Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería en Electrónica y Automática Industrial

Trabajo Fin de Grado

Comparativa de estrategias de SLAM visual en sistemas empotrados de bajo coste.

Autor: Antonio Guerrero Gómez-Olmedo

Tutor: Sebastián Sánchez Prieto

2021

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería en Electrónica y Automática Industrial

Trabajo Fin de Grado

Comparativa de estrategias de SLAM visual en sistemas empotrados de bajo coste.

Autor: Antonio Guerrero Gómez-Olmedo

Director: Sebastián Sánchez Prieto

Tribunal:

Presidente: Julia María Clemente Párraga

Vocal 1º: Concepción Batanero Ochaíta

Vocal 2º: Sebastián Sánchez Prieto

Calificación: .....

Fecha: .....

Quiero dedicar este trabajo a mi madre y a mi hermano por haberme acompañado y apoyado a lo largo del mismo.

Y agradecer a Sebastián y a Manuel por haberme dado la oportunidad de realizarlo y por haber hecho todo lo que estaba en su mano.

Resumen

En este proyecto se pretende evaluar la posibilidad de implementar un sistema empotrado de SLAM visual a partir de la cámara de Intel ZR300, la placa de desarrollo UDOO y comparar el comportamiento de dos bibliotecas de SLAM con opción de integración en ROS, ORBSLAM2 y RTAB-Map.

Palabras clave: SLAM Visual, ROS, Sistemas empotrados

Abstract

This project aims to evaluate the possibility of accomplish an embedded visual SLAM system from the Intel ZR300 camera, the UDOO development board and to compare the behaviour of two SLAM libraries with the option of integration in ROS, ORBSLAM2 y RTAB-Map.

Keywords: SLAM-Visual, ROS, Sistemas empotrados.

Índice general

Índice de figuras

Capítulo 1

Introducción

We are at the very beginning of time for the human race. It is not unreasonable that we grapple with problems. But there are tens of thousands of years in the future. Our responsibility is to do what we can, learn what we can, improve the solutions, and pass them on.

Richard P. Feynman, What Do You Care What Other People Think?

En nuestros días, gracias a la investigación y el desarrollo en distintos campos de la tecnología, disponemos de sistemas de procesamiento muy pequeños y ligeros. Ellos nos permiten orientar su tiempo de computación de forma mucho más específica y controlada que un ordenador común, lo que nos sirve para trabajar en aplicaciones concretas en tiempo real, esto es, ejecutando acciones en un tiempo acotado. Los que son capaces de cumplir estas condiciones son clasificados como sistemas empotrados. Como ejemplos más conocidos de este tipo de sistemas tenemos las plataformas Arduino o Raspberry Pi para un público más extendido o DSP's y FPGA's enmarcados en ámbitos más profesionales.

Por otro lado, las técnicas SLAM, (*Simultaneous Localization and Mapping*), intentan abordar un problema ampliamente enfrentado por la robótica orientada a la navegación. Este dilema proviene de la dificultad de crear un mapa del entorno de un robot debido a la unión entre los errores de sus sensores de percepción y los errores de medición de su propio desplazamiento. En los primeros intentos de resolver este problema, se le añadía el factor del procesamiento que, por el nivel de tecnología de la época cuando se empezó a plantear (desde su primera mención en la conferencia del IEEE en 1986 de San Francisco)[1], se debía realizar externamente, lo que permitía poco margen de movimiento. Este aspecto puede haber sido resuelto con los sistemas empotrados mencionados en el párrafo anterior y es sobre el que va tratar este proyecto, que va a consistir en el empleo y la evaluación de la viabilidad para un sistema concreto. Las fases del trabajo propuesto son las siguientes:

- Realizar una caracterización de los componentes que van a ser empleados en el trabajo.
- Implementar un sistema de visión artificial y establecer un entorno de trabajo.
- Analizar el comportamiento de las bibliotecas de RGB-D SLAM `RTAB-Map` y `ORB-SLAM2`.
- Documentar ventajas y desventajas de las distintas estrategias de mapeado y parámetros de configuración.

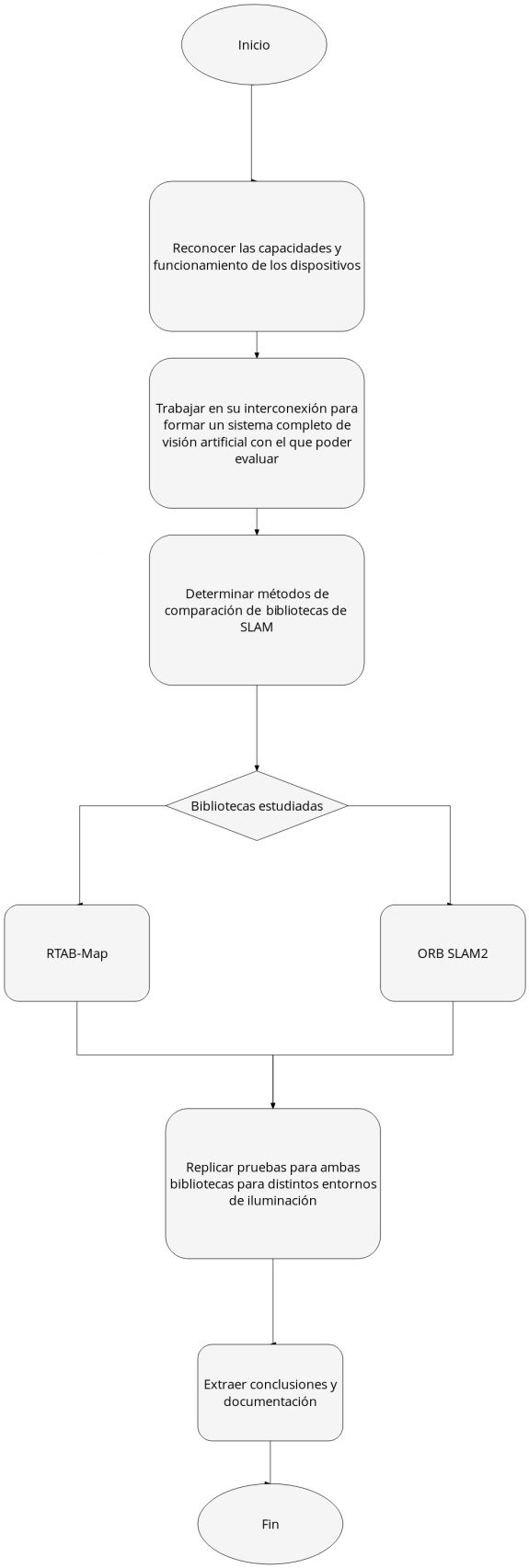


Figura 1.1: Diagrama de flujo de trabajo

Capítulo 2

Estudio teórico

2.1 Introducción

En este capítulo voy a comenzar por establecer el estado actual del *SLAM*, así como las diferentes aplicaciones y obstáculos a los que se está enfrentando. Continuaré con un apartado sobre los sistemas empotrados, para entender en qué márgenes de operación nos movemos. Siguiendo con el *hardware*, el siguiente paso es conocer cuáles son los principios de funcionamiento de una cámara, de cara a obtener su modelo matemático y los aspectos que influyen en su calibración; ofreciendo así un primer punto de vista, que se completará en el capítulo 3. Después, incluiré un breve resumen histórico sobre cómo se ha llegado hasta los algoritmos de *SLAM* actuales, junto con las distintas perspectivas desde las que se está trabajando. Finalmente, hablaré del algoritmo principal en el que se centra el *SLAM* visual, incluyendo un apartado sobre *ROS* y su implicación en este tipo de proyectos, hasta llegar a las bibliotecas concretas de que se compone este trabajo.

2.2 Estado del arte

Me gustaría comenzar definiendo cuatro conceptos que emplearemos a lo largo del texto: Fotogrametría, *Structure from Motion*, *Visual Odometry* y por último el *SLAM*.
Empezando por la fotogrametría, que es la que queda más alejada de este ámbito, fue, en su inicio, expresada como la técnica de realización de mapas de grandes extensiones por medio de fotografía aérea. Hago mención de ella, porque la estructura desde el movimiento es una técnica de fotogrametría, la cual, consiste en estimar medidas de estructuras en 3 dimensiones, a partir de secuencias de imágenes emparejadas de 2 dimensiones.

Siguiendo con la odometría visual, consiste en estimar el desplazamiento de la posición de la cámara o el sistema, en función del desplazamiento de las posiciones de marcas localizadas, entre dos o más imágenes en principio consecutivas.

Finalmente, el *SLAM*, por su nombre, consiste en el mapeado o estimación de medidas y la localización, estimación del desplazamiento simultáneamente, lo que se podría entender como la unión entre los procesos anteriores. El *SLAM*, a día de hoy se considera como un problema resuelto desde el punto teórico, pero a la hora de llevarlo a la práctica aún quedan factores por refinar para tener una ejecución óptima. Los principales problemas serían la localización en un entorno dinámico, que, al haber personas u objetos móviles, se pueda falsear la posición; el reconocimiento de lugares ya visitados, que, bien realizado,

implicaría el cierre de lazo y erróneamente provocaría un solapamiento perceptual o el problema del secuestro que consiste en la oclusión de sensores y colocación del robot en otra posición.

Una vez diferenciados estos aspectos de procedimientos relacionados, podemos comenzar con la descripción del estado del arte, desde una competición organizada por la Agencia de proyectos de investigación avanzados de defensa estadounidense o *DARPA*, conocida como *SubT Challenge.* que se encuentra en estos momentos entrando en su etapa final

En ella, se ha planteado a equipos de investigación robótica de todo el mundo, una serie de desafíos para intentar superar el reto de desplazamiento y navegación en terrenos subterráneos desconocidos, como puedan ocurrir en operaciones militares o entornos de primera respuesta de rescate. Con lo que intenta promover nuevos enfoques destinados a mejorar las estrategias de mapeado, navegación y búsqueda de elementos y/o personas para situaciones de alto riesgo de participación de personal.

Esta competición se ha dividido en dos categorías pudiendo inscribirse tanto en una, como en ambas al ser éstas búsquedas complementarias entre sí.

- *System Track*: El objetivo de esta categoría es desarrollar sistemas físicos, con los que competir, e interactuar con escenarios específicos concentrándose en el avance y la evaluación de las nuevas soluciones implementadas.
- *Virtual Track*: Para esta opción, los equipos han tenido que especializarse en desarrollar *software* y algoritmos usando modelos virtuales de sistemas y de entornos para participar en los distintos eventos que se planteen. También permite como explorar su aplicación a recorridos de larga duración y extensión, en entornos simulados desarrollados específicamente para poner a prueba dichos aspectos.

Dado que esta competición está muy avanzada, ya se conocen las distintas soluciones implementadas por los 18 equipos participantes hasta ahora, de las que voy a describir las más significativas.

Varios equipos han optado por soluciones mixtas en las que explorar con *UAV's* aprovechando su versatilidad y dejando para vehículos terrestres tareas más pesadas o las cuestiones de autonomía.

A destacar, tenemos el equipo *CoSTAR* ganador de la segunda etapa, el circuito urbano, formado por *Caltech*, el *MIT*, el *JPL (Jet Propulsion Laboratory de la NASA)* y *KAIST* (Corea del Sur) que participaron en la categoría *System Track* con un equipo multi-robots que incluye un vehículo híbrido, el *Rollocopter*, de gran rendimiento energético con el que sobrevolar obstáculos o situaciones complicadas de forma puntual, un enjambre con el que realizar procesos de *SLAM* en paralelo o un sensor con el que localizarse por medio de la lectura de campos magnéticos.

Para el caso de la categoría virtual, el equipo *Cynet.ai* ha decidido trabajar también con enjambres con la particularidad de aprovechar el *Deep Learning* para desarrollar algoritmos centrados en búsqueda y rescate para situaciones de comunicación limitada.

Un último caso a destacar es el equipo *Robotika*, que participó en ambas categorías con el propósito de alcanzar de la forma más simple y eficiente posible al menos un objetivo, lo que han denominado como programación extrema.

Otros enfoques se caracterizaron por el uso de *RADAR*, por el equipo *MARBLE* para la categoría física o el equipo *Metastable Labs*, que, para el caso virtual, basó su algoritmo intentando emular los equipos de rescate. Bajo esta premisa, consideraron que, reduciendo la información a la forma más concisa posible, permite una comunicación, coordinación y sincronización óptimas entre los componentes.

Por otra parte, al margen de los grandes laboratorios de investigación, también se están dando aplicaciones a los programas que se van a tratar en este trabajo.
Empezando por el `RTAB-Map` tenemos un desarrollo de una silla de ruedas autónoma bajo la perspectiva de optimización de costes [2].
Para el caso de `ORBSLAM` tenemos, para el campo médico más específicamente, un estudio de viabilidad de la Universidad de Zaragoza para realizar *visual SLAM* monocular con secuencias de Endoscopia [3] que ha derivado en el actual programa de investigación *Endomapper*.

2.3 Sistemas empotrados

Profundizando un poco más sobre el concepto comentado en la introducción, un sistema empotrado consiste, generalmente, en un sistema de tiempo real que forma parte de un sistema más grande. Está compuesto de *hardware* y *software* escogidos con el objetivo de realizar una tarea concreta de la forma más eficiente posible.

Para alcanzar y garantizar estas características, el desarrollo se realiza en base a métricas sobre los procesos que se van a realizar. En función de los requerimientos, estas métricas van a controlar parámetros tales como coste de diseño, coste de fabricación, consumo de energía, tamaño físico o por supuesto, tiempo de respuesta.

Una vez determinadas, el siguiente paso es valorar qué procesador y qué tecnología se pueden adaptar mejor a las necesidades. En ambos casos hay que llegar a un compromiso entre coste y especialización frente a flexibilidad y portabilidad. Para los procesadores se puede trabajar con unidades de propósito único, de aplicación específica o de propósito general.

Mientras que, para el caso de tecnología estaríamos hablando de diseño *full custom*, cuando la personalización de las capas es completa; diseño *semi-custom* basado en celdas, cuando se trabaja con bibliotecas de las mismas; diseño *semi-custom* basado en arrays, cuando se trabaja sobre las conexiones entre una capa de transistores previamente colocados y, finalmente, los dispositivos de lógica programable, que independizan los procesos de fabricación y de diseño, entre los que se encuentran las *FPGA*, que se modelarían por medio de lenguajes *HDL*.

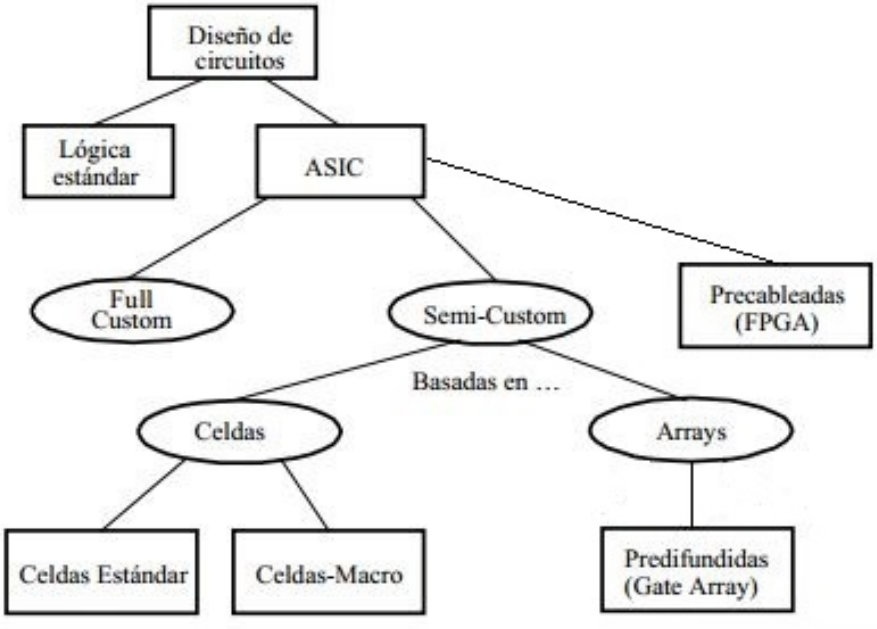


Figura 2.1: Alternativas de tecnología de diseño

Lo que nos lleva a una imagen de la selección según el análisis de requisitos con el siguiente aspecto.

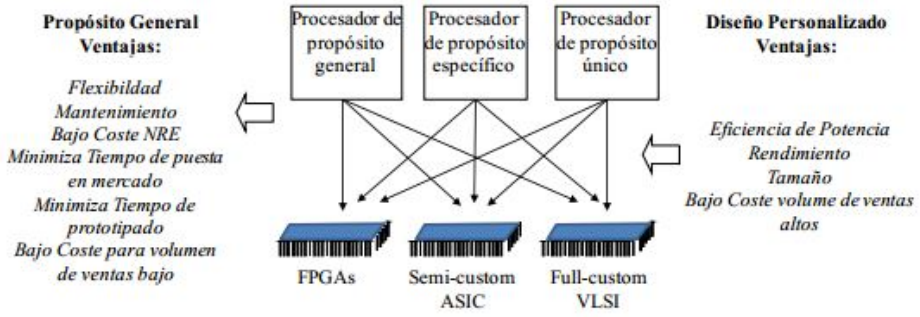


Figura 2.2: Esquema de la relación entre alternativas y métricas

Desde el punto de vista del *software*, pueden emplearse sistemas operativos en tiempo real o *RTOS*, que se clasifican en función de la criticidad de su funcionamiento o de sus tareas. Partiendo de la base de que un tipo de sistema no implica que todas sus tareas tengan esas características, ya que muchas veces podrían necesitar intercalarse con tareas interactivas: Si nos encontramos ante un sistema de cuya respuesta depende la vida de personas o que puede invalidar un proceso de tareas críticas entero, estaremos hablando de un sistema de tiempo real duro o estricto. Si el incumplimiento del tiempo de respuesta puede implicar pérdidas o disminuir la calidad, pero no invalida el desarrollo de la función completa, es un sistema de tiempo real blando o flexible. El caso del sistema de tiempo real firme implica que la pérdida de tiempo en la respuesta hace que esta se descarte. Por último, también hay sistemas empotrados que pueden trabajar sin sistemas de tiempo real o incluso actuando sin sistema operativo, directamente sobre *hardware*.

Entrando más en la programación, cabe destacar los métodos posibles de planificación de tareas. Una posibilidad es trabajar por medio de interrupciones, que indicaran según la prioridad asignada, qué tarea debería ejecutarse. Otra opción necesitaría conseguir plantear todas las tareas como periódicas.

En ese caso, podremos orientarlo como un conjunto de ejecutivos cíclicos, esto es, realizar las tareas secundarias en submúltiplos del período de la tarea principal.

Sirviendo como ejemplos tenemos que el sistema de SLAM visual estará montado sobre un ordenador de placa única. Suelen tener microprocesadores de propósito general ya que están orientados al desarrollo o a propósitos educativos.

Por otra parte, el sistema empotrado de imagen 3D de la cámara ZR300, se encarga de realizar el cálculo de profundidad para su posterior reconstrucción. (Proceso desarrollado en el apartado de componentes del capítulo 3)

Como cámara orientada a la investigación, este tipo de sistema correspondería a sistema de tiempo real flexible. En este caso, de acuerdo al manual [4], se utiliza una *ASIC* integrado en la cámara, que recibe los datos obtenidos desde dos cámaras infrarrojas, para realizar los cálculos.

2.4 Cámara

Como sensor de visión y con la posibilidad de escoger entre monocular, omnidireccional, estéreo y *RGB-D*, la opción escogida es el sistema *RGB-D*, que implica una fusión de sensores que permiten una reconstrucción con profundidad a color.

De cara al *SLAM*, las ventajas que representan frente a las demás son robustez frente a las derivas de giros para los sistemas monoculares y omnidireccionales, y reducción de proceso de cálculo entre las dos cámaras del caso estéreo. Por contra, tiene que, al ser el cálculo de profundidad un proceso de reconstrucción por láser, la longitud de onda de éste, se ve ocultada entre la radiación solar, saturando el sensor y prácticamente restringiendo su uso a interiores.

2.4.1 Óptica

Para realizar operaciones de procesamiento con una cámara digital, se requiere conocer un modelo matemático que se corresponda con el funcionamiento, la sensibilidad del sensor y su relación en este caso con respecto al mundo real. El conocimiento de este comportamiento es para lo que se realiza el proceso de calibración, que se comentará un poco más adelante.

Para hablar del modelado de cámaras, hay que contar el principio de una cámara simplificada que es conocido como el modelo *pinhole*. Este modelo se realiza a partir del funcionamiento de la cámara más simple que se pueda diseñar, conocida como cámara oscura o estenopeica (su adaptación en tamaño manejable, antiguamente se trabajaba sobre una sala entera). La cámara estenopeica está compuesta únicamente por una caja con una película fotográfica en su interior. Esta película está aislada del exterior, salvo por un pequeño agujero en la pared contraria, a través del cual entrarán los fotones que emita la imagen que se tenga delante. El agujero inicialmente está cubierto, y la fotografía se realiza descubriéndolo y volviéndolo a tapar. La duración de este proceso determina el tiempo de exposición, que, junto con el tamaño de ese agujero (apertura), sirve para controlar la cantidad de luz y de rayos que inciden sobre la película respectivamente, evitando así que se sature, que haya falta de nitidez (agujero muy grande) o problemas de difracción (agujero muy pequeño).

Al ser estos rayos los únicos que podrán entrar por el agujero, por su orientación, se grabarán en la película de forma invertida, tanto vertical como horizontalmente según el siguiente diagrama.

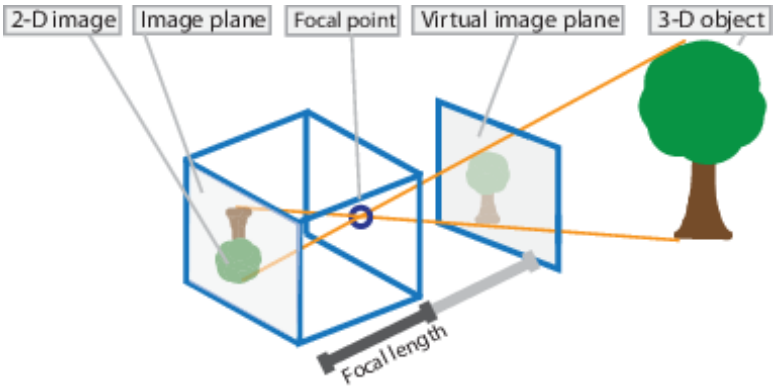


Figura 2.3: Esquema *pinhole*

El resultado de la imagen obtenida es lo que nos lleva a describir la variable de distancia focal, que se corresponde con la longitud desde la película hasta el agujero. La parte de imagen que se obtenga con nitidez, al ser un sistema sin lente, depende de este factor, de forma que, en función de la apertura, esa parte de la imagen podrá abarcar más o menos profundidad.

Actualmente, hay dos tipos de tecnologías en el mercado que harían las veces de la película previamente mencionada, los dispositivos de carga acoplada (condensadores) o *CCD* por sus siglas en inglés y los de tecnología de semiconductores o *CMOS*. Ambas se componen de una matriz de sensores, cuya cantidad determinará los píxeles del dispositivo, en los que se cargará la imagen por efecto fotoeléctrico. La principal diferencia entre ambos será durante el proceso de conversión de la señal luminosa a digital. Mientras que los *CCD* desplazan la señal recorriendo una fila para llegar hasta el conversor analógico-digital 2.4, en la tecnología *CMOS* al trabajar con transistores, requiere menos tamaño y puede permitirse un conversor por cada fotodiodo.

Por una parte, al permitir acceder a cada uno por separado, el *CMOS* resuelve los problemas de los *CCD* conocidos como *blooming*, que añadiría un *offset* de carga a los sensores vecinos en caso de saturación y *smearing*, que extiende el efecto del *blooming* a toda la columna del registro de desplazamiento, si el desbordamiento es muy grande. También reduce el consumo y puede mejorar la velocidad de transferencia si trabajamos con regiones de interés o ROI's. Por otra, en cambio, esa misma reducción implica una pérdida de calidad al perder superficie de captación de luz.

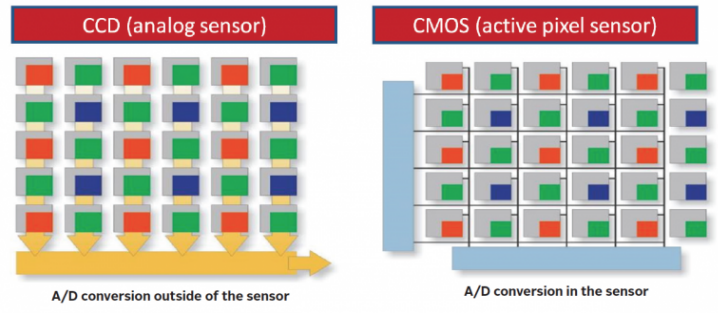


Figura 2.4: Tipos de conversión de señal

Volviendo al ejemplo de la cámara oscura, actualizando la construcción, ahora tendremos un conjunto de sensores en vez de la película y una lente en vez de agujero. Dicha lente concentra los rayos de luz, lo que permite aumentar el agujero para obtener una imagen más iluminada sin perder nitidez. Por contra, introducirá una distorsión en función de sus características, cuyo efecto habrá que añadir al modelo comentado.

Con estos dos cambios, se hace patente la necesidad de determinar cómo se corresponde la parte de imagen recogida en cada sensor con la imagen real. Aquí es donde entran los sistemas de coordenadas y los parámetros que vamos a tener en cuenta para la calibración.

2.4.1.1 Calibración

El proceso de calibración consiste en la comparación de un sistema de medida cuya exactitud es desconocida, con un sistema estándar para asegurar que funciona dentro de unas especificaciones y/o eliminar errores sistemáticos. [5]

En este caso la comparación y conversión se debe realizar entre tres sistemas de coordenadas distintos, las del mundo real, las de la cámara en tres dimensiones, es decir, su posición y las de la cámara en dos dimensiones o píxeles, que corresponden a la matriz de sensores.

Como podemos ver en la figura 2.3 se puede hablar de una imagen virtual, previa a la lente y a la distancia focal, que equivaldría a la imagen obtenida en la orientación correcta.

De este modo, tenemos la siguiente imagen 2.5, extraída de la descripción del funcionamiento del algoritmo de calibración de *OpenCV* [6], en la que ya podemos observar una relación entre las coordenadas del punto P (subíndice w de *world*); las coordenadas (u,v), de la imagen virtual equivalente a la matriz de píxeles y las coordenadas Fc (*Frame camera*) correspondientes a la posición de la cámara.

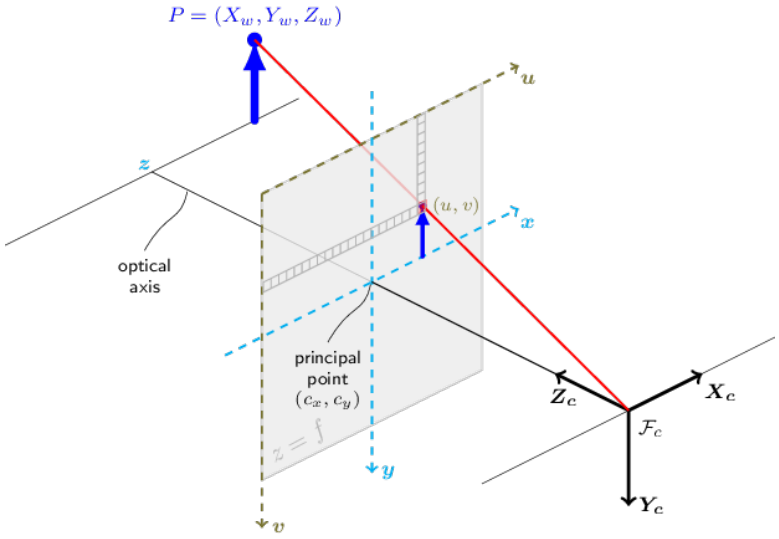


Figura 2.5: Relación de coordenadas

Esta imagen, además, indica la orientación de las coordenadas Fc que utiliza para su algoritmo.

Para el caso de nuestra cámara realsense ZR300 podemos ver cómo se corresponden con ellas, adjudicadas en la función:

`virtual core::status rs::object_recognition::or_configuration_interface::enable_object_center_estimation (const bool enable)` [7]

The coordinates are provided with respect to the color camera coordinate system:

x: right y: down z: forward origin: the color camera's principal point

Partiendo de las coordenadas del mundo real para un objeto, tenemos este esquema en el que se identifica la relación entre ellas.

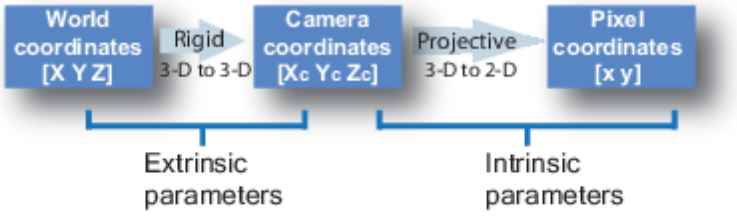


Figura 2.6: Coordenadas y parámetros

El modelo de calibración se compone de tres conjuntos de parámetros que debemos tener en cuenta: extrínsecos, intrínsecos y de distorsión. Los primeros son una matriz de rotación y otra de traslación, con las que se indicará la posición relativa de la cámara, respecto al objeto de control o viceversa.
Los parámetros intrínsecos se conforman en una matriz 3x3 estructurada de esta forma:

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

Para la que tenemos por un lado **fx** y **fy** correspondiendo a la distancia focal, y por otro, **cx** y **cy** que componen las coordenadas en píxeles (o coordenadas (u,v)) del punto principal o centro de proyección.

Para un caso sin lente, es decir sin distorsión, se obtendrían de la siguiente manera:

$u = \frac{x_c}{z_c} + c_x * f_x$

$v = \frac{y_c}{z_c} + c_y * f_y$

Por último tenemos la distorsión. Cuando hablamos de la distorsión hay que referirse al modelado de la lente de la cámara. Modelar esta lente implica fijarse en dos tipos de distorsión: radial y tangencial.

La distorsión radial consiste en el efecto de modificación de la imagen por la curvatura de la lente, de forma que, según si ésta es cóncava o convexa, la proporción de tamaño de la imagen obtenida será distinta en los extremos y en el centro del siguiente modo.

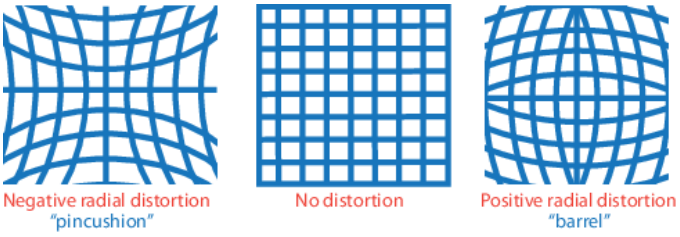


Figura 2.7: A la izquierda estaría el caso de lente cóncava o efecto cojín. A la derecha, el efecto barril correspondiente a una lente convexa.

La distorsión tangencial hace referencia al resultado de tratar de obtener una imagen con un sensor en un plano distinto al de la lente, que serían los casos de la siguiente imagen.

Figura 2.8: Efectos para distintos planos de distorsión.

El modelado de estos dos casos está recogido en la matriz de distorsión. Para el caso de la cámara de este trabajo se compone de 5 valores, tres correspondientes al efecto radial (k_1, k_2 y k_3) y otros dos para el tangencial (t_1 y t_2) .

Finalmente, entre todos, se realiza la proyección de la imagen espacial a la imagen plana, para este caso particular, según la función *rs_project_point_to_pixel* [8] siguiendo el modelo correspondiente, una modificación del modelo de lente Brown-Conrady. Siendo `point[i]` la matriz de valores espaciales x, y, z, respectivamente; `coeff[0], coeff[1], y coeff[4]`, los valores de los parámetros radiales; `coeff[2] y coeff[3]`, los parámetros tangenciales y `pixel [i]`, la matriz donde se guardarán los valores de las coordenadas proyectadas.

```
float x = point[0] / point[2], y = point[1] / point[2];
if(intrin->model == RS_DISTORTION_MODIFIED_BROWN_CONRADY)
{ float r2 = x*x + y*y;
float f = 1 + intrin->coeffs[0]*r2 + intrin->coeffs[1]*r2*r2 +
intrin->coeffs[4]*r2*r2*r2;
x *= f;
y *= f;
float dx = x + 2*intrin->coeffs[2]*x*y + intrin->coeffs[3]*(r2 + 2*x*x);
float dy = y + 2*intrin->coeffs[3]*x*y + intrin->coeffs[2]*(r2 + 2*y*y);
x = dx;
y = dy;}
pixel[0] = x * intrin->fx + intrin->cx;
pixel[1] = y * intrin->fy + intrin->cy;
```

Más adelante, el apartado 3.3 tratará sobre cómo se obtienen estos párametros en un caso real.

2.5 SLAM

Dejando a un lado los dispositivos, vamos a hablar sobre cómo empezaron a desarrollarse el apartado de software y las bases del SLAM. Inicialmente, y según describen Hugh Durrant-Whyte y Tim Bailey [1], se planteaba que pudiera haber un problema, porque la cantidad necesaria de marcas dispuestas para el mapeado, junto con la dirección y orientación del vehículo *pose*, tendrían que ser actualizados a cada observación, acarreando una cantidad acumulada de errores que conducirían al desplazamiento errático sin vistas de corrección y a su posterior pérdida de localización.

Esto produjo que los trabajos se decantaran bien hacia la navegación, o bien hacia la construcción de mapas. Por ello, derivó en enfrentarse a aproximaciones de las tareas, hasta llegar a considerar centrarse únicamente en intentar minimizar la correlación de las marcas del mapeado, para reducir así la carga computacional trabajando con series de marcas inconexas.

Más tarde, se observó que, trabajando sobre estas correlaciones, podrían convertir ambas cuestiones en un solo problema de estimación, con lo que podían modificar la tendencia de aparición de errores. Estas correlaciones acabaron resultando determinantes para conseguir una convergencia, dado que aumentarlas se correspondía con mejores resultados.

Con ello, en 1999 se realizó la primera sesión de *SLAM* y los algoritmos basados en filtros de Kalman y los modelos probabilísticos empezaron a compartir resultados.

Actualmente, el problema de *SLAM* quedaría estructurado, desde la parte de localización, como la resolución del problema local y/o global y, desde la parte de mapeado, bajo las opciones de determinar mapas, o bien métricos, o bien topológicos (mediante zonas características).

A la hora de la localización, se hace referencia a las necesidades requeridas en función del tipo de variables que se van a utilizar para afrontarlo. Para el caso del problema local hacen falta unas condiciones iniciales para que el trabajo con odometría sea efectivo, en cambio, para el global, trabajando por ejemplo con señales *WiFi* o con *GPS*, tratará de encontrarse independientemente de donde comience. Los programas que se van a estudiar son capaces de resolver ambos.

2.5.1 Enfoques

Desde entonces se han establecido clasificaciones de ellos desde distintos frentes.

A grandes rasgos, este esquema extraído del trabajo de Niko Sünderhauf [9] serían las ramas más destacadas.

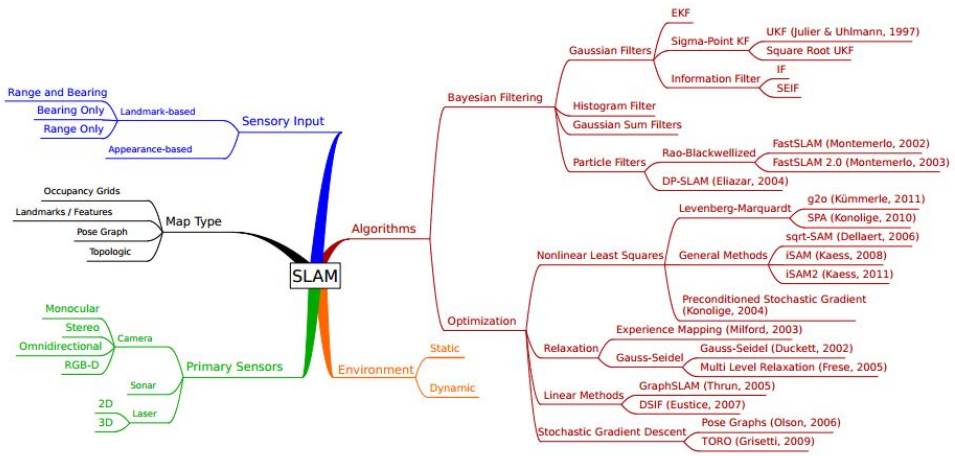


Figura 2.9: Esquema SLAM

Por la parte de la implementación en *hardware* y dependiendo del entorno tenemos:

Según la autonomía del vehículo podemos distinguir entre activo, si su desplazamiento es independiente, o pasivo, si es teledirigido por un operador.

Según el tipo de escenario en el que se va a trabajar, podemos distinguir entre estático o dinámico, dependiendo de la ausencia o existencia de elementos móviles. También es necesario identificar entre *SLAM* para interiores o para exteriores, ya que esto delimita el tipo de sensores que puedes utilizar.

Finalmente, quedaría la distinción de si el *SLAM* se realizará de forma individual o por medio de un enjambre de robots, siendo este último un método para procesar áreas más grandes o repartir tareas.

Por otra parte, para el caso del *software* debemos hablar de otros aspectos.

Según la cantidad de posiciones utilizadas en el proceso, pueden ser o *Full SLAM*, en el que se estima la trayectoria completa u *Online SLAM* que trabaja únicamente con la posición actual de forma sucesiva, ambos con estimación del mapa.

Finalmente, la división más importante que tenemos que hacer en este apartado, aparece en la perspectiva del algoritmo de localización.

Como puntualización, la forma más básica de localización es determinística haciendo referencia al proceso de ubicación sobre un mapa respecto a marcas preestablecidas o bien para el cálculo del mismo, donde no se tiene en cuenta las posibles incertidumbres de los modelos utilizados.

Dentro de los procesos estocásticos, se trabaja con funciones de densidad o distribuciones de probabilidad del estado (posición y orientación) o posición para tratar de reducir, cuanto sea posible, las incertidumbres omitidas en el caso anterior.

En este marco podemos distinguir entre técnicas de filtrado y técnicas de optimización, también llamadas de suavizado.

Cuando hablamos de filtrado la herramienta matemática principal para afrontar este estilo se conoce como filtro de Bayes o bayesiano, a partir de él derivan muchos algoritmos de localización. La razón de su uso es porque la variable que se necesita conocer, corresponde a lo que se conoce como problema inverso. Esto es un tipo de problema, para el cual, tratamos de averiguar bajo qué estados es más probable que se reproduzcan los datos de la medida que nosotros ya hemos tomado.

Una breve descripción del teorema de Bayes:

$$P(B|A)_1 = \frac{P(A|B)_2 * P(B)_3}{P(A)_4}$$

Se compone de dos probabilidades condicionadas: la probabilidad a posteriori (1), nuestra incógnita, y la verosimilitud (2), que es la que relaciona una hipótesis B, con los datos que podemos obtener, A. También tenemos la probabilidad a priori (3), que indica el conocimiento que podemos tener acerca de esa hipótesis y finalmente la evidencia (4), que indica todos los posibles resultados mutuamente excluyentes que se podrían dar.

Esta evidencia se obtiene del teorema de probabilidad total.

$$p(A) = \sum_{i=1}^{n} p(B_i) * p(A|B_i)$$

No me extenderé en él ya que, al no estar en función del estado, según la aplicación, o se traslada a un factor de normalización, o directamente se descarta.

Con su aplicación recursiva, la probabilidad a posteriori (1) de una iteración, pasará a ser la verosimilitud (2) de la siguiente, mejorando cada vez más su estimación. Desde aquí, si se cumple la propiedad de Markov, que establece que, si se conoce el estado actual, para calcular la nueva iteración no dependemos de las mediciones anteriores, podemos simplificar los cálculos.

Volviendo al proceso de filtrado, el filtro de Bayes se caracteriza por trabajar con dos modelos para incluir la repercusión de las incertidumbres del movimiento de un sistema y la correspondiente covarianza entre ellas. El modelo de actuación añade el ruido correspondiente a las acciones y el de percepción aporta las incertidumbres de las observaciones. Su aplicación puede darse en lo que se conoce como forma compacta:

$Bel(s_t) = \eta * p(o_t|s_t) \int p(s_t|s_{t-1}, a_{t-1}) * Bel(s_{t-1}) * ds_{t-1}$

Donde nos encontramos la creencia $Bel(s_t)$ por su nombre en inglés, que es la probabilidad a posteriori del teorema y η que es un factor de normalización para que los límites de la distribución esten entre 0 y 1.

O bien por pasos, en los que encontraríamos la etapa de predicción con el modelo de actuación y la etapa de corrección o estimación con el modelo de percepción.

Con esta estructura obtenemos la creencia, que si bien nos proporciona la información que queríamos, si el estado es continuo, la cantidad de información es inabarcable. Por ello, en vez de calcularse la distribución posterior completa se suele trabajar o bien discretizando los estados, o con lo que se llaman estimadores puntuales.

Los estimadores puntuales, en vez de darnos la información completa, siguiendo el artículo [10] podemos encontrarnos con distintos criterios para encontrar un filtro óptimo, como el de búsqueda de máximo a posteriori o MAP, con el que obtendríamos la moda de la distribución, o su variante particularizada a una distribución a priori uniforme, la máxima verosimilitud, por su nombre en ingles ML.

A partir de el MAP, en este mismo artículo tendríamos el desarrollo para llegar hasta lo que se conoce como filtro de Kalman. Dicho artículo también incluye amplia información sobre lo que se conocen como filtros de partículas que comentaré brevemente a continuación.

2.5.1.1 Filtro de Kalman

Este tipo de estimador se basa en unas condiciones de trabajo para un sistema lineal, cuadrático, para el que las variables como los ruidos o la creencia se modelan como funciones gausianas.

Como ventaja, tiene que nos permite trabajar en el espacio de estados, con lo que facilita usar álgebra matricial para su control. Partiendo de las dos etapas mencionadas previamente en la definición de Bayes, tenemos en primer lugar la etapa de predicción. En ella, incorporando las ecuaciones de cinemática de posición y velocidad al espacio de estados, se obtiene la función de transición de estado:

$\hat{x}_k = F * \hat{x}_{k-1} + B * u_k$

Para la siguiente etapa, se aplica una corrección al resultado obtenido, por medio de una ganancia que cuantifica el peso que se le aplica al error de la medida predicha en comparación con la medida tomada por el sensor. Si la ganancia es muy pequeña, la nueva distribución a posteriori sería más parecida al comportamiento establecido por la predicción y si es grande, tendría más dependencia de los valores medidos. Esta ganancia viene determinada por la covarianza del error a priori y la covarianza del error observado.

Dicha ganancia también interviene en la actualización para la covarianza de la nueva distribución a posteriori.

Este tipo de estimación tiene sus limitaciones: Tiene que utilizarse a partir de una posición conocida y unas marcas reconocibles, para poderse modelar como distribuciones gaussianas, y sólo puede resolver el problema de localización local. Por otra parte, al realizar estimaciones en base a las marcas, el tamaño de las matrices con las que trabaja crecen exponencialmente y no se pueden manejar más que unos pocos

cientos de marcas. Además, los modelos tienen que ser lineales, de lo contrario, habría que linealizarlos por medio de series de Taylor, en lo que se conoce como filtro extendido de Kalman o EKF.

2.5.1.2 Filtro de partículas

El siguiente método de localización sigue la misma idea que el filtro de Kalman. Partes desde un sistema con un estado que desconoces, evolucionando en el tiempo en función de las ecuaciones de movimiento de los modelos y el ruido. Por medio de filtros bayesianos, tienes una etapa de estimación y otra de predicción con las que tratas de inferir el estado actual a posteriori, es decir obteniendo una creencia.

En este caso, la creencia se discretiza en las muestras conocidas como partículas para poder hacer la estimación tratable computacionalmente. Las partículas se componen de posición y peso e inicialmente parten del mismo valor o factor de importancia.

Para comenzar, se escoge un estado de referencia y se toman muestras aleatorias a las que se le asignan distintos pesos en función de la similitud al mismo.

Contando ya con la etapa de predicción, a partir de los pesos imprecisos iniciales, se seleccionan partículas en función del peso decreciente, se realiza un movimiento y se obtienen nuevas muestras.

Después, en la etapa de estimación, se hace una nueva medida y comparándola con las estimaciones, se recalculan los pesos de las partículas.

Con este sistema es capaz de resolver tanto la localización local, como la global. Trabajar con partículas, tiene la ventaja de que puede concentrar la capacidad de cálculo en las zonas con más probabilidad de estar ubicado, pudiendo ademas limitar los recursos por medio de la cantidad de partículas utilizadas. Como siempre, esto supone un compromiso. Después de cada remuestreo, si no se trabaja con suficientes partículas, puede provocar que se pierda y no vuelva a poder ubicarse. Esto demuestra que no puede resolver el problema del secuestro con el algoritmo clásico.

2.5.1.3 SLAM Visual

Llegados a este punto tenemos el SLAM visual. Si bien hay opciones para trabajar con algoritmos de filtrado para SLAM visual, por ejemplo el algoritmo de condensación desde el filtro de partículas, desde el articulo *Visual SLAM Why filter?* [11] han evaluado que, a partir de un aumento de *frames* ya no mejora la robustez del algoritmo, en cambio, sí que hay mejora en precisión, a mayor cantidad de características disponibles. Dado que los órdenes de procesamiento con el aumento de características son de mayor crecimiento para el filtrado, que para el *bundle adjustment* la opción de optimización se comporta mejor para el SLAM visual. De todas formas, este es un artículo antiguo sobre un estudio basado en condiciones ideales, con lo que sirve únicamente como recomendación, mas que como resultado determinante.

Como tal, no existe un SLAM visual único, si no distintas soluciones para resolver un patrón más o menos definido de cuestiones a partir del esquema característico que es *frontend* u odometría visual, *backend* o procesamiento de datos y mapeado.
A saber:

- Cómo conocer la posición de inicio (asociación de datos).
- Cómo detectar el movimiento.
- Cómo aprovechar ese recorrido para obtener un mapa y poder ubicarse en futuros desplazamientos.
- Cómo reconocer cuando has vuelto a pasar por un mismo sitio. Cierre de lazos.

En principio, el primer paso de la localización se realiza mediante odometría visual, mencionada en el apartado 2.2. Profundizando sobre ella, tendríamos la detección de características o *features* de la imagen. Estas características son la suma de puntos reconocibles o *keypoints* de una imagen y sus descriptores, vectores asociados a esos puntos.

Con una cámara monocular, ésto por sí solo no sería suficiente, ya que se trabajaría sobre las detecciones en una imagen plana, sin información de distancias. Desde el punto de vista monocular, requeriría un primer desplazamiento o una posición predeterminada. En cambio en cámaras con detección de profundidad (bien RGB-D o estereoscópicas) este punto queda resuelto.

El siguiente paso, sería establecer la relación entre las distintas características de cada imagen a medida que se avanza. En cada *frame* obtenido, las características se mantendrán o se moverán en el mismo, de forma proporcional al movimiento de la cámara. Por medio de una comparación de la distancia euclidiana entre los vectores asociados a dichos puntos, es una de las formas de determinar si corresponden al mismo en ambas imágenes, y por tanto, se procede a la asociación entre ambos. Esta comparación no es una tarea trivial. Para que coincidan ambos puntos en las distintas imágenes hay muchas estrategias de detección de características encaminadas a problemas concretos. Una buena característica debería ser invariante al escalado; cuando sacamos dos imágenes de un mismo objeto a distintas distancias, los objetos en el plano se perciben con distinto tamaño. Es necesario que se reconozcan las características que corresponden al mismo objeto. Otra propiedad a tener en cuenta, sería la invarianza a la rotación, ésto permitiría reconocer un mismo objeto, visto desde distintos ángulos. También se ha estudiado la invarianza al ruido o a la iluminación, para imágenes con mal procesamiento o distinta intensidad.

Como ejemplos de ellas, las estrategias más conocidas son SIFT (*Scale Invariant Feature Transform*), la comparación euclidiana antes mencionada, y SURF (*Speed-up Robust Feature*) que, aunque son muy fiables, requieren mucho tiempo de cálculo. Los métodos de comparación desarrollados a lo largo de los años han derivado en una larga lista de opciones y mezclas entre ellos, como el caso del ORB (***O****riented FAST (Features from Accelerated Segment Test)* ***R****otated* ***B****RIEF (Binary Robust Independent Elementary Features)*), un poco más manejable al trabajar con vectores binarios, o el detector *Good features to track* o método Shi-Tomasi, que es una adaptación del método de detección de esquinas de Harris. Para una breve descripción y comparación entre ellos se realizó este articulo [12] y para más información sobre cómo actúa cada uno se puede empezar por acudir a las descripciones utilizadas por OpenCV. [13]

Una vez establecidas posibles asociaciones entre imágenes, otra cuestión sería la estimación de movimiento a partir de ellas. Según el formato de los datos adquiridos se puede trabajar sobre imágenes de dos dimensiones, realizando un traspaso de información desde tres a dos dimensiones cuando tienes datos de profundidad y procesas datos sobre imágenes o trabajar desde tres dimensiones y tienes una representación volumétrica, como pueden ser con nubes de puntos o voxels.

El caso de las cámaras RGB-D es el paso de tres a dos dimensiones, en lo que se suele conocer como problema de perspectiva desde n puntos o por sus siglas en inglés `PnP`. Consiste en minimizar el error entre las características encontradas en el espacio y los puntos asociados en el plano, también conocido como error de reproyección. En ocasiones suele utilizarse después de aplicar el algoritmo `RANSAC` para eliminar todos los datos atípicos posibles. Es una técnica de mínimos cuadrados iterativa, por lo que cada nueva aplicación mejorará el resultado, a costa del tiempo empleado.

El último aspecto a considerar en el SLAM visual es el cierre de lazos. Este es uno de los puntos más importantes para obtener un SLAM preciso. Los errores sobre el desplazamiento recorrido son acumulativos y debido a que toda odometría puede ir teniendo pequeños desfases entre cada posición que estima y la posición real en la que ese encuentra, el mapa final puede tener una desviación que haga inviable la posterior localización. El cierre de lazo, al reconocer una posición por la que ya ha pasado permite

calcular el error con el que llega, respecto a la medición anterior, y con ello restaurar hacia atrás cada uno de los desfases previos.

Esto se puede plantear nuevamente como una técnica de matching, pero tiene el problema de no saber qué imagen va a tener que compararse y cuanto más grande sea el recorrido, más lento será realizar las comparaciones.

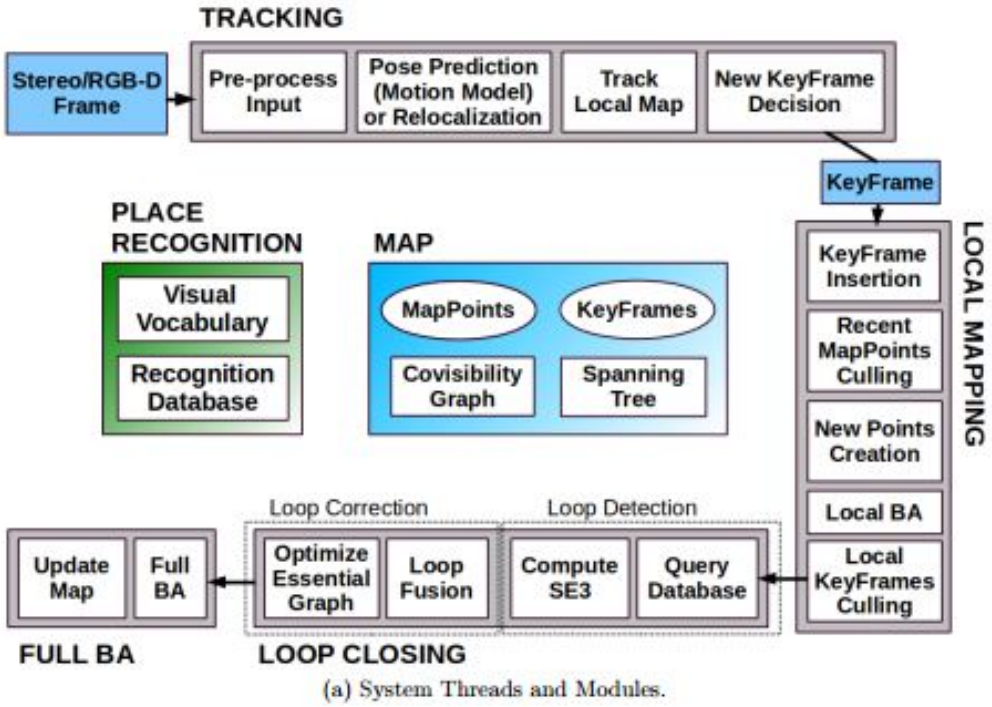
Por ello, se suele emplear una técnica de división en clústers de imágenes conocida como *bag of words*. A un conjunto de características que formen algo identificable de una imagen, una montaña, una ventana o un vaso, se le llama palabra y varias de estas, componen un diccionario o vocabulario. A partir de este diccionario de palabras se describe una imagen, con lo que se agrupan en esos clústers las que tengan las mismas palabras, para luego buscarlas más fácilmente.

Finalmente llegamos a las librerías concretas que se comparan en este trabajo.

2.6 ORB-SLAM2

Empezando por este proyecto de la Universidad de Zaragoza [14], es un sistema de SLAM basado en obtención de características de la imagen, para construir un mapa en 3 dimensiones disperso, en paralelo a un grafo de *keyframes* llamado grafo de covisibilidad. Una simplificación de este grafo, llamado grafo esencial, permitirá un refinamiento por medio del optimizador G2o[15], para terminar usando *bundle adjustement* sobre el mapa completo, lo que le da su carácter de *Full SLAM*.

Su forma de trabajar se basa en tres hilos de algoritmos corriendo en paralelo.



(a) System Threads and Modules.

Figura 2.10: Estructura ORB-SLAM2

Esta subdivisión se debe a que tanto el proceso de obtención de características, como el de refinamiento por *bundle adjustement*, suelen ser ambos muy costosos computacionalmente y, procesarlos en un mismo ciclo, impediría obtener una ejecución en tiempo real para el sistema de rastreo o *tracking*.

Concretamente, el gran coste de la extracción de características suele venir de utilizar los métodos SURF o SIFT previamente mencionados en 2.5.1.3. Uno de los puntos clave de este sistema es trabajar con características ORB, que permiten un procesamiento y un emparejamiento más rápido al usar descriptores de tipo binario. Esto permite que, en vez de procesar vectores de los descriptores, la comparación se realice por medio de la distancia de Hamming, mucho menos costosa al poder efectuarse con una simple operación XOR. Como se describe en el apartado sobre el descriptor BRIEF del trabajo de Niko Sünderhauf. [9]

Por otra parte, está el *bundle adjustement*. Consiste en encontrar una solución óptima conjunta en la reconstrucción visual en 3D, a partir de las características extraídas y los parámetros indicados de calibración o estimados de posición de la cámara. El hecho de que sea una solución óptima, implica que los resultados se obtienen al minimizar una función de coste, que evalúe un modelo apropiado del error.

Volviendo a los tres hilos de ejecución paralelos, en cada uno de ellos y de nuevo, tal como se detalla en su artículo de presentación, [14], existe una versión reducida del algoritmo *bundle adjustement* orientada para procesos más concretos.

El primero, de rastreo, trata de optimizar la posición de la cámara, lo que se conoce como *Motion-only Bundle Adjustement*. El segundo hilo realiza un ajuste, principalmente, sobre los *keyframes* y puntos que se encuentran accesibles dentro de la ventana de trabajo del mapeado local, apoyándose en los restantes como información para la función de coste. Finalmente, después de completar el cierre de lazo con el tercer hilo, realiza un equivalente del refinamiento local, pero aplicado a todos los *keyframes* y puntos registrados.

Al margen del refinamiento, esta segunda versión se diseñó pensando en aumentar las capacidades de la versión previa, trabajando con cámaras estereoscópicas, pero por medio de una adaptación de la información de profundidad recibida, añade la posibilidad de trabajar del mismo modo con cámaras RGB-D en el paso de preprocesamiento de la entrada.

Avanzando en el esquema de la figura 2.10, el punto de conexión entre el hilo de rastreo y el de mapeado es la introducción de nuevos *keyframes*, al ser, junto con los puntos del mapa, las unidades que conforman el grafo de covisibilidad.

Estos *keyframes* aportarán robustez al rastreo, de forma que actúen como punto de relocalización en caso de pérdida. Tienen una serie condiciones heredadas de la anterior etapa del algoritmo monocular, con el propósito de asegurar cambios visuales mínimos entre los últimos *keyframes*, que permitan un buen rastreo o que detengan el *bundle adjustement* local si se ha detectado mucha información desde que empezó, para actualizarlo.

Además, añadir la profundidad en esta nueva iteración ha permitido incluir una condición, para afinar el proceso de desplazamiento. Aprovechando la distinción entre puntos lejanos y cercanos, para escenas en que la parte lejana de la imagen es mayor que la cercana, si la cantidad de puntos cercanos entra dentro de un umbral mínimo configurable, introduce un nuevo *keyframe*.

Por último, para el cierre de lazo, tiene diseñado un módulo de reconocimiento de lugar aplicando una variación del algoritmo de bolsa de palabras, mencionado en el apartado anterior. En este caso conocido como DBOW2, con el que tiene a Juan Tardós en común en el proceso de desarrollo.

2.7 RTAB-Map

El nombre de este corresponde a las siglas en inglés de mapeado basado en apariencia en tiempo real. Hace referencia al sistema de reconocimiento de su posición mediante odometría y un algoritmo de bolsa de palabras. Propone un tipo de SLAM *online* basado en optimización de grafos en el que muestra un tipo de mapa denso generalmente por nube de puntos. Es un SLAM muy completo con una interfaz que permite trabajar tanto con cámaras estereoscópicas, como con cámaras de profundidad, así como con LIDAR de dos o tres dimensiones.

El *pipeline* comenzaría por la detección de características, por medio del método de ShiTomasi, después se realiza el matching por medio de un algoritmo de el vecino más cercano `FLANN`. Con estos emparejamientos, se realiza la estimación de movimiento resolviendo el problema de perspectiva de n puntos utilizando RANSAC como está implementado en OpenCV. Despues se realiza una optimización del grafo que se obtiene del paso anterior por medio del algoritmo G2o basado en la optimización no lineal de Levenberg-Marquadt. Desde aquí se actualiza el estado o pose. Este estado se publica al hilo de odometría, se transfiere a la transformada y se evalúa a ver si se convierte en un *Key frame* en función de si tiene un mínimo de correspondencias con otros *frames*.

Su parte más característica es el sistema de empleo de memoria, con esta estructura extraída de su artículo [16]

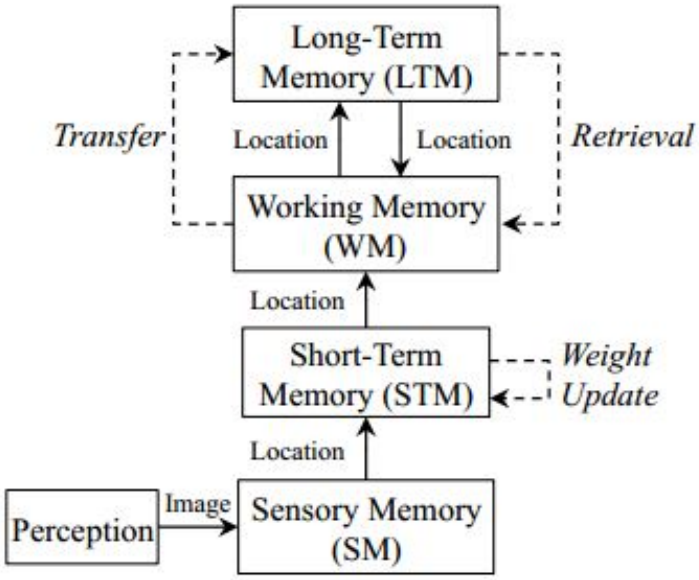


Figura 2.11: Gestión de memoria de RTAB-Map

Mantener una memoria de trabajo activa, junto con una memoria a largo plazo que solo se comprueba una vez se hayan detectado cierres de lazos es lo que permite operar en tiempo real, mientras trabaja con un tipo de mapa denso. La distinción entre memorias se establece por las palabras que se mantienen en el vocabulario, en la memoria de trabajo, o se eliminan para pasar a formar parte de la memoria a largo plazo. El sistema sería el siguiente. Partimos de adquirir una imagen. Sobre ella se localizan una cantidad determinada de características SURF en un tiempo limitado y se extraen para preparar un índice de descriptores procesados por medio del algoritmo de vecino más cercano `FLANN`. Con este índice los cuantifica para formar un vocabulario. Con respecto a ese vocabulario, a una imagen se le asigna una

firma y un peso, convirtiéndose en una localización. Para reducir los posibles candidatos para un cierre de lazo, se comprueba si se han podido extraer suficientes características de una imagen. Si está por debajo de un umbral escogido, como podría ocurrir con una pared de color o textura planas, se utiliza únicamente como parte del mapa, pero no se incluye para la comparación. Si es correcta se enlaza con la localización previa y se agrega a la memoria de corto plazo. En ella se le modifica el peso en relación a las palabras o características de la localización anterior.

El tamaño de esta memoria a corto plazo está controlada por la velocidad de desplazamiento del robot y la tasa de *frames* que va recibiendo. Una vez que alcanza un valor máximo, la localización más antigua se traspasa a la memoria de trabajo. En esta memoria de trabajo se evalúa la nueva localización traspasada por medio de un filtro de Bayes, comparándola con las otras que están actualmente en ella. Si se produce un cierre de lazo, ambas son enlazadas, se aumenta el peso de la nueva localización y se reinicia el peso de la antigua. Por otra parte, la gestión de la memoria de trabajo queda determinada por un proceso de recuperación y de transferencia con la memoria a largo plazo. Por un lado, la recuperación de localizaciones se activa después de la detección de un cierre de lazo y se comparan las firmas para enlazar las que tengan palabras similares. Las palabras de las nuevas localizaciones que no coincidan con ninguna existente en ese momento se añaden al vocabulario. En cambio, la transferencia a la memoria a largo plazo se realiza en el momento en el que el procesamiento de una imagen alcanza un tiempo máximo. En ese momento, se evalúan las localizaciones con el peso más bajo que lleven más tiempo y, junto con las palabras que hayan dejado de estar presentes en la memoria de trabajo, se traspasan a la memoria a largo plazo esperando a un nuevo cierre de lazo.

Los enlaces que se forman entre las localizaciones de la memoria de trabajo, bien por estar incluidas en la memoria a corto plazo, o bien por ser un cierre de lazo, son los límites del grafo sobre el que se va a establecer el mapa. Este mapa organiza la información obtenida de la cámara en celdas de ocupación, bien de dos dimensiones, haciendo la proyección desde la nube de puntos de la información de profundidad o en tres dimensiones en nube de puntos, o bien en Octomap, en función de los párametros activados.

Este mapa está estructurado en torno al grafo de *keyframes*, de forma que, cuando se produzcan correcciones al cerrar un lazo, se corrija la nube de puntos al mismo tiempo. Este grafo a su vez, permite un sistema de navegación y planificación topológico estableciendo las trayectorias a través de los *keyframes* asegurando así que sean rutas despejadas, en el caso de estar trabajando en entornos estáticos.

Capítulo 3

Desarrollo

3.1 Componentes

En esta sección se hará una ligera descripción de los dispositivos a los que se hará referencia a lo largo del capítulo.

- Portátil HP modelo OMEN 15-ce020ns

  El principal ordenador de trabajo es un ordenador con un procesador de 7ª generación Intel Core i7-7700HQ a 2,8 GHz.

  Consta de una tarjeta gráfica dedicada NVIDIA GeForce GTX 1050 y una integrada Intel® HD Graphics 630.

  Memoria SDRAM DDR4 de 16384 MB.

  Una unidad de estado sólido de 256 GB, junto con un disco duro de 1TB componen la memoria secundaria.

  Y un monitor IPS LED de alta definición de 15,6" a 60Hz.

- UDOO modelo x86 Ultra

  Considerada en el grupo de las *Single Board Computers*, como puedan ser las Raspberry Pi, está claramente a un nivel superior a éstas y aún permanece por encima de la nueva de cuarta generación.

  Dispone de:

  - un procesador Intel Pentium N3710
  - Una unidad de procesamiento gráfica Intel HD Graphics 405
  - 8 GB de memoria RAM DDR3L *DUAL CHANNEL*
  - 32 GB de memoria Flash eMMC (*embedded MultiMediaCard*)
  - Un microprocesador Intel Quark SE core 32 MHz plus 32-bit ARC core 32MHz junto con un *pinout* compatible con Arduino101 y la mayoría de sus módulos.[17]

- Cámara ZR300 [18]

  Este dispositivo se diseñó como cámara de profundidad experimental a partir del modelo R200 ya en comercialización.

Se compone de 4 sensores de visión: una cámara RGB para el color, dos de infrarrojos para el procesado de profundidad mediante estereoscopía por medio de una luz estructurada, y un sensor de ojo de pez.

El cálculo de profundidad funciona de la siguiente manera. Por medio de un proyector láser infrarrojo incorporado en la cámara se ilumina un área con una malla. Dicha malla se modificará en función de la superficie sobre la que este proyectada. Estas variaciones se comprueban desde las cámaras infrarrojas y se comparan las diferencias entre cada pixel de ambos flujos de imágenes. La malla se modificará más cuanto mas cerca se encuentre la superficie, dentro del rango de trabajo.

Con esta cámara buscaron hacer un producto de desarrollo orientado al reconocimiento de objetos, al seguimiento de personas y al SLAM. Para ello, y como innovación respecto al anterior modelo, se le añadió un módulo IMU, compuesto de un acelerómetro que muestrea a 250 Hz con un rango de +-4G y un giroscopio que trabaja a 200 Hz con el rango de 1000 deg.

Otra aportación respecto al modelo anterior es un sensor de visión de ojo de pez, que permite ampliar el campo de visión de la cámara. Para aumentar las capacidades se le han incorporado dos conectores GPIO para incluir sensores externos. Y, por último, un microcontrolador para proporcionar marcas de tiempo de alta frecuencia, para poder sincronizar entre los distintos flujos de sensores.

3.2 Desarrollo del sistema de experimentación

En este apartado voy a detallar los pasos utilizados hasta la instalación de los distintos productos (o indicar las opciones escogidas cuando haya un proceso ya establecido por el fabricante) y las alternativas o mejoras a lo largo de la implementación de este sistema compuesto por la cámara ZR300, conectada a la placa UDOO, y esta, a su vez, visualizada a través del ordenador portátil. UDOO y cámara están alimentados por una batería de modelo Ravpower xtreme series RP-PB14 23Ah/85Wh y el ordenador por su propia batería integrada.

Teniendo como objetivo el poder implementar un sistema de *SLAM* que pudiera funcionar en la placa UDOO, primero había que trabajar en un sistema funcional por sí mismo, y ver su adaptabilidad posterior a la SBC. Al ser ésta tan potente como para poder ejecutar el sistema operativo de Linux, gran parte del tiempo empleado fue en la configuración de un entorno de trabajo en el portátil que pudiera replicar y en la investigación sobre el funcionamiento de los distintos algoritmos.

Para el entorno de trabajo, tanto ambas bibliotecas, como la cámara, tienen la posibilidad de trabajar y recibir instrucciones a través de ROS, con lo que comenzaremos por él.

Configuré el portátil desde cero usando como sistema operativo Ubuntu, concretamente la distribución LTS *"Long Term Support"* de Linux 16.04, conocida como Xenial. Para esta distribución, desde la página de ROS se recomendaba la décima versión: Kinetic Kame [19]

Con el sistema operativo recién instalado, al tener apenas configuraciones realizadas, escogí el manual de instalación indicado [20] para la versión *desktop full*, porque permitía el uso de los archivos de navegación y percepción, ausentes en el resto de opciones.

Siguiendo el primer tutorial [21] establecí un espacio de trabajo donde instalar los repositorios correspondientes a las versiones de las bibliotecas de SLAM operativas con ROS.

Estableciendo las variables de entorno en la sesión `.bashrc` automáticamente, se facilita realizar las pruebas al reducir la cantidad de órdenes en cada intento, lo que significa menos fallos.

Un detalle que generaría problemas importantes durante la instalación de bibliotecas fue qué, ademas de incluirlas, había que ejecutarlas una primera vez, bajo la sentencia `export variable` en la linea de órdenes para que la reconociera.

Por último, realicé el resto de tutoriales para poder familiarizarme con la forma de trabajar en este entorno. [22]

Continuando con la cámara, he utilizado la *API* multiplataforma específica de Intel [23]
Por medio de sus ejemplos [24] pude realizar los primeros experimentos de medición distancias y reconocimiento de objetos.

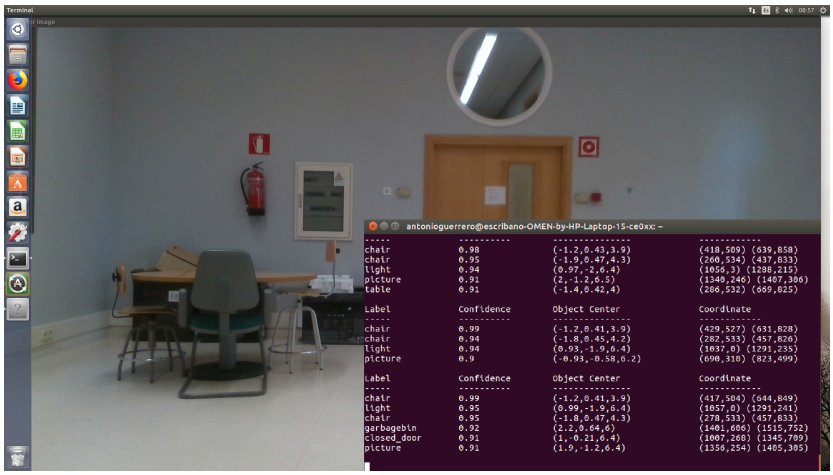


Figura 3.1: Tercer tutorial de reconocimiento de objetos de realsense (or_tutorial_3)

Con ellos también pude detectar los primeros errores de conexión:

"`rs.warn: Subdevice 0 bad magic number 0x0`
`rs.warn: Subdevice 1 bad magic number 0x`"

Estos errores surgieron durante la exploración de las capacidades del dispositivo, tanto por medio de las órdenes: `rostopic echo /camera...` seguido del dispositivo a estudiar: /camera_info, /depth, /rgb, /fisheye o /ir/ir2 y `rosservice call /camera/driver/get_imu_info` para los parámetros de las cámaras y de la IMU respectivamente, y del programa *cpp-enumerate-devices* para sus características físicas y funciones, como durante su intento de calibración 3.3.

Aunque estén catalogados como avisos, esos mensajes hacían referencia a una desconexión completa del flujo de imágenes de la cámara con lente de ojo de pez. Por parte de Intel [25] aportaron esta información:

"In our experience, bad magic numbers come from issues with the USB 3.0 host controller or long USB cables. If the camera and software stack is working fine with one machine and completely borked [broken] on another, at best I can say there is a hardware compatibility issue and there's nothing that can be fixed on the librealsense end."

Con la cual, reconectando el cable USB, efectivamente se resolvía. Barajaban con que el problema pudiera provenir de una cámara mal calibrada, aspecto que se desarrollará con mayor detalle en la sección 3.3.
Dando con esto por concluida la caracterización de la cámara, teniendo que asumir los parámetros proporcionados por ella.

El siguiente contratiempo fue al intentar utilizar el *topic Pointcloud* habiendo arrancado con el paquete *default* al iniciar la cámara: `roslaunch realsense_camera zr300_nodelet_default.launch`

y al ejecutar el RVIZ, este *topic* no se encontraba.

Buscando en la wiki de ROS sobre la biblioteca Realsense [26] encontré sobre los parámetros estáticos:

" Enable_pointcloud (bool, default: false) Specify if to enable or not the native pointcloud. By default, it is set to false due to performance issues. This option is depreciated in favor of the rgbd_launch pointcloud and will be removed in the near future."

Por lo que, desde entonces, la inicialización de la cámara a través de ROS que creo conveniente es `roslaunch realsense_camera zr300_nodelet_rgbd.launch` para utilizar el *topic POINTCLOUD2* para visualizarla correctamente. De lo que obtuve que el funcionamiento de un dispositivo de investigación depende mucho de la retroalimentación que le aporten los desarrolladores y de la comunicación con el fabricante, cuando es posible, para refinar su uso en las herramientas que se utilicen.

Con la cámara plenamente funcional comencé a trabajar con el software. Su instalación fue un proceso iterativo, consultando con el desarrollador, en el caso de `RTAB-Map`, y en los foros para el caso de `ORB-SLAM2`, para solucionar diferentes cuestiones, hasta dar con la configuración adecuada. Empezando por `RTAB-Map` comencé con la versión diseñada para trabajar con ROS. Es una versión con una interfaz más reducida y requería trabajar con el código para poder activar una serie de configuraciones, como la activación del optimizador g2o que mejoraba notablemente su rendimiento. Esto producía tener que detener, modificar y ver el resultado del cambio, con lo que para conocer los distintos parámetros se hizo necesario pasar a la versión independiente. Desde ella, aún requirió algún intercambio de información y análisis del funcionamiento del *pipeline* de la cámara, para la integración del `ORB-SLAM2` en el mismo, que estaba aún en desarrollo para el caso de la cámara ZR300.

Para el caso de `ORB-SLAM2`, el proceso de prueba y error dependió de aprender a encontrar las librerías dónde modificar la ubicación de las dependencias. Como ejemplo pudo ser el error `error while loading shared libraries...` que se corrige añadiendo `export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/ros/kinetic/lib/x86_64-linux-gnu` al final del archivo `/.bashrc`. De esta formam, le estás indicando, por medio de la creación de una variable de entorno, dónde tiene que buscar las bibliotecas que no es capaz de localizar.

Otro factor importante para poder exportarlo y trabajar en la placa es que la carga del vocabulario inicial era muy pesada y podía dedicar un tiempo demasiado alto para permitir pruebas cargando distintos ficheros de configuración de la cámara. Para esto, en el foro [27] se implementó una solución consistente en la modificación del formato de las imágenes del vocabulario convirtiéndolo a datos binarios, reduciendo así el tiempo de carga a la quinta parte del sistema inicial.

Finalmente llegamos al dispositivo empotrado, la placa UDOO. Por su capacidad, salvo la recomendación de no incluir una partición *SWAP*, que perjudica la vida útil de este tipo de memoria, repliqué el entorno de trabajo instalado en el portátil. (Ubuntu 16.04, Ros Kinetic y Realsense Crossplatform API)

Como primera prueba ejecuté el visualizador de ROS, RVIZ reproduciendo el mismo comportamiento que en el ordenador, aunque con obvias limitaciones reflejadas en un *frame rate* disponible prácticamente nulo.
El siguiente paso fue establecer la comunicación entre el portátil y la placa, necesario para poder efectuar pruebas de realización de SLAM utilizando el portátil como interfaz de la placa.

El primer intento fue por medio de conexión wifi desde el adaptador inalámbrico USB D-Link DWL-G122, a través de el programa de acceso remoto a escritorio `TEAMVIEWER`.
Esta opción tenía demasiada latencia aún sin ejecutar la cámara, por lo que no continúe con mas pruebas, al no ser viable para un comportamiento en tiempo real.

Para las conexiones de red entre ordenador y *hub* se utilizan cables directos, aquellos que siguen la misma norma T568A o T568B en ambos extremos. En cambio, establecer una red entre dos ordenadores, antiguamente requería cables cruzados en función de la norma utilizada para sus puertos.

Desde hace unos años se llevan implementando *drivers* con una característica conocida como Configuración Automática MDI/MDI-X o Auto-MDIX, que tienen la capacidad de modificar este tipo de conectores, de forma que se evita la necesidad de los cables cruzados, reconfigurándose para adaptarse a la necesidad. El caso concreto de UDOO es el controlador REALTEK RTL8111G que la identifica con la característica *"Crossover Detection & Auto-Correction"* [28]

De esta forma se puede acceder al estándar Gigabit que permite la placa UDOO.

La siguiente opción fue a través de una conexión *ssh* entre ambos sistemas. Instalé el programa según las indicaciones en [29], modificando el número de puerto al 2222 para no tener el parámetro por defecto.

Una vez establecida la conexión física, nos encontramos con lo que acabaría siendo un problema de compatibilidad de las unidades gráficas.

Utilizando el ejemplo de prueba de cámaras `cppcapture` (color, infrarrojas y ojo de pez) de realsense (por ser los mas ligeros al no requerir la etapa intermedia de ROS) no respondía. Probando funciones a través de ROS, por sí podía haber algún problema con la cámara, se podía ejecutar el *launch* previamente mencionado, pero de nuevo se detenía, al igual que con el programa común de comprobación de unidades gráficas `glxgears` y su modo de depuración, esta vez con los indicadores:

```
libGL error: No matching fbConfigs or visuals found
libGL error: failed to load driver: swrast
```

Esto indica que no se ha hecho una buena configuración al acceso de la GPU, así que no se ha podido cargar el software de renderizado.

```
escribano@escribano-UDOO-x86:~$ LIBGL_DEBUG glxgears
LIBGL_DEBUG: no se encontró la orden
escribano@escribano-UDOO-x86:~$ glxgears
libGL error: No matching fbConfigs or visuals found
libGL error: failed to load driver: swrast
Error: couldn't get an RGB, Double-buffered visual
escribano@escribano-UDOO-x86:~$ rosrun rviz rviz
[ INFO] [1524733944.322713158]: rviz version 1.12.15
[ INFO] [1524733944.322829578]: compiled against Qt version 5.5.1
[ INFO] [1524733944.322870039]: compiled against OGRE version 1.9.0 (Ghadamon)
libGL error: No matching fbConfigs or visuals found
libGL error: failed to load driver: swrast
Could not initialize OpenGL for RasterGLSurface, reverting to RasterSurface.
libGL error: No matching fbConfigs or visuals found
libGL error: failed to load driver: swrast
Violación de segmento (`core' generado)
```

Figura 3.2: Captura del terminal con las ejecuciones fallidas.

Para seguir trabajando, encontré esta consulta [30] relacionada con el error mostrado. Dado que el UDOO no tiene GPU de Nvidia, seguí hasta el penúltimo comentario, donde encontré cómo efectuar análisis respecto a las características de cada equipo.

`ldconfig -p | grep -i gl.so`

Con esta orden te muestra por pantalla con qué *driver*s se conectan las distintas bibliotecas de `OpenGL`. Haciendo la comparación, figura 3.3, para ver si había alguna forma de continuar, encontré las dos bibliotecas coincidentes.

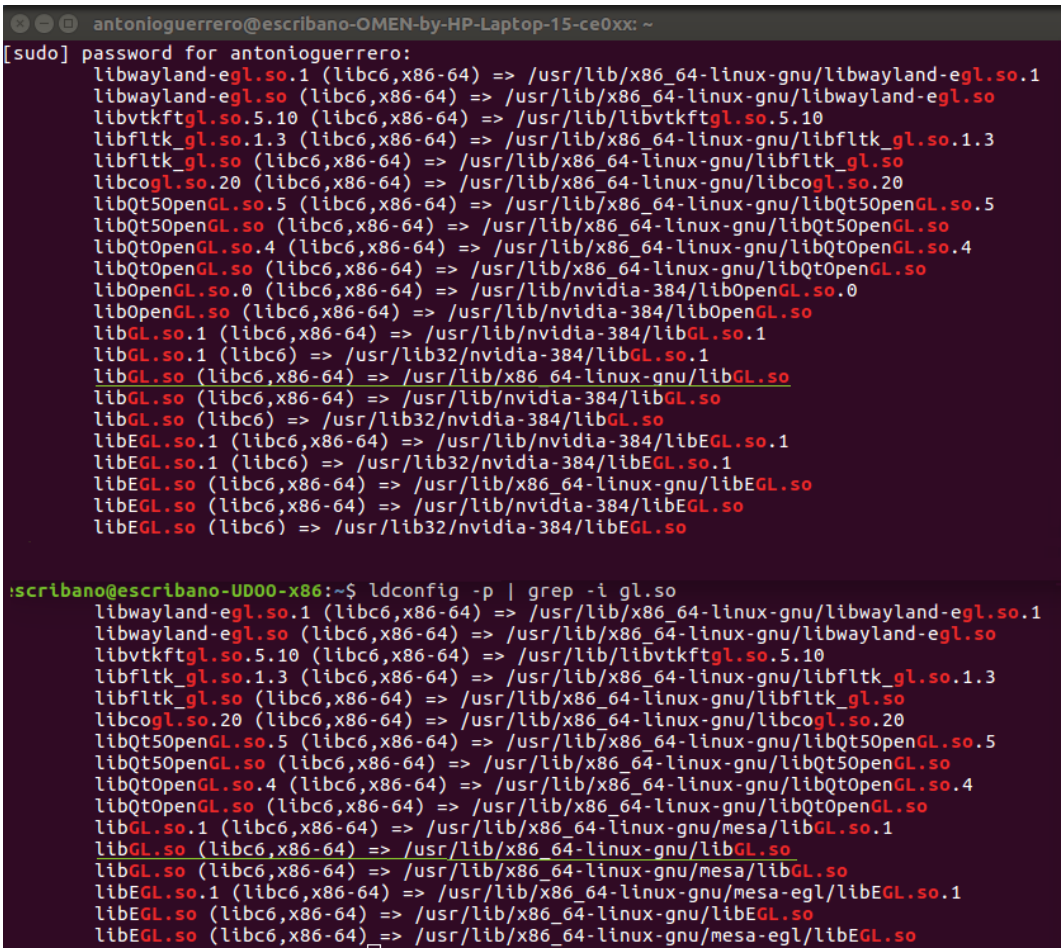


Figura 3.3: En la captura del terminal superior están las bibliotecas gráficas del portátil, frente a las correspondientes al UDOO en el inferior. Subrayada, está la biblioteca coincidente entre ambas.

Esto me hizo pensar que aparte de la tarjeta de Nvidia, el portátil podría tener una segunda tarjeta integrada de Intel y, en efecto, la orden `lspci | grep vga` confirmaba un segundo controlador de Intel. Como en las especificaciones habituales no hacen mención, fui a buscar información sobre el procesador, en el cuál estaba asociada una Intel HD Graphics 630. Sabiendo ésto, fui a los ajustes de Nvidia 3.4 desde donde podría modificar la utilización de una u otra.

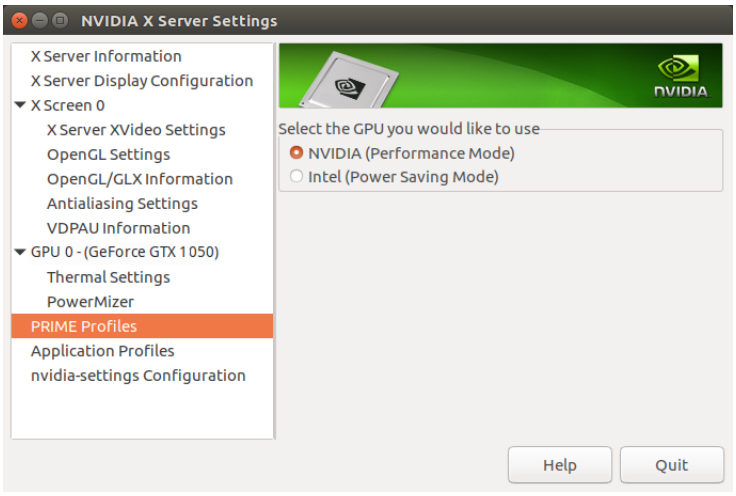


Figura 3.4: Perfil prioritario

Esta modificación permite la conexión y ejecución de programas que utilicen en `OpenGL`, porque se realiza el procesamiento en UDOO. Si se trata de ejecutar el programa, en estas condiciones directamente desde el portátil, volveremos a encontrarnos con el mismo error al estar presente el *driver* de `Nvidia`.

Una vez modificada, y después de reiniciar para establecer los cambios, ejecuté el programa `glxgears` en ambos para tener una idea de la capacidad de rendimiento gráfico del funcionamiento en ambas plataformas trabajando con las gráficas de Intel.

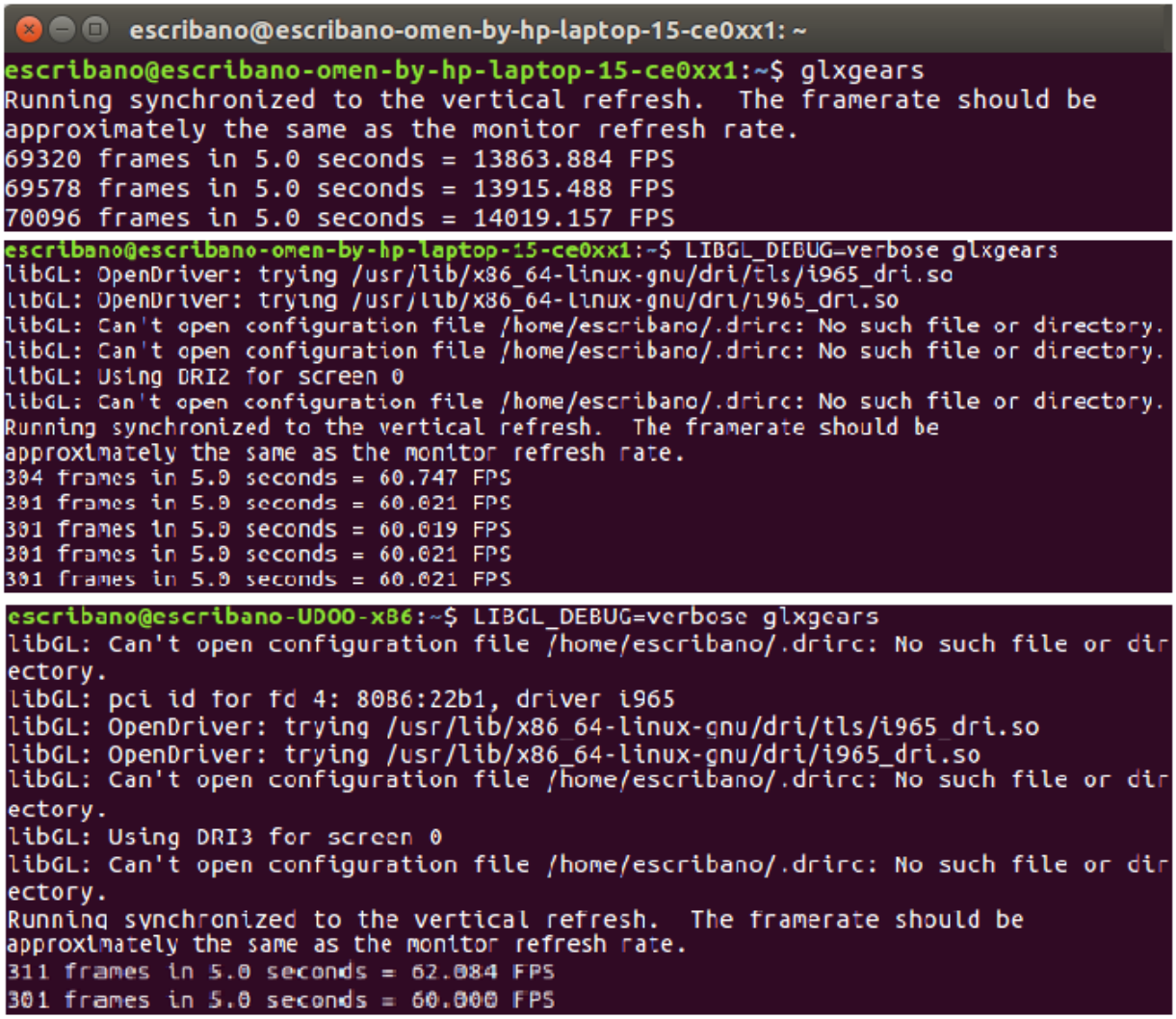


Figura 3.5: Comparación de frecuencias de adquisición en distintas configuraciones. De arriba hacia abajo, podemos encontrar en primer lugar, la ejecución en el portátil con la `Nvidia` activada. Debajo de ella está el comportamiento del portátil sin *drivers* de `Nvidia` y, por último, está el comportamiento del UDOO, similar al del portátil con la tarjeta gráfica de Intel.

Ya con el sistema funcionando, quedaban por controlar aspectos más técnicos, que sobre el papel pueden no llegar a verse hasta que ya está pensado.
El primero, mecánico, el soporte de la cámara tenía una sujeción magnética al trípode muy resistente a movimientos verticales, pero muy débil a los movimientos o golpes en direcciones laterales, para lo que hacia falta un soporte que lo anclara a la base.

Aproveché esta necesidad para ver la viabilidad de realizar un prototipo de cubierta con `Autocad` a partir de las medidas recogidas de la base y el imán e imprimirla en 3D por deposición fundida o FDM, que solventó el problema.

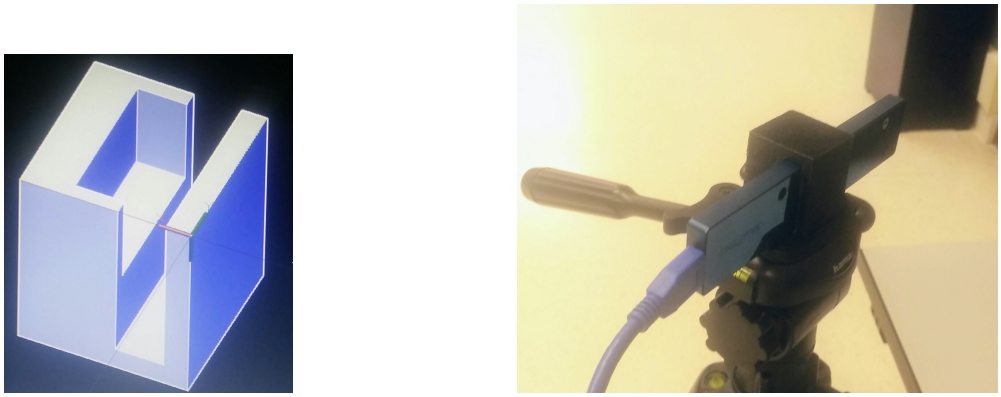


Figura 3.6: A la izquierda esta la pieza invertida para mejor visualización del interior y a la derecha como queda ya colocada.

El segundo, térmico, repercute en el rendimiento del sistema. Por una parte la cámara alcanzaba una temperatura bastante alta, que consideré necesario determinar. Según el manual de características, en la lista de necesidades térmicas 3.7 encontramos que la temperatura más baja que no debe alcanzar son 60°C, tanto para la cámara de color, como para la de ojo de pez.

4.6 Thermal Requirements

To ensure proper functionality of the module and sensor components, the camera module must not exceed the thermal requirements. The peripheral is designed not to exceed specifications in typical operating conditions. However, because usages and environments vary it may be necessary to verify the following conditions are met. Failure to meet the conditions below will result in degraded performance and accelerated component failure.

Table 4-7: Component Case Temperature Limits

Component	Case Temperature Limit	ΔT Junction to Case	Junction Temperature (Est.)
1080P Color Camera Sensor	60ºC	9ºC	69ºC
IR Camera Sensors	62ºC	8ºC	70ºC
Laser Projector	69ºC †	6ºC	75ºC
ASIC	95ºC	-	-
Fisheye Camera	60ºC	-	-
Motion Control Unit	85ºC	-	-

Figura 3.7: Tabla de especificaciones térmicas

Siendo estas temperaturas referidas a la cubierta del sensor, establecieron unos puntos de control descritos en la siguiente página. 3.8

Figure 4-3: Module Thermal Probe Points

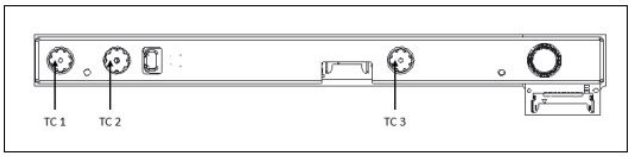


Table 4-8: Component Case Temperature vs. Junction Temperature

Point	Component	Junction Temp (Est.)
TC1	Right IR Camera Surface	Case Temp +8ºC
TC2	RGB Camera Surface	Case Temp +9ºC
TC3	Left IR Camera Surface	Case Temp +8ºC

Figura 3.8: Puntos de comprobación de temperatura

Realicé la medición de estos puntos con un termopar tipo K de un multímetro, con el que obtuve que, en funcionamiento a pleno rendimiento, no sobrepasaba los 45°C, con lo que estaba dentro de las condiciones indicadas, por debajo de los 69°C.

A continuación, y por expresa indicación del fabricante [17][Pg.15]
"Operating temperature: 0° C - +60° C (Commercial temperature)
Temperatures indicated are the maximum temperature that the heatspreader / heatsink can reach in any of its parts. This means that it is customer's responsibility to use any passive cooling solution along with an application-dependent cooling system, capable to ensure that the heatspreader / heatsink temperature remains in the range above indicated."
tuve que realizar la misma comprobación sobre la placa UDOO.

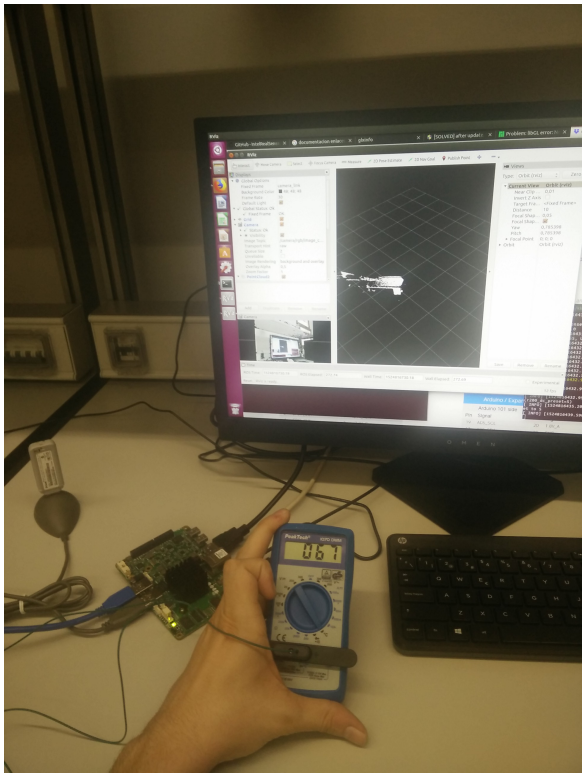


Figura 3.9: Medición en el disipador del procesador de UDOO

Obteniendo en este caso un resultado distinto al de la cámara, al alcanzar temperaturas de hasta 67°C, por encima de los 60°C recomendados para el disipador.

Esto deriva en un proceso conocido como estrangulamiento térmico o *thermal throttling*, que consiste en que ciertos dispositivos como procesadores o gpu, al ver superado determinados límites de temperatura, reducen su frecuencia por debajo de su frecuencia base para intentar evitar que resulte dañado. Como podemos comprobar entre las características del procesador de UDOO. [31]

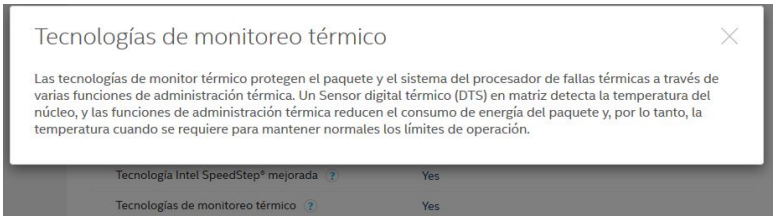


Figura 3.10: Control de temperatura Intel N3710

Con lo que, para un buen funcionamiento, esta placa requiere de ventilación activa.

3.3 Sistemas de evaluación

En este apartado describiré el proceso de calibración utilizado para conocer las características de la cámara y los métodos de comparación entre ambas librerías de SLAM.

3.3.1 Calibración

De inicio, las cámaras vienen con una calibración de fábrica accesible desde el nodo de ROS por medio del *topic*:

```
rostopic echo /camera/RGB/camera_info
```

D: [0.05677056685090065, -0.111920066177845, 0.0004538370412774384, 0.000584672496188432, 0.0]

D:[k1, k2, t1, t2, k3]

$$K = \begin{pmatrix} 612{,}8259887695312 & 0{,}0 & 320{,}830078125 \\ 0{,}0 & 613{,}188720703125 & 236{,}89515686035156 \\ 0{,}0 & 0{,}0 & 1{,}0 \end{pmatrix}$$

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

$$R = \begin{pmatrix} 1{,}0 & 0{,}0 & 0{,}0 \\ 0{,}0 & 1{,}0 & 0{,}0 \\ 0{,}0 & 0{,}0 & 1{,}0 \end{pmatrix}$$

$$P = \begin{pmatrix} 612{,}8259887695312 & 0{,}0 & 320{,}830078125 & 0{,}0 \\ 0{,}0 & 613{,}188720703125 & 236{,}89515686035156 & 0{,}0 \\ 0{,}0 & 0{,}0 & 1{,}0 & 0{,}0 \end{pmatrix}$$

$$P = \begin{pmatrix} fx' & 0{,}0 & cx' & Tx \\ 0{,}0 & fy' & cy' & Ty \\ 0{,}0 & 0{,}0 & 1{,}0 & 0{,}0 \end{pmatrix}$$

Aún con eso, estimamos necesario realizar una calibración propia para verificarlos.

Basándonos en los fundamentos de la cámara de la sección correspondiente 2.4.1.1 del capítulo Estudio teórico, el objetivo será conocer los parámetros intrínsecos, extrínsecos y los coeficientes de distorsión de la lente de la cámara, que permitan hacer una transformación de la escena en el espacio a la imagen en píxeles que proporcionará, para trabajar posteriormente con ella.

Para ello seguí el método de calibración de ROS, por medio del nodo cameracalibrator.py. [32] Este nodo es una adaptación del método ofrecido por la biblioteca `OpenCV`, que consiste por una parte, en hacer una estimación de los factores de escala entre las coordenadas del mundo real y las de la cámara, a partir de distintas posiciones de distancia, y por otra, de las matrices de rotación y transformación por medio de la modificación de la orientación de un tablero como el de la figura 3.11.

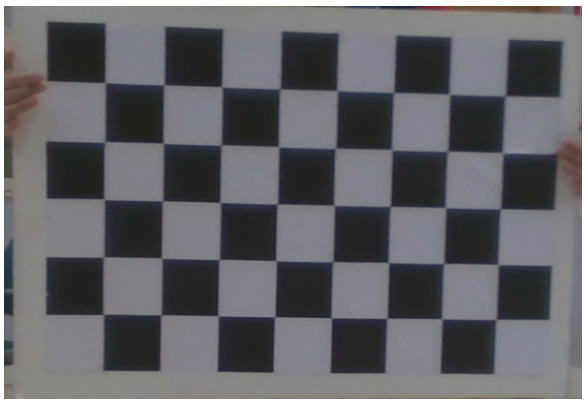


Figura 3.11: Tablero de ajedrez

Siguiendo los pasos de [33] ya en en el primer punto nos encontramos con el primer obstáculo. A pesar de tener su propio *wrapper* en ROS [34], en su punto [4.1.2] ya podemos comprobar que no cumple las condiciones de interfaz de una cámara estándar ROS, al no contar entre sus servicios el equivalente al `set_camera_info`, lo que produce esta respuesta al arrancarlo:
`"Waiting for service /camera/set_camera_info ...`
`Service not found"`
Aún así, previeron la opción `-no-service-check` que permitía arrancar sin él. Volveremos más tarde sobre este punto.

Este proceso se realiza, una vez ejecutado el nodo mediante:
`rosrun camera_calibration cameracalibrator.py -size 9x6 -square 0.108 image:=/camera/color/image_raw camera:=/camera`
colocándose frente a la cámara y, después de comprobar que reconoce el patrón. Actuando conforme a la interfaz 3.12 y alejándose unos pocos pasos, habrá que desplazar el tablero a lo largo y ancho de la pantalla, deteniéndose entre muestra y muestra, hasta que vuelva a reconocer el patrón. En el momento en que se llenen las barras de x e y, indicará que tiene suficientes muestras de escala y se repetirán los pasos inclinando el tablero en ambos ejes, hasta que se llené la barra de *skew*.

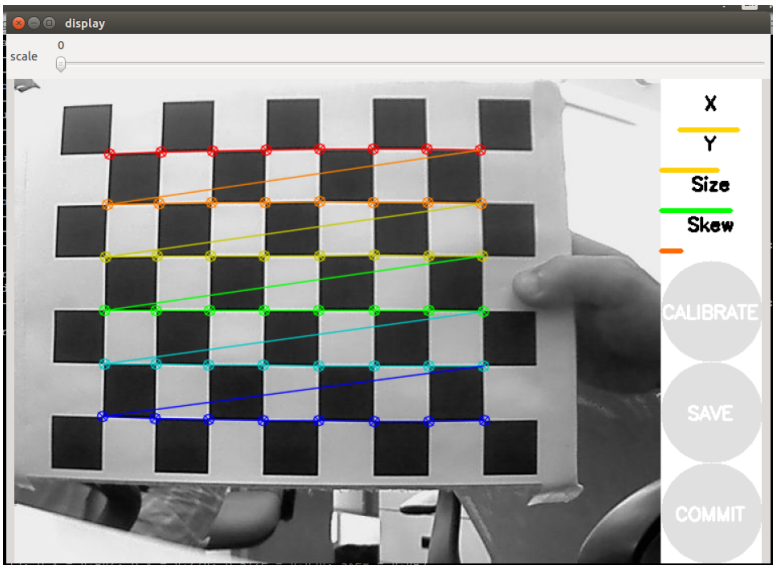


Figura 3.12: Interfaz de calibración.

En ese momento se activará el pulsador *Calibrate* y podrá realizar la calibración. Una vez realizada, si es correcta y estás de acuerdo, tienes la opción de guardar la calibración en tu cámara. Esto se realiza por medio de una llamada al servicio correspondiente a tu cámara `x/set_camera_info`, previamente mencionado, que no tiene la ZR300, por lo que no se podrán modificar los valores guardados.

En cualquier caso, nos sirve para hacer una comparación con los datos obtenidos. Para una calibración de 102 muestras obtuve estas matrices:

D: [0.1110677595148336, -0.3126v31394401532, -0.000930842140935718, 0.000847027688381279, 0.0]

$$K = \begin{pmatrix} 611{,}4216436791013 & 0{,}0 & 320{,}74738239038356 \\ 0{,}0 & 606{,}6674293670651 & 226{,}58378663760578 \\ 0{,}0 & 0{,}0 & 1{,}0 \end{pmatrix}$$

$$R = \begin{pmatrix} 1{,}0 & 0{,}0 & 0{,}0 \\ 0{,}0 & 1{,}0 & 0{,}0 \\ 0{,}0 & 0{,}0 & 1{,}0 \end{pmatrix}$$

$$P = \begin{pmatrix} 615{,}7793579101562 & 0{,}0 & 320{,}3380836467077 & 0{,}0 \\ 0{,}0 & 612{,}7356567382812 & 226{,}29533504920255 & 0{,}0 \\ 0{,}0 & 0{,}0 & 1{,}0 & 0{,}0 \end{pmatrix}$$

Con esto, podemos apreciar diferencias en las variables fy y cy para la matriz K, que representan una modificación de la representación de la altura de la imagen en el mundo real, en la imagen en píxeles. Ademas, también hay una variación notable en el modelo de lente mostrado en ambas calibraciones mediante la matriz D.

Debido a esto, intenté continuar investigando la forma de cargar los valores en la cámara. Buscando consultas similares en las páginas de Intel, otros usuarios se encontraron con la misma necesidad, lo que llevó Intel a dar este mensaje: [**?**] *"After further investigation, I have to inform you that there is no way to calibrate the camera on the field. The only solution available is to return the camera."*

Continuando con el hilo, mas abajo, proponían una aplicación con la que podría ser posible modificar los datos, pero, dado que la respuesta por parte de la compañía es que al no haber sido comprobada por ellos, no se hacían responsables de los resultados en que derivara, asumí la calibración de serie.

3.3.2 SLAM

Un mapa creado a partir de SLAM se puede valorar por varios métodos, se pueden hacer comparaciones de los errores de percepción, aunque tampoco aseguran mejores mapas o se puede trabajar sobre el proceso terminado, bien sea el propio mapa o la trayectoria seguida para formarlo, siendo este último enfoque, descrito en [35], el utilizado en este trabajo.

Para la evaluación de la calidad de estas bibliotecas, tanto desde los desarrolladores de ORB-SLAM2 [36] [Apart. 6 RGBD Example], como desde el foro de RTAB-Map [37], tuvieron en común la recomendación del proceso de comparación de su sistema de entrada de datos, para nuestro caso, RGB-D, el uso de los dataset de la Universidad Técnica de Munich (TUM) que fueron realizados para el estudio mencionado en el párrafo previo.
Estos *datasets* consisten en distintos archivos compuestos por un conjunto de imágenes en color, un conjunto de imágenes de profundidad, un archivo de texto con marcas de tiempo asociadas a cada imagen de cada grupo y un archivo conocido como *ground truth* que es contra el que se realiza la comparación.
Entre ellos, escogí, de la clasificación de SLAM de desplazamiento manual, más acorde a mis condiciones de trabajo, el archivo *freiburg3/long_office_household* por ser el más reciente.
El proceso requiere una sincronización previa de ambos grupos de imágenes, debido a que la cámara Kinect recoge los flujos de imágenes a color y de profundidad por separado. Para ello se utiliza el *script* desarrollado también por la TUM `associate.py` [38] que proporcionará una carpeta con las imágenes en las que puedan establecerse una conexión y un archivo de texto con las marcas de tiempo de ellas.

El siguiente paso será crear un archivo `yaml` en el que introduciremos los valores de la calibración aportada por la universidad. Esto es porque ahora vamos a comprobar la capacidad y rendimiento de los programas independiente de nuestra cámara. Sólo queremos comprobar cómo se comportan en el portátil y en el UDOO. Así que se utilizarán los valores de la cámara Kinect que utilizaron para grabar los *datasets*.

El siguiente paso será ejecutar el programa para que siga el *dataset* y fabrique una trayectoria de posiciones que comparar.

Para evaluar ORB-SLAM2 ejecutaremos estas órdenes:
`ORB_SLAM2/Examples/RGB-D/rgbd_tum`
-Esta determina el tipo de cámara que se quiere utilizar.
`ORB_SLAM2/Vocabulary/ORBvoc.txt (.bin)`
-Esta es el vocabulario previamente mencionado en 2.6 y la opción de escoger el formato.
`ORB_SLAM2/Examples/RGB-D/TUM1.yaml`
-En este argumento podremos escoger el formato de calibración correspondiente a la cámara que usemos.
`ORB_SLAM2/datasets/rgbd_dataset_freiburg1_xyz`
-Desde aquí se tiene que designar el directorio contenedor de las secuencias de imágenes asociadas.
`ORB_SLAM2/Examples/RGB-D/associations/fr1_xyz.txt`
-Y, por último aquí se tiene que direccionar el archivo con las marcas de tiempo de los archivos asociados.
Que nos proporcionara el fichero `keyframes.txt`

Para el caso del RTAB-Map, salvo el vocabulario, específico de ORB-SLAM2, la ubicación de todos estos parámetros se tienen que indicar desde la interfaz. Como recomendación, añade la posibilidad de activar un parámetro que crea nodos intermedios entre los que detecte, para realizar una comparación aún mas fiable. Una vez finalizado el recorrido, obtienes el archivo `poses.txt` desde el menú avanzado. Ahora que ya tenemos las trayectorias, sólo resta la comparación contra el *ground truth* y analizar los resultados.

Iniciamos la herramienta de evaluación programada en Python, que determina el ATE o error absoluto de trayectoria en inglés, con la orden:

```
python evaluate_ate.py /home/antonioguerrero/ORB_SLAM2/datasets/
rgbd_dataset_freiburg3_long_office_household_validation/groundtruth.txt
KeyFrameTrajectory.txt -plot ateorb2.png -verbose
```

Los argumentos corresponden a las ubicaciones de los elementos a comparar; *plot* a la orden de dibujar ambas trayectorias y su relación 3.13 y *verbose*, que permite mostrar todos los datos de evaluación calculados.

Por la procedencia del desarrollador, para RTAB-Map, requiere una modificación, ya que él trabaja con unidades imperiales y da lugar a diferencias en la medición de errores.

En el método `read_file_list` en la linea 66 hay que añadir `replace(",", ".")` a `lines = data.replace(",", ).replace("\t",).split("\n")`

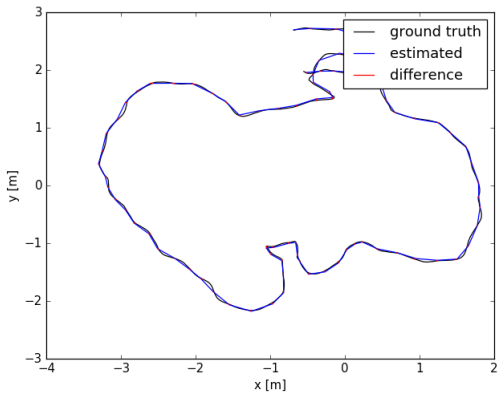


Figura 3.13: Ejemplo de resultado de `Evaluate.py`

Capítulo 4

Pruebas y resultados

Como se ha mencionado en el apartado de SLAM 3.3.2 del capítulo previo, he utilizado el *dataset* de la universidad de Munich, sobre el que he querido evaluar los comportamientos para las distintas posibilidades. Introduciendo los ficheros de trayectorias de los algoritmos `ORB-SLAM2`, `RTAB-Map` y `RTAB-Map` con la integración de `ORB-SLAM2` se obtienen estos resultados.

Plataforma		Omen	
Biblioteca	`ORB-SLAM2`	`RTAB-Map`	`RTAB-Map` con `ORB-SLAM2`
Poses file-pairs	KFT** - 218	poses05-6g2o.txt - 200	posesrtorb2.txt - 218
ATE*	-	-	-
Rmse	0.106127	0.033321	0.022486
Media	0.103663	0.031861	0.019410
Mediana	0.106146	0.030131	0.016580
Estandar	0.022737	0.009754	0.011352
Minimo	0.041373	0.016545	0.004763
Maximo	0.145973	0.061636	0.063049
Plataforma		UDOO	
Biblioteca	`ORB-SLAM2`	`RTAB-Map`	`RTAB-Map` con `ORB-SLAM2`
Poses file-pairs	KFT** - 223	posesrtabmap_5-06_30hz.txt= - 289	poses.txt-165
ATE*	-	-	-
Rmse	0.107879	0.061848	0.041545
Media	0.105704	0.055130	0.039059
Mediana	0.107907	0.047995	0.036335
Estandar	0.021554	0.028032	0.014155
Minimo	0.056950	0.007329	0.015617
Maximo	0.146003	0.121167	0.092031

*ATE *Absolute Trajectory Error* **KFT *Keyframe Trajectory*

Se pueden observar cosas interesantes a partir de ellos. Atendiendo a la variación entre plataformas se aprecia que para el ORB-SLAM2 la diferencia del error es apenas perceptible, en cambio para ambas mediciones de RTAB-Map se obtiene aproximadamente el doble de error en su integración sobre la placa de desarrollo. Esto se explica porque el algoritmo de ORB-SLAM2 es robusto debido a su forma de representación dispersa o mas esparcida del mapa y puede adaptarse correctamente a la placa. Es importante recalcar que para las condiciones de la placa, aún siendo capaz de obtener estos resultados, en las pruebas de funcionamiento sobre el terreno requiere una ventilación apropiada para poder realizar un trabajo continuado o detendrá el algoritmo por la disminución o ausencia de características detectadas.

Por otra parte, se observan mejores resultados para el RTAB-Map y levemente mejores para la integración de ambos, mas apreciables en la prueba en UDOO, dónde las limitaciones de procesamiento son mayores.

Con respecto a su capacidad de trabajar en exteriores, como cámara RGB-D al trabajar con un proyector de infrarrojos, la luz del sol se puede superponer a la imagen proyectada e impedir obtener información de profundidad, pero si se deshabilita el modo automático, desde [39] nos aportan una configuración de parámetros que permiten su uso como podemos comprobar en la imagen siguiente.

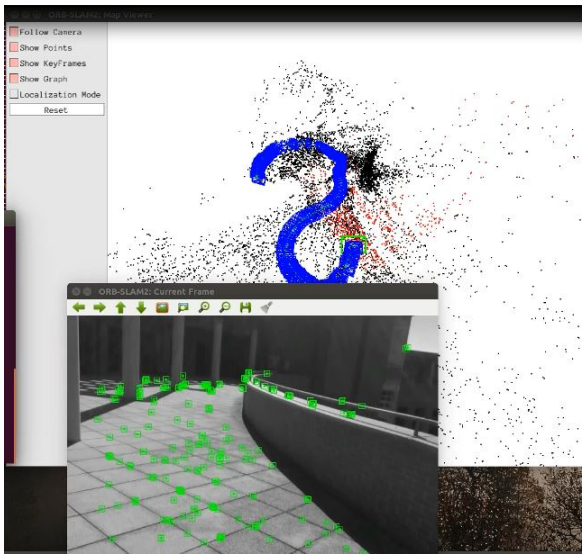


Figura 4.1: Detección de características en ORB-SLAM2 de la cámara con parámetros ajustados para su funcionamiento en exteriores.

El siguiente aspecto estudiado fue la recuperación de odometría.
Esto se produce cuando en la etapa de realización del mapa, encuentra una zona que o bien por el sensor, o bien por el entorno, no consigue detectar nuevas características. En ese caso el problema se resuelve volviendo a la última posición detectada.

ORB-SLAM2 tiene una cualidad muy buena, que es la capacidad de detectar muchas características, por lo que generalmente, para un slam pasivo o teleoperado se puede conseguir sin mucha dificultad.

Sin embargo, si el entorno tiene muy poca textura o no se adapta bien a las capacidades del sensor, tiene un gran inconveniente debido a la interfaz. En ella dispones de una ventana para odometría y otra para situar el grafo, los *keyframes* y las características. Al perder la odometría, el movimiento del mapa se detiene, con lo que no conservas información sobre el punto en el que se perdió, hacia el que se pueda volver.

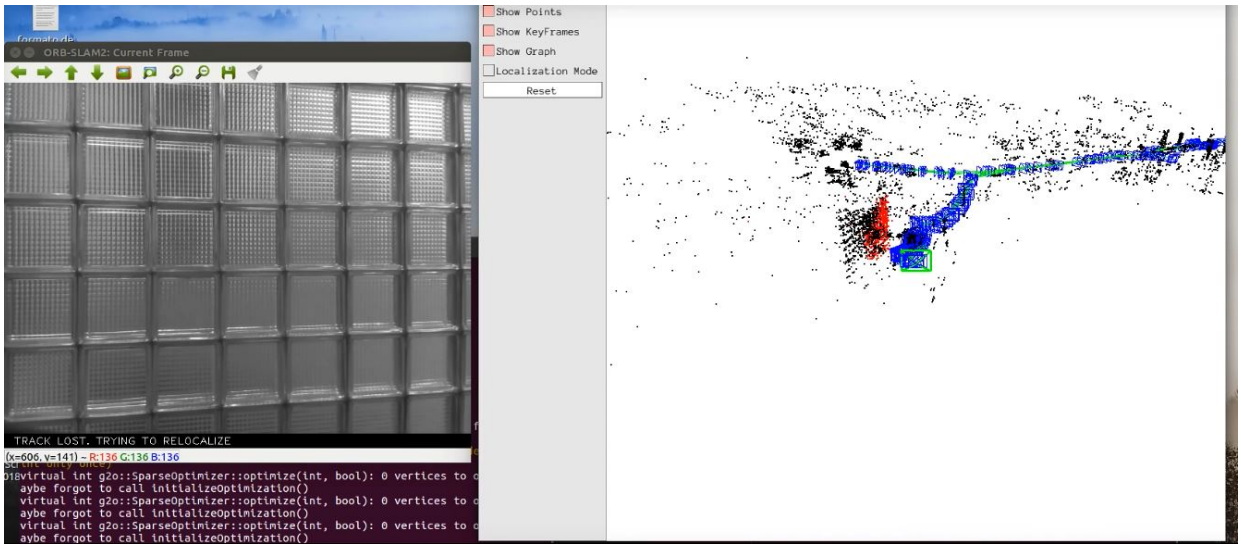


Figura 4.2: A pesar de estar mirando hacia los cristales, a la izquierda, la orientación de la cámara, en verde, sigue mirando hacia el pasillo.

En RTAB-Map se captura la escena en cuanto pierde la odometría y se superpone degradada al flujo de la cámara, con la que puedes encontrar fácilmente la orientación y posición para poder continuar.

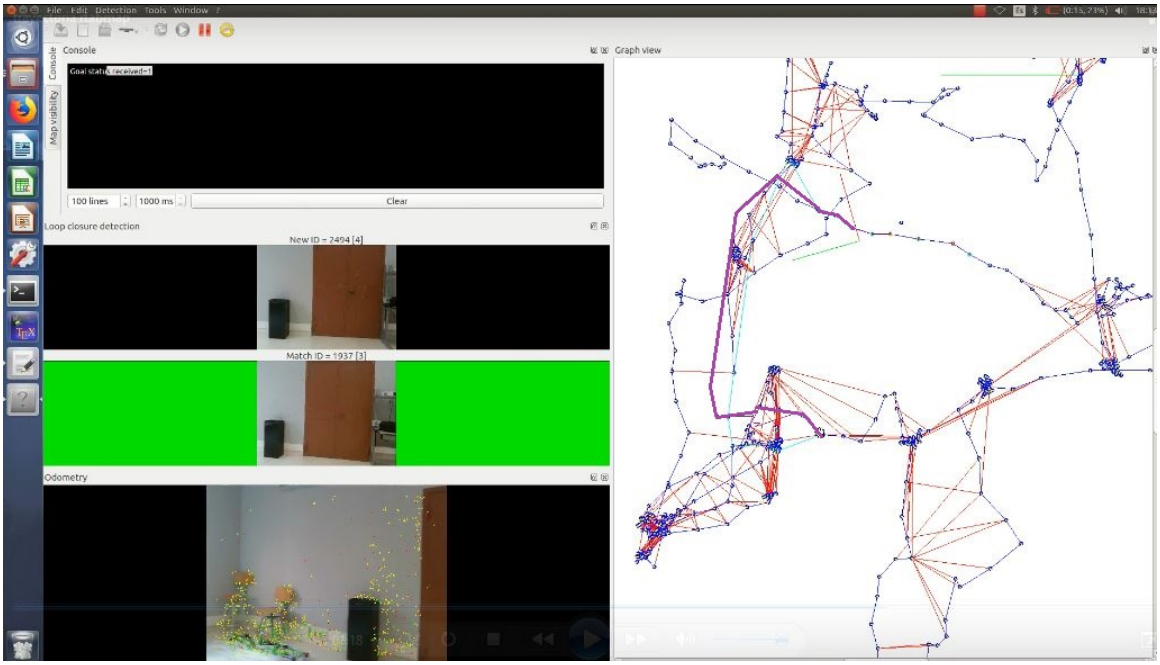


Figura 4.3: El modulo de odometría se puede ver en este caso en el segmento inferior izquierda.

Una vez obtenido un mapa, se repite el estudio de relocalización. Como ahora se puede encontrar mas fácilmente por la cantidad de marcas obtenidas, se trata de evaluar el problema del secuestro. Consiste en cortar la odometría, en este caso tapar la cámara y desplazarte a otra posición de tu mapa a ver si es capaz de ubicarse.

De nuevo, empezando por ORB-SLAM2, este ha sido capaz de distinguir y ubicarse en todas las localizaciones previamente observadas e incluso en alguna con una rotación de la escena cercana a 90 grados.

En esta prueba RTAB-Map también ha sido capaz de dar un buen rendimiento, pero no de forma tan contundente como el algoritmo de Zaragoza.

Por último, quise comprobar su comportamiento para la corrección del mapa con el cierre de lazos.

Para esta comprobación fue mas difÃcil de determinar un buen comportamiento en el algoritmo ORB-SLAM2, puesto que durante la prueba más larga, apenas fue un lazo de unos pocos metros y es posible que la desviación entrara dentro de los mínimos admisibles, por lo que no realizo ninguna corrección.

En cambio en el caso de RTAB-Map sí que pudimos obtener resultados fácilmente, como es el caso de las siguientes figuras.

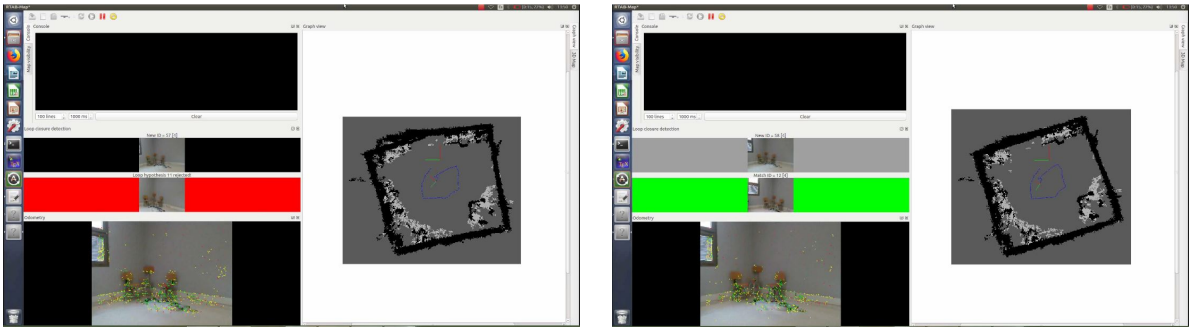


Figura 4.4: En estas figuras se puede ver una detección de candidato a cierre de lazo. A la izquierda se puede ver cómo no cumple la condición y lo rechaza (en rojo) y a la derecha ha encontrado la imagen correcta y ha restaurado la forma correspondiente.

Como prueba final, y ya sin establecer una comparación, ya que en el caso de ORB-SLAM2 este módulo no estaba implementado, examiné la capacidad deRTAB-Map para establecer trayectorias. Desde la posición que te encuentres, escoge el nodo más cercano y va trazando un camino hasta el nodo con la etiqueta que le hayas indicado. También dispone de la posibilidad de seleccionar varios a puntos a la vez y de esta forma trazar una ruta más controlada.

Volviendo a la imagen 4.3, se puede comprobar como en el grafo del lado derecho aparece un segmento morado, remarcado para esta explicación, que trazaría la ruta desde la posición inicial de la cámara, que son los segmentos perpendiculares verde y rojo, hasta el nodo escogido. Tiene una variable, *GoalRadius* para regular la precisión con la que quieras detectar haber alcanzado la meta. En este caso estaba a 0,5 metros, porque era una prueba sobre desplazamiento manual de la propia cámara.

Capítulo 5

Conclusiones y líneas de trabajo futuras

En este trabajo se ha descrito como caracterizar y calibrar los dispositivos necesarios para formar un sistema de visión artificial. Describiendo sus procesos de interconexión se ha podido implementar dicho sistema para poder evaluar distintas estrategias de SLAM. También se han expuesto métricas de comparación para medir el comportamiento de ambas bibliotecas para distintas situaciones.

Con todo ello, encuentro como conclusiones que un sistema empotrado de SLAM a partir de la información que puede proporcionar una cámara de profundidad para estas dos bibliotecas, y especialmente sobre una placa SBC sin capacidad de procesamiento de GPU, aun tiene limitaciones para poder trabajar de forma autónoma, por la facilidad de encontrarse con situaciones en un entorno real y dinámico de pérdida de odometría. Puede detectar suficientes características en entornos ideales, recuperar posiciones y seguir trayectorias, en entornos que no tengan superficies planas demasiado extensas, que permiten establecer mapas relativamente precisos, pero sin más apoyo de odometría de otros sensores, para otro tipo de entornos, no lo considero mas allá de una plataforma de prototipado o aprendizaje.

Con respecto a las bibliotecas, el procesamiento de un mapa denso como el caso de RTAB-Map es demasiado para la placa UDOO, con lo que aún se podría trabajar, pero corriendo el riesgo de no encontrarse en caso de pérdida de odometría, ya que depende de la información que no puede proporcionar, si se busca un comportamiento de un robot teleoperado.

Como líneas de trabajos futuros que se podrían continuar desde aquí podríamos:

- Incorporar la información de la IMU integrada en la camara ZR300 teniendo así la posibilidad de un sistema de SLAM visual inercial más robusto que mejore las recuperaciones de odometría.
- Utilizar la aplicación de `RTAB-Map` como parte de un método de enseñanza al ser una plataforma tan versátil a la hora de incorporar sensores.
- Investigar cómo las nuevas iteraciones de `ORB-SLAM` como `ORBSLAM Atlas` u `ORB-SLAM3` resuelven los problemas de procesamiento o pérdida de odometría.
- Modificar la placa por una con capacidad de procesamiento de GPU que permita implementar un mejor reconocimiento de imágenes por medio de CUDA, paralelizando las tareas.

Bibliografía

[1] T. B. H. Durrant-Whyte, "Simultaneous localization and mapping: Part i," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[2] M. Burhanpurkar, M. Labbé, X. Gong, C. Guan, F. Michaud, and J. Kelly, "Cheap or Robust? The Practical Realization of Self-Driving Wheelchair Technology," *arXiv e-prints*, p. arXiv:1707.05301, Jul 2017.

[3] I. Çirauqui Viloria and J. M. Martínez Montiel, "Evaluación de orbslam en secuencias de endoscopia médica," "2016".

[4] *Intel RealSense 3D Camera ZR300 Product Datasheet - Revision 1.0*, Intel, January 2017.

[5] L. Vázquez, C. Andrés, F. J. Meca Meca, P. Pérez, and E. Daniel Santiso Gómez, *Apuntes de Instrumentación Electrónica (600018)*, Politécnica Superior de Alcalá de Henares, 2018.

[6] "Explicacion calibracion opencv," https://docs.opencv.org/trunk/d9/d0c/group___calib3d.html [Último acceso 10/marzo/2021].

[7] "Orientación coordenadas cámara," https://software.intel.com/content/www/us/en/develop/documentation/realsense-for-linux-documentation/top/rsobjectrecognition-namespace-reference/rsobjectrecognitionlocalizationdata-struct-reference/rsobjectrecognitionlocalizationdata-member-list/rsobjectrecognitionorconfigurationinterface-class-reference.html [Último acceso 10/marzo/2021].

[8] "Algoritmo de proyeccion," http://docs.ros.org/kinetic/api/librealsense/html/rsutil_8h_source.html#l00011l [Último acceso 10/marzo/2021].

[9] N. Sunderhauf, "Robust optimization for simultaneous localization and mapping," Ph.D. dissertation, 2012.

[10] Z. Chen, "Bayesian filtering: From kalman filters to particle filters, and beyond," *Statistics*, vol. 182, 01 2003.

[11] H. Strasdat, J. Montiel, and A. J. Davison, "Visual slam: Why filter?" *Image and Vision Computing*, vol. 30, no. 2, pp. 65–77, 2012.

[12] E. Karami, S. Prasad, and M. Shehata, "Image matching using sift, surf, brief and orb: Performance comparison for distorted images," 2017.

[13] "Descripcion caracteristicas en opencv," https://docs.opencv.org/master/db/d27/tutorial_py_table_of_contents_feature2d.html [Último acceso 10/marzo/2021].

[14] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[15] R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G2o: A general framework for graph optimization," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 3607–3613.

[16] M. Labbe and F. Michaud, "Appearance-based loop closure detection for online large-scale and long-term operation," *Robotics, IEEE Transactions on*, vol. 29, pp. 734–745, 06 2013.

[17] S. R. by M.B, *UDOO X86 User Manual - Last Edition: 1.4*, UDOO, September 2017.

[18] "Guia de desarrollo de realsense," https://github.com/IntelRealSense/librealsense/tree/v1.12.1 [Último acceso 10/marzo/2021].

[19] "Version ros," http://wiki.ros.org/kinetic [Último acceso 10/marzo/2021].

[20] "Proceso de instalacion de ros kinetic," http://wiki.ros.org/kinetic/Installation/Ubuntu [Último acceso 10/marzo/2021].

[21] "Configuracion de entorno de trabajo de ros," http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment [Último acceso 10/marzo/2021].

[22] "Tutoriales de ros," http://wiki.ros.org/ROS/Tutorials [Último acceso 10/marzo/2021].

[23] "Instalacion biblioteca realsense," https://github.com/IntelRealSense/librealsense/blob/v1.12.1/doc/installation.md [Último acceso 10/marzo/2021].

[24] "Tutoriales de realsense," https://github.com/IntelRealSense/realsense_samples/tree/master/samples [Último acceso 10/marzo/2021].

[25] "Corrección," https://community.intel.com/t5/Items-with-no-label/ZR300-fisheye-intrinsics/td-p/516884 [Último acceso 10/marzo/2021].

[26] "Documentacion intel realsense," https://github.com/IntelRealSense/librealsense/blob/v1.12.1/doc/whats_new.md [Último acceso 10/marzo/2021].

[27] "Modificacion del formato del vocabulario," https://github.com/raulmur/ORB_SLAM2/pull/21 [Último acceso 10/marzo/2021].

[28] "Información driver ethernet de udoo," https://www.realtek.com/en/products/communications-network-ics/item/rtl8111g [Último acceso 10/marzo/2021].

[29] "Instalacion ssh," http://ubuntuhandbook.org/index.php/2016/04/enable-ssh-ubuntu-16-04-lts/ [Último acceso 10/marzo/2021].

[30] "Swrast," https://askubuntu.com/questions/541343/problems-with-libgl-fbconfigs-swrast-through-each-update [Último acceso 10/marzo/2021].

[31] "Control temperatura inteln3710," https://ark.intel.com/content/www/es/es/ark/products/91830/intel-pentium-processor-n3710-2m-cache-up-to-2-56-ghz.html [Último acceso 10/marzo/2021].

[32] "Codigo fuente camera calibrator.py,"
https://github.com/strawlab/image_pipeline/blob/master/camera_calibration/nodes/cameracalibrator.py.

[33] "Documentacion calibrator.py," http://wiki.ros.org/camera_calibration.

[34] "Intel realsense sdk for linux ros samples,"
https://github.com/IntelRealSense/realsense_samples_ros [Último acceso 10/marzo/2021].

[35] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[36] "Página de orb-slam2," https://github.com/raulmur/ORB_SLAM2 [Último acceso 10/marzo/2021].

[37] "Recomendación dataset rtabmap," http://official-rtab-map-forum.67519.x6.nabble.com/How-to-process-RGBD-SLAM-datasets-with-RTAB-Map-tp939p647.html [Último acceso 10/marzo/2021].

[38] "Herramientas de evaluacion rgb-d de la universidad de munich,"
https://vision.in.tum.de/data/datasets/rgbd-dataset/tools.

[39] "Ajustes para exteriores," https://github.com/IntelRealSense/librealsense/issues/208 [Último acceso 10/marzo/2021].

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá