Trabajo Fin de Grado

Grado en Ingeniería de las Tecnologías Industriales

Sistema de odometría visual para localización monocular

Autor: Ángel Manuel Lema Fulgencio

Tutor: Begoña Chiquinquira Arrue Ulles

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Sistema de odometría visual para localización monocular

Autor:
Ángel Manuel Lema Fulgencio

Tutora:
Begoña Chiquinquira Arrue Ulles
Profesora titular

Dpto. de Ingeniería de Sistema y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Trabajo Fin de Grado: Sistema de odometría visual para localización monocular

Autor: Ángel Manuel Lema Fulgencio

Tutora: Begoña Chiquinquira Arrue Ulles

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

A mi familia

A mis Amigos

A mis profesores

Agradecimientos

Quisiera dar las gracias a mi tutora Begoña y a Pablo por toda su implicación en el proyecto, ya que sin sus ideas y sugerencias no podría haber llevado a cabo este trabajo.

También dar las gracias a mi familia y amigos por todo el apoyo que me han brindado y por la confianza que depositaron en mí cuando decidí estudiar una ingeniería.

Ángel Manuel Lema Fulgencio

Sevilla, 2019

Resumen

En este documento se pretende resolver el problema de posicionamiento del robot móvil a través de la técnica de odometría visual monocular, la cual, como su propio nombre indica, utiliza una única cámara como sensor. Las cámaras son sensores de coste relativamente bajo que además tienen la ventaja de aportar gran cantidad de información sobre el entorno que rodea al robot móvil.

En este proyecto se diseña un algoritmo de odometría visual monocular, pensado para tiempo real y con frecuencia de captura de imágenes relativamente alta (30 fps).

En el estado del arte se describen las diferentes técnicas existentes para abordar la odometría visual monocular. Discutiremos el modelo matemático utilizado para proyectar puntos tridimensionales del mundo real en el plano bidimensional de la imagen, los diversos medios para extraer información relevante de una imagen y la forma de emplear dicha información para obtener la trayectoria recorrida por el robot móvil.

En el tercer apartado se hará hincapié en el método propuesto para resolver el problema de odometría visual monocular. Tras esto, se propone un cuarto apartado en el que se analizan los resultados arrojados por el algoritmo propuesto.

Finalmente se proponen una serie de mejoras de cara a ser implantadas en un futuro, ya que este trabajo está basado en un área de estudio que actualmente se encuentra en investigación.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

fps	*Fotogramas por segundo*
SLAM	*Simultaneous Localization And Mapping*
SFM	*Structure From Motion*
DoG	*Diferencia de gaussianas (difference of gaussians)*
NCC	*Normalized Cross-Correlation*
SSD	*Sum of Squared Differences*
SAD	*Sum of Absolute Differences*
etc.	*Etcetera*
2D	*Dos dimensiones*
3D	*Tres dimensiones*
FAST	*Features from Accelerated Segmented Test*
SIFT	*Scale Invariant Feature Transform*
SURF	*Speeded-Up Robust Features*
CenSurE	*Center Surround Extrema*
BRIEF	*Binary Robust Independent Elementary Features*
ORB	*Oriented FAST and Rotated BRIEF*
BRISK	*Binary Robust Invariant Scalable Keypoints*
FREAK	*Fast Retina Keypoint*
FLANN	*Fast Library for Approximate Nearest Neighbors*
GPS	*Global Positioning System*
RANSAC	*RANdom SAmple Consensus*
PnP	*Perspectiva desde n Puntos*
ROS	*Robot Operating System*

1 INTRODUCCIÓN

Todo robot móvil autónomo tiene la necesidad de conocer su localización en tiempo real. Para resolver este problema de posicionamiento se suelen utilizar sensores que proporcionan información sobre la localización actual del robot o bien sobre las modificaciones en el entorno donde se desarrolla la actividad de este.

Definimos odometría visual como el proceso mediante el que se calcula, de forma aproximada, la trayectoria recorrida por un robot móvil a partir de la información obtenida del sistema de visión de este.

La técnica de *Simultaneous Localization And Mapping* (SLAM) consiste en realizar dos tareas de forma simultánea: construir un mapa de un entorno desconocido en el que se encuentra un robot móvil y calcular la trayectoria que recorre el robot dentro de dicho entorno [1]. Por consiguiente, la odometría visual se podría considerar como un módulo que resuelve parte del problema de SLAM.

1.1 Evolución Histórica de la Odometría Visual

En la literatura, el procedimiento mediante el cual se utiliza una secuencia de imágenes para el cálculo de la posición relativa de la cámara respecto de un entorno y la estructura tridimensional del mismo, se denomina estructura a partir del movimiento o *Structure From Motion* (SFM) [2,3]. Teniendo en cuenta la definición de odometría visual proporcionada en párrafos anteriores, podemos afirmar que la odometría visual es un caso particular de SFM.

La primera persona en pensar que la cámara sería un buen sensor para estimar la posición de un robot móvil fue Moravec [4].

En su tesis publicada en el año 1980 [4], Moravec propuso un método de detección de obstáculos durante la navegación que utilizaba como sensor un sistema de captura de imágenes. Dicho sistema estaba constituido por una cámara que podía desplazarse horizontalmente sobre un raíl colocado en el robot móvil. La figura 1.1.1 extraída de la tesis de Moravec [4], muestra la forma del robot móvil empleado.

Cada cierto tiempo el robot móvil se paraba y la cámara realizaba una serie de capturas. La cámara se movía de forma equidistante a lo largo del raíl para realizar dichas capturas. Esto le permitía obtener distintas perspectivas de la misma escena.

Con esta secuencia de imágenes bidimensionales se pretendía obtener la posición tridimensional de cada objeto que apareciese en el campo de visión del robot. Para ello era necesario procesar adecuadamente las imágenes. En primer lugar, se buscaban una serie de puntos característicos. Para ello, Moravec desarrolló el que vino a ser uno de los primeros detectores de características al que denominó “Operador de Interés” y que actualmente se denomina “Detector de Esquinas de Moravec” [5]. Este detector se encargaba de encontrar esquinas en cada una de las imágenes. Una vez detectadas las esquinas, se procedía a buscar coincidencias entre las esquinas de una imagen y las esquinas de las imágenes del resto de la secuencia.

El siguiente paso consistía en buscar posibles coincidencias con las posibles posiciones del robot en el estado anterior. Posteriormente se utilizaba la secuencia completa de imágenes para filtrar puntos que resultaban inconsistentes por su profundidad. Para concluir, se calculaba el movimiento efectuado y la posición relativa de los obstáculos con respecto al robot móvil a través de transformaciones rígidas que alineaban los puntos tridimensionales observados desde dos posiciones consecutivas del robot. Este procedimiento generaba un sistema de ecuaciones que se resolvía mediante la técnica de mínimos cuadrados ponderados. Los pesos asignados a cada ecuación disminuían conforme aumentaba la distancia entre el robot y el punto tridimensional calculado.

Aunque pudiese parecer que esta fue la primera solución utilizando un sistema de visión monocular, es decir constituido por una única cámara, no es del todo cierto, ya que la distancia recorrida por la cámara entre captura y captura era conocida y, por consiguiente, lo que se tenía realmente, era un sistema de visión estéreo en el que cada par de imágenes capturadas por la cámara constituían un par estéreo.

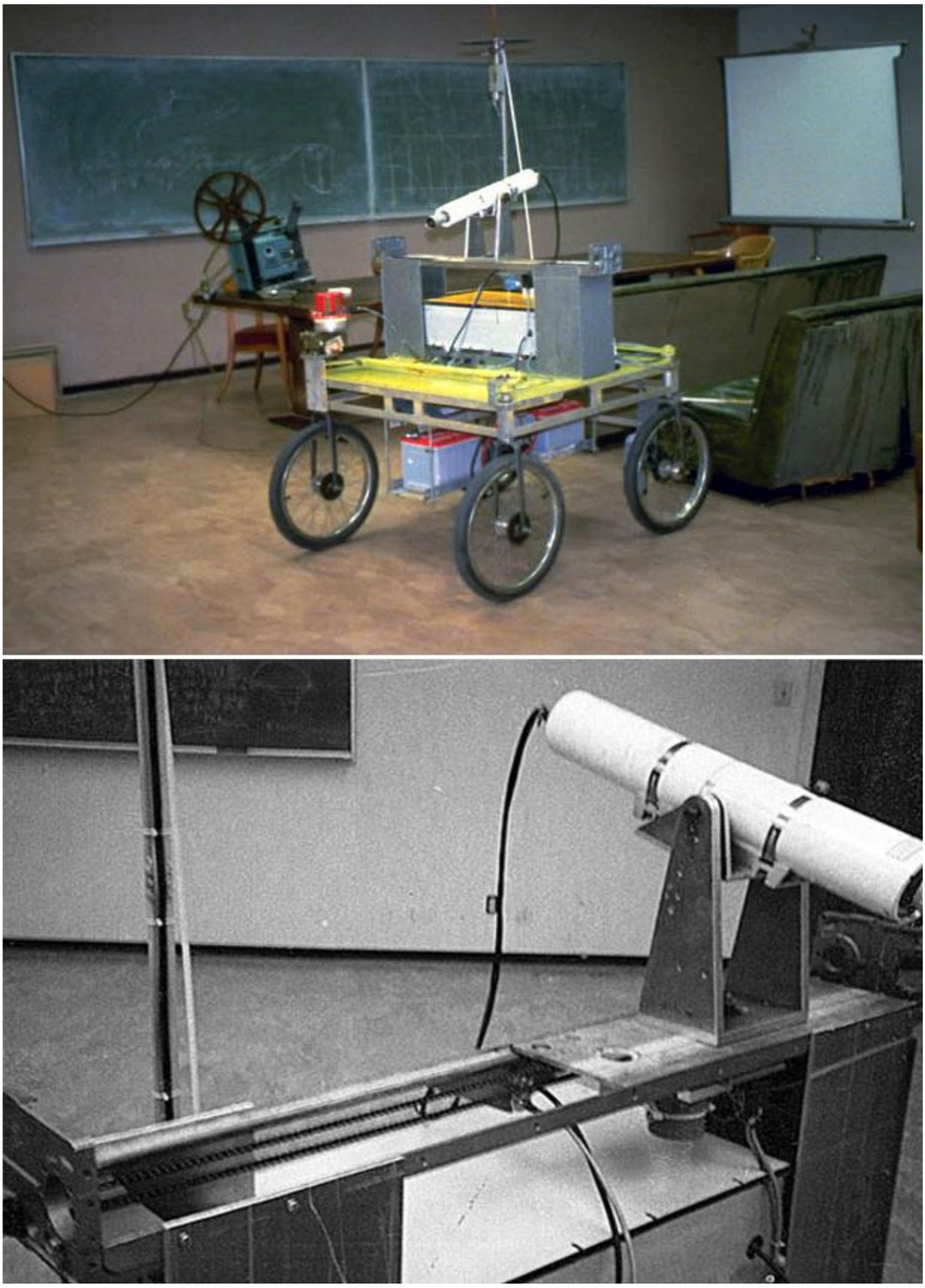


Figura 1.1.1: Robot móvil utilizado por Moravec y su correspondiente sistema de visión

El trabajo de Moravec serviría de referencia para posteriores trabajos relacionados con la NASA. Sus ideas se utilizarían en los programas de exploración espacial de Marte, dotando a los robots no tripulados conocidos como "rovers" de un sistema de visión que les permitiese desenvolverse en entornos carentes de balizamiento y con terrenos irregulares.

Como mejora del trabajo de Moravec, merece la pena destacar el robot utilizado por Lacroix et al. en su proyecto [6]. En dicho proyecto se propone un método de localización basado en un sistema de visión estéreo como el propuesto por Moravec [4], pero con una serie de mejoras con respecto a la estimación de la posición del robot móvil.

Lacroix et al. propusieron crear una especie de almacén de objetos tridimensionales. Por cada objeto detectado en la imagen durante la ejecución del algoritmo de localización, se creaba una nueva entidad en el almacén a la que denominaban "aspecto". En dichos aspectos se guardaba información geométrica del objeto como el volumen o el momento de inercia. Además, también se almacenaban la distancia y la orientación de la cámara respecto al objeto detectado.

Si al procesar las siguientes imágenes se detectaban aspectos que guardasen algún tipo de correlación con los ya existentes en el almacén, se procedía a recalcular la posición del robot móvil en función de la posición del aspecto guardo en el almacén. Con ello se conseguía refinar la estimación de la posición del robot móvil. No obstante, el método estaba limitado a la detección de objetos sencillos tales como piedras que destacasen sobre entornos planos.

A pesar de sus limitaciones, estos dos últimos trabajos [4,6] suponen un punto de partida para el campo de la odometría visual.

Posteriormente Nistér et al. [7] en su artículo "*Visual Odometry"* publicado en el año 2004, plantean la forma de desarrollar un sistema de visión útil para la localización de un robot móvil tanto en configuración estéreo como en configuración monocular.

En este artículo es la primera vez que se utiliza el término de odometría visual como tal. Dicho termino se acuña por la similitud existente con la odometría por ruedas. En la odometría por ruedas, se dispone de unos dispositivos denominados encoders que nos permiten calcular la distancia recorrida en un determinado intervalo de tiempo. Si partimos de una posición conocida podemos calcular fácilmente la posición actual de forma incremental sumando a la posición conocida el incremento de posición obtenido a través de las medidas proporcionadas por los encoders de las ruedas.

En la odometría visual el razonamiento a seguir es similar. Si asumimos que la posición en la que se encuentra el robot inicialmente actúa de sistema de referencia y además disponemos de imágenes capturadas en un instante de tiempo correspondiente a la posición conocida y en un instante de tiempo posterior, se puede calcular el incremento de posición entre ambos instantes de tiempo procesando adecuadamente la información extraída de ambas imágenes. Finalmente, se obtiene la nueva posición del robot móvil sumando a la posición conocida el incremento calculado.

Partiendo de esta idea Nistér et al. [7] diseñaron un programa capaz de obtener la posición relativa del robot móvil utilizando un algoritmo de 5 puntos, la herramienta iterativa *Random sample consensus* (RANSAC) para la obtención de parámetros de un modelo matemático y el método de triangulación. Con este programa consiguieron reconstruir la trayectoria de un robot móvil durante unos 184 m con un error de 4,1 m. Este método resultó ser más eficiente que la odometría calculada a través del movimiento de las ruedas.

En la figura 1.1.2 podemos apreciar el algoritmo seguido por Nistér para el caso monocular. Este algoritmo se ha extraído directamente del artículo "*Visual Odometry for Ground Vehicle Applications"* [8].

1. Track features over a certain number of frames. Estimate the relative poses between three of the frames using the 5-point algorithm (Nister 2003a) and preemptive RANSAC (Nister 2003b) followed by iterative refinement.

2. Triangulate the observed feature tracks into 3D points using the first and last observation on each track and optimal triangulation according to directional error. This can be achieved in closed form (Oliensis and Genc 1999). If this is not the first time through the loop, estimate the scale factor between the present reconstruction and the previous camera trajectory with another preemptive RANSAC procedure. Put the present reconstruction in the coordinate system of the previous one.

3. Track for a certain additional number of frames. Compute the pose of the camera with respect to the known 3D points using the 3-point algorithm (Haralick et al. 2004) and preemptive RANSAC followed by iterative refinement.

4. Re-triangulate the 3D points using the first and last observations on their image track. Repeat from Step 3 a certain number of times.

5. Repeat from Step 1 a certain number of times.

6. Insert a firewall and repeat from Step 1

Figura 1.1.2: Algoritmo de Nistér et al. para el caso monocular.

1.2 Definición del problema

La formulación del problema sería la siguiente: un robot móvil se mueve en un entorno desconocido capturando imágenes con su sistema de visión en instantes de tiempo discretos. El conjunto de imágenes capturadas se denota como $I_{0:n} = \{I_0, I_1, \ldots, I_n\}$ donde el subíndice indica el instante de tiempo en el que las imágenes fueron capturadas.

Dos posiciones consecutivas de la cámara del sistema de visión se relacionan a través de la siguiente transformación:

$$T_{k-1:k} = \begin{pmatrix} R_{k-1:k} & t_{k-1:k} \\ 1 & 0 \end{pmatrix} \tag{1.1}$$

Donde R es la matriz de rotación y t es el vector de traslación. El objetivo de la odometría visual es estimar todas las transformaciones existentes entre cada par de imágenes. Una vez conocidas estas transformaciones, la trayectoria del robot móvil se puede reconstruir de forma incremental concatenando transformaciones consecutivas. Sea C_k la posición que se pretende obtener y C_{k-1} la posición conocida en el instante de tiempo anterior. Conocida la transformación entre dos posiciones consecutivas de la cámara que se muestra en la ecuación (1.1), podemos calcular la nueva posición del robot móvil como sigue:

$$C_k = C_{k-1} T_{k-1:k} \tag{1.2}$$

Para que este procedimiento funcione de forma adecuada es necesario que se cumplan las siguientes condiciones:

- El entorno en el que se desarrolla la actividad del robot debe estar suficientemente iluminado.
- El entorno debe ser prácticamente estático, es decir, las diferentes entidades que aparezcan en la escena deben permanecer en la misma posición durante todo el movimiento del robot.
- Debe haber suficiente textura para que podamos extraer el movimiento aparente entre vistas.

1.3 Motivación y Objetivos

La mayoría de los algoritmos de odometría visual de los que disponemos actualmente están pensados para trabajar de forma *offline*. Trabajar fuera de línea implica que no se procede a reconstruir la trayectoria recorrida por el robot móvil hasta que este finaliza su movimiento, es decir, primero se realizan fotografías del entorno que rodea al robot móvil y una vez finalizado el movimiento de este, se utiliza la información extraída de las imágenes para intentar reconstruir la trayectoria recorrida por el mismo.

Otros algoritmos si son aptos para trabajar en tiempo real. El problema de estos algoritmos es que la frecuencia de captura de imágenes es muy baja, lo que impone una restricción a la velocidad máxima que puede alcanzar el robot durante su movimiento. Esto último es debido a que los algoritmos de odometría visual realizan cálculos muy pesados, tediosos y de alta carga computacional. Por tanto, es necesario dejar tiempo suficiente entre una captura y la siguiente para que se puedan realizar los cálculos pertinentes para estimar la posición del robot móvil. Para poder ejecutar estos algoritmos se requieren equipos informáticos de alto coste.

Dado que en este documento se desarrolla un trabajo de formación académica, se intentan evitar a toda costa equipos informáticos de precio desorbitado como los mencionados anteriormente.

Este contexto suscita la aparición de una necesidad: un método para obtener la localización en tiempo real de un robot móvil a través de un sistema de visión que captura imágenes a una frecuencia relativamente alta.

Teniendo en cuenta lo anterior, en este trabajo se pretende desarrollar un algoritmo de odometría visual que cumpla las siguientes especificaciones:

- Funcionar en tiempo de ejecución.
- Ser capaz de estimar la posición del robot cometiendo poco error.
- Ser capaz de procesar todas las imágenes que recibe en un tiempo aceptable.
- La carga computacional del algoritmo no debe ser demasiado alta ya que se pretende ejecutar en un ordenador personal de gama media.

Además, se intentará que el algoritmo sea lo más modular posible, de forma que podamos añadir futuras mejoras de forma sencilla. El hecho de diseñarlo de forma modular nos podría permitir acoplar nuestro sistema a otros sistemas encargados de resolver el problema de SLAM completo.

2 ESTADO DEL ARTE

En este apartado se realiza una revisión de los trabajos que han contribuido al desarrollo de la odometría visual. Para ello nos ayudamos de una tesis que resume con bastante acierto toda la contribución a este campo [13]. Con esto podremos enmarcar nuestro trabajo en un determinado contexto y averiguaremos en que grado de desarrollo se encuentra este campo de investigación. Además, se desarrollan los conceptos teóricos necesarios para poder comprender este trabajo. El nivel de profundidad que se abordará será meramente introductorio, aunque se adjuntan referencias a documentos que desarrollan extensamente dichos conceptos teóricos.

Podemos realizar una primera clasificación de los métodos empleados en el cálculo de la odometría visual. En esta clasificación se distinguen dos vertientes claramente diferenciadas:

- Métodos que utilizan más de una cámara o de visión estéreo.
- Métodos que utilizan una única cámara o de visión monocular.

En el caso de la odometría visual estéreo, al conocer la posición relativa existente entre las cámaras que intervienen en la captura de imágenes, es relativamente sencillo obtener una estructura tridimensional del entorno y, por consiguiente, realizar medidas. En el caso monocular necesitamos información adicional para poder realizar medidas. El conocimiento de las dimensiones de alguno de los objetos que aparecen en la escena, las mediciones proporcionadas por algún sensor adicional o la imposición de alguna restricción al movimiento suelen ser algunas de las optativas que se utilizan como fuente de información adicional para obtener medidas en el caso monocular.

2.1 Odometría visual estéreo

Las técnicas de odometría visual estéreo parten del esquema propuesto por Moravec [4]:

1. Detección de puntos o zonas de interés en la imagen.
2. Cálculo de la estructura tridimensional del entorno (se formula un modelo matemático del error en el cálculo de la triangulación).
3. Seguimiento o *tracking* de los puntos de interés a lo largo de una secuencia de imágenes.
4. Eliminación de valores anómalos que carecen de sentido.
5. Estimación del movimiento del dispositivo.

Matthies y Shafer [9] parten del mismo esquema que Moravec, pero utilizan un modelo diferente para obtener el error en el cálculo de la triangulación: en lugar de un modelo de error escalar, utilizan un modelo basado en una distribución gaussiana tridimensional. Con esta modificación lograron un error del 2% en un recorrido de 5.5 metros y un error de 1º en la orientación.

Olson et al. [10] llegaron a la conclusión de que el incremento en el orden del error era mayor que lineal debido a los fallos en el cálculo de la orientación. Mientras que un fallo durante el avance tan solo da lugar a un pequeño error aditivo que aumenta con el tiempo, un fallo en el cálculo del giro implica consecuencias mayores, ya que, al realizar los próximos cálculos de distancia recorrida, se está cometiendo un fallo tanto en la posición como en la orientación del dispositivo. Por esta razón, emplearon un sensor de orientación absoluto encargado de corregir los fallos en el cálculo del giro. Al utilizar un sensor de orientación absoluto se consigue que el error aumente de forma lineal con la distancia. Esta solución les permitió disminuir el error total llegando a conseguir un error del 1% en un trayecto de 20 metros.

Cheng et al. [11] utilizaron el trabajo de Lacroix et al. [6] en una implementación de odometría visual pensada para el rover de Marte. Con esta implementación se mejoró la solución propuesta por Olson et al. [10]. Tras la detección de puntos de interés mediante el detector de esquinas de Harris, el algoritmo calculaba el error a través del modelo de error propuesto por Lacroix et al. Además, para mejorar la estimación del movimiento, Cheng et al propusieron utilizar el algoritmo RANSAC, *RAndom SAmple Consensus* [12] para descartar aquellos valores que carecían de sentido. Cheng et al. pusieron a prueba su trabajo en un terreno con irregularidades y realizaron una comparativa entre el error obtenido por la odometría de las ruedas y el error que se obtenía al utilizar su implementación. La figura 2.1.1 muestra los resultados de dicho trabajo.

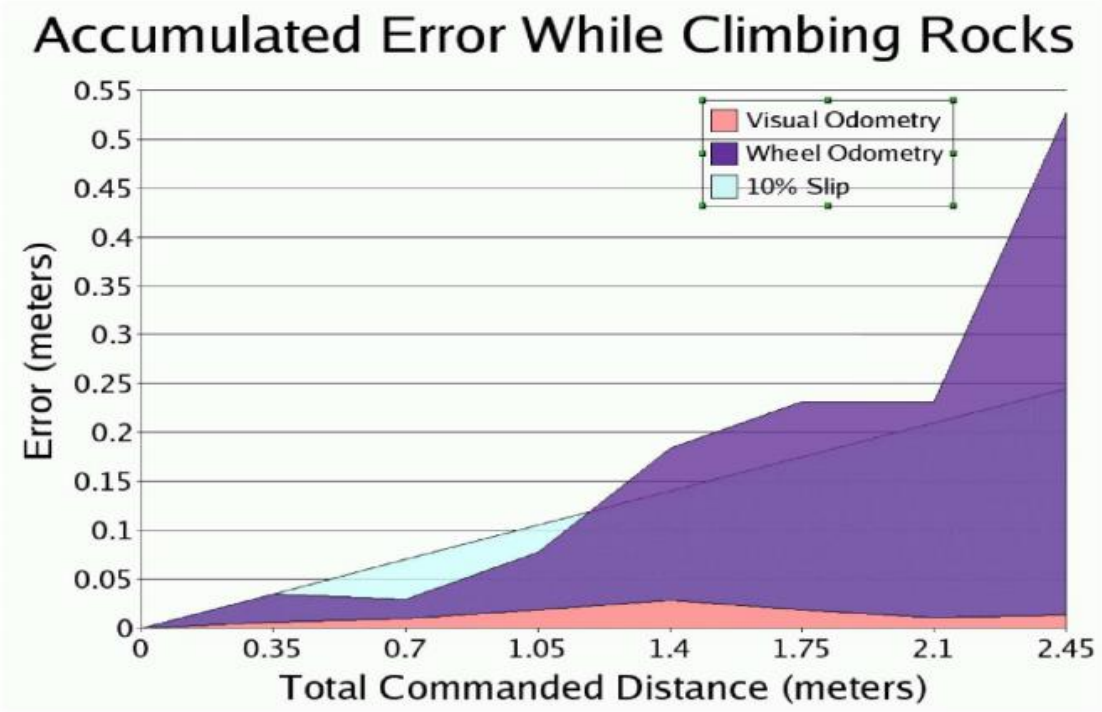


Figura 2.1.1: Resultados del trabajo de Cheng et al.

Milella y Siegwart [14] proponen una nueva solución para la estimación del movimiento de un vehículo todo terreno. Para la detección de puntos de interés en la imagen se sirven de dos herramientas: el motor estéreo SRI [15] (actualmente patentado) y el detector de esquinas Shi-Tomasi [16]. Se considera que se obtendrán puntos tridimensionales fiables de aquellas características que tengan correspondencia mutua en dos imágenes sucesivas, es decir, que dadas dos características c1 y c2 correspondientes a dos imágenes sucesivas en el tiempo, se considera que son fiables si c1 es el punto de interés con mayor coeficiente de correlación para c2 y viceversa. Además, se realiza un filtro estadístico para descartar valores anómalos. Este filtro emplea la desviación típica y la mediana [17]. Para estimar el movimiento utilizan el conocido algoritmo ICP, *Iterative Closest Point.*

Nistér et al. [7] propusieron la primera implementación de odometría visual en tiempo real tanto para el caso estéreo como para el caso monocular. La diferencia con trabajos anteriores es que ellos no realizaban un seguimiento o *tracking* de los puntos de interés a lo largo de la secuencia de imágenes, sino que calculaban todos los puntos de interés de las imágenes y posteriormente buscaban correspondencias entre ellos. Esta alternativa se denomina *matching*. En este proceso se buscaban, para cada par de imágenes, los puntos de interés mediante el detector de esquinas Harris y posteriormente se buscaban correspondencias mutuas entre dichos puntos característicos. Para ello se consideraba un punto característico de la primera imagen y se buscaban analogías con puntos característicos de la segunda imagen que no sobrepasasen un determinado umbral de distancia. Una vez hallada dicha analogía se comprobaba que fuese recíproca, es decir, si la mejor coincidencia para una característica c1 de la imagen I1 era c2 en la imagen I2, la mejor coincidencia para la característica c2 en la imagen I1 debía ser c1. La estimación del movimiento se realizaba aplicando un algoritmo RANSAC preventivo de 3 puntos [18].

2.2 Odometría visual monocular

En la odometría visual monocular solo disponemos de datos bidimensionales para calcular la estructura tridimensional del entorno y el movimiento relativo. Es un hecho que resulta imposible medir distancias de forma absoluta en imágenes bidimensionales. Es por ello por lo que se recurre a realizar medidas relativas. Es decir, se toma como referencia el movimiento realizado en el primer instante de tiempo y a partir de él se calculan los movimientos posteriores. En los últimos años se han conseguido resultados aceptables utilizando cámaras comunes u omnidireccionales[8,19, 20, 21, 22].

Podemos dividir las técnicas de odometría visual monocular en tres bloques: basadas en características, globales e híbridas. En los métodos basados en características se utilizan ciertos puntos o zonas de la imagen que destacan con respecto al resto de la imagen y que son fácilmente identificables a lo largo de toda la secuencia de imágenes. Los métodos globales utilizan la información de intensidad de todos los píxeles que componen la imagen en lugar de utilizar solo ciertas regiones de esta. Por último, tenemos un tercer grupo que engloba técnicas mixtas entre los métodos basados en características y los métodos globales.

Nistér et al. [7] fueron los primeros en presentar una solución al caso monocular basado en características. Para pasar de la información bidimensional de la que disponían a los cálculos tridimensionales, realizaron una serie de hipótesis geométricas. Para estimar el movimiento propio utilizaron un solucionador de cinco puntos combinado con RANSAC [23]. En la literatura el RANSAC de cinco puntos ha sido el algoritmo más utilizado para tratar el problema de odometría visual.

Lhuillier [24] y Mouragnon et al. [25] propusieron el uso de una ventana temporal de imágenes que serviría para reconstruir la trayectoria recorrida por el dispositivo y para realizar un mapa tridimensional del entorno. Para descartar valores anómalos utilizaron un algoritmo RANSAC de cinco puntos [23].

Tardif et al. [20] desarrollaron un método similar a los ya expuestos. La novedad consistía en la creación de un mapa tridimensional. En cuanto a la forma de proceder, presentaba el mismo esquema que otras técnicas de odometría visual:

1. Detección de puntos o zonas de interés en la imagen. En este caso se utilizaba el algoritmo SIFT [26].
2. Seguimiento o *tracking* de los puntos de interés.
3. Estimación de la posición y orientación de la cámara.
4. Descarte de valores anómalos mediante el uso del algoritmo RANSAC preventivo de cinco puntos [18].

En este método el cálculo del movimiento se realizaba considerando únicamente la imagen correspondiente al instante de tiempo anterior, renunciando así a la optimización que se obtenía cuando se utilizaba una ventana temporal de imágenes. A cambio se obtenía una estructura del entorno mediante la creación de un mapa tridimensional del tramo recorrido.

Merece la pena destacar el trabajo de Milford y Wyeth [27]. En su trabajo plantearon un método en el que a través de una cámara anclada a un vehículo eran capaces de obtener la velocidad de rotación y de avance. La idea consistía en realizar el "*tracking*" o seguimiento de un modelo situado en la zona central de la escena.

Los procedimientos globales son bastante sensibles a escenas cambiantes con el tiempo y a las oclusiones de objetos en las imágenes. Estas oclusiones pueden deberse a que aparezca un nuevo objeto en la escena que tape una porción de uno ya existente, a la geometría del propio objeto que según el punto de vista podría ocultar una parte de sí mismo, o a la aparición de sombras que nos impidan identificar las siluetas de los objetos con los que interacciona el robot móvil.

Por esta razón Scaramuzza y Siegwart [21] decidieron combinar los métodos de odometría visual globales con los basados en características. Con un método global solucionaron el problema del cálculo de la rotación del vehículo y con un algoritmo basado en características obtuvieron el desplazamiento de este. En otras palabras, el algoritmo basado en características se utilizaba para refinar los resultados proporcionados por el método global y descartar así algunos fallos.

Con esta forma de proceder se tenían dos "*trackers*" diferentes en lugar de uno solo. Uno de ellos se encargaba de detectar y seguir puntos característicos situados en el suelo a través de homografía y el otro se ocupaba de realizar el seguimiento de regiones a través de las imágenes para poder calcular la rotación.

En la Figura 2.2.1 se muestra una comparativa entre el cálculo de la odometría visual a través de un método híbrido y el cálculo de la odometría visual a través de un procedimiento basado únicamente en características.

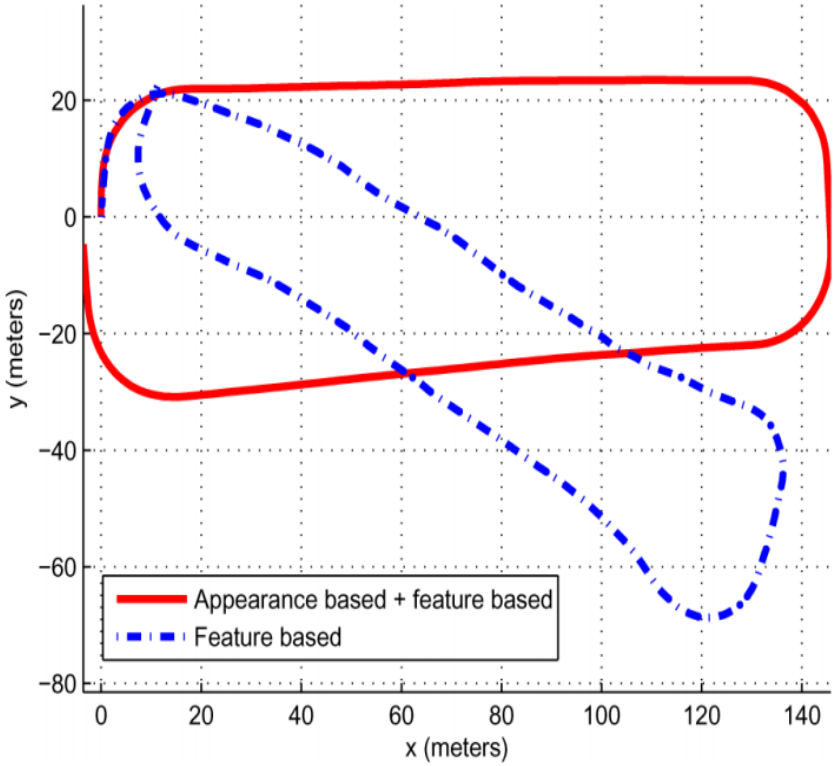


Figura 2.2.1: Comparación del cálculo de odometría visual a través de un método híbrido y un método basado en características según Scaramuzza.

2.3 Modelo de cámara

2.3.1 Funcionamiento físico de los sensores de las cámaras

Las imágenes se obtienen a través de cámaras digitales. Los sensores de estas cámaras están formados por millones de celdas fotosensibles, cada una de ellas de tamaño microscópico. Cada celda incluye un fotodiodo que convierte la luz que recibe (fotones) en electricidad (electrones).

Analicemos el proceso de captura de la imagen: cuando pulsamos el disparador de la cámara para obtener una imagen de la escena, se abre el obturador dejando pasar los fotones. Estos fotones llegan al fotodiodo que se encarga de transformarlos en electrones que se almacenan en un condensador. Cuando se produce el cierre del obturador de la cámara, la luz deja de pasar y se procede a realizar un recuento de los electrones almacenados en el condensador de cada celda fotosensible. Si en una determinada celda se observa que no hay electrones, esto quiere decir que no se han recibido fotones y que, por tanto, nos encontramos en una zona oscura de la imagen. Si por el contrario el condensador está cargado, nos encontraríamos en una zona clara de la imagen.

Además, hay que destacar que estas celdas solo captan la intensidad lumínica (fotones por unidad de tiempo) y no el color. Por tanto, es habitual que se incorporen filtros o mosaicos de Bayer [28] que descomponen la luz en tres componentes: rojo, verde y azul. De esta forma en cada celda solo se recibe la intensidad lumínica correspondiente a una determinada longitud de onda. Una vez aplicado el filtro de Bayer, se utiliza la información de cuatro fotodiodos vecinos para obtener un píxel de color.

Actualmente la mayoría de los sensores están basados en tecnología CMOS (*Complementary Metal-Oxide-Semiconductor*) [29].

En la figura 2.3.1 se muestra el funcionamiento del filtro de Bayer en la captura de una imagen.

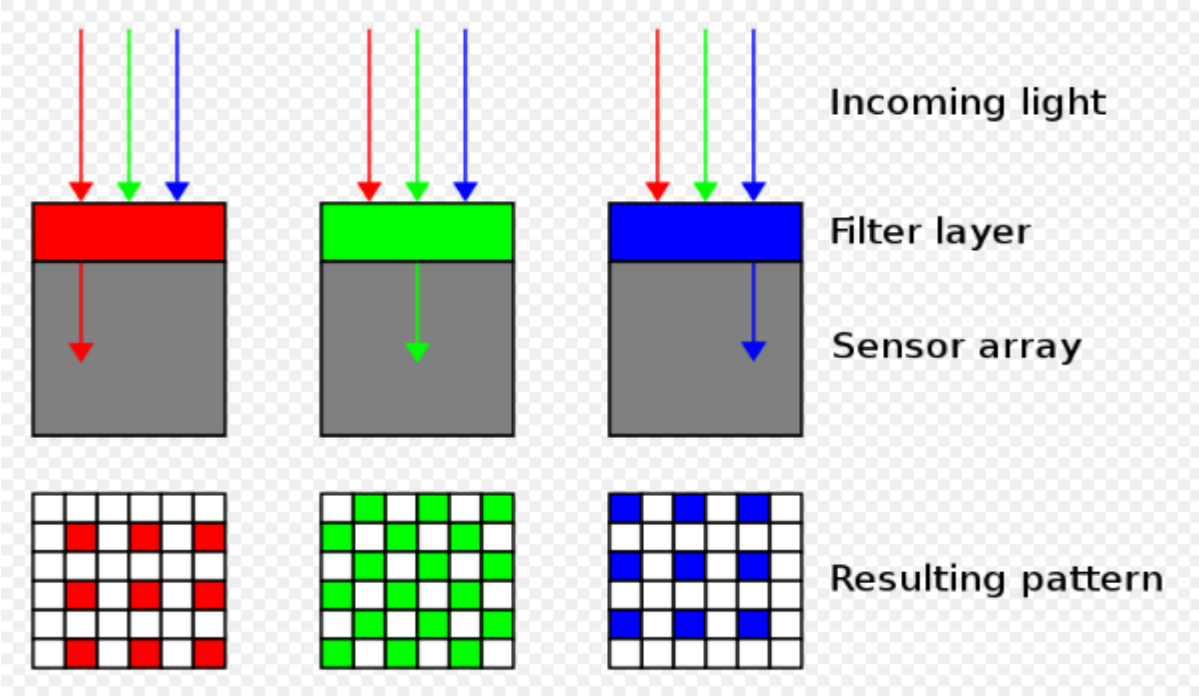


Figura 2.3.1: Filtro de Bayer.

2.3.2 Modelo matemático de la cámara

En este proyecto hacemos uso del mismo modelo de cámara utilizado en la librería de código abierto OpenCV [30]. Este modelo está basado en el modelo de cámara *Pinhole* [31], pero además se añaden unos factores adicionales para tener en cuenta los efectos de distorsión de lente a la hora de obtener una imagen. En la figura 2.3.2 podemos observar un esquema de este modelo.

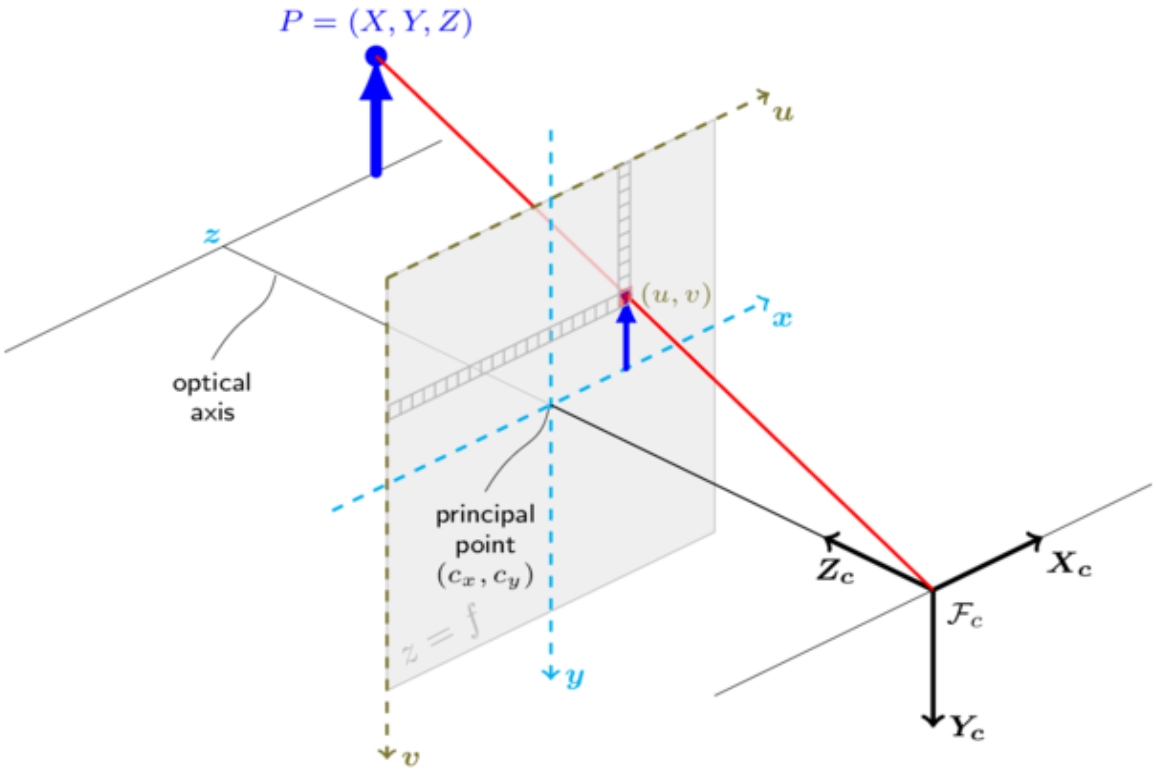


Figura 2.3.2: Modelo de cámara Pinhole según la librería OpenCV.

El modelo de cámara *Pinhole* plantea la relación matemática existente entre las coordenadas de un punto tridimensional y su correspondiente proyección en el plano de la imagen. La luz reflejada por un objeto es capturada por la cámara a través de una apertura infinitesimal y posteriormente se proyecta dicha luz sobre un plano. Teniendo en cuenta el esquema presentado en la figura 2.3.2 la librería OpenCV propone la siguiente transformación de perspectiva para pasar de un punto tridimensional a uno bidimensional:

$$S\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.1}$$

Donde:

- (X, Y, Z) son las coordenadas de un punto 3D en ejes de coordenadas globales.
- (u, v) son las coordenadas del punto en el plano de la imagen expresadas en píxeles.
- (cx, cy) Es el punto principal que suele estar en el centro de la imagen. Se encuentra en la intersección del eje óptico con el plano de la imagen.
- fx, fy son las distancias focales expresadas en píxeles. La distancia focal de una cámara es la distancia existente entre los ejes de coordenadas de esta y el punto principal.

La primera matriz a la derecha de la igualdad de la ecuación (2.1) representa los parámetros intrínsecos de la cámara. Los parámetros intrínsecos son aquellos que definen la geometría interna y la óptica de la cámara. Éstos determinan las condiciones de formación de la imagen, es decir, la forma en la que se proyectan los puntos del mundo tridimensional al plano de la imagen. Los parámetros intrínsecos permanecen constantes siempre y cuando no varíen las características y posiciones relativas entre la óptica y el sensor encargado de capturar la luz.

La matriz que aparece a continuación de la anterior es conocida como la matriz de parámetros extrínsecos o de rotación-traslación [R|t]. Estos parámetros relacionan el sistema de referencia global con el sistema de referencia de la cámara, de forma que podemos describir el movimiento de la cámara alrededor de una escena estática.

utilizando la notación presente en la figura 2.3.2 y suponiendo que z sea distinto de cero, la ecuación (2.1) se transforma en el siguiente conjunto de ecuaciones:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t$$
$$x' = x/z$$
$$y' = y/z \tag{2.2}$$
$$u = f_x * x' + c_x$$
$$v = f_y * y' + c_y$$

Llegados a este punto ya tenemos una expresión matemática para el modelo de cámara *Pinhole*. A continuación, se comentan los tipos de distorsión de lente que podrían existir en las imágenes y como se incorporan matemáticamente a este modelo.

Las imágenes capturadas difieren de la escena real que se pretende representar debido a que los centros de las lentes de las cámaras no están perfectamente alineados. Normalmente la distorsión se suele dividir en dos componentes: una radial y una tangencial. La más importante de ellas es la distorsión radial, que se manifiesta en la imagen como una curvatura en una línea que en realidad debería ser recta.

Dependiendo del tipo de distorsión radial las coordenadas de la imagen se dispersarán (efecto barril) o se concentrarán en torno al centro de la imagen (*pincushion*). En la figura 2.3.3 podemos apreciar ejemplos de estos tipos de distorsión.

La distorsión depende de forma no lineal de la distancia radial r. Teniendo en cuenta el conjunto de ecuaciones (2.2) el cuadrado de la distancia radial r se calcula como sigue:

$$r^2 = x'^2 + y'^2 \tag{2.3}$$

Finalmente, para incorporar los efectos de distorsión de lente radial y tangencial, la librería OpenCV propone añadir al conjunto de ecuaciones (2.2) las siguientes expresiones:

$$x'' = x' \frac{1+k_1 r^2+k_2 r^4+k_3 r^6}{1+k_4 r^2+k_5 r^4+k_6 r^6} + 2p_1 x' y' + p_2(r^2 + 2x'^2) \tag{2.4}$$

$$y'' = y' \frac{1+k_1 r^2+k_2 r^4+k_3 r^6}{1+k_4 r^2+k_5 r^4+k_6 r^6} + p_1(r^2 + 2y'^2) + 2p_2 x' y' \tag{2.5}$$

Con estos coeficientes x´´ e y´´ podemos calcular las coordenadas del punto proyectado en el plano de la imagen sustituyendo las dos últimas ecuaciones del conjunto (2.2) por las que se muestran a continuación:

$$u = f_x * x'' + c_x \tag{2.6}$$
$$v = f_y * y'' + c_y \tag{2.7}$$

Recapitulando, tendríamos un modelo de cámara *Pinhole* extendido que tiene en cuenta los efectos de distorsión de lente. El modelo quedaría resumido en el siguiente conjunto de ecuaciones matemáticas:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t$$
$$x' = x/z$$
$$y' = y/z$$
$$r^2 = x'^2 + y'^2 \tag{2.8}$$
$$x'' = x' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + 2p_1 x' y' + p_2(r^2 + 2x'^2)$$
$$y'' = y' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + p_1(r^2 + 2y'^2) + 2p_2 x' y'$$
$$u = f_x * x'' + c_x$$
$$v = f_y * y'' + c_y$$

Donde:

- k_1 k_2, k_3, k_4, k_5 y k_6 son coeficientes de distorsión radial.
- p_1 y p_2 son coeficientes de distorsión tangencial.

Tanto los coeficientes de distorsión radial y tangencial como las matrices intrínseca y extrínseca de la cámara se obtienen en el proceso de calibración de la cámara que se explica en el siguiente apartado.

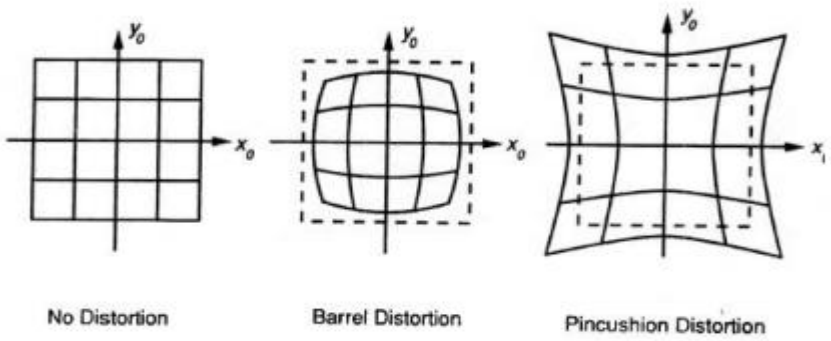


Figura 2.3.3: Tipos de distorsión de lente.

2.3.3 Calibración de la cámara

Entendemos por calibración de la cámara aquel proceso en el que se tratan de averiguar los parámetros intrínsecos y extrínsecos del modelo de cámara. Estos parámetros nos permiten determinar cómo son mapeados los puntos tridimensionales de la escena en el plano de la imagen.

Uno de los procedimientos más utilizados para obtener dichos parámetros consiste en realizar una secuencia de capturas fotográficas a un patrón real de dimensiones conocidas. Este patrón suele ser una cuadrícula de ajedrez que se coloca en distintas posiciones y orientaciones.

Dado que se conocen tanto las coordenadas bidimensionales de las esquinas del patrón como las coordenadas de dichas esquinas en el plano de la imagen, se procede a calcular los parámetros de la cámara a través de un problema de optimización de mínimos cuadrados. La precisión de este proceso aumenta conforme más imágenes se utilicen.

La librería OpenCV dispone de una implementación en C/C++ que nos permite resolver la calibración. Dado que en este proyecto se utiliza un *dataset* que ya dispone de la información de calibración de la cámara no se realizará la implementación para resolver la calibración.

2.4 Detección de puntos característicos de una imagen

El proceso de detección de características consiste en buscar una serie de puntos de interés (*keypoints*) en una imagen.

Definimos como punto de interés aquel que es fácilmente distinguible y comparable en una imagen. Estos puntos se caracterizan por tener una alta probabilidad de ser encontrados nuevamente en imágenes diferentes de la misma escena.

A los algoritmos encargados de localizar dichos puntos se les denomina detectores de características. Dentro de los detectores de características distinguimos dos grandes grupos: detectores de esquinas y detectores de manchas o *blobs*. Los primeros, como su propio nombre indica, se encargan de localizar esquinas. Se define esquina como la intersección entre dos o más bordes. Los segundos identifican regiones de la imagen que se diferencian de su entorno más próximo en términos de intensidad, color y textura.

Las propiedades que debe tener un buen detector de puntos de interés son las siguientes:

- Precisión en la localización: la localización de los puntos característicos debe ser precisa tanto en la imagen como en la escala. Esto es de crucial importancia cuando posteriormente se pretende realizar una reconstrucción tridimensional de la escena.
- Número de puntos detectados: la cantidad de puntos característicos a retener por el detector depende en gran medida de la aplicación en la que se vayan a emplear dichos puntos. Para la odometría visual, que es lo que a nosotros nos interesa, es importante disponer de gran cantidad de puntos para que la estimación de la orientación de la cámara sea precisa.
- Invarianza: los puntos de interés encontrados deben ser insensibles a cambios en la iluminación, a cambios de escala y a cambios de perspectiva.
- Robustez: las características encontradas deben ser insensibles al ruido, a procesos de compresión, a efectos de discretización y a cambios en la iluminación y en la geometría.
- Coste computacional: se pretende en todo momento que el tiempo empleado en los algoritmos de detección de características sea el mínimo posible. Esto último es debido a que en robótica la mayoría de las aplicaciones están pensadas para trabajar en tiempo real. Por tanto, un tiempo de detección elevado podría comprometer el sistema y provocar un funcionamiento erróneo de la aplicación. Sin embargo, hay que tener en cuenta que la eficiencia del algoritmo depende de la invarianza. Cuanta más invarianza se pretenda conseguir, mayor será el tiempo de detección de características en la imagen. Por consiguiente, hay que llegar a una solución de compromiso en función de cuales sean las especificaciones del sistema en el que vamos a implementar el detector de características.

2.4.1 Detectores de esquinas

Moravec diseñó uno de los primeros detectores de esquinas [5]. En este detector se consideraba como esquina todo aquel punto en el que el gradiente de intensidad fuese elevado en todas las direcciones, es decir, una esquina era la intersección de dos o más bordes. A continuación, se muestran algunos de los detectores de esquinas más utilizados.

2.4.1.1. Detector Harris

En este detector de esquinas [32] se realiza un salto de calidad con respecto al detector de Moravec. El procedimiento consiste en buscar la diferencia de intensidad para un desplazamiento (u,v) en todas las direcciones de la imagen, es decir, se consideran las derivadas parciales de la suma de diferencias al cuadrado entre dos regiones cuadradas de la imagen. Esto se expresa como sigue:

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+v) - I(x,y)]^2 \tag{2.9}$$

Donde:

- E (u,v) es la función objetivo a maximizar para obtener una esquina.
- w (x,y) es una ventana rectangular o una ventana gaussiana que asigna pesos a los píxeles de la imagen.
- I (x+u,y+v) representa la intensidad en la región rectangular desplazada.
- I(x,y) representa la intensidad en la región rectangular de referencia.

Aplicando un desarrollo en serie de Taylor y simplificando llegamos a la siguiente ecuación:

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix} \tag{2.10}$$

Donde:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix} \tag{2.11}$$

Siendo I_x e I_y los gradientes de intensidad de la imagen en las direcciones x e y respectivamente.

Para determinar si alguna de las regiones rectangulares de la imagen podía contener una esquina, se propuso la siguiente variable que servía a modo de puntuación:

$$R = \det(M) - k\big(traza(M)\big)^2 \tag{2.12}$$

Donde:

- $\det(M) = \lambda_1 \lambda_2$
- $traza(M) = \lambda_1 + \lambda_2$
- λ_1 y λ_2 son los valores propios o autovalores de la matriz M.

En función de estos autovalores se puede determinar si una región es una esquina, un borde o si por el contrario se trata de una región plana:

- Si λ_1 y λ_2 son pequeños la suma de diferencias al cuadrado (SSD) es pequeña en todas las direcciones y nos encontramos ante una región plana
- Si $R < 0$, lo que sucede cuando $\lambda_1 >> \lambda_2$ o cuando $\lambda_2 >> \lambda_1$ la región es un borde.
- Si R presenta un valor grande, lo que sucede cuando tanto λ_1 como λ_2 son grandes, la región es una esquina.

Para concluir, se eliminan todos aquellos puntos que no son máximos en sus regiones, de forma que solo permanecen aquellos puntos con mayor respuesta.

En la figura 2.4.1 se muestra un esquema del tipo de región en función de los valores de los autovalores λ_1 y λ_2.

Figura 2.4.1: Tipo de región en función de los autovalores λ_1 y λ_2.

2.4.1.2 Detector Shi-Tomasi

En este detector [33] se toma como base el detector Harris. Sus autores descubrieron que había una forma mucho más efectiva de obtener esquinas. El procedimiento consistía en sustituir la variable R descrita en la ecuación (2.12), de forma que el nuevo factor de puntuación fuese el mínimo de los autovalores de la matriz M.

2.4.1.3 Detector FAST

El detector FAST (*Features from Accelerated Segmented Test*) [34] trata de buscar esquinas siguiendo el siguiente método:

- Se selecciona un píxel de la imagen. Supongamos que este píxel se denomina "p". Sea Ip la intensidad lumínica de dicho píxel.
- A continuación, se utiliza un círculo en cuyo centro se encuentra el píxel p y cuyos límites están delimitados por una circunferencia de 16 píxeles tal y como se muestra en la figura 2.4.2.
- Posteriormente se considera un umbral "t". Diremos que p es una esquina en el caso de que encontremos un conjunto de "n" píxeles contiguos sobre la circunferencia que sean más claros que Ip+t o más oscuros que Ip-t. Estos píxeles son los que aparecen sobre la línea discontinua de la figura 2.4.2. Se llegó a la conclusión de que el conjunto n debía ser de 12 píxeles.

Resumiendo, se considera un círculo cuyo centro es el píxel que se pretende examinar. Si una sección continua de la circunferencia de dicho círculo es más oscura o brillante que el centro, el punto se clasifica como esquina. Se ha demostrado que este algoritmo es del orden de 20 a 30 veces más rápido que el detector de esquinas Harris.

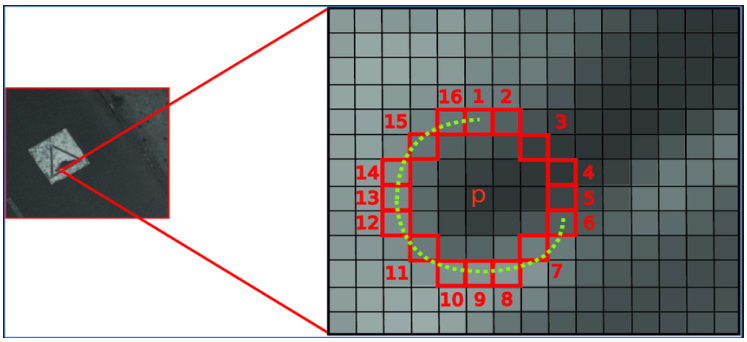


Figura 2.4.2: Esquema del algoritmo de detección de esquinas FAST.

2.4.2 Detectores de manchas

Como ya se ha comentado se define mancha o *blob* como una región de la imagen que difiere de su entorno más cercano en términos de intensidad, color y textura. A pesar de que la precisión en la localización de una mancha es bastante menor que la precisión en la detección de esquinas, la mancha está mucho mejor definida tanto en forma como en escala. A continuación, se muestran algunos de los métodos detección de manchas más conocidos.

2.4.2.1 Detector SIFT

SIFT (*Scale Invariant Feature Transform* [26]) es un algoritmo utilizado en la detección y emparejamiento de puntos de interés. Las características visuales obtenidas con este detector son muy particulares, de forma que pueden ser emparejadas incluso cuando existan cambios drásticos en la iluminación, escala y rotación.

En primer lugar, el algoritmo realiza una convolución con una función gaussiana centrada en cada píxel de la imagen, de forma que el valor final de cada píxel en la nueva imagen depende de la ponderación que la función gaussiana asigne a sus píxeles vecinos. El resultado sería una nueva imagen suavizada en la que se ha perdido calidad en el detalle con respecto a la imagen original. El filtro gaussiano aplicado sería el siguiente:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.13}$$

Donde:

- σ es la desviación típica y un aumento de su valor implicaría un mayor suavizado de la imagen.
- x e y son las coordenadas del píxel

Esta convolución se realiza para valores crecientes de la desviación típica de forma que obtenemos la misma imagen con distintos grados de suavizado. A la diferencia entre un par de imágenes suavizadas se le denomina diferencia de gaussianas (DoG) y esta sirve de aproximación al operador Laplaciano de Gauss. Esta operación permite detectar regiones con cambios de intensidad lumínica y la desviación típica del filtro gaussiano indicaría en que escala (resolución) se ha detectado dicho cambio.

El algoritmo SIFT se encarga de calcular todas las DoG para cada par de imágenes suavizadas sucesivas. Cada píxel de una DoG se compara con sus ocho vecinos más próximos en la misma escala y, además, con otros 18 vecinos, 9 de la escala anterior y 9 de la escala posterior. Si intensidad del píxel resulta ser un máximo o un mínimo local para el conjunto de todos los vecinos considerados, se selecciona como posible punto de interés.

Además, el proceso se repite submuestreando la imagen original en un factor de dos una y otra vez. De esta forma obtenemos una secuencia de imágenes constituida por la misma imagen cada vez más pequeña.

En cada paso se aplicaría un filtro gaussiano y posteriormente un reescalado, de forma que podríamos detectar características de diferentes tamaños. El proceso de filtrado y reescalado se repite hasta que tenemos todas las DoG de la imagen original y de tres submuestreos.

Para concluir el algoritmo incluye un paso para refinar la localización, tanto en espacio como en escala. Para ello utiliza un proceso de interpolación utilizando información cercana a los puntos candidatos. Como función de interpolación se utiliza la expansión en serie de Taylor de la función de diferencias gaussianas DoG(x,y,σ):

$$DoG(p) = DoG + \frac{\partial DoG^T}{\partial p} p + \frac{1}{2} p^T \frac{\partial^2 DoG}{\partial p^2} p \tag{2.14}$$

Donde DoG y sus derivadas se evalúan en el punto de interés y p = (x,y,σ) es el *offset* con respecto al punto de interés.

Tomando la derivada de la función anterior con respecto a p e igualando a cero podemos obtener la localización del extremo:

$$\hat{p} = -\begin{bmatrix} \frac{\partial^2 DoG}{\partial x^2} & \frac{\partial^2 DoG}{\partial x \partial y} \\ \frac{\partial^2 DoG}{\partial y \partial x} & \frac{\partial^2 DoG}{\partial y^2} \end{bmatrix}^{-1} \begin{bmatrix} \frac{\partial DoG}{\partial x} \\ \frac{\partial DoG}{\partial y} \end{bmatrix} \tag{2.15}$$

Sustituyendo el extremo en la expresión (2.14) obtenemos una expresión que nos sirve para descartar puntos de bajo contraste:

$$DoG(\hat{p}) = DoG + \frac{1}{2} \frac{\partial DoG^T}{\partial p} \hat{p} \tag{2.16}$$

A partir de esta última ecuación, tenemos que establecer un umbral mínimo que deban alcanzar los puntos candidatos para no ser rechazados. Se ha establecido que este umbral debe ser de 0.03, de forma que todos los puntos que no cumplan la condición $|Dog(\hat{p})| > 0.03$ serán descartados por tener bajo contraste.

Finalmente se descartan los puntos que se localizan sobre contornos. Para ello SIFT emplea la matriz Hessiana obtenida de derivar la función de diferencias gaussianas.

Se consideran los puntos de interés para los que ambos autovalores de la matriz Hessiana no difieren en más de un orden de magnitud ya que, en caso contrario, tendríamos un borde y no una esquina, tal y como se infiere del detector de esquinas Harris mencionado anteriormente.

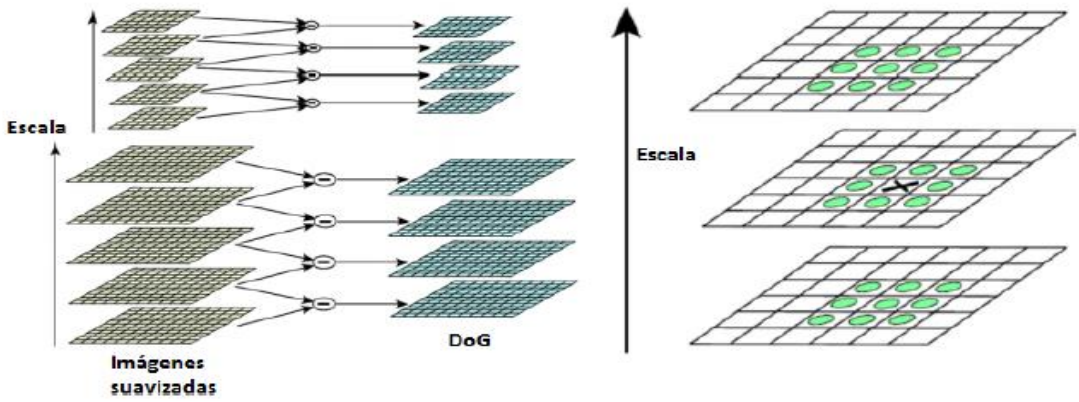


Figura 2.4.3: Esquema del algoritmo de detección de manchas SIFT. Se observa la obtención de las diferencias gaussianas y la selección de un punto de interés con mayor respuesta entre sus vecinos

2.4.2.2 Detector SURF

SURF (*Speeded-Up Robust Features* [35]) es un algoritmo de detección de características visuales fuertemente influenciado por el detector SIFT, pero mucho más rápido y más robusto frente a posibles transformaciones de la imagen.

Este detector hace uso del determinante de la matriz Hessiana para determinar la localización y la escala de los puntos. Se decide utilizar la matriz Hessiana por su rendimiento en cuanto a precisión y eficiencia en el cálculo. La novedad de este detector es que utiliza la matriz Hessiana tanto para el cálculo de la posición como de la escala, al contrario que en otros detectores que necesitan diferentes medidas para el cálculo de estas variables.

Dado un punto p = (x,y) de la imagen, la matriz Hessiana H (x,y,σ) se calcula como sigue:

$$H(x,y,\sigma) = \begin{bmatrix} \frac{\delta^2}{\delta x^2} G(\sigma) * I(x,y) & \frac{\delta}{\delta x}\frac{\delta}{\delta y} G(\sigma) * I(x,y) \\ \frac{\delta}{\delta x}\frac{\delta}{\delta y} G(\sigma) * I(x,y) & \frac{\delta^2}{\delta y^2} G(\sigma) * I(x,y) \end{bmatrix} \tag{2.17}$$

Donde cada término de la matriz Hessiana representa la convolución de segundo orden de la Gaussiana de la ecuación (2.13) con la imagen I en el punto de coordenadas (x, y).

En el detector SURF, en lugar de realizar una convolución con la segunda derivada de la Gaussiana, se utilizan una serie de filtros más sencillos para aproximar dichas convoluciones. En la figura 2.4.4 podemos apreciar una comparativa de las derivadas parciales de segundo orden de la Gaussiana en las direcciones xy e y con sus respectivas aproximaciones utilizando los filtros propuestos por SURF.

Para determinar si un píxel es un punto característico se utiliza la siguiente función de puntuación:

$$c(x,y,\sigma) = D_{xx}(\sigma)D_{yy}(\sigma) - \left(0.9D_{xy}(\sigma)\right)^2 \approx H(x,y,\sigma) \tag{2.18}$$

En esta última ecuación Dxx, Dxy y Dyy son los resultados de aplicar la convolución con los filtros mencionados anteriormente. Si c es mayor que cero se considera que el punto es de interés y en caso contrario se descarta el candidato. Además, al igual que en SIFT, se consideran los ocho píxeles vecinos del punto candidato y sus nueve vecinos en las escalas anterior y posterior. Si la intensidad del píxel resulta ser un máximo o un mínimo local para el conjunto de los vecinos considerados, se selecciona como posible punto de interés.

Para concluir, el algoritmo también dispone de un último paso para refinar la posición del punto obtenido.

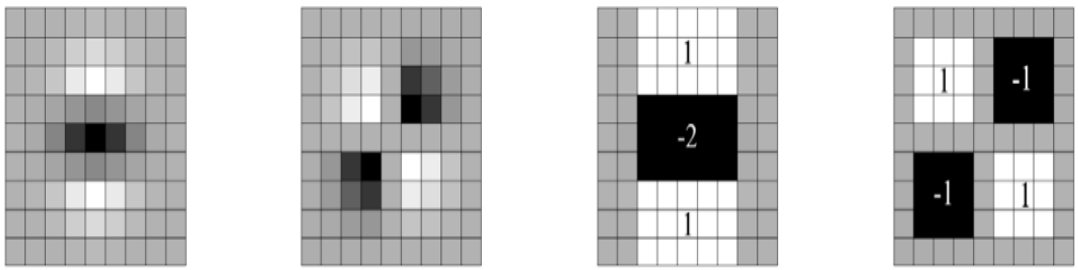


Figura 2.4.4: Derivadas parciales de segundo orden de la Gaussiana en las direcciones y e xy (izquierda) frente a los filtros aproximados propuestos por el detector SURF (derecha).

2.4.2.3 Detector CenSurE

CenSurE (*Center Surround Extrema* [36]) es un algoritmo de detección de características visuales en el que se intenta aproximar el método de diferencias gaussianas ya visto en los otros detectores, utilizando filtros binivel con valores -1 y 1.

El filtro más parecido al laplaciano de la Gaussiana es aquel que presenta una forma circular, pero presenta el inconveniente de que es computacionalmente costoso calcularlo. Por tanto, se utilizan distintas formas como octágonos, hexágonos y cuadrados. Estas formas cada vez se parecen menos al círculo, pero presentan la ventaja de que son más fáciles de obtener en términos computacionales. Al utilizar formas distintas del círculo como filtros, estamos sacrificando precisión en la localización en favor de una mayor eficiencia en la ejecución del algoritmo. Hay que llegar a una solución de compromiso en función de cuales sean los objetivos de la aplicación en la que se vaya a utilizar este detector de características visuales. En la figura 2.4.5 podemos observar los distintos filtros utilizados en el detector CenSurE.

En lugar de submuestrear la imagen original como en los detectores de manchas anteriormente mencionados, el detector CenSurE aplica filtros de diversos tamaños a la imagen original para obtener invarianza frente a la escala. Con esto último se consigue que el algoritmo no pierda precisión en la localización del punto de interés, ya que no se requiere ninguna interpolación entre escalas puesto que estamos trabajando siempre con la misma imagen.

El método de selección y descarte de puntos de interés es similar al utilizado en los detectores SIFT y SURF.

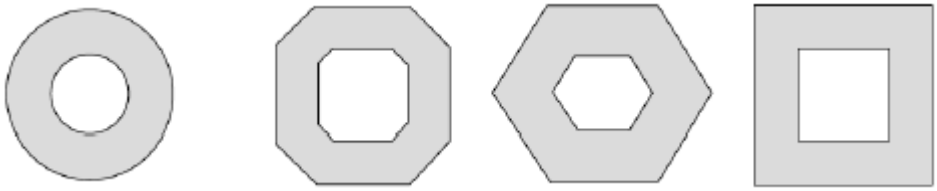


Figura 2.4.5: filtros binivel empleados en el detector CenSurE. De izquierda a derecha disminuye el coste computacional, pero aumenta el error en la localización del punto de interés.

2.4.3 Comparativa entre detectores

Los detectores de esquinas presentan la ventaja de ser veloces y de encontrar una gran cantidad de puntos de interés. Además, tienen una buena precisión en la localización. Sin embargo, presentan el inconveniente de que bastantes de las características que detectan carecen de repetibilidad, es decir, que los puntos de interés encontrados podrían no encontrarse de nuevo en otras imágenes de la misma escena. Esto último sucede sobre todo cuando existen transformaciones afines.

Podemos afirmar que los detectores de esquinas son robustos frente a rotaciones, ya que las esquinas no se ven afectadas por este tipo de movimientos. Sin embargo, estos detectores son débiles frente a cambios de escala, puesto que una esquina podría pasar a convertirse en un borde o incluso desaparecer si modificamos la escala.

En el otro extremo tenemos los detectores de manchas. Estos últimos presentan unas velocidades de ejecución mucho más lentas que las de los algoritmos de detección de esquinas, pero los puntos de interés encontrados si disponen de repetibilidad. La probabilidad de volver a encontrar las mismas características en otras imágenes de la misma escena es alta incluso cuando existen cambios en la iluminación y en la perspectiva. Por mucho que cambiemos la escala o rotemos la imagen, las manchas son muy particulares y por consiguiente fácilmente identificables.

El principal inconveniente de los detectores de manchas es, sin duda, el estimar la localización de la mancha. Esto último se debe a que la mancha está constituida por un conjunto de puntos y no por uno único. Esto último implica que determinar unas coordenadas en unidades de píxeles puede ser una tarea compleja.

Llegamos a la conclusión de que la elección del detector de características a utilizar en una aplicación concreta no es una tarea trivial. Tenemos que evaluar las diversas necesidades de la tarea a realizar.

Se deben tener en cuenta tanto las restricciones computacionales (recursos disponibles y restricciones temporales) como el tipo de entorno en el que se vaya a desarrollar la actividad. Además, también hay que tener en cuenta el tipo de movimiento que se vaya a realizar. En la figura 2.4.6 se muestra una comparativa entre los distintos detectores de características. Esta figura ha sido extraída de [37].

	Corner Detector	Blob Detector	Rotation Invariant	Scale Invariant	Affine Invariant	Repeatability	Localization Accuracy	Robustness	Efficiency
Haris	x		x			+++	+++	++	++
Shi-Tomasi	x		x			+++	+++	++	++
FAST	x		x	x		++	++	++	++++
SIFT		x	x	x	x	+++	++	+++	+
SURF		x	x	x	x	+++	++	++	++
CENSURE		x	x	x	x	+++	++	+++	+++

Figura 2.4.6: Comparación de detectores de características: propiedades y rendimiento según Friedrich Fraundorfer y Davide Scaramuzza

2.5 Descriptores de características

La siguiente tarea tras la obtención de las características visuales de una imagen, es obtener información relevante sobre el entorno que rodea a los puntos de interés. Este proceso se conoce como descripción de características. La información extraída deberá ser simple y compacta de forma que pueda ser fácilmente comparada con otros descriptores de características de imágenes diferentes.

Los descriptores más simples se basan en la apariencia, es decir, en la intensidad lumínica de todos los píxeles vecinos que rodean al punto de interés. Para obtener un descriptor de este tipo se considera un área de la imagen centrada en la característica visual en cuestión. Con esta región se realizan cierto tipo de operaciones que nos permiten obtener unos valores o variables que describen la característica visual de forma unívoca. Estos valores podrían compararse con los obtenidos para otra región de una imagen diferente, de forma que podamos identificar la misma característica visual en imágenes diferentes.

Supongamos que queremos comparar una región rectangular de una imagen I_1 con dimensiones m x n y centro en (u, v) con otra región de las mismas dimensiones centrada en (u', v') de la imagen I_2. Algunos de los criterios más populares para evaluar la similitud entre dos regiones de diferentes imágenes son los siguientes:

- Suma de diferencias absolutas (SAD):

$$SAD = \sum_{k=-a}^{a} \sum_{l=-b}^{b} |I_1(u+k, v+l) - I_2(u'+k, v'+l)| \tag{2.19}$$

- Suma de las diferencias al cuadrado (SSD):

$$SSD = \sum_{k=-a}^{a}\sum_{l=-b}^{b}[I_1(u+k, v+l) - I_2(u'+k, v'+l)]^2 \tag{2.20}$$

- Correlación cruzada normalizada (NCC):

$$NCC = \frac{\sum_{k=-a}^{a}\sum_{l=-b}^{b}[I_1(u+k,v+l)-\mu_1][I_2(u\prime+k,v\prime+l)-\mu_2]}{\sqrt{\sum_{k=-a}^{a}\sum_{l=-b}^{b}[I_1(u+k,v+l)-\mu_1]^2\sum_{k=-a}^{a}\sum_{l=-b}^{b}[I_2(u'+k,v'+l)-\mu_2]^2}} \tag{2.21}$$

Donde:

$$\mu_1 = \frac{1}{mn}\sum_{k=-a}^{a}\sum_{l=-b}^{b} I_1(u+k, v+l)$$

$$\mu_2 = \frac{1}{mn}\sum_{k=-a}^{a}\sum_{l=-b}^{b} I_2(u'+k, v'+l)$$

La SAD es la función de similitud más simple de todas. Se obtiene al sumar las diferencias en valor absoluto entre los píxeles de la imagen de referencia I_1 y los píxeles de la imagen objetivo I_2. Esta operación se realiza para los píxeles contenidos en una región delimitada de dimensiones m x n. La SSD parte de la misma idea de restar píxeles, pero en lugar de tomar el valor absoluto toma el cuadrado de la diferencia, por lo que tiene mayor coste computacional que la SAD. Independientemente de que utilicemos la SAD o la SSD, debemos obtener un valor nulo en el caso de que las regiones a comparar encajen a la perfección.

Por último, tenemos la NCC que es la más costosa de las tres computacionalmente hablando. No obstante, es mucho más distintiva que las dos anteriores ya que es invariante a transformaciones afines y cambios en la intensidad. Al estar normalizada, como su propio nombre indica, los valores de la NCC oscilan entre -1 y 1 donde el valor de 1 corresponde a la mayor similitud posible entre las dos regiones que se comparan.

Generalmente, la información que obtenemos de la intensidad de los píxeles vecinos a nuestro punto de interés no es suficiente para constituir un buen descriptor para dicha característica visual. Esto se debe a que la intensidad puede variar ante cambios de escala, orientación o perspectiva. Este inconveniente hace que los tres métodos anteriormente mencionados solo sean útiles en el caso de que las imágenes a comparar hayan sido tomadas desde posiciones cercanas entre ellas.

Dada la necesidad de contemplar la posibilidad de que existan cambios de perspectiva, orientación y escala entre las distintas imágenes que componen la misma escena, existen otros descriptores más sofisticados constituidos por un vector que contiene información sobre la intensidad y la textura de la vecindad que rodea al punto de interés. A continuación, se procede a describir cómo se forman estos descriptores.

2.5.1 Descriptor SIFT

En primer lugar, el descriptor SIFT comienza por seleccionar las regiones de la imagen sobre las que va a buscar los descriptores. Tal y como se explicó en el apartado 2.4.2.1 el detector SIFT obtiene puntos característicos para cada nivel de la pirámide de imágenes y para una determinada escala de suavizado dentro de dicho nivel de pirámide.

Una vez encontrada la imagen en la que se obtuvo el punto de interés (nivel de pirámide y escala) se selecciona una ventana de tamaño fijo centrada en dicho punto. Dado que la ventana es de tamaño fijo y que el tamaño de la imagen original va disminuyendo conforme avanzamos sobre la pirámide de imágenes, llegamos a la conclusión de que conforme mayor es el nivel de la pirámide de imágenes, mayor es la región de la imagen original considerada por el descriptor. Tal y como se establece en [26] el tamaño de la ventana considerada es de 16 x 16 píxeles. En la figura 2.5.1 se ilustra la idea descrita en este párrafo.

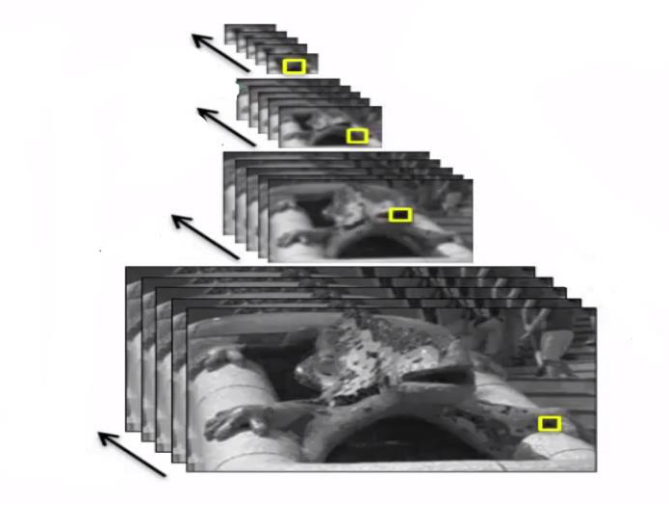


Figura 2.5.1: Tamaño de la región considerada por el descriptor SIFT (rectángulo amarillo) en función del nivel de la pirámide de imágenes.

De esta forma el área empleada en el cálculo del descriptor se adapta de forma natural a la escala en la que se ha detectado el punto de interés, lo que permite que el descriptor sea invariante ante cambios de tamaño en los objetos detectados.

Una vez definida la región de la imagen que utilizamos para calcular el descriptor estamos en condiciones de comenzar a realizar cálculos.

El descriptor está basado en la idea del gradiente, que es un concepto muy utilizado en visión por computador. El gradiente se define en cada punto de la imagen como el cambio en la intensidad de la imagen en una determinada dirección. Se considera aquella dirección en la que el cambio de intensidad alrededor del punto es máximo. Se puede decir entonces que el gradiente está constituido por dos valores:

- Dirección donde el cambio de intensidad es máximo.
- Magnitud de dicho cambio.

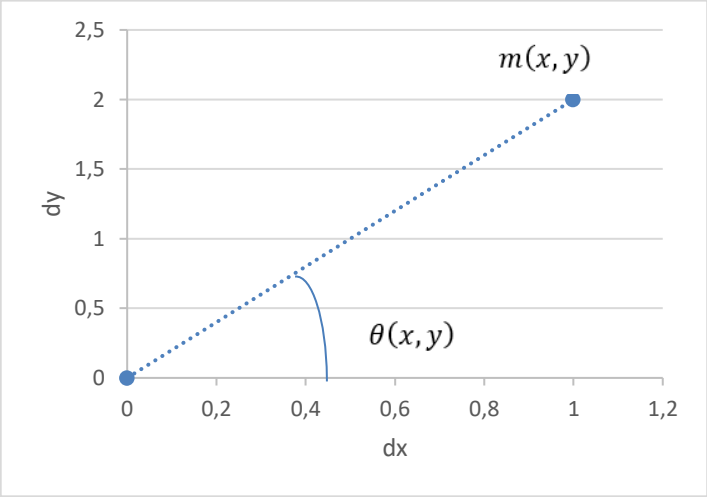
El gradiente puede calcularse fácilmente como la diferencia de intensidad de los píxeles adyacentes en dirección horizontal y vertical:

$$dx = I(x+1, y) - I(x-1, y) \tag{2.22}$$

$$dy = I(x, y+1) - I(x, y-1) \tag{2.23}$$

Donde dx y dy serían los valores horizontal y vertical del gradiente respectivamente e I (x,y) representaría el valor de la intensidad en el píxel (x,y).

Una vez tenemos los valores dx y dy, los situamos en un eje de coordenadas de forma que estos definen un vector al que denominamos vector gradiente. El ángulo del vector gradiente con respecto a la horizontal nos daría la orientación del gradiente y la longitud de dicho vector nos proporcionaría la magnitud del cambio de intensidad. En la figura 2.5.2 se muestra un ejemplo de vector gradiente y las ecuaciones necesarias para obtener la orientación y la magnitud de este.



$$\theta(x,y) = \arctan\left(\frac{dx}{dy}\right)$$

$$m(x,y) = \sqrt{dx^2 + dy^2}$$

Figura 2.5.2: Ejemplo de representación gráfica del vector gradiente y ecuaciones para su cálculo. θ(x,y) representa la orientación del vector gradiente y m(x,y) la magnitud del este.

Una vez tenemos los vectores gradientes de todos los píxeles de la región de interés, se procede a compactar toda la información en un histograma de orientaciones del gradiente. Para ello se procede a dividir el rango de orientaciones (de 0º a 360º) en un número de intervalos fijo tal y como se muestra en la figura 2.5.3. A cada píxel de la región se le asigna uno de dichos intervalos según el valor de la orientación de su vector gradiente. En cuanto todos los píxeles ya tengan asignado un intervalo ya se podría construir el histograma.

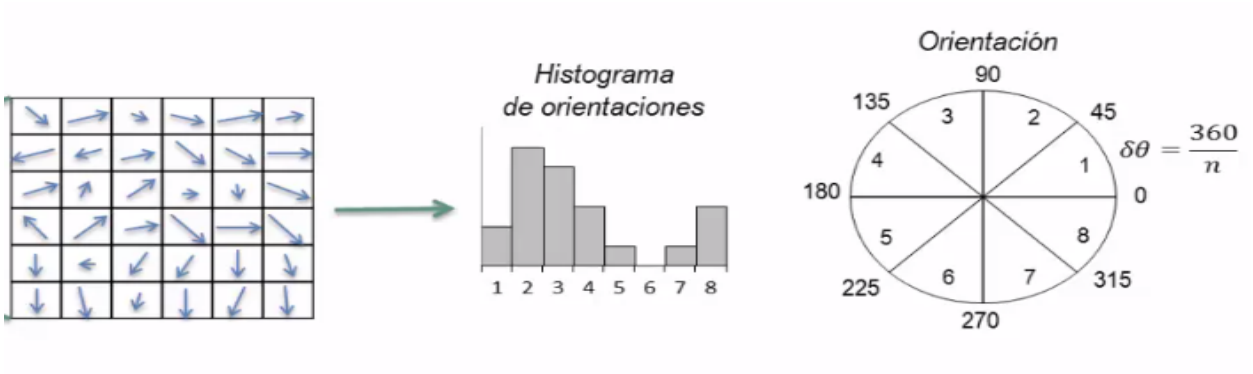


Figura 2.5.3: Construcción del histograma de orientaciones de los vectores gradiente.

El histograma tendrá tantos valores como subdivisiones se hayan hecho del rango de orientaciones. Dichos valores serán la suma de las magnitudes de todos los vectores gradientes que formen parte del mismo intervalo de orientaciones.

Además, se añade un paso adicional al cálculo del histograma. Este paso consiste en dar mayor importancia a los píxeles más próximos al centro de la región de interés, ya que son más estables, y menos importancia a aquellos que se sitúan en la frontera de la región, puesto que se considera que son más susceptibles al ruido.

Esto se consigue ponderando la aportación de cada píxel con la función Gaussiana ya vista en la ecuación (2.13).

El histograma de orientaciones nos da información acerca de la distribución de las orientaciones del gradiente en una determinada región. Por lo tanto, lo podemos utilizar para determinar la orientación dominante en la región considerada. Esta orientación se utiliza en el cálculo del descriptor para rotar la zona de interés de manera que conseguimos una descripción invariante ante rotaciones. El algoritmo SIFT propone un histograma de orientaciones con 36 intervalos donde la orientación dominante es aquella que corresponde al máximo global del histograma. Además, ante la posible existencia de ruido, se consideran como orientaciones alternativas aquellas que corresponden a máximos locales del histograma cuyo valor es superior al 80% del máximo global.

Una vez determinada la orientación dominante, la región de interés queda rotada de forma que queda alineada con dicha orientación.

Para calcular el descriptor procedemos a subdividir la región de interés en una cuadrícula de n x n celdas de igual tamaño. Generalmente se tiene que n = 4. Para cada una de estas celdillas se calcula nuevamente un histograma de orientaciones tal y como se ha explicado anteriormente. En este caso los histogramas de orientación son de 8 intervalos y se utilizan únicamente los píxeles que pertenecen a la celda.

El descriptor se construye almacenando todos los valores de los histogramas uno tras otro en un vector, de forma que tendríamos un descriptor con 16 histogramas, cada uno de esos histogramas tendría 8 posibles orientaciones lo que nos proporcionaría un descriptor de 128 componentes. En la figura 2.5.4 se observa la forma en la que se construye el descriptor SIFT.

Para concluir, el vector se normaliza para que sea invariante ante cambios en la iluminación.

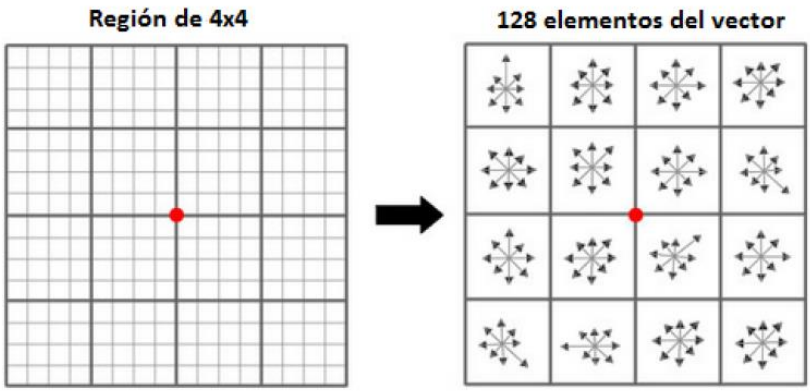


Figura 2.5.4: Construcción del descriptor SIFT

2.5.2 Descriptor SURF

Al igual que en el detector SIFT, se comienza por determinar tanto el nivel de la pirámide de imágenes como la escala en la que se obtuvo un determinado punto de interés. Esta tarea es realizada por el detector de características, cuyo funcionamiento ya se explicó en la sección 2.4.2.2.

El siguiente paso consiste en calcular una orientación dominante para el punto de interés. Para ello se emplean detectores de contornos. En este caso se opta por aplicar filtros de Haar a la región que rodea al punto de interés en lugar de calcular el valor exacto de los gradientes. Con esto se consigue que el coste computacional sea mucho menor. Los filtros de Haar son filtros binarios cuyos valores pueden ser -1 o 1 y que son capaces de obtener una respuesta parecida a la que se obtendría si se analizase el gradiente.

La región seleccionada para aplicar los filtros de Haar viene determinada por la escala en la que se detectó la característica visual, de forma que, si el punto de interés se encontró para una escala de valor S, el tamaño de la región a analizar debe ser circular de radio 6 x S según [35].

Las respuestas al filtro de Haar (horizontal y vertical), ponderadas con una Gaussiana alrededor del punto de interés, se representan como vectores en el espacio bidimensional. Los 360º del espacio bidimensional se dividen en ventanas angulares de 60º de forma que el espacio queda divido en 6 regiones. Para cada una de esas 6 regiones se suman todos los vectores que caigan dentro de dicha región. Esto proporciona un vector resultante para cada una de las ventanas angulares. El vector que presente un mayor tamaño determina la orientación del punto de interés, y dicha orientación vendrá dada por el ángulo que forme este vector con la horizontal.

En la figura 2.5.5 vemos un ejemplo del proceso descrito en el párrafo anterior.

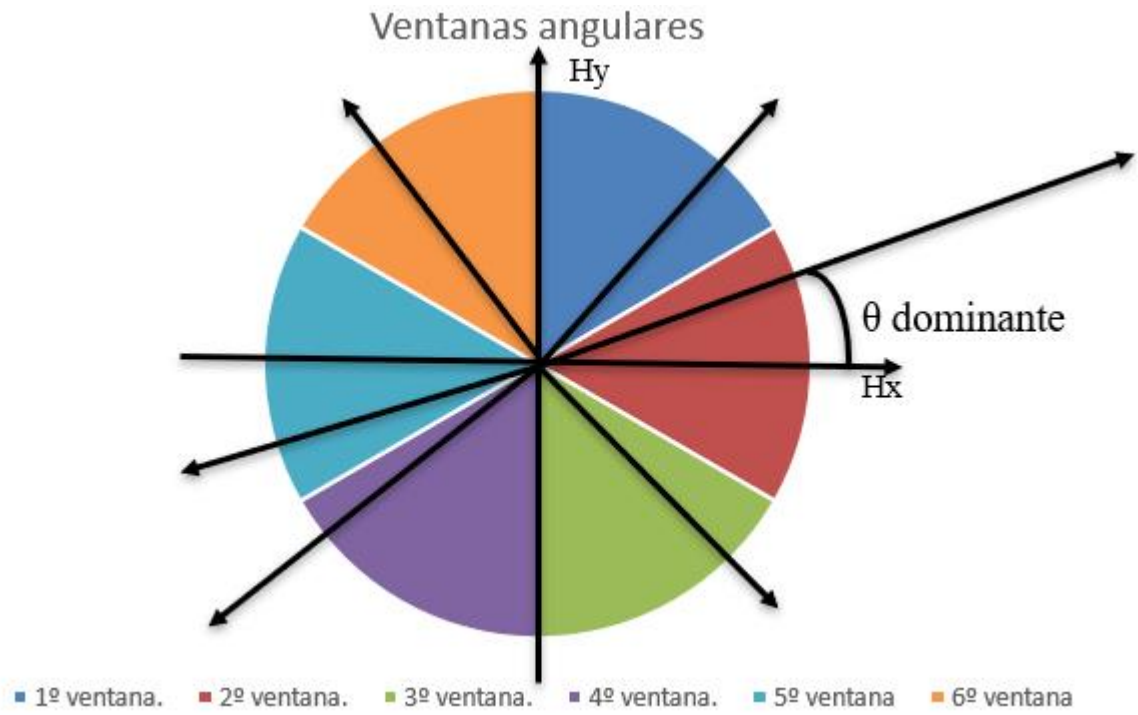


Figura 2.5.5: Obtención de la orientación dominante en el algoritmo SURF. Hx y Hy representan las respuestas a los filtros de Haar en las direcciones x e y. Se muestran los vectores resultantes para cada una de las ventanas angulares en las que se divide el espacio bidimensional. El vector de mayor módulo determina la orientación dominante θ.

Una vez tenemos la orientación dominante, el último paso consiste en calcular la descripción, en forma de vector compacto, de la región que rodea al punto de interés. Para ello seleccionamos un área rectangular centrada en el punto de interés que posicionamos según la orientación dominante obtenida en el paso anterior. El tamaño de esta región debe ser de 20 x S según [35], siendo S la escala mencionada anteriormente.

A continuación, se subdivide esta región en 16 subregiones de forma uniforme. En cada una de estas subregiones se consideran 25 puntos de muestreo uniformemente distribuidos y se seleccionan unas áreas circulares de radio 2 x S centradas en cada uno de esos 25 puntos. Posteriormente se calculan las respuestas que presentan cada una de esas regiones circulares ante los filtros de Haar y se suman dando lugar al siguiente vector de descripción de 4 componentes:

$$[\sum dx\,, \sum dy\,, \sum|dx|\,, \sum|dy|\ ] \tag{2.24}$$

Donde dx y dy son las respuestas horizontal y vertical que presentan las áreas circulares mencionadas anteriormente ante los filtros de Haar y todos los sumatorios son sobre puntos uniformemente espaciados que pertenecen a la misma subregión. Se concatenan los vectores descriptores de cada subregión para formar un vector único que identifique al punto de interés. Este vector será de 64 componentes (16 subregiones x 4 componentes cada región).

Finalmente, para poder comparar vectores descriptores de diferentes puntos de interés, se procede a normalizarlos de forma que la suma de todas sus componentes sea igual a la unidad.

2.5.3 Descriptor CenSurE

El algoritmo CenSurE no posee un descriptor propio, sino que modifica el descriptor SURF de forma que reduce los efectos de borde. Para ello, en [36], se propone un algoritmo denominado MU-SURF que consigue disminuir dichos efectos de borde solapando cuadros de las subregiones propuestas por SURF. Además, este procedimiento resultó ser más eficiente que el propio descriptor SURF, presentando un menor coste computacional en su ejecución.

2.5.4 Descriptores binarios

En robótica, siempre se persigue mejorar la eficiencia de los algoritmos con el fin de que tanto el tiempo de ejecución como el coste computacional disminuyan. Por tanto, surge la necesidad de encontrar descriptores que se calculen más rápido, que sean más compactos y que no se vean obligados a sacrificar su robustez ante cambios en la imagen, ya sean debidos a rotaciones, cambios de escala o ruido.

Como solución a esta necesidad aparecen los descriptores binarios. Estos descriptores presentan una serie de características que difieren en ciertos puntos con las propiedades de los descriptores ya vistos hasta ahora:

- Los descriptores binarios utilizan menos memoria para almacenar el vector que describe la característica visual. Esto se debe a que los descriptores binarios almacenan un cero o un uno en cada componente del vector, mientras que en los descriptores SIFT o SURF cada elemento del vector requiere, como mínimo, espacio suficiente para almacenar un punto flotante.
- En cuanto al tiempo de ejecución, las operaciones realizadas por los descriptores binarios son mucho más rápidas que las realizadas por los descriptores SIFT o SURF. Esto se debe a que solo se compara la intensidad entre varios pares de píxeles escogidos alrededor del punto de interés, con previa aplicación de un filtro Gaussiano para reducir el ruido. Por el contrario, en los descriptores SIFT y SURF se trabaja con un mayor número de píxeles, ya que se consideran regiones completas centradas en el punto de interés.
- Para calcular la distancia entre dos vectores obtenidos a través de los descriptores SIFT y SURF se utiliza la norma euclídea. Por el contrario, para obtener la distancia entre vectores generados por descriptores binarios, se utiliza la norma Hamming [56]. Esta última se calcula utilizando la operación lógica XOR que, desde el punto de vista computacional, es mucho más eficiente que el cálculo de la norma euclídea. Esto último se traduce en un proceso de emparejamiento de puntos de interés mucho más rápido.

2.5.4.1 Descriptor BRIEF

El descriptor BRIEF (*Binary Robust Independent Elementary Features* [38]) selecciona 512 pares de píxeles pertenecientes a una región situada alrededor del punto de interés. Las posiciones de estos píxeles se escogen de manera aleatoria utilizando una función Gaussiana alrededor del punto característico proporcionado por cualquiera de los detectores de características mencionados en el apartado 2.4. de este documento.

Para cada par de píxeles se procede a comparar la intensidad. Sean “P” y “Q” el primer par de píxeles. Si la intensidad de Q es mayor que la de P el resultado a almacenar en el descriptor es 1, en caso contrario se almacena un 0. Repitiendo esta comparación para el conjunto de los 512 pares de píxeles se consigue un vector descriptor de la característica visual de 512 componentes (64 si se trabajan con bytes en lugar de bits).

Este descriptor de características no considera cambios de escala ni de rotación, por lo que esta información debe ser proporcionada por el detector de características utilizado.

2.5.4.2 Descriptor ORB

El procedimiento ORB (*Oriented FAST and Rotated BRIEF* [39]) utiliza el detector FAST y el descriptor BRIEF, pero con ciertas modificaciones que mejoran su funcionamiento.

En primer lugar, utiliza el detector de características FAST para encontrar los puntos de interés y posteriormente utiliza la puntuación propuesta en el detector Harris como filtro para quedarse únicamente con las características visuales más robustas. Además, también utiliza una pirámide de imágenes para generar características visuales a múltiples escalas.

Posteriormente, considera una región centrada en el punto de interés. Para esta porción de la imagen calcula el centroide ponderado por la intensidad, de forma que la orientación del punto de interés vendría dada por la dirección del vector que conecta el propio punto de interés con el centroide calculado.

Para finalizar, el algoritmo ORB dirige el descriptor BRIEF según la orientación de los puntos característicos. Para ello utiliza la orientación de la característica visual para calcular una matriz de rotación que se utiliza para girar todos los puntos que vayan a ser utilizados por BRIEF para calcular el vector descriptor.

2.5.4.3 Descriptor BRISK

El descriptor BRISK (*Binary Robust Invariant Scalable Keypoints* [40] también es invariante ante movimientos de rotación y cambios de escala. El vector descriptor de la característica visual se construye utilizando un patrón circular de muestreo sobre la región que rodea al punto de interés. Podemos apreciar dicho patrón en la figura 2.5.6.

El patrón selecciona un número determinado de puntos igualmente espaciados sobre círculos concéntricos. Los pares de puntos obtenidos se dividen en función de la distancia que los separa. Los que superan un determinado umbral son considerados pares largos y se utilizan para estimar la orientación del punto de interés. Aquellos que no superan el umbral son considerados pares cortos y se utilizan para construir el descriptor una vez rotado el patrón según la orientación obtenida. La operación que se realiza para generar el vector descriptor consiste nuevamente en comparar la intensidad de cada par de píxeles.

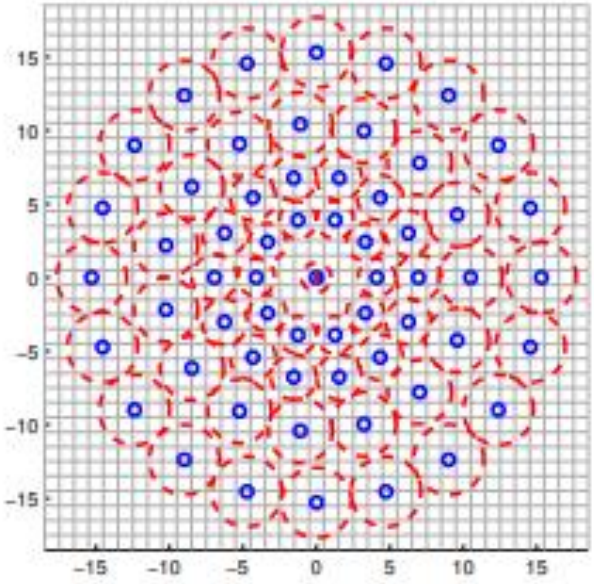


Figura 2.5.6: Patrón de muestreo utilizado por el descriptor BRISK para caracterizar un punto de interés

2.5.4.4 Descriptor FREAK

El descriptor FREAK (*Fast Retina Keypoint* [41]) se basa en el descriptor BRISK, pero además se inspira en el sistema de visión humana. El patrón utilizado para seleccionar los pares de píxeles a utilizar en la construcción del vector descriptor es similar al utilizado en BRISK, pero se otorga mayor importancia a los píxeles concentrados en el área central de la región de interés. De esta forma se obtiene una imitación de los campos receptivos de la retina humana, que presentan mayor concentración de receptores visuales en la zona central del ojo. Conforme nos alejamos del centro, la cantidad de píxeles considerados disminuye exponencialmente, al igual que sucede con los receptores visuales del sistema de visión humano. En la figura 2.5.7 podemos apreciar el patrón mencionado.

Un filtro Gaussiano con diferentes desviaciones estándar por cada nivel es aplicado para reducir el ruido.

El descriptor se forma calculando las diferencias de intensidad entre pares de puntos que han sido sometidos al mismo filtro. Si la operación resulta positiva, a la componente en cuestión del vector descriptor se le asigna un 1, en caso contrario se le asigna un 0.

Los resultados experimentales extraídos de [41] arrojan la conclusión de que con 512 pares de puntos se obtiene la mayor cantidad de información posible del punto de interés. Estos pares de puntos están formados por aquellas parejas de píxeles que presentan mayor varianza.

Al igual que en BRISK, la orientación se obtiene realizando cálculos con aquellos pares de píxeles que son considerados pares largos, al presentar distancias largas sobre la configuración simétrica.

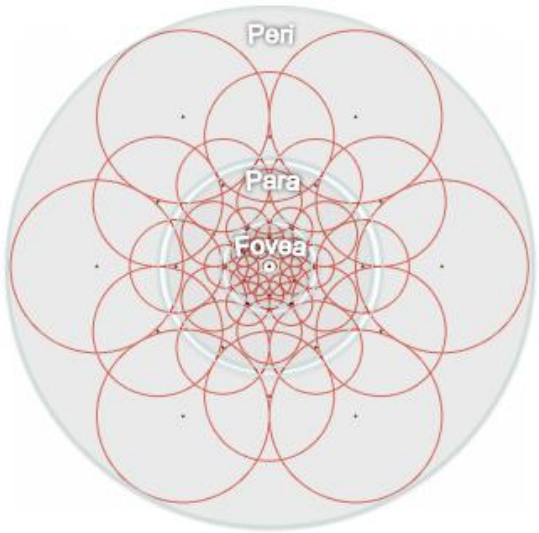


Figura 2.5.7: Patrón utilizado por el descriptor FREAK encargado de simular el funcionamiento de la retina humana.

2.6 Emparejamiento de características visuales

El emparejamiento de características visuales es un proceso que consiste en localizar un determinado punto de interés en distintas imágenes. De esta forma podríamos realizar un seguimiento de la característica visual a lo largo de toda una secuencia de fotogramas.

Hay diversas formas de plantear esta tarea. La primera de ellas consiste en considerar el emparejamiento de puntos de interés como un problema de coincidencia (*matching* en inglés*)* entre vectores descriptores de características visuales correspondientes a distintas imágenes. Para realizar esta comparación se suele recurrir a técnicas de *Machine learning* como *k-nearest neighbors* [42].

El procedimiento más común de *k-nearest neighbors* consiste en utilizar un emparejador de Fuerza Bruta que compara todos los vectores descriptores de características visuales de dos imágenes diferentes. Para cada vector descriptor de un punto de interés de la primera imagen, el emparejador busca, entre todos los vectores descriptores de características visuales de la segunda imagen, el que mejor se ajusta en términos de mayor similitud o menor distancia.

Para comparar cada par de vectores descriptores en términos de similitud se procede a emplear criterios como la SAD, la SSD o la NCC (ya mencionados en el apartado 2.5 de este documento). Si por el contrario quisiéramos obtener la distancia entre los vectores descriptores, procederíamos a calcular la norma vectorial. Esta norma sería la distancia euclidiana si se tratase de vectores de tipo flotante como los generados por descriptores como SIFT o SURF, o la distancia Hamming [56] si trabajásemos con vectores binarios como los que se obtienen de los descriptores FREAK o BRISK. Puede darse el caso de que un vector descriptor de una característica visual de la primera imagen quede emparejado con más de un vector descriptor de la segunda imagen. En este contexto, se puede realizar un análisis de consistencia mutua o reciprocidad en el que un emparejamiento se considera válido sí y solo sí ambos vectores descriptores son los vectores descriptores más parecidos en la imagen el uno para el otro, y viceversa.

A pesar de ofrecer buenos resultados, el *matching* supone un alto coste computacional que dificulta que pueda ser utilizado en sistemas que funcionen bajo restricciones de tiempo real. No obstante, se han propuesto una serie de métodos que han sacrificado un poco la precisión de *k-nearest neighbors* en favor de poder reducir el coste computacional. Estos algoritmos, conocidos como *approximate-nearest neighbors*, forman parte de una librería de software denominada FLANN (*Fast Library for Approximate Nearest Neighbors* [43]).

Como alternativa al *matching* tenemos otra vertiente de actuación denominada seguimiento (*tracking* en inglés). Esta alternativa es viable para imágenes que se han tomado desde posiciones relativamente cercanas. El procedimiento consiste en detectar una característica visual en la primera imagen y limitar su búsqueda a una determinada región de las siguientes imágenes en las que se predice que pueda aparecer. De alguna forma se calcula un modelo matemático que predice el movimiento de la cámara a lo largo de una secuencia de imágenes y de esta manera obtenemos una región aproximada en la que se puede encontrar un determinado punto de interés en cada uno de los fotogramas que componen la secuencia. Este modelo matemático se suele construir con la ayuda de información procedente de otros sensores como un GPS, un sensor láser, etc.

Si el seguimiento se realizase a lo largo de una secuencia de imágenes demasiado larga, puede suceder que las características visuales del último fotograma difieran en gran medida de los puntos de interés de la primera imagen. En este caso se recurre al conocido *KanadeLucasTomasi tracker* [44]. Este algoritmo ha sido mejorado con el tiempo tal y como puede apreciarse en el trabajo de Bouguet [45].

Los métodos de emparejamiento de características visuales descritos en este apartado no son perfectos. De esto último concluimos que podrían existir malas asociaciones que den lugar a errores catastróficos a la hora de intentar reconstruir la trayectoria recorrida por la cámara. Estos errores de emparejamiento pueden deberse a diversas causas tales como oclusiones, ruido o cambios de perspectiva o iluminación en la imagen.

Por tanto, para garantizar un correcto funcionamiento de nuestro sistema de odometría visual, tenemos que buscar alguna forma de eliminar los malos emparejamientos de características visuales. Una solución empleada globalmente para la eliminación de estos emparejamientos anómalos consiste en tener en cuenta el modelo de movimiento de la cámara y las restricciones geométricas que este impone. El algoritmo más utilizado para esta tarea es RANSAC (RANdom SAmple Consensus [12]).

RANSAC no es más que un algoritmo encargado de determinar de forma robusta ciertos parámetros de un modelo matemático en presencia de información errónea. La idea es obtener un modelo como hipótesis a partir de información escogida de forma aleatoria y posteriormente verificar dicha hipótesis con la información restante. La hipótesis que consiga un mayor consenso con el resto de los datos tras un determinado número de iteraciones del algoritmo será seleccionada como solución. Evidentemente, a mayor número de iteraciones más preciso será el algoritmo.

RANSAC es un método probabilístico y no determinista. Esto último quiere decir que la solución proporcionada por el algoritmo es diferente para cada una de las ejecuciones de este. No obstante, cuanto mayor es el número de iteraciones, mayor es la estabilidad que se obtiene en la solución final.

El algoritmo no necesita comprobar todas las combinaciones, le basta con tan solo un subconjunto de ellas.

Con el emparejamiento de características visuales realizado antes de aplicar el algoritmo RANSAC, puede obtenerse una idea inicial sobre los valores anómalos presentes en la asociación de características visuales. Con esta información y conociendo las restricciones geométricas entre las vistas (la geometría epipolar y el error de reproyección de puntos tridimensionales en el plano de la imagen) podemos estimar las iteraciones que debe realizar el algoritmo RANSAC para converger a una solución estable con la siguiente fórmula:

$$N = \frac{\log(1-p)}{\log(1-(1-\varepsilon)^s)} \tag{2.25}$$

Donde N es el número de iteraciones, p es la probabilidad de éxito que se le exige al algoritmo, Ɛ es el porcentaje de valores anómalos esperado y S es el número mínimo de puntos necesarios. Este último parámetro viene determinado por las restricciones geométricas del modelo. Por ejemplo, si la cámara con la que se captura la secuencia de imágenes está calibrada, se puede calcular un modelo de movimiento con seis grados de libertad, lo que exigiría un mínimo de cinco puntos característicos por cada imagen para poder aplicar RANSAC [23].

Algoritmo 2.1: RANSAC

1. Inicialización: X es un conjunto de N correspondencias de puntos
2. Repetir mientras no se alcance el máximo número de iteraciones:
   - 2.1 Seleccionar aleatoriamente un subconjunto de S puntos de X
   - 2.2 Construir un modelo utilizando la información de S
   - 2.3 Evaluar el subconjunto de puntos restante con el modelo y medir el error
   - 2.4 Construir el conjunto de puntos válidos (aquellos cuya distancia al modelo sea inferior a un umbral)
   - 2.5 Almacenar puntos válidos
3. Seleccionar el conjunto con el mayor número de puntos válidos
4. Estimación del modelo empleando los puntos válidos

2.7 Obtención del movimiento

Supongamos que disponemos de una secuencia de imágenes que se han obtenido a lo largo del movimiento de un robot que portaba una cámara. Por cada par de imágenes consecutivas de dicha secuencia se calcula el movimiento realizado por la cámara entre dos instantes de tiempo consecutivos a saber k y k-1. Concatenando todos los movimientos incrementales obtenidos se podría reconstruir la trayectoria realizada por la cámara y, por consiguiente, la del robot en el que se encuentra.

Para calcular la transformación existente entre dos imágenes consecutivas, a las que denominaremos Ik e Ik-1, se utiliza como punto de partida la información recogida en dos conjuntos de características visuales, a los que denominaremos fk y fk-1.

Estos conjuntos de características reúnen información sobre ciertas regiones o puntos característicos de la imagen y pueden estar definidos en dos o tres dimensiones. Para el caso de odometría monocular podemos optar por alguno de los siguientes métodos para calcular la transformación existente entre dos imágenes consecutivas:

- Métodos 2-D a 2-D: los conjuntos de características fk y fk-1 están expresados en 2D.
- Métodos 3-D a 2-D: el conjunto fk-1 está definido en tres dimensiones mientras que el conjunto fk identifica las proyecciones de los puntos característicos en el plano de la imagen.

Una vez obtenida la transformación relativa existente entre las dos vistas, se procedería a calcular la profundidad que tienen los puntos tridimensionales correspondientes a cada par de correspondencias. Para ello se utiliza el método de triangulación que se describe en el siguiente apartado.

2.7.1 Triangulación

La triangulación es un proceso a través del cual se pretende obtener la posición de un punto tridimensional en el espacio. Como información de partida se dispone de las coordenadas 2D de un determinado punto de interés en dos imágenes consecutivas. Estas imágenes se obtienen con cámaras calibradas cuya posición en el espacio es conocida.

La idea consiste en utilizar un par de características visuales que han sido emparejadas para obtener las coordenadas de un punto 3D. cada una de estas características visuales se corresponde con una línea en el mundo 3D. Entonces, para conocer la profundidad del punto 3D al que representan, se requiere que las líneas correspondientes a cada una de estas características visuales intersequen en el espacio. En el caso de que no exista ruido, el problema es muy sencillo y se resume a unos simples cálculos geométricos. Sin embargo, en presencia de ruido (ya sea debido a la calibración de la cámara o al error que se comete en la localización de un punto de interés en la imagen) el problema se complica bastante ya que las líneas nunca llegarían a intersecar.

Como primera aproximación a la resolución de este problema, se considera que el punto de intersección entre ambas líneas es el punto medio de la recta más corta que une ambas líneas. Este método es el que se ilustra en la Figura 2.7.1.

Se ha demostrado que este método no proporciona soluciones óptimas y se han propuesto algunos algoritmos más robustos para estimar la profundidad de un punto tridimensional en la escena. Algunos de ellos aparecen mencionados en el artículo de Hartley y Sturm [48].

De entre todos los métodos descritos en dicho artículo merece la pena destacar el método polinomial. Este método propone un algoritmo no iterativo que pretende disminuir el error cometido al establecer correspondencias entre puntos de interés de imágenes que cumplan la restricción epipolar (ecuación 2.27). Sean $p\ y\ p'$ las correspondencias antes de aplicar el algoritmo. Se procede a calcular unas nuevas correspondencias $\hat{p}\ y\ \hat{p}'$ que sean capaces de disminuir el valor de la siguiente función de coste:

$$f(p,p') = d(p,\hat{p})^2 + d(p',\hat{p}')^2 \tag{2.31}$$

Donde d(x, y) es la distancia euclidiana entre los puntos. Una vez obtenidas las nuevas correspondencias entre los puntos de interés de las imágenes, se procede a estimar la profundidad del punto tridimensional mediante

triangulación lineal. Este método se describe en [65].

Se ha demostrado que cuando el ruido presenta forma Gaussiana el algoritmo de triangulación polinomial ofrece resultados óptimos en lo que a reconstrucción proyectiva se refiere [48]. Sin embargo, presenta el gran inconveniente de tener un gran tiempo de ejecución, lo que lo haría indeseable para sistemas que trabajen bajo restricciones de tiempo real.

Como alternativa al método polinomial se propone el método *Iterative-LS*, cuyos tiempos de ejecución son menores. El problema de este algoritmo es que no converge para algunos de los puntos, lo que ocasionaría un fallo catastrófico a la hora de realizar la reconstrucción proyectiva.

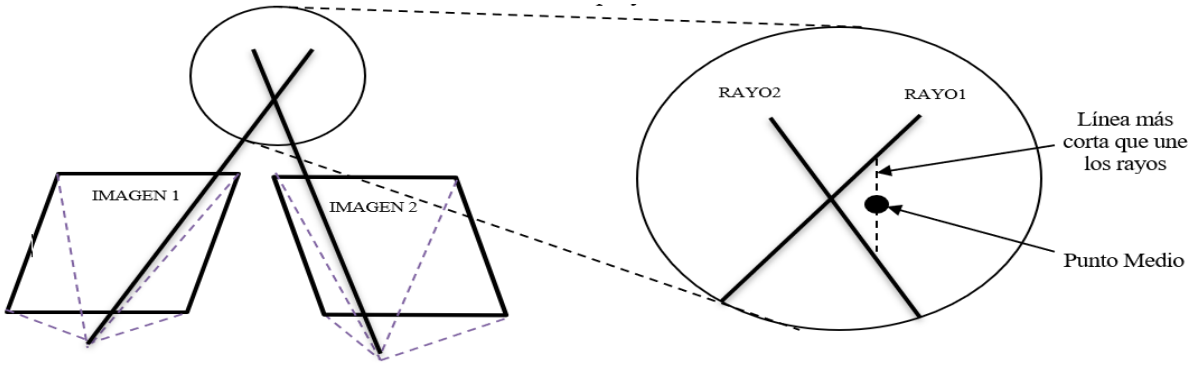


Figura 2.7.1: Triangulación a través del método del punto medio.

2.7.2 Estimación del movimiento con correspondencias 2D a 2D

Para describir la relación existente entre dos imágenes que observan la misma escena nos servimos de la geometría epipolar [46]. La geometría epipolar depende únicamente de los parámetros intrínsecos de la cámara y de la posición relativa entre dos vistas de la escena. Siguiendo la nomenclatura de la figura 2.7.2 se procede a describir el funcionamiento de dicha geometría.

Supongamos un punto tridimensional P_i que es proyectado sobre cada uno de los planos de ambas cámaras. La relación existente entre dicho punto, sus respectivas proyecciones y los centros de las cámaras (C_{k-1} y C_k) da origen al denominado plano epipolar. Más concretamente, la nomenclatura a utilizar es la siguiente:

- Se define como epipolo (e_k y e_{k-1}) el punto de intersección entre la línea que une los centros de las cámaras (línea base) y el plano de la imagen.
- El plano epipolar es aquel formado por la línea base y por el haz resultante de proyectar el punto tridimensional en ambas vistas.
- Entendemos por línea epipolar aquella que resulta de la intersección entre un plano epipolar y el plano de la imagen. Todas las líneas epipolares desembocan en el epipolo. En la imagen sería aquella línea que conecta los puntos l y e.

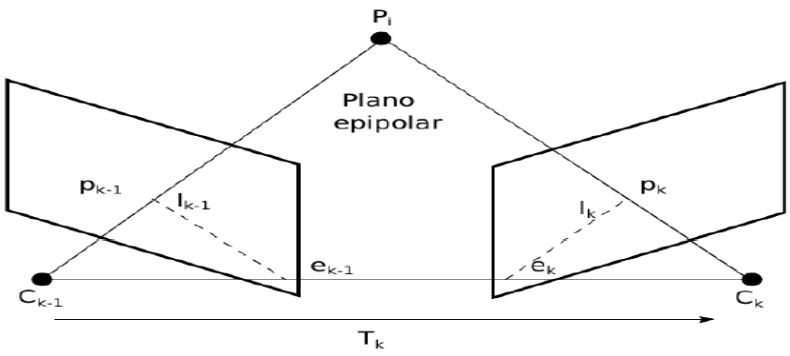


Figura 2.7.2: Esquema que describe el funcionamiento de la geometría epipolar.

En términos matemáticos disponemos de una matriz de dimensiones 3x3 que nos permite relacionar un punto de la imagen con su correspondiente línea epipolar. Esta matriz se denomina matriz Fundamental y su misión consiste en mapear las coordenadas de la línea epipolar a las coordenadas del punto de la imagen:

$$\begin{bmatrix} l_x \\ l_y \\ l_z \end{bmatrix} = F \begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix} \tag{2.26}$$

Donde

- F es la matriz fundamental.
- P_x y P_y son las coordenadas de la proyección del punto.
- lx, ly e lz son las coordenadas de la línea epipolar.

Si se conoce un punto X de la escena cuyas proyecciones en ambas imágenes son x y x' respectivamente, se tiene que la matriz Fundamental debe satisfacer la siguiente condición:

$$xFx' = 0 \tag{2.27}$$

Por consiguiente, la matriz Fundamental puede calcularse si son conocidas varias correspondencias de puntos entre ambas imágenes. Uno de los algoritmos más utilizados es el de los ocho puntos [63], en el que, como su propio nombre indica, se necesitan ocho correspondencias entre puntos de ambas imágenes. Una vez conocidas dichas correspondencias, se calcula la matriz Fundamental a través de un sistema de ecuaciones lineales.

Llegados a este punto, sabemos que la matriz Fundamental relaciona algebraicamente las proyecciones de un mismo punto tridimensional en dos vistas diferentes de la misma escena. Si además queremos conocer la traslación unitaria y la rotación relativa entre ambas imágenes debemos calcular la matriz Esencial.

Esta última matriz es una especialización de la matriz Fundamental en la que las coordenadas de la imagen se encuentran normalizadas. Mientras que la matriz fundamental tiene siete grados de libertad, la matriz Esencial tan solo posee cinco. Tres grados de libertad están relacionados con la rotación y otros tres con la traslación. Al existir una incertidumbre en la escala, la magnitud de la traslación almacenada en la matriz puede contener diferentes escalas, por lo que se elimina un grado de libertad.

Conocida la matriz Fundamental, la matriz Esencial puede calcularse a través de la siguiente expresión:

$$E = K^T F K \tag{2.28}$$

Donde:

- E es la matriz Esencial.
- F es la matriz Fundamental.
- K es la matriz de parámetros intrínsecos de la cámara.

Aunque en este caso también se podría usar el método de los ocho puntos, sale más rentable usar el método de los cinco puntos propuesto por Níster, que es más eficiente y menos susceptible al ruido [23].

Podemos expresar la matriz Esencial en términos de la rotación y la traslación y obtendríamos lo siguiente:

$$E \approx t_k^* R_k \tag{2.29}$$

Donde:

- $t_k = [t_x\ t_y\ t_z]^T$ es el vector de traslación.
- R_k es la matriz de rotación
- $t_k^* = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}$

Una vez obtenida la matriz Esencial se procede a extraer el vector de traslación y la matriz de rotación a través de una descomposición en valores singulares [64]. Esta operación puede devolver hasta cuatro posibles soluciones para la rotación y la traslación. Para escoger la solución adecuada se recurre a utilizar un punto obtenido mediante triangulación. La solución correcta para el movimiento relativo será aquella en la que el punto obtenido mediante triangulación se encuentre frente a las dos vistas consideradas.

A pesar de que la matriz Esencial no proporciona información referente a la escala absoluta, se puede estimar dicha escala utilizando las escalas relativas entre vistas consecutivas [37].

2.7.3 Estimación del movimiento con correspondencias 3D a 2D.

Si conocemos la geometría tridimensional del entorno es posible calcular la transformación de cuerpo rígido [R|t] (donde R es la matriz de rotación y t el vector de traslación) entre un par de imágenes de la escena. A este problema se le denomina Perspectiva desde n Puntos (PnP) [47].

El problema PnP consiste en obtener la posición y orientación de una cámara dados sus parámetros intrínsecos, y un conjunto de n correspondencias entre puntos tridimensionales y sus respectivas proyecciones en el plano de la imagen.

Para solventar este problema tenemos dos formas diferentes de proceder. La primera opción consistiría en utilizar métodos iterativos. En estos métodos se pretende minimizar una determinada función que suele estar relacionada con el error de reproyección de los puntos tridimensionales sobre la imagen. El problema consistiría entonces en encontrar una transformación de cuerpo rígido, a la que denominamos T_k, que sea capaz de minimizar el siguiente criterio:

$$\min T_k \sum_i \left\| p_k^i - p^{*i}_{k-1} \right\|^2 \tag{2.30}$$

Donde p^{*i}_{k-1} son las reproyecciones de los puntos tridimensionales X^i_{k-1} en la imagen I_k y p_k^i son los puntos bidimensionales de la imagen I_k que se corresponden con los puntos tridimensionales mencionados.

Como ventaja, puede decirse que estos métodos son muy precisos si convergen adecuadamente. En contraprestación, estos algoritmos son muy susceptibles al ruido además de ser computacionalmente complejos.

En la otra mano, tenemos los métodos de ejecución única. Estos algoritmos son menos complejos que los iterativos, lo que los hace más rápidos. Sin embargo, hay que mencionar que son muchísimo menos precisos.

2.7.4 Selección de fotogramas clave o *Keyframes*

El proceso de obtención de la posición de la cámara depende en gran medida de la robustez del algoritmo encargado de emparejar puntos característicos de diferentes imágenes. Si además utilizamos correspondencias 3D a 2D, la estimación de la posición de la cámara dependerá de la precisión de la reconstrucción tridimensional de la escena.

Si para realizar los cálculos pertinentes se seleccionan dos fotogramas contiguos, la información que se obtiene es prácticamente la misma. Esto último se debe a que el intervalo de tiempo transcurrido es muy corto y, por consiguiente, el movimiento realizado por la cámara es muy pequeño. Es por ello por lo que se necesita encontrar algún criterio que nos permita detectar aquellos fotogramas que aporten la mayor cantidad de información posible sobre los puntos de interés o sobre los cambios en el movimiento de la cámara. A estos fotogramas se les denomina *Keyframes*. En la literatura se han propuesto algunos métodos heurísticos para detectar dichos fotogramas:

- Debe dejarse un determinado intervalo de tiempo entre *Keyframes*.
- El error de reproyección de los puntos tridimensionales sobre el fotograma candidato a ser considerado *Keyframe* debe ser inferior a un determinado umbral.
- La posición de la cámara debe encontrarse alejada de la posición en la que se obtuvo el último *Keyframe* para que el fotograma actual pueda ser considerado *Keyframe*.
- Se requiere que haya una buena fracción de puntos de interés emparejados entre fotogramas.

En el apartado 3.2 de este documento se propone un método más sofisticado para la obtención de fotogramas clave. Este módulo de detección de *keyframes* es el que se ha utilizado en el algoritmo encargado de resolver el problema de odometría visual que se propone en este proyecto.

2.8 Optimización de la trayectoria obtenida

La odometría visual calcula la trayectoria descrita por la cámara de forma incremental. Cada una de las nuevas posiciones de la cámara se obtiene tras aplicar una transformación de cuerpo rígido a la última posición conocida. Dado que los métodos empleados para estimar la posición de la cámara no son perfectos, cada una de las posiciones de la cámara lleva asociada una determinada incertidumbre. Si basamos el cálculo de una nueva posición en otra posición que ya posee incertidumbre, la incertidumbre obtenida para la nueva posición es aún mayor. Es por este motivo que se genera un error acumulativo que hace que la trayectoria estimada presente una mayor deriva con respecto a la trayectoria original conforme avanza el tiempo.

Por consiguiente, para que la trayectoria estimada se parezca lo máximo posible a la trayectoria original, tenemos que buscar algún método que nos permita disminuir dicho error acumulativo al mínimo. Para acometer esta tarea se suelen emplear métodos de optimización local sobre un número N de posiciones estimadas. Algunas de las técnicas más conocidas son el local bundle adjustment [49] y la optimización de grafos.

2.8.1 Local Bundle Adjustment

Definimos como *Bundle Adjustment* al problema de refinar simultáneamente tanto la estructura tridimensional del entorno como la posición de la cámara y los parámetros intrínsecos de la misma. Este método trata de minimizar una determinada función de costo relacionada con el error de reproyección. Más detalladamente el problema se formularía como sigue:

Supongamos un conjunto X_j de N puntos tridimensionales de la escena que son visibles desde un conjunto de M cámaras. Sea $x_{i,j}$ la proyección del punto j en la cámara i. Sea $v_{i,j}$ una variable booleana que vale 1 si en la cámara i es visible el punto j y cero en caso contrario. Supongamos que cada cámara está parametrizada por un vector C_i y cada punto tridimensional por un vector P_j. La idea consiste en minimizar el error de reproyección total con respecto a todos los puntos tridimensionales de la escena y los parámetros de la cámara utilizando la siguiente función de costo:

$$min_{C_i P_j} \sum_{j=1}^{j=N} \sum_{i=1}^{i=M} v_{i,j}\, d(\, Q(C_i, P_j)\, , x_{i,j}\,)^2 \tag{2.32}$$

Donde $Q(C_i, P_j)$ es la predicción de la proyección del punto j en la imagen i. d(x,y) sería la distancia Euclidiana entre dos puntos de la imagen representados por los vectores x e y. Por tanto, la función de costo sería la suma de los errores de reproyección al cuadrado.

Estamos pues ante un problema de minimización no lineal que se resuelve usando algoritmos de mínimos cuadrados tales como Levenberg-Marquardt [50]. En este caso se resuelven ecuaciones que presentan la siguiente forma:

$$(J^T J + \mu I)\left(\frac{1}{\mu}\right)(J^T f) = -(J^T f) \tag{2.33}$$

Donde J es el Jacobiano de la función de reproyección f, la cual usa como parámetros las matrices de reproyección, a las que denominaremos RP_i, y los puntos tridimensionales del entorno X_j para generar las coordenadas bidimensionales $x_{i,j}$ de cada punto en la imagen.

La matriz Jacobiana se construiría con las derivadas parciales $\partial x_{i,j} / \partial RP_k$ y $\partial x_{i,j} / \partial X_k$ de forma que solo obtendríamos valores no nulos en aquellas derivadas en las que k = i cuando se deriva con respecto a las matrices de reproyección y en las que k = j cuando se deriva con respecto a los puntos tridimensionales. Esto último se debe a que las coordenadas proyectadas de un punto tridimensional j en la imagen i dependen exclusivamente de la matriz de proyección de la imagen i y del propio punto j. Por consiguiente, se obtiene una matriz jacobiana dispersa en la que solo algunos valores difieren de cero, lo que permite utilizar técnicas como la descomposición de Cholesky [51].

A medida que aumentan el número de vistas y el número de puntos 3D que conforman la estructura del entorno, aumenta el coste computacional requerido para ejecutar el *Bundle Adjustment*. Es por ello por lo que este método no es deseable para aplicaciones que trabajen bajo restricciones de tiempo real. No obstante, se podría aplicar el algoritmo únicamente sobre una ventana temporal de imágenes que contengan la información más reciente, de forma que se refinarían los parámetros de un subconjunto de cámaras en lugar de optimizar los parámetros de todas las cámaras. Esto es lo que se conoce como *Local Bundle Adjustment* que aparece ilustrado en la figura 2.8.1.

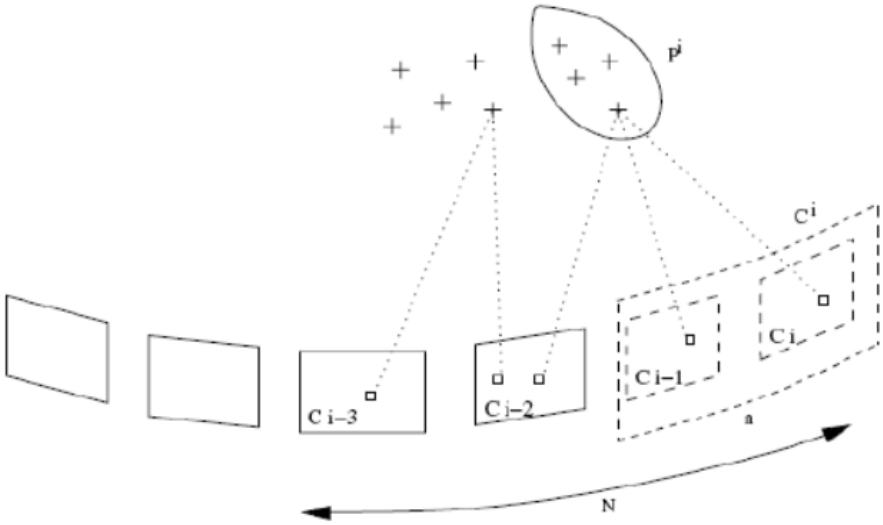


Figura 2.8.1: Ilustración del método *Local Bundle Adjustment*. La variable N indica el número de imágenes que componen la ventana temporal. En este caso serían cuatro.

3 ARQUITECTURA Y DESARROLLO DEL SISTEMA

En este apartado se describe un sistema de odometría visual cuyo propósito es calcular la posición de un robot en el espacio. Para ello se propone un algoritmo que, a través de las imágenes capturadas por una cámara acoplada a un robot móvil, trata de reconstruir la trayectoria recorrida por dicho robot. Además, se supone que las características de la cámara son conocidas. Por este motivo, el algoritmo no tendrá que ocuparse del proceso de calibración de la cámara.

Para la confección del algoritmo se ha utilizado la biblioteca de software OpenCV[52]. OpenCV es una plataforma muy popular en lo que a visión por ordenador se refiere. Dispone de una gran cantidad de estructuras de datos para el manejo eficiente de imágenes y, además, cuenta con infinidad de algoritmos capaces de realizar las tareas de procesamiento de imágenes, detección de características visuales y descripción de puntos de interés. Esta biblioteca es multiplataforma y está programada tanto en código C/C++ como en Python.

El módulo de inicialización de este sistema requiere de un optimizador. Es por ello por lo que recurrimos a un paquete externo denominado g2o[53]. g2o es una librería de código abierto programada en C++ pensada para optimizar funciones de error no lineales que pueden ser representadas como grafos [54]. La librería g2o ha sido diseñada para ser fácilmente extensible a una amplia gama de problemas tales como el *Bundle Adjustment* y algunas variantes de *SLAM*.

Para visualizar los resultados arrojados por el algoritmo se han utilizado algunos ficheros programados en Python y la librería programada en C++ denominada PCL (*Point Cloud Library*) [55].

En la figura 3.1 se muestra la secuencia que sigue el algoritmo de odometría visual y los distintos bloques que lo componen. En la figura 3.2 se muestra la subrutina correspondiente al bloque encargado de crear un mapa inicial del entorno.

Con este apartado se pretende que el lector de este documento sea capaz de seguir la línea de ejecución del algoritmo y ver cómo funciona sin entrar en detalles sobre la programación que hay detrás de este.

No obstante, el código de programación utilizado para implementar este algoritmo se encuentra en el siguiente repositorio: https://github.com/lema18/map-triangulation. De esta forma el lector podrá consultarlo siempre que lo considere oportuno.

En los próximos subapartados se procede a describir el funcionamiento de todos los bloques que componen el algoritmo de odometría visual propuesto en la figura 3.1.

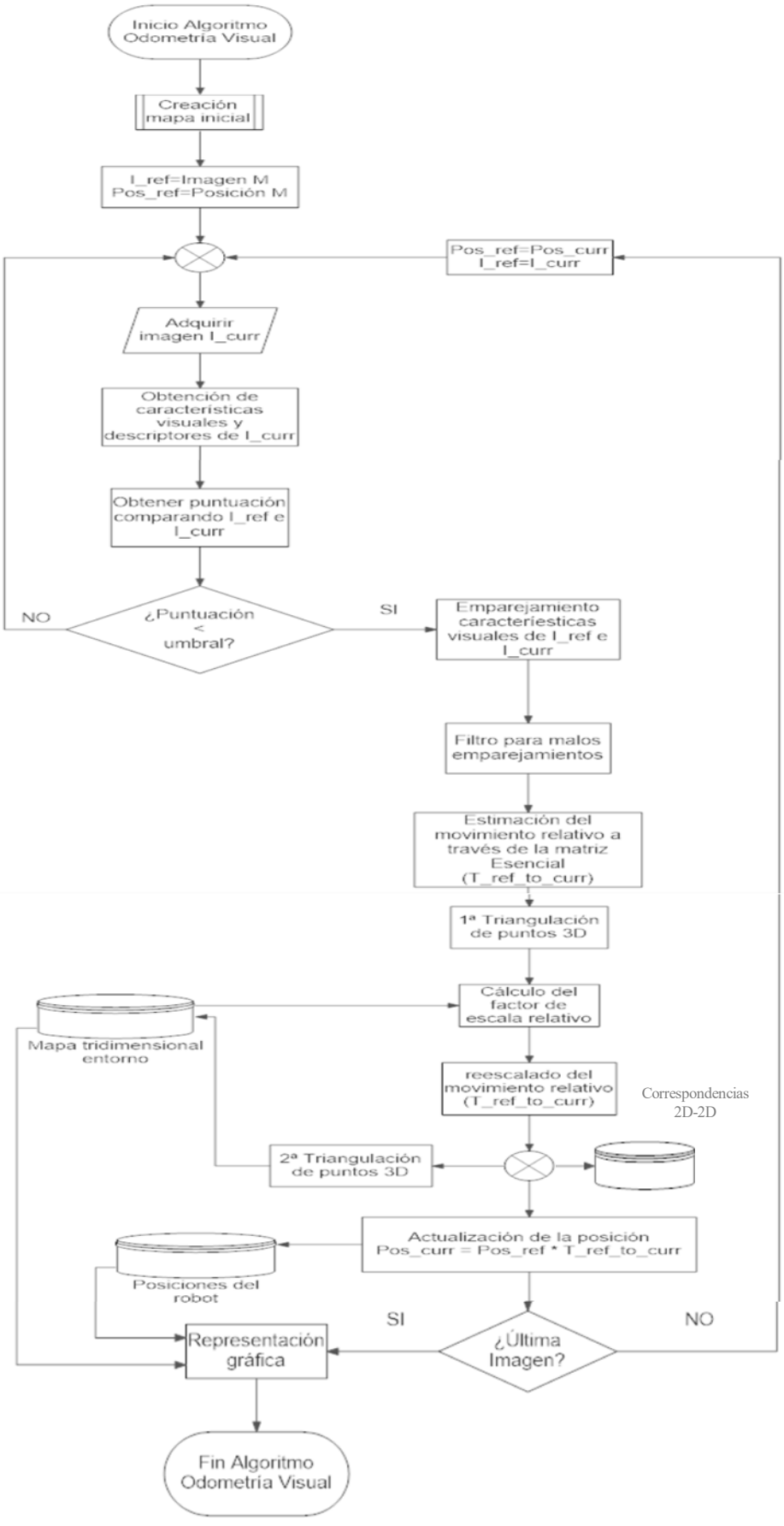


Figura 3.1: Algoritmo de odometría visual propuesto

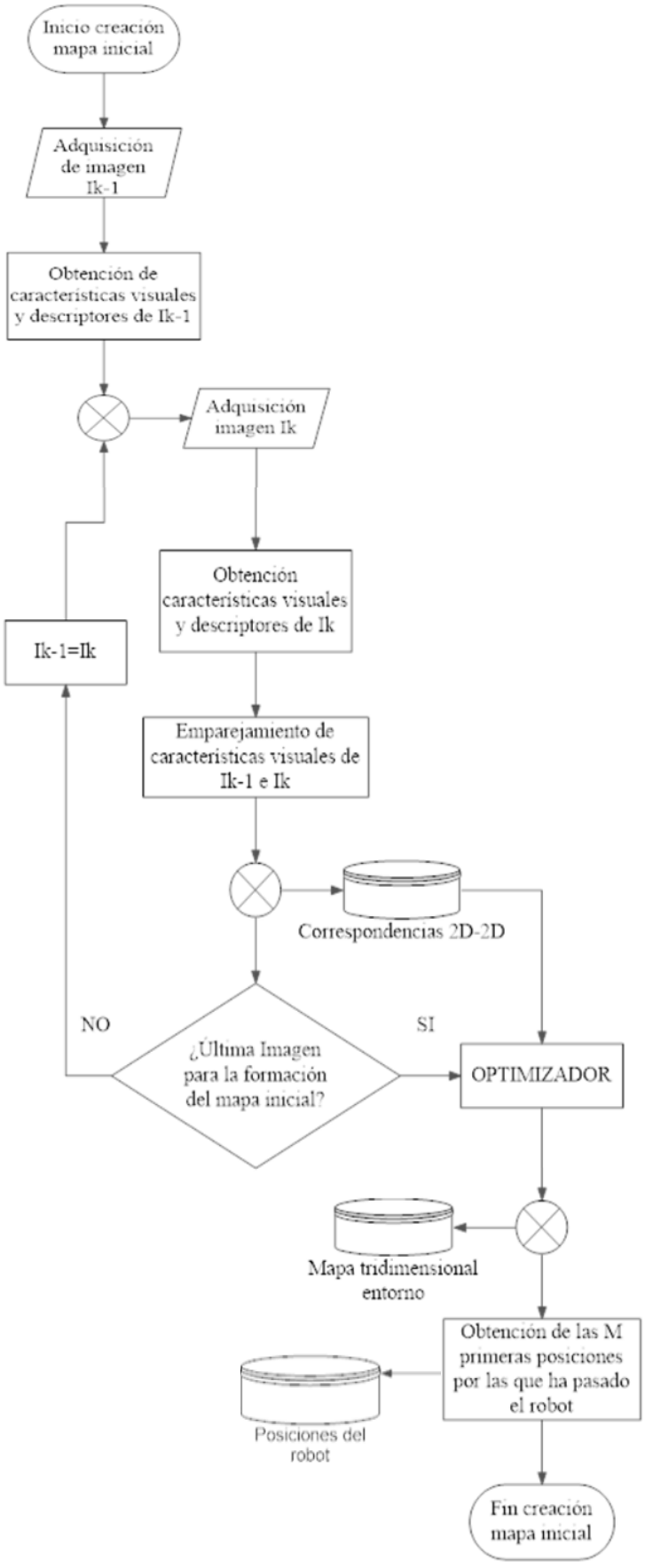


Figura 3.2: Rutina correspondiente al bloque “Creación mapa inicial” del algoritmo de odometría visual.

3.1 Creación del mapa inicial

Se asume que la posición de partida del robot será nuestro origen de coordenadas para el sistema de odometría visual. Nuestra única herramienta es una cámara calibrada que captura 30 imágenes por segundo.

En la posición de partida se toma la primera imagen y a continuación, el robot inicia su movimiento. Cada vez que una nueva imagen es capturada por la cámara, el módulo de creación del mapa inicial hace lo siguiente:

- Extrae los puntos de interés de la imagen utilizando el detector de características visuales ORB [39].
- Calcula los descriptores de dichas características visuales utilizando el procedimiento ORB [39].
- Realiza un emparejamiento de las características visuales de la imagen actual y de la imagen anterior empleando un emparejador de fuerza bruta basado en la norma Hamming [56]. Para cada característica visual de la primera imagen se buscan las dos mejores coincidencias en la segunda imagen. Supongamos que tenemos una característica visual de la primera imagen a cuyo vector descriptor denominamos F1 y que las dos mejores coincidencias para esa característica visual en la segunda imagen quedan definidas por los vectores descriptores F2F y F2S. Un emparejamiento se considera aceptable cuando se cumple la siguiente relación:

$$Distancia\_hamming\ (F1, F2F) < umbral * Distancia\_hamming\ (F1, F2S) \quad (3.1)$$

  Donde el umbral varía entre 0 y 1.

  Se aplica además un filtro adicional para descartar malos emparejamientos. Este filtro consiste en comprobar que las características visuales emparejadas cumplen la restricción impuesta por la geometría epipolar:

$$X_i^T * E * X_j = 0 \quad (3.2)$$

  Donde $X_i^T = (u_i, v_i, 1)^T$ son las coordenadas homogéneas de una característica visual de la imagen i, $X_j = (u_j, v_j, 1)$ son las coordenadas homogéneas de su característica visual correspondiente en la imagen j y E es la matriz Esencial.

- Actualiza una estructura de datos que almacena el seguimiento de las características visuales a lo largo de la secuencia de imágenes. Para seguir con la nomenclatura de las figuras 3.1 y 3.2, a esta estructura de datos la denominaremos "Correspondencias 2D-2D". Esta estructura de datos posee un identificador para cada característica visual que nos da acceso a una tabla de tantas filas como veces sea detectada la característica visual en la secuencia de imágenes y tres columnas. En la primera columna, se almacenan los índices de las imágenes en las que ha sido detectada la característica visual. En la segunda columna, se almacenan las coordenadas en píxeles de la posición de la característica visual en cada una de las imágenes que ha sido detectada. Por último, en la tercera columna, se almacenan los índices que nos dan acceso al vector descriptor de la característica visual en cada una de las imágenes. Esta tercera columna es utilizada por el código C++ para realizar ciertas operaciones, pero de aquí en adelante pasaremos a ignorarla porque no aporta información útil para entender el funcionamiento del sistema.

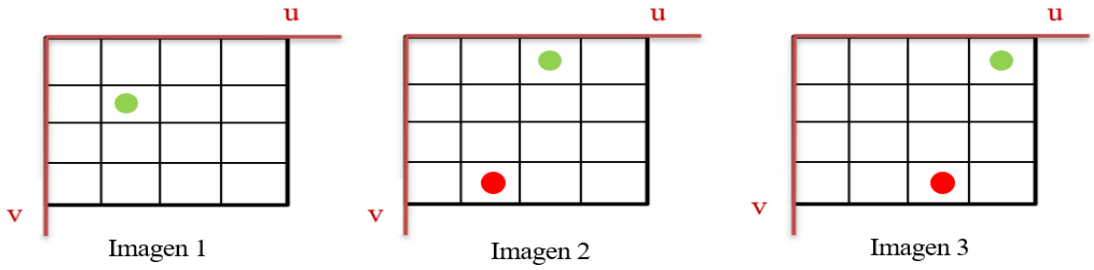
  Para actualizar esta estructura de datos el algoritmo debe determinar si una característica visual es nueva o no. Esta tarea es relativamente sencilla si se consideran dos emparejamientos sucesivos. Supongamos que tenemos tres imágenes, a saber, I1, I2 e I3. Para estas imágenes se calculan los emparejamientos de características visuales entre I1 e I2 y entre I2 e I3. Supongamos, además, que la información correspondiente al emparejamiento de características visuales entre las imágenes I1 e I2 ya está almacenada en la estructura de datos.

Si una característica visual de I1 es emparejada con una de I2, y a su vez esa misma característica visual de I2 es emparejada con una I3, habremos detectado una característica visual que se repite en las tres imágenes.

Como la información correspondiente a dicha característica visual ya existe para las dos primeras imágenes, el algoritmo tan solo tendría que añadir la información de la característica visual correspondiente a la tercera imagen.

Si por el contrario hay alguna característica visual de I2 que no se ha emparejado con alguna de I1, pero sí con alguna de I3, se trataría de una característica visual nueva. En este caso el algoritmo tendría que generar una nueva entrada en la estructura de datos para dicha característica visual, y tendría que añadir la información correspondiente a dicha característica visual para las imágenes I2 e I3.

Para entender un poco mejor el funcionamiento de la estructura de datos se propone la figura 3.1.1 en la que se expone un caso simple de una característica visual que se ha seguido a lo largo de tres imágenes y de otra característica visual que tan solo se ha seguido a lo largo de dos imágenes. La primera característica visual estaría representada por un punto verde, y la segunda por un punto rojo. Se muestran tres hipotéticas imágenes sucesivas de la secuencia. Cada una de estas imágenes está dividida en celdas, siendo cada una de estas celdas un píxel de coordenadas (u,v).



Característica verde (identificador numérico)	Imagen	Coordenadas en la imagen (u,v)
0	1	(2,2)
	2	(3,1)
	3	(4,1)

Característica roja (identificador numérico)	Imagen	Coordenadas en la imagen (u,v)
1	2	(2,4)
	3	(3,4)

Figura 3.1.1: Estructura encargada de realizar el seguimiento de características visuales a lo largo de una secuencia de imágenes.

El módulo de creación del mapa inicial sigue adquiriendo imágenes y repitiendo la secuencia descrita anteriormente hasta que el número de imágenes adquiridas sea igual al número de imágenes deseado para construir el mapa inicial. Llegados a este punto el robot se detiene momentáneamente y deja de capturar imágenes en lo que el sistema calcula un mapa inicial del entorno y las primeras posiciones por las que ha ido pasando el robot.

Lo que hemos conseguido hasta este punto es una estructura de datos en la que se almacenan un conjunto de características visuales y sus respectivas proyecciones en las imágenes en las que se han ido detectando, es decir, disponemos de la información correspondiente al *tracking* de puntos de interés.

La idea ahora es obtener las coordenadas tridimensionales de dichas características visuales. Dado un punto tridimensional en el espacio en coordenadas homogéneas $X_j = (x_j, y_j, z_j, 1)^T$ y su proyección en una determinada imagen i, $x_{ij} = (u_{ij}, v_{ij}, 1)^T$, tenemos que dichos puntos están relacionados a través de la siguiente relación:

$$x_{ij} = A * X_j \tag{3.3}$$

Siendo A una matriz de dimensiones 3x4 formada por el producto de los siguientes elementos:

$$A = K * [R|T] \tag{3.4}$$

Donde K es una matriz de dimensiones 3x3 que contiene los parámetros intrínsecos de la cámara (que en nuestro caso son conocidos), y [R|T] representa una matriz 3x4 que contiene la información correspondiente a la rotación y traslación de la cámara con respecto al sistema de referencia tridimensional, que en nuestro caso sería la posición en la que se encontraba el robot al comenzar el movimiento.

Por consiguiente, nuestro problema consiste en determinar tanto las coordenadas tridimensionales de las características visuales como las posiciones del robot a lo largo de la primera secuencia de imágenes capturadas. Se propone el método sugerido por Szeliski y Kang [57] que utiliza un proceso iterativo para obtener los parámetros desconocidos de nuestro problema, utilizando algoritmos de optimización general como *Levenberg-Marquardt* [50].

El problema se formula como la minimización de todos los errores de reproyección para todas las posiciones de la cámara y todos los puntos tridimensionales del entorno, dadas las proyecciones de dichos puntos tridimensionales:

$$min_{v_{ij}} \sum_i \sum_j \left\| w_j(\pi(v_{ij}) - x_{ij}) \right\|^2 \tag{3.5}$$

Donde π es una función no lineal encargada de proyectar los puntos tridimensionales al plano de la imagen y $v_{ij} = (X_j, [R_i|T_i], K)$ es un vector que contiene los parámetros desconocidos: las coordenadas X_j de los puntos tridimensionales, la posición de la cámara $p_i = [R_i|T_i]$ cuando capturó una determinada imagen i y la matriz de parámetros intrínsecos de la cámara K. w_j representa la ponderación que se le otorga a cada punto tridimensional. Al aumentar las iteraciones esta variable disminuye para aquellos puntos que presentan un error de reproyección alto.

Según [58] si inicializamos todos los puntos tridimensionales en un mismo plano x-y según el siguiente conjunto de ecuaciones y establecemos como posición inicial de todas las posiciones del robot, la posición en la que se capturó la primera imagen, el problema formulado en (3.5) converge:

$$\begin{aligned} X_w &= \frac{u - c_x}{f} * z_{plano} \\ Y_w &= \frac{v - c_y}{f} * z_{plano} \\ Z_w &= z_{plano} \end{aligned} \tag{3.6}$$

Donde X_w, Y_w, Z_w son las coordenadas tridimensionales de una característica visual determinada y z_{plano} es la distancia fija a la que se sitúa el plano x-y en el que inicializamos todos los puntos tridimensionales.

c_x y c_y son las coordenadas del punto principal de la cámara y f sería la distancia focal de la cámara. Estos tres últimos parámetros ya fueron definidos en el apartado 2.3.2 de este documento.

Para que este último procedimiento funcione adecuadamente se requiere que el robot no haya recorrido una gran distancia durante la captura de la primera secuencia de imágenes dedicada a la creación del mapa inicial.

En nuestro caso el algoritmo encargado de resolver este problema de optimización es el optimizador de grafos g2o [53]. Para ello debemos traducir la información de nuestra estructura de datos a un grafo que pueda ser correctamente interpretado por este optimizador.

La forma que debe presentar este grafo se muestra en la figura 3.1.2. En ella se aprecian los siguientes elementos:

- Los vértices de tipo *VertexSE3Expmap* que representan las distintas posiciones de la cámara, y por tanto del robot. En la figura se representan de color negro.
- Los vértices de tipo *VertexSBAPointXYZ* que se corresponden con los puntos 3D del entorno. En la figura se representan de color azul.
- Las aristas de tipo *EdgeProjectXYZ2UV*. En la figura se representan de color rojo. Estas aristas relacionan los dos tipos de vértices anteriores a través de una ecuación de proyección. Se les asocia el valor de las coordenadas bidimensionales de una determinada característica visual en la imagen que dictamina el vértice de tipo *VertexSE3Expmap* que lleva conectado a uno de sus extremos. Este valor es conocido, tan solo tenemos que rescatar dicho valor de la estructura de datos a la que anteriormente denominamos "Correspondencias 2D-2D". El optimizador puede ir comparando este último valor con el valor que obtiene de proyectar el vértice de tipo *VertexSBAPointXYZ* en el plano de la imagen correspondiente al vértice de tipo *VertexSE3Expmap* que se encuentra en el otro extremo de la arista. De esta forma obtendría el error de reproyección para una determinada arista. El optimizador intenta que dicho error sea mínimo para todas las aristas.

Para inicializar los puntos tridimensionales, es decir, los vértices de tipo *VertexSBAPointXYZ*, tan solo tenemos que rescatar de la estructura de datos "Correspondencias 2D-2D" las coordenadas de una determinada característica visual en una determinada imagen y aplicarles el conjunto de ecuaciones 3.6.

Para inicializar las posiciones de la cámara, y por tanto del robot, se hace lo mencionado anteriormente. Se establece que la primera posición del robot sirva de inicialización para todas las demás posiciones por las que fue pasando mientras capturaba la primera secuencia de imágenes. Por consiguiente, todos los vértices de tipo *VertexSE3Expmap* se inicializan en la misma posición.

El optimizador también recibe como variables de entrada los parámetros intrínsecos de la cámara que son conocidos.

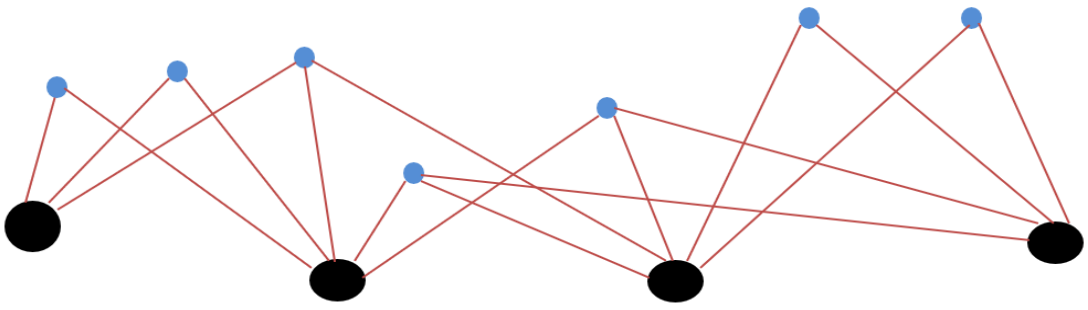


Figura 3.1.2 Forma del grafo a optimizar por el optimizador g2o. En negro se representan las posiciones de las cámaras y en azul las coordenadas tridimensionales de una determinada característica visual. En rojo observamos las distintas aristas que conectan los puntos tridimensionales del entorno a las distintas posiciones de las cámaras a través de una ecuación de proyección.

Una vez el optimizador ha finalizado, ya tenemos tanto un mapa inicial de los puntos tridimensionales del entorno como las posiciones del robot en los instantes de tiempo en los que capturó las imágenes que componen la secuencia utilizada para la creación del mapa inicial.

Para seguir con la nomenclatura de las figuras 3.1 y 3.2, el algoritmo de creación del mapa inicial del entorno almacenaría los puntos tridimensionales obtenidos en una estructura de datos que denominamos "Mapa tridimensional entorno" y las posiciones del robot en la estructura de datos "Posiciones del robot". En la estructura "Mapa tridimensional entorno" cada punto tridimensional llevaría el mismo identificador que el conjunto de características visuales que se utilizaron para generarlo a través del optimizador.

Hay que recalcar que para la creación del mapa inicial no se pueden utilizar demasiadas imágenes, tal y como se mencionó anteriormente. Para que el problema converja adecuadamente el robot no debe haber realizado un desplazamiento demasiado grande. Es por ello por lo que a partir de este momento se necesita un nuevo método para seguir obteniendo la posición del robot a través de correspondencias bidimensionales entre imágenes. Este asunto se trata en los próximos apartados.

3.2 Selección de fotogramas de interés o *"Keyframes"*

Una vez finalizada la creación del mapa inicial a través de una pequeña secuencia de imágenes, ya conocemos tanto un pequeño mapa que describe los puntos tridimensionales del entorno como la trayectoria que recorrió el robot durante la captura de imágenes de dicha secuencia.

Estamos ahora con el robot parado en la posición en la que capturó la última imagen para la creación del mapa inicial (imagen a la que denominamos M en la figura 3.1) y conocemos su posición actual (Posición M en la figura 3.1). En este momento se toma dicha posición como referencia y el robot reanuda el movimiento capturando imágenes nuevamente.

La idea consiste en seguir buscando correspondencias bidimensionales entre la imagen que se ha tomado como referencia (imagen M) y la siguiente imagen que capture el robot a través de su cámara. Estas correspondencias serán utilizadas posteriormente para estimar la trayectoria relativa entre la posición de referencia y la posición en la que el robot capturó la siguiente imagen. Dicho proceso implica calcular la matriz Esencial.

Una vez determinada la trayectoria relativa a través de la matriz Esencial, pueden calcularse puntos tridimensionales del entorno a través de triangulación. La triangulación depende en gran medida de las correspondencias bidimensionales obtenidas y de la magnitud del movimiento relativo realizado por el robot. Es por ello por lo que algoritmo necesita un módulo para determinar cuándo se deben calcular las correspondencias bidimensionales y cuando no, es decir, requiere de un proceso de selección inteligente de fotogramas clave.

Expliquemos esto último con más detalle. Nuestra cámara captura imágenes con una frecuencia de 30 fps. Esto implica que, si consideramos una imagen y la inmediatamente posterior, el movimiento que ha realizado el robot entre ambos instantes es prácticamente imperceptible, ya que tan solo habrían transcurrido 0,033 segundos. Si a través de las correspondencias bidimensionales entre estas dos imágenes obtuviésemos la matriz Esencial y de ella obtuviéramos tanto la traslación como la rotación, observaríamos que el desplazamiento sería muy pequeño.

Supongamos que queremos utilizar estas dos imágenes consecutivas para obtener puntos tridimensionales mediante el método de triangulación. Una vez obtenidas las correspondencias entre las características visuales de ambas imágenes, procederíamos a reproyectar cada pareja de puntos relacionados al plano tridimensional, lo que nos proporcionaría rectas que tan solo intersecan en el infinito. Esto último provocaría que la incertidumbre en la reconstrucción de puntos tridimensionales fuese demasiado alta, lo que haría fallar por completo nuestro algoritmo. En la figura 3.2.1 se muestra una ilustración de como varía la incertidumbre en la reconstrucción de puntos tridimensionales en función de la distancia que separa a las imágenes consideradas para aplicar el método de triangulación.

Queda pues justificada la necesidad de un método que nos permita identificar que fotogramas son útiles para estimar la profundidad de los puntos tridimensionales o, dicho de otra forma, un módulo que nos permita determinar cuándo se ha producido un desplazamiento suficiente por parte del robot para que la incertidumbre en la estimación de la profundidad de los puntos tridimensionales no sea demasiado elevada.

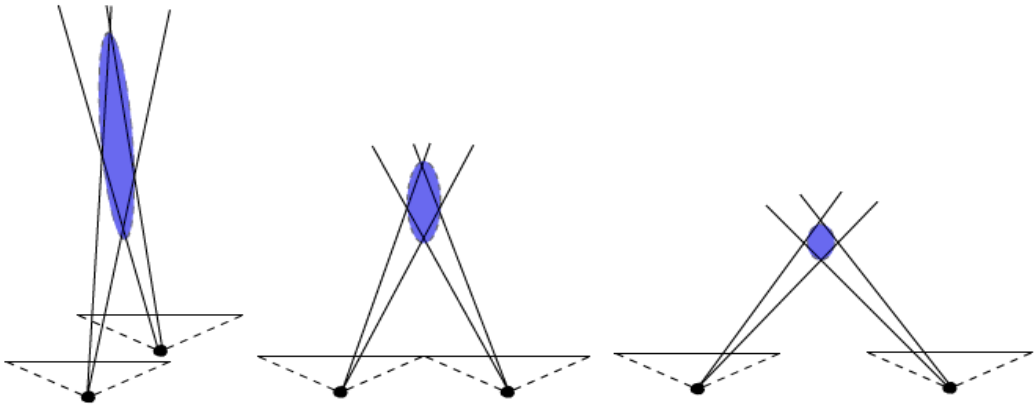


Figura 3.2.1: Incertidumbre en la estimación de la profundidad de los puntos tridimensionales en función de la distancia que separa a las cámaras. La región azul representa dicha incertidumbre.

Expliquemos el proceso de selección inteligente de *keyframes*. La idea es generar un vocabulario a través del modelo de *Bag of Words* [59] que nos permita expresar una determinada imagen como un conjunto de palabras visuales. Para crear dicho vocabulario se utiliza una batería de imágenes representativa del entorno en el que el robot vaya a desarrollar su actividad. Este conjunto de imágenes se obtiene de forma *offline*, de forma que el vocabulario debe generarse antes de que el robot comience su actividad, ya que en cada instante de tiempo deberá poder tener acceso a dicho vocabulario.

Veamos, a grandes rasgos, como funciona la creación de palabras para el vocabulario. En primer lugar, se toman todas las imágenes de la batería y se procede a calcular tanto sus características visuales como los vectores descriptores de dichas características. El método utilizado tanto para la detección como para la descripción de características visuales es el procedimiento ORB [39], ya que es uno de los pocos métodos compatibles con la librería C++ encargada del proceso de creación del vocabulario.

Los vectores descriptores de todas las características visuales presentes en la batería de imágenes son sometidos a un proceso de *Clustering* [60]. Este proceso utiliza un algoritmo iterativo denominado *k-means* a través del cual divide un conjunto de M vectores descriptores en K categorías, siendo K < M. Este algoritmo recibe como parámetros de entrada los M vectores descriptores de características visuales y el número de categorías K en las que se quiere dividir el conjunto de M vectores descriptores. Cada una de estas categorías recibe el nombre de Clúster. En la siguiente tabla se describe el funcionamiento del algoritmo:

Algoritmo 3.2.1: *k-means*

1. Inicializar un vector descriptor representante para cada clúster. Para ello se selecciona un vector descriptor aleatorio del conjunto de vectores descriptores que se le proporcionan al algoritmo.
2. Repetir mientras se produzcan cambios en la asignación de un vector descriptor a un determinado clúster:

   2.1 Paso de asignación: asignar cada uno de los vectores descriptores al clúster cuyo vector descriptor representante sea más cercano según la distancia Hamming. Se utiliza la norma Hamming porque los vectores descriptores proporcionados por el procedimiento ORB son binarios.

   2.2 Paso de modificación: se recalcula el vector descriptor representante para cada clúster como la media de los vectores descriptores que han sido asignados a dicho clúster.

De la tabla anterior concluimos que el algoritmo *k-means* finaliza cuando converge a una solución estable en la que, aunque continuasen las iteraciones, cada uno de los vectores descriptores siempre se asociaría al mismo clúster.

Cada clúster obtenido de este algoritmo pasa a ser una de las palabras visuales de nuestro vocabulario. En los próximos párrafos se describe como se utiliza este vocabulario para la detección inteligente de *keyframes.*

Sea *I_ref* nuestra imagen de referencia e *I_curr* una imagen que es candidata a ser un fotograma clave. Para cada una de estas imágenes se procede a calcular tanto las características visuales como los vectores descriptores de dichas características a través del procedimiento ORB. Una vez se tienen los descriptores de los puntos de interés de cada imagen se procede a buscar en el vocabulario a que palabra visual pertenece cada uno de estos vectores descriptores o, lo que es lo mismo, a que clúster han sido asignados. Como siempre, el clúster adecuado será aquél cuyo vector descriptor representante presente una menor distancia Hamming con respecto al vector descriptor considerado.

Esta asignación nos permite construir para cada imagen un histograma que contabiliza el número de veces que cada palabra visual ha sido detectada en la imagen. Veámoslo con un ejemplo muy simple. Sean A, B, C y D el nombre que reciben cuatro clústeres diferentes en un vocabulario compuesto únicamente por cuatro palabras. Cada uno de estos clústeres tiene un vector descriptor representante de dos componentes en el que la primera componente representaría el color de la característica visual (0 para color rojo, 1 para color azul) y la segunda componente representaría la forma de la característica visual (0 para forma ovalada, 1 para forma rectangular). Por consiguiente, nuestro vocabulario sería el siguiente:

- A [0,0] = característica roja y ovalada.
- B [0,1] = característica roja y rectangular.
- C [1,0] = característica azul y ovalada.
- D [1,1]= característica azul y rectangular.

Una vez tenemos el vocabulario analizamos las imágenes y construimos el histograma que representa la frecuencia con la que las palabras aparecen en la imagen. Se consideran las dos imágenes hipotéticas que se representan en la figura 3.2.2. En dichas imágenes se ha omitido el resto de la imagen y aparecen tan solo las características visuales. Al pie de cada imagen aparecen los histogramas que representan la frecuencia con la que cada palabra del vocabulario aparece en la imagen.

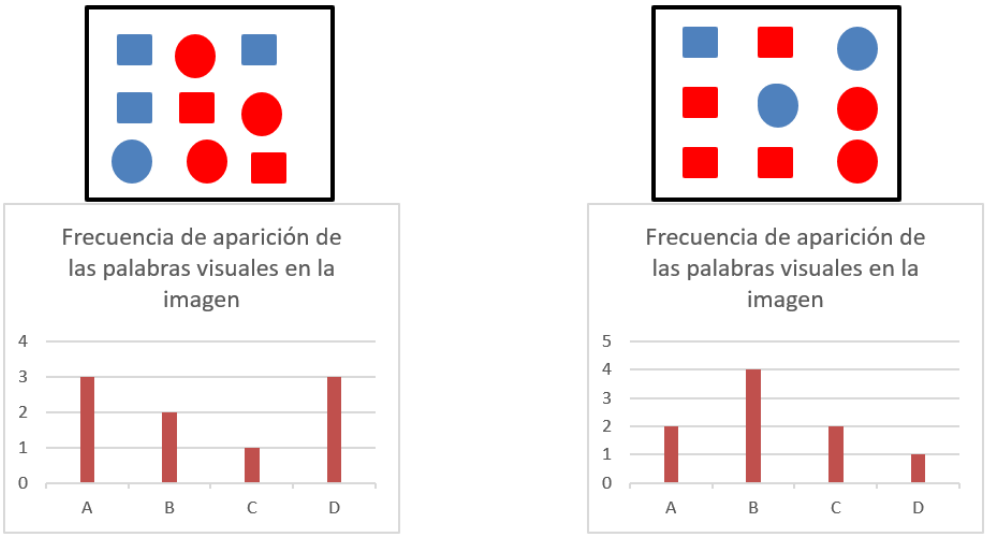


Figura 3.2.2 Ejemplo ficticio de histograma de aparición de palabras del vocabulario en una imagen determinada.

Evidentemente este es un caso muy simple que solo sirve para facilitar la comprensión del método empleado. En lo que concierne a este proyecto, el vector descriptor de cada clúster contiene 512 componentes.

Tras calcular los vectores descriptores de las características visuales de las imágenes *I_ref* e *I*_curr y una vez obtenidos los histogramas de frecuencia de aparición de las palabras del vocabulario en dichas imágenes, se procede a comparar dichos histogramas.

Si con nuestra cámara que captura imágenes a una frecuencia de 30fps se considerase que *I_curr* es la imagen que se captura inmediatamente después de *I_ref*, tan solo habrían transcurrido 0,033 segundos entre una captura y la siguiente. Esto implicaría que las características visuales obtenidas para ambas imágenes serían prácticamente idénticas, ya que representarían prácticamente la misma zona de la escena.

Al tener características visuales tan parecidas el histograma de aparición de palabras del vocabulario en la imagen *I_ref* es prácticamente idéntico al histograma de aparición de palabras del vocabulario en la imagen *I_curr*. De esta información deducimos que el movimiento realizado por el robot móvil entre la captura de una imagen y la siguiente es insuficiente para que el método de triangulación de puntos tridimensionales funcione adecuadamente.

Lo que el algoritmo hace para dotar de inteligencia al robot a la hora de seleccionar fotogramas clave es tomar una imagen como referencia y calcular el histograma de aparición de palabras del vocabulario en dicha imagen. Una vez que lo tiene, va comparando este histograma con el histograma que obtiene de las imágenes que va capturando posteriormente.

De esta comparación se obtiene una puntuación entre 0 y 1, donde 1 implica que los histogramas son idénticos y 0 implica que no se parecen en nada. Cuando la puntuación resultante de comparar ambos histogramas decae por debajo de un cierto umbral, se considera que la imagen en cuestión es un fotograma clave o, en otras palabras, se ha conseguido una imagen que no es prácticamente igual a la anterior y que comparte algunas características visuales con ella. Esto último se traduce en que el robot ha realizado un desplazamiento suficiente para que la imagen *I_curr* abarque una nueva zona de la escena que aún contiene una pequeña región de la zona de la escena que abarcaba la imagen *I_ref*. En estas condiciones ya es viable estimar el movimiento relativo que ha realizado el robot entre la captura de la imagen *I_ref* y la captura del nuevo *keyframe.*

Para la creación del vocabulario y para calcular la puntuación resultante de comparar los histogramas se ha recurrido a la librería programada en C++ DBoW2 [61]. Si el lector de este documento desea conocer más detalles sobre la implementación de esta librería se ruega que consulte la siguiente referencia bibliográfica [62].

3.3 Emparejamiento robusto de características visuales

Una vez detectado el siguiente fotograma clave, el siguiente paso consiste en emparejar las características visuales de dicho fotograma con las del *keyframe* anterior. Para ello se optó por utilizar un emparejador de tipo Fuerza Bruta. Se determinó que, a pesar de tener un coste de ejecución ligeramente superior, el emparejador de tipo Fuerza Bruta presentaba un mejor desempeño que las técnicas *Approximate-Nearest Neighbors*. En nuestro caso, tan solo se tenían que emparejar 500 características visuales ORB por cada par de imágenes y, en este contexto, el emparejador de tipo Fuerza Bruta ofrecía más emparejamientos válidos que cualquiera de las técnicas implementadas en la librería de software FLANN (*Fast Library for Approximate Nearest Neighbors* [43]).

No obstante, tras este primer emparejamiento, aún quedan bastantes asociaciones erróneas. Se propone nuevamente el filtro de la ecuación (3.1). Este filtro propone obtener las dos mejores coincidencias para una determinada característica visual de la primera imagen en la segunda imagen.

Para estas coincidencias se calcula la distancia de sus vectores descriptores al vector descriptor de la característica visual de la primera imagen. Si el cociente entre estas distancias permanece por debajo de un determinado umbral, el emparejamiento entre la característica visual de la primera imagen y su mejor coincidencia en la segunda imagen se considera válido.

Tras aplicar este primer filtro aún existen posibilidades de encontrar malos emparejamientos. Para mejorar el filtrado se consideran las restricciones geométricas que impone la geometría epipolar.

En este punto se puede utilizar cualquiera de los siguientes métodos para filtrar los emparejamientos:

- Obtener la matriz Fundamental con el algoritmo de los 8 puntos [63].
- Calcular la matriz Esencial con el algoritmo de los 5 puntos [23].

Tanto para el cálculo de la matriz Esencial como para el cálculo de la matriz Fundamental se utiliza un algoritmo basado en el esquema RANSAC, ya explicado en el apartado en el apartado 2.7 de este documento. Las ventajas y desventajas de utilizar la matriz Esencial o la matriz Fundamental para el filtrado de emparejamientos erróneos se muestran resumidas en la tabla 3.3.1.

	Matriz Fundamental	Matriz Esencial
Restricciones	El método es impreciso cuando el movimiento producido entre las dos imágenes consideradas es paralelo al plano de la imagen	No presenta restricciones
Complejidad	Es menor, ya que tan solo hay que resolver un sistema de ecuaciones lineales	Es mayor, puesto que hay que calcular los coeficientes de un polinomio de grado 10 y posteriormente calcular sus raíces
Precisión	Menor, es muy susceptible al ruido	Mayor, la solución explota mejor las restricciones geométricas impuestas por la geometría epipolar
Características Particulares de Aplicación del Algoritmo RANSAC	El algoritmo debe generar un mayor número de hipótesis para encontrar una solución aceptable, pero el coste de ejecutar cada una de ellas es menor	El algoritmo requiere un menor número de hipótesis, pero el tiempo requerido para ejecutar cada una de ellas es mayor

Tabla 3.3.1: Comparación entre el filtrado de puntos utilizando la matriz Esencial y el filtrado de puntos utilizando la matriz Fundamental.

Recordemos que la matriz Fundamental se puede calcular sin conocer la calibración de la cámara, mientras que la matriz Esencial no. En nuestro caso conocemos los parámetros intrínsecos de la cámara. Por este motivo, y dada su mayor precisión, se utiliza el método de la matriz Esencial para filtrar los malos emparejamientos.

3.4 Estimación de la trayectoria

Recapitulemos un poco. Veamos de que información disponemos hasta el momento y en qué punto del algoritmo de la figura 3.1 nos encontramos. Ya hemos finalizado la creación del mapa inicial y hemos tomado como referencia la posición en la que el robot capturó la última imagen que se utilizó para crear el mapa inicial. Esta posición, como ya se mencionó anteriormente, es conocida puesto que el optimizador descrito en la sección 3.1 nos devuelve tanto los puntos tridimensionales del entorno como las posiciones por las que fue pasando el robot al capturar las imágenes que se utilizaron para la creación del mapa. Posteriormente, hemos dejado que el robot siga capturando imágenes y hemos detectado un nuevo *keyframe* utilizando el método propuesto en el apartado 3.2. Una vez obtenido el nuevo fotograma clave hemos realizado un emparejamiento robusto de características visuales entre la imagen de referencia y el nuevo fotograma y de este emparejamiento hemos obtenido la matriz Esencial.

Estamos ahora en el bloque "Estimación del movimiento relativo a través de la matriz Esencial" de la figura 3.1. La idea consiste en extraer la rotación y traslación relativas entre la posición de referencia y la nueva posición, que aún es desconocida.

La matriz Esencial que calculamos en el apartado anterior para filtrar los malos emparejamientos de características visuales se puede descomponer como sigue:

$$E \approx t_k^* R_k \tag{3.7}$$

Donde:

- $t_k = [t_x\ t_y\ t_z]^T$ es el vector de traslación.
- R_k es la matriz de rotación
- $t_k^* = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}$
- E es la matriz Esencial

La ecuación 3.7 es algo ambigua, puesto que se cumple para un determinado factor de escala, que no es necesariamente el de nuestro problema.

Para poder extraer la rotación y la traslación de la matriz Esencial se plantea un problema de descomposición en valores singulares [64] de la matriz Esencial:

$$E = UDV^T \tag{3.8}$$

Donde $U\ y\ V^T$ son matrices ortogonales y D es una matriz formada con los valores singulares de la matriz Esencial en su diagonal principal. Dadas las restricciones de la propia matriz Esencial, la matriz D debe presentar la siguiente forma:

$$D = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.9}$$

Es decir, la matriz D debe presentar dos valores singulares idénticos y uno nulo.

Si definimos una matriz W como la que sigue:

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{con} \quad W^T = W^{-1} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

Podemos afirmar que se cumplen las siguientes relaciones:

$$t_k^* = UWDU^T \tag{3.11}$$
$$R = UW^{-1}V^T$$

Del conjunto de ecuaciones 3.11 deducimos que para una determinada matriz Esencial existen dos posibles rotaciones y dos posibles traslaciones, es decir, hasta cuatro posibilidades de distintas combinaciones entre traslaciones y rotaciones. Dichas posibilidades vienen descritas en el siguiente conjunto de ecuaciones:

$$\begin{aligned} R_1 &= UW^T V^T \\ R_2 &= UWV^T \\ T_1 &= T \\ T_2 &= -T \end{aligned} \tag{3.12}$$

Donde U y V son las matrices en la ecuación 3.8.

El vector T, que representa al vector de traslación, se obtendría de resolver el siguiente sistema homogéneo:

$$ET = 0 \tag{3.13}$$

Otra posibilidad sería considerar que el vector T es la tercera columna de la matriz U.

Una vez obtenidas las posibles rotaciones y traslaciones relativas entre la posición de referencia y la nueva posición, la siguiente tarea consiste en determinar cuál de las cuatro combinaciones de posibles de traslaciones y rotaciones es la adecuada.

Para ello se procede como sigue:

- Calculamos las 4 posibles nuevas posiciones utilizando las diferentes rotaciones y traslaciones obtenidas. Como conocemos la posición de la primera cámara, la posición en la que se sitúa la segunda cámara se obtiene de multiplicar la posición de la primera cámara por una matriz de transformación construida a través de la rotación y traslación relativas.
- Utilizamos una característica visual que haya sido detectada en ambas imágenes y obtenemos sus coordenadas tridimensionales mediante triangulación. Para poder triangular se requieren las matrices de proyección de ambas cámaras. Dichas matrices se construyen con los parámetros intrínsecos de la cámara y con las posiciones de las cámaras. Estas últimas se calcularon en el punto anterior.
- De las 4 posibles soluciones, tomamos aquella en la que el punto tridimensional obtenido quede frente al plano de ambas cámaras.

Supongamos que las matrices de rotación y traslación válidas se denominan respectivamente R_end y t_end, donde R_end es una matriz de dimensiones 3x3 y t_end es una matriz de dimensiones 3x1. Si la posición de las cámaras viene expresada mediante una matriz de transformación homogénea de dimensiones 4x4, la posición de la cámara en el nuevo *keyframe* vendrá dada por:

$$Pos_{curr} = Pos_{ref} \begin{bmatrix} R\_end & t\_end \\ 0\ 0\ 0 & 1 \end{bmatrix} \tag{3.14}$$

Donde Pos_{curr} sería la posición de la cámara en el nuevo *keyframe* y Pos_{ref} la posición de la cámara que tomamos como referencia.

El problema es que, a pesar de que la trayectoria calculada si va en la misma dirección que la trayectoria real, la magnitud del desplazamiento no es la misma. Esto último es debido a que la matriz Esencial se calcula para un determinado factor de escala, cuyo valor no podemos predecir.

Es por ello por lo que nuestro sistema requiere de algún método para calcular un factor de escala relativa. Una vez obtenido dicho factor de escala el problema es trivial, ya que tan solo tenemos que multiplicar el vector de traslación por dicho factor de escala para que la trayectoria estimada sea de los mismos órdenes de magnitud que la trayectoria real.

En los próximos apartados se explica cómo se obtiene dicho factor de escala.

3.5 Primera triangulación de puntos tridimensionales

En este punto, ya conocemos las posiciones del robot en dos *keyframes* consecutivos, a falta de resolver el problema de escala.

Para obtener los puntos tridimensionales del entorno a través de las posiciones del robot en dos *keyframes* consecutivos se utiliza el método de triangulación polinomial descrito en el artículo de Hartley y Sturm [48]. Las nociones básicas sobre este método de triangulación ya se explicaron en el apartado 2.8.3 de este documento.

En nuestro sistema, Hay una función de la librería OpenCV que se encarga de implementar este método. Como parámetros de entrada recibe las posiciones de las cámaras, los parámetros intrínsecos de las mimas y las características visuales de ambas imágenes que hemos sido capaces de relacionar mediante nuestro proceso de emparejamiento robusto de características visuales.

Esta función nos devuelve los puntos tridimensionales del entorno. Nuevamente estos puntos tridimensionales estarían sujetos al problema de escala, es decir, guardan una dependencia lineal con las coordenadas reales de los puntos tridimensionales del entorno:

$$Punto_{real} = Punto_{triangulado} * f_{escala} \tag{3.15}$$

En el próximo apartado se describe como se calcula el factor de escala.

3.6 Cálculo del factor de escala relativa

Recordemos que en el apartado de creación del mapa inicial generamos un pequeño mapa que almacenaba unos cuantos puntos tridimensionales del entorno. Dicho mapa recibe el nombre de “Mapa tridimensional entorno” en la figura 3.1.

El robot se encuentra ahora en la posición del *keyframe* actual y tomaremos como referencia el keyframe anterior para explicar el funcionamiento del método de obtención del factor de escala relativa.

Para el *keyframe* de referencia conocemos las coordenadas tridimensionales de algunas de sus características visuales. Estas coordenadas 3D se encuentran almacenadas en la ya mencionada estructura de datos a la que denominamos “Mapa tridimensional entorno”. Además, llegados a este punto del algoritmo, ya conocemos el emparejamiento de características visuales entre el *keyframe* de referencia y el *keyframe* actual.

La tarea consiste en averiguar qué características visuales de la imagen actual están relacionadas con puntos tridimensionales que ya existen en nuestra estructura de datos. Esta tarea es trivial, puesto que cada punto tridimensional del entorno recibe en la estructura de datos “Mapa tridimensional entorno” el mismo identificador que la componente de la estructura de datos “Correspondencias 2D-2D” que se utilizó para generar dicho punto tridimensional. Por consiguiente, solo tenemos que utilizar nuestro emparejamiento robusto de características visuales para buscar que puntos de interés de la imagen actual existen en la imagen de referencia y tomar las coordenadas que tengan dichas características visuales en la imagen de referencia.

Con esta información y con el índice que identifica a la imagen de referencia, accedemos a la estructura de datos "Correspondencias 2D-2D" y buscamos los identificadores numéricos de las componentes de la estructura de datos para las que tanto el índice de imagen como las coordenadas de la característica visual coincidan. Este identificador nos da acceso a la componente de la estructura de datos "Mapa tridimensional entorno" que contiene las coordenadas del punto tridimensional en cuestión.

Gracias a este último procedimiento hemos detectado con que puntos tridimensionales del entorno se corresponden algunas de las características visuales de la imagen actual.

Prestando un poco más de atención podemos darnos cuenta de que, para algunas de las características visuales de la imagen actual, conocemos tanto los puntos tridimensionales del entorno con los que están relacionadas como los puntos tridimensionales que se obtuvieron a través de ellas mediante triangulación. Estas características visuales son aquellas que están relacionadas con características visuales de la imagen de referencia de las que conocemos las coordenadas 3D y que además han sido utilizadas para reconstruir puntos 3D a través del proceso de triangulación.

De este último párrafo concluimos que hemos encontrado una serie de correspondencias "3D-3D" entre puntos del entorno real y puntos que se han obtenido mediante triangulación y que, por tanto, están mal escalados.

Para calcular un factor de escala que nos permita redimensionar los puntos 3D mal escalados se propone el siguiente algoritmo:

Algoritmo 3.6.1: Obtención del factor de escala relativa

- Repetir mientras existan correspondencias 3D-3D:
    1. Se utilizan dos correspondencias 3D-3D del conjunto de todas las disponibles. Este par de correspondencias estaría constituido por dos puntos reales del entorno (extraídos de nuestra estructura de datos) y sus correspondientes puntos tridimensionales que presentan el problema de estar escalados inadecuadamente (los obtenidos por triangulación).
    2. Se calculan dos distancias según la norma Euclídea: la distancia entre los dos puntos reales del entorno y la distancia entre los dos puntos obtenidos por triangulación.
    3. Se calcula el cociente entre la distancia obtenida de los puntos reales del entorno y la distancia obtenida de los puntos obtenidos por triangulación. Se agrega dicho cociente a un vector encargado de almacenar todos los cocientes para cada par de correspondencias 3D-3D.
- Llegados a este punto tenemos un vector que almacena los cocientes mencionados anteriormente para todas las correspondencias "3D-3D". Procedemos a ordenar las componentes de dicho vector de menor a mayor y tomamos como factor de escala relativa la mediana de dicho vector. Esto último se debe a que aún pueden existir malos emparejamientos de características visuales que hayan generado malas correspondencias 3D-3D. Al ordenar este vector, los valores inadecuados quedarían en los extremos. Por consiguiente, si consideramos que nuestro factor de escala es la mediana de dicho vector, los *outliers* o malos emparejamientos no estarían afectando al cálculo del factor de escala relativa.

3.7 Reescalado del movimiento relativo, 2º Triangulación de puntos 3D y Actualización de variables

Llegados a este punto tan solo tenemos que aplicar el factor de escala obtenido del apartado anterior al vector de traslación que se obtuvo en el apartado 3.4 de este documento. De esta forma conseguimos que el desplazamiento calculado por nuestro algoritmo esté en la misma escala que el desplazamiento realizado por el robot móvil entre dos *keyframes* consecutivos. Respetando la nomenclatura de la ecuación 3.14 el nuevo vector de traslación pasaría a ser:

$$t\_new = t\_end * factor\_escala\_relativa \tag{3.16}$$

Y por consiguiente la posición del robot en el *keyframe* actual sería:

$$Pos_{curr} = Pos_{ref} \begin{bmatrix} R\_end & t\_new \\ 0\ 0\ 0 & 1 \end{bmatrix} \tag{3.17}$$

Dado que ya tenemos la trayectoria en los órdenes de magnitud correctos, tan solo tenemos que repetir el proceso de triangulación descrito en el apartado 3.5. En esta ocasión la función encargada de obtener los puntos tridimensionales a través del método de triangulación recibe como entrada unas posiciones de la cámara que no están supeditadas al problema de escala. Por este motivo los puntos tridimensionales que proporciona la función estarían en la misma escala que los puntos tridimensionales del entorno.

A continuación, se procede a actualizar las estructuras de datos y las variables del algoritmo. Para mantener la nomenclatura, las estructuras de datos reciben el mismo nombre que se les asigna en la figura 3.1:

- A la estructura "Correspondencias 2D-2D" se añaden todas las características visuales que han superado el emparejamiento robusto entre el *keyframe* de referencia y el *keyframe* actual. Se contemplan dos posibilidades. La primera sería que una determinada característica visual ya exista en la estructura, es decir, que se haya detectado tanto en los *keyframes* actual y de referencia como en *keyframes* correspondientes a iteraciones anteriores del algoritmo. En este caso se actualiza una entrada de la estructura de datos ya existente. La segunda posibilidad es que una determinada característica visual que haya superado el emparejamiento robusto se detecte por primera vez en el *keyframe* de referencia. En este caso se genera una nueva entrada en la estructura de datos que contiene la información de dicha característica visual para los *keyframes* actual y de referencia.
- A la estructura "Mapa tridimensional entorno" se agregan los puntos tridimensionales nuevos obtenidos mediante triangulación. Estos puntos serían aquellos que provienen de aplicar el método de triangulación a características visuales que han superado el emparejamiento robusto y que no han sido utilizadas para la obtención del factor de escala relativa, es decir, aquellas características visuales que representan a puntos tridimensionales cuya primera aparición en la escena se produce en el *keyframe* de referencia.
- Se actualiza la estructura de datos "Posiciones del robot" añadiendo la posición actual del robot. El valor de esta posición es el que se indica en la ecuación 3.17.

Llegados a este punto tenemos dos posibilidades. La primera sería que el robot siga su movimiento, en cuyo caso se establece que el *keyframe* actual pase a ser el nuevo *keyframe* de referencia y se procede a repetir todo el proceso descrito en los apartados 3.2, 3.3, 3.4, 3.5, 3.6 y 3.7.

La otra posibilidad sería que el robot finalice el movimiento. En este caso se procede a representar gráficamente el movimiento realizado por el robot y la nube de puntos tridimensionales del entorno. Tras esto, el algoritmo llegaría a su fin.

4 EXPERIMENTACIÓN Y RESULTADOS

El objetivo de este apartado es probar el funcionamiento de nuestro algoritmo. Para poder realizar esta tarea se requiere un conjunto de imágenes que haya capturado el robot móvil a lo largo de su desplazamiento en un entorno determinado.

En primer lugar, se describen tanto la secuencia de imágenes utilizada como el sensor que fue utilizado para capturarlas. Acto seguido se procede a analizar el funcionamiento de los bloques más importantes de nuestro algoritmo por separado.

4.1 Secuencia de imágenes y sensor utilizados

Hemos utilizado el conjunto de imágenes *freiburg1_xyz* confeccionado por el departamento de informática de TUM *(Technical University of Munich)* [66], [67].

Este *dataset* está constituido por un conjunto de imágenes tomadas por el sensor Kinect desarrollado por Microsoft.

Figura 4.1: Sensor Kinect.

Este sensor proporciona imágenes RGB-D con resolución de 640 x 480 píxeles a una frecuencia de captura de 30 fps. En nuestro caso solo utilizaremos la información RGB, es decir, las imágenes a color. Como el objetivo del proyecto es conseguir un sistema de odometría visual que tan solo requiera la información correspondiente a imágenes bidimensionales, no tendremos en cuenta la información de profundidad que proporciona el sensor.

La calibración de la cámara es conocida, y se muestra en la tabla 4.1. En esta tabla los coeficientes de distorsión d0, d1, d2, d3 y d4 corresponderían respectivamente a los coeficientes k1, k2, p1, p2 y k3 del modelo matemático descrito en el apartado 2.3.2. Para este mismo modelo matemático los coeficientes k4, k5 y k6 se consideran nulos.

Parámetros Intrínsecos Cámara	fx	fy	cx	cy	d0	d1	d2	d3	d4
	517.3	516.5	318.6	255.3	0.2624	-0.9531	-0.0054	0.0026	1.1633

Tabla 4.1: Parámetros intrínsecos del sensor Kinect.

Como siempre, los parámetros cx y cy representan las coordenadas del punto principal de la cámara y fx y fy son las distancias focales de la cámara en las direcciones x e y.

Para obtener la secuencia de imágenes, el sensor Kinect se dispuso apuntando a una mesa de oficina. Una vez situado frente a la mesa, se procedió a desplazar el sensor realizando movimientos de traslación a lo largo de los ejes X, Y e Z, tratando de mantener fija la orientación.

4.2 Análisis del módulo de emparejamiento de características visuales

Tanto para la creación del mapa inicial del entorno como para el proceso de obtención de la trayectoria se requiere un conjunto de correspondencias 2D-2D. Recordemos que para obtener unos emparejamientos viables se proponían los filtros descritos en el apartado 3.3 de este documento. En este apartado se analiza la efectividad de dichos filtros.

El primer filtro consideraba una característica visual de un determinado *keyframe* de referencia y buscaba las dos mejores coincidencias para dicha característica visual en el siguiente *keyframe*. Posteriormente calculaba la distancia entre cada uno de los vectores descriptores de las coincidencias obtenidas y el vector descriptor de la característica visual del *keyframe* de referencia. Una vez calculadas estas dos distancias procedía a dividirlas. El emparejamiento se consideraba válido si este cociente permanecía por debajo de un determinado umbral que variaba entre 0 y 1.

Nuestra tarea consiste en determinar el valor idóneo para dicho umbral. Un valor muy pequeño indicaría que el vector descriptor de la mejor coincidencia se parece mucho más al vector descriptor de la característica visual de referencia que el vector descriptor de la segunda mejor coincidencia. Esto último se traduce en que obtendríamos emparejamientos entre características visuales muy singulares, es decir, que no guardan parentesco alguno con otros puntos de interés presentes en la imagen. Aunque esto último proporcionaría emparejamientos muy robustos, reduciría demasiado la cantidad de información disponible.

Si por el contrario estableciésemos un valor muy alto para dicho umbral estaríamos obteniendo unos emparejamientos de características visuales poco fiables. Esto último se debe a que se estarían utilizando características visuales de la imagen que se parecen demasiado a otros puntos de interés de la misma imagen. Por consiguiente, a la hora de realizar el emparejamiento de estas características visuales con las del siguiente *keyframe*, sería muy fácil obtener una asociación errónea.

De los dos últimos párrafos concluimos que tenemos que llegar a una solución de compromiso entre la cantidad de información disponible y el grado de precisión requerido en el emparejamiento. Experimentalmente se determinó que los valores idóneos para este umbral oscilaban entre 0.65 y 0.75. En el caso particular de este proyecto se considera un valor de 0.7.

Una vez superado este primer filtro se propone un segundo filtro en el que se introduce la restricción impuesta por la geometría epipolar.

Dado que los emparejamientos de características visuales van a ser utilizados para estimar la trayectoria recorrida por el robot móvil, es necesario determinar que emparejamientos de los que han superado el primer filtro son capaces de satisfacer la geometría epipolar.

Para esta tarea se utiliza el algoritmo RANSAC para estimar la matriz esencial. Este algoritmo va tomando subconjuntos de los emparejamientos que han superado el primer filtro y construye con ellos distintas matrices Esenciales. Posteriormente comprueba, para todos los emparejamientos proporcionados al algoritmo, cuantos son capaces de satisfacer la siguiente ecuación:

$$X'^{T} E X \leq umbral\_distancia \tag{4.1}$$

Donde X' y X representan las coordenadas de dos características visuales de *keyframes* diferentes que han sido relacionadas y E es la matriz Esencial.

El algoritmo RANSAC establece como válida aquella matriz Esencial para la que existe un mayor número de emparejamientos de características visuales que satisfacen la ecuación 4.1.

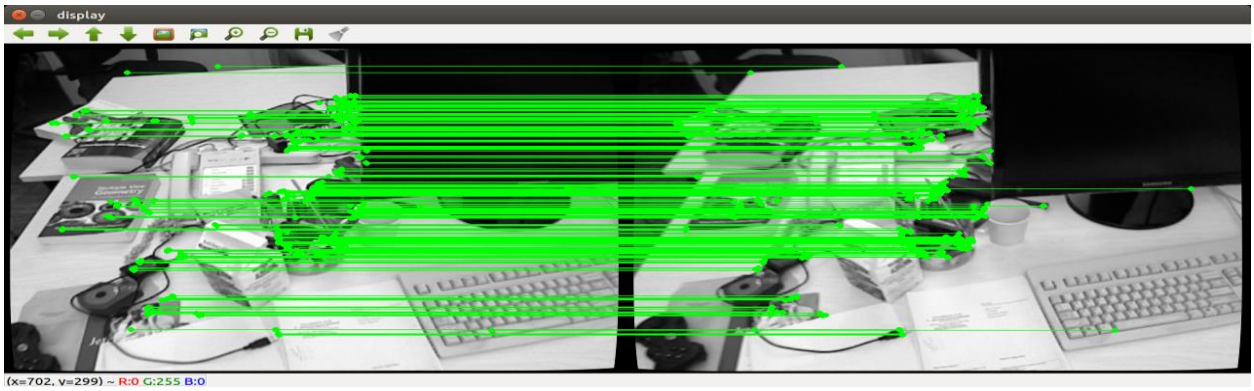
Nosotros tenemos la posibilidad de alterar los dos siguientes parámetros del algoritmo RANSAC:

- La probabilidad de éxito del algoritmo. Este parámetro influye directamente sobre el número de iteraciones a realizar por el algoritmo RANSAC. A mayor probabilidad de éxito mayor número de iteraciones se requieren, tal y como se puede deducir de la ecuación 2.25.
- La distancia mínima usada por el algoritmo RANSAC para determinar si un punto es parte del modelo estimado. En nuestro caso sería el umbral de distancia que aparece en la ecuación 4.1.

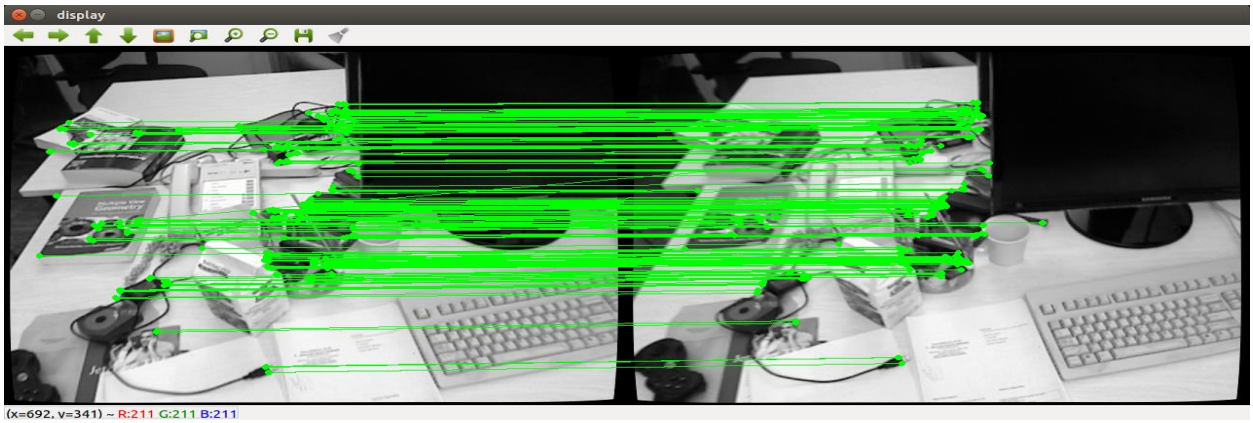
Dado que nuestro algoritmo es muy susceptible a malos emparejamientos, se requiere que se consideren solo aquellos emparejamientos que satisfagan plenamente la restricción epipolar. Es por ello por lo que se establece que la probabilidad de éxito del algoritmo sea del 99,9% y el umbral de distancia sea la unidad.

En la figura 4.2.1 se muestra el resultado de aplicar estos filtros a dos *keyframes* consecutivos de la secuencia. Se muestra también el número de emparejamientos resultante tras aplicar cada filtro.

Resultado tras el emparejamiento bruto: 500 emparejamientos.



Resultado tras el primer filtro: 307 emparejamientos.



Resultado tras el segundo filtro: 178 emparejamientos.

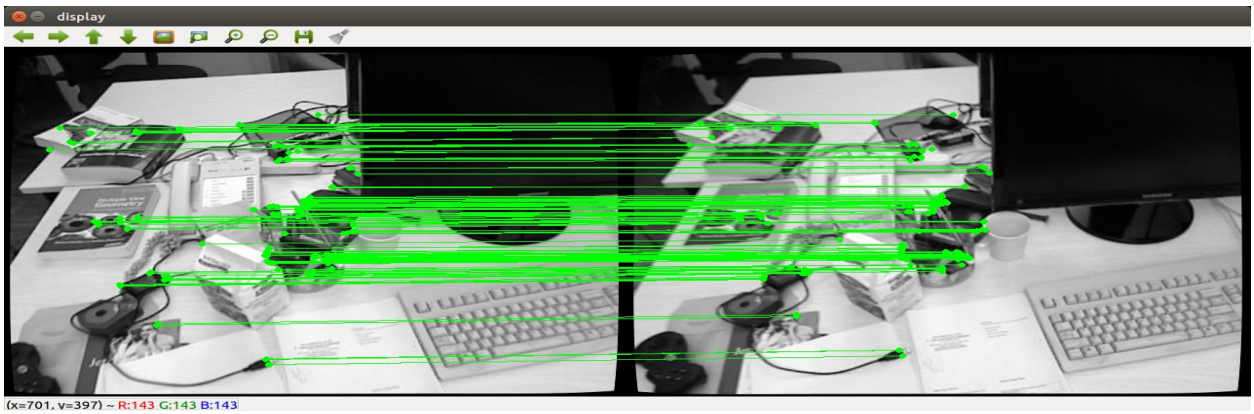


Figura 4.2.1 Resultado de aplicar los dos filtros propuestos a un par *keyframes* consecutivos de la secuencia.

4.3 Análisis del módulo de creación del mapa inicial del entorno

En este módulo se necesitan definir dos parámetros:

- El número adecuado de imágenes para construir el mapa inicial.
- La distancia existente entre el eje de coordenadas de la cámara y el plano x-y en el que se inicializan todos los puntos tridimensionales que se le proporcionan al optimizador encargado de construir el mapa inicial.

Se procede a determinar empíricamente cual sería la mejor inicialización. Para ello se proponen distintas combinaciones de los dos parámetros mencionados anteriormente. Para cada combinación se proporciona la suma de los errores resultantes de comparar punto a punto la trayectoria obtenida del optimizador y la trayectoria real descrita por el robot móvil. También se proporciona un error promedio, que se calcula dividiendo la suma anterior entre el número de imágenes implicadas en la creación del mapa inicial.

En las tablas 4.3.1, 4.3.2, 4.3.3 y 4.3.4 se proporcionan los resultados mencionados anteriormente para creaciones del mapa inicial con 20, 30, 40 y 50 imágenes y para profundidades del plano x-y que varían entre 0,5 y 1,75 metros.

Nº de Imágenes	Profundidad del plano x-y (metros)	Error acumulado punto a punto (metros)	Error promedio (metros)
20	0,5	1,5135	0,0757
20	0,75	1,0595	0,0530
20	1	0,6800	0,0340
20	1,25	0,5509	0,0275
20	1,5	0,6238	0,0312
20	1,75	0,2582	0,0129

Tabla 4.3.1: Creación del mapa inicial con 20 imágenes y profundidades del plano x-y comprendidas entre 0,5 y 1,75 metros.

Nº de Imágenes	Profundidad del plano x-y (metros)	Error acumulado punto a punto (metros)	Error promedio (metros)
30	0,5	4,1889	0,1396
30	0,75	3,5208	0,1174
30	1	2,6740	0,0891
30	1,25	2,1572	0,0719
30	1,5	1,4435	0,0481
30	1,75	0,8020	0,0267

Tabla 4.3.2: Creación del mapa inicial con 30 imágenes y profundidades del plano x-y comprendidas entre 0,5 y 1,75 metros.

Nº de Imágenes	Profundidad del plano x-y (metros)	Error acumulado punto a punto (metros)	Error promedio (metros)
40	0,5	6,4884	0,1622
40	0,75	4,8341	0,1209
40	1	3,5069	0,0877
40	1,25	2,5373	0,0634
40	1,5	1,5254	0,0381
40	1,75	1,3708	0,0343

Tabla 4.3.3: Creación del mapa inicial con 40 imágenes y profundidades del plano x-y comprendidas entre 0,5 y 1,75 metros.

Nº de Imágenes	Profundidad del plano x-y (metros)	Error acumulado punto a punto (metros)	Error promedio (metros)
50	0,5	8,8129	0,1763
50	0,75	7,0783	0,1416
50	1	4,9728	0,0995
50	1,25	2,8396	0,0568
50	1,5	0,4928	0,0099
50	1,75	1,4838	0,0297

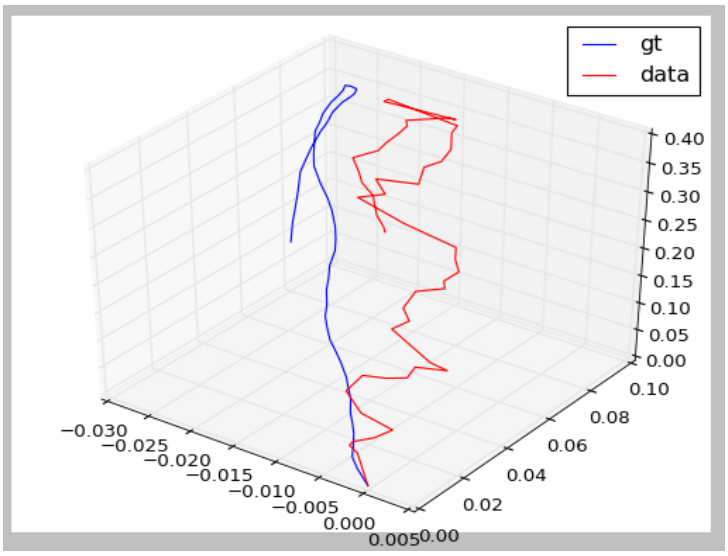
Tabla 4.3.4: Creación del mapa inicial con 50 imágenes y profundidades del plano x-y comprendidas entre 0,5 y 1,75 metros.

De las tablas anteriores se consideran los dos mejores resultados:

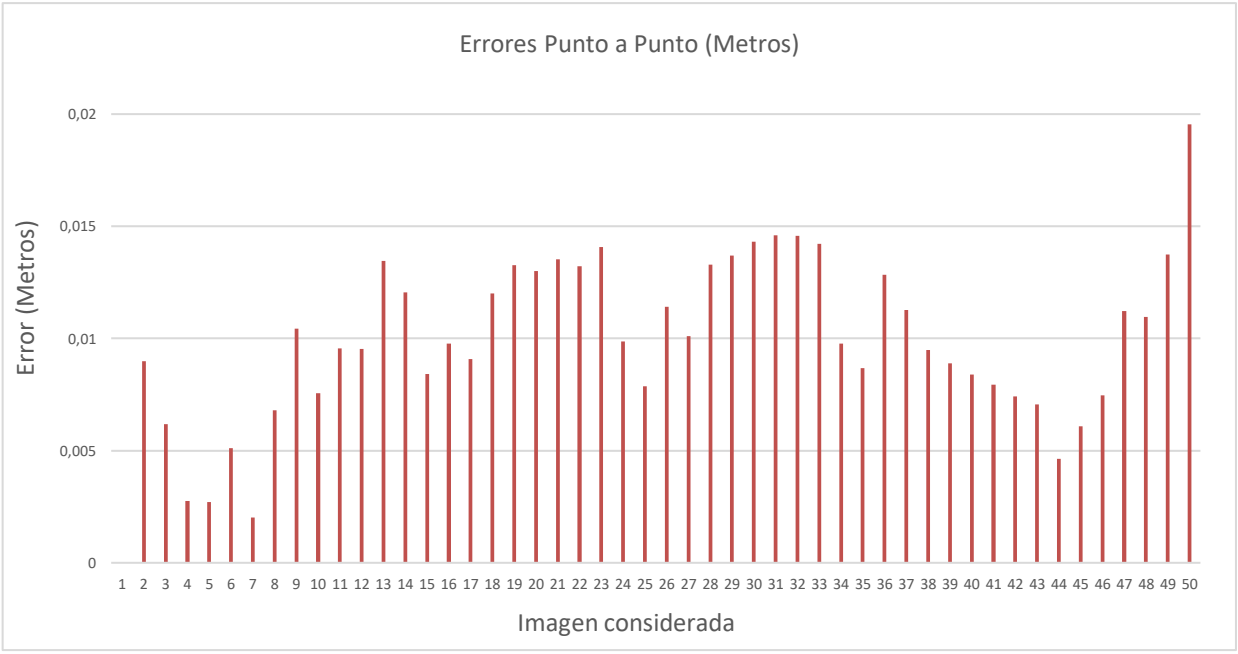
- Creación del mapa inicial con 50 imágenes y profundidad del plano x-y de 1,5 metros. Esta solución proporciona un error acumulado de 0,4928 metros y un error promedio de 0,0099 metros.
- Creación del mapa inicial con 20 imágenes y profundidad del plano x-y de 1,75 metros. Esta solución proporciona un error acumulado de 0,2582 metros y un error promedio de 0,0129 metros.

Experimentalmente se comprobó que utilizar 20 imágenes no proporcionaba información suficiente sobre el entorno, así que se optó por utilizar únicamente la inicialización con 50 imágenes.

En la figura 4.3.1 se muestra una comparativa entre la trayectoria obtenida a través del optimizador encargado de crear el mapa inicial y la trayectoria real descrita por el robot móvil.



(a)



(b)

Figura 4.3.1 Resultado obtenido tras la creación del mapa inicial con 50 imágenes y con una profundidad del plano x-y para la inicialización de los puntos tridimensionales de 1,5 metros. En (a) la línea roja representa la trayectoria estimada y la línea azul se corresponde con la trayectoria real. En (b) se observa una gráfica que nos permite observar el error de estimación por imagen.

De la figura 4.3.1 deducimos que se ha conseguido una inicialización en la que el error cometido al calcular la posición del robot no supera los 2 centímetros. Esta inicialización ha requerido un tiempo de 16,89 segundos. Podríamos crear un mapa del entorno en menos tiempo si para acometer dicha tarea empleásemos un menor número de imágenes, pero de las tablas 4.3.1, 4.3.2, 4.3.3 y 4.3.4 deducimos que tendríamos que sacrificar precisión para disminuir el tiempo de ejecución.

Nuevamente hay que alcanzar una solución de compromiso. Se seleccionaría una inicialización u otra en función de las especificaciones del sistema en cuanto a tiempo de ejecución y error asumible. Dado que este proyecto no tiene fines comerciales, se selecciona la inicialización que proporciona los resultados más precisos.

4.4 Análisis del método de obtención de *keyframes*

Como ya mencionamos anteriormente, el método de triangulación dependía en gran medida de las correspondencias 2D-2D obtenidas y, por consiguiente, de los *keyframes* considerados. En el apartado 3.2 de este documento llegamos a la conclusión de que si se consideraban todas las imágenes capturadas por la cámara a lo largo del movimiento del robot la incertidumbre en la reconstrucción de puntos tridimensionales del entorno era demasiado elevada.

Igualmente, si se dejan pasar un número fijo de imágenes entre un *keyframe* y el siguiente, no se estaría teniendo en cuenta que no podemos predecir el movimiento del robot. Supongamos que en un determinado instante el robot tiene que detenerse ante alguna eventualidad. En este caso el sistema estaría recibiendo la misma imagen durante varios instantes de tiempo. Si tenemos la mala suerte de que el número de imágenes consideradas entre un keyframe y el siguiente es inferior a la cantidad de imágenes que se reciben por repetido, nuestro sistema trataría de aplicar el método de triangulación a correspondencias de características visuales de dos imágenes idénticas, lo que daría como resultado una profundidad infinita para los puntos tridimensionales. Esto último haría fallar el sistema de forma estrepitosa.

Llegamos a la conclusión de que el número de imágenes que se dejan pasar entre un *keframe* y el siguiente debe ser variable y calculado de forma inteligente.

Se procede a describir cómo deben ser cada uno de los elementos implicados en el proceso de selección inteligente de *keyframes* descrito en el apartado 3.2.

En primer lugar, recordemos que era necesario un vocabulario que nos permitiese establecer una correspondencia entre una determinada característica visual de una imagen y una palabra de dicho vocabulario. Este vocabulario se creó de forma offline con el conjunto de todas las imágenes que constituyen el *dataset freiburg1_xyz* y con la ayuda de la librería de C++ DBoW2[62].

El vocabulario ocupa un espacio en memoria y nos interesa que sea lo más compacto posible. Además, cuanto mayor sea el vocabulario, más tardará nuestro sistema en crear el histograma de aparición de palabras del vocabulario para cada una de las imágenes. Esto último retrasaría el proceso de obtención de un nuevo *keyframe*. No obstante, se requiere que el vocabulario sea lo suficientemente grande para generar histogramas con suficientes matices que nos permitan distinguir unas imágenes de otras.

Experimentalmente se ha llegado a una solución de compromiso en la que el número de palabras del vocabulario debe estar comprendido entre 1.000 y 10.000 palabras para que no ocupe demasiado espacio en memoria. En el caso particular de este proyecto se ha utilizado un vocabulario de 6.561 palabras. Este vocabulario no ocupa más de 500 kilobytes en memoria.

De la misma forma se establece que la puntuación resultante de comparar el histograma de aparición de palabras del vocabulario en el *keyframe* de referencia y el histograma de aparición de palabras del vocabulario en el candidato a ser el siguiente *keyframe* debe ser inferior a un umbral de 0,18.

Para el vocabulario y el umbral mencionados en los dos párrafos anteriores se consiguieron un total de 78 *keyframes* para una secuencia de 300 imágenes. Como estos valores no son múltiplos entre sí, concluimos que nuestro módulo de detección de *keyframes* consiguió dejar pasar un número variable de imágenes entre cada par de *keyframes*.

4.5 Análisis de la ejecución completa del algoritmo

Se procede a evaluar el algoritmo de odometría visual para comprobar si este es capaz de estimar la trayectoria descrita por el robot móvil. La ejecución del algoritmo se realizó para los valores de los parámetros indicados en la Tabla 4.5.1

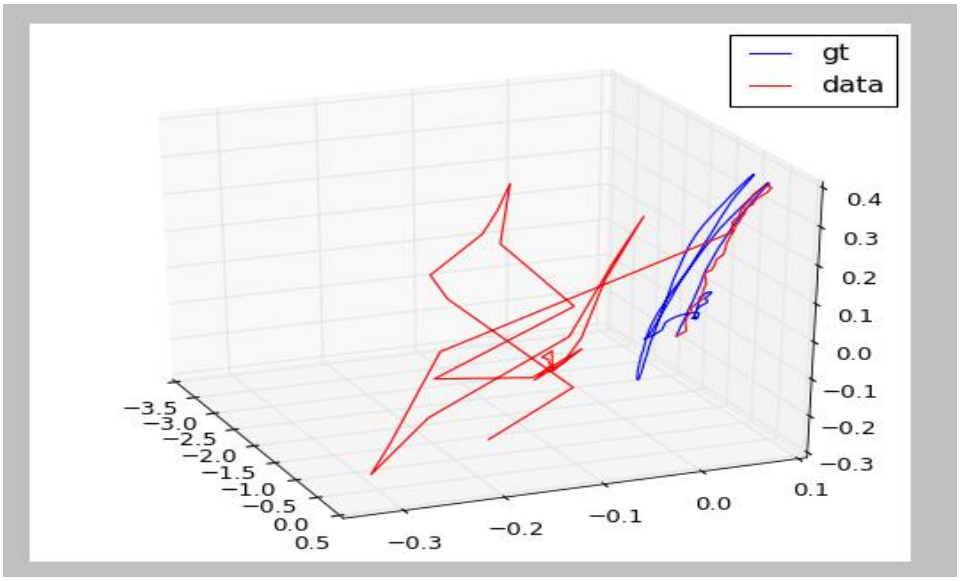
Nº de imágenes para la creación del mapa inicial	50
Profundidad a la que se inicializan los puntos 3D	1,5 metros
Probabilidad de éxito del algoritmo RANSAC	99,9 %
Distancia mínima usada por el algoritmo RANSAC para determinar si un punto es parte del modelo estimado	1
Umbral por debajo del cual debe permanecer la puntuación resultante de comparar el histograma de aparición de palabras del vocabulario en el *keyframe* de referencia con el histograma de aparición de palabras del vocabulario en la imagen candidata a ser *keyframe*	0,18
Nº de palabras del vocabulario visual	6.561
Nº de iteraciones realizadas por el optimizador g2o para la creación del mapa inicial del entorno	50
Umbral del primer filtro empleado en el emparejamiento robusto de características visuales	0,7
Nº de imágenes del dataset consideradas	300

Tabla 4.5.1: Valores asignados a las distintas variables del algoritmo de odometría visual propuesto.

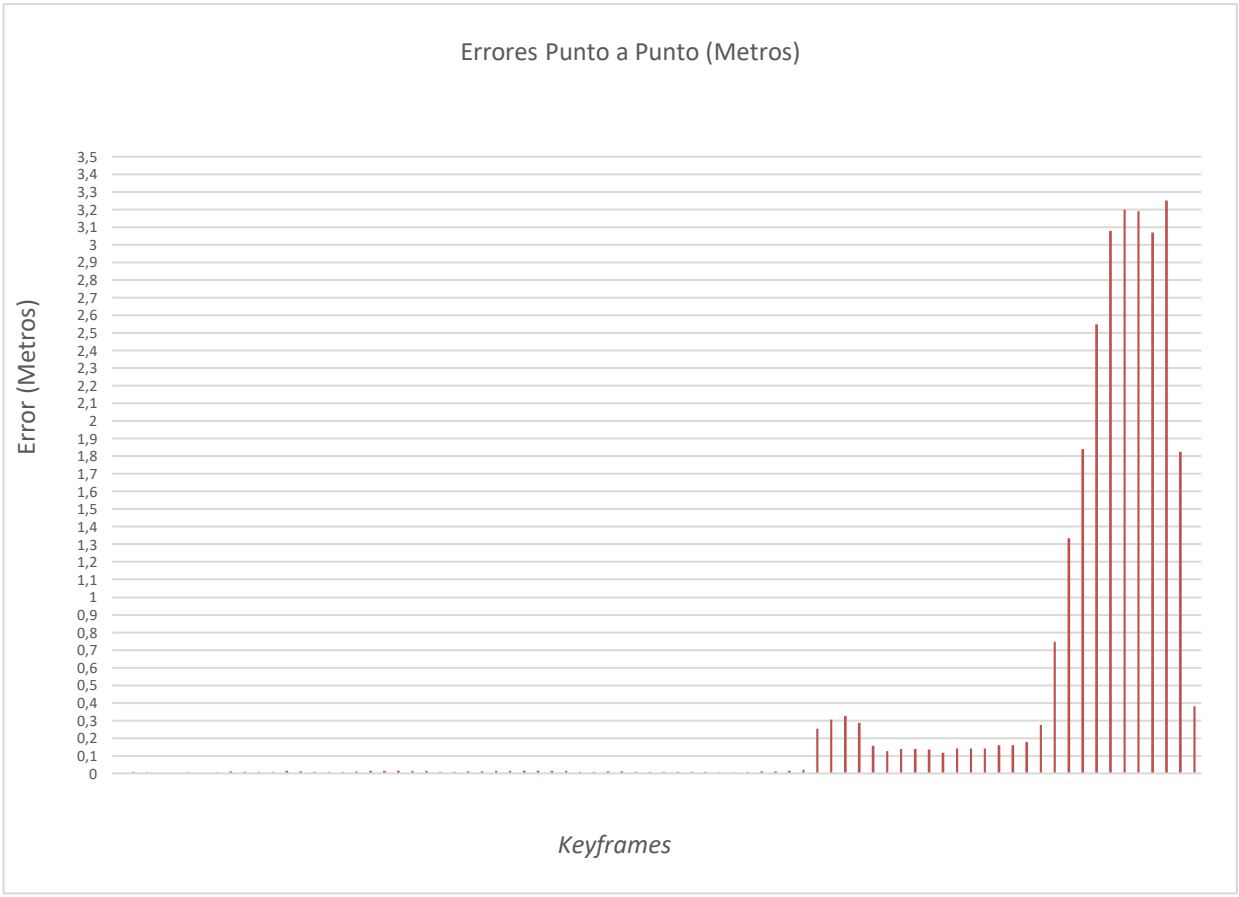
Los resultados de esta primera ejecución serían los que se muestran en la figura 4.5.1. Podemos apreciar claramente un problema inherente a la odometría visual que está directamente relacionado con la incertidumbre que se introduce a la hora de estimar cada nueva posición. Esto provoca lo que denominamos "salto de odometría" que hace que la trayectoria estimada aparezca desplazada con respecto a la trayectoria real a pesar de seguir la misma dirección.

Para comprobar si podíamos disminuir este error se propuso, como módulo adicional para nuestro algoritmo de odometría visual, un proceso para la optimización local de la trayectoria estimada. Este método consiste en considerar un número determinado de *keyframes* anteriores a la última posición estimada y utilizarlos junto con esta última posición y el mapa tridimensional del entorno para construir un optimizador similar al descrito en el apartado 3.1 de este documento. Evidentemente el hecho de añadir este módulo adicional supone aumentar ligeramente el tiempo de ejecución de nuestro algoritmo.

Los resultados obtenidos de aplicar el algoritmo con este nuevo módulo de optimización y volviendo a utilizar las mismas variables que en la Tabla 4.5.1 se muestran en la figura 4.5.2.

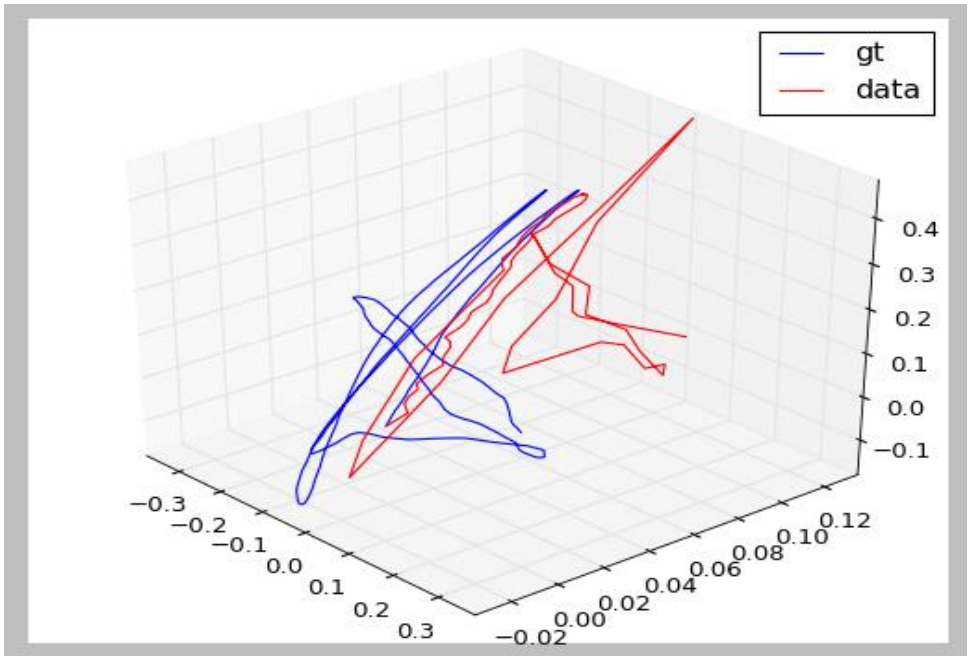


(a)

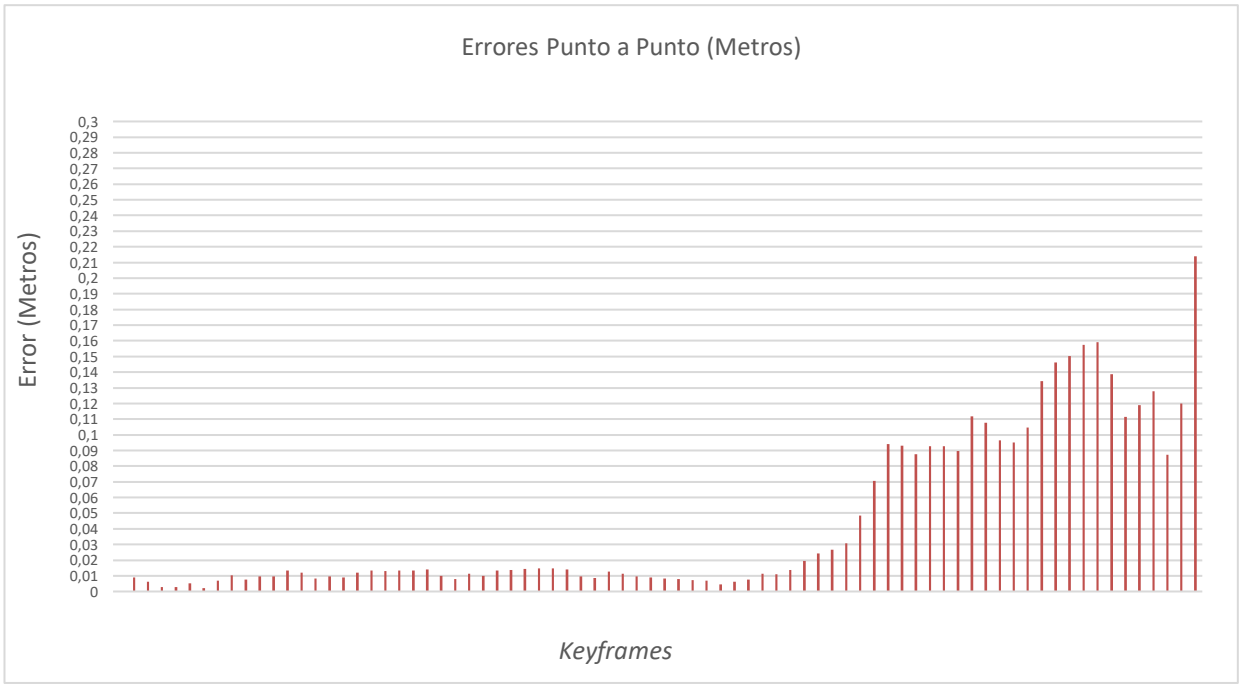


(b)

Figura 4.5.1 Resultados de aplicar el algoritmo de odometría visual sin optimización local. En (a) se muestra una comparación entre la trayectoria real (línea azul) y la trayectoria estimada (línea roja). En (b) se muestra el error cometido en cada *keyframe*.



(a)



(b)

Figura 4.5.2 Resultados de aplicar el algoritmo de odometría visual con optimización local. En (a) se muestra una comparación entre la trayectoria real (línea azul) y la trayectoria estimada (línea roja). En (b) se muestra el error cometido en cada *keyframe*.

Tras aplicar el optimizador local se observa como el efecto introducido por el denominado "salto de odometría" disminuye drásticamente. El error máximo que se cometía empleando el algoritmo sin optimización local era de 3,25 metros, mientras que ahora, al aplicar el mismo algoritmo con optimización local, se obtiene un error máximo de 21,39 centímetros.

4.6 Análisis del coste temporal

Para una secuencia de 300 imágenes del dataset *freiburg1_xyz* se analiza el tiempo invertido en cada uno de los módulos de nuestro algoritmo.

En primer lugar, el algoritmo crea un mapa inicial del entorno. En esta tarea se invirtieron 16,8969 segundos. Posteriormente comienza un bucle en el que en cada iteración se repiten las siguientes tareas:

- Detección y descripción de características visuales de la imagen actual.
- Determinar si la imagen actual es un fotograma clave.

Llegados a este último punto hay dos posibilidades: que la imagen actual no sea un fotograma clave, en cuyo caso se pasa a la siguiente iteración del algoritmo, o que la imagen actual sea un fotograma clave. En este caso el algoritmo continuaría en la misma iteración y pasaría a realizar las siguientes tareas:

- Emparejamiento robusto de características visuales entre la imagen actual y la imagen de referencia.
- Estimación de la nueva posición del robot.
- Optimización local para refinar la nueva posición del robot.

Una vez concluidas estas tareas, el algoritmo pasaría a la siguiente iteración. El bucle descrito anteriormente se repite hasta que se agotan todas las imágenes que se le proporciona al algoritmo.

De todo esto concluimos que tenemos dos tipos de iteraciones: aquellas en las que se ha determinado que la imagen es un fotograma clave y aquellas en las que se sucede lo contrario.

Para cada una de estas posibilidades se calculan los tiempos de ejecución para una sola iteración.

En la tabla 4.6.1 se proporcionan los resultados correspondientes a una iteración en la se ha descartado la imagen candidata a ser *keyframe*.

En la tabla 4.6.2 se muestran los resultados para una iteración en la que se determina que la imagen considerada es un fotograma clave.

Para cada una de las tareas que aparecen en estas tablas, el tiempo de ejecución para una iteración se ha calculado como el valor medio de la suma de todos los tiempos invertidos en realizar dicha tarea para cada una de las iteraciones.

En la figura 4.6.1 se muestra el porcentaje de tiempo empleado en cada una de estas tareas. Este porcentaje se ha calculado sobre el tiempo invertido en una iteración.

Pasemos a analizar la iteración en la que trabajamos con un fotograma clave, que es la realmente interesante. De la figura 4.6.1 deducimos que las actividades más costosas son, respectivamente, la optimización local encargada de refinar la nueva posición y el proceso de estimación de la nueva posición. Del total de los 71,1407 segundos empleados en la ejecución completa del algoritmo 34,4710 segundos se consumen en tareas de optimización local de la trayectoria y 11,3188 segundos son empleados en estimar la posición. Estas dos tareas suponen un 64,37 % sobre el total del tiempo que requiere la ejecución completa del algoritmo.

Iteración en la que la imagen considerada no es un fotograma clave	
Tareas realizadas	Coste de ejecución (segundos)
Detección y descripción de características visuales	0,0083
Determinar si la imagen es *keyframe*	0,0804

Tabla 4.6.1. Coste temporal de las tareas para una iteración en la que la imagen no es un fotograma clave.

Iteración en la que la imagen considerada es un fotograma clave	
Tareas realizadas	Coste de ejecución (segundos)
Detección y descripción de características visuales	0,0062
Determinar si la imagen es *keyframe*	0,0462
Emparejamiento robusto de características visuales	0,0197
Estimación de la nueva posición	0,1451
Optimización local para refinar la nueva posición	0,4419

Tabla 4.6.2: Coste temporal de las tareas para una iteración en la que la imagen es un fotograma clave.

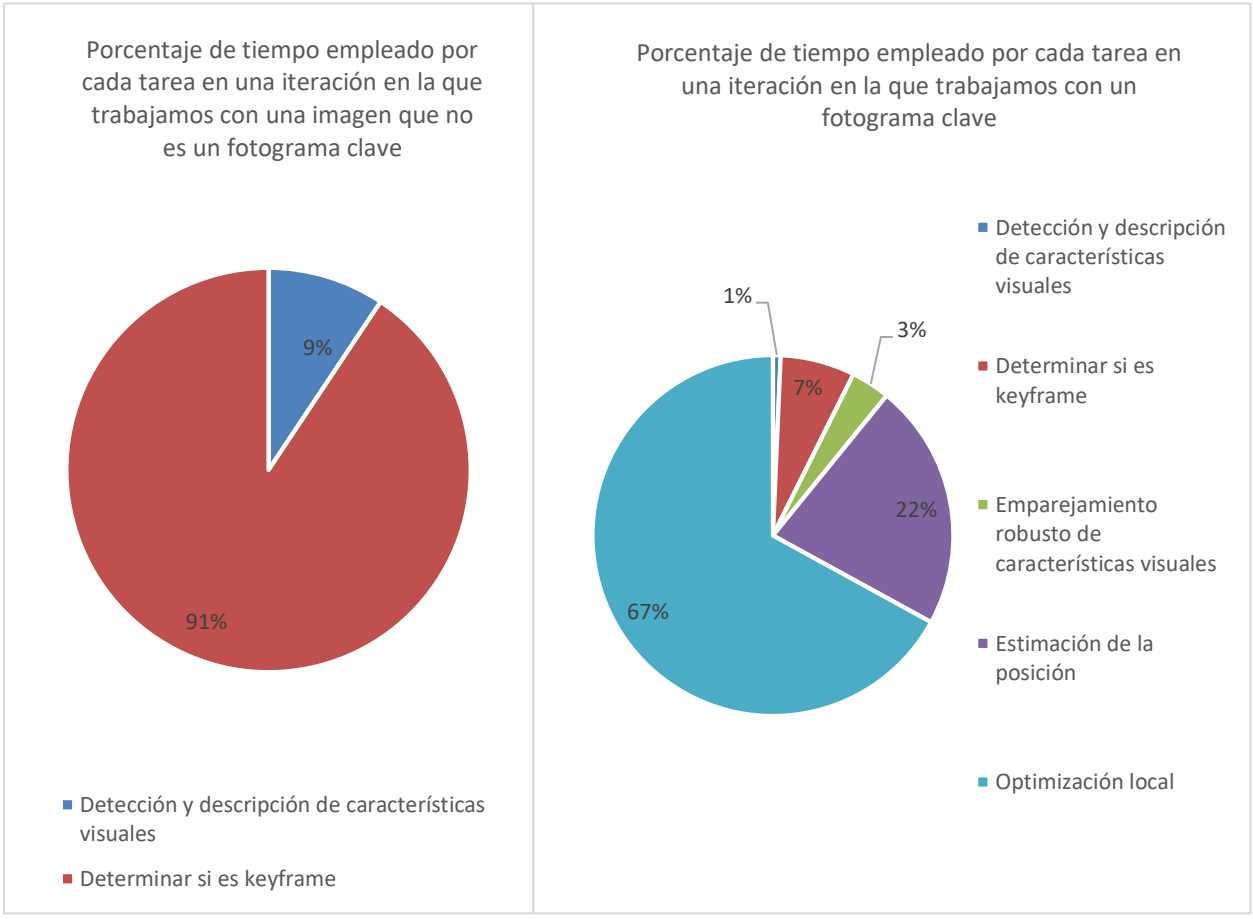


Figura 4.6.1 Porcentaje de tiempo empleado por cada una de las tareas según el tipo de iteración. A la izquierda estarían las tareas correspondientes a una iteración en la que la imagen no es un fotograma clave. A la derecha aparecen las tareas correspondientes a una iteración en la que se trabaja con un fotograma clave.

5 CONCLUSIONES Y TRABAJO FUTURO

5.1 Conclusiones

De los resultados obtenidos del apartado de experimentación y resultados podemos extraer las siguientes conclusiones:

- La creación del mapa inicial del entorno debe ser lo más precisa posible. Recordemos que los puntos 3D que se generan en la creación del mapa inicial del entorno son utilizados en el proceso de reconstrucción de la escala relativa. Si las coordenadas de estos puntos iniciales no se parecen demasiado a las coordenadas de los puntos del entorno a los que representan, nuestro sistema podría proporcionar una trayectoria errónea.
- A pesar de que nuestro algoritmo proporciona una trayectoria que sigue la misma dirección que la trayectoria real descrita por el robot móvil, acumula demasiado error en muy poco tiempo. Este error es del orden de metros lo cual es inaceptable. Esto último nos obligó a implementar un módulo adicional de optimización local para refinar la posición proporcionada por nuestro algoritmo en cada iteración. Tras implementar este módulo se consiguieron errores del orden de centímetros que si son aceptables para un sistema de odometría visual.
- El hecho de tener que invertir varios segundos en la creación del mapa inicial, así como la necesidad de implantar el módulo de optimización local hicieron que el tiempo de ejecución del algoritmo aumentase considerablemente. Esto hace que la implementación de este sistema en un robot móvil que trabaje en tiempo real esté muy limitada.
- El criterio de selección de *keyframes* tan solo tiene en cuenta información visual extraída de las imágenes. Esto implica que no se preocupa de la calidad de la estimación de la posición obtenida del módulo de estimación de la trayectoria. Podría darse el caso de que nuestro sistema de detección inteligente de *keyframes* considere adecuada una imagen que después no es útil para reconstruir puntos tridimensionales a través del método de triangulación. En este caso el algoritmo simplemente no actualiza la posición y pasa a la siguiente iteración. Aunque no supone un problema para el desempeño de nuestro sistema, introduce un coste temporal innecesario, ya que se ha invertido tiempo en estimar una posición no válida que pasa a ser descartada.
- El módulo de optimización local funcionó tal y como se esperaba, puesto que consiguió reducir errores del orden de metros a errores de tan solo 1 o 2 centímetros.
- Este sistema de odometría visual presenta problemas a la hora de estimar la posición en el caso de que se produzcan movimientos rotacionales puros. Este es uno de los problemas más difíciles con los que hay que lidiar a la hora de trabajar con un sensor monocular. Para poder conocer la posición de la cámara es necesario calcular previamente una serie de puntos tridimensionales a través del método de triangulación. Para que el método de triangulación tenga éxito se requiere que haya cierta disparidad entre las imágenes que se consideran para la aplicación de dicho método. Si se produjese un movimiento rotacional puro, la traslación sería inexistente y no habría disparidad alguna entre las imágenes consideradas.
- El algoritmo de odometría visual planteado tiene un coste económico muy bajo. El sensor Kinect utilizado para la captura de imágenes tiene un precio en mercado que oscila en torno a los 30 €, y las librerías de programación C++ utilizadas son la mayoría de código abierto. Se podría diseñar un algoritmo que utilice la información de más sensores y proporcione resultados más precisos, pero esto aumentaría el coste. Como este proyecto tan solo tenía fines meramente académicos, se buscó la solución más barata posible.

En el siguiente apartado se plantean ideas que pueden mejorar el desempeño de nuestro sistema de odometría visual monocular.

5.2 Trabajo futuro

Nuestro sistema de odometría visual es bastante básico. Se proponen una serie de mejoras que en teoría deberían mejorar el desempeño de nuestro algoritmo:

- El algoritmo de odometría visual se diseñó inicialmente de forma secuencial. De esta forma para que un módulo entrase en funcionamiento dependía de que terminase el anterior. Se podría traducir este algoritmo secuencial a un sistema distribuido en distintos hilos de ejecución capaces de realizar algunas de las operaciones en paralelo. Tan solo tendríamos que molestarnos en garantizar que los módulos no interactúen de forma anómala modificando simultáneamente variables globales, tales como las estructuras de datos encargadas de almacenar las posiciones de la cámara y los puntos 3D del entorno.
- Integración con ROS [68]. ROS define variables y funciones que pueden ser usados de forma concurrente por un robot. Al utilizar ROS mejoraríamos la portabilidad de nuestro sistema de odometría visual.
- Añadir un módulo de reconocimiento de lugares. Nuestro algoritmo de odometría visual va almacenando las correspondencias 2D-2D en una estructura de datos. Una vez finalizado el seguimiento de una determinada característica visual, al desaparecer esta del campo de visión del robot, nuestro algoritmo no tiene forma de volver a localizarla. Si en momentos posteriores de su desplazamiento el robot volviese a pasar por el mismo sitio del entorno, podría darse el caso de que detecte una característica visual antigua y se considere que es nueva, generando una nueva entrada en la estructura de datos encargada de almacenar el *tracking* de características visuales. A la hora de utilizar esta información para obtener un punto tridimensional a través del método de triangulación estaríamos obteniendo un puto ya existente y lo tendríamos almacenado por duplicado sin saberlo. Teniendo en cuenta que esto puede repetirse para un gran número de características visuales estaríamos ocupando demasiado espacio en memoria con información duplicada inútil. Una solución exitosa al problema de reconocimiento de lugares o *place recognition* se puede consultar en [69].
- Fusionar la información de nuestro sensor con la información de otros sensores ya existentes en el robot en el que se vaya a implementar nuestro sistema de odometría visual. De esta forma deberíamos de obtener una estimación de la posición más precisa.
- Añadir al proceso de estimación de la posición la información que proporciona la matriz de Homografía tal y como se describe en [70]. De esta forma en cada iteración del algoritmo tendríamos dos alternativas diferentes para calcular la posición del robot: la que proporciona la matriz Esencial y la que proporciona la matriz de Homografía.

BIBLIOGRAFÍA

[1] https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping

[2] Christopher G Harris and JM Pike. 3d positional integration from image sequences. *Image and Vision Computing*, 6(2):87–90, 1988.

[3] H Christopher Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Readings in Computer Vision: Issues, Problems, Principles, and Paradigms, MA Fischler and O. Firschein, eds*, pages 61–62, 1987.

[4] Hans P. Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. Technical report, DTIC Document,1980.

[5] Hans P. Moravec. Towards automatic visual obstacle avoidance. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence - Volume 2,* IJCAI'77, pages 584–584, San Francisco, CA, USA, 1977. Morgan Kaufmann Publishers Inc.

[6] Simon Lacroix, Anthony Mallet, Raja Chatila, and Laurent Gallo. Rover self localization in planetary-like environments. In *Artificial Intelligence, Robotics and Automation in Space*, volume 440, page 433, 1999.

[7] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on,* volume 1, pages I–652. IEEE, 2004.

[8] David Nistér, Oleg Naroditsky, and James Bergen. *Visual odometry for ground vehicle applications. Journal of Field Robotics*, 23(1):3–20, 2006.

[9] Larry Matthies and Steven A Shafer. Error modeling in stereo navigation. *Robotics and Automation, IEEE Journal of*, 3(3):239–248, 1987.

[10] Clark F Olson, Larry H Matthies, Marcel Schoppers, and Mark W Maimone. Robust stereo ego-motion for long distance navigation. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 2, pages 453–458. IEEE, 2000.

[11] Yang Cheng, Mark Maimone, and Larry Matthies. Visual odometry on the mars exploration rovers. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, volume 1, pages 903–910. IEEE, 2005.

[12] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[13] http://gro.usal.es/web/thesis/tesisJFRA.pdf

[14] Annalisa Milella and Roland Siegwart. Stereo-based ego-motion estimation using pixel tracking and iterative closest point. In *Computer Vision Systems, 2006 ICVS'06. IEEE International Conference on*, pages 21–21. IEEE, 2006.

[15] Kurt Konolige. Small vision systems: Hardware and implementation. In *Robotics Research*, pages 203–212. Springer, 1998.

[16] Jianbo Shi and Carlo Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE, 1994.

[17] Miguel Angel Garcia and Agusti Solanas. 3d simultaneous localization and modeling from stereo vision. In *Robotics and Automation,2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 1, pages 847–853. IEEE, 2004.

[18] David Nistér. Preemptive ransac for live structure and motion estimation. *Machine Vision and Applications*, 16(5):321–329, 2005.

[19] Peter Corke, Dennis Strelow, and Sanjiv Singh. Omnidirectional visual odometry for a planetary rover. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 4, pages 4007–4012. IEEE, 2004.

[20] J-P Tardif, Yanis Pavlidis, and Kostas Daniilidis. Monocular visual odometry in urban environments using an omnidirectional camera. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 2531–2538. IEEE, 2008.

[21] Davide Scaramuzza and Roland Siegwart. Appearance-guided monocular omnidirectional visual odometry for outdoor ground vehicles. *Robotics, IEEE Transactions on*, 24(5):1015–1026, 2008.

[22] Jason Campbell, Rahul Sukthankar, Illah Nourbakhsh, and Aroon Pahwa. A robust visual odometry and precipice detection system using consumer-grade monocular vision. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 3421–3427. IEEE, 2005.

[23] David Nistér. An efficient solution to the five-point relative pose problem. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(6):756–770, 2004.

[24] Maxime Lhuillier. Automatic scene structure and camera motion using a catadioptric system. *Computer Vision and Image Understanding*, 109(2):186–203, 2008.

[25] Etienne Mouragnon, Maxime Lhuillier, Michel Dhome, Fabien Dekeyser, and Patrick Sayd. Real time localization and 3d reconstruction. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference o*n, volume 1, pages 363–370. IEEE, 2006.

[26] DavidG. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[27] Michael J Milford and Gordon F Wyeth. Single camera vision-only slam on a suburban road network. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 3684–3689. IEEE, 2008.

[28] https://en.wikipedia.org/wiki/Bayer_filter

[29] https://en.wikipedia.org/wiki/CMOS

[30] https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

[31] https://en.wikipedia.org/wiki/Pinhole_camera_model

[32] Chris Harris and Mike Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.

[33] J. Shi and C. Tomasi. Good features to track. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.

[34] Edward Rosten and Tom Drummond. Machine learning for highspeed corner detection. In *Computer Vision–ECCV 2006*, pages 430–443. Springer, 2006.

[35] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer Vision–ECCV 2006*, pages 404–417. Springer, 2006.

[36] Motilal Agrawal, Kurt Konolige, and Morten Rufus Blas. Censure: Center surround extremas for realtime feature detection and matching. In *ECCV (4)*, pages 102–115, 2008.

[37] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry [tutorial]. *IEEE Robot. Automat. Mag.*, 18(4):80–92, 2011.

[38] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *ECCV (4),* pages 778–792, 2010.

[39] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary R. Bradski. Orb: An efficient alternative to sift or surf. In *ICCV*, pages 2564–2571, 2011.

[40] Stefan Leutenegger, Margarita Chli, and Roland Siegwart. Brisk: Binary robust invariant scalable keypoints. In *ICCV*, pages 2548–2555, 2011.

[41] Alexandre Alahi, Raphael Ortiz, and Pierre Vandergheynst. Freak: Fast retina keypoint. In *CVPR*, pages 510–517, 2012.

[42] https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[43] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09*, pages 331–340. INSTICC Press, 2009.

[44] Jianbo Shi and Carlo Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE, 1994.

[45] Jean-Yves Bouguet. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. *Intel Corporation*, 5, 2001.

[46] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[47] https://en.wikipedia.org/wiki/Perspective-n-Point

[48] Richard I. Hartley and Peter F. Sturm. Triangulation. *Computer Vision and Image Understanding*, 68(2):146–157, 1997.

[49] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment - a modern synthesis. In *Workshop on Vision Algorithms*, pages 298–372, 1999.

[50] https://en.wikipedia.org/wiki/Levenberg%E2%80%93Marquardt_algorithm

[51] https://en.wikipedia.org/wiki/Cholesky_decomposition

[52] https://opencv.org/about/

[53] https://github.com/RainerKuemmerle/g2o

[54] https://es.wikipedia.org/wiki/Teor%C3%ADa_de_grafos

[55] http://pointclouds.org/

[56] https://en.wikipedia.org/wiki/Hamming_distance

[57] Richard Szeliski and Sing Bing Kang. Recovering 3D shape and motion from image streams using nonlinear least squares. *Journal of Visual Communication and Image Representation*, 5(1), pages 10-28, 1994.

[58] http://www.cs.ubc.ca/labs/lci/papers/docs2006/lowe_gordon.pdf

[59] https://en.wikipedia.org/wiki/Bag-of-words_model_in_computer_vision

[60] https://en.wikipedia.org/wiki/K-means_clustering

[61] http://doriangalvez.com/papers/GalvezTRO12.pdf

[62] https://github.com/dorian3d/DBoW2

[63] https://en.wikipedia.org/wiki/Eight-point_algorithm

[64] https://es.wikipedia.org/wiki/Descomposici%C3%B3n_en_valores_singulares

[65] R. Hartley, R. Gupta, and T. Chang, Stereo from uncalibrated cameras, in Proceedings, IEEE Conference on Computer Vision and Pattern Recognition, 1992, pp. 761–764.

[66] J. Sturm, W. Burgard and D. Cremers, Evaluating Egomotion and Structure-from-Motion Approaches Using the TUM RGB-D Benchmark. In *Proc. of the Workshop on Color-Depth Camera Fusion in Robotics at the IEEE/RJS International Conference on Intelligent Robot Systems (IROS),* 2012.

[67] 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),* 2012.

[68] https://www.ros.org/

[69] David Nistér and Henrik Stewénius. Scalable recognition with a vocabulary tree. In *CVPR (2)*, pages 2161–2168, 2006.

[70] Raúl Mur-Artal, J. M. M. Montiel and Juan D. Tardós. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147-1163, 2015. (2015 IEEE Transactions on Robotics Best Paper Award).