Universidad Zaragoza

Trabajo de Fin de Grado

Localización y construcción de mapas con puntos en profundidad inversa y con cámaras gran angular en ORB-SLAM2

Localization and mapping with inverse depth points and wide angle cameras in ORB-SLAM2

Autor

Juan José Gómez Rodríguez

Director

Juan Domingo Tardós Solano

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2018

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Juan José Gómez Rodríguez,

con nº de DNI 18173281-S en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Localización y construcción de mapas con puntos en profundidad inversa y con cámaras gran angular en ORB-SLAM2,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 25 de Junio de 2018

Fdo: Juan José Gómez Rodríguez

AGRADECIMIENTOS

Agradecer en primer lugar a Juan D. Tardós, tanto por aceptarme como alumno suyo para la realización de esta investigación, como por su apoyo y su implicación, lo que me ha abierto importantes salidas profesionales. *¡Muchas gracias Mingo!*.

También me gustaría agradecer a José María Martínez Montiel, que sin ser mi tutor, siempre ha mostrado interés por el desarrollo de este proyecto.

Finalmente, agradecer a mi familia y amigos por el apoyo durante todo este tiempo.

RESUMEN

El problema de *localización y mapeo simultáneos*, más conocido por sus siglas en inglés SLAM, consiste en localizar un agente (por ejemplo, un robot, un dron, un coche autónomo, o unas gafas de realidad aumentada) dentro de un mapa generado gracias a información obtenida por una serie de sensores de a bordo. ORB-SLAM2 es un sistema de SLAM desarrollado por el grupo de Robótica, Percepción y Tiempo Real de la Universidad de Zaragoza, que utiliza cámaras como sensores. Como la mayoría de de sistemas de SLAM visual, representa los puntos del mapa en coordenadas Cartesianas y utiliza un modelo de cámara oscura (*pinhole* en inglés) para la toma de imágenes.

En el presente trabajo se ha investigado el uso de una representación alternativa para los puntos del mapa de ORB-SLAM2, en el que se almacenan sus coordenadas angulares y la inversa de su profundidad con respecto de la primera cámara que los observó. Esta representación exhibe una serie de propiedades teóricas que resultan muy beneficiosas en sistemas de SLAM basados en el filtro de Kalman extendido. Sin embargo, nuestros resultados muestran que en el caso de ORB-SLAM2, que utiliza optimización no lineal, la representación en profundidad inversa aumenta el coste computacional sin mejoras significativas en la precisión.

Por otro lado, se ha estudiado el modelo de cámaras con objetivos gran angular y ojo de pez y, modificando a fondo ORB-SLAM2, se ha desarrollado el primer sistema de SLAM visual capaz de trabajar con este tipo de cámaras, tanto en monocular como en estéreo. Los resultados obtenidos con unas gafas de realidad aumentada muestran que este tipo de cámaras, al cubrir un mayor campo de visión que las basadas en el modelo *pinhole*, permiten mapear regiones más amplias, mejorando la robustez del SLAM visual ante oclusiones.

Índice

Capítulo 1

Introducción y objetivos

1.1. Sistemas de localización y mapeo simultáneo

El problema de *localización y mapeo simultáneo*, más conocido por sus siglas en inglés SLAM, surgió en el ámbito de la robótica ante la necesidad de crear un mapa del entorno de un robot y, a la vez, localizarlo dentro del mismo utilizando la información que brindan los diferentes sensores que pueda llevar el robot.

Este problema se ha extendido a otras muchas aplicaciones, tales como gafas de realidad aumentada (AR) o realidad virtual (VR) donde lo que se pretende localizar dentro de un mapa ya no es un robot si no al propio usuario de las gafas para modificar en tiempo real la visualización que se le presenta.

Existen una gran variedad de sensores que se pueden utilizar para resolver el problema de SLAM, sin embargo, los más populares, tanto por su coste como por su versatilidad, son las cámaras. Una solución al problema de SLAM visual, propuesta por el grupo de Robótica, Percepción y Tiempo Real de la Universidad de Zaragoza es ORB-SLAM2 [10].

1.2. Contexto

El presente documento es el resultado de un proyecto de iniciación a la investigación realizado en el *SLAMLab* del grupo de investigación Robótica, Percepción y Tiempo Real de la Universidad de Zaragoza bajo un contrato de Prácticas extra-curriculares con cargo a la Linea de Investigación: *LTI3A0105: Visión por computadora para robótica y realidad virtual*, donde el investigador principal es Juan D. Tardós.

De igual manera, durante la presente investigación se ha realizado una colaboración en el proyecto *OTRI 2016/1099 : Technical assistance for implementing ORB-SLAM monocular V1.0*, financiado por *Meta Company*[1], donde el investigador principal es

[1] `http://www.metavision.com`

Juan D. Tardós. Para llevar a cabo tal colaboración, la empresa cedió un kit de gafas de realidad aumentada *Meta 2*.

1.3. Objetivos y alcance del proyecto

ORB-SLAM2, al igual que la inmensa mayoría de sistemas de SLAM visual, representa los puntos del mapa que va construyendo con coordenadas Cartesianas. Sin embargo, en [1] se presenta una nueva parametrización de los puntos del mapa llamada *puntos en profundidad inversa* con una serie de propiedades teóricas de especial interés en el contexto de SLAM.

Por otro lado, ORB-SLAM2 soporta 3 tipos diferentes de cámaras siguiendo un modelo de cámara oscura (*pinhole* en ingles): monocular, estereoscópica y RGB-D. Actualmente, muchas aplicaciones que requieren de un sistema de SLAM, tales como gafas de AR, necesitan poder ver constantemente todo el entorno que rodea al agente a localizar. Es por ello por lo que el uso de cámaras gran angular y objetivos ojo de pez (*Fish Eye* en ingles), que destacan por un elevado campo de visión (de hasta 180º o superior), es cada vez más común.

Por tanto, los objetivos de la presente investigación son los siguientes:

1. Estudio de la parametrización de puntos en profundidad inversa.
2. Implementación de puntos en profundidad inversa en ORB-SLAM2.
3. Evaluación de la precision, robustez y coste computacional de ORB-SLAM2 con puntos en profundidad inversa.
4. Estudio del modelo de cámaras con objetivos gran angular y ojo de pez.
5. Implementación de ORB-SLAM2 monocular con cámaras gran angular y ojo de pez.
6. Implementación de ORB-SLAM2 estéreo con cámaras gran angular y ojo de pez.
7. Evaluación del sistema con las gafas de realidad aumentada *Meta 2*.

1.4. Herramientas utilizadas

Para el desarrollo del presente proyecto, se ha partido de la versión de ORB-SLAM2 presente en su repositorio oficial (`https://github.com/raulmur/ORB_SLAM2`). En base a ella, se ha utilizado el siguiente listado de herramientas para la realización del proyecto:

- C++: lenguaje de programación de ORB-SLAM2.
- OpenCV: librería de código abierto con estructuras de datos y algoritmos de visión por computador.
- g2o: librería de código abierto para optimización de funciones no lineales usada para resolver el problema de optimización *Bundle Adjustment*.
- MATLAB: lenguaje de programación matemático. Usado para llevar a cabo simulaciones de cámaras gran angular.
- Bash: lenguaje de comandos. Usado para automatizar la ejecución de pruebas y recolección de resultados experimentales.
- Git + GitHub: software y plataforma de desarrollo para el control de versiones.
- Gafas de realidad aumentada *Meta 2* (`http://www.metavision.com`) usadas para la toma de imágenes con cámaras gran angular.
- LaTeX: sistema de composición de textos para la redacción del presente documento.
- Google scholar: motor de búsqueda enfocado en la búsqueda de contenido y literatura científico-académica usado como apoyo a la investigación.

1.5. Estructura del documento

La investigación desarrollada se expone a lo largo de 4 capítulos en el presente documento. El capítulo 2 esta dedicado a presentar conceptos básicos que serán utilizados a lo largo de todo el documento. En el capítulo 3 se presenta el estudio, implementación y evaluación experimental de ORB-SLAM2 con puntos en profundidad inversa, junto con las conclusiones obtenidas. El capítulo 4 esta dedicado al estudio, implementación y resultados de incluir soporte a cámaras gran angular y objetivos ojo de pez en ORB-SLAM2. Finalmente, el capítulo 5 recoge las conclusiones finales de la presente investigación, futuras líneas de trabajo en base a los resultados obtenidos, resultados del aprendizaje y gestión del proyecto.

Los anexos del proyecto recogen información adicional y complementaria al mismo. En el anexo A se recogen los Jacobianos analíticos usados en el *Bundle Adjustment*, tanto para el caso de puntos en profundidad inversa como para cámaras gran angular en el caso monocular y estéreo.

Capítulo 2

Conceptos básicos sobre SLAM

Para facilitar la comprensión del presente documento, a continuación se presentan una serie de conceptos que serán utilizados a lo largo del documento relacionados con diversos elementos utilizados por un sistema de SLAM como ORB-SLAM2.

2.1. ORB-SLAM2

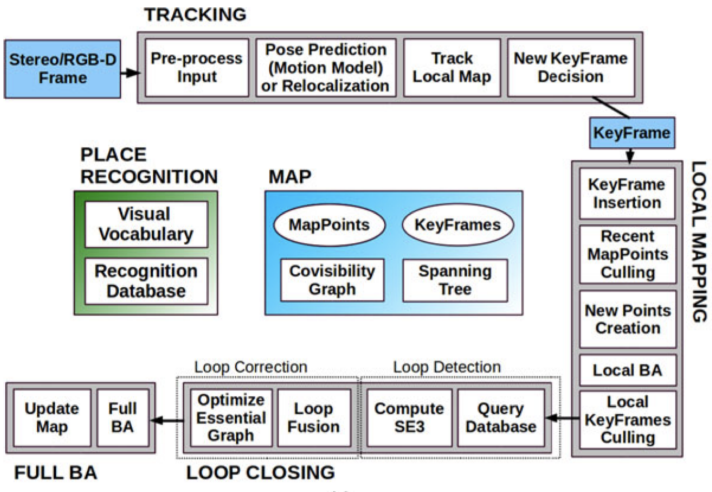
ORB-SLAM2 es un software desarrollado por la Universidad de Zaragoza propuesto como solución al problema de SLAM. Su principal característica es el ser un sistema de SLAM basado en *KeyFrames* (fotogramas especiales de la secuencia seleccionados por aportar gran cantidad de información sobre el entorno) que utiliza el mismo tipo de características para todas las tareas de SLAM.

La estructura de ORB-SLAM2 (Figura 2.1) se divide en 3 hilos de ejecución paralelos que realizan tareas diferentes: el hilo de *tracking*, encargado de estimar la pose de la cámara en cada *frame* del vídeo; el hilo de *local mapping*, encargado de triangular y añadir nuevos puntos al mapa, y de optimizar tanto las posiciones de las cámaras como del mapa 3D construido; y *loop closing*, encargado de detectar y cerrar bucles en la trayectoria de la cámara.

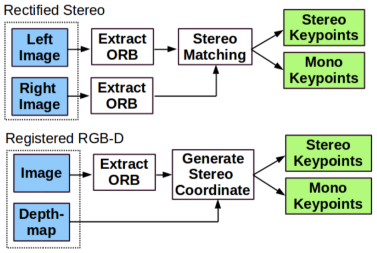
Como la mayoría de sistemas de SLAM visual, ORB-SLAM2 utiliza coordenadas Cartesianas para la representación de los puntos del mapa y el modelo de cámara oscura para la cámara. Partiendo de esta estructura, se han ido realizando modificaciones sobre la misma para implementar puntos en profundidad inversa y soporte para cámaras gran angular.

2.2. Modelo de cámara oscura

ORB-SLAM2 utiliza sensores de visión para obtener información del entorno que rodea a la cámara y poder construir un mapa 3D donde localizar a la misma. Sin



(a) Hilos y módulos del sistema.



(b) Preprocesado de las imágenes de entrada.

Figura 2.1: Estructura general de ORB-SLAM 2. El hilo de *Tracking* preprocesa las imágenes de entrada de manera que el resto del sistema funcione independientemente del tipo de cámara usada.

embargo, es necesario definir un modelo matemático para poder calcular cómo un punto 3D se proyecta a un píxel concreto en la imagen (modelo de proyección) o cómo un píxel en la imagen se transforma en el vector director de la recta que va desde el centro óptico de la cámara al punto observado (modelo inverso de proyección).

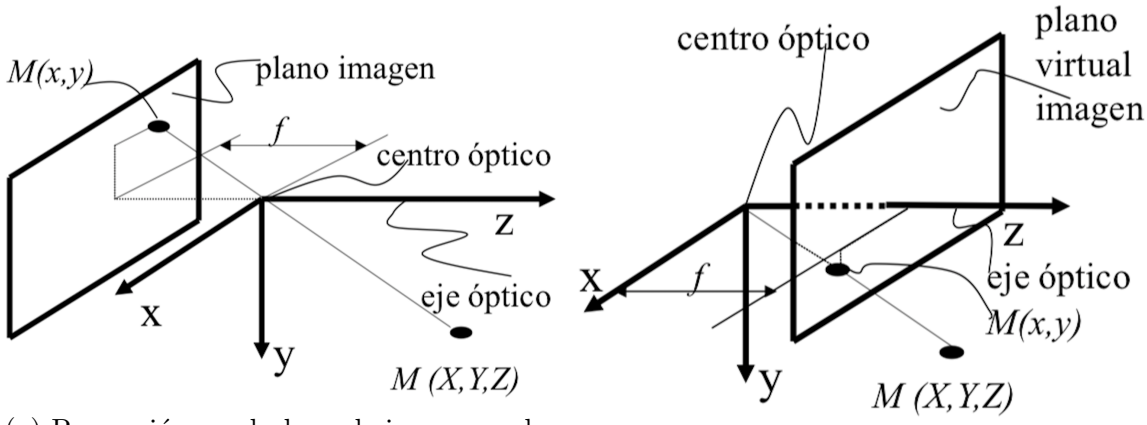
El modelo más usado es el denominado de *cámara oscura* (o *pinhole* en inglés). La razón de su amplio uso es debido a que se trata de un modelo general y simple que modela correctamente una gran cantidad de cámaras presentes en el mercado.

2.2.1. Modelo de proyección pinhole

El modelo de proyección pinhole (Fig. 2.2) es una función $\Pi : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ que transforma un punto 3D $\mathbf{X_c} = \begin{pmatrix} x_c & y_c & z_c \end{pmatrix}^T$ en el sistema de referencia de la cámara a un punto en la imagen de coordenadas (en píxeles) $\mathbf{X_i} = \begin{pmatrix} u & v \end{pmatrix}^T$ tal y como se muestra a continuación:

$$\mathbf{X_i} = \begin{bmatrix} u \\ v \end{bmatrix} = \Pi(\mathbf{X_c}) = \begin{bmatrix} \frac{f}{d_u} \cdot \frac{x_c}{z_c} + u_c \\ \frac{f}{d_v} \cdot \frac{y_c}{z_c} + v_c \end{bmatrix} = \begin{bmatrix} f_u \cdot \frac{x_c}{z_c} + u_c \\ f_v \cdot \frac{y_c}{z_c} + v_c \end{bmatrix} \tag{2.1}$$

siendo f la distancia focal de la cámara, u_c, v_c las coordenadas en la imagen del centro óptico y d_u, d_v es el tamaño de un píxel en la imagen.



(a) Proyección en el plano de imagen real.

(b) Proyección en el plano de imagen virtual.

Figura 2.2: Funcionamiento de una cámara *pinhole*. Nótese que al proyectar un punto en el plano de imagen se obtiene una imagen rotada 180º. Para evitar esto, se considera un plano de imagen virtual por delante del centro óptico de la cámara.

2.2.2. Modelo inverso de proyección pinhole

Se define como modelo inverso de proyección a aquella función $\Pi^{-1} : \mathbb{R}^2 \rightarrow \mathbb{R}^3$ que mapea un punto en la imagen de coordenadas $\mathbf{X_i} = \begin{pmatrix} u & v \end{pmatrix}$ al vector director $\overrightarrow{X_c} = \begin{pmatrix} x_c & y_c & z_c \end{pmatrix}$ del rayo proyectante que une el centro óptico de la cámara con un punto 3D observado $\mathbf{X_c} = \begin{pmatrix} x_c & y_c & z_c \end{pmatrix}$ en el sistema de referencia de la cámara.

El modelo de cámara oscura define el modelo inverso de proyección mediante la siguiente ecuación:

$$\overrightarrow{X_c} = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \Pi^{-1}(\mathbf{x_i}) = \begin{bmatrix} \frac{u - u_c}{f_x} \cdot s \\ \frac{v - v_c}{f_y} \cdot s \\ s \end{bmatrix} \tag{2.2}$$

Nótese que dado un punto en la imagen, no es posible estimar la profundidad a la que se encuentra el punto 3D observado en una sola imagen y que ésta se representa en función de un valor arbitrario s.

2.3. Bundle Adjustment

Todo sensor toma medidas que están alteradas por ruido. Las cámaras no son una excepción y, por tanto, cuando un sistema de SLAM construye un mapa e intenta localizar la cámara en el mismo, debe realizar un proceso de refinamiento tanto de las coordenadas de los puntos como de las posiciones de las cámaras que los observan. Esta optimización se realiza mediante el algoritmo conocido como *Bundle Adjustment* [13],

el cual estima la posición 3D de los puntos y las poses de las cámaras minimizando el error de reproyección.

2.3.1. Definición

Bundle Adjustment optimiza las coordenadas Cartesianas $\mathbf{X}_i \in \mathbb{R}^3$ de un conjunto $\mathcal{P}$ de puntos 3D, y las poses $T_k \in SE(3)$ de un conjunto de *KeyFrames* $\mathcal{K}$, minimizando el error de reproyección $E(k,i)$ de cada punto i observado en un *KeyFrame* k respecto de la coordenada $\mathbf{x}_{ki} \in \mathbb{R}^2$ del punto de interés en la imagen k con la que se ha emparejado. De esta manera, la función a optimizar es:

$$\begin{aligned} \{\mathbf{X}_i, T_k | i \in \mathcal{P}, k \in \mathcal{K}\} &= \underset{\mathbf{X}_i, T_k}{\operatorname{argmin}} \sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{P}} \rho\left(E(k,i)\right) \\ E(k,i) &= \left\| \mathbf{x}_{ki} - \Pi\left(\mathbf{R}_k \mathbf{X}_i + \mathbf{t}_k\right) \right\|^2_{\Sigma_{ki}} \end{aligned} \tag{2.3}$$

donde R_k y t_k es la matriz de rotación y el vector de traslación de T_k, Σ_{ki} es la covarianza del punto de interes $\mathbf{x}_{ki}$ y ρ es la función de coste robusto de *Hubber* [13], utilizada para reducir el impacto de los emparejamientos espúreos en el resultado de la optimización.

2.3.2. Optimización no lineal

Al tratarse de un problema de mínimos cuadrados no lineal, la solución óptima se halla de manera iterativa, calculando incrementos sobre los parámetros iniciales Δx a partir de una estimación inicial de los mismos. Existen diferentes métodos para tal fin. Los mas extendidos en la literatura se presentan a continuación.

Método de Gauss-Newton

Dado un problema de optimización $S : \mathbb{R}^n \to \mathbb{R}$ con un vector de parámetros $\mathbf{x} \in \mathbb{R}^n$ de la forma:

$$\underset{x}{\operatorname{argmin}}\, S(x) = a \cdot f(x)^T \Lambda f(x) \tag{2.4}$$

siendo $a > 0$, $f : \mathbb{R}^n \to \mathbb{R}^m$ doblemente diferenciable y $\Lambda \in \mathbb{R}^{m \times m}$ una matriz simétrica y definida semi-positiva. A pesar de ser un domino específico, se cubren una gran cantidad de problemas, entre ellos el propuesto en (2.3).

El método de Gauss-Newton aproxima la solución iterativamente de manera que con cada iteración, el vector de parámetros x se actualiza resolviendo la *ecuación normal* tal que:

$$\begin{aligned} (J_f^T \Lambda J_f) \Delta x &= -J_f^T \Lambda f \\ x_{new} &= x + \Delta x \end{aligned} \tag{2.5}$$

Es sabido que este método converge rápidamente si la estimación inicial se encuentra cerca de la solución. Sin embargo, si la estimación inicial es mala, el método tiene problemas de convergencia, pudiendo llegar a divergir.

Método de Levenberg-Marquardt

Para solventar los problemas de convergencia del método de Gauss-Newton cuando se está lejos de la solución, el método de Levenberg-Marquardt interpola entre el método de Gauss-Newton y descenso de gradiente alterando la ecuación normal tal que:

$$(J_f^T \Lambda J_f + \mu I)\Delta x = -J_f^T \Lambda f \tag{2.6}$$

Si $\mu \to 0$, Levenberg-Marquardt se aproxima a una actualización de Gauss-Newton. Por otro lado, si $\mu \to \infty$, la actualización tomada por Levenberg-Marquardt se aproxima a una actualización de descenso de gradiente

$$\Delta x = -\alpha J_f^T \Lambda f \tag{2.7}$$

con un valor para el hiperparámetro $\alpha \to 0$ para reducir el error mediante una actualización pequeña.

De esta manera, el método de Levenberg-Marquardt funciona de la siguiente manera: si la nueva actualización $x_{new} = x + \Delta x$ reduce el error, es decir, $S(x + \Delta x) \ll S(x)$, la actualización es aceptada. En este caso, Levenberg-Marquardt asume que nos estamos acercando a la solución, por lo que μ es reducido para que la siguiente actualización se aproxime más a Gauss-Newton. Por el contrario, si $x_{new} = x + \Delta x$ no reduce el error, la actualización es rechazada, tomándose en su lugar un paso en la dirección de descenso de gradiente e incrementándose μ para las consecutivas iteraciones.

Método Dog-Leg

El método de Dog-Leg, al igual que el de Levenberg-Marquardt, interpola entre Gauss-Newton y descenso de gradiente. Sin embargo, y en contraste con Levenberg-Marquardt, la dirección de la actualización no es controlada por un parámetro μ sino que esta implícitamente controlada por una región (Fig. 2.3) de confianza.

Así pues, el método de Dog-Leg construye un modelo cuadrático L que aproxima la función S en el punto x actual. Este modelo L es una aproximación de S dentro de una hiperesfera de radio δ centrada sobre el punto actual. De esta manera, la nueva

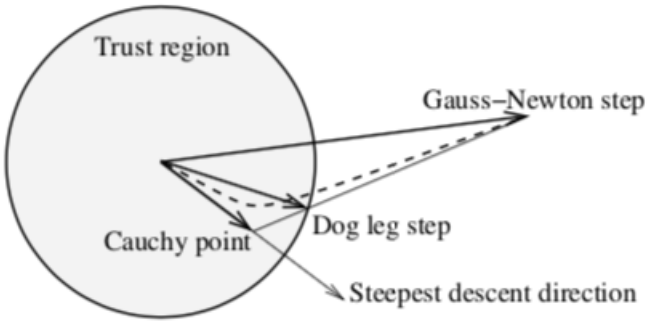


Figura 2.3: Aproximación realiazda por el algoritmo de Dog-Leg de la dirección de actualización (Figura extraída de [7]).

actualización Δx se calcula minimizando el modelo L sobre la región de confianza tal que

$$L(\Delta x) = f(x)^T f(x) + \Delta x^T J^T J \Delta x - (J^T f(x))^T \Delta x \tag{2.8}$$

con la restricción de

$$\underset{\Delta x}{\operatorname{argmin}} L(\Delta x) \text{ sujeto a } \|\Delta x\| \leq \delta \tag{2.9}$$

La selección del radio de la región de confianza δ es crucial para el correcto funcionamiento del método. En la práctica, se calcula en base al éxito de las aproximaciones de iteraciones anteriores: si los anteriores modelos son confiables, el radio δ aumenta, mientras que en caso contrario, el radio se reduce y se vuelve a resolver (2.9).

Capítulo 3

SLAM con puntos en Profundidad Inversa

Los sistemas de SLAM utilizan puntos de interés para crear un mapa 3D donde localizar la cámara. Normalmente, estos puntos de interés se representan con coordenadas Cartesianas como un vector $\mathbf{X} = \begin{pmatrix} x & y & z \end{pmatrix}^T$.

Sin embargo, en [1] se presenta una nueva parametrización para los puntos del mapa denominada *puntos en Profundidad Inversa*. Esta parametrización exhibe ciertas propiedades de interés que la hacen atractiva frente a la parametrización tradicional en coordenadas Cartesianas. A continuación, se define dicha parametrización y se presentan las propiedades principales de la misma.

3.1. Representación en Profundidad Inversa

En esta representación, las coordenadas de un punto están calculadas en el sistema de referencia de la primera cámara que lo observa, llamada a partir de ahora *ancla*, e invertidas por su coordenada z. Por tanto, es necesario guardar junto a las coordenadas en profundidad inversa del punto la pose del *ancla*.

De esta manera, un punto del mapa $\mathbf{X} = \begin{pmatrix} x & y & z \end{pmatrix}^T$ representado en coordenadas Cartesianas en el sistema de referencia del *ancla* es transformado a coordenadas en profundidad inversa $\psi = \begin{pmatrix} x_\psi & y_\psi & z_\psi \end{pmatrix}^T$ mediante la función $\Lambda : \mathbb{R}^3 \to \mathbb{R}^3$:

$$\psi = \Lambda(\mathbf{X}) = \begin{pmatrix} \frac{x}{z} & \frac{y}{z} & \frac{1}{z} \end{pmatrix}^T \tag{3.1}$$

Nótese que la transformación de ψ a $\mathbf{X}$ se puede realizar igualmente usando Λ tal y como sigue:

$$\mathbf{X} = \Lambda(\psi) = \begin{pmatrix} \frac{x_\psi}{z_\psi} & \frac{y_\psi}{z_\psi} & \frac{1}{z_\psi} \end{pmatrix}^T \tag{3.2}$$

3.1.1. Ventajas

En [1] se presenta una comparación entre la parametrización de puntos en coordenadas Cartesianas y en profundidad inversa. De ahí se extrae la principal ventaja de la paramatrización en profundidad inversa: una función de medida mucho más lineal. En términos de optimización matemática, esto es de especial interés ya que cuanto más lineal es la función objetivo a optimizar, mejor condicionado esta el problema.

Por otro lado, la paramatrización en profundidad inversa permite trabajar con puntos con profundidad infinita, es decir, puntos que son observados con paralaje cero. Esto permite inicializar puntos sin requerir que haya sido visto desde diferentes puntos de vista con un paralaje significativo. En el contexto de SLAM, estos puntos en el infinito no aportan información a la hora de estimar la traslación de la cámara pero si son de especial utilidad para estimar la rotación de la misma.

De esta manera, se espera que con esta representación de los puntos del mapa se obtenga una mejora significativa en la estimación de la rotación de la cámara. Además, dada una mayor linealidad en la función de medida, se espera que el algoritmo de *Bundle Adjustment* sea más eficiente, aumentando la convergencia del mismo y obteniendo un incremento general de la precisión del sistema.

3.2. Implementación en ORB-SLAM2

Los cambios realizados en ORB-SLAM2 (Fig. 3.1) para soportar puntos en Profundidad Inversa han sido los siguientes (explicados en detalle a continuación):

1. **Modificación de los Puntos del Mapa**, de manera que coexisten las dos representaciones: coordenadas Cartesianas y en Profundidad Inversa.

2. **Modificación del Bundle Adjustment**, tanto global como local.

3. **Relajación de las condiciones** para admitir nuevos Puntos del Mapa.

Nótese que los cambios realizados están orientados al hilo de *Local Mapping* en el cual se realiza la triangulación de nuevos puntos y optimización del mapa.

De igual manera, nótese la coexistencia de la representación de los puntos con coordenadas Cartesianas y en profundidad inversa. Esto es debido a que en el hilo de *Tracking* se optimiza la pose de la cámara asumiendo los puntos constantes por lo que un cambio en la representación de los puntos no aporta nada. Lo mismo ocurre en el hilo de *Loop Closing*, donde se buscan emparejamientos entre imágenes en base a la bolsa de palabras extraída en cada una.

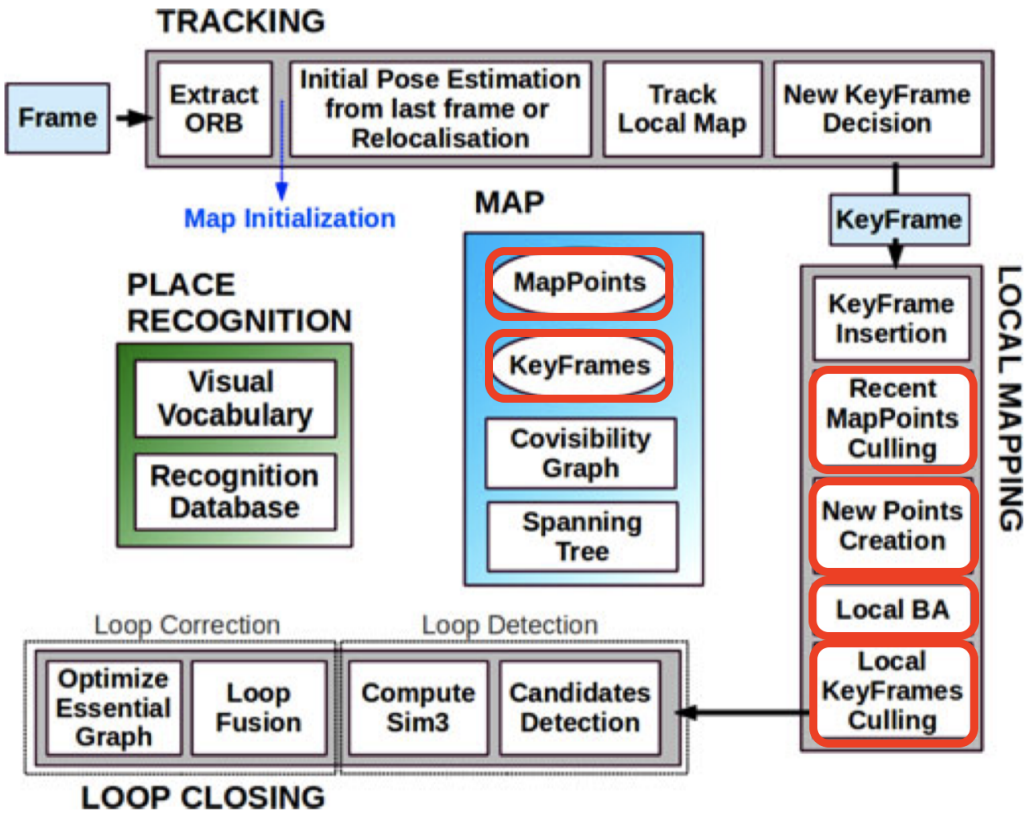


Figura 3.1: En rojo, las modificaciones realizadas sobre ORB-SLAM 2 para introducir puntos en profundidad inversa.

3.2.1. Modificiación de los puntos del mapa

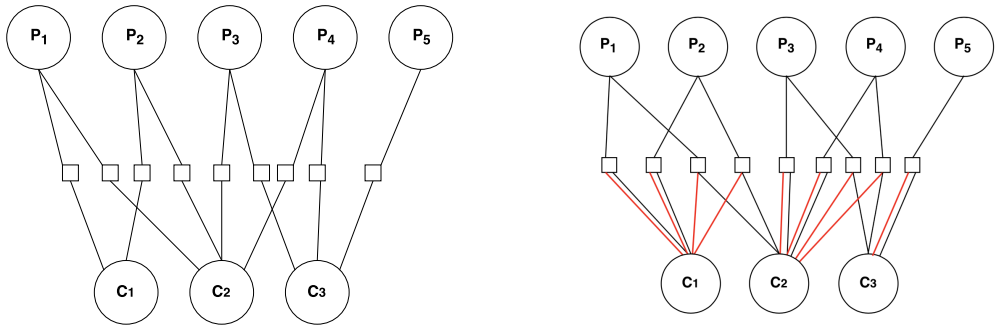
Los puntos del mapa se han modificado internamente de manera que ambas representaciones conviven simultáneamente, propagándose los cambios de una representación a otra de acuerdo a las fórmulas 3.2 y 3.1.

De esta manera, es necesario almacenar las coordenadas de *Profundidad Inversa* y guardar el *KeyFrame* que actúa como ancla de dichas coordenadas. Debido al proceso de eliminación de *KeyFrames* redundantes realizado por ORB-SLAM2, cada vez que un *KeyFrame* es eliminado, es necesario actualizar todos los puntos cuya ancla es el *KeyFrame* eliminado y, por tanto, recalcular las coordenadas en Profundidad Inversa respecto del nuevo ancla.

3.2.2. Modificación del Bundle Adjustment

El *Bundle Adjustment* ha sido modificado de manera que ahora sea capaz de trabajar con puntos en Profundidad Inversa. En ORB-SLAM2, el *Bundle Adjustment* se realiza con la biblioteca externa *g2o* [5] para optimización no lineal basada en hipergrafos. De esta manera, *g2o* codifica las variables a optimizar como vértices del grafo y las restricciones entre variables como aristas. En este caso, las variables a optimizar son las poses de las cámaras y los puntos del mapa, siendo las restricciones la proyección en las imágenes de los puntos del mapa. La figura 3.2b muestra el problema de *Bundle Adjustment* con 5 puntos que han sido observados por 3 cámaras. Cuando se usan coordenadas Cartesianas, cada observación define una arista que une el punto y la cámara que lo observa. En profundidad inversa cada observación corresponde a una hiperarista ternaria, que conecta el punto, su ancla y la cámara que lo está observando. Para implementarlo, ha sido necesario añadir nuevas hiperaristas en *g2o*, de manera que conecten un punto del mapa con su ancla y un *KeyFrame* que observa el punto. Para reducir el coste computacional de la optimización, *g2o* ofrece la posibilidad de introducir los Jacobianos analíticos de las aristas, de manera que no se te tengan que calcular numéricamente. Así pues, los Jacobianos originales para la optimzación de puntos en coordenadas Cartesianas y los nuevos Jacobianos para la optimización de puntos en Profundidad Inversa son los mostrados en A.4.

El hecho de que estas aristas sean ternarias (y no binarias como en la implementacion original), hace que el coste computacional del *Bundle Adjustment* se vea penalizado. En [7] se introduce la idea de que el algoritmo de optimización *Dog-Leg* es computacionalmente más eficiente que el de *Levenberg-Marquardt*. Debido a ello, se ha cambiado el algoritmo de *Levenberg-Marquardt*, usado en la implementación original, por el de *Dog-Leg*.



(a) Bundle Adjustment con puntos con coordenadas Cartesianas.

(b) Bundle Adjustment con puntos en Profundidad Inversa.

Figura 3.2: Representación del problema de Bundle Adjustment como hipergrafo, donde los nodos P_i son puntos y C_j son poses de las cámaras y los cuadrados son las hiperaristas que representan la observación de un punto en una cámara. En (a) se observan las aristas binarias que unen un punto con cada cámara que lo observa. En (b) se observan las hiperaristas ternarias donde se conecta un punto con su ancla y con una cámara que lo observa. Nótese en rojo el brazo de la arista que conecta con el ancla.

3.2.3. Relajación de las condiciones para admitir nuevos Puntos del mapas

Una de las principales motivaciones para aplicar Profundidad Inversa es el poder admitir puntos con muy poco paralaje. De esta manera, puntos muy lejanos o incluso en el infinito son aceptados por el sistema de manera que contribuyen a estimar la rotación de la cámara.

Debido a que en Profundidad Inversa la profundidad infinita es representada con cero, es posible que algunos puntos puedan tener *profundidad negativa*, es decir, son triangulados detrás de la cámara pues el infinito positivo esta "conectado" con el infinito negativo. Por ello, puntos que aparezcan con profundidad inversa negativa pero muy próxima a cero en la representación de Profundidad Inversa son puntos del infinito y, por tanto, de interés para estimar la rotación. Sin embargo, los puntos con profundidad inversa negativa y distante de cero son puntos mal triangulados y no deben ser usados por el sistema. Por ello, se ha establecido un umbral experimental de profundidad inversa mínima de $-0,01$.

3.3. Resultados experimentales

Se ha evaluado la implementación de puntos en Profundidad Inversa sobre ORB-SLAM2 en 2 *datasets* y se ha comparado con la implementación original con puntos representados con coordenadas Cartesianas. Se han ejecutado las evaluaciones sobre un ordenador de sobremesa convencional con una CPU Intel Core i7-3770 @3,49

GHz x 8 con 8GB de memoria RAM. Para evitar el no determinismo inherente a sistemas multi hilo, cada secuencia ha sido ejecutada 10 veces: 5 con la implementación original y 5 con puntos en Profundidad Inversa, mostrándose las medianas. En el caso del dataset TUM, se ha realizado un *Bundle Adjustment* final de 50 iteraciones, mostrándose también los resultados de esta última optimización.

3.3.1. TUM Dataset

El Dataset TUM-D [12] contiene secuencias de interiores obtenidas con un sensor RGB-D agrupadas en diferentes categorías, para evaluar métodos de SLAM bajo diferentes condiciones de textura, iluminación y estructura.

Para la evaluación, se ha usado SLAM monocular usando *Levenberg-Marquardt* como algoritmo de optimización dentro del *Bundle Adjustment*. Los resultados obtenidos se muestran en la tabla 3.1, mostrándose solo aquellas escenas en la que el sistema ha sido capaz de realizar una estimación de la trayectoria de la cámara. Se observa que ambos métodos obtienen tasas de error muy similares.

3.3.2. KITTI Dataset

El KITTI dataset [3] contiene secuencias en estéreo grabadas desde un coche en entornos urbanos y de autopista. El sensor estéreo tiene ∼54cm de linea base y funciona a una tasa de 10Hz con resolución (después de la rectificación) de 1240 x 376 píxeles. La secuencia 01 es de especial interés ya que se trata de un recorrido en autopista, con pocos objetos cercanos que pueden ser rastreados y usados para estimar la traslación de la cámara. Los resultados de la evaluación monocular se muestran en la tabla 3.2 usando *Levenberg-Marquardt* como algoritmo de optimización dentro del *Bundle Adjustment*.

3.3.3. Algoritmo de optimización

Se han hecho pruebas con los 3 algoritmos de optimización (Tabla 3.2) presentados en 2.3.2, comparando el rendimiento en términos de si el sistema es capaz de inicializarse y habiéndose inicializado, si es capaz de procesar toda la escena.

3.3.4. Tiempos de ejecución

Para comprobar el coste computacional en tiempo de la nueva implementación, al final de cada ejecución de las pruebas para el TUM dataset, se ha realizado un *Bundle Adjustment* global sobre todo el mapa obtenido de 50 iteraciones y se ha medido el tiempo medio por iteración (Tabla 3.4). A raíz de los resultados arrojados en la tabla 3.3, solo se han comparado los algoritmos de Levenberg-Marquardt y de Dog-Leg.

Tabla 3.1: Comparación de los RMSE de traslación en el TUM Dataset. Entre paréntesis se muestran los resultados tras realizar un Bundle Adjustment final de 50 iteraciones

	Coordenadas Cartesianas	Profundidad Inversa
Secuencia	Error RMSE (cm)	Error RMSE (cm)
fr1_xyz	**0,88 (0,88)**	0,9 (0,89)
fr1_desk	**1,46 (1.45)**	1,50 (1,48)
fr2_xyz	**0,23 (0,24)**	0,24 (0,25)
fr2_desk	**0,88 (0,83)**	1,11 (1,09)
fr3_sit_halfsph	**1,42 (1,30)**	1,63 (1,64)
fr3_sit_xyz	**0,88 (0,88)**	1,05 (1,05)
fr3_str_tex_far	**0,90 (0,90)**	0,93 (0,91)
fr3_str_tex_near	**1,12 (1,14)**	4,30 (3,94)
fr3_walk_halfsph	1,82 (1.82)	**1,79 (1,79)**

Tabla 3.2: Comparación de los RMSE de traslación y rotación en el KITTI Dataset.

	Coordenadas Cartesianas		Profundidad Inversa	
Secuencia	RMSE Traslación (m)	RMSE Rotación (rad)	RMSE Traslación (m)	RMSE Rotación (rad)
00	**8,712**	**0,030**	9,448	0,042
01	533,808	**1,757**	**312,630**	1,833
02	27,645	**0,274**	**26,550**	0,448
03	**1,020**	0,006	1,737	**0,006**
04	**0,438**	**0,182**	0,702	0,711
05	**4,891**	**0,014**	7,702	0,024
06	16,690	**0,012**	**16,617**	0,013
07	**2,318**	**0,024**	3,826	0,033
08	**50,213**	**0,034**	57,571	0,050
09	**45,560**	0,083	47,908	**0,083**
10	**4,401**	**0,487**	6,823	0,491

Tabla 3.3: Funcionamiento de los diferentes algoritmos de optimización probados en el Bundle Adjustment con ambas parametrizaciones (*CC* para Coordenadas Cartesianas y *PI* para Profundidad Inversa). Una ✗ denota que ORB-SLAM2 se pierde nada mas inicializar la escena, ◯ indica que se procesa una parte significativa de la escena antes de que el sistema se pierda mientras que ✓ indica que la escena ha sido procesada correctamente.

	Gauss-Newton		Levenberg-Marquardt		Dog-Leg	
Escena	*CC*	*PI*	*CC*	*PI*	*CC*	*PI*
fr1_xyz	✗	✓	✓	✓	✓	✓
fr1_desk	✗	✓	✓	✓	✓	✓
fr2_xyz	✗	✓	✓	✓	✗	✓
fr2_desk	✗	✗	✓	✓	✓	✓
fr3_sit_halfsph	✗	✓	✓	✓	✗	✓
fr3_sit_xyz	✗	✓	✓	✓	✗	✓
fr3_str_tex_far	✗	✗	✓	✓	✓	✓
fr3_str_tex_near	✗	✓	✓	✗	✓	✓
fr3_walk_halfsph	✗	✓	✓	✓	✓	✓
00	✗	◯	✓	✓	✗	✓
01	✗	◯	✓	✓	✓	✓
02	✗	◯	✓	✓	✓	✓
03	✗	✓	✓	✓	✓	✓
04	✗	✓	✓	✓	✓	✓
05	✗	◯	✓	✓	✓	✓
06	✗	✓	✓	✓	✓	✓
07	✗	✓	✓	✓	✓	✓
08	✗	◯	✓	✓	◯	✓
09	✗	◯	✓	✓	✓	✓
10	✗	✓	✓	✓	✗	✓

Tabla 3.4: Tiempos de ejecución del *Bundle Adjustment* final de 50 iteraciones.

	Levenberg-Marquardt	Dog-Leg
Coordenadas Cartesianas	0.0150	0.0310
Profundidad Inversa	0.0289	0.0662

3.4. Discusión

En vista a los resultados experimentales obtenidos en 3.3, se concluye que la representación de puntos en profundidad inversa no aporta mejora en entornos de interior, llegando a obtener resultados similares a los obtenidos usando puntos parametrizados con coordenadas Cartesianas. Sin embargo, en entornos de exterior como el presentado en la escena 01 del KITTI dataset se observa cierta mejoría. Esto se debe probablemente al hecho de aceptar y triangular puntos con poco paralaje, mejorando la estimación de la rotación de la cámara. Sin embargo, se trata de una secuencia especialmente complicada para SLAM monocular en la cual la parametrización en profundidad inversa mejora los resultados pero siguen siendo inaceptables.

Además, la tabla 3.3 muestra cómo el algoritmo de Gauss-Newton, el cual depende tanto de la linealidad de la función a optimizar como de la cercanía de la estimación inicial a la solución, funciona significativamente mejor con puntos en profundidad inversa. Ello respalda la idea de que la función de medida para puntos en profundidad inversa es efectivamente más lineal que su contraparte con coordenadas Cartesianas.

Por otro lado, se comprueba de igual manera que el algoritmo de Levenberg-Marquardt es computacionalmente mas eficiente, independientemente de la representación de los puntos. Adicionalmente, se ha comprobado que el *Bundle Adjusment* con puntos en profundidad inversa, independientemente del algoritmo de optimización usado, es el doble de costoso computacionalmente que su contraparte con puntos en coordenadas Cartesianas.

Finalmente, como conclusión global, se extrae que ambas representaciones de puntos, tanto en profundidad inversa como con coordenadas Cartesianas, convergen a soluciones similares. Si bien la linealidad de profundidad inversa es una propiedad interesante, el algoritmo de *Bundle Adjustment* no es tan sensible a la linealidad de la función a optimizar a diferencia de sus métodos predecesores como el filtro de Kalman extendido, donde la linealidad era un aspecto crítico.

Capítulo 4

SLAM con cámaras gran angular

Las cámaras gran angular se caracterizan por tener un gran campo de visión (FOV). Dependiendo del mismo, se pueden clasificar los diferentes objetivos entre gran angular (FOV de hasta 120º) u ojo de pez (FOV de hasta 180º o superior). Esta característica las hace especialmente atractivas para aplicaciones en las que se requiera ver constantemente el entorno que rodea a la cámara, como por ejemplo, cámaras en automóviles o gafas de realidad aumentada.

En el presente capítulo, se va a asumir un objetivo ojo de pez con un FOV de $\sim 180^{o}$ siguiendo el modelo *Kannala-Brandt* [4]. Se ha seleccionado este modelo por ser el que la librería OpenCV implementa además de haber muchas herramientas para calibrar cámaras siguiendo este mismo modelo.

4.1. Modelo de cámara gran angular

4.1.1. ¿Por qué no funciona el modelo de cámara oscura?

Matemáticamente, el modelo de cámara oscura presenta una singularidad con puntos que se acercan a 90º con respecto al eje óptico de la misma (es decir, puntos cuya profundidad relativa a la cámara se aproxima a cero). De hecho, el modelo de cámara oscura es adecuado únicamente para cámaras con FOV de hasta 100º, momento a partir del cual la proyección de un objeto en la imagen aumenta significativamente de tamaño.

De esta manera, supóngase un punto de coordenadas en el sistema de referencia de la cámara con las siguientes coordenadas $\mathbf{X_c} = \begin{pmatrix} x_c & y_c & 0 \end{pmatrix}$. De acuerdo con el modelo de proyección presentado en 2.2, el resultado de proyectar el punto $\mathbf{x_c}$ en la imagen seria el siguiente:

(a) Imagen tomada con una cámara *pinhole* con un FOV de $\sim 80^{o}$. Nótese que las rectas aparecen rectas.

(b) Imagen tomada con una cámara gran angular con un FOV de $\sim 180^{o}$. Nótese la presencia de una fuerte distorsión de barril.

Figura 4.1: Comparación de imagenes de la misma escena tomadas por una cámara *pinhole* y por una cámara gran angular. Nótese la diferencia en el campo de visión.

$$\Pi(\mathbf{x_c}) = \begin{bmatrix} f_x \cdot \frac{x}{z} + u_c \\ f_y \cdot \frac{y}{z} + v_c \end{bmatrix} = \begin{bmatrix} f_x \cdot \frac{x}{0} + u_c \\ f_y \cdot \frac{y}{0} + v_c \end{bmatrix} = \begin{bmatrix} \infty \\ \infty \end{bmatrix} \tag{4.1}$$

4.1.2. Modelo de proyección gran angular

Dado un punto de coordenadas $\mathbf{X_c} = \begin{pmatrix} x_c & y_c & z_c \end{pmatrix}^T$ en el sistema de referencia de la cámara, se pueden obtener el vector director en coordenadas polares $\Phi = \begin{pmatrix} \theta, \varphi \end{pmatrix}^T$ del rayo proyectante que une $\mathbf{X_c}$ con el centro óptico de la cámara (Fig. 4.2) mediante la función $\mathcal{R} : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ definida cómo:

$$\Phi = \begin{pmatrix} \theta \\ \varphi \end{pmatrix} = \mathcal{R}(\mathbf{X}_c) = \begin{pmatrix} \arctan\left(\sqrt{\frac{x_c^2 + y_c^2}{z_c^2}}\right) \\ \arctan\left(\frac{y_c}{x_c}\right) \end{pmatrix} \tag{4.2}$$

que mapea un punto $\mathbf{X_c}$ con las coordenadas polares de su rayo proyectante Φ. En las cámaras *gran angular*, la distorsión se puede definir en función de θ mediante la siguiente ecuación:

$$\theta_d = \theta(1 + k_1\theta^2 + k_2\theta^4 + k_3\theta^6 + k_4\theta^8) \tag{4.3}$$

Donde k_1, k_2, k_3, k_4 son los parámetros de distorsión de la lente ojo de pez. De esta manera, el modelo de proyección para objetivos ojo de pez teniendo en cuenta la distorsión inducida por la lente que mapea un punto 3D $\mathbf{X_c}$ a un punto en la imagen

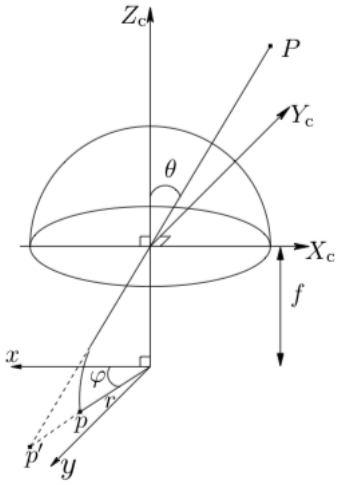


Figura 4.2: Proyección de un punto en una cámara gran angular (imagen extraída de [4]).

las coordenadas en píxeles $\mathbf{X_i} = \begin{pmatrix} u_i \\ v_i \end{pmatrix}$ es:

$$\mathbf{X_i} = \begin{pmatrix} u_i \\ v_i \end{pmatrix} = \Pi_d(\mathbf{X_c}) = \mathcal{A} \circ \mathcal{F}_d \circ \mathcal{R} \tag{4.4}$$

donde la función $\mathcal{F}_d : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ mapea el rayo entrante Φ a un punto con coordenadas $\boldsymbol{X}_d = \begin{pmatrix} x_d \\ y_d \end{pmatrix}$ en la retina normalizada:

$$\boldsymbol{X}_d = \begin{pmatrix} x_d \\ y_d \end{pmatrix} = \mathcal{F}_d(\Phi) = \theta_d \begin{pmatrix} \cos(\varphi) \\ \sin(\varphi) \end{pmatrix} \tag{4.5}$$

y $\mathcal{A} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ transforma las coordenadas en la retina normalizada a coordenadas en la imagen real $\mathbf{X_i}$:

$$\mathbf{X_i} = \begin{pmatrix} u_i \\ v_i \end{pmatrix} = \mathcal{A}(\boldsymbol{X}_d) = \begin{bmatrix} f_u & 0 \\ 0 & f_v \end{bmatrix} \begin{pmatrix} x_d \\ y_d \end{pmatrix} + \begin{pmatrix} c_x \\ c_y \end{pmatrix} \tag{4.6}$$

donde $\begin{pmatrix} c_x, c_y \end{pmatrix}^T$ son las coordenadas del centro óptico y f_x, f_y la distancia focal en píxeles.

4.1.3. Modelo inverso de proyección gran angular

El modelo inverso de proyección que transforma un punto en la imagen $\mathbf{X_i} = \begin{pmatrix} u_i, v_i \end{pmatrix}^T$ al vector director $\overrightarrow{X_c} = \begin{pmatrix} x_c & y_c & z_c \end{pmatrix}^T$ del rayo proyectante que une el centro óptico de la cámara con un punto 3D observado $\mathbf{X_c} = \begin{pmatrix} x_c & y_c & z_c \end{pmatrix}^T$ en el sistema de referencia de la cámara es:

$$\overrightarrow{X_c} = \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} = \Pi_d^{-1}(\mathbf{X_i}) = \mathcal{R}^{-1} \circ \mathcal{F}_d^{-1} \circ \mathcal{A}^{-1} \tag{4.7}$$

Teniendo que:

$$\boldsymbol{X}_d = \begin{pmatrix} x_d \\ y_d \end{pmatrix} = \mathcal{A}^{-1}(\mathbf{X_i}) = \begin{bmatrix} \frac{1}{f_x} & 0 \\ 0 & \frac{1}{f_y} \end{bmatrix} \left(\begin{pmatrix} u_i \\ v_i \end{pmatrix} - \begin{pmatrix} c_x \\ c_y \end{pmatrix} \right) \tag{4.8}$$

Los valores de φ y θ_d se obtienen de la siguiente manera:

$$\varphi = atan\left(\frac{y_d}{x_d}\right) \quad \text{y} \quad \theta_d = +\sqrt{x_d^2 + y_d^2} \tag{4.9}$$

Para finalmente obtener el valor de θ, que requiere la resolución de un polinomio de grado 9:

$$\theta_d = \theta(1 + k_1\theta^2 + k_2\theta^4 + k_3\theta^6 + k_4\theta^8) \tag{4.10}$$

el cual puede ser resuelto mediante métodos numéricos como el método de Newton. Finalmente, a partir de φ y θ se obtienen las coordenadas del vector director $\overrightarrow{X_c}$ del rayo proyectante de la siguiente manera:

$$\overrightarrow{X_c} = \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} = \begin{pmatrix} \sin(\theta) \cdot \cos(\varphi) \cdot s \\ \sin(\theta) \cdot \sin(\varphi) \cdot s \\ \cos(\theta) \cdot s \end{pmatrix} \tag{4.11}$$

Nótese que, al igual que en el modelo inverso de proyección para cámaras *pinhole* presentado en 2.2.2, no es posible estimar la profundidad a la que se encuentra el punto 3D observado en una sola imagen y que esta se representa en función de un valor arbitrario s.

4.1.4. Desdistorsión de imágenes gran angular

Las imágenes tomadas por una cámara con objetivo ojo de pez pueden ser desdistorsionadas y transformadas de manera que sigan el modelo de cámara oscura. Sin embargo, este proceso hace que se pierda la principal propiedad de las cámaras gran angular: el campo de visión.

De esta manera, el proceso de desdistorsión toma un punto $\mathbf{x_d} = \begin{bmatrix} u_d & v_d \end{bmatrix}^T$ en una imagen tomada por una cámara gran angular y lo transforma a un punto de coordenadas $\mathbf{x} = \begin{bmatrix} u & v \end{bmatrix}^T$ siguiendo el modelo de cámara oscura. Esto se consigue aplicando el modelo de proyección inverso gran angular Π_d^{-1} (Eq. 4.7) para obtener la dirección del rayo proyectante y posteriormente, proyectarlo usando el modelo de proyección de cámara oscura Π (Eq. 2.2). Matemáticamente, se define como una función $\mathcal{D} : \mathbb{R}^2 \to \mathbb{R}^2$ tal y como sigue:

$$\mathcal{D}(\mathbf{x_d}) = \Pi \circ \Pi_d^{-1} = \mathbf{x} = \begin{bmatrix} u \\ v \end{bmatrix} \tag{4.12}$$

Nótese que al intervenir el modelo de proyección *pinhole*, es necesario definir unos parámetros intrínsecos para la cámara *pinhole* resultante (Fig. 4.3). Si se toman unas nuevas distancias focales grandes, se pierde el amplio angulo de visión. Por otro lado, si se toman nuevas distancias focales con valores pequeños, se pierde resolución en el centro de la imagen a la vez que los objetos en los laterales de la imagen ven incrementado su tamaño.

4.2. ORB-SLAM2 monocular con cámaras gran angular

La inclusión de soporte a cámaras con objetivo de pez en ORB-SLAM2 monocular puede ser realizado siguiendo 2 aproximaciones diferentes:

- Usar ORB-SLAM2 con imágenes desdistorsionadas mediante el método presentado en 4.1.4, añadiendo una nueva etapa en el preprocesamiento de imágenes.
- Usar las imágenes originales tomadas por la cámara, modificando ORB-SLAM2 incluyendo el nuevo modelo de cámara.

La primera aproximación tiene los inconvenientes presentados en 4.1.4, los cuales afectan negativamente a la extracción de puntos ORB y por tanto, a la robustez y precisión del sistema además de perder el amplio FOV característico de las cámaras con objetivo ojo de pez.

La segunda aproximación tiene como único inconveniente la necesidad de realizar modificaciones en profundidad a ORB-SLAM2, por lo que resulta más atractiva. De esta manera, se ha seleccionado esta aproximación, siendo los cambios realizados en ORB-SLAM2 para añadir soporte a cámaras gran angular y con objetivo ojo de pez (Fig. 4.4) los siguientes:

- Hilo de **Tracking**:
  - Inicialización del mapa (presentado en 4.2.1).
  - Cálculo del mapa local (presentado en 4.2.2).
  - Inserción de *KeyFrames* (presentado en 4.2.3).

(a) $f_x = 229, f_y = 220$.

(b) $f_x = 100, f_y = 100$

Figura 4.3: Imagen 4.1 desdistorsionada usando diferentes valores para las nuevas distancias focales f_x, f_y y con la misma resolución en píxeles. Nótese que conforme se alejan del centro óptico, los objetos ven aumentado su tamaño.

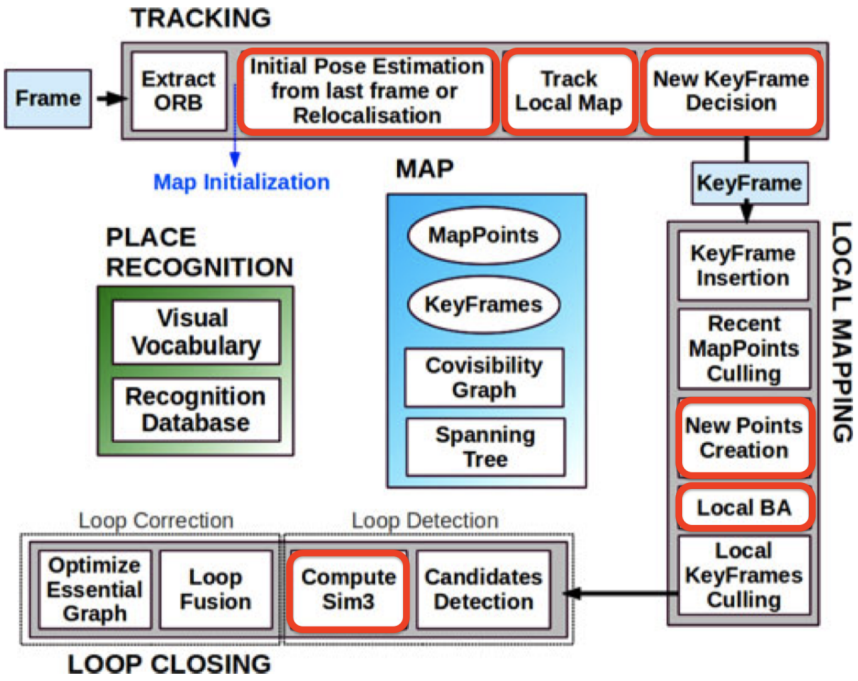


Figura 4.4: En rojo, las modificaciones realizadas sobre ORB-SLAM2 para introducir soporte a cámaras con objetivo ojo de pez.

- Relocalización (presentado en 4.2.4)

– Hilo de **Local Mapping**:

- Creación de puntos (presentado en 4.2.5).
- Bundle Adjustment (Jacobianos analíticos presentados en A).

– Hilo de **Loop Closing**:

- Cálculo de la transformación entre 2 *KeyFrames* candidatos a cerrar un bucle de manera que siga el modelo de cámara presentado en [4].

4.2.1. Inicialización del mapa

El objetivo de inicializar el mapa es el de calcular la pose relativa entre dos *Frames* (considerando como *Frame* aquella estructura de datos usada por ORB-SLAM2 que almacena información relativa a una imagen como la pose de la cámara o los puntos ORB extraídos) de manera que se pueda triangular un conjunto inicial de puntos del mapa. Este proceso debe ser independiente de la geometría de la escena y no debería requerir de la intervención del usuario para seleccionar los dos *Frames* iniciales a partir de los cuales se realizara todo el proceso de inicialización.

Los pasos seguidos en originalmente en ORB-SLAM2 monocular son los siguientes:

1. Extraer puntos ORB en el *Frame* actual F_a y buscar emparejamientos $\mathbf{x}_a \leftrightarrow \mathbf{x}_r$ con los puntos ORB extraídos en el *Frame* de referencia F_r.

2. Computar paralelamente usando *RANSAC* dos modelos geométricos que expliquen la escena: una *homografía* $\mathbf{H}_{ar}$ y una matriz fundamental $\mathbf{F}_{ar}$:

$$\mathbf{x}_a = \mathbf{H}_{ar}\mathbf{x}_r \tag{4.13}$$

$$\mathbf{x}_c^T\mathbf{F}_{ar}\mathbf{x}_r = 0 \tag{4.14}$$

   A cada modelo se le da una puntuación S_H y S_F en función de los errores de reproyección al aplicar el modelo sobre los dos *Frames*. Más detalles en [9].

3. Selección del modelo adecuado: si una escena es planar, la homografía representa correctamente la escena. Sin embargo, si se trata de una escena no planar, la escena esta mejor representada por una matriz fundamental. Para la correcta selección, se usa la siguiente heurística:

$$R_H = \frac{S_H}{S_H + S_F} \tag{4.15}$$

seleccionando la homografía si $R_H > 0{,}45$. En caso contrario, se selecciona la matriz fundamental.

4. Una vez seleccionado el modelo, se reconstruye finalmente el entorno que ven los dos *Frames* y se recupera el movimiento relativo entre ambos, inicializando así el mapa.

Debido a que, tanto la homografía como la matriz fundamental de las eq. 4.13 y 4.14 asumen un modelo de cámara oscura, la inicialización para cámaras gran angular se ha realizado usando el proceso de desdistorsión presentado en 4.1.4 para desdistorsionar los puntos de interés extraídos en los *Frames* y que sigan un modelo de cámara oscura. Esta es la única parte de la implementación en la que ha sido necesario desdistorsionar los puntos.

4.2.2. Cálculo del mapa local

Para el seguimiento de la cámara, ORB-SLAM2 crea un mapa local, compuesto por un conjunto reducido de *KeyFrames* (*Frames* en los cuales se observan nuevos puntos de interés que no están siendo seguidos por el sistema) y puntos del mapa, sobre el cual se realiza el seguimiento de la pose del *Frame* actual ademas de realizar el *Bundle Adjustment*. La razón por la que se crea este mapa local es el de reducir el coste de estimar la pose de la cámara y el de el*Bundle Adjustment* al trabajar con un mapa mas pequeño.

La selección de los *KeyFrames* y puntos del mapa que conforman el mapa local se ciñe a las siguientes reglas:

1. Se añaden al mapa local todos los puntos del mapa $\mathcal{M}$ que observa el *frame* actual.

2. Se añade al mapa local el conjunto de *KeyFrames* $\mathcal{K}_1$ que observan algún punto de $\mathcal{M}$.

3. Se añade el conjunto de *KeyFrames* $\mathcal{K}_2$, los cuales son covisibles con los *KeyFrames* del conjunto $\mathcal{K}_1$ con vecindad 10.

Sin embargo, el hecho del amplio FOV de las cámaras con objetivo ojo de pez provoca que el mapa local crezca hasta ser del mismo tamaño que el mapa global

debido a que es mas fácil que un mismo punto sea visto desde muchos más *KeyFrames*. Ello provoca que el mapa local pierda su utilidad y el coste de estimar la pose de la cámara y del *Bundle Adjustment* crezca con el tamaño del mapa global.

Para solventar esto, se han tomado las siguientes medidas (Fig. 4.5):

- Se ha modificado la regla (2) de la creación del mapa local: ahora solo se añaden al mapa local el conjunto de *KeyFrames* $\mathcal{K}_1$ que observan como mínimo **20 puntos** de $\mathcal{M}$.
- Se ha modificado la regla (3) de la creación del mapa local: ahora solo se añaden al mapa local el conjunto de *KeyFrames* $\mathcal{K}_2$, los cuales son covisibles con los *KeyFrames* del conjunto $\mathcal{K}_1$ con **vecindad 2**.

4.2.3. Insercción de *KeyFrames*

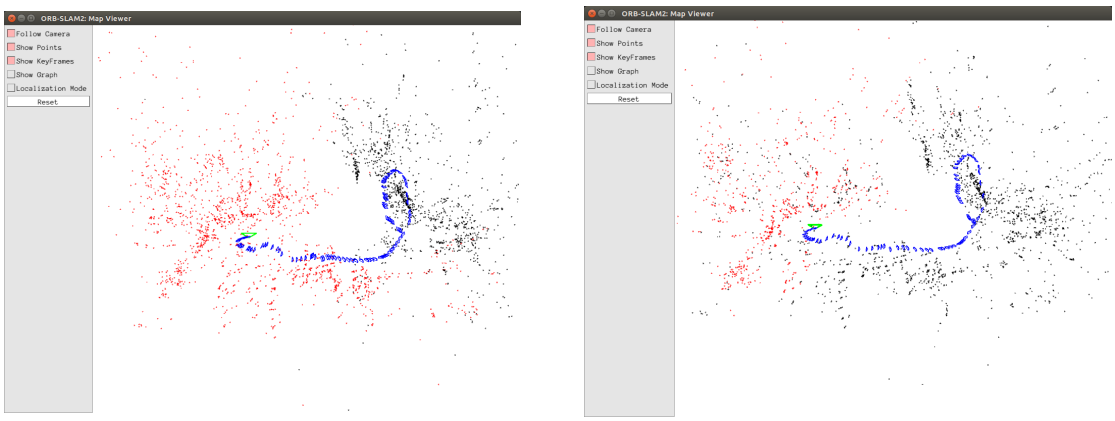
La inserción de nuevos *KeyFrames* al mapa se realiza a medida que se explora el entorno bajo demanda del sistema para asegurar un seguimiento de la cámara estable y robusto. Las condiciones para que un *Frame* sea introducido en el mapa como *KeyFrame* son las siguientes:

1. Que hayan pasado más de 20 *Frames* desde la ultima relocalización de la cámara.
2. El hilo de *Local Mapping* esta en *idle* o que hayan pasado mas de 20 *Frames* desde la inserción del ultimo *KeyFrame*.
3. Que el numero de puntos seguidos por el *Frame* actual sea menor que el 90 % de los seguidos en el ultimo *KeyFrame*.

De nuevo, el amplio FOV de las cámaras con objetivo ojo de pez provoca que entre diferentes *KeyFrames* se observen mayor cantidad de puntos del mapa en común. Por tanto, se ha modificado la regla (3) para la inserción de *KeyFrames*: ahora se insertará un *KeyFrame* si el numero de puntos del mapa que sigue el actual *Frame* es menor que el **80 %** de los seguidos por el ultimo *KeyFrame* insertado.

4.2.4. Relocalización

En un sistema de SLAM, debido a diversos fenómenos, tales como la oclusión total o parcial de la cámara o giros bruscos, el seguimiento de la misma puede fallar, dando como resultado que no se pueda estimar la pose del *Frame* actual ni de posteriores. En estos casos, se dice que el seguimiento se ha *perdido*.



(a) En rojo, los puntos que forman el mapa local sin limitar el tamaño del mismo con cámaras gran angular.

(b) En rojo, los puntos que forman el mapa local aplicando las restricciones al tamaño del mismo.

Figura 4.5: Comparación entre los mapas locales con cámaras gran angular con y sin restricción sobre el tamaño del mismo.

Sin embargo, si estando la cámara perdida, visita una zona ya mapeada con antelación, es posible relocalizarla de nuevo en el mapa, recuperando el seguimiento de la misma. Para ello, se realizan potenciales emparejamientos entre los puntos de interés extraídos en el *Frame* actual con puntos del mapa de manera que se pueda recuperar la pose de la cámara. Este problema es el llamado *Perspective-n-Point* (PNP).

Existen muchos algoritmos que resuelven el problema de PNP. Uno de los más extendidos y usado en la versión original de ORB-SLAM2 es el ePNP [6]. Sin embargo, este algoritmo asume un modelo de cámara oscura, por lo que no es adecuado para trabajar con objetivos gran angular.

En su lugar, se ha usado el algoritmo de MLPnP[1] [14], el cual es independiente del modelo de cámara ya que trabaja con los rayos proyectantes resultados de aplicar el modelo de proyección inverso de la cámara. De esta manera, se usa el algoritmo de MLPnP junto con RANSAC para relocalizar la cámara de manera robusta.

4.2.5. Creación de puntos

Conforme la cámara explora el entorno, es necesario triangular nuevos puntos para asegurar un seguimiento de la cámara estable y robusto. Para realizar dicha triangulación, un punto tiene que ser observado por dos *KeyFrames* diferentes en los cuales se han emparejado dos puntos ORB. Asumiendo un modelo de cámara oscura, se puede aplicar la restricción epipolar para hacer este proceso mas robusto. Sin embargo, con objetivos gran angular, dicha linea epipolar se proyecta como una curva, aumentando el coste de aplicar dicha restricción.

[1] `https://github.com/urbste/opengv`

Por ello, para la triangulación de nuevos puntos usando cámaras *gran angular* (Algoritmo 1) se ignora la restricción epipolar. Sin embargo, para hacer este proceso robusto, se aplican las siguientes comprobaciones para decidir si el emparejamiento de puntos ORB y el punto triangulado es correcto o no:

- Comprobación del paralaje entre los rayos proyectantes calculados a partir de los puntos ORB en ambos *KeyFrames*.
- Que el punto triangulado a partir de ambos rayos proyectantes esté por delante de las cámaras.
- Que el error de reproyección al proyectar el punto 3D triangulado sobre los *KeyFrames* no supere el valor de la distribución χ^2 con 2 grados de libertad y una confianza del 95 %.

4.2.6. Resultados

Para probar el funcionamiento de la implementación realizada de ORB-SLAM2 monocular con cámaras gran angular, se han tomado diferentes secuencias con las gafas de AR Meta 2 en el Lab 1.7 del edificio Ada Byron de la Escuela de Ingeniería y Arquitectura de la Universidad de Zaragoza.

En el presente documento se presentan 3 secuencias diferentes. Una primera (Figuras 4.6 y 4.7) en las cuales se han posicionado 3 cajas formando ángulos rectos. El objetivo de esta secuencia es comprobar si el sistema es capaz de inicializar correctamente con una composición sencilla y que el mapa construido es correcto.

En la segunda secuencia (Figuras 4.8, 4.9 y 4.10), la cámara ha seguido una trayectoria circular. El principal objetivo de esta secuencia es comprobar si la trayectoria de la cámara estimada es correcta a la vez que el sistema es capaz de detectar y cerrar bucles.

Finalmente, en la tercera secuencia (Figuras 4.11), se han tomado imágenes desde las dos cámaras gran angular de las gafas de realidad aumentada para comparar los resultados con la implementación en estéreo. De esta manera, se ha ejecutado ORB-SLAM2 gran angular con las imágenes de ambas cámaras.

Los resultados muestran cómo el sistema inicializa y es capaz de realizar un seguimiento de la cámara estable y robusto a una frecuencia de vídeo de 30Hz, siendo el tiempo medio de seguimiento de un *Frame* de $24,64ms$.

Algorithm 1: Algoritmo para la triangulación de puntos con cámaras con objetivo ojo de pez. Nótese que en el presente documento, el producto de $T \in SE(3)$ siendo una matriz de transformación homogénea por un punto $x \in \mathbb{R}^3$ se denota como $T \cdot x = Rx + t$ siendo R y t la matriz de rotación y el vector de traslación de T respectivamente.

Data: p_1: punto en la imagen 1, p_2: punto en la imagen 2
T_{wc1}: pose de la cámara 1, T_{wc2}: pose de la cámara 2
Result: Punto 3D triangulado a partir de las observaciones

{Reproyección de los puntos en la imagen a rayos proyectantes en el marco del mundo usando el modelo inverso de proyección gran angular Π_d^{-1}} ;
$\overrightarrow{X}_{w1} := T_{wc1} \cdot \Pi_d^{-1}(p_1)$;
$\overrightarrow{X}_{w2} := T_{wc2} \cdot \Pi_d^{-1}(p_2)$;

{Comprobar el paralaje entre los vectores directores} ;
if *parallax*$(r_{w1}, r_{w2}) < 0{,}9998$ **then**
  return fallo;

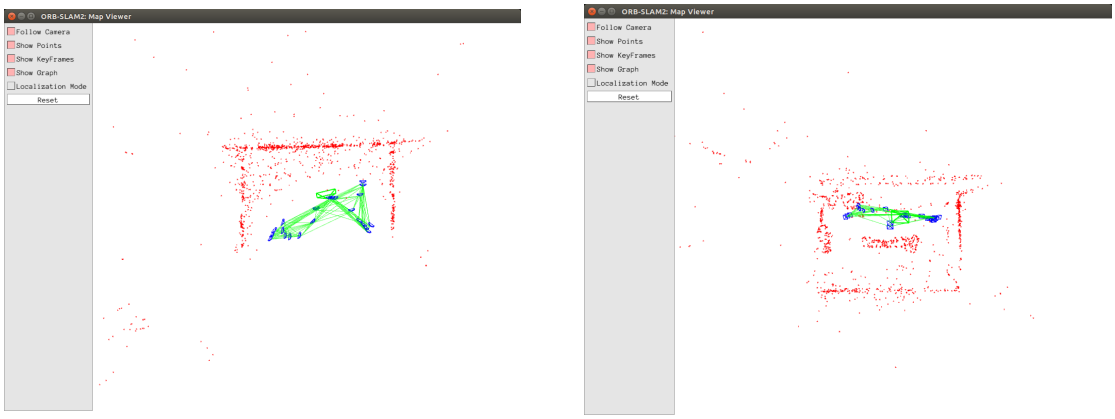
{Triangular el punto 3D a partir de los rayos} ;
$p3D_w := triangulate(r_{w1}, r_{w2})$;

{Comprobar que el punto triangulado está por delante de las cámaras} ;
$p3D_{c1} := T_{wc1}^{-1} \cdot p3D_w$;
$p3D_{c2} := T_{wc2}^{-1} \cdot p3D_w$;
if $p3D_{c1}.z < 0 \quad || \quad p3D_{c2}.z < 0$ **then**
  return fallo;

{Comprobar el error de reproyección en ambas cámaras} ;
$err_1 := projectionError(\Pi_d(p3D_{c1}), p_1)$;
$err_2 := projectionError(\Pi_d(p3D_{c2}), p_2)$;
if $err_1 > 5{,}99 \quad || \quad err_2 > 5{,}99$ **then**
  return fallo;

return $p3D_w$;

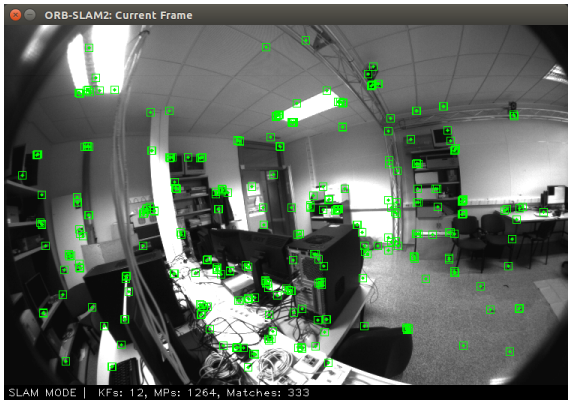
Figura 4.6: Extructura de la primera secuencia tomada con 3 cajas formando 3 planos.



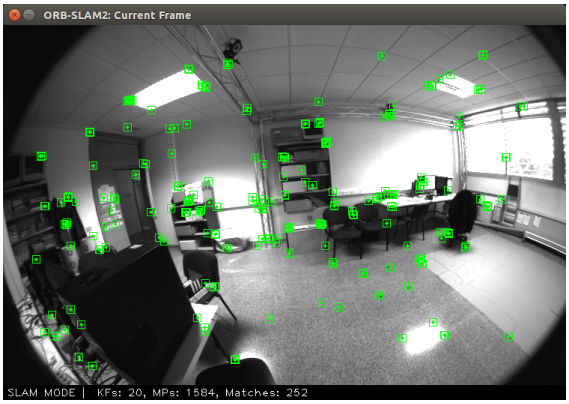
(a) Vista en planta del mapa.

(b) Vista en alzado del mapa.

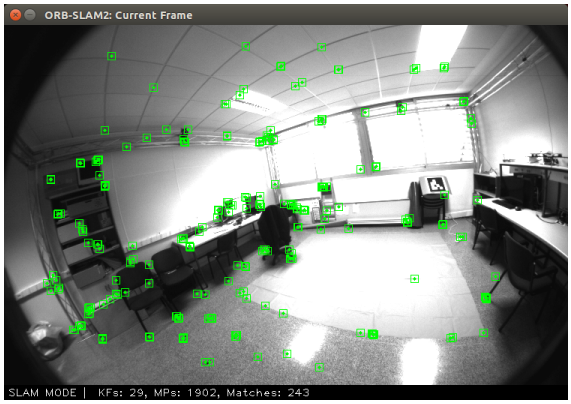
Figura 4.7: Mapa creado a partir de la primera secuencia mostrada en 4.6.



(a) Primer *Frame* de la secuencia despues de inicializar el sistema.

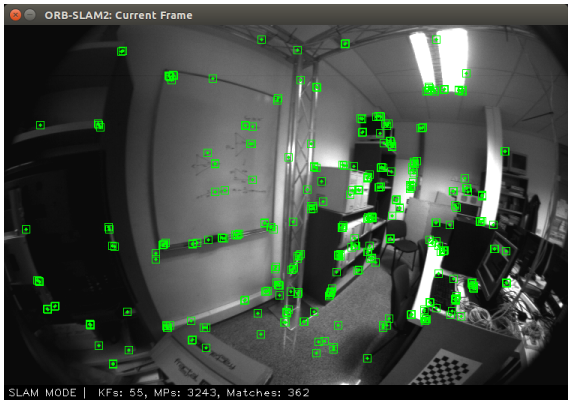


(b) *Frame* número 100.



(c) *Frame* número 200.

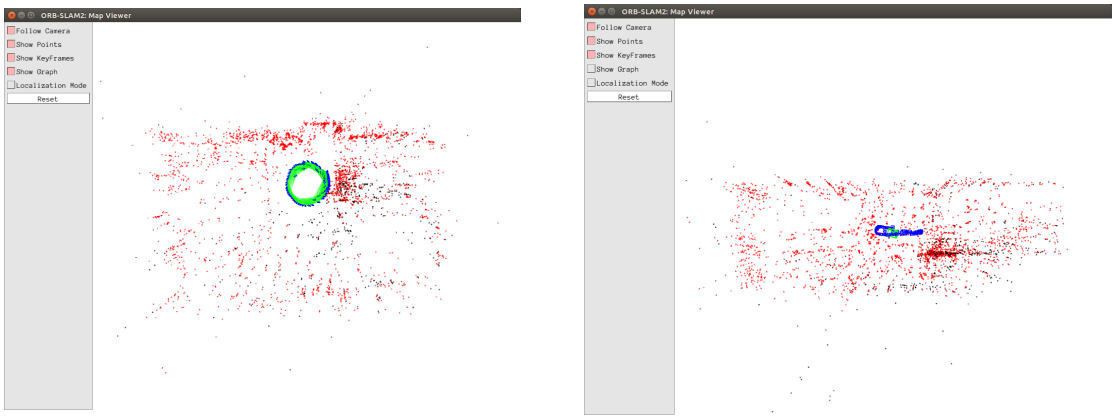
(d) *Frame* número 300.



(e) *Frame* número 400.

(f) *Frame* número 500. En este *Frame*, ORB-SLAM2 detecta un bucle.

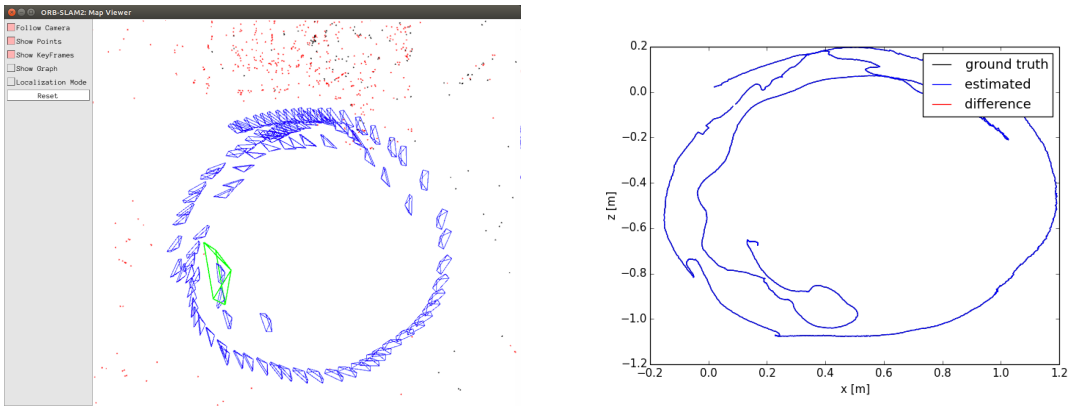
Figura 4.8: Diferentes *Frames* de la segunda secuencia.



(a) Vista en planta del mapa generado.

(b) Vista en alzado del mapa generado.

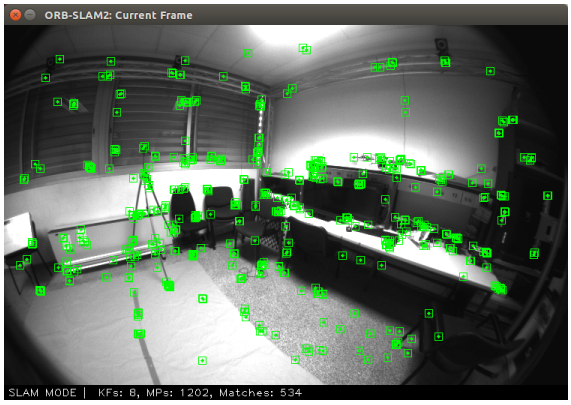
Figura 4.9: Mapa generado por ORB-SLAM2 de la segunda secuencia. Nótese la forma rectangular del laboratorio.



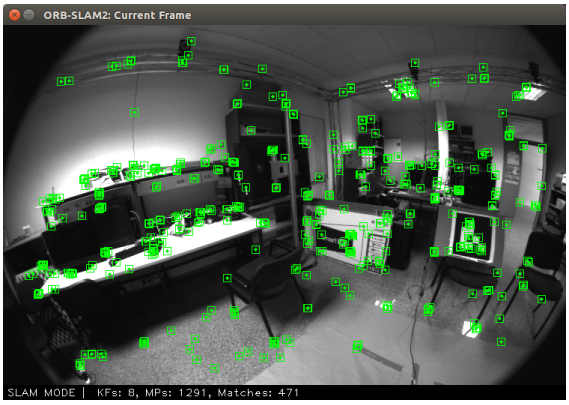
(a) *KeyFrames* generados por ORB-SLAM 2 en el mapa.

(b) Trayectoria de cada uno de los *Frames* procesados por ORB-SLAM2.

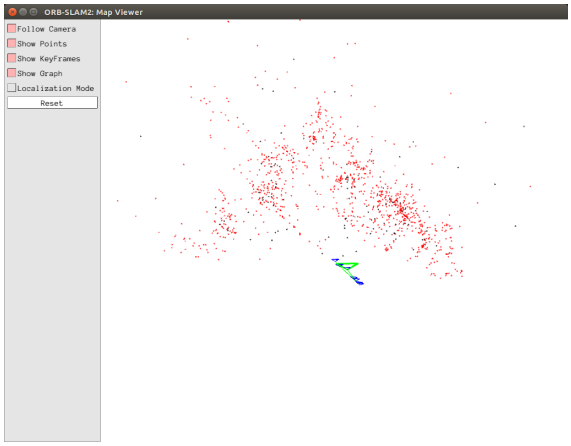
Figura 4.10: Trayectoria generada por ORB-SLAM2 de la segunda secuencia.



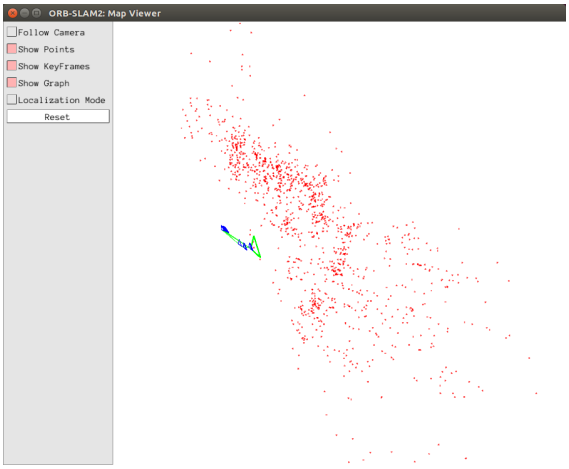
(a) Último *Frame* de la cámara izquierda.



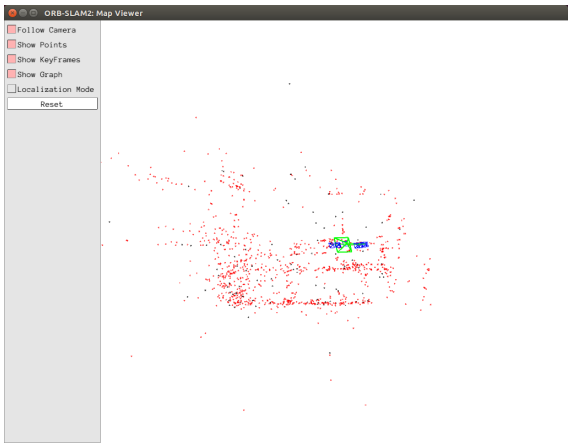
(b) Último *Frame* de la cámara derecha.



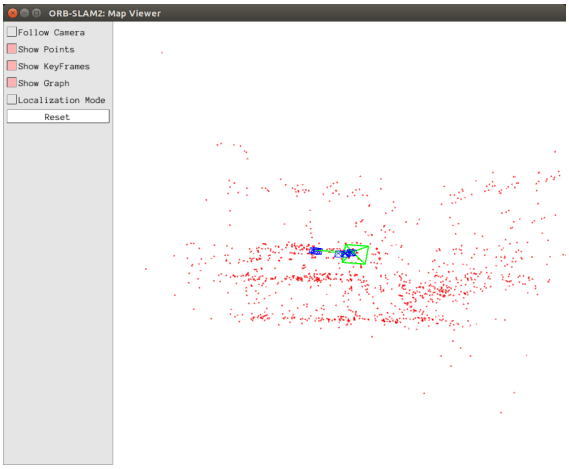
(c) Vista en planta del mapa generado por la cámara izquierda.



(d) Vista en planta del mapa generado por la cámara derecha.



(e) Vista en alzado del mapa generado por la cámara izquierda.



(f) Vista en alzado del mapa generado por la cámara derecha.

Figura 4.11: Resultados de la tercera secuencia usando las imágenes de ambas cámaras.

4.3. ORB-SLAM2 estéreo con cámaras gran angular

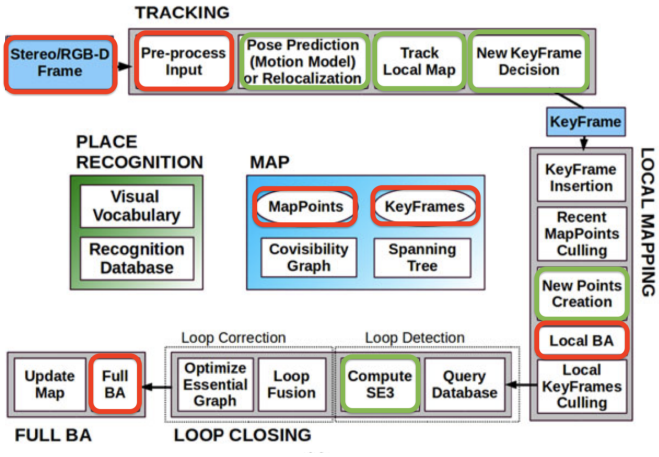
Al igual que en su versión monocular, se ha estudiado la introducción de cámaras estéreo gran angular a ORB-SLAM2. De la misma forma, existen dos aproximaciones a la hora de implementar ORB-SLAM2 estéreo con cámaras gran angular:

- Desdistorsionar y rectificar (hacer ambos planos de imagen coplanares) las imágenes estéreo tomadas por las cámaras, añadiendo una nueva etapa en el preprocesamiento de imágenes.
- Usar las imagenes originales tomadas por ambas cámaras, modificando el emparejamiento de puntos ORB entre imagenes tomadas por el par estéreo y tomando como referencia la cámara izquierda, considerando el sistema estéreo como dos cámaras monoculares independientes unidas por un cuerpo rígido.

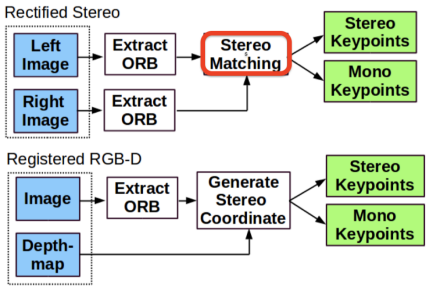
De nuevo, ambas aproximaciones tienen sus ventajas e inconvenientes. Por un lado, la primera aproximación tiene como ventaja el no modificar ORB-SLAM2 a costa de perder el amplio FOV de ambas cámaras gran angular como consecuencia del proceso de rectificado.

La segunda aproximación implica modificar ORB-SLAM2 pero permite aprovechar el FOV de ambas cámaras. En la presente investigación, se ha optado por implementar esta aproximación. Por tanto, partiendo de la versión de ORB-SLAM2 monocular con cámaras gran angular, se han realizado las siguientes modificaciones (Fig. 4.12):

- Estructuras de datos:
  - *KeyFrames*, de manera que ahora se guarden los puntos ORB extraídos en ambas imágenes.
  - *ORBMatcher*, clase utilizada para emparejar puntos ORB entre ambas imágenes. Se ha modificado de manera que ahora se emparejen puntos en ambas imágenes.
- Hilo de **Tracking**:
  - Emparejamiento estéreo presentado en 4.3.1.
- Hilo de **Local Mapping**:
  - Bundle Adjustment (Jacobianos analíticos presentados en A), presentado en 4.3.2.



(a) Hilos y módulos del sistema.



(b) Preprocesado de las imágenes de entrada.

Figura 4.12: En rojo, las modificaciones realizadas sobre la versión de ORB-SLAM2 monocular con cámaras gran angular propuesta en la sección anterior. En verde, los cambios introducidos en dicha versión que no han sido modificados en la presente versión para cámaras estéreo.

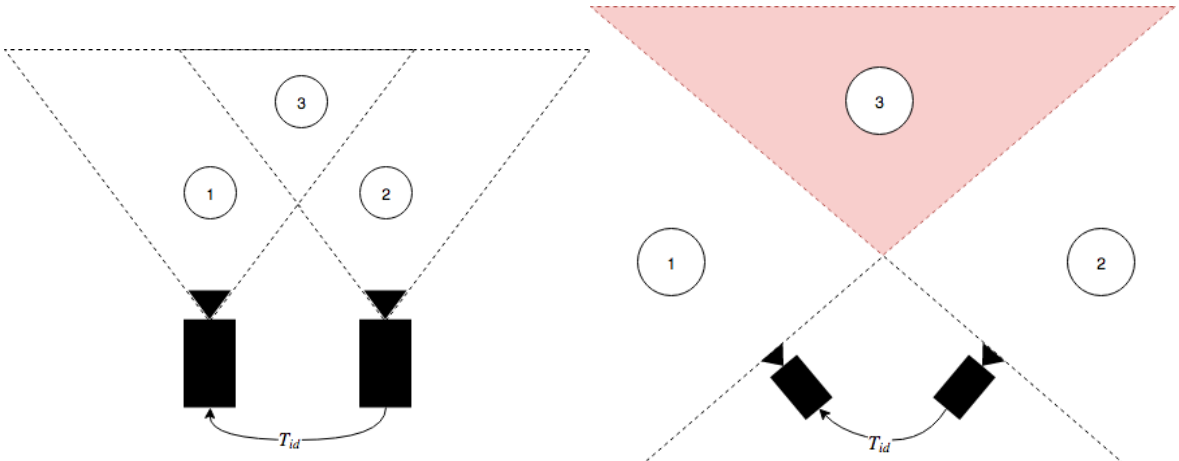
4.3.1. Emparejamientos estéreo

Dadas dos imágenes tomadas en un mismo instante de tiempo por cada una de las cámaras que conforma el par estéreo, es posible emparejar puntos de interés observados en ambas imágenes y triangular su posición 3D, estimando la profundidad de los mismos y, por tanto, la escala real del mapa a construir.

Usando imágenes rectificadas (es decir, ambos planos de imagen son coplanares) y geometría epipolar, esto es especialmente sencillo ya que basta con buscar el correspondiente punto en la línea epipolar en la otra imagen, la cual se proyecta como una recta sobre una misma fila. Sin embargo, en el caso del par estéreo de las gafas de AR de Meta, los planos de imagen son divergentes, ello sumado a la distorsión inducida por los objetivos ojos de pez, la línea epipolar se proyecta como una curva, complicando el proceso de emparejamiento estéreo.

De esta manera, la obtención de correspondencias estéreo se ha obtenido realizando emparejamientos por los descriptores de los puntos ORB extraídos en cada una de las dos imágenes de aquellos puntos puntos que se encuentran sobre el área de solape entre ambas imágenes (el área 3 en la imagen 4.13), aplicando el *ratio de Lowe* [8] para un emparejamiento robusto. Una vez obtenidos dichos emparejamientos, se ha procedido a triangular los correspondientes puntos del mapa usando el algoritmo 1 presentado en la sección anterior.

Gracias a esto, el sistema es capaz de inicializar a partir del primer par de imágenes que el sistema recibe, fijando la escala del mismo, siempre y cuando se encuentren suficientes correspondencias.



(a) Sistema estéreo convencional.

(b) Sistema estéreo divergente ojo de pez.

Figura 4.13: Comparación entre la disposición de las cámaras en un sistema estéreo convencional y el considerado en la presenta investigación.

4.3.2. Bundle Adjustment

En la implementación presentada, se utiliza la pose de la cámara izquierda como referencia. Por tanto, en el *Bundle Adjustment* se optimiza únicamente dicha pose pero usando las observaciones realizadas por ambas cámaras.

De esta manera, para observaciones hechas por la cámara izquierda (la de referencia), se pueden usar los mismos Jacobianos usados para el caso monocular. Así pues, sea $\mathbf{X_w} = \begin{pmatrix} x_i & y_i & z_i \end{pmatrix}^T$ un punto 3D en el sistema de referencia del mundo, $T_{iw} \in SE(3)$ la pose de la camara izquierda y $\hat{\mathbf{x}}_\mathbf{i} = \begin{pmatrix} u_i & v_i \end{pmatrix}^T$ un punto en la imagen izquierda, la función de proyección para las observaciones hechas por la cámara izquierda serÍa:

$$\hat{\mathbf{x}}_\mathbf{i} = \begin{bmatrix} u_i \\ v_i \end{bmatrix} = \Pi_d(T_{iw} \cdot \mathbf{X_w}) \tag{4.16}$$

En el caso de observaciones monoculares realizadas por la cámara derecha donde el punto en la imagen es $\hat{\mathbf{x}}_\mathbf{d} = \begin{pmatrix} u_d & v_d \end{pmatrix}^T$, es necesario llevar el punto $\mathbf{X_w}$ al sistema de referencia de la cámara izquierda mediante la matriz de transformación homogénea entre cámaras $T_{di} \in SE(3)$ quedando la función de reproyección para dicha cámara de la siguiente manera:

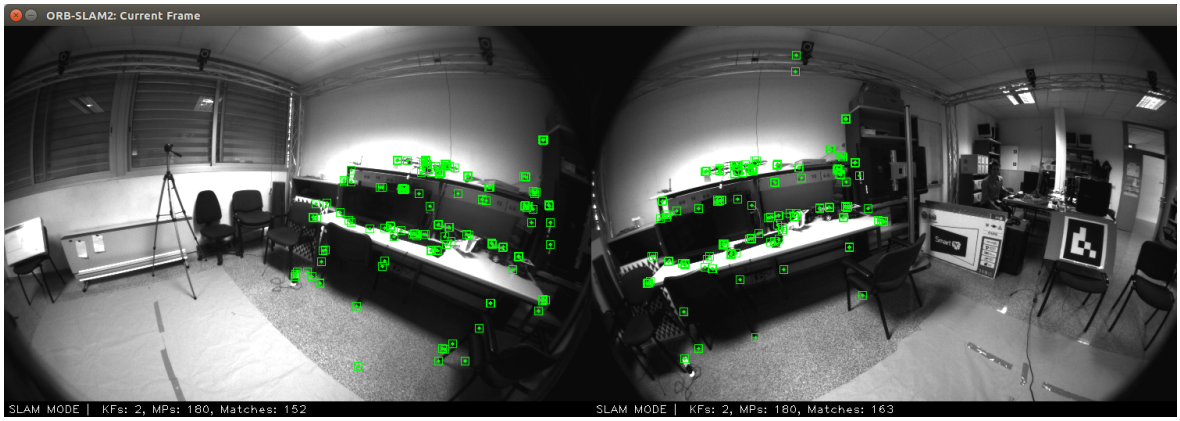
$$\hat{\mathbf{x}}_\mathbf{d} = \begin{bmatrix} u_d \\ v_d \end{bmatrix} = \Pi_d(T_{di} \cdot T_{iw} \cdot \mathbf{X_w}) \tag{4.17}$$

El motivo de optimizar solo una pose de la cámara es para evitar la ambigüedad en la escala: la posición relativa entre las cámaras se asume constante y da medidas reales sobre el mapa, permitiendo estimar la escala del mismo de manera correcta. Si se optimizasen ambas poses, la transformación relativa de las cámaras podría variar y con ella, la escala del propio mapa, perdiendo una de las principales propiedades de realizar SLAM con cámaras estéreo.

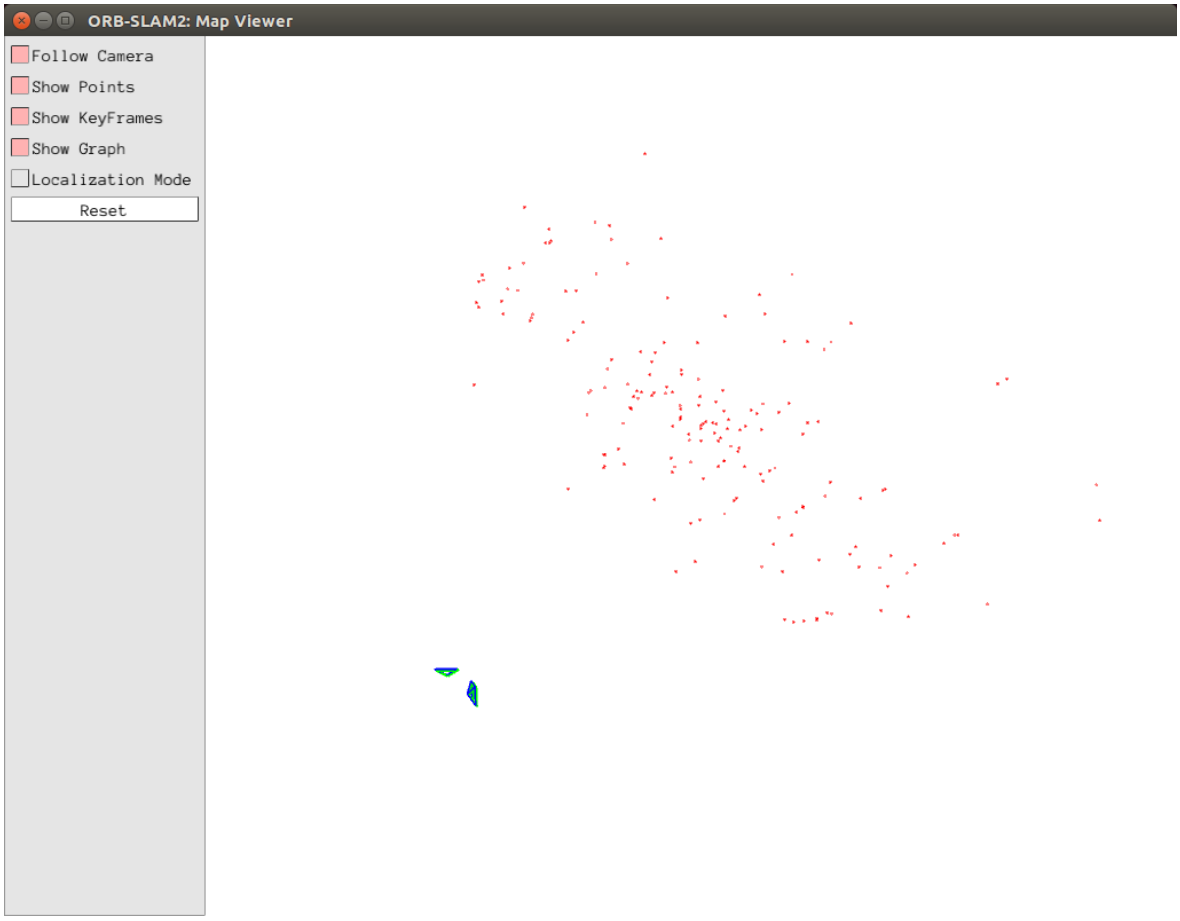
4.3.3. Resultados

Con la escena 3 presentada en 4.2.6, se ha probado ORB-SLAM2 estéreo con cámaras gran angular (imágenes 4.14 y 4.15). El objetivo es comprobar la inicalización del mapa y la generación del mapa a la vez de comparar las ventajas de aplicar ambas cámaras como un sistema estéreo en vez de utilizar una sola cámara como un sistema monocular.

Comparando con los resultados obtenidos al procesar la misma escena considerando cámaras monoculares (4.11), se comprueba como usando ambas cámaras como un par estéreo, se obtiene un campo de visión superior además de conocer la escala del mapa. De igual manera, la trayectoria de ambas cámaras y el mapa generado es el resultado de fusionar los resultados obtenidos con ambas cámaras monoculares independientes.

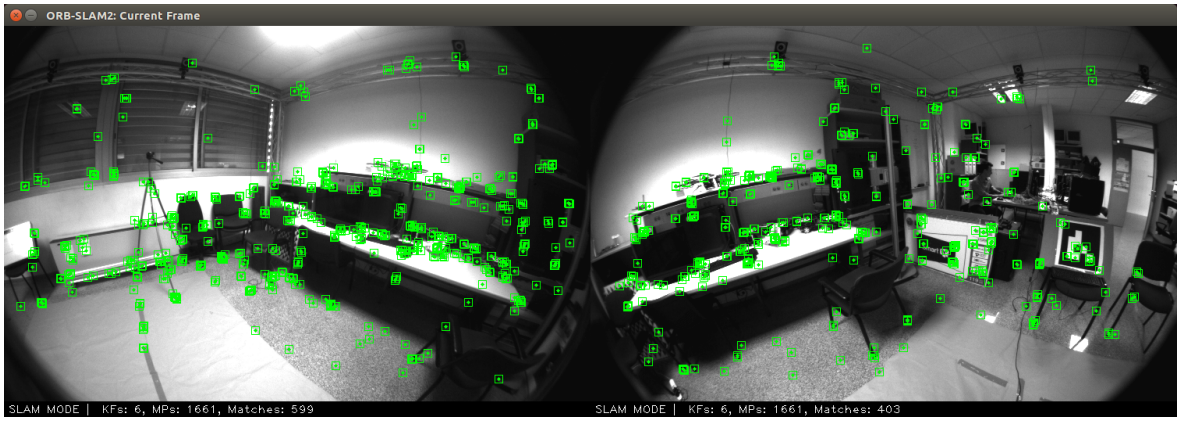


(a) *Frame* de inicialización.

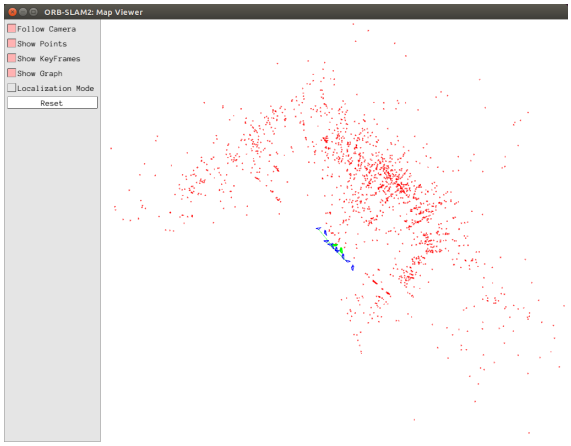


(b) Mapa inicializado.

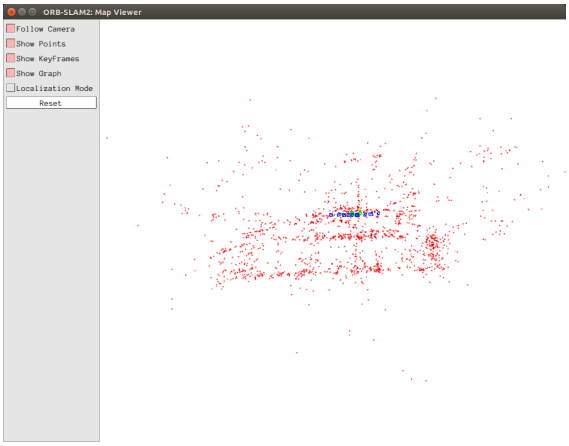
Figura 4.14: Inicialización del sistema estéreo.



(a) Último *Frame* de la secuencia.



(b) Vista en planta del mapa generado.



(c) Vista en alzado del mapa generado.

Figura 4.15: Resultados de la tercera secuencia (4.2.6) usando ambas cámaras como un par estéreo.

Capítulo 5

Conclusiones

5.1. Conclusiones generales

En vista de los resultados obtenidos, se concluye que la parametrización de puntos en Profundidad Inversa en ORB-SLAM2 no supone una mejora significativa en la precisión del sistema. En dichos resultados se observa cómo ambas representaciones consiguen precisiones similares. Sin embargo, la parametrización en profundidad inversa es computacionalmente más costosa. Esto es debido a que en el *Bundle Adjustment* la optimización es realizada respecto de tres elementos: punto del mapa, observación y ancla.

Por otro lado, la estimación inicial para puntos en profundidad inversa aparece más cerca de la solución de la optimización. Esto es especialmente importante en sistemas de SLAM basados en el filtro de Kalman aunque no es una propiedad tan relevante en sistemas basados en *Bundle Adjustment*.

En cuanto al uso de cámaras gran angular, se ha extendido correctamente ORB-SLAM2 de manera que es el primer sistema de SLAM capaz de trabajar con este tipo de cámaras, tanto en el caso monocular como en el estéreo. Este sistema ha mantenido las propiedades características que presentaba ORB-SLAM2 en su versión original, como la relocalización o el cerrado de bucles, dando como resultado un sistema robusto y estable a la vez que aprovecha el amplio campo de visión de las cámaras con objetivo gran angular y objetivo de pez, pudiendo mapear regiones más amplias de los alrededores de la cámara. Debido a ello, esta versión de la librería va a ser registrada en el registro de la propiedad intelectual y a comercializar bajo el nombre de *ORB-SLAM2 for Fisheye cameras*.

5.2. Trabajo futuro

A raíz de los resultados arrojados en el presente trabajo, se han evidenciado posibles lineas de investigación y de mejora sobre la implementación con cámaras gran angular de ORB-SLAM2 ya realizada. La primera de todas es el estudio de un nuevo método para inicializar el mapa cuando se hace SLAM monocular sin tener que desdistorisonar la imagen. De esta manera, surge la idea de un nuevo método de inicialización independiente del tipo de cámara que no requiera de un procesamiento extra sobre las imágenes de entrada. De igual manera, se ha evidenciado la idea de un método de inicialización para sistemas estéreo basado tanto en observaciones estéreo como en observaciones monoculares cuando no se consiguen extraer suficientes observaciones estéreo.

Por otro lado, en la presente investigación se ha presentado la implementación de ORB-SLAM2 estéreo con cámaras gran angular. En esta versión, el par estéreo es considerado como dos cámaras monoculares independientes unidas por un cuerpo rígido. Esta idea, junto al hecho de no realizar un rectificado sobre las imágenes abre la puerta a extender el sistema para que soporte multicámaras, donde el sistema se compone de múltiples cámaras unidas por un cuerpo rígido que no necesariamente ven las mismas partes de la escena.

5.3. Resultados del aprendizaje

Este proyecto ha sido mi primera toma de contacto con el mundo de la investigación. Como tal, me he enfrentado a múltiples problemas tales como un proyecto de grandes dimensiones en cuya temática principal no tenía experiencia previa. Ello ha provocado una etapa de aprendizaje que ha implicado aprender a buscar, leer e interpretar adecuadamente literatura científica.

Además, este proyecto también ha supuesto un primer contacto con el mundo empresarial, más concretamente con la empresa *Meta*, situada en Sillicon Valley. Con dicha empresa ha habido una colaboración de la cual surgió la segunda parte de este proyecto (ORB-SLAM2 con camaras gran angular) y con la que continuaré gracias a un *intership* de 6 meses en su sede de San Mateo como ingeniero en visión por computador.

Finalmente, en este proyecto he descubierto una pasión, no solo por la temática del problema de SLAM y sus aplicaciones, sino también por el universo de la investigación: resolver un problema no mediante una receta ya establecida si no mediante la proposición y prueba de hipótesis. De esta manera, considero este proyecto como una experiencia maravillosa que me gustaría continuar con una futura tesis doctoral.

5.4. Gestión del proyecto

Tabla 5.1: Desglose de las horas dedicadas al proyecto en las diferentes etapas del mismo.

Etapa	Horas dedicadas
Aprendizaje de ORB-SLAM2	120
Puntos en profundidad inversa	150
Memoria puntos en profundidad inversa	15
Pruebas autocalibración	60
ORB-SLAM2 monocular con cámeras gran angular	300
ORB-SLAM2 estéreo con cámaras gran angular	120
Memoria ORB-SLAM2 con cámaras gran angular	15
Total	780

El desarrollo del presente proyecto se ha compatibilizado con el curso académico, de manera que se han dedicado 15 horas semanales al mismo desde el 1 de Julio de 2017 hasta el 29 de Junio de 2018. De esta manera, se han calculado un total de 780 horas dedicadas al proyecto (Tabla 5.1).

.

El reparto de tareas (Fig. 5.1) se ha planificado en base a una fase inicial de aprendizaje del sistema ORB-SLAM2, a partir de la cual, se han ido implementado los diferentes objetivos del proyecto. Cabe destacar la etapa de pruebas de autocalibración, que era uno de los objetivos iniciales del proyecto, siendo sustituido por el uso de cámaras gran angular fruto de la colaboración con la empresa *Meta*.

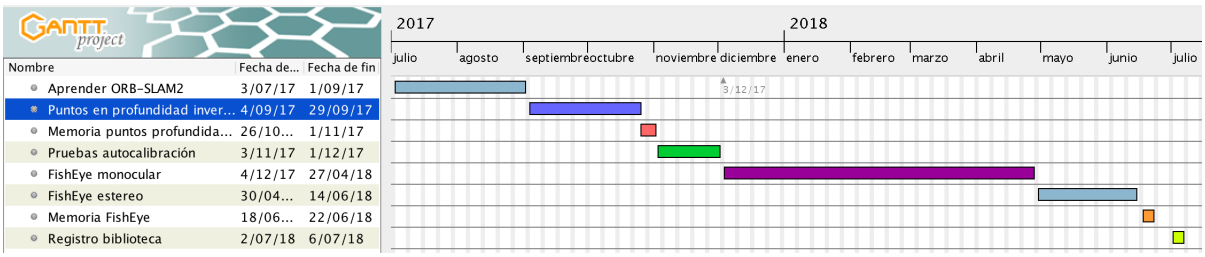


Figura 5.1: Diagrama de Gantt del proyecto.

Capítulo 6

Bibliografía

[1] Civera, J., Davison, A. J., and Montiel, J. M. Inverse depth parametrization for monocular slam. *IEEE transactions on robotics 24*, 5 (2008), 932–945.

[2] Eade, E. Lie groups for 2d and 3d transformations.

[3] Geiger, A., Lenz, P., Stiller, C., and Urtasun, R. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research 32*, 11 (2013), 1231–1237.

[4] Kannala, J., and Brandt, S. S. A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*, 8 (2006), 1335–1340.

[5] Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., and Burgard, W. g 2 o: A general framework for graph optimization. 3607–3613.

[6] Lepetit, V., Moreno-Noguer, F., and Fua, P. Epnp: An accurate o (n) solution to the pnp problem. *International journal of computer vision 81*, 2 (2009), 155.

[7] Lourakis, M., and Argyros, A. A. Is levenberg-marquardt the most efficient optimization algorithm for implementing bundle adjustment? 1526–1531.

[8] Lowe, D. G. Distinctive image features from scale-invariant keypoints. *International journal of computer vision 60*, 2 (2004), 91–110.

[9] Mur-Artal, R., Montiel, J. M. M., and Tardós, J. D. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics 31*, 5 (2015), 1147–1163.

[10] Mur-Artal, R., and Tardós, J. D. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics 33*, 5 (2017), 1255–1262.

[11] Strasdat, H. *Local accuracy and global consistency for efficient visual SLAM.* PhD thesis, Imperial College, London, 2012.

[12] Sturm, J., Engelhard, N., Endres, F., Burgard, W., and Cremers, D. A benchmark for the evaluation of rgb-d slam systems. 573–580.

[13] Triggs, B., McLauchlan, P. F., Hartley, R. I., and Fitzgibbon, A. W. Bundle adjustment—a modern synthesis. 298–372.

[14] Urban, S., Leitloff, J., and Hinz, S. MLPnP-A Real-Time Maximum Likelihood Solution to the Perspective-n-Point Problem. *arXiv preprint arXiv:1607.08112* (2016).

Lista de Figuras

Lista de Tablas

Anexos

Anexos A

Jacobianos

A.1. Modelo de proyección pinhole

Sea $\Pi(x)$ el modelo de proyección de cámaras pinhole definido en 2.2.1 como:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \Pi(x) = \begin{bmatrix} \frac{f}{d_x} \cdot \frac{x_1}{x_3} + u_c \\ \frac{f}{d_y} \cdot \frac{x_2}{x_3} + v_c \end{bmatrix} = \begin{bmatrix} f_x \cdot \frac{x_1}{x_3} + u_c \\ f_u \cdot \frac{x_2}{x_3} + v_c \end{bmatrix} \tag{A.1}$$

su Jacobiano analitico [11] respecto del punto x puede ser representado mediante:

$$\frac{\partial \Pi(x)}{\partial x} = \frac{1}{x_3} \begin{bmatrix} f_x & 0 & -\frac{x_1 \cdot f_x}{x_3} \\ 0 & f_y & -\frac{x_2 \cdot f_y}{x_3} \end{bmatrix} \tag{A.2}$$

A.2. Modelo de proyección gran angular

Sea $\Pi_d(x)$ el modelo de proyección de cámaras gran angular definido en 4.1.2 como:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \Pi_d(x) = \mathcal{A} \circ \mathcal{F}_d \circ \mathcal{R} \tag{A.3}$$

donde:

$$\Phi = \begin{pmatrix} \theta \\ \varphi \end{pmatrix} = \mathcal{R}(x) = \begin{pmatrix} \arctan\left(\sqrt{\frac{x_1^2 + x_2^2}{x_3^2}}\right) \\ \arctan\left(\frac{x_2}{x_1}\right) \end{pmatrix} \tag{A.4}$$

$$\theta_d = \theta(1 + k_1\theta^2 + k_2\theta^4 + k_3\theta^6 + k_4\theta^8) \tag{A.5}$$

$$\boldsymbol{X}_d = \begin{pmatrix} x_d \\ y_d \end{pmatrix} = \mathcal{F}_d(\Phi) = \theta_d \begin{pmatrix} \cos(\varphi) \\ \sin(\varphi) \end{pmatrix} \tag{A.6}$$

$$\mathbf{X_i} = \begin{pmatrix} u_d \\ v_d \end{pmatrix} = \mathcal{A}(\boldsymbol{X}_d) = \begin{bmatrix} f_u & 0 \\ 0 & f_v \end{bmatrix} \begin{pmatrix} u_d \\ v_d \end{pmatrix} + \begin{pmatrix} c_u \\ c_v \end{pmatrix} \tag{A.7}$$

sus derivadas parciales respecto del punto x son:

$$\frac{\partial \Pi_d(x)}{\partial x_1} = \begin{bmatrix} f_x \left(\dfrac{f' x_3}{r^2(r^2+x_3^2)} x_1^2 + \dfrac{\theta_d}{r^3} x_2^2 \right) \\ f_y \left(\dfrac{f' x_3}{r^2(r^2+x_3^2)} x_1 x_2 - \dfrac{\theta_d}{r^3} x_1 x_2 \right) \end{bmatrix} \tag{A.8}$$

$$\frac{\partial \Pi_d(x)}{\partial x_2} = \begin{bmatrix} f_x \left(\dfrac{f' x_3}{r^2(r^2+x_3^2)} x_1 x_2 - \dfrac{\theta_d}{r^3} x_1 x_2 \right) \\ f_y \left(\dfrac{f' x_3}{r^2(r^2+x_3^2)} x_2^2 + \dfrac{\theta_d}{r^3} x_1^2 \right) \end{bmatrix} \tag{A.9}$$

$$\frac{\partial \Pi_d(x)}{\partial x_3} = \begin{bmatrix} -f_x \dfrac{f'}{r^2+x_3^2} x_1 \\ -f_y \dfrac{f'}{r^2+x_3^2} x_2 \end{bmatrix} \tag{A.10}$$

donde $f' = 1 + 3k_1\theta^2 + 5k_2\theta^4 + 7k_3\theta^6 + 9k_4\theta^9$ es la derivada de θ_d.

A.3. Bundle Adjustment con puntos con coordenadas Cartesianas

Bajo la suposición de una cámara monocular, donde $y \in \mathbb{R}^3$ es un punto del mapa, z su proyección en la imagen y $T \in SE(3)$ es la matriz de transformación homogénea de la cámara (4×4), considerese el producto $T \cdot y = Ry + t$ siendo R y t la matriz de rotacion y el vector de traslacion de T respectivamente. El Jacobiano de los errores de reproyección respecto de un punto [11] es:

$$\begin{aligned} \frac{\partial (z - \Pi(T \cdot y))}{\partial y} &= -\left. \frac{\partial \Pi(x)}{\partial x} \right|_{x=Ry+t} \cdot \frac{\partial (Ry+t)}{\partial y} \\ &= -\frac{1}{x_3} \begin{bmatrix} f_x & 0 & -\frac{x_1 \cdot f_x}{x_3} \\ 0 & f_y & -\frac{x_2 \cdot f_y}{x_3} \end{bmatrix} \cdot R \end{aligned} \tag{A.11}$$

Sin embargo, el Jacobiano analitico respecto de la pose de la camara no es inmediato. Esto es debido a que el espacio $SE(3)$ de rotaciones y traslaciones en el que se representan las poses de las camaras es un grupo de Lie [2]. Por tanto, para calcular el

Jacobiano analítico en este grupo se realiza calculando pequeños incrementos $\epsilon \in se(3)$, donde $se(3)$ es el espacio tangente de $SE(3)$ (Fig. A.1).

El incremento ϵ, de 6 grados de libertad (3 para rotación y 3 para traslación) se puede transformar al espacio $SE(3)$ mediante el *mapeo exponencial*, denotado como e^{ϵ}. De esta manera, sea T^k la pose de la cámara actual a optimizar, la actualización de la misma mediante el incremento ϵ puede ser expresado mediante:

$$T^{k+1} = e^{\epsilon} \cdot T^k \tag{A.12}$$

Por ello, el Jacobiano respecto de la cámara [11] se realiza usando el *mapeo exponencial* sobre T tal que:

$$\begin{aligned} &\frac{\partial(z - \Pi(e^{\epsilon} \cdot T \cdot y))}{\partial \epsilon}\bigg|_{\epsilon=0} \\ &= -\frac{\partial \Pi(T \cdot y)(x)}{\partial x}\bigg|_{x=Ry+t} \cdot \frac{\partial \Pi(e^{\epsilon} \cdot x)}{\partial \epsilon}\bigg|_{\epsilon=0} \\ &= -\frac{1}{x_3}\begin{bmatrix} f_x & 0 & -\frac{x_1 \cdot f_x}{x_3} \\ 0 & f_y & -\frac{x_2 \cdot f_y}{x_3} \end{bmatrix} \cdot \begin{bmatrix} I_{3\times 3} & -[x]_{\times} \end{bmatrix} \end{aligned} \tag{A.13}$$

donde $[x]_{\times}$ es la matriz *skew* simétrica de un vector $x \in \mathbb{R}^3$ definida como:

$$[x]_{\times} = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix} \tag{A.14}$$

A.4. Bundle Adjustment con puntos en Profundidad Inversa

El error de reproyección para puntos en Profundidad Inversa se define como:

$$z - \Pi(T_{ow} \cdot T_{aw}^{-1} \cdot \Lambda(\psi)) \tag{A.15}$$

donde $T_{ow} \in SE(3)$ es la pose de la cámara desde la que se realiza la observación, $T_{aw} \in SE(3)$ es la posición de la cámara que actúa como ancla y $\psi \in \mathbb{R}^3$ es el punto en Profundidad Inversa. Ademas, sea $T_{oa} = T_{ow} \cdot T_{aw}^{-1}$ e $y = \Lambda(\psi)$. El Jacobiano respecto

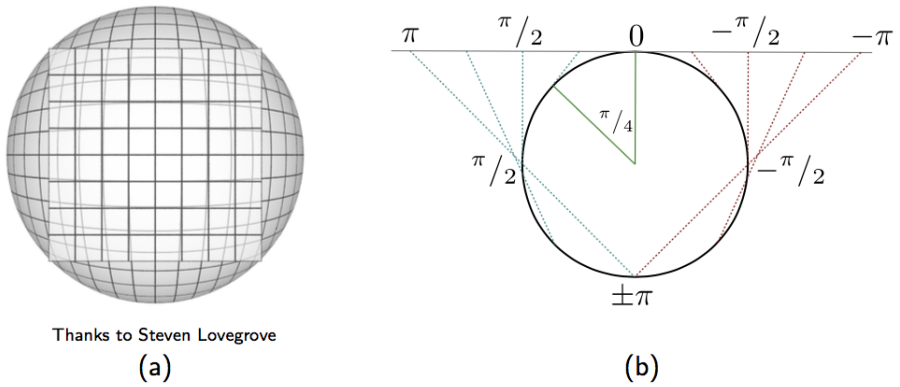


Figura A.1: (a) Una (2-)esfera como ejemplo de una variedad. Localmente, puede ser aproximada por un plano tangente. (b) El grupo circulo (o 1-esfera) representa rotaciones en el plano. Es un grupo de Lie conmutativo (imagen extraida de [11]).

del punto es:

$$\begin{aligned}
&\frac{\partial(z - \Pi(T_{ow} \cdot T_{aw}^{-1} \cdot \Lambda(\psi)))}{\partial\psi} \\
&= -\frac{\partial\Pi(x)}{\partial x}\bigg|_{x=R_{oa}y+t_{oa}} \cdot \frac{\partial(R_{oa} \cdot \Lambda(\psi) + t_{oa})}{\partial\psi} \\
&= -\frac{1}{x_3}\begin{bmatrix} f_x & 0 & -\frac{x_1 \cdot f_x}{x_3} \\ 0 & f_y & -\frac{x_2 \cdot f_y}{x_3} \end{bmatrix} \cdot \begin{bmatrix} r_{oa1} & r_{oa2} & -R_{oa}y \end{bmatrix}
\end{aligned} \tag{A.16}$$

donde r_{oa1} y r_{oa2} son las dos primeras columnas de R_{oa}. El Jacobiano respecto la posición de la cámara desde la que se realiza la observación es:

$$\begin{aligned}
&\frac{\partial(z - \Pi(e^{\epsilon_o} \cdot T_{ow} \cdot T_{aw}^{-1}))}{\partial\epsilon_o}\bigg|_{\epsilon_o=0} \\
&= -\frac{\partial\Pi(x)}{\partial x}\bigg|_{x=R_{oa}y+t_{oa}} \cdot \frac{\Pi(e^{\epsilon_o} \cdot x)}{\partial\epsilon_o}\bigg|_{\epsilon_o=0} \\
&= -\frac{1}{x_3}\begin{bmatrix} f_x & 0 & -\frac{x_1 \cdot f_x}{x_3} \\ 0 & f_y & -\frac{x_2 \cdot f_y}{x_3} \end{bmatrix} \cdot \begin{bmatrix} I_{3\times3} & -[x]_\times \end{bmatrix}
\end{aligned} \tag{A.17}$$

Finalmente, el Jacobiano respecto de la cámara que actúa como ancla es:

$$\begin{aligned}
&\left.\frac{\partial(z-\Pi(T_{ow}\cdot(e^{\epsilon_a}\cdot T_{aw})^{-1})\cdot\Lambda(\psi))}{\partial\epsilon_a}\right|_{\epsilon_a=0}\\
&=\left.\frac{\partial(z-\Pi(T_{ow}\cdot T_{aw}^{-1}\cdot e^{-\epsilon_a})\cdot\Lambda(\psi))}{\partial\epsilon_a}\right|_{\epsilon_a=0}\\
&=\left.\frac{\partial(z-\Pi(T_{oa}\cdot e^{-\epsilon_a})\cdot\Lambda(\psi))}{\partial\epsilon_a}\right|_{\epsilon_a=0}\\
&=\left.\frac{\partial\Pi(x)}{\partial x}\right|_{x=R_{oa}y+t_{oa}}\cdot\frac{\partial\Pi(T_{oa}y)}{\partial y}\cdot\left.\frac{\partial\Pi(e^{\epsilon_a})\cdot y)}{\partial\epsilon_a}\right|_{\epsilon_a=0}\\
&=\frac{1}{x_3}\begin{bmatrix}f_x & 0 & -\frac{x_1\cdot f_x}{x_3}\\ 0 & f_y & -\frac{x_2\cdot f_y}{x_3}\end{bmatrix}\cdot R_{oa}\cdot\begin{bmatrix}I_{3\times3} & -[y]_\times\end{bmatrix}
\end{aligned}\tag{A.18}$$

A.5. Bundle Adjustment gran angular

De manera similar a los Jacobianos definidos para el *Bundle Adjustment* para puntos en Profundidad Inversa, nosotros hemos desarrollado los Jacobianos para el *Bundle Adjustment* usando cámaras gran angular respecto de un punto del mapa, que es:

$$\begin{aligned}
\frac{\partial z-\Pi_d(T\cdot y)}{\partial y}&=-\left.\frac{\partial\Pi_d(x)}{\partial x}\right|_{x=R\cdot y+t}\cdot\frac{R\cdot x+t}{\partial x}\\
&=-\begin{bmatrix}\frac{\partial\Pi_d(x)}{\partial x_1} & \frac{\partial\Pi_d(x)}{\partial x_2} & \frac{\partial\Pi_d(x)}{\partial x_3}\end{bmatrix}\cdot R
\end{aligned}\tag{A.19}$$

Finalmente, el Jacobiano del *Bundle Adjustment* para cámaras gran angular respecto de la pose de la cámara es:

$$\begin{aligned}
&\left.\frac{\partial(z-\Pi_d(e^{\epsilon})\cdot T\cdot y))}{\partial\epsilon}\right|_{\epsilon=0}\\
&=-\left.\frac{\partial\Pi_d(x)}{\partial x}\right|_{x=R\cdot y+t}\cdot\left.\frac{\Pi_d(exp(\hat{\epsilon})\cdot x)}{\partial\epsilon}\right|_{\epsilon=0}\\
&=-\begin{bmatrix}\frac{\partial\Pi_d(x)}{\partial x_1} & \frac{\partial\Pi_d(x)}{\partial x_2} & \frac{\partial\Pi_d(x)}{\partial x_3}\end{bmatrix}\cdot\begin{bmatrix}I_{3\times3} & -[x]_\times\end{bmatrix}
\end{aligned}\tag{A.20}$$