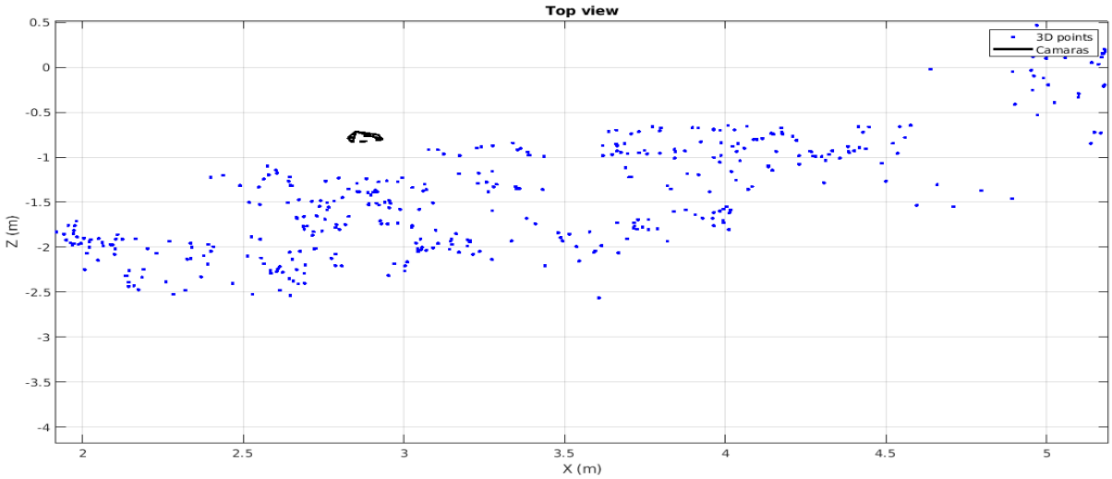


Figura 54. Zoom en la localización del dron usando la imagen 2 en el mapa 3D, vista cenital.

Capítulo 4. Conclusiones y trabajos futuros

En este proyecto se han desarrollado dos líneas principales de trabajo con el objetivo de evaluar la viabilidad de aplicar técnicas de VSLAM (Visual Simultaneous Localization and Mapping) a imágenes capturadas por un dron comercial de bajo coste.

Por un lado, se ha llevado a cabo un estudio detallado del dron Crazyflie 2.0, tanto a nivel de hardware como de software, lo que ha permitido comprender sus capacidades, limitaciones y posibles aplicaciones dentro de un entorno de investigación. Por otro lado, se ha realizado la calibración de la cámara utilizada por el dron, con el fin de aplicar sobre ella técnicas de VSLAM y analizar su desempeño.

El dron Crazyflie 2.0 ha sido el medio para la obtención de las imágenes. Si bien las técnicas de VSLAM están en constante evolución, en este proyecto se ha tomado como referencia el trabajo desarrollado por el equipo de la Universidad de Zaragoza con ORB-SLAM [10]. No obstante, en este caso se ha optado por realizar la implementación completa en MATLAB, debido a su facilidad para el prototipado y la visualización intermedia de resultados, lo que ha facilitado el análisis detallado de cada una de las etapas del proceso.

Los resultados obtenidos muestran que es viable implementar un modelo VSLAM utilizando la cámara incluida por defecto en el dron. Se ha podido crear un modelo 3D de un entorno de interior y, a su vez, se ha podido comprobar que también es posible localizar el dron en el mismo. No obstante, se han identificado algunas limitaciones significativas. En primer lugar, el uso en exteriores resulta inviable. El bajo peso del dron impide mantener un vuelo estable en presencia de viento, y las condiciones de iluminación exterior generan imágenes que

no son adecuadas para la aplicación de técnicas VSLAM. En segundo lugar, incluso en entornos interiores, la calidad de las imágenes capturadas por la cámara es baja. Su alta sensibilidad a la iluminación y su resolución limitada dificultan la detección de puntos de interés, haciendo que en muchos casos no se pueda obtener suficiente información visual para realizar el seguimiento.

Como posibles líneas de investigación y desarrollo a futuro, se proponen las siguientes:

- Implementación en tiempo real: Para ello, sería necesario optimizar los algoritmos empleados y migrar el código actual a un lenguaje de más bajo nivel, como C, que permita una mayor eficiencia computacional.
- Ejecución en el propio dron: Una vez optimizado el código, se podría estudiar la posibilidad de ejecutarlo directamente en el dron Crazyflie, concretamente en la AI-deck. Para ello, sería fundamental verificar que dicha plataforma cuenta con la capacidad de cómputo y memoria necesarias para procesar VSLAM en tiempo real.
- Fusión sensorial con la IMU: Incorporar las mediciones de la Unidad de Medida Inercial (IMU) del dron como entrada adicional al algoritmo de VSLAM, con el objetivo de mejorar la robustez y precisión del sistema de localización y mapeado.
- Integración de ORB-SLAM directamente: Explorar la integración directa de la librería ORB-SLAM desarrollada por la Universidad de Zaragoza. Aunque en este trabajo se ha optado por Matlab por motivos didácticos y de análisis, una implementación con dicha librería permitiría una evaluación más realista y comparable con desarrollos actuales del estado del arte.

Bibliografía

[1] Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous localization and mapping: part I. IEEE Robotics & Automation Magazine, 13(2), 99–110. https://doi.org/10.1109/mra.2006.1638022

[2] SLAM (localización y mapeo simultáneos) – MATLAB y Simulink. (s.f.). MathWorks - Creador de MATLAB y Simulink - MATLAB y Simulink - MATLAB & Simulink. https://es.mathworks.com/discovery/slam.html

[3] Escáner Láser Slam100 – Escáner láser portátil SLAM200/SLAM100/SLAM2000 de Feima Robotics. (s.f.). Feima Robotics SLAM200/SLAM100/SLAM2000 handheld lidar scanner. http://www.slam100.com/es/escaner-laser-slam100/

[4] Arducam 1MP*2 Wide Angle Stereo Camera for Raspberry Pi, Jetson Nano and Xavier NX, Dual OV9281 Monochrome Global Shutter Camera Module. (s.f.). Simplifying embedded vision for all. - Arducam. https://www.arducam.com/arducam-1mp2-wide-angle-stereo-camera-for-raspberry-pi-jetson-nano-and-xavier-nx-dual-ov9281-monochrome-global-shutter-camera-module.html

[5] Colaboradores de los proyectos Wikimedia. (2009, 2 de junio). Kinect - Wikipedia, la enciclopedia libre. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Kinect#/media/Archivo:Xbox-360-Kinect-Standalone.png

[6] Mur-Artal, R., & Tardos, J. D. (2017). ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. IEEE Transactions on Robotics, 33(5), 1255–1262. https://doi.org/10.1109/tro.2017.2705103

[7] Tourani, A., Bavle, H., Sanchez-Lopez, J. L., & Voos, H. (2022). Visual SLAM: What Are the Current Trends and What to Expect? Sensors, 22(23), 9297. https://doi.org/10.3390/s22239297

[8] Davison, A. J., Reid, I. D., Molton, N. D., & Stasse, O. (2007). MonoSLAM: Real-Time Single Camera SLAM. IEEE Transactions on Pattern Analysis and Machine Intelligence, 29(6), 1052–1067. https://doi.org/10.1109/tpami.2007.1049

[9] Parallel Tracking and Mapping for Small AR Workspaces (PTAM). (s.f.). Information Engineering Main/Home Page. https://www.robots.ox.ac.uk/~gk/PTAM/

[10] Computer Vision Group - Visual SLAM - LSD-SLAM: Large-Scale Direct Monocular SLAM. (s.f.). Computer Vision Group - Home. https://cvg.cit.tum.de/research/vslam/lsdslam

[11] Mur-Artal, R., Montiel, J. M. M., & Tardos, J. D. (2015). ORB-SLAM: A Versatile and Accurate Monocular SLAM System. IEEE Transactions on Robotics, 31(5), 1147–1163. https://doi.org/10.1109/tro.2015.2463671

[12] GitHub - raulmur/ORB_SLAM: A Versatile and Accurate Monocular SLAM. (s.f.). GitHub. https://github.com/raulmur/ORB_SLAM

[13] GitHub - raulmur/ORB_SLAM2: Real-Time SLAM for Monocular, Stereo and RGB-D Cameras, with Loop Detection and Relocalization Capabilities. (s.f.). GitHub. https://github.com/raulmur/ORB_SLAM2

[14] Campos, C., Elvira, R., Rodriguez, J. J. G., M. Montiel, J. M., & D. Tardos, J. (2021). ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM. IEEE Transactions on Robotics, 1–17. https://doi.org/10.1109/tro.2021.3075644

[15] GitHub - UZ-SLAMLab/ORB_SLAM3: ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM. (s.f.). GitHub. https://github.com/UZ-SLAMLab/ORB_SLAM3

[16] System overview | Bitcraze. (s.f.). Home | Bitcraze. https://www.bitcraze.io/documentation/system/

[17] Juan Miguel Sánchez García, Trabajo Final de Máster, “Creación de mapas fotográficos 3D del interior de edificaciones mediante el uso de drones“ (Febrero-2024)

[18] Crazyflie 2.0 | Bitcraze. (s.f.). Home | Bitcraze. https://www.bitcraze.io/products/old-products/crazyflie-2-0/

[19] Crazyflie 2.1| Bitcraze. (s.f.). Home | Bitcraze. https://www.bitcraze.io/products/old-products/crazyflie-2-1/

[20] Crazyflie 2.1+ | Bitcraze. (s.f.). Home | Bitcraze. https://www.bitcraze.io/products/crazyflie-2-1-plus/

[21] Crazyflie 2.1 Brushless | Bitcraze. (s.f.). Home | Bitcraze. https://www.bitcraze.io/products/crazyflie-2-1-brushless/

[22] Expansion decks of the Crazyflie 2.x | Bitcraze. (s.f.). Home | Bitcraze. https://www.bitcraze.io/documentation/system/platform/cf2-expansiondecks/

[23] Crazyradio 2.0 | Bitcraze. (s.f.). Home | Bitcraze.

https://www.bitcraze.io/products/crazyradio-2-0/

[24] AI-deck 1.1. (s.f.). Bitcraze Store. https://store.bitcraze.io/collections/decks/products/ai-deck-1-1

[25] Flow deck v2. (s.f.). Bitcraze Store. https://store.bitcraze.io/collections/decks/products/flow-deck-v2

[26] Installing USB driver on Windows | Bitcraze. (s.f.). Home | Bitcraze. https://www.bitcraze.io/documentation/repository/crazyradio-firmware/master/building/usbwindows/

[27] USB permissions | Bitcraze. (s.f.). Home | Bitcraze. https://www.bitcraze.io/documentation/repository/crazyflie-lib-python/master/installation/usb_permissions/

[28] Installation Instructions | Bitcraze. (s.f.). Home | Bitcraze. https://www.bitcraze.io/documentation/repository/crazyflie-clients-python/master/installation/install/

[29] Getting started with the AI deck | Bitcraze. (s.f.). Home | Bitcraze. https://www.bitcraze.io/documentation/tutorials/getting-started-with-aideck/

[30] aideck-gap8-examples/docs/examples/wifi-streamer.md at master · bitcraze/aideck-gap8-examples. (s.f.). GitHub. https://github.com/bitcraze/aideck-gap8-examples/blob/master/docs/examples/wifi-streamer.md

[31] Home | Bitcraze. (s.f.). Home | Bitcraze. https://www.bitcraze.io/documentation/repository/crazyflie-lib-python/master/

[32] What Is Camera Calibration? - MATLAB & Simulink. (s.f.). https://es.mathworks.com/help/vision/ug/camera-calibration.html

[33] OpenCV: Camera Calibration. (s.f.). OpenCV documentation index. https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html

[34] Calibration Patterns - MATLAB & Simulink. (s.f.). Access Denied. https://es.mathworks.com/help/vision/ug/calibration-patterns.html

[35] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: an efficient alternative to SIFT or SURF.

In IEEE International Conference on Computer Vision (ICCV), pages 2564–2571, Barcelona, Spain, 2011.

[36] extractFeatures - Extract interest point descriptors - MATLAB. (s.f.). Access Denied. https://es.mathworks.com/help/vision/ref/extractfeatures.html

[37] matchFeatures - Find matching features - MATLAB. (s.f.). Access Denied. https://es.mathworks.com/help/vision/ref/matchfeatures.html

[38] Gao, X.-S., X.-R. Hou, J. Tang, and H.F. Cheng. "Complete Solution Classification for the Perspective-Three-Point Problem." IEEE Transactions on Pattern Analysis and Machine Intelligence. Volume 25,Issue 8, pp. 930–943, August 2003.

[39] estimateEssentialMatrix - Estimate essential matrix from corresponding points ina pair of images - MATLAB. (s.f.). Access Denied. https://www.mathworks.com/help/vision/ref/estimateessentialmatrix.html

[40] estrelpose - Calculate relative rotation and translation between camera poses - MATLAB. (s.f.). Access Denied. https://es.mathworks.com/help/vision/ref/estrelpose.html

[41] estworldpose - Estimate camera pose from 3-D to 2-D point correspondences - MATLAB. (s.f.). Access Denied. https://es.mathworks.com/help/vision/ref/estworldpose.html

[42] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon. Bundle adjustment a modern synthesis. In Vision algorithms: theory and practice, pages 298–372. 2000.

[43] bundleAdjustment - Adjust collection of 3-D points and camera poses - MATLAB. (s.f.). Access Denied. https://es.mathworks.com/help/vision/ref/bundleadjustment.html

[44] Galvez-López, D., and J. D. Tardos. “Bags of Binary Words for Fast Place Recognition in Image Sequences.” IEEE Transactions on Robotics, vol. 28, no. 5, Oct. 2012, pp. 1188–97. DOI.org (Crossref), https://doi.org/10.1109/TRO.2012.2197158.

[45] Monocular Visual Simultaneous Localization and Mapping - MATLAB & Simulink. (s.f.). Access Denied. https://es.mathworks.com/help/vision/ug/monocular-visual-simultaneous-localization-and-mapping.html

Listado de siglas, abreviaturas y acrónimos

UAV: Unmanned Aerial Vehicle

SLAM: Simultaneous Localization and Mapping

VSLAM: Visual Simultaneous Localization and Mapping

LIDAR: Light Detection and Ranging

GPS: Global Positioning System

IMU: Inertial Measurement Unit

ORB: Oriented FAST and Rotated BRIEF

SURF: Speeded-Up Robust Features

FAST: Features from Accelerated Segment Test

BRIEF: Binary Robust Independent Elementary Features

RANSAC: RANdom SAmple Consensus

Apéndice A

Todo el material creado para este proyecto está disponible en el siguiente repositorio de GitHub. Esto incluye el código de Matlab para ejecutar el algoritmo VSLAM, las imágenes utilizadas para calibrar la cámara del dron y las imágenes utilizadas para generar el modelo VSLAM

https://github.com/efrentxo/VSLAM_TFM