Trabajo Fin de Grado en Ingeniería Informática

Inicialización del estado para un sistema monocular inercial

State initialization for monocular inertial systems

Autor

LAURA OLIVA MAZA

Director

JAVIER CIVERA SANCHO

Escuela de Ingeniería y Arquitectura

2017-2018

Inicialización del estado para un sistema monocular inercial

Resumen

La visión por computador es uno de los campos tecnológicos con mayor actividad. Esto es debido a la gran cantidad de aplicaciones potenciales relacionadas con ella. Dentro de la visión por computador se encuentra la visión 3D cuyo objetivo es generar una reconstrucción 3D de una escena a partir de imágenes 2D de la misma. La visión 3D tiene aplicaciones, por ejemplo, dentro de la robótica y la realidad aumentada.

Hoy en día, para generar un mapa en 3D se utilizan algoritmos de estimación. Estos algoritmos reciben un estado y unas medidas, y devuelven un estado actualizado. Para obtener estas medidas lo más habitual es utilizar un sensor visual-inercial. Este sensor está compuesto por una cámara y una unidad inercial (IMU).

Uno de los problemas de los algoritmos de estimación es que, aunque son bastante robustos, a la hora de inicializar necesitan una semilla inicial precisa. La generación de semillas iniciales ó inicialización del estado visual-inercial es un problema no resuelto.

En este trabajo de fin de grado se ha estudiado, implementado y evaluado un algoritmo de inicialización del estado visual-inercial. Para la evaluación del algoritmo se ha utilizado el dataset EuRoC y se han diseñado diferentes experimentos para evaluar cómo afectan los diferentes parámetros a su precisión y robustez.

A partir de los resultados obtenidos en los experimentos se han escogido los parámetros con los que se obtienen mejores resultados, permitiendo ejecutar el algoritmo en tiempo real y obteniendo elevadas tasas de robustez y alta precisión.

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

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

D./Dª. Laura Oliva Maza,

con nº de DNI 73130322M en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Inicialización del estado para un sistema monocular inercial,

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

Zaragoza, 19 de Junio del 2018

Fdo: Laura Oliva Maza

Agradecimientos

Lo primero de todo querría agradecer a Javier Civera por haberme dado la oportunidad de realizar este trabajo y por la ayuda y apoyo que me ha dado.

También quería dar las gracias a mi familia y amigos por ayudarme a relajarme y aguantarme en los momentos más desesperantes.

Índice

Lista de Figuras

1 Introducción

El objetivo general de la visión por computador es conseguir que los ordenadores vean. Sin embargo, para las personas el proceso de ver involucra muchos aspectos. Por ejemplo: reconocer objetos, escenas, estimar profundidad o detectar movimiento o cambios. Por este motivo la visión por computador es un campo muy fragmentado. Cada uno de los sub-campos en los que está dividida, aspira a replicar cada una de dichas capacidades. La visión 3D es uno de estos sub-campos cuyo objetivo es generar una reconstrucción 3D a partir de las imágenes obtenidas por una cámara.

Hoy en día, la visión por computador es uno de los campos tecnológicos con mayor interés debido a la gran cantidad de aplicaciones potenciales. Por ejemplo, dentro de la visión 3D están la robótica y la realidad aumentada.

Uno de las aplicaciones más relevantes de la visión por computador es la robótica. Los robots necesitan conocer el entorno en el que se encuentran para poder interactuar con él o moverse por él sin colisionar. Para ello deben ser capaces de crear con bastante precisión un mapa 3D del entorno en el que se encuentran. En la **figura 1.1(a)** se puede observar el mapa estimado por una cámara que se mueve por una habitación de la que no tiene conocimiento previo.

Otra de las aplicaciones en las que se utiliza visión por computador es la realidad aumentada. La realidad aumentada permite integrar objetos virtuales en la vida real. Para lograr esto es necesario conocer la estructura de la habitación o el entorno en el que se encuentra el dispositivo y el movimiento de la cámara (**Figura 1.1(b)**). Esto se consigue mediante la visión 3D.

Para poder crear un mapa del entorno en el que se encuentra un robot, o un dispositivo en el que se está ejecutando una aplicación de realidad aumentada, una de las configuraciones más habituales es que éste disponga de un sensor visual-inercial. Un sensor visual-inercial está compuesto por una cámara (sensor visual), un acelerómetro y un giroscopio (IMU). Para estimar una reconstrucción 3D, si se utilizan únicamente los datos obtenidos por el sensor visual no se tiene la escala real de la escena. Si se utilizan solo los datos del sensor inercial la estimación tiene deriva puesto que las mediciones son propioceptivas y no se refieren a elementos externos de la escena. La principal ventaja de utilizar los datos obtenidos por ambos sensores es que es posible obtener la escala real de la escena y la trayectoria del robot de manera consistente y sin deriva. Esta combinación

(a) Mapa 3D del entorno de una cámara moviéndose por una habitación

(b) Realidad aumentada

Figura 1.1: Ejemplos de aplicaciones que utilizan visión por computador

sensorial tiene un tamaño pequeño por lo que se puede integrar en cualquier dispositivo. Por ejemplo, se puede integrar en los teléfonos móviles, permitiendo así el uso de aplicaciones de realidad aumentada en estos.

Actualmente, para crear este mapa, existen algoritmos de estimación bastante robustos que, dado un estado y unas medidas de los sensores, devuelven un estado actualizado. Aunque estos algoritmos son robustos para la estimación, tienen un problema en la inicialización, ya que necesitan una semilla inicial precisa del estado. Los sistemas de localización visual-inercial no son ni robustos ni precisos en los instantes iniciales de la estimación. Y uno de los principales motivos es la falta de algoritmos que proporcionen una buena semilla para la estimación posterior.

En este trabajo de fin de grado se va a trabajar con un algoritmo de inicialización del estado visual-inercial. Este algoritmo no necesita ningún tipo de conocimiento previo de la escena para determinar su estructura 3D.

El principal objetivo de este trabajo es implementar el algoritmo de inicialización del estado visual-inercial y evaluar cómo afectan los diferentes parámetros del algoritmo a su precisión y robustez. Como este algoritmo es relativamente nuevo (2014) [6] [3], algunos de los experimentos y evaluaciones realizadas en este trabajo no se pueden encontrar en la literatura sobre el tema.

Para lograr estos objetivos se han realizado las siguientes tareas:

- Estudio de la bibliografía. Lectura de los diferentes artículos que hablan sobre dicho algoritmo.
- Implementación del algoritmo de inicialización visual-inercial en C++.
- Estudio del dataset EuRoC sobre el que se realizarán las pruebas. Se utiliza este dataset ya que contiene "ground truth" de posición y orientación. Estos datos son muy útiles para comprobar los resultados obtenidos en los experimentos.
- Diseño de las pruebas con las que se evaluará el algoritmo.
- Implementación y ejecución de las pruebas sobre el dataset.
- Análisis de los resultados obtenidos comparando los datos obtenidos con los datos reales (*ground truth*).

2 Inicialización visual inercial

El algoritmo de inicialización del estado visual-inercial permite estimar el mapa 3D del entorno en el que se encuentra la cámara y los movimientos de la misma. De aquí en adelante se referirá al dispositivo que contiene la cámara y la IMU como “plataforma”.

Se va a suponer que la cámara observa simultáneamente N puntos de interés durante un corto intervalo de tiempo. Existen algoritmos automáticos para la detección y extracción de puntos en imágenes. En la **figura 2.1** se puede ver un ejemplo de los puntos de interés de una imagen. Sean $t_1, ..., t_n$ los tiempos de los intervalos en los que la cámara obtiene una imagen de esos puntos, n_i el número de imágenes durante las cuales se observan esos puntos y $r(t_j)$ la posición de la plataforma en el instante t_j. Se asume que $t_1 = 0$.

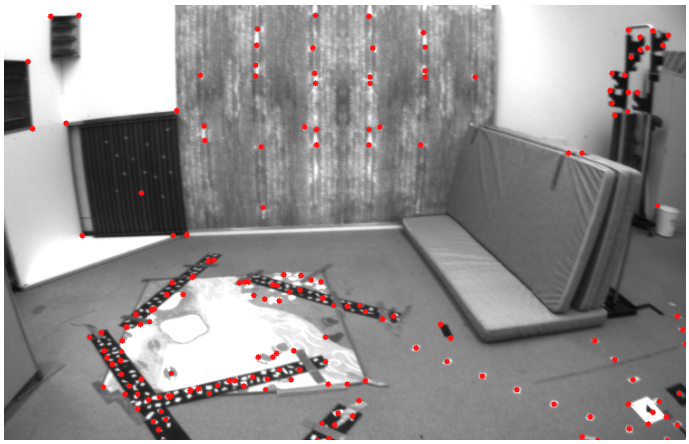


Figura 2.1: Puntos de interés en una imagen

A partir de la ecuación del movimiento uniformemente acelerado se puede obtener la posición de la plataforma en un instante determinado t_j.

$$r(t_j) = r(t_1) + Vt_j + \int_{t_0}^{t_j} \int_{t_0}^{\tau} a(\xi)\, d\xi\, d\tau \tag{2.1}$$

Donde $r(t_1)$ es la posición de la plataforma en el instante t_1, V es la velocidad de la plataforma en el instante t_1 y $a(\xi)$ es la aceleración de la plataforma en el instante ξ. El último término de la ecuación 2.1 es una doble integral que representa la aceleración de la plataforma. Esta doble integral se puede simplificar integrando por partes y se obtiene la siguiente ecuación:

$$r(t_j) = r(t_1) + Vt_j + \int_{t_0}^{t_j} (t_j - \tau)a(\tau)\, d\tau \tag{2.2}$$

El acelerómetro no mide la aceleración de la plataforma ($a(\tau)$ en la **ecuación 2.2**) sino que obtiene la aceleración inercial en el sistema de referencia local (sistema de referencia de la IMU) y la gravedad. Además, los datos medidos por este sensor tienen un sesgo, por lo que hay que sumarle una constante B (sesgo/*bias*) a la medida. En este trabajo se considera que el sesgo B es conocido. Teniendo en cuenta esto en la **ecuación 2.2** hay que sustituir $a(\tau)$ por $A_\tau^{inertial} - G_\tau + B$ quedando de la siguiente forma:

$$r(t_j) = r(t_1) + Vt_j + \int_{t_0}^{t_j} (t_j - \tau)(A_\tau^{inertial} - G_\tau + B)\, d\tau \tag{2.3}$$

Sabiendo que la gravedad es constante ($-G_\tau = -G\frac{t_j^2}{2}$) y que el sesgo ($B$) varía lentamente (es prácticamente constante para periodos cortos) estos dos términos se pueden sacar de la integral en la **ecuación 2.3**. Teniendo en cuenta esto, la integral quedaría $\int_{t_0}^{t_j}(t_j - \tau)(A_\tau^{inertial})\, d\tau - B$. Con esta integral se obtienen los datos del sensor inercial preintegrados y se sustituye en la ecuación 2.3 por S_j. Esta integral se obtiene numéricamente de la siguiente forma: $S_j = \int_{t_0}^{t_j}(t_j - \tau)(A_\tau^{inertial})\, d\tau + B \simeq \sum_k A_k^{inertial}\Delta t + B$. Haciendo estas sustituciones la ecuación 2.3 quedaría:

$$r(t_j) = r(t_1) + Vt_j - G\frac{t_j^2}{2} + S_j \tag{2.4}$$

Una vez obtenida la ecuación del movimiento de la plataforma (ecuación 2.4) se ha de obtener la posición de los puntos extraídos de la imagen. Siendo p_i la posición real del punto se puede calcular esta posición con la siguiente ecuación:

$$p^i = r(t_j) + \lambda_j^i \mu_j^i \tag{2.5}$$

Donde:

- $r(t_j)$ es la posición de la plataforma en el instante t_j.
- λ_j^i es la distancia al punto i en el instante t_j.
- μ_j^i es un vector normalizado que representa la dirección del punto i en el instante t_j.

La ecuación 2.5 en el instante t_1 sería $p^i = r(t_1) + \lambda_1^i \mu_1^i$. Como la posición real del punto (p^i) no varía se cumple la siguiente ecuación.

$$p^i = r(t_j) + \lambda_j^i \mu_j^i = r(t_1) + \lambda_1^i \mu_1^i \tag{2.6}$$

Despejando $r(t_1)$ de la ecuación 2.6 se obtiene: $r(t_1) = r(t_j) + \lambda_j^i \mu_j^i - \lambda_1^i \mu_1^i$. Si se sustituye $r(t_1)$ en la ecuación 2.4 y se despeja S_j esta ecuación quedaría:

$$S_j = \lambda_1^i \mu_1^i - V t_j - G\frac{t_j^2}{2} - \lambda_j^i \mu_j^i \tag{2.7}$$

La **figura 2.2** muestra las principales magnitudes que intervienen en las ecuaciones anteriores.

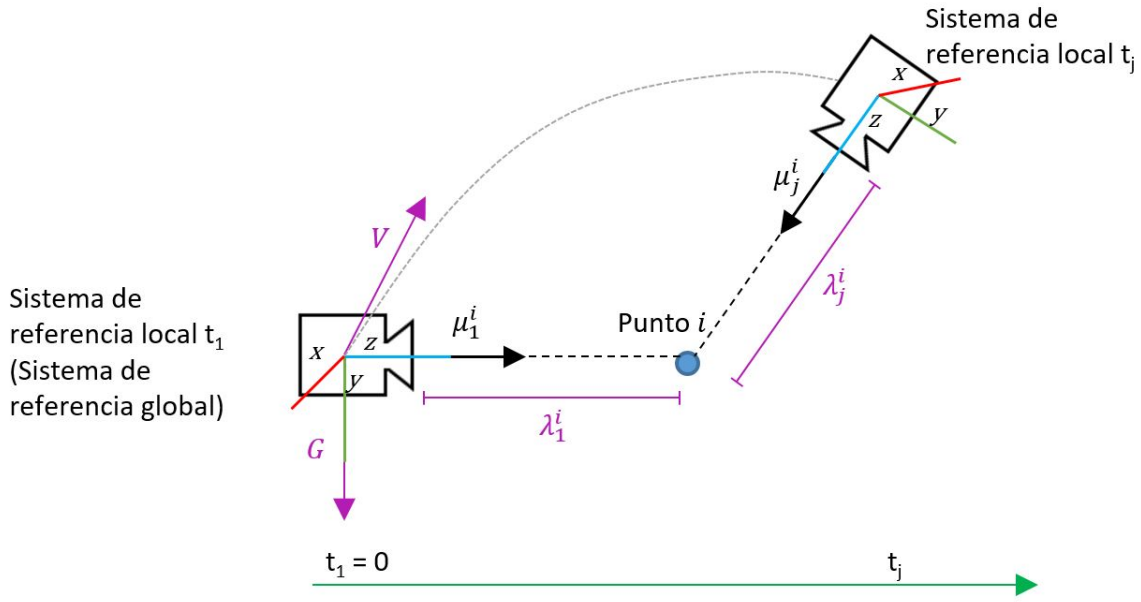


Figura 2.2: Magnitudes principales de la ecuación 2.7. Las variables que aparecen en morado son las incógnitas

Las incógnitas de la **ecuación 2.7** son el escalar λ_j^i y los vectores V y G, es decir, la distancia entre los puntos y la cámara, la velocidad inicial de la cámara y la gravedad. Los vectores μ_j^i están determinados por las medidas visuales y las medidas del giroscopio. El vector S_j está determinado por las medidas del acelerómetro y del giroscopio.

Para obtener el valor de las incógnitas se necesita crear un sistema de ecuaciones. Para ello se va a utilizar la **ecuación 2.7** a partir de la cual se obtendrán tres ecuaciones escalares por cada punto característico $i = 1, ..., N$ y por cada imagen capturada empezando por la segunda $j = 2, ..., n_i$. Se empieza desde la segunda imagen porque la primera imagen no proporciona información ya que cuando se obtiene esta imagen en el instante t_1 siempre se cumple la ecuación 2.7 ($S_j = \lambda_1^i \mu_1^i - V t_1 - G\frac{t_1^2}{2} - \lambda_1^i \mu_1^i \rightarrow S_j = -V t_j - G$, que es la integración de

las medidas inerciales). Con estas ecuaciones se puede construir un sistema lineal con $3(n_i - 1)N$ ecuaciones y $6 + Nn_i$ incógnitas. Una vez construida la matriz que contiene el sistema de ecuaciones, éste se puede resolver invirtiendo dicha matriz. La matriz construida quedaría de la siguiente forma:

$$AX = S \tag{2.8}$$

Donde:

- A es la matriz que contiene la dirección de cada punto en cada una de las imágenes, empezando por la segunda, y los instantes en los que han sido tomadas cada una de las imágenes.
- S es el vector que contiene los datos integrados.
- X es el vector que contiene las incógnitas del sistema de ecuaciones.

Estos vectores y la matriz se pueden representar de la siguiente forma:

$$A = \begin{bmatrix}
T_2 & S_2 & \mu_1^1 & 0_3 & 0_3 & -\mu_2^1 & 0_3 & 0_3 & 0_3 & 0_3 & 0_3 \\
T_2 & S_2 & 0_3 & \mu_1^2 & 0_3 & 0_3 & -\mu_2^2 & 0_3 & 0_3 & 0_3 & 0_3 \\
... & ... & ... & ... & ... & ... & ... & ... & ... & ... & ... \\
T_2 & S_2 & 0_3 & 0_3 & \mu_1^N & 0_3 & 0_3 & -\mu_2^N & 0_3 & 0_3 & 0_3 \\
... & ... & ... & ... & ... & ... & ... & ... & ... & ... & ... \\
T_{n_i} & S_{n_i} & \mu_1^1 & 0_3 & 0_3 & 0_3 & 0_3 & 0_3 & -\mu_{n_i}^1 & 0_3 & 0_3 \\
T_{n_i} & S_{n_i} & 0_3 & \mu_1^2 & 0_3 & 0_3 & 0_3 & 0_3 & 0_3 & -\mu_{n_i}^2 & 0_3 \\
... & ... & ... & ... & ... & ... & ... & ... & ... & ... & ... \\
T_{n_i} & S_{n_i} & 0_3 & 0_3 & \mu_1^N & 0_3 & 0_3 & 0_3 & 0_3 & 0_3 & -\mu_{n_i}^N
\end{bmatrix}$$

$$S = \begin{bmatrix} S_2^T & 0_3 & ... & 0_3 & S_3^T & 0_3 & ... & 0_3 & ... & S_{n_i}^T & 0_3 & ... & 0_3 \end{bmatrix}^T$$

$$X = \begin{bmatrix} G^T & V^T & \lambda_1^1 & ... & \lambda_1^N & ... & \lambda_{n_i}^1 & ... & \lambda_{n_i}^N \end{bmatrix}^T$$

Donde $T_j = \frac{t_j^2}{2} I_3$, $S_j = t_j^2 I_3$, I_3 es la matriz identidad de tamaño 3x3, 0_3 es la matriz de ceros de tamaño 3x1 y t_j es el instante en el que se ha obtenido la imagen j. Aunque la matriz A tiene gran tamaño, la mayoría de sus elementos son ceros. Cada fila de la matriz sólo contiene ocho elementos que no sean nulos. Teniendo en cuenta esto el número de elementos que son distintos de cero es $8 * (6 + Nn_i)$. La matriz contiene $(6 + Nn_i) * 3(n_i - 1)N$ elementos por lo que el ratio de elementos no nulos es $\frac{8}{3(n_i-1)N}$. Se puede observar que este ratio es muy pequeño por lo que la matriz es muy dispersa. Para resolver este sistema de ecuaciones se va a usar un método que optimice la resolución de matrices dispersas. En el siguiente apartado se hablará sobre el método usado para resolver este sistema.

Este sistema de ecuaciones sólo se construirá cuando el número de imágenes procesadas sea mayor que cinco [6] y cuando el número de puntos de interés comunes encontrados en las imágenes procesadas sea mayor que cinco. Si no se ha encontrado este número de puntos no se podrá construir esta matriz, considerando que el

sistema no se ha inicializado de forma exitosa. En [6] el número de puntos de interés necesarios para construir el sistema de ecuaciones es 1, pero se ha observado que los resultados son muy imprecisos con pocos puntos.

Los vectores de este sistema de ecuaciones deben estar en el mismo sistema de referencia para poder realizar las diferentes operaciones y comparaciones. Este sistema de referencia, que denominaremos sistema de referencia global, es el sistema de referencia local de la plataforma en el instante t_1. Asociaremos el sistema de referencia local al sistema de referencia de la IMU. Este sistema depende del tiempo ya que varía conforme la plataforma se traslada y rota. Como el sistema de referencia va variando, cada vez que los sensores midan nuevos datos (vectores) estos deben ser multiplicados por una matriz de rotación (R^{1j}) para que pasen de estar en el sistema de referencia local del instante en el que han sido obtenidos, a estar en el sistema de referencia global. Por este motivo, los vectores normalizados que representan la dirección de un punto en un instante (μ^i_j en la ecuación 2.7) se han calculado multiplicando las medidas obtenidas por la matriz de rotación:

- $\mu^i_j = R^{1j} R^{IC} \mu^{iCj}_j$
- $R^{1j} = \sum_1^j \omega \Delta t$

Donde R^{1j} es la matriz de rotación del sistema de referencia j al sistema de referencia global y R^{IC} es la matriz de rotación del sistema de referencia de la cámara al de la IMU.

Uno de los parámetros de este algoritmo es el número de imágenes. Este parámetro puede fijarse desde el principio. Si se fija, la cámara captura ese número de imágenes y luego se aplica el algoritmo sobre los datos recogidos por los sensores. Si este parámetro no se fija, cada vez que la cámara obtenga una imagen se aplicará el algoritmo sobre los nuevos datos medidos y los almacenados de las imágenes anteriores. Esto se realizará hasta llegar a un número máximo de imágenes establecido por defecto o a un número mínimo de puntos encontrados. Este parámetro del algoritmo es uno de los que se va a evaluar en las siguientes secciones. Con estos experimentos se busca obtener que es más óptimo: fijar el número de imágenes o tener un número de imágenes incremental.

Al sistema de ecuaciones de este algoritmo se le puede añadir una restricción no lineal, ya que se puede asumir que la aceleración de la gravedad es conocida *a priori*. Esta restricción sería $|G| = g$ y puede expresarse con la siguiente ecuación:

$$|IX|^2 = g^2 \tag{2.9}$$

Siendo $I = [I_3, 0_3, ..., 0_3]$. De esta forma se asegura que la norma de las tres primeras variables del vector X, que representan las 3 coordenadas de la variable G, correspondan a la aceleración de la gravedad ($g = 9{,}81$). Si se aplica esta restricción, para resolver el sistema será necesario encontrar un vector X que satisfaga tanto la **ecuación 2.7** como la **ecuación 2.9**.

En los apartados posteriores se discutirá sobre las ventajas y desventajas de utilizar esta restricción no lineal. También se hablará de los diferentes algoritmos que existen para detectar y extraer los puntos característicos

de una imagen y de cómo saber si un punto está presente en varias imágenes. Además, también se explicarán las diferencias en los resultados a la hora de establecer el número de imágenes, tanto si se es fijo como si es incremental. A continuación, se va a explicar las diferentes herramientas y métodos utilizados para implementar este algoritmo.

3 Implementación

El algoritmo descrito en la sección anterior se ha implementado en C++ para su evaluación. Para dicha implementación se han utilizado las librerías Eigen, IPOPT y OpenCV. A continuación, se explica el uso de cada una de las librerías.

3.1. C++

El algoritmo se ha implementado en C++ para poder medir los tiempos en las diferentes configuraciones. De esta forma sabiendo estos tiempos se puede saber cuáles son las opciones/parámetros que más se adecuan al algoritmo.

Para implementar este algoritmo se ha creado una clase *Initialization* que contiene todas las funciones necesarias para poder ejecutar el algoritmo. La función principal de esta clase, *initializate*, devuelve la nube de puntos creada (puntos 3D), las coordenadas de estos puntos en la primera y en la última imagen procesada (puntos 2D) y la posición y orientación actual de la plataforma. Estos datos solo se devuelven en el caso en el que el algoritmo haya terminado con éxito, es decir, si se han encontrado más de cinco puntos en las primeras cinco imágenes.

La clase *Initialization* tiene un atributo de tipo *Cam_Imu*. Este atributo almacena toda la información relacionada con los parámetros extrínsecos e intrínsecos de la cámara y de la IMU.

Además, la clase *Initialization* utiliza la clase *Pose* y la clase *Solver_Constraint*. La clase *Pose* representa la posición y orientación de la plataforma. La clase *Solver_Constraint* se utiliza para resolver el sistema con la restricción no lineal.

En la **figura 3.1** se puede observar el diagrama de clases de esta implementación.

3.2. Eigen

Como se puede observar en la **figura 3.1**, la mayoría de las funciones de este algoritmo utilizan matrices y vectores. Por esta razón, se ha decidido usar la librería Eigen [**] que contiene la clase *Matrix* y *Vector* y permite realizar las operaciones necesarias entre ambas clases.

[**] http://eigen.tuxfamily.org/index.php?title=Main_Page

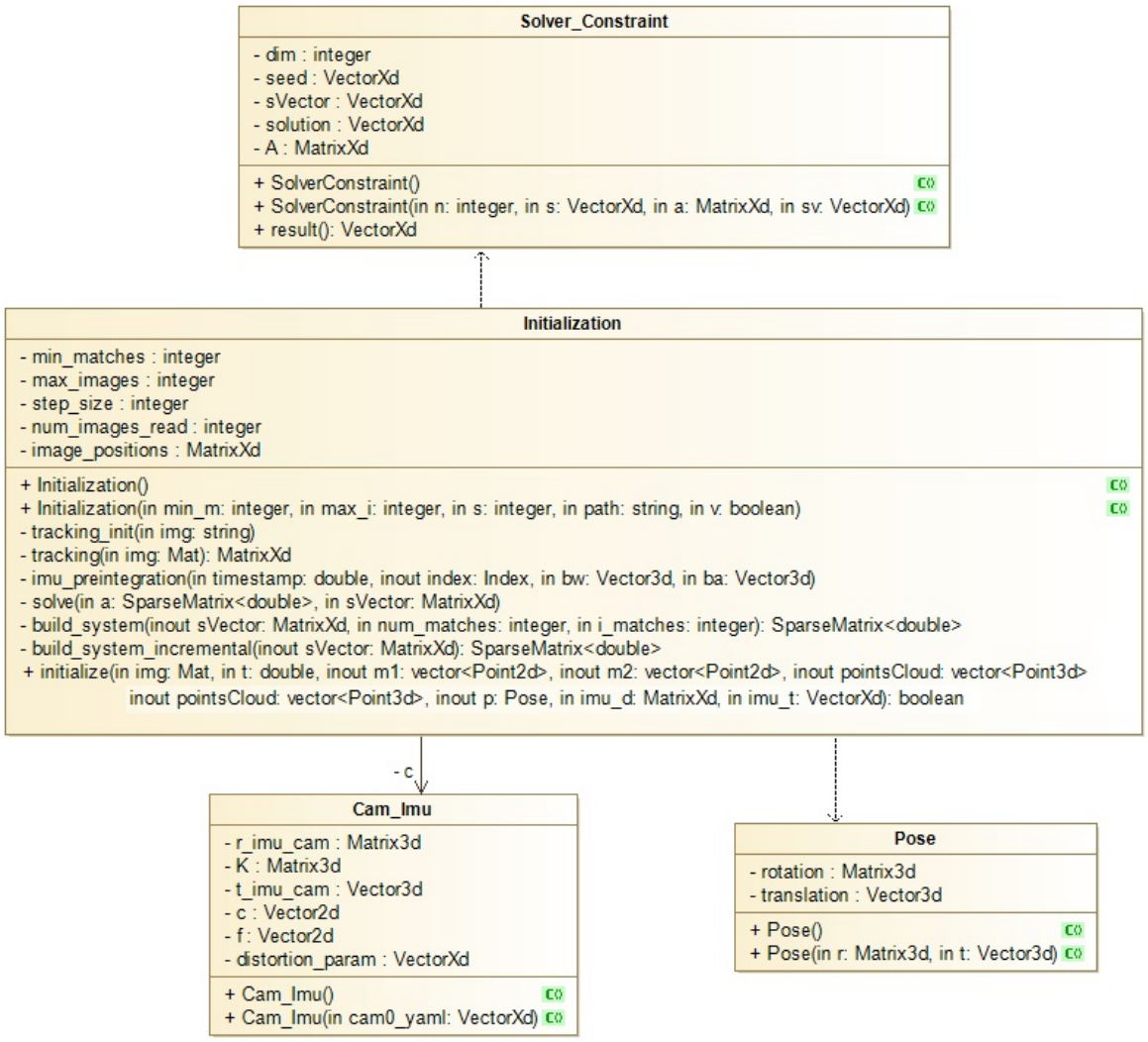


Figura 3.1: Diagrama de clases

Otra razón por la que se decidió utilizar la librería Eigen es por la gran variedad de métodos que tiene para resolver sistemas lineales de ecuaciones. El primer método que se probó para resolver el sistema de ecuaciones fue el de la descomposición Jacobi SVD. Este método descompone la matriz en tres matrices (S, V y U) y obtiene los valores y los vectores singulares. Pero, como ya se ha mencionado antes, la matriz que contiene el sistema de ecuaciones es muy dispersa. La librería Eigen tiene diferentes métodos, tanto directos como iterativos, para optimizar la resolución de matrices dispersas. Tras evaluar las pre-condiciones de cada uno de estos métodos se decidió evaluar el método de factorización QR (directo) y el método de los mínimos cuadrados del gradiente conjugado (Least Squares Conjugate Gradient, iterativo).

Jacobi SVD

Jacobi SVD no es un método optimizado para resolver matrices dispersas. Se decidió utilizar este método para poder evaluar la ganancia de los otros dos métodos. Este método descompone la matriz A de tamaño nxp en USV^* donde U es un vector unitario de tamaño nxn, V es un vector unitario de pxp y S es una matriz nxp que contiene en su diagonal los valores singulares de A y el resto de elementos son nulos. Los vectores U y V son conocidos como los vectores singulares. A partir de los valores y vectores singulares se puede resolver el sistema de ecuaciones.

Factorización QR

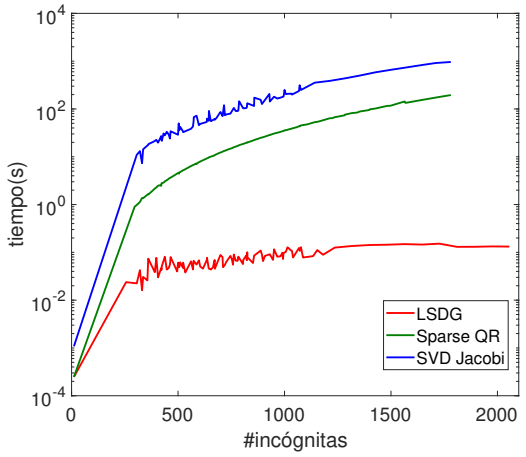
Factorización QR (left-looking rank-revealing QR decomposition) de matrices dispersas consiste en descomponer la matriz A en $AP = QR$ donde P es la matriz A permutada por columnas, Q es una matriz ortogonal representada como el producto de la transformación de Householder (Householder reflections, transformación lineal del espacio que consiste en una reflexión pura con respecto a un plano) y R es una matriz dispersa triangular o trapezoidal. Una vez realizada la descomposición se resuelve el sistema.

Mínimos Cuadrados del Gradiente Conjugado

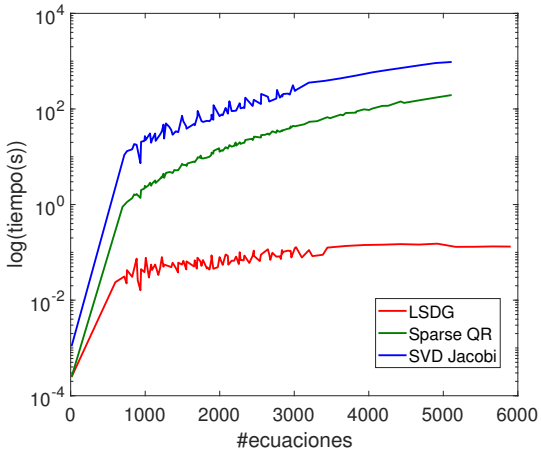
El método de Mínimos Cuadrados del Gradiente Conjugado (Least Squares Conjugate Gradient) resuelve el problema de mínimos cuadrados ($min|Ax - b|^2$) utilizando el algoritmo iterativo del gradiente conjugado. El gradiente conjugado es un método de descenso de gradiente iterativo utilizado para minimizar funciones cuadráticas convexas. El tiempo de cómputo de este método es menor en matrices dispersas ya que para calcular el gradiente se ha de calcular la derivada de los elementos de la matriz. Cuantos menos elementos distintos de cero tenga la matriz menos derivadas se han de calcular y, por lo tanto, menor es el tiempo de cómputo.

Para decidir qué método de resolución utilizar sobre la matriz (Jacobi SVD, factorización QR o mínimos cuadrados del gradiente conjugado (*LSCG*)) se han resuelto los mismos sistemas de ecuaciones con los tres métodos midiendo el tiempo que tarda cada uno conforme aumenta el tamaño del sistema de ecuaciones. Esta comparación se puede observar en la **figura 3.2**.

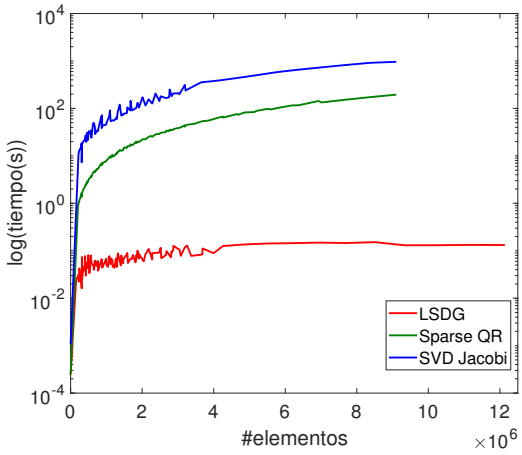
En la gráfica de la **figura 3.2** se puede comprobar que el método más rápido para resolver el sistema de ecuaciones es el de mínimos cuadrados del gradiente conjugado (LSCG). Este método es sustancialmente más rápido ya que está optimizado para matrices dispersas y es un método iterativo.



(a) Numero de incógnitas (número de columnas)



(b) Número de ecuaciones (número de filas)



(c) Número de elementos de la matriz

Figura 3.2: Comparación entre los diferentes métodos utilizados para resolver el sistema de ecuaciones

3.3. IPOPT

Para implementar la restricción no lineal, se decidió utilizar la librería IPOPT [***] (Interior Point OPTimizer) que permite usar este tipo de restricciones sobre sistemas lineales. Esta librería permite resolver problemas del estilo:

[***] `https://projects.coin-or.org/Ipopt`

$$\begin{aligned} \min_{x \in R_n} \quad & f(x) \\ \textbf{s.t.} \quad & g^L \leq g(x) \leq g^U \\ & x^L \leq x \leq x^U \end{aligned} \tag{3.1}$$

Donde $f(x)$ es el sistema de ecuaciones que hay que resolver ($AX - S = 0$, ecuación 2.8) y $g(x)$ es la restricción no lineal que se va a aplicar sobre el sistema (ecuación 2.9). Para solucionar esta ecuación, IPOPT implementa el método del punto interior (*interior point*) que trata de buscar una solución local de la ecuación 3.1. La librería Eigen no dispone de métodos para resolver este tipo de problemas, con restricciones no lineales, de ahí la necesidad de usar IPOPT.

Para usar esta librería se ha de tener suficiente información sobre la matriz que contiene el sistema de ecuaciones, ya que es necesario introducir su primera y segunda derivada, la estructura de Jacobiano y de Hessiano. También es necesario calcular la segunda derivada de esta matriz. Para ello se ha utilizado la aproximación de quasi-Newton, de modo que no es necesario calcular la segunda derivada ni conocer el Hessiano.

Otro parámetro importante para utilizar la librería de IPOPT es la tolerancia. Se considerará que se ha encontrado una solución local cuando el sistema de ecuaciones converja con la tolerancia establecida. Además, se ha de fijar un número de iteraciones máximo para que, en caso de que no encuentre una solución, haya un criterio de parada.

3.4. OpenCV

Para poder detectar los puntos de interés en una imagen se ha utilizado la librería OpenCV[****]. En esta librería se pueden encontrar las diferentes funciones para utilizar los algoritmos de extracción y emparejamiento de puntos. Además, también contiene funciones que implementan algoritmos de emparejamiento de puntos y seguimiento de flujo. La evaluación de las distintas configuraciones extractor-descriptor pueden observarse en el siguiente capítulo.

[****] `https://opencv.org/`

4 Evaluación

Para analizar el algoritmo implementado se han realizado una serie de experimentos utilizando la base de datos ***EuRoC***. En estos experimentos se ha medido tanto el error en la trayectoria (traslación) y en la rotación como el tiempo que tarda el algoritmo en los diferentes experimentos. A partir de los resultados obtenidos se ha mejorado la eficiencia y eficacia del algoritmo.

En esta sección se van a explicar los cuatro experimentos realizados. El primer experimento consiste en analizar los diferentes algoritmos de detección y extracción de puntos. En el segundo experimento se ha evaluado cómo afecta el número de imágenes en el algoritmo en el caso en el que se establece un número de imágenes fijo (modo *offline*). Mientras que, en el tercer experimento, se ha evaluado el algoritmo cuando el número de imágenes es incremental (modo *online*). En el cuarto y último experimento se ha analizado el uso de la restricción no lineal de la aceleración de la gravedad (ecuación 2.9) sobre el sistema de ecuaciones lineal.

Estos experimentos se han realizado en un ordenador con 8 GB de memoria RAM y un procesador Intel(R) Core(TM) i7-5500U CPU@2.40 GHz.

4.1. Base de datos EuRoC

EuRoC [2] es una base de datos en la que se pueden encontrar imágenes estéreo capturadas por 2 cámaras monocromáticas (2x20 FPS, 20Hz) junto con sus medidas del sensor IMU sincronizadas (velocidad angular y aceleración lineal, 200Hz). Además, en esta base de datos también se puede encontrar el movimiento real de la cámara y la estructura real de la escena (*ground truth*). A parte de datos sobre la escena, también se pueden encontrar datos sobre el hardware utilizado como los parámetros intrínsecos de la cámara, los parámetros extrínsecos de la cámara-IMU y la alineación espacio-temporal real.

Las imágenes y las medidas de los sensores de esta base de datos han sido obtenidas con un helicóptero Asctec Firefly hex-rotor (**figura 4.1**). Este robot tiene un sensor visual-inercial (cámara-IMU) y también cuenta con un sistema de captura del movimiento de 6 grados de libertad para obtener la estructura real de la escena (*ground truth*).

Como ya se ha comentado antes, la frecuencia de las cámaras es de 20 Hz y la de la IMU es de 200 Hz. Es decir, se obtiene una imagen cada 50 ms y medidas inerciales cada 5 ms. Por cada imagen se almacenan

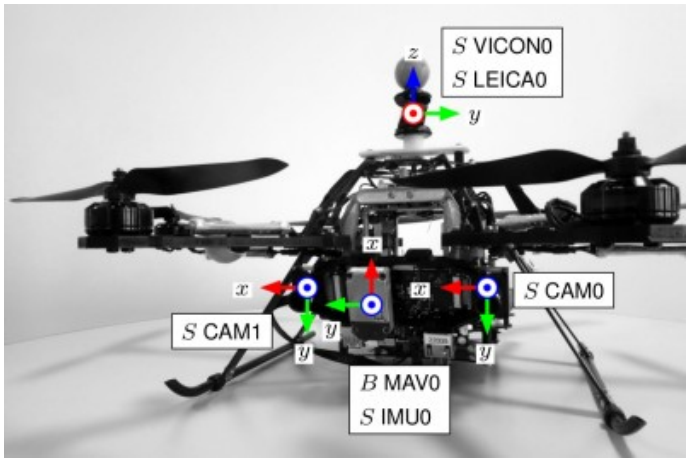


Figura 4.1: Robot utilizado para obtener las imágenes de la base de datos EuRoC. Se pueden observar las dos cámaras con las que se obtienen las imágenes (CAM1 y CAM2), el sensor IMU (IMU0) y los sistemas utilizados para obtener el *ground truth* (LEICA0, posición y VICON0, posición y orientación)

10 medidas de la IMU que se utilizarán para la integración. El tiempo del algoritmo se debe ajustar a estas frecuencias de forma que, cuando llegue una nueva imagen, la anterior ya se haya procesado.

Para evaluar el algoritmo se han utilizado las imágenes de la base de datos correspondientes a la habitación *Vicon 1*. En la **figura 4.2** se pueden ver diferentes objetos dispersos a lo largo de la habitación. Este conjunto de imágenes tiene tres niveles de dificultad (fácil, medio y difícil). Para los experimentos realizados se ha elegido el set de imágenes con nivel de dificultad fácil, con el objeto de aislar la evaluación del algoritmo de otros aspectos como movimientos bruscos o escenas con poca luz.

Figura 4.2: Habitación Vicon 1 (nivel fácil) de la base de datos EuRoc

4.2. Evaluación de los puntos salientes y el algoritmo de Lucas-Kanade

El primer experimento realizado trata de encontrar la mejor combinación de detectores y extractores de puntos salientes. Los detectores localizan los puntos característicos dentro de una imagen, es decir, aquellos pixeles en los que hay cambio de contraste, como por ejemplo una esquina. Una vez detectadas esas zonas, los extractores obtienen los puntos característicos y los pixeles que le rodean (descriptores). Una vez extraídos todos los puntos y sus descriptores, se pueden comparar con los descriptores de otras imágenes. Como se ha mencionado en la sección anterior, entre imagen e imagen hay 50 ms.

En este experimento se han analizado cinco algoritmos diferentes para la detección y extracción de puntos característicos: ***SIFT*** (detector y extractor), ***SURF*** (detector y extractor), ***FAST*** (detector) & ***BRIEF*** (extractor), ***FAST*** (detector) & ***ORB*** (extractor) y ***ORB*** (detector y extractor). Una vez extraídos los puntos, se ha de realizar el emparejamiento entre los puntos extraídos de dos imágenes distintas para ver si los puntos de una imagen están en la siguiente imagen. Para realizar este emparejamiento se ha utilizado el algoritmo FLANN. Además, se ha analizado el método de Lucas-Kanade. A diferencia de los otros algoritmos, Lucas-Kanade hace un seguimiento de los puntos encontrados en la primera imagen.

SIFT [4] El primer algoritmo analizado ha sido *SIFT (Scale-Invariant Feature Transform)*. *SIFT* sirve tanto para la detección como para la extracción de puntos característicos. La principal ventaja de este algoritmo es que es invariante tanto a la rotación como a la escala de la imagen. Es decir, este algoritmo puede encontrar los mismos puntos independientemente de si la imagen ha sido rotada o si ha sido redimensionada.

En los algoritmos de detección se utiliza el Laplaciano de la Gaussiana (LoG) para detectar las diferentes regiones. Este método es muy lento por lo que cada detector lo aproxima con diferentes métodos. En el caso de *SIFT*, LoG se aproxima con la diferencia de Gaussianas (DoG). Esto permite crear la pirámide Gaussiana de la imagen en la que en cada capa la imagen es más pequeña. Un punto de interés es un máximo local en espacio (puntos vecinos) y en escala (capas de la pirámide). Es decir, un píxel en una imagen se compara con sus n vecinos en todas las escalas de la imagen. Si es un máximo local, será un punto de interés. De esta forma, se garantiza la invarianza frente a la escala.

Para cada uno de los puntos detectados se toma un conjunto de 16x16 puntos que se divide en sub-bloques de 4x4. Para cada uno de estos sub-bloques se crea un histograma con la orientación del gradiente y se normaliza con respecto a la orientación dominante. De esta forma, se garantiza la invarianza frente a la rotación. Los descriptores de cada punto de interés son la concatenación de los histogramas de cada uno de los 16 sub-bloques. Debido a esto, los descriptores tienen un tamaño de 128 bytes (16 sub-bloques x 8 casillas del histograma). SIFT tiene diferentes parámetros para encontrar los puntos característicos. Utiliza el umbral de contraste (0.01) para detectar los puntos en regiones semi-constantes. Utiliza otro umbral para detectar las esquinas (10). Además, para la diferencia Gaussiana utiliza otro parámetro σ (1.6).

SURF [1] El segundo algoritmo analizado ha sido *SURF (Speeded-Up Robust Features)*. Este algoritmo está basado en el algoritmo *SIFT* por lo que también es invariante frente a escala y rotación. *SURF* sigue los mismos pasos que *SIFT*, pero utiliza un esquema diferente con el que se consigue mayor rapidez.

La primera diferencia entre estos dos algoritmos es que *SURF* está basado en el determinante del Hessiano. En este caso un punto de interés es un máximo local en posición y escala del determinante del Hessiano siendo el determinante del hessiano: $detH(x, y; t) = t^2(L_{xx}L_{yy} - L_{xy}^2)$. Para aproximar las segundas derivadas se utilizan filtros de caja. Un filtro de caja es el sumatorio de los pesos de un bloque de píxeles. Para que el filtro de caja sea más rápido se utilizan imágenes integrales. Los píxeles de una imagen integral contienen el sumatorio de los pesos de los píxeles que tienen arriba y a la derecha. Además, para que un punto sea de interés, el determinante del Hessiano también debe ser mayor que un mínimo. En este experimento el mínimo Hessiano es 300. Además, sus descriptores ocupan 64 bytes. Para la orientación, *SURF* utiliza Wavelet de Haar.

FAST [7] El siguiente algoritmo analizado ha sido *FAST (Features from Accelerated Segment Test)*. Este algoritmo solo realiza la detección de puntos característicos por lo que se ha de usar con un algoritmo de extracción. La principal ventaja de este algoritmo es el tiempo de ejecución ya que es mucho más rápido que los anteriores algoritmos explicados. La principal desventaja es que tiene menor eficacia ya que no es invariante frente a la escala. Este algoritmo utiliza un umbral t para seleccionar los puntos de interés a partir de la intensidad de sus puntos vecinos. Además, introduce una aceleración comprobando solo algunos de sus puntos vecinos. Un punto se considerará de interés si está rodeado por n vecinos seguidos que son más claros u oscuros que dicho punto. Aunque esta aceleración reduzca el tiempo de ejecución, puede presentar muchos falsos positivos (detecta un punto característico que, en realidad, no lo es).

BRIEF A diferencia del anterior, el algoritmo *BRIEF (Binary Robust Independent Elementary Features)* es solo de extracción. La principal diferencia de este algoritmo con el resto es que utiliza descriptores binarios. *BRIEF* guarda una cadena de bits que corresponde a la comparación entre dos píxeles x e y. Si x tiene más brillo que y se guardará un 1 en esa cadena, en caso contrario, se guardará un 0. Esta cadena de bits es el descriptor. Por último, se aplica suavizado gaussiano con $\sigma = 2$ para mejorar la eficacia. *BRIEF* no es invariante ni a la escala ni a la rotación. La ventaja de *BRIEF* es que más rápido de calcular, utiliza menos memoria y es más rápido comparar descriptores BRIEF. Este algoritmo se ha usado junto con el algoritmo de detección *FAST*.

ORB [8] El último algoritmo de detección y extracción que se ha analizado ha sido el algoritmo *ORB (Orientated FAST and Rotated BRIEF)*. *ORB* es una fusión entre el detector *FAST* y el descriptor *BRIEF*. Primero utiliza *FAST* para encontrar los puntos característicos y luego aplica *BRIEF* para obtener los descriptores. Al igual que en algunos de los anteriores algoritmos descritos, *ORB* también crea la pirámide gaussiana para buscar los puntos en diferentes escalas. El problema es que *FAST* no es invariante frente a la escala y *BRIEF* no lo es a la orientación. *ORB* presenta mejoras de ambos algoritmos.

El algoritmo *FAST* utilizado en *ORB* presenta mejoras con respecto a la eficacia y a la escala. *ORB* elimina los puntos poco salientes ya que se ordenan de mayor a menor según la medida de *"esquinidad" de Harris* y se escogen los *N* primeros puntos. Una vez escogidos estos puntos se calcula la pirámide gaussiana. El algoritmo *FAST* se aplica sobre cada nivel de dicha pirámide. En el experimento realizado se cogen los 600 mejores resultados obtenidos por *FAST*.

Con respecto a las mejoras sobre *BRIEF*, en *ORB* se obtiene la orientación principal del gradiente y se rotan los píxeles con respecto a esa rotación.

FLANN Una vez obtenidos los puntos característicos y sus descriptores se han de emparejar los puntos característicos de dos imágenes consecutivas. Para ello se ha utilizado el algoritmo *FLANN (Fast Library for Aproximage Nearest Neighbors)* y el *test de ratio*. El algoritmo *FLANN* busca el vecino más próximo aproximado. Para ello crea un kd-tree con los descriptores. *FLANN* devuelve una lista con los dos descriptores más cercanos al descriptor que se desea emparejar. Sobre esta lista se aplica el test ratio que descarta aquellos emparejamientos en los que el segundo descriptor está más cerca que el primero debido al ruido. Estos emparejamientos se pueden observar en la **figura 4.3**. *FLANN* utiliza el emparejamiento pasivo. En este emparejamiento se obtienen los puntos en ambas imágenes utilizando uno de los detectores y extractores explicados y luego se aplica *FLANN* sobre los descriptores para emparejar los puntos que sean más similares.

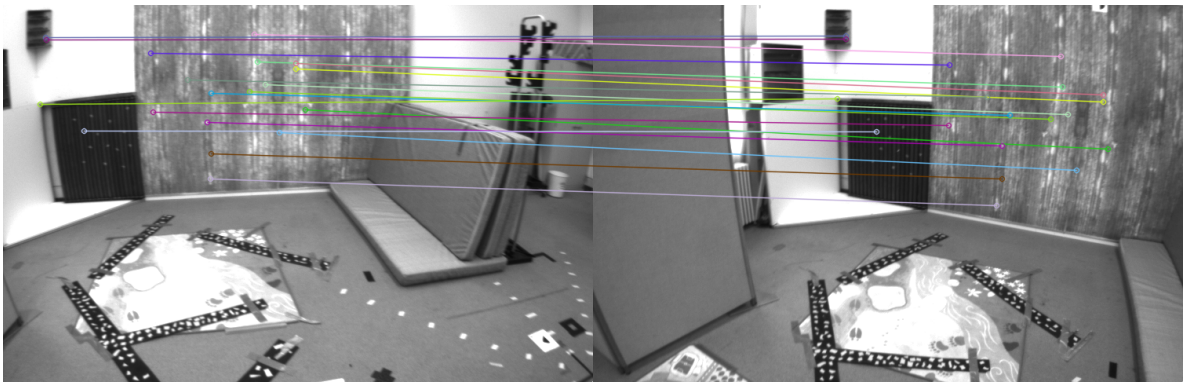


Figura 4.3: Algoritmo *FLANN* aplicado sobre los puntos característicos obtenidos utilizando *SURF*

Lucas-Kanade [5] A parte de los algoritmos de detección y extracción, se decidió analizar el método *Lucas-Kanade* basado en el flujo óptico. El flujo óptico es una técnica de análisis de imágenes que permite determinar el movimiento de un objeto dentro de una secuencia de imágenes. El método de *Lucas-Kanade* asume que el desplazamiento del contenido de la imagen entre dos imágenes consecutivas es pequeño. También se asume que ese contenido es aproximadamente constante en el bloque de pixeles que se está estudiando. Teniendo en cuenta esto, el algoritmo estima el vector de movimiento c en el bloque B para cada píxel x. Después, resuelve la ecuación básica del flujo óptico para todos los pixeles de ese bloque por el criterio de mínimos cuadrados.

El seguimiento de los puntos de una imagen hasta otra se puede ver en la **figura 4.4**. A diferencia de *FLANN*, *Lucas-Kanade* utiliza el emparejamiento activo. Es decir, busca los puntos extraídos en la imagen anterior en la nueva imagen.

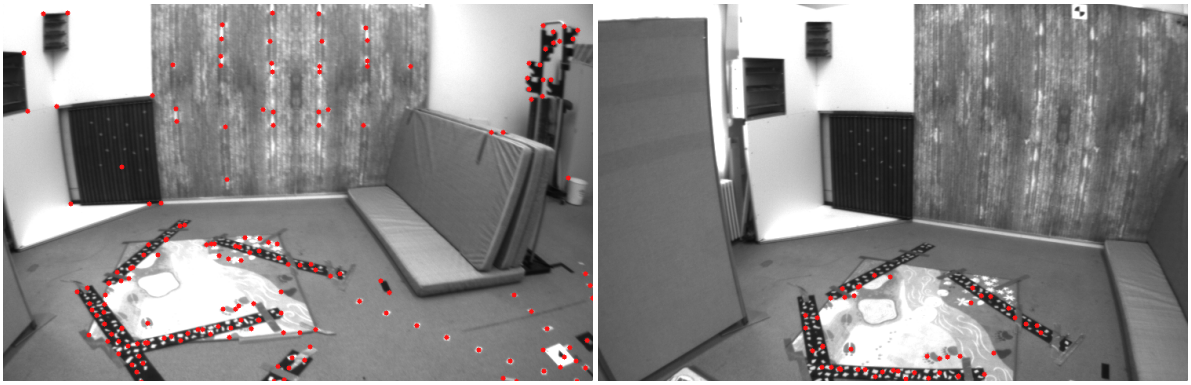


Figura 4.4: Algoritmo *Lucas-Kanade* basado en el flujo óptico para el seguimiento de puntos

Para evaluar cuál de los anteriores algoritmos es el que más se ajusta (en tiempo y precisión) al algoritmo de inicialización del estado visual-inercial se han realizado dos experimentos diferentes. En el primero se ha ejecutado el algoritmo con los diferentes detectores y extractores, y con el método de Lucas-Kanade, y se ha registrado el tiempo de cada uno de ellos (**figura 4.5**). En el segundo se muestra el número de puntos que encuentra cada uno de ellos en relación al número de imágenes consideradas (**figura 4.6**).

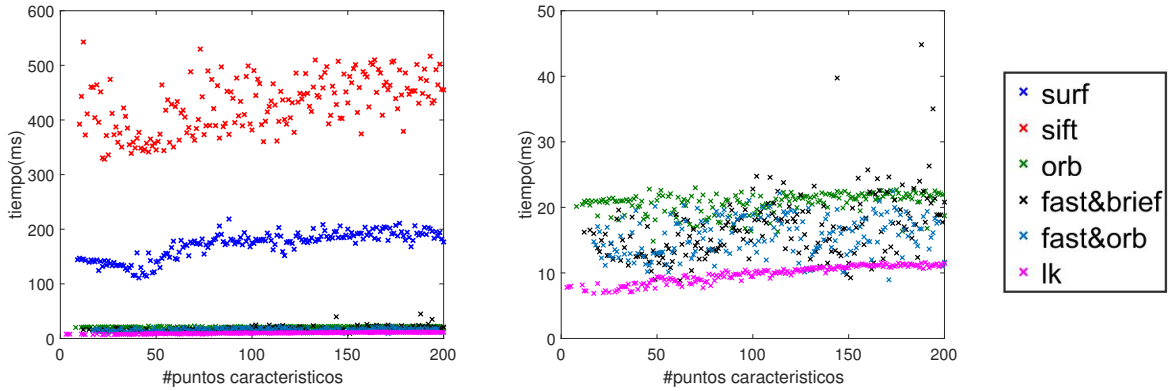


Figura 4.5: Tiempo de detección y extracción de puntos utilizando los diferentes algoritmos descritos y el tiempo de seguimiento de puntos del algoritmo de *Lucas-Kanade*. A la derecha, la gráfica aumentada para poder comparar *FAST&BRIEF, FAST&ORB, ORB y LUCAS-KANADE*

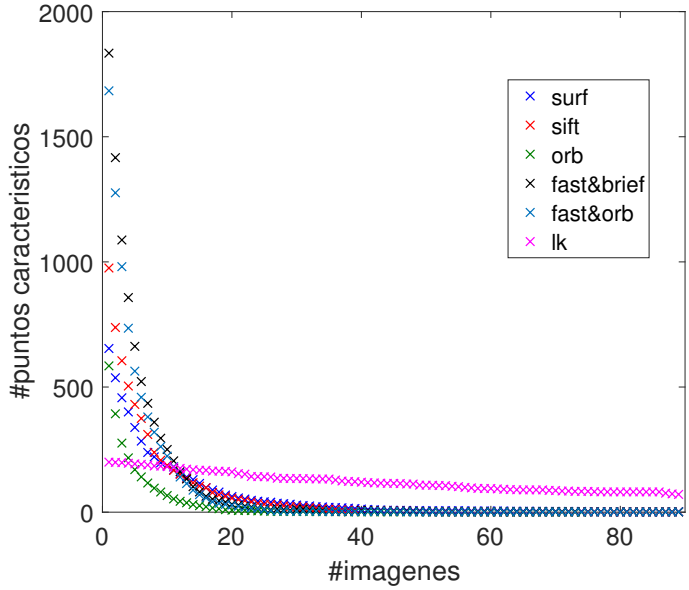


Figura 4.6: Número de puntos característicos se encuentran en cada imagen conforme aumenta el número de imágenes utilizando los diferentes métodos descritos

En las **gráficas 4.5 y 4.6** se puede observar como la mejor opción es utilizar el método Lucas-Kanade. Este método es el que menos tiempo tarda en buscar los puntos en la imagen y es el que más puntos encuentra (el máximo es 200 puntos ya que se ha puesto ese límite de puntos).

Además, se pueden observar las diferentes comparaciones que se han comentado anteriormente. *SIFT* encuentra más puntos que *SURF* pero tarda alrededor de 400 ms en realizar la detección y extracción de los puntos característicos. Esto es debido a que utiliza la diferencia de Gaussianas para aproximar el Laplaciano de la Gaussiana. Como la detección y extracción de puntos debe tardar menos de 50 ms, ya que la frecuencia de la cámara es de 20 Hz, *SIFT* no se puede utilizar. *SURF* tarda menos que *SIFT* ya que utiliza un filtro de caja e imágenes integrales para aproximar el Laplaciano de la Gaussiana pero también tarda más de 50 ms por lo que es descartado.

En estas gráficas también se puede observar como *ORB* sí que se podría utilizar, ya que su tiempo de emparejamiento medio está por debajo de los 50 ms. Este algoritmo no se utiliza ya que tiene baja repetibilidad (no encuentra puntos cuando hay más de 40 imágenes, **gráfica 4.6**).

Además, en la **gráfica 4.6** se pueden ver las mejoras de ORB con respecto a FAST. ORB encuentra menos puntos porque obtiene solo aquellos puntos detectados por *FAST* cuya medida de *"esquinidad" de Harris* sea alta. Es decir, ORB selecciona los mejores puntos de FAST.

El la **gráfica 4.6** no se ha tenido en cuenta el tiempo de emparejamiento con el algoritmo FLANN. Como sin tener en cuenta este tiempo el algoritmo de Lucas-Kanade es el más rápido, no se han evaluado otros algoritmos de emparejamiento como el algoritmo de fuerza bruta.

El método Lucas-Kanade es el que más puntos encuentra en el menor tiempo. Esto es debido a que busca los puntos de la anterior imagen en la nueva imagen (emparejamiento activo) teniendo en cuenta que ha habido un desplazamiento pequeño. Como asume que el desplazamiento ha sido pequeño solo busca los puntos en un radio cercano a la posición en la que se encontraba en la anterior imagen. Esta es la principal diferencia con el emparejamiento pasivo utilizado en el resto de algoritmos. Estos algoritmos buscan los puntos característicos en la nueva imagen y en la imagen inicial y luego buscan emparejarlos con todos los puntos característicos cercanos encontrados en la imagen inicial.

El resto de experimentos se han realizado con el método de Lucas-Kanade ya que es la opción con la que se obtienen mejores resultados con respecto a tiempo y número de puntos característicos.

4.3. Evaluación del algoritmo con un número de imágenes fijo (Offline)

Este experimento consiste en encontrar el número de imágenes óptimo para ejecutar este algoritmo. En este experimento se va a evaluar el algoritmo de modo que se establece un número fijo de imágenes y luego se procesa este número de imágenes. Procesar una imagen consiste en encontrar los puntos característicos comunes a lo largo de todas las imágenes. Una vez procesadas todas las imágenes se aplica el algoritmo de inicialización descrito. Es decir, si se han encontrado más de cinco puntos, se preintegran los datos de la IMU, se construye la matriz que contiene el sistema de ecuaciones a partir de los puntos encontrados y se resuelve dicho sistema.

Para comprobar cuál es el número de imágenes que más se aproxima al óptimo se ha probado con 30, 60 y 90 imágenes y con diferentes imágenes iniciales. En este experimento se utiliza el algoritmo de Lucas-Kanade. Este experimento se puede observar en la **figura 4.7** en la que se muestra el número de puntos encontrados, el número de imágenes establecido y el tiempo que tarda el algoritmo.

En la **figura 4.7** se puede observar como el tiempo aumenta conforme aumenta el número de imágenes y el número de puntos encontrados. Esto es debido a que la matriz que se ha de construir es más grande conforme aumenta el número de puntos encontrados y el número de imágenes. El tamaño de la matriz es cuadráticamente proporcional al número de imágenes y al número de puntos característicos. Cuanto mayor sea el número de imágenes o el número de puntos característicos, mayor será el tiempo de construcción y resolución del sistema.

Además, se puede observar que, conforme aumenta el número de imágenes, disminuye la probabilidad de encontrar, al menos, cinco puntos característicos en todas las imágenes. Esto se puede ver en la **figura 4.7** ya que cuando el número de imágenes es 90 el número de puntos está más concentrado sobre los 0 - 50 puntos. Mientras que cuando es 30 está más concentrado entre los 150 – 200 puntos.

La principal desventaja de tener un número de imágenes fijo es que, aunque este número sea pequeño (30 imágenes) si la cámara se mueve muy rápido no se podrá inicializar con éxito. Uno de los motivos por lo que

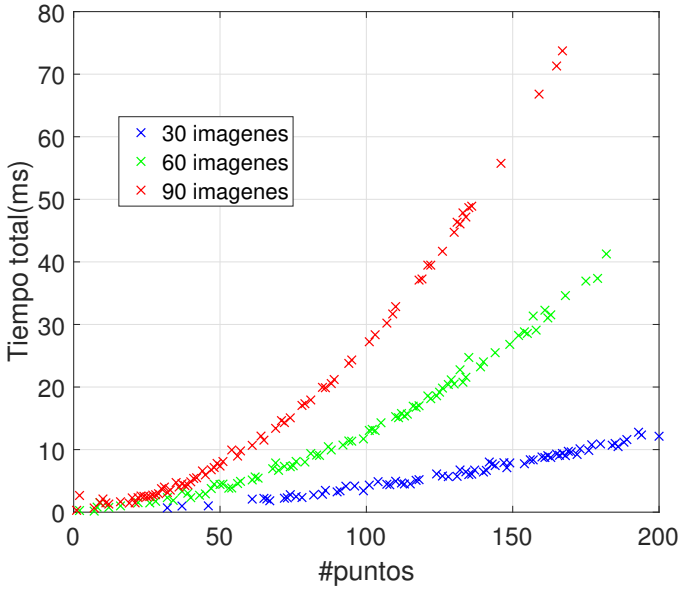


Figura 4.7: Tiempo que tarda el algoritmo, usando el método de *Lucas-Kanade* dependiendo del número de imágenes (30, 60 o 90) con el método offline y del número de puntos encontrados

esto ocurre es debido a que si el contenido varía mucho la probabilidad de encontrar al menos cinco puntos comunes a lo largo de todas las imágenes es pequeña. Además, si hasta la imagen 29 encuentra alguno de los puntos de la primera imagen, pero en la 30 ya no los encuentra ya no se podría inicializar con éxito.

Otra desventaja de este algoritmo es que no se tiene en cuenta la calidad de los puntos, es decir, no se tiene en cuenta si un punto se ha encontrado correctamente en la nueva imagen. Si a lo largo del recorrido no se descartan los puntos espúreos, el error con respecto a la trayectoria que causan estos puntos se irá acumulando a la hora de realizar la preintegración. Este error acumulado se verá reflejado en los resultados finales.

Estas desventajas se han solucionado eliminando la restricción de que se tenga que procesar un número establecido de imágenes para poder aplicar el algoritmo. Para ello se aplica el algoritmo cada vez que se procesa una nueva imagen. La explicación y evaluación de esta solución se explica en la siguiente sección.

4.4. Evaluación del algoritmo con un número de imágenes incremental (Online)

Esta versión del algoritmo soluciona las desventajas del método offline. El método online consiste en aplicar el algoritmo de inicialización cada vez que se procesa una nueva imagen, siempre que se hayan procesado más de cinco imágenes. Además, ya no se procesan todas las imágenes capturadas si no que se procesa una imagen de cada dos imágenes capturadas. El criterio de parada es un número máximo de imágenes procesadas (por defecto 26) o un número mínimo de puntos encontrados (por defecto 5).

En el criterio de parada se ha de tener en cuenta el número de puntos encontrados ya que esto establece el número de imágenes que se van a capturar dependiendo de la velocidad a la que se mueve el dispositivo. Si

se mueve rápido, el contenido de las imágenes capturadas variará bastante y, por lo tanto, el número de puntos encontrados a lo largo de las imágenes estará por debajo de cinco tras capturar pocas imágenes. Sin embargo, si se mueve despacio, el contenido de las imágenes capturadas apenas variará y, por lo tanto, se capturarán muchas imágenes antes de que el número de puntos esté por debajo de cinco. Por esta razón, en el criterio de parada también se ha de tener en cuenta el número de imágenes procesadas. Esto es debido a que las medidas de la IMU tienen ruido que se va acumulando conforme se procesan las imágenes. Por lo que hay que establecer un número máximo de imágenes para que dicho ruido no afecte demasiado a los datos medidos.

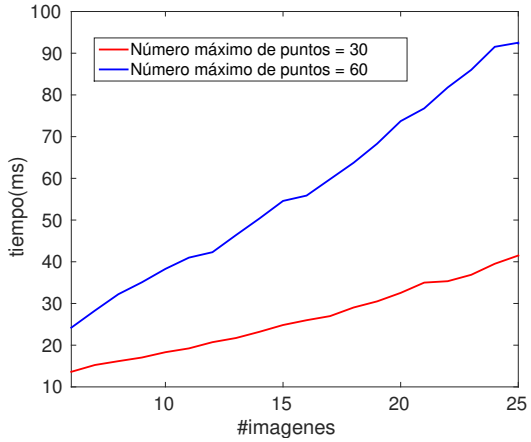
En el método online, cuando se recibe la primera imagen se buscan todos los puntos característicos hasta un máximo de 30 puntos. Durante las siguientes cinco imágenes se buscan esos puntos característicos en la nueva imagen capturada. Si se han encontrado más de cinco puntos presentes en esas cinco imágenes se crea la matriz de ecuaciones por cada punto encontrado y se resuelve. Una vez resueltos todos los sistemas de ecuaciones se seleccionarán los puntos con los que se hayan obtenidos valores de la gravedad y de la velocidad razonables. Un valor de la gravedad razonable es aquel que esta alrededor de la mediana de todos los valores de la gravedad obtenidos. Un valor de la velocidad razonable es lo mismo pero con la mediana de todos los valores de la velocidad obtenidos. Una vez seleccionados los puntos que cumplan esas dos condiciones, se construye el sistema de ecuaciones con esos puntos y se resuelve. En el resto de imágenes, se buscan los puntos característicos de la anterior imagen en la nueva imagen, se construye el sistema de ecuaciones y se resuelve. Esto se realiza hasta llegar a un número máximo de imágenes o a un número mínimo de puntos característicos.

El número máximo de puntos que se buscan en las cinco primeras imágenes se ha ido modificando para que el algoritmo se pueda ejecutar en tiempo real. Es decir, hasta que el tiempo del algoritmo fuera menor o igual que el tiempo entre imagen e imagen saltando una imagen (100 ms). Cuanto mayor es este número de puntos menor es el error con respecto a la trayectoria, ya que hay más puntos para estimar la trayectoria. Sin embargo, cuantos más puntos y más imágenes haya más grande será la matriz que contiene el sistema de ecuaciones y, por lo tanto, mayor será el tiempo de construir y resolver dicho sistema. Esto se puede observar en la **figura 4.8**.

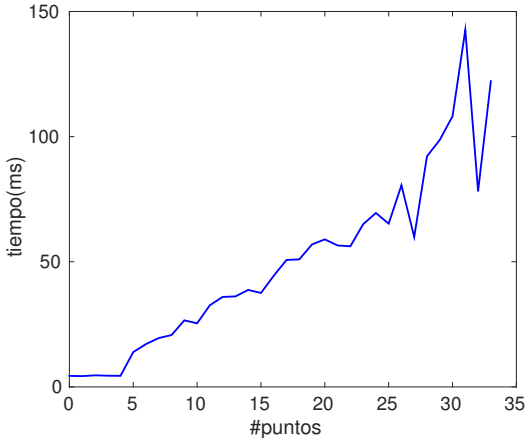
Una de las optimizaciones realizadas elimina los puntos espúreos. Un punto espúreo es un punto que no se ha localizado bien en la nueva imagen. Como se eliminan estos puntos se mejora la precisión y se disminuye el error con respecto a la trayectoria.

Otra optimización del algoritmo, es que el número de imágenes aumenta de dos en dos. Esto se hace porque entre dos imágenes consecutivas se supone que la variación es pequeña, por lo que se puede saltar una imagen. De esta forma la información que se obtenía antes procesando 60 imágenes ahora lo obtiene con 30, almacenando menos datos en memoria, por lo que construir la matriz y resolver el sistema será más rápido.

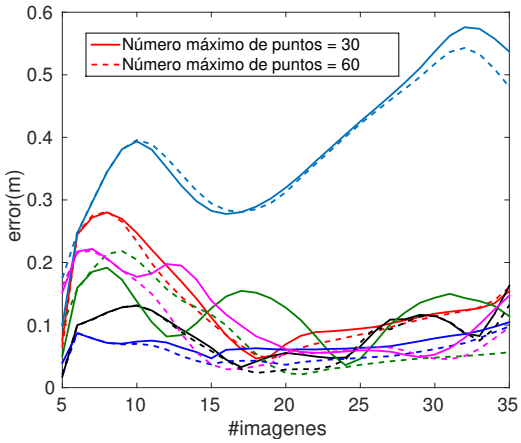
Además, para reducir el tiempo de construcción de la matriz, que en el caso anterior era lo que más tardaba, ahora se construye de forma incremental. En el modo online, se almacena esta matriz y se modifica cada vez que se obtienen los puntos característicos en una nueva imagen. Esta matriz se modifica eliminando aquellas



(a) Tiempo con respecto al número de imágenes



(b) Tiempo con respecto al número de puntos



(c) Error con respecto a la trayectoria

Figura 4.8: Comparación entre el número máximo de puntos que puede encontrar el algoritmo Lucas-Kanade (30 - 60)

filas que pertenezcan a puntos que no se han encontrado en la nueva imagen y añadiendo los nuevos datos. De esta forma, se consigue ahorrar bastante tiempo como se puede observar en la **figura 4.9**.

En la **figura 4.9** se puede observar que cuando el número de imágenes leídas es menor que cinco, el tiempo de ejecución del algoritmo es de unos 10 ms ya que sólo tiene que buscar los puntos en la nueva imagen. Como se ha visto en el primer experimento, el método de Lucas-Kanade tarda muy poco tiempo en localizar estos puntos. Cuando el número de puntos es 30, el tiempo de detección de puntos es mayor ya que se trata de la primera imagen. En este caso el algoritmo de Lucas-Kanade ha de detectar y extraer los puntos característicos de la imagen. En el resto de casos, el algoritmo de Lucas-Kanade, solo realiza el emparejamiento activo explicado anteriormente. El tiempo de ejecución es mayor cuando el número de imágenes procesadas es cinco ya que

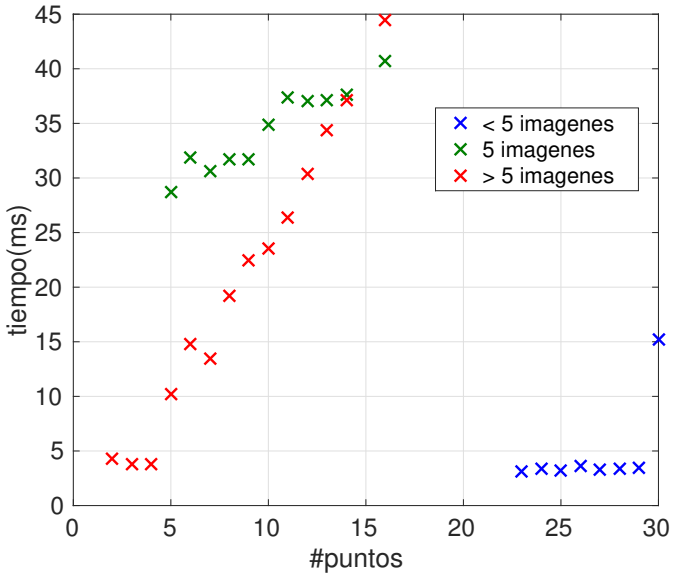


Figura 4.9: Tiempo que tarda el algoritmo, usando el método de *Lucas-Kanade* en encontrar *N* puntos dependiendo del número de imágenes (1-30) con el método online

tiene que construir el sistema y resolverlo para cada uno de los puntos encontrados en la nueva imagen, aplicar el filtro de la mediana sobre estos puntos, construir el sistema y resolverlo con los puntos que hayan pasado el filtro. Por último, cuando el número de imágenes es mayor que cinco, el algoritmo ha de modificar la matriz y resolver el sistema, por lo que el tiempo de ejecución es menor que el caso anterior.

En el modo offline, entre imagen e imagen solo se debían encontrar los puntos característicos en la nueva imagen. En este caso, se deben encontrar los puntos característicos en la nueva imagen y, dependiendo del número de imágenes que se hayan procesado se realizarán una serie de operaciones u otras. Como se puede observar, el modo online necesita más tiempo que el modo offline. Este es otro de los motivos por los que se salta una imagen. De esta forma se dispone para realizar lo mencionado anteriormente 100 ms en lugar de 50 ms.

En resumen, las principales diferencias de utilizar el modo *online* en vez del *offline* son las siguientes:

- En el modo *online* se aplica el algoritmo de inicialización cada vez que llega una nueva imagen, cuando se han procesado más de cinco imágenes. Esto se realiza hasta llegar a un número máximo de imágenes o a un número mínimo de puntos característicos. En el modo *offline* solo se aplicaba el algoritmo de inicialización una vez procesadas el número de imágenes establecido.
- En el modo *online* se aplica el filtro de la mediana para descartar puntos espúreos.
- Mientras que en el modo *offline* se procesaban todas las imágenes capturadas, en el modo *online* se procesa una imagen de cada dos.

- Para disminuir el tiempo de ejecución en el modo *online* se almacena la matriz que contiene el sistema de ecuaciones y se modifica, añadiendo y eliminando las filas y columnas necesarias.

Para comprobar la precisión de este algoritmo con el modo *online* se han realizado diferentes pruebas y se ha calculado el error con respecto a la trayectoria. En la **gráfica 4.10** se puede observar como dicho error va variando conforme se procesa una nueva imagen.

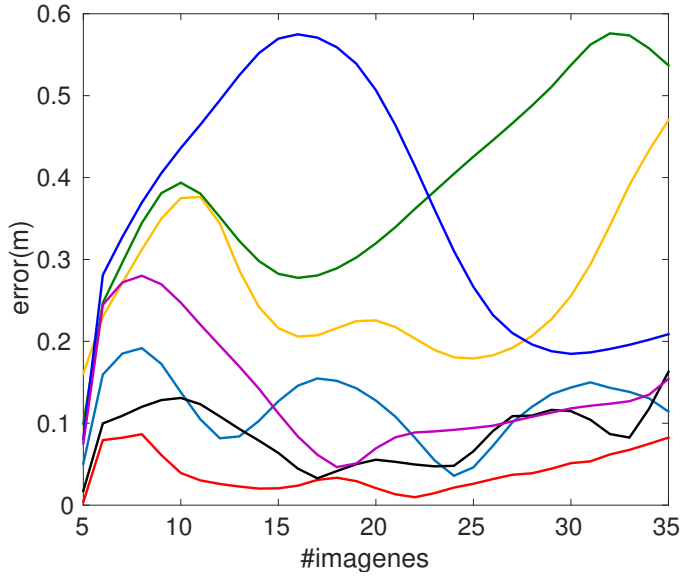


Figura 4.10: Error con respecto a la trayectoria conforme se procesa una nueva imagen

En la **figura 4.10** se puede observar como en todas las pruebas realizadas, el error con respecto a la trayectoria sigue un comportamiento similar. Este comportamiento tiene tres fases diferentes.

1. Al principio, el error es alto ya que no hay suficientes datos para integrar.
2. Conforme se van integrando los nuevos datos obtenidos por los sensores, el error va disminuyendo.
3. Una vez el error llega a su punto mínimo, este va aumentando debido al ruido de los sensores. Este ruido se va integrando y acumulando.

En la **figura 4.11** se puede observar el histograma normalizado del porcentaje del error obtenido con respecto a la trayectoria real. Este algoritmo proporciona una semilla para una optimización no lineal. En trabajos que evalúan esta optimización se ha obtenido que la optimización converge cuando el error con respecto a la trayectoria de este algoritmo es del 30-40 %. A partir de estos datos y de la **figura 4.11** se puede estimar que la tasa de éxito de este algoritmo es de aproximadamente un 75 %, por lo que este algoritmo todavía tiene margen de mejora.

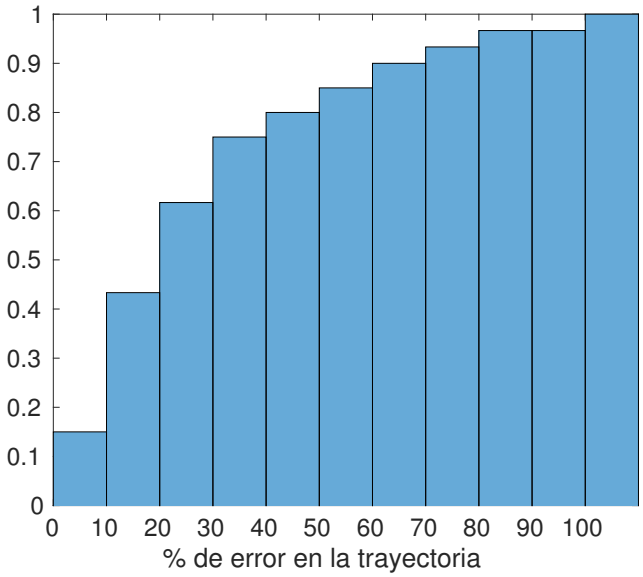
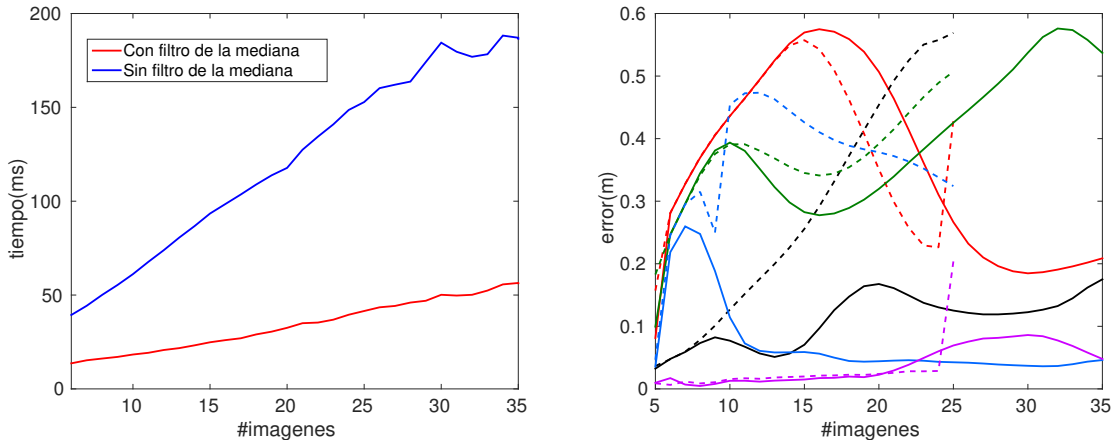


Figura 4.11: Histograma normalizado del porcentaje de error en la trayectoria

El error también puede aumentar debido a que haya poca traslación. Si hay poca traslación, no se puede triangular con precisión para calcular la distancia entre un punto y la cámara (λ_j^i). Además, si hay poco movimiento, el IMU sólo obtendrán ruido. Este ruido se integrará como datos de la IMU y, por lo tanto, se obtendrán resultados erróneos que se irán acumulando.

Con respecto al filtro de la mediana, las mejoras se pueden observar en la **figura 4.12**. En la **figura 4.12(a)** se pueden observar las mejoras con respecto al tiempo. En esta figura se puede ver como el tiempo es menor si se utiliza el filtro de la mediana. Esto es debido a que con este filtro se eliminan los puntos espúreos, por lo que disminuye el tamaño de la matriz. Como el tamaño de la matriz que contiene el sistema de ecuaciones es menor, se necesita menos tiempo construir y resolver este sistema. En la **figura 4.12(b)** se puede observar como el filtro de la mediana también disminuye el error de la trayectoria. Esto es debido a que, a la hora de calcular la posición y la orientación, con las que luego se calculará la traslación de la cámara, solo se utilizan los puntos buenos.

Con las optimizaciones realizadas se puede observar como el tiempo disminuye (**figura 4.9**), ya que la matriz se construye de forma incremental y tiene menos elementos debido al filtro de la mediana. El error también disminuye ya que con este filtro al llegar a la quinta imagen se eliminan los puntos espúreos (**figura 4.12(b)**). Además, la probabilidad de inicializar con éxito aumenta. Esto es debido a que, en este caso, se inicializa con éxito si tras procesar cinco imágenes se consigue construir y resolver el sistema de ecuaciones con al menos cinco puntos. En el caso anterior (*offline*), solo se inicializaba con éxito si se encontraban cinco puntos o más durante el número de imágenes establecido.



(a) Tiempo sin filtro de mediana y con filtro de mediana

(b) Error sin filtro de mediana (línea discontinua) y usando filtro de mediana (línea continua)

Figura 4.12: Comparación entre usar o no el filtro de la mediana

Como se puede observar las optimizaciones realizadas mejoran la eficiencia y eficacia del algoritmo por lo que para el resto de experimentos se va a utilizar el modo *online*.

4.5. Evaluación de la restricción no lineal de la aceleración de la gravedad

En este experimento se va a evaluar si es necesario aplicar la restricción no lineal sobre el sistema de ecuaciones creado. En esta restricción se asume que se conoce *a priori* el valor de la aceleración de la gravedad. Como ya se ha comentado, esta restricción sería $|G| = g$ y puede expresarse con la siguiente ecuación:

$$|IX|^2 = g^2 \tag{4.1}$$

Siendo $I = [I_3, 0_3, ..., 0_3]$, $X = [G^T V^T \lambda_1^1 ... \lambda_1^N ... \lambda_{n_i}^1 ... \lambda_{n_i}^N]^T$ y $g = 9{,}81$. De esta forma se asegura que la norma de las tres primeras componentes del vector X sea la aceleración de la gravedad. Aplicando esta restricción se busca una mejora con respecto a la eficacia.

Para ver cómo afecta esta restricción no lineal se ha medido el tiempo de resolución del sistema de ecuaciones. Este experimento se puede observar en la **figura 4.13**. En esta figura se puede observar como el tiempo de resolución del sistema aplicando esta restricción es mayor.

En la **figura 4.14** se puede observar como el error con respecto a la trayectoria aplicando la restricción no lineal apenas varía. Suele ser el mismo y, en algunos casos, incluso peor. Esto es debido a que el algoritmo aproxima bastante bien el valor de la gravedad y, aplicando el filtro de la mediana, se descartan aquellos puntos cuyo valor de la gravedad se aleje del valor real. Además, en algunos casos puede empeorar ya que al ajustar los tres primeros valores del vector X el resto de valores de este vector también se modifican. Esto hace que

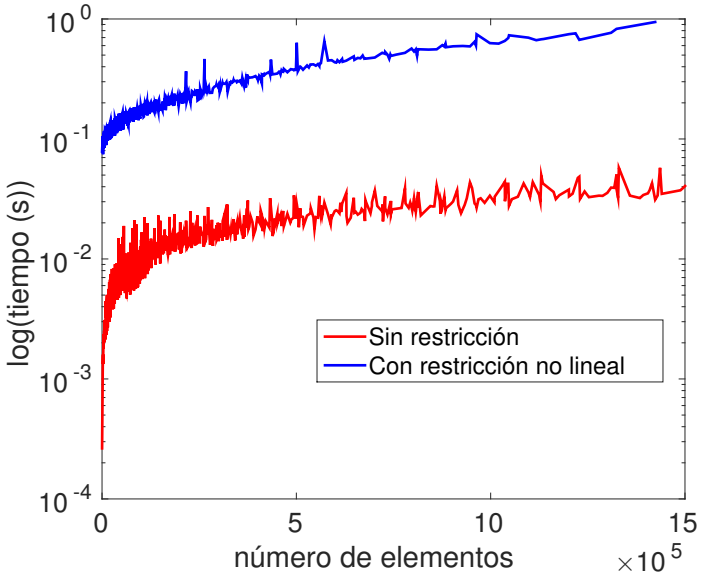


Figura 4.13: Comparación entre el tiempo que tarda el algoritmo si se aplica la restricción lineal y si no se usa

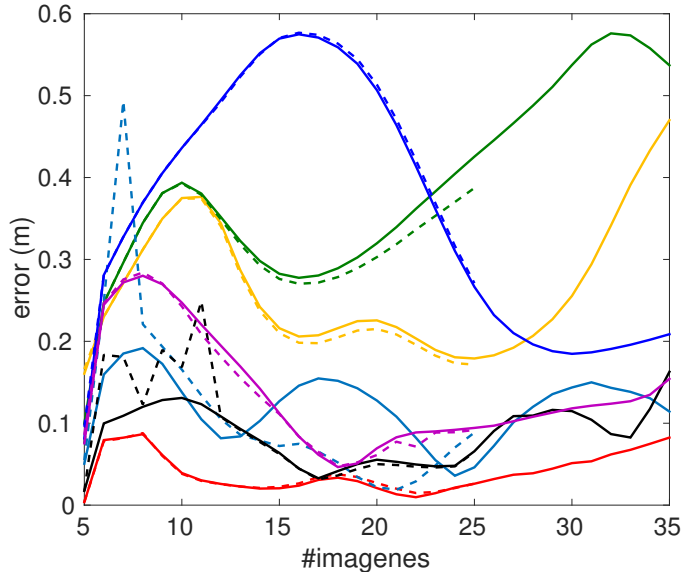


Figura 4.14: Comparación entre el error en la trayectoria obtenido utilizando la restricción no lineal (línea discontinua) y sin usarla (línea continua)

también cambie el valor de la distancia entre un punto y la cámara haciendo que la aproximación con respecto a la trayectoria real sea peor.

Como se puede observar en las **figuras 4.13 y 4.14** si se aplica la restricción no lineal a la hora de resolver el sistema de ecuaciones, el tiempo del algoritmo aumenta y el error con respecto a la trayectoria no varía o incluso puede llegar a empeorar en algunos casos. Con estas observaciones se ha llegado a la conclusión de que no es una buena idea aplicar esta restricción no lineal a la hora de resolver el sistema.

En este trabajo el sesgo/*bias* se obtiene leyendo el *ground truth* del dataset **EuRoC**. En otros trabajos que evalúan este algoritmo estimando el *bias* esta restricción sí que es necesaria para obtener resultados más precisos.

5 Conclusiones

En este trabajo de fin de grado se ha implementado en C++ un algoritmo de inicialización del estado visual-inercial. Una vez implementado se han evaluado los diferentes parámetros del algoritmo para ver cómo afectan a su robustez y precisión. También se ha evaluado su coste. Estos experimentos se han realizado sobre el dataset EuRoC. Se ha utilizado este dataset ya que contiene los datos reales (*ground truth*) por lo que se pueden comparar los resultados obtenidos con estos datos. En total se han realizado cuatro experimentos diferentes.

- Evaluación de los puntos salientes y del algoritmo de Lucas Kanade. En este experimento se ha buscado el modo más óptimo de encontrar emparejamientos entre puntos salientes en varias imágenes.
- Evaluación de la versión *offline*. Se ha evaluado el algoritmo utilizando un número de imágenes fijo. En este caso, primero se obtienen los puntos comunes en ese número de imágenes y luego se aplica el algoritmo de inicialización.
- Evaluación de la versión *online*. En este caso, cada vez que se obtiene una nueva imagen se buscan los puntos comunes y se aplica el algoritmo de inicialización hasta que el número de puntos encontrados es menor que cinco o hasta que se hayan procesado 25 imágenes.
- Evaluación del uso de la restricción no lineal. En este experimento se ha evaluado el algoritmo aplicando la restricción de que se conoce la gravedad *a priori*. También se ha comparado con el algoritmo sin la restricción para comprobar que es más óptimo.

Tras realizar estos cuatro experimentos se han llegado a diferentes conclusiones que se van a recopilar y analizar en este apartado.

Con el algoritmo de inicialización del estado visual-inercial implementado ya no es necesario estimar un estado inicial preciso. Este algoritmo está basado en las medidas visuales e inerciales obtenidas por los diferentes sensores por lo que se puede crear una nube de puntos 3D de una escena sin tener conocimiento previo de ella.

Para encontrar los puntos característicos en una imagen existen diferentes métodos de detección y extracción de puntos como *SIFT, SURF, FAST&BRIEF, FAST&ORB y ORB*. Una vez obtenidos estos puntos, para comprobar si algunos de estos puntos están en otra imagen, se realiza un emparejamiento de puntos mediante el algoritmo *FLANN*. Se ha comprobado que algunos de estos algoritmos están dentro del tiempo límite estable-

cido por la frecuencia de la cámara (20Hz, 50ms). Aunque, también se ha comprobado que, con el algoritmo de Lucas-Kanade se obtienen más puntos en menos tiempo, por lo que se ha decidido utilizar el método de Lucas-Kanade.

Además, también se han analizado y comparado dos versiones diferentes del algoritmo, la primera de ellas (*offline*) captura un cierto número de imágenes establecido y después construye y resuelve el sistema. En la segunda (*online*), cada vez que se procesa una nueva imagen se aplica el algoritmo de inicialización y se va construyendo el sistema de manera incremental. Esto lo realiza hasta llegar a un número máximo de imágenes o a un número mínimo de puntos encontrados en la nueva imagen. Comparando estas dos versiones se puede observar como con la segunda se consigue inicializar el sistema más veces de manera exitosa. Además, en la versión *online* se aplica el filtro de la mediana. Con este filtro se eliminan los puntos espúreos, mejorando el error con respecto a la trayectoria.

Con el último experimento se busca comprobar si aplicar la restricción no lineal, con la que se asume que se conoce *a priori* la aceleración de la gravedad, optimiza el algoritmo o no. Con los resultados de este experimento se ha llegado a la conclusión de que es mejor no aplicar esta restricción ya que hace que el algoritmo sea más lento y se obtienen los mismos resultados o incluso peores. Aunque si se estima el *bias* sí que se debe aplicar.

También se ha comprobado que este algoritmo no es perfecto. Su tasa de éxito ronda el 75 %, teniendo en cuenta que un experimento se considera inicializado con éxito si el error de la trayectoria es menor o igual que el 30 % de la trayectoria real.

Además, como este algoritmo es bastante reciente, algunas de estas conclusiones y experimentos no se pueden encontrar en otros trabajos relacionados.

Con los resultados obtenidos en todos estos experimentos se ha llegado a la conclusión de que, con un algoritmo que no necesita conocimiento previo de la escena se evita el problema de estimar un estado inicial. Por lo que se puede usar el algoritmo de inicialización del estado visual-inercial que utiliza las medidas de los sensores para obtener información sobre la estructura 3D del entorno en el que se encuentra. Además, para utilizar este algoritmo de forma eficaz se han de usar las siguientes opciones:

- El algoritmo de ***Lucas-Kanade*** para localizar los puntos característicos presentes a lo largo de varias imágenes.
- La versión ***online*** del algoritmo (número de imágenes incremental).
- **No se debe aplicar la restricción no lineal** (si el *bias* es conocido) ya que aumenta el tiempo de ejecución del algoritmo y apenas afecta al error con respecto a la trayectoria.

Bibliografía

[1] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[2] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, M Achtelik, and Roland Siegwart. The euroc mav datasets. *Int. J. of Robotics Research*, 2015.

[3] Jacques Kaiser, Agostino Martinelli, Flavio Fontana, and Davide Scaramuzza. Simultaneous state initialization and gyroscope bias calibration in visual inertial aided navigation. *IEEE Robotics and Automation Letters*, 2(1):18–25, 2017.

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

[5] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. 1981.

[6] Agostino Martinelli. Closed-form solution of visual-inertial structure from motion. *International journal of computer vision*, 106(2):138–152, 2014.

[7] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006.

[8] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE international conference on*, pages 2564–2571. IEEE, 2011.