Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Estudio de técnicas de odometría visuo-inercial. Algoritmo VINS-Mono

Autor: María Rodríguez Rodríguez

Tutor: Manuel Vargas Villanueva

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

Sevilla, 2020

Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Estudio de técnicas de odometría visuo-inercial. Algoritmo VINS-Mono

Autor:
María Rodríguez Rodríguez

Tutor:
Manuel Vargas Villanueva
Profesor Titular

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

Sevilla, 2020

Trabajo Fin de Grado: Estudio de técnicas de odometría visuo-inercial. Algoritmo VINS-Mono

Autor: María Rodríguez Rodríguez
Tutor: Manuel Vargas Villanueva

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

En primer lugar, quiero agradecer a mi familia que siempre ha confiado en mi.
A mi tutor, Manuel, por ayudarme a desarrollar este trabajo.
Por último, a mis amigos de la carrera, por hacer estos años tan especiales y haber compartido tantos momentos.

María Rodríguez Rodríguez
Sevilla, 2020

Resumen

El presente trabajo recoge una técnica de estimación de la posición y la orientación de un vehículo a partir de la fusión sensorial de una unidad de medida inercial (IMU) y un sistema de visión monocular. El algoritmo utilizado para ello ha sido VINS-Mono.

Se ha estudiado el fundamento teórico de dicho algoritmo, y también se han realizado una serie de simulaciones. Para una de las simulaciones realizadas se ha utilizado un dataset público de EuRoC MAV que contiene imágenes capturadas por un dron y medidas tomadas por su IMU. Por otro lado, también se han realizado varias simulaciones en Gazebo. Una de ellas es sobre el vuelo de un dron, a partir de un paquete de ROS ya creado. La segunda simulación, en cambio, es sobre un vehículo terrestre modelado desde cero en Gazebo. Este vehículo lleva incorporado una IMU y una cámara, con el objetivo de estimar la trayectoria seguida en tiempo real. Para ello se ha utilizado el software especialista en robótica, ROS.

Abstract

This work includes a technique for estimating the position and orientation of a vehicle from the sensory fusion of an inertial measurement unit (IMU) and a monocular vision system. The name of the algorithm used for this has been VINS-Mono.

The theoretical foundation of this algorithm has been studied, and a series of simulations have also been carried out. For one of the simulations carried out, a public EuRoC MAV dataset was used and it contains images captured by a drone and measurements taken by its IMU. On the other hand, several simulations have also been carried out in Gazebo. One of them is about the flight of a drone, based on an already created ROS package. The second simulation, instead, is on a ground vehicle, which has been fully modeled in Gazebo. This vehicle incorporates an IMU and a camera, in order to estimate the trajectory followed in real time. For this, ROS has been used, that is the robotics specialist software.

Índice Abreviado

Índice

Notación y Siglas

ROS	Robot Operating System. Sistema Operativo Robótico
SDF	Simulation Description Format. Formato Decripción Simulación
VINS	Visual Inertial System. Sistema visuo-inercial
IMU	Inertial Measurement Unit. Unidad de medida inercial
VIO	Visual Inertial Odometry. Odometría visuo-inercial
KLT	Kanade Lucas Tomasi
RANSAC	Random Sample Consesus. Consenso de muestra aleatoria
SLAM	Simultaneous Localization And Mapping. Trazado de mapas y localización simultánea
SfM	Structure from Motion
PnP	Perspective n Point. Perspectiva desde n puntos
GPS	Global Positioning System. Sistema de Posicionamiento Global
DOF	Degrees Of Freedom. Grados de Libertad
$(\cdot)^w$	Sistema de referencia del mundo
$(\cdot)^b$	Sistema de referencia del cuerpo
$(\cdot)^c$	Sistema de referencia de la cámara
$\hat{(\cdot)}$	Estimación de una cantidad
$\mathbf{R}$	Matriz de rotación
$\mathbf{q}$	Cuaternio
$\mathbf{q}_b^w$	Rotación del marco del cuerpo al marco del mundo
$\mathbf{p}_b^w$	Traslación del marco del cuerpo al marco del mundo
b_a	Sesgo del acelerómetro
b_w	Sesgo del giroscopio
b_k	Marco del cuerpo mientras se toma la imagen k^{th}
Δt_k	Duración del intervalo de tiempo $[t_k, t_{k+1}]$
$\mathbf{g}^w$	Vector de gravedad en el marco del mundo
$\mathbf{v}_{b_k}^{b_k}$	Velocidad respecto el sistema de referencia del cuerpo mientras se toma la imagen k^{th}
s	Escala que convierte SfM monocular a unidades métricas
$\mathbf{x}_k$	Estado de la IMU es el momento en el que se toma la imagen k^{th}
λ_l	Inversa de la distancia de la característica l^{th}

$\mathbf{r}_{\mathscr{B}}(\hat{\mathbf{z}}_{b_{k+1}}^{b_k}, \mathscr{X})$	Residuos derivados de la IMU
$\mathbf{r}_{\mathscr{C}}(\hat{\mathbf{z}}_{l}^{c_j}, \mathscr{X})$	Residuos derivados de la visión
$\mathbf{B}$	Conjunto de las medidas de la IMU
$\mathbf{C}$	Conjunto de las características que han sido al menos observadas dos veces en la ventana deslizante actual

1 Introducción

1.1 Motivación

Actualmente, cada vez es más común liberar a los seres humanos de tareas a partir de la aparición de los robots autónomos, los cuales perciben el medio que les rodea y pueden operar sin la supervisión de una persona. Juegan un papel importante en aplicaciones inteligentes como exploración, conducción autónoma, realidad virtual (VR) y realidad aumentada (AR). Este trabajo se centra en vehículos autónomos, tanto aéreos como terrestres.

La presencia de los Vehículos Aéreos No Tripulados (UAV) es cada vez mayor. Se tratan de naves que vuelan sin tripulación y las cuales ejercen su función de forma remota. Su comienzo fue en aplicaciones militares en los Vehículos Aéreos de Combate No Tripulados (UCAV), pero su fama hizo que el campo en el que se utilizaban fuese cada vez más amplio y muchas empresas los han introducido en su trabajo. En los últimos años, también se han utilizado tanto en fábricas como para entretenimiento, gracias al descenso de sus precios.

Por otro lado, tal y como se ha mencionado, también los robots terrestres autónomos se han implantado en la actualidad; un ejemplo de ello son los conocidos robots de limpieza. Otros que siguen siendo muy estudiados son los vehículos autónomos, los cuales detrás ya no sólo tienen estudios de investigación de ingeniería para su funcionamiento, sino también exhaustivos estudios éticos.

(a) Vehículo Aéreo de Combate No Tripulado (UCAV). **(b)** Dron en el interior de una fábrica.

Figura 1.1 Ejemplos de vehículos aéreos.

(a) Robot de limpieza. **(b)** Vehículo autónomo.

Figura 1.2 Ejemplos de vehículos terrestres.

Todos estos robots deben ser capaces de desenvolverse en entornos desconocidos y, para ello, es necesaria la presencia de sensores, de forma que puedan moverse y orientarse sin ayuda de seres humanos.

1.2 Objetivo

En Robótica, para la navegación autónoma es necesario que el robot tenga conocimiento de su posición. Para obtener una buena estimación de la localización existen muchos sensores y técnicas, como los encoders, GPS, sistemas de navegación inerciales y odometría inercial. El problema es que los sensores siempre tienen limitaciones: los encoders pueden tener errores elevados debido al deslizamiento, el GPS puede tener problemas cuando se encuentra en espacios cerrados, y el sensor laser es muy dependiente del material y orientación de las superficies. Nuestro objetivo en este trabajo es proporcionar una técnica de estimación de la trayectoria válida tanto en espacios interiores como exteriores. Además, la estimación de la trayectoria se realiza respecto al punto de partida (a diferencia del GPS, que es un sensor que proporciona coordenadas absolutas, como son las coordenadas geográficas).

Para estimar la posición de un robot se utiliza la odometría. Además, en esta técnica existen diferentes sensores para conseguir el objetivo comentado anteriormente: odometría por encoders en las ruedas (en el caso de vehículos terrestres), sensores de barrido laser o similares, sensores visuales estereoscópicos y odometría visuo-inercial. Esta última es la técnica que se va a desarrollar en este trabajo, la cual fusiona sensores inerciales y sensores visuales. Hay que tener en cuenta dos aspectos: por un lado, tal y como se ha comentado, los sensores siempre tienen cierto error y, por tanto, esta técnica no tiene gran precisión; por otro lado, las posiciones estimadas siempre son relativas al punto de inicio. Aunque se vaya a utilizar la odometría visuo-inercial en este proyecto, se va a explicar cómo se llegó hasta ella.

La odometría convencional se basaba inicialmente en la suposición de que las revoluciones de las ruedas se pueden traducir en un desplazamiento lineal con relación al suelo. Proporciona una buena precisión a corto plazo, de bajo coste y permite altas velocidades de muestreo. Además, si se fusiona con medidas de posición absolutas, como el GPS, se mejora la estimación. Pero alguno de los errores que aparecen son: mal alineamiento de las ruedas, los diámetros de ambas ruedas no son iguales, patinaje de las ruedas...

Figura 1.3 Encoder incremental. Sensor utilizado para determinar la posición, velocidad o dirección.

Una alternativa es la odometría visual (VO), basada en la estimación de la posición y orientación de un robot a partir de una secuencia de imágenes tomadas por una o varias cámaras. Esta se encarga de examinar los cambios en las imágenes capturadas, debidos al movimiento. En la actualidad, se ha visto aumentado el uso de la odometría visual monocular debido, entre otras causas, a su equilibrio entre coste y precisión. La cámara tiene un bajo coste y su precisión suele ser mayor que técnicas de odometría convencionales como, por ejemplo, el encoder (figura 1.3). Por otro lado, si se compara con otras técnicas más actuales, como el GPS, también tiene sus ventajas como su buen funcionamiento en lugares donde este último no sería capaz.

VO se puede dividir en dos categorías según el tipo de cámara que utiliza: estéreoscópica o monocular. La primera tiene dos lentes y es capaz de proporcionar profundidad. Por otro lado, las cámaras monoculares reducen los errores de calibración y son cada vez más usadas en aplicaciones de odometría visual. El principal problema de este tipo de cámaras es que no proporciona la escala métrica.

Para solucionar este problema, en este trabajo se va a utilizar también un sensor inercial o IMU. VINS, o sistema visuo-inercial monocular, fue creado para eliminar los errores producidos por la ausencia de la escala. El sistema mínimo necesario para odometría visuo-inercial (VIO) es aquel formado por una IMU y una cámara monocular. Las cámaras e IMUs son complementarios (la cámara proporciona secuencias de imágenes, mientras que la IMU proporciona aceleraciones y velocidades angulares a una mayor frecuencia). El sensor IMU es un sensor de bajo coste y su principal ventaja es que posee escala métrica. Además, la IMU proporciona valores de los ángulos *roll*, *pitch* y *yaw* (ángulos de alabeo, cabeceo y guiñada, respectivamente). La diferencia entre estos ángulos se explica a continuación mediante el ejemplo de un avión para facilitar su entendimiento:

- *roll*: Rotación respecto al eje que une el morro y la cola del avión.
- *pitch*: Rotación respecto al eje que une punta a punta las alas del avión.
- *yaw*: Rotación respecto al eje que pasa por el centro de gravedad y es perpendicular a los dos ejes mencionados anteriormente.

(a) Cámara estereo. **(b)** Cámara monocular.

Figura 1.4 Tipos de cámara en odometría visual.

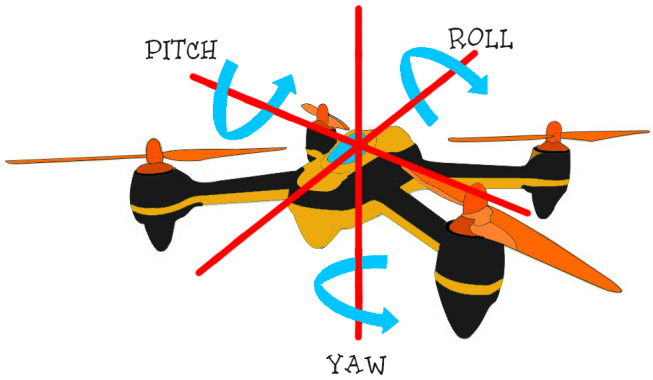


Figura 1.5 Ejes del dron y movimiento sobre ellos.

El sensor inercial también compensa la falta de riqueza de información (contrastes o texturas) en las imágenes, la cual permite extraer características para que posteriormente sean rastreadas.

Como se observará más adelante, las ecuaciones de estimación de movimiento que pueden plantearse a partir de la información visuo-inercial incluye fuertes no linealidades. Como la escala no es observable desde la cámara monocular, es difícil fusionar las medidas de la cámara y la IMU directamente. De ahí, la necesidad de unos buenos valores iniciales. Se verá en el proyecto cómo la etapa decisiva para que la estimación de la trayectoria de un robot sea buena será la inicialización. La existencia de dos sensores hace que sea muy importante la calibración de los parámetros extrínsecos que describen la posición y orientación relativa IMU-cámara.

Para abordar todos estos problemas, en este trabajo estudio en detalle VINS-Mono, un estimador de estados 6 DOF visuo-inercial propuesto por un grupo de robótica aérea en la Universidad de Ciencia y Tecnología de Hong Kong en 2017. Puede ser utilizado en vehículos terrestres, MAVs, *smartphones* y otras plataformas inteligentes. Este estimador hace uso sólamente de un sensor inercial y una cámara monocular.

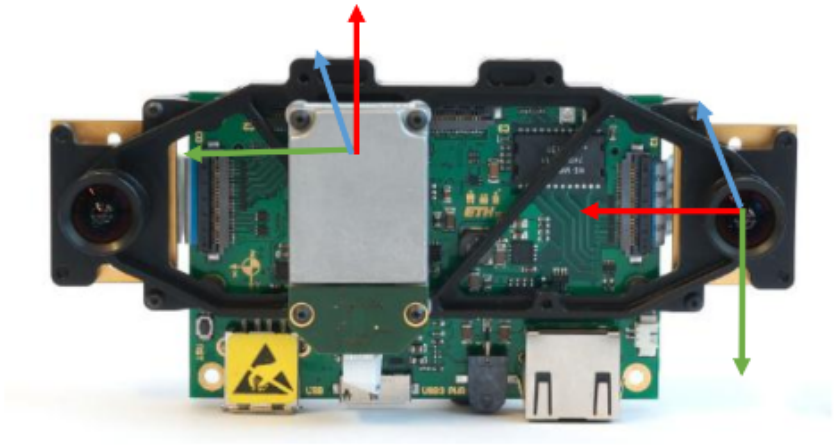


Figura 1.6 Sensores IMU y cámara estereoscópica utilizados en el *dataset* de EuRoC MAV. Este *dataset* se utiliza en el primer experimento del algoritmo de VINS-Mono.

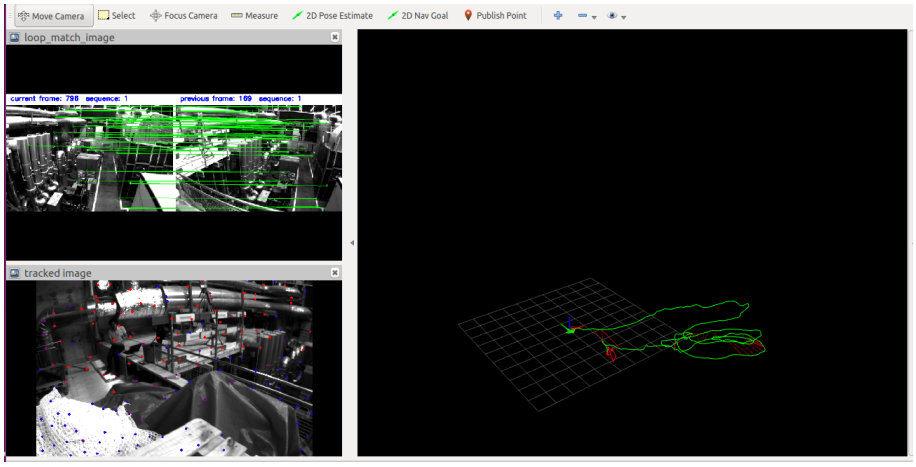


Figura 1.7 Estimación de la trayectoria de un dron a partir de un *dataset* de EuRoC MAV.

Con el objetivo de probar el algoritmo VINS-Mono se realiza una demostración con un *dataset* público de EuRoC MAV. Además, para demostrar su funcionamiento en tiempo real, se han realizado un par de simulaciones en Gazebo. A través del software de robótica ROS, se va a ejecutar el algoritmo en tiempo real con dos modelos de vehículos diferentes (un dron y un robot terrestre). Se pueden manejar y controlar los vehículos por teclado y, a la vez, se puede observar la estimación de la trayectoria obtenida con el algoritmo en un visualizador 3D denominado RViz.

1.3 Estructura del resto del documento

Este proyecto está presentado de forma que se comienza con la explicación del fundamento teórico del algoritmo VINS-Mono, se continúa con los requisitos obligatorios que se han de tener instalados para lanzar el algoritmo, y se termina con varias demostraciones del funcionamiento del mismo.

- **Capítulo 2: Estado del arte**

Clasificación de algoritmos visuo-inerciales según el método utilizado y el nivel de acoplamiento. Presentación de algoritmos capaces de fusionar las medidas de diferentes conjuntos de sensores.

- **Capítulo 3: Algoritmo VINS-Mono : Fundamento Teórico**
  Base teórica sobre el algoritmo y explicación de las diferentes etapas que se pueden observar en su desarrollo: inicialización, preprocesamiento de medidas visuales e inerciales, odometría visuo-inercial, relocalización y optimización.
- **Capítulo 4: Herramientas necesarias**
  Descripción de los programas y librerías necesarias para el buen funcionamiento de VINS-Mono. Explicación de los comandos y herramientas más utilizadas de ROS y Gazebo.
- **Capítulo 5: Algoritmo VINS-Mono : Simulaciones**
  Descripción de los experimentos realizados. El algoritmo ha sido probado tanto en un vehículo aéreo como uno terrestre. También se ha probado en tiempo real, en la simulacion creada de Gazebo. Exposición y comparación de los resultados.
- **Capítulo 6: Conclusiones**
  Reflexión sobre los resultados obtenidos, así como posibles ampliaciones del trabajo.

2 Estado del arte

En la naturaleza, el sistema de localización humano está basado en un sensor inercial biológico y los ojos. Los órganos responsables del sentido del equilibrio, localizados en nuestro oído medio-interno, pueden detectar giros y aceleraciones lineales. Esta es la misma información que perciben los giroscopios y acelerómetros, típicamente integrados en las llamadas IMUs. Esto puede ser traslado en el campo de la ingeniería a VINS (Sistemas de navegación visuo-inerciales). Se puede ver, por tanto, que a nivel biológico, nuestro cerebro también realiza esta fusión visuo-inercial. Esto queda patente por el hecho de que, cuando no existe coherencia entre lo percibido por el sentido del equilibrio y de la vista (como cuando vamos en un vehículo y fijamos la vista en el interior del mismo), podemos sentir malestar o mareo.

En la ingeniería, tal y como se comentó en el apartado 1.2, las estimaciones de la localización tienen errores debido a las incertidumbres de los sensores. Si se utilizan múltiples sensores, estos errores se pueden limitar y reducir. La IMU, por ejemplo, siempre tiene una cierta incertidumbre. En la actualidad, se utilizan cámaras para complementar la información de la IMU y así reducir los errores. Los sistemas de navegación visuo-inerciales basados en la fusión de la información procedente de una cámara y una IMU recibe el nombre de VINS. Estos son capaces de estimar la posición y orientación de un vehículo desenvolviéndose en un entorno desconocido a priori, además de recuperar la escala métrica.

VINS se puede clasificar generalmente en sistemas de fusión estrechamente o débilmente acoplados [17]. En los sistemas débilmente acoplados hay dos variantes: la información de la IMU se incorpora como medidas independientes para optimizar las estimaciones realizadas con las medidas visuales, o bien en el filtro de Kalman Extendido se utilizan las medidas de la IMU en la fase de predicción de estados, mientras que las visuales sólo en la etapa de actualización. Por otro lado, en los sistemas estrechamente acoplados, se optimiza de forma conjunta las medidas de todos los sensores, proporcionando una mayor precisión. Este último sistema es el foco principal de estudio gracias a los avances de la tecnología de computación.

VINS también puede dividirse según el método en el que se base: en filtro, en optimización o en *deep learning* [21]. Los métodos basados en filtros normalmente utilizan el filtro de Kalman extendido (EKF). Las medidas del sensor inercial de alta frecuencia son usadas en la propagación de estados, mientras que las medidas visuales son usadas para la etapa de actualización. Por ejemplo, el algoritmo MSCKF (Multi-State Constraint Kalman Filter) está basado en el EKF, y consiste en mantener poses estimadas anteriores y medidas visuales de una misma característica desde diferentes puntos de vista. Los métodos basados en filtros son sensibles a la sincronización en el tiempo, ya que cualquier medida que llegue tarde causa problemas debido a que los estados no

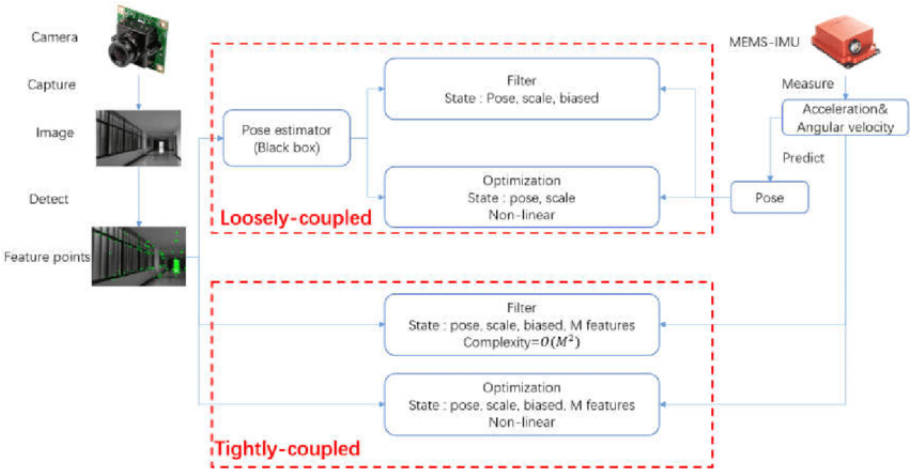


Figura 2.1 Clasificación VINS.

pueden ser propagados atrás en el tiempo. En cambio, los métodos basados en optimización no tienen este problema. Estos últimos optimizan muchas variables a la vez y, además, tienen mayor precisión a costa de complejidad computacional. Maplab y VINS-Mono son los algoritmos típicos de estos métodos (basados en filtros y en optimización, respectivamente) y ambos son *open source* (de código abierto). En resumen, los métodos basados en optimización consiguen mayor precisión en la localización y necesitan menos memoria, mientras que los métodos badados en filtro tienen menor coste computacional.

Por último, los métodos basados en *deep learning* han ganado especial atención debido a su capacidad de aprendizaje y robustez ante entornos desafiantes. Comparado con los métodos tradicionales, reduce el impacto de la incorrecta calibración IMU-cámara, la desincronización y los datos desaparecidos. Dentro de este tipo también se puede hacer una subdivisión en métodos supervisados, semi-supervisados o no supervisados.

Además del algoritmo VINS-Mono, el cual se va a desarrollar y explicar en este trabajo, hay muchos VIO *pipelines* disponibles [11] como MSCKF, OKKVIS, ROVIO, SVO+MSF, y SVO+GTSAM. Sobre todo han sido desarrollados y desplegados en vehículos aéreos. La mayoría son monoculares debido al hecho de que sólo llevar incorporada una cámara reduce peso y consumo de potencia, respecto a otros sistemas como son estéreo o multi-cámara.

Actualmente, los estudios más recientes ya están investigando algoritmos que puedan emplearse para diferentes conjuntos de sensores [19]. Hasta ahora, la mayoría de algoritmos funcionan para un sólo sensor o para un conjunto de sensores concreto (como cámaras estereo, cámara monocular con una IMU o cámaras estéreo con una IMU). Muy pocos pueden ser usados para diferentes opciones. Es por ello, que se busca un algoritmo que soporte diferentes conjuntos de sensores. La principal ventaja es la posibilidad de eliminar un sensor en el momento en el que falle, y sustituirlo por otro rápidamente.

Uno de los algoritmos VINS que puede ser usado para múltiples sensores es Multi-Sensor Fusion Towards VINS. Las entradas son la cámara estéreo, LiDAR 3D e IMU. Se encarga de estimar el estado del robot y genera una trayectoria global. La metodología de la fusión multi-sensor utiliza una optimización de gráfico de poses con detección de bucle cerrado.

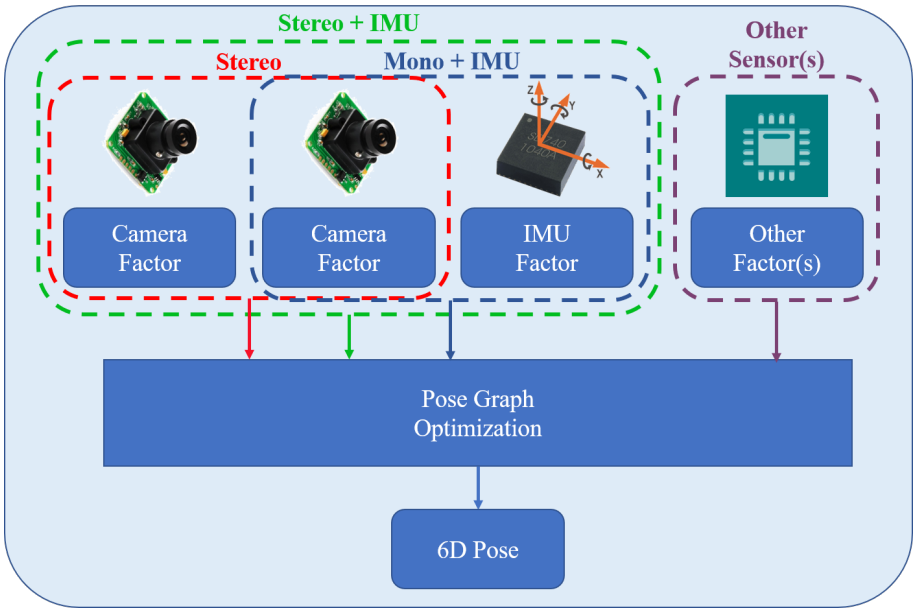


Figura 2.2 *Framework* propuesto para la estimación de estados que soporta múltiples conjuntos de sensores.

3 Algoritmo VINS-Mono: Fundamento teórico

Tal y como se comentó en la introducción del trabajo, VINS-Mono es un algoritmo cuyo objetivo es la estimación de estados basándose en las medidas de una unidad de medida inercial, o IMU, y una cámara monocular (se trata del conjunto de sensores mínimo).

Algunas de las características principales de este algoritmo son las siguientes:

- Consta de un proceso de inicialización robusto capaz de arrancar el sistema a partir de valores iniciales desconocidos.
- Estrechamente acoplado, es un método basado en optimización (tal y como se explicó anteriormente es más complejo pero a la vez más preciso).
- Detección de bucles y relocalización.
- Optimización de grafo de poses global de 4 grados de libertad (las derivas sólo existen en x,y,z y *yaw*, ya que gracias a la IMU los ángulos *roll* y *pitch* son observables).

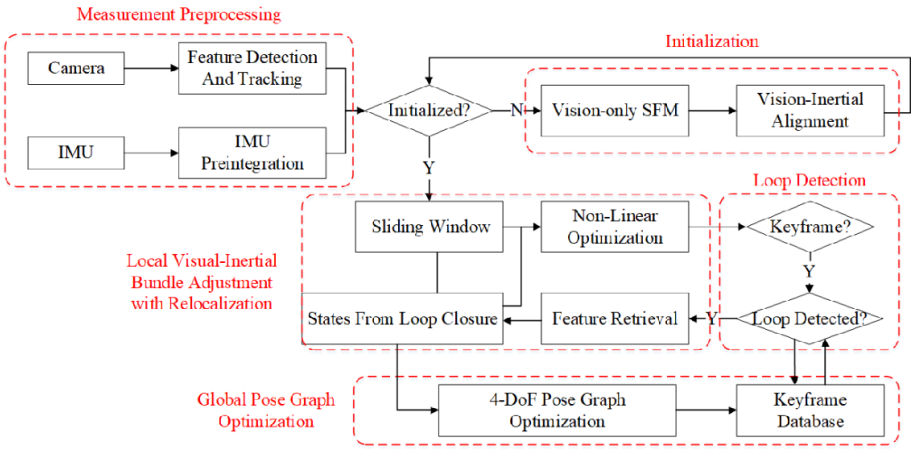


Figura 3.1 Esquema general VINS-Mono.

Como se puede observar en la figura 3.1, este sistema comienza con el preprocesamiento de las medidas: por un lado, se extraen las características de las imágenes captadas por la cámara, y por otro lado, se preintegran las medidas de la IMU de dos imágenes consecutivas. La etapa de inicialización proporciona todos los valores necesarios como la posición y velocidad inicial, el

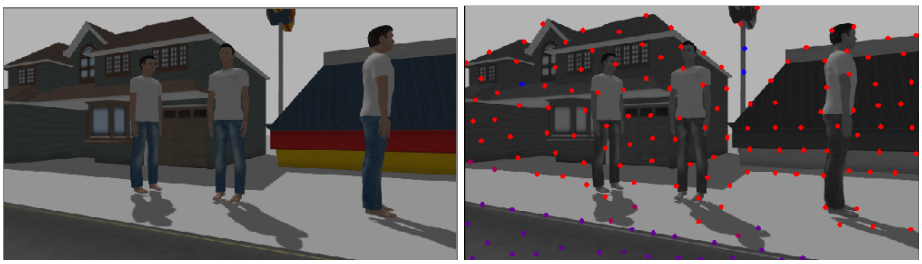
vector de gravedad y los sesgos. Posteriormente, la fase VIO fusiona las medidas de la IMU, las características observadas y las que se vuelven a observar tras el cierre de lazo. En último lugar, se lleva a cabo la optimización del grafo de poses, eliminando las derivas. Las etapas de VIO, relocalización y optimización se ejecutan en paralelo.

3.1 Preprocesamiento de medidas

En este capítulo se hablará tanto del preprocesamiento de las medidas inerciales como visuales.

3.1.1 Preprocesamiento de medidas visuales

Por un lado, esta etapa tiene como objetivo rastrear en la nueva imagen captada puntos característicos encontrados en imágenes anteriores a partir del algoritmo KLT [7]. Además, un detector busca nuevos puntos característicos para mantener un número mínimo de éstos puntos por imagen (100-300), proporcionando una distribución uniforme gracias a la restricción de que exista una distancia mínima entre los puntos rastreados. Para eliminar los valores atípicos se utiliza el algoritmo RANSAC, el cual es uno de los estimadores más robustos en el campo de la visión por computador [10].



(a) Imagen original. **(b)** Imagen con características encontradas.

Figura 3.2 Detección de carácterísticas en una imagen.

Por otro lado, se eligen los fotogramas clave (o *keyframes*) a partir de dos criteros. Un fotograma clave es aquel que se toma como referencia con el objetivo de sólo almacenar dicho fotograma y a partir de ese almacenar los cambios de los siguientes fotogramas en referencia al primero (de esta forma se ahorra mucho espacio de almacenaje). Los criterios son los siguientes:

- El paralaje medio de los puntos rastreados entre un fotograma y el anterior debe ser superior a un límite. Aqui influye tanto la rotación como la traslación.
- Si el número de características detectadas entre imágenes es más baja que un cierto límite, también se tratará como un nuevo *keyframe*. El objetivo es evitar la pérdida de características de la imagen.

3.1.2 Preintegración IMU

La preintegración de la IMU fue propuesta por primera vez en [16] y permite resumir muchas medidas inerciales en una única restricción de movimiento relativa. La velocidad con la que la IMU publica medidas es mucho más elevada que la de la cámara. Esto provoca que existan muchas medidas inerciales entre dos fotogramas consecutivos.

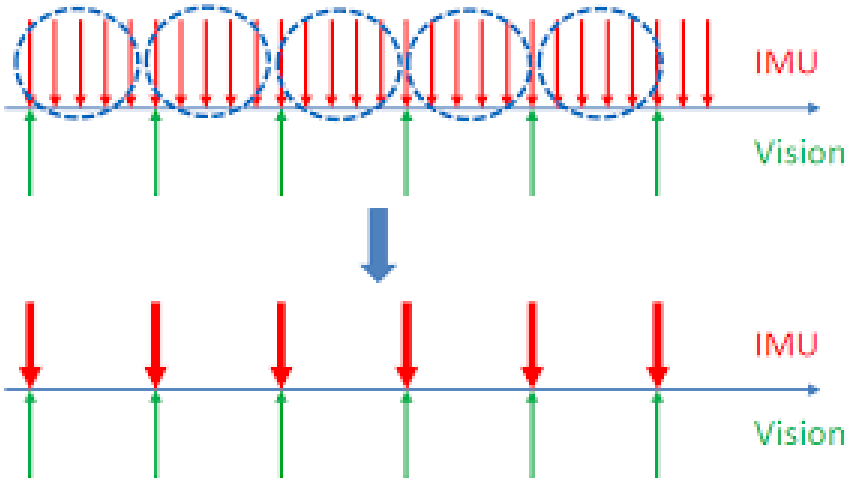


Figura 3.3 Esquema de la preintegración IMU.

Suponemos que queremos estimar el estado de un sistema formado por una IMU y una cámara monocular, asumiendo que el marco $\{B\}$ de la IMU coincide con el marco del cuerpo, y que la disposición entre la IMU y la cámara es fija. Las entradas de nuestra estimación son las medidas de la cámara y la IMU. Las medidas de la cámara en el keyframe i es C_i; por otro lado, I_{ij} es el conjunto de medidas entre dos keyframes consecutivos i y j. Cada conjunto I_{ij} pueden tener cientos de medidas inerciales. El conjunto de medidas recogidas hasta el instante k es

$$Z_k = \{C_i, I_{ij}\}_{(i,j)\varepsilon K_k} \tag{3.1}$$

Se ha de recordar que una IMU está formada por un giroscopio y un acelerómetro que miden la velocidad angular ($\hat{\mathbf{w}}$) y la aceleración lineal ($\hat{\mathbf{a}}$), respectivamente. Estas medidas se ven afectadas tanto por sesgos como por ruidos, y las ecuaciones son las siguientes:

$$\begin{aligned} \hat{\mathbf{a}}_\mathbf{t} &= \mathbf{a}_\mathbf{t} + \mathbf{b}_{\mathbf{at}} + \mathbf{R}_\mathbf{w}^\mathbf{t}\mathbf{g}^\mathbf{w} + \mathbf{n}_\mathbf{a} \\ \hat{\mathbf{w}}_\mathbf{t} &= \mathbf{w}_\mathbf{t} + \mathbf{b}_{\mathbf{wt}} + \mathbf{n}_\mathbf{w} \end{aligned} \tag{3.2}$$

$\mathbf{b}_{\mathbf{at}}$ y $\mathbf{b}_{\mathbf{wt}}$ son los sesgos del acelerómetro y del giroscopio, respectivamente. Por otro lado, $\mathbf{n}_\mathbf{a}$ y $\mathbf{n}_\mathbf{w}$ son los ruidos del acelerómetro y del giroscopio. Las medidas obtenidas de la IMU son respecto el marco del cuerpo. Dados dos instantes de tiempo correspondientes con las imágenes b_k y b_{k+1}, los estados de la posición, velocidad y orientación pueden ser propagados a partir de las medidas inerciales en el intervalo $[t_k, t_{k+1}]$:

$$\begin{aligned} \mathbf{p}_{\mathbf{b_k}+\mathbf{1}}^\mathbf{w} &= \mathbf{p}_{\mathbf{b_k}}^\mathbf{w} + \mathbf{v}_{\mathbf{b_k}}^\mathbf{w}\Delta t_k + \iint_{[t_k,t_{k+1}]} \left(\mathbf{R}_\mathbf{t}^\mathbf{w}\left(\hat{\mathbf{a}}_\mathbf{t} - \mathbf{b}_{\mathbf{at}} - \mathbf{n}_\mathbf{a}\right) - \mathbf{g}^\mathbf{w}\right) dt^2 \\ \mathbf{v}_{\mathbf{b_k}+\mathbf{1}}^\mathbf{w} &= \mathbf{v}_{\mathbf{b_k}}^\mathbf{w} + \iint_{[t_k,t_{k+1}]} \left(\mathbf{R}_\mathbf{t}^\mathbf{w}\left(\hat{\mathbf{a}}_\mathbf{t} - \mathbf{b}_{\mathbf{at}} - \mathbf{n}_\mathbf{a}\right) - \mathbf{g}^\mathbf{w}\right) dt^2 \\ \mathbf{q}_{\mathbf{b_k}+\mathbf{1}}^\mathbf{w} &= \mathbf{q}_{\mathbf{b_k}}^\mathbf{w} + \int_{[t_k,t_{k+1}]} \frac{1}{2}\Omega\left(\hat{\mathbf{w}}_\mathbf{t} - \mathbf{b}_{\mathbf{wt}} - \mathbf{n}_\mathbf{w}\right)\mathbf{q}_\mathbf{t}^{\mathbf{b_k}} dt \end{aligned} \tag{3.3}$$

En la ecuación (3.3) se puede obtener una estimación del movimiento entre los instantes k y $k+1$,y se puede observar que los estados de propagación de la IMU dependen de la posición, velocidad y rotación del fotograma b_k. Por tanto, cuando los estados de b_k cambian, se vuelven a repropagar las

medidas IMU. Esto ocurre por ejemplo cuando se ajustan las poses en la etapa de optimización.

El objetivo es evitar repetidas integraciones. Si cambiamos el marco de referencia del mundo al local b_k, en la ecuación (3.5) se observa que α, β y γ dependen de los sesgos de la IMU, pero no de los estados de b_k y b_{k+1}. Por tanto, cuando la estimación de los sesgos varía poco, se pueden ajustar α, β y γ; en caso contrario, se vuelve a realizar la repropagación. Esto reduce el coste computacional.

$$\begin{aligned}
\mathbf{R}_w^{b_k}\mathbf{p}_{b_{k+1}}^w &= \mathbf{R}_w^{b_k}\left(\mathbf{p}_{b_k}^w + \mathbf{v}_{b_k}^w\Delta t_k - \frac{1}{2}\mathbf{g}^w\Delta t_k^2\right) + \boldsymbol{\alpha}_{b_{k+1}}^{b_k} \\
\mathbf{R}_w^{b_k}\mathbf{v}_{b_{k+1}}^w &= \mathbf{R}_w^{b_k}(\mathbf{v}_{b_k}^w - \mathbf{g}^w\Delta t_k) + \boldsymbol{\beta}_{b_{k+1}}^{b_k} \\
\mathbf{q}_w^{b_k}\otimes\mathbf{q}_{b_{k+1}}^w &= \boldsymbol{\gamma}_{b_{k+1}}^{b_k}
\end{aligned} \tag{3.4}$$

donde

$$\begin{aligned}
\boldsymbol{\alpha}_{b_{k+1}}^{b_k} &= \iint_{t\in[t_k,t_{k+1}]} \mathbf{R}_t^{b_k}(\hat{\mathbf{a}}_t - \mathbf{b}_{a_t} - \mathbf{n}_a)dt^2 \\
\boldsymbol{\beta}_{b_{k+1}}^{b_k} &= \int_{t\in[t_k,t_{k+1}]} \mathbf{R}_t^{b_k}(\hat{\mathbf{a}}_t - \mathbf{b}_{a_t} - \mathbf{n}_a)dt \\
\boldsymbol{\gamma}_{b_{k+1}}^{b_k} &= \int_{t\in[t_k,t_{k+1}]} \frac{1}{2}\boldsymbol{\Omega}(\hat{\boldsymbol{\omega}}_t - \mathbf{b}_{w_t} - \mathbf{n}_w)\boldsymbol{\gamma}_t^{b_k}dt.
\end{aligned} \tag{3.5}$$

$\boldsymbol{\alpha}_{b_{k+1}}^{b_k}$, $\boldsymbol{\beta}_{b_{k+1}}^{b_k}$ y $\boldsymbol{\gamma}_{b_{k+1}}^{b_k}$ son los términos incrementales resultantes de la preintegración IMU.

3.2 Inicialización

Como ya se ha comentado durante este trabajo, la odometría visuo-inercial monocular es un sistema altamente no lineal: la escala no es observable a partir de la cámara monocular y, por tanto, es muy difícil fusionar medidas de ambos sensores directamente (son necesarios unos buenos valores iniciales). Esto provoca que la inicialización sea la fase más delicada para este algoritmo. Además, esta complejidad aumenta si la IMU tiene elevados errores y sesgos.

3.2.1 Ventana deslizante sólo visión SfM

Se emplea un método débilmente acoplado para la fusión sensorial, que comienza con un Structure from Motion (SfM) [20] en una ventana deslizante. Con un sistema sólo visual es posible obtener los valores iniciales, pero si luego se alinea con las medidas de la IMU se puede obtener también los valores de la gravedad, velocidad, sesgos y escala.

Esta ventana de fotogramas SfM se utiliza para reducir complejidad. En primer lugar, se calcula la posición relativa entre dos imágenes por el algoritmo de los 5 puntos cuando se cumpla la siguiente condición: existen más de 30 puntos característicos comunes y suficiente paralaje entre un fotograma anterior y el último. El fotograma antiguo se convierte en el de referencia $\{C_0\}$ (este es el marco de referencia de la cámara en la primera posición). En el caso de que no se cumpla la condición anterior, se añade el fotograma a la ventana y se esperan nuevos fotogramas. Si el algoritmo de 5 puntos tiene éxito, se establece la escala y se triangulan todas las características entre los 2 fotogramas y, por último, se aplica un ajuste para minimizar los errores de proyección. Posteriormente, se estimará la pose del resto de fotogramas de la ventana. Todos estos valores obtenidos son

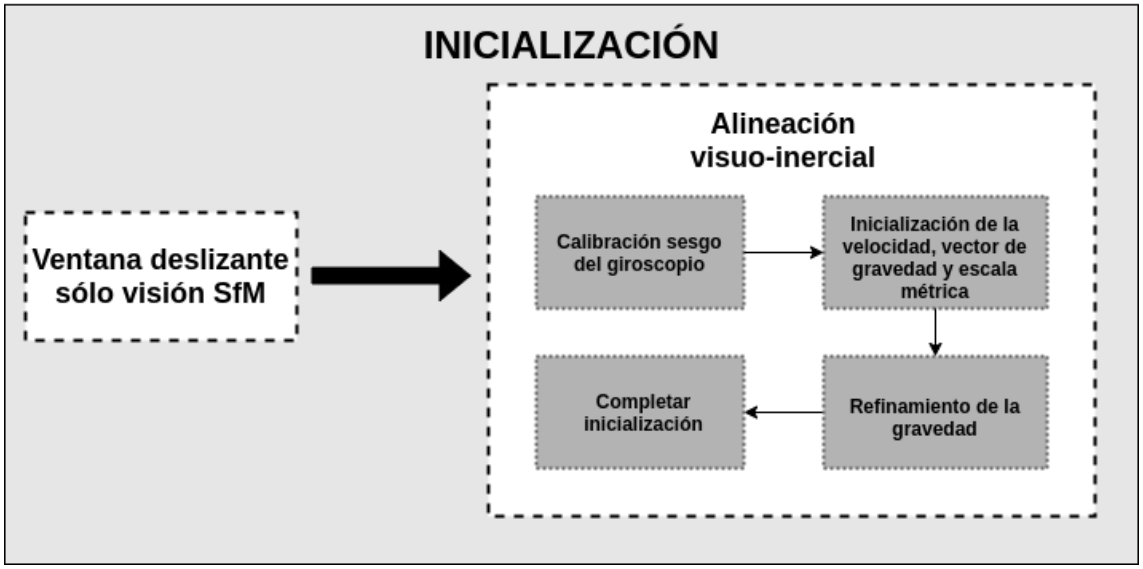


Figura 3.4 Esquema resumen etapa de inicialización.

posiciones y rotaciones, siempre respecto $\{C_0\}$ debido a que no se conoce sobre el marco del cuerpo.

Conocidos los parámetros extrínsecos, se pueden transformar las poses respecto la cámara al marco del cuerpo (IMU) en la ecuación (3.6). El parámatro s es la escala, y es un factor fundamental para poder alinear las medidas visuales con las inerciales, y así obtener una buena alinealización.

$$\begin{aligned} \mathbf{q}_{b_k}^{c_0} &= \mathbf{q}_{c_k}^{c_0} \times \left(\mathbf{q}_c^b\right)^{-1} \\ s\bar{\mathbf{p}}_{b_k}^{c_0} &= s\bar{\mathbf{p}}_{c_k}^{c_0} - \mathbf{R}_{b_k}^{c_0}\mathbf{p}_c^b \end{aligned} \tag{3.6}$$

3.2.2 Alineación visuo-inercial

En esta etapa, mediante la alineación de las medidas visuales e inerciales se puede obtener la escala, la gravedad, la velocidad e incluso el sesgo. El sesgo del acelerómetro se ignora en la inicialización debido a que está acoplado con la gravedad, cuyo valor es mucho mayor y provoca que el sesgo sea difícil de observar.

En primer lugar, se calibra el sesgo del giroscopio, minimizando la función (3.7). En esta ecuación se puede observar cómo se utilizan tanto las estimaciones visuales obtenidas de SfM ($\mathbf{q}_{b_{k+1}}^{c_0}{}^{-1}$,$\mathbf{q}_{b_k}^{c_0}$), como el término incremental de la preintegración de la IMU de orientación ($\hat{\boldsymbol{\gamma}}_{b_{k+1}}^{b_k}$).

$$\begin{aligned} &\min_{\delta b_w} \sum_{k\in\mathcal{B}} \left\| \mathbf{q}_{b_{k+1}}^{c_0}{}^{-1} \otimes \mathbf{q}_{b_k}^{c_0} \otimes \boldsymbol{\gamma}_{b_{k+1}}^{b_k} \right\|^2 \\ &\boldsymbol{\gamma}_{b_{k+1}}^{b_k} \approx \hat{\boldsymbol{\gamma}}_{b_{k+1}}^{b_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\mathbf{J}_{b_w}^{\gamma}\delta\mathbf{b}_w \end{bmatrix} \end{aligned} \tag{3.7}$$

A partir de esta estimación, con las ecuaciones de preintegración de la IMU de la ec. (3.5), se vuelven a propagar las variables incrementales α, β, y γ, que como se comentó anteriormente dependen de los sesgos de la IMU.

Una vez finalizada la inicialización del sesgo el giroscopio, se procede a inicializar los estados de las velocidades, gravedad y escala métrica:

$$X_I = \left[\mathbf{v}_{b_0}^{b_0}, \mathbf{v}_{b_1}^{b_1}, \cdots \mathbf{v}_{b_n}^{b_n}, \mathbf{g}^{c_0}, s\right] \tag{3.8}$$

$\mathbf{v}_{b_k}^{b_k}$ es la velocidad respecto el marco del cuerpo mientras se toma la imagen k^{th}, $\mathbf{g}^{c_0}$ es el vector de gravedad respecto el marco $\{C_0\}$, y s es la escala métrica.

A partir de las variables del vector contenido en (3.8), se plantea la ecuación (3.9), donde los términos incrementales $\boldsymbol{\alpha}_{b_{k+1}}^{b_k}$ y $\boldsymbol{\beta}_{b_{k+1}}^{b_k}$ se expresan en función de la información obtenida de la visión. Las variables a calcular son las velocidades respecto al marco del cuerpo, la gravedad respecto $\{C_0\}$ y el parámetro de escala (variables del vector de estados). Dichos valores son estimados al sustituir los valores conocidos de la información obtenida de la SfM ($\mathbf{R}_{b_k}^{c_0}$, $\mathbf{R}_{b_{k+1}}^{c_0}$,$\bar{\mathbf{p}}_{b_k}^{c_0}$, $\bar{\mathbf{p}}_{b_k+1}^{c_0}$) y al sustituir $\boldsymbol{\alpha}_{b_{k+1}}^{b_k}$ y $\boldsymbol{\beta}_{b_{k+1}}^{b_k}$ por sus valores actualizados. Δt_k es el intervalo de tiempo entre dos frames consecutivos.

$$\begin{aligned}
\boldsymbol{\alpha}_{b_{k+1}}^{b_k} &= \mathbf{R}_{c_0}^{b_k}(s(\bar{\mathbf{p}}_{b_{k+1}}^{c_0} - \bar{\mathbf{p}}_{b_k}^{c_0}) + \frac{1}{2}\mathbf{g}^{c_0}\Delta t_k^2 - \mathbf{R}_{b_k}^{c_0}\mathbf{v}_{b_k}^{b_k}\Delta t_k) \\
\boldsymbol{\beta}_{b_{k+1}}^{b_k} &= \mathbf{R}_{c_0}^{b_k}(\mathbf{R}_{b_{k+1}}^{c_0}\mathbf{v}_{b_{k+1}}^{b_{k+1}} + \mathbf{g}^{c_0}\Delta t_k - \mathbf{R}_{b_k}^{c_0}\mathbf{v}_{b_k}^{b_k}).
\end{aligned} \tag{3.9}$$

Una vez conocidas $\mathbf{v}_{b_n}^{b_n}, \mathbf{g}^{c_0}$ y s, el siguiente objetivo es mejorar el vector de gravedad obtenido. Un vector en el espacio tridimensional depende de su magnitud y proyección en los tres ejes. Como la magnitud de la gravedad es conocida, el número de grados de libertad de la gravedad se reduce a 2.

Tras este ajuste, al calcular el vector de la gravedad $\mathbf{g}^{c_0}$, se puede obtener la rotación $\mathbf{q}_{c_0}^{w}$ entre el marco de la cámara y el mundo y, por tanto, transformar todas las variables y velocidades del sistema de referencia de la cámara $\{C_0\}$ al del mundo $\{W\}$. Las medidas visuales también serán escalados a la escala métrica. Todos estos valores serán necesarios para la siguiente etapa de VIO.

3.3 VIO monocular estrechamente acoplado

El vector de estados completo de la ventana deslizante en este VIO es:

$$\begin{aligned}
\mathscr{X} &= \left[\mathbf{x}_0, \mathbf{x}_1, \ldots \mathbf{x}_n, \mathbf{x}_c^b, \lambda_0, \lambda_1, \ldots \lambda_m\right] \\
\mathbf{x}_k &= \left[\mathbf{p}_{b_k}^w, \mathbf{v}_{b_k}^w, \mathbf{q}_{b_k}^w, \mathbf{b}_a, \mathbf{b}_g\right], k \in [0,n] \\
\mathbf{x}_c^b &= \left[\mathbf{p}_c^b, \mathbf{q}_c^b\right]
\end{aligned} \tag{3.10}$$

El vector de estados $\mathscr{X}$ contiene los estados de la IMU cuando se captura el frame b_k ($\mathbf{x}_n$), la posición y orientación de la cámara respecto a la IMU ($\mathbf{x}_c^b$), y la inversa de la profundidad de la l^{th} característica, vista desde su primera observación (λ_l). En primer lugar, los estados de la IMU tienen información sobre la posición , velocidad, rotación y sesgos de dicho sensor. Por otro lado, se utiliza la inversa de la distancia para proporcionar mayor estabilidad. Además, n es el número total de *keyframes* y m se corresponde con el número total de características en la ventana deslizante que han sido observadas al menos dos veces.

Las observaciones se pueden medir con los sensores (la cámara e IMU), las cuales se suponen con ruido. Se tiene un modelo de observación que, a partir del estado, proporciona las observacioness

ideales, el cual aparece en la ecuación (3.11). Por otro lado, las observaciones reales se denotan como $\hat{z}$.

$$z = h(\mathscr{X}) \tag{3.11}$$

Por un lado, las observaciones del sensor IMU deben medir información sobre la posición, velocidad, orientación, y sesgos en la aceleración y en la orientación. Estos datos se plantean de forma incremental entre 2 *frames*. El incremento de sesgo no se puede medir por su naturaleza, de forma que se fuerza su incremento a cero.

$$\begin{bmatrix} \hat{\boldsymbol{\alpha}}_{b_{k+1}}^{b_k} \\ \hat{\boldsymbol{\beta}}_{b_{k+1}}^{b_k} \\ \hat{\boldsymbol{\gamma}}_{b_{k+1}}^{b_k} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{R}_w^{b_k}(\mathbf{p}_{b_{k+1}}^w - \mathbf{p}_{b_k}^w + \frac{1}{2}\mathbf{g}^w\Delta t_k^2 - \mathbf{v}_{b_k}^w\Delta t_k) \\ \mathbf{R}_w^{b_k}(\mathbf{v}_{b_{k+1}}^w + \mathbf{g}^w\Delta t_k - \mathbf{v}_{b_k}^w) \\ \mathbf{q}_{b_k}^{w^{-1}} \otimes \mathbf{q}_{b_{k+1}}^w \\ \mathbf{b}_{ab_{k+1}} - \mathbf{b}_{ab_k} \\ \mathbf{b}_{wb_{k+1}} - \mathbf{b}_{wb_k} \end{bmatrix}. \tag{3.12}$$

Por otro lado, las observaciones de la cámara consisten en la posición observada de las características en las imágenes, las cuales se usan como referencia visual. $\hat{\mathbf{z}}_l^{c_i}$ es la observación de una característica por primer vez en el frame i, mientras que $\hat{\mathbf{z}}_l^{c_j}$ es la observación de dicha característica en un frame posterior j.

$$\begin{aligned} \hat{\mathbf{z}}_l^{c_j} &= \begin{bmatrix} \hat{u}_l^{c_j} \\ \hat{v}_l^{c_j} \end{bmatrix} \\ \hat{\mathbf{z}}_l^{c_i} &= \begin{bmatrix} \hat{u}_l^{c_i} \\ \hat{v}_l^{c_i} \end{bmatrix} \end{aligned} \tag{3.13}$$

La obtención de las medidas tanto de la IMU como de la cámara se explicó detalladamente en el apartado 3.1.

También se debe mencionar qué son los residuos. Estos son la diferencia entre la observación real y la ideal (es decir, el valor predicho por el modelo de observación).

$$\mathbf{z} = \hat{\mathbf{z}} + \delta\mathbf{z} \Rightarrow \delta\mathbf{z} = \mathbf{z} - \hat{\mathbf{z}} = \mathbf{h}(\mathscr{X}) - \hat{\mathbf{z}} \tag{3.14}$$

Generalmente, los residuos, tanto de la IMU como de la cámara se suelen denotar de forma genérica como r y siguen una distribución normal, de media nula y de covarianza P (matriz de covarianza del residuo).

$$r \sim N(\phi, P) \tag{3.15}$$

$$P(r) = \frac{1}{\sqrt{(2\pi)^N ||P||}} e^{-\frac{1}{2}(r-\phi)^T P^{-1}(r-\phi)} \tag{3.16}$$

La maximización de esta probabilidad es equivalente a la minimización de la distancia de Mahalanobis correspondiente. El problema de minimización se plantea como un ajuste visuo-inercial (*visual-inertial bundle adjustment*).

$$maxP(r) \sim min||r||_P^2 = min||h(X) - \hat{z}||_P^2 \tag{3.17}$$

El problema de estimación también se puede expresar en términos probabilísticos, haciendo referencia a algunas de las siguientes funciones de probabilidad:

- P(Z | X): Probabilidad condicionada de la observación. Probabilidad de observar el valor z en el vector de observación, supuesto un estado X.

- P(X | Z): Probabilidad a posteriori del estado. Probabilidad de que el estado sea X, a la vista de un observación.

- P(X): Probabilidad del estado a priori. Probabilidad de que el vector de que el vector de estados tenga un cierto valor.

Existen varias formas de estimar el estado: MLE (*Maximum Likehood Estimation*) o MAP (*Maximum A posteriori Estimation*). La última mencionada es la que utiliza VINS-Mono y, por tanto, la estimación del estado es igual que el cálculo de la distribución de probabilidad condicionada a partir de observaciones conocidas.

$$P(X|Z) \tag{3.18}$$

Atendiendo al teorema de Bayes:

$$P(X|Z) = \frac{P(Z|X) \cdot P(X)}{P(Z)} \propto P(Z|X) \cdot P(X) \tag{3.19}$$

En VINS-Mono, la información a priori procede de la inicialización, y durante el funcionamiento posterior, del resultado de la marginalización. La incertidumbre de las medidas se consideran como una distribución gaussiana $\mathbf{z} \sim \mathbf{N}(\bar{\mathbf{z}},\mathbf{Q})$. Por lo tanto,

$$X^* = min_{\mathscr{X}} \{P(z|x)\} = min_{\mathscr{X}} \left\{\sum ||z - h(x)||_Q\right\} \tag{3.20}$$

Finalmente se llega a que para la estimación del vector (3.10), se minimiza la siguiente función. Se utiliza un ajuste visuo-inercial para obtener la mejor estimación posible:

$$\begin{aligned} \min_{\mathscr{X}} \Bigg\{ & \left\|\mathbf{r}_p - \mathbf{H}_p \mathscr{X}\right\|^2 + \sum_{k \in \mathscr{B}} \left\|\mathbf{r}_{\mathscr{B}}(\hat{\mathbf{z}}_{b_{k+1}}^{b_k}, \mathscr{X})\right\|^2_{\mathbf{P}_{b_{k+1}}^{b_k}} \\ & + \sum_{(l,j) \in \mathscr{C}} \rho\left(\left\|\mathbf{r}_{\mathscr{C}}(\hat{\mathbf{z}}_l^{c_j}, \mathscr{X})\right\|^2_{\mathbf{P}_l^{c_j}}\right) \Bigg\} \end{aligned} \tag{3.21}$$

donde la norma de Huber es definida como

$$\rho(s) = \begin{cases} s & s \leq 1 \\ 2\sqrt{s} - 1 & s > 1. \end{cases} \tag{3.22}$$

$\mathscr{B}$ es el conjunto de todas las medidas de la IMU, y $\mathscr{C}$ es el conjunto de las características que se han observado al menos dos veces en la ventana actual. $\mathbf{r}_p$ y $\mathbf{H}_p$ es la información previa de marginalización.

En la ecuación (3.21) aparecen tanto los residuos inerciales como los visuales. En el caso de los residuos inerciales tendríamos:

$$\begin{aligned}\mathbf{r}_{\mathscr{B}}(\hat{\mathbf{z}}^{b_k}_{b_{k+1}}, \mathscr{X}) &= \begin{bmatrix} \delta\boldsymbol{\alpha}^{b_k}_{b_{k+1}} \\ \delta\boldsymbol{\beta}^{b_k}_{b_{k+1}} \\ \delta\boldsymbol{\theta}^{b_k}_{b_{k+1}} \\ \delta\mathbf{b}_a \\ \delta\mathbf{b}_g \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{R}^{b_k}_w(\mathbf{p}^w_{b_{k+1}} - \mathbf{p}^w_{b_k} + \frac{1}{2}\mathbf{g}^w\Delta t_k^2 - \mathbf{v}^w_{b_k}\Delta t_k) - \hat{\boldsymbol{\alpha}}^{b_k}_{b_{k+1}} \\ \mathbf{R}^{b_k}_w(\mathbf{v}^w_{b_{k+1}} + \mathbf{g}^w\Delta t_k - \mathbf{v}^w_{b_k}) - \hat{\boldsymbol{\beta}}^{b_k}_{b_{k+1}} \\ 2\left[\ \mathbf{q}^{w^{-1}}_{b_k} \otimes \mathbf{q}^w_{b_{k+1}} \otimes (\hat{\boldsymbol{\gamma}}^{b_k}_{b_{k+1}})^{-1}\ \right]_{xyz} \\ \mathbf{b}_{ab_{k+1}} - \mathbf{b}_{ab_k} \\ \mathbf{b}_{wb_{k+1}} - \mathbf{b}_{wb_k} \end{bmatrix}\end{aligned} \tag{3.23}$$

Los residuos visuales tienen la función de minimizar el error en la estimación de la posición de una misma característica. Se define la medida residual de la cámara en una esfera unidad. La óptica de la mayoría de cámaras se puede modelar como un rayo unidad conectando la superficie de una esfera unidad. Estos residuos visuales consideran las coordenadas de la característica l^{th} en el primer frame en el que se detectó, y las coordenadas en la imagen posterior en la que se sigue observando dicha caracterítica.

$$\begin{aligned}\mathbf{r}_{\mathscr{C}}(\hat{\mathbf{z}}^{c_j}_l, \mathscr{X}) &= \begin{bmatrix}\ \mathbf{b}_1\ \ \mathbf{b}_2\ \end{bmatrix}^T \cdot \left(\hat{\bar{\mathscr{P}}}^{c_j}_l - \frac{\mathscr{P}^{c_j}_l}{\|\mathscr{P}^{c_j}_l\|}\right) \\ \hat{\bar{\mathscr{P}}}^{c_j}_l &= {\pi_c}^{-1}\left(\begin{bmatrix} \hat{u}^{c_j}_l \\ \hat{v}^{c_j}_l \end{bmatrix}\right) \\ \mathscr{P}^{c_j}_l &= \mathbf{R}^c_b\left(\mathbf{R}^{b_j}_w\left(\mathbf{R}^w_{b_i}\left(\mathbf{R}^b_c\frac{1}{\lambda_l}{\pi_c}^{-1}\left(\begin{bmatrix} \hat{u}^{c_i}_l \\ \hat{v}^{c_i}_l \end{bmatrix}\right)\right.\right.\right. \\ &\qquad \left.\left.\left. + \mathbf{p}^b_c\right) + \mathbf{p}^w_{b_i} - \mathbf{p}^w_{b_j}\right) - \mathbf{p}^b_c\right)\end{aligned} \tag{3.24}$$

La función π_c^{-1} transforma la localización de un píxel en un vector unidad a partir de los parámetros intrínsecos. Como el vector residual tiene dos grados de libertad, se puede proyectar en el plano tangente (figura 3.5). Durante esta etapa, la distancia inversa de las características, la posición y velocidad de cada frame, los parámetros extrínsecos y los sesgos de la IMU se optimizan juntos.

3.3.1 Marginalización

La marginalización es un procedimiento que fue originalmente desarrollado para odometría visuo-inercial [22], y se encarga de limitar la complejidad computacional. Esto es debido a que esta complejidad aumenta cuando se observan entornos desconocidos y, por tanto, el vector de estados tiende a crecer. El resultado propuesto es reducir este coste marginalizando parte de los estados previos.

El procedimiento de la figura 3.6 se basa en mirar el penúltimo fotograma clave de la ventana:

- Si es un fotograma clave, se mantiene en la ventana y el fotograma más antiguo de la ventana se marginaliza (es decir, se desecha).

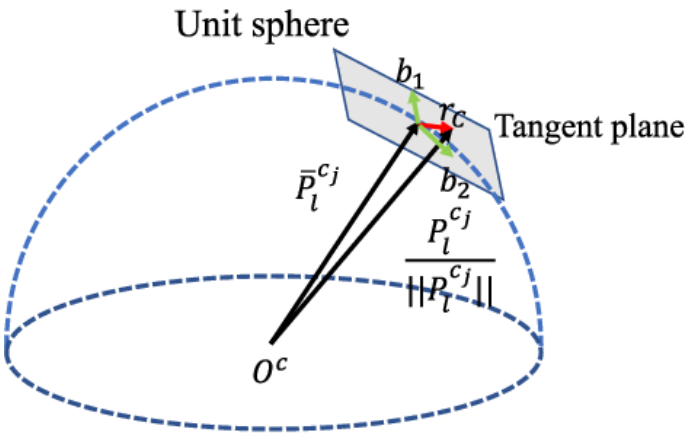


Figura 3.5 Imagen del residuo visual representado en una esfera unidad.

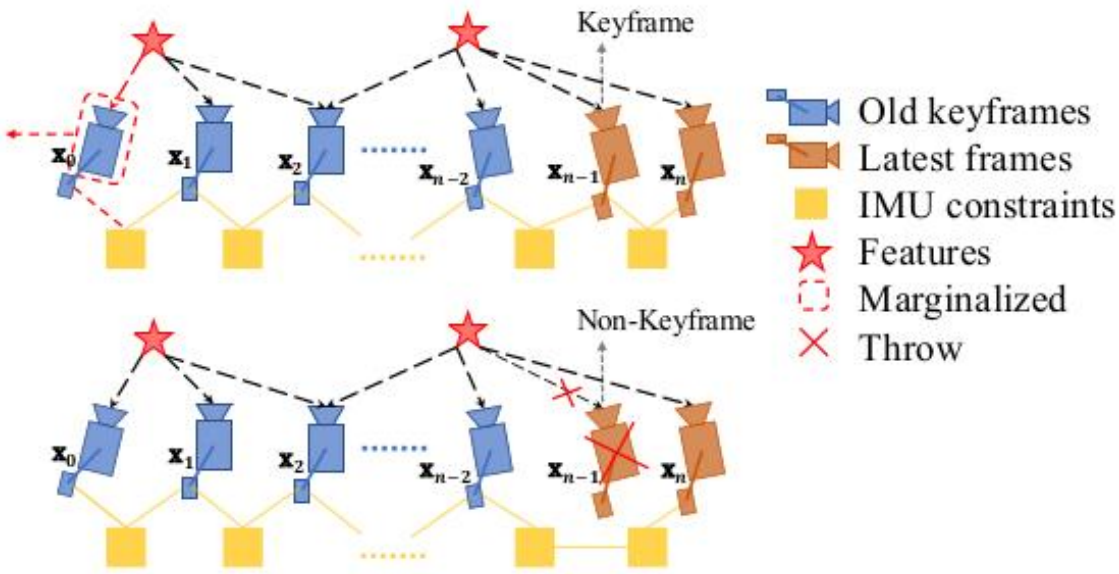


Figura 3.6 Procedimiento de marginalización.

- Por otro lado, si no es un fotograma clave, se elimina el fotograma y sus medidas visuales, pero se mantienen las medidas inerciales.

Otra función de este paso es mantener espaciados los fotogramas clave de la ventana: por un lado, asegura que tienen suficiente paralaje para la triangulación de características, y por otro lado, maximiza la probabilidad de mantener las medidas del acelerómetro con mayor excitación. Con la marginalización se construyen unas nuevas condiciones a priori del problema, basadas en las medidas marginalizadas del estado eliminado.

3.3.2 Detección de fallos y recuperación

VIO es muy robusto ante cambios de entornos. Posee un módulo de detección de fallos que observa que las salidas del estimador no sean inusuales. Se considera que existe fallo cuando el número de caracteríticas en común con el frame anterior está por debajo de un límite, cuando existe un vacío en rotación o traslación en la salida del esimador, o cuando hay un cambio muy grande en la estimación

de los parámetros extrínsecos o sesgos. Es inevitable que se produzcan fallos en el sistema, pero se pueden identificar y corregir. Por ello, VINS-Mono vuelve a la fase de inicialización y vuelve a crear un nuevo segmento para la trayectoria seguida.

3.4 Relocalización

La complejidad computacional se limita gracias a la ventana deslizante y a la marginalización, pero esto también provoca que se acumulen derivas. Estas derivas tienen lugar en la posición respecto a los tres ejes (x,y, z) y en la rotación alrededor de la dirección de la gravedad (*yaw*). Para eliminarlas se utiliza este módulo de relocalización. La observación repetida de las mismas características es utilizada en la relocalización, mejorando la estimación y la exactitud. El procedimiento seguido es el siguiente:

- **Detección de lazos:**
  Se utiliza DBoW2 [13], que es una librería mejorada de DBoW, la cual implementa un árbol jeráquico para aproximar vecinos cercanos en el espacio de características de la imagen.

  Para aumentar la tasa de recuperación por cierre de lazo se detectan 500 esquinas más con el descriptor BRIEF [9]. Se mantienen todos los descriptores de las características pero sin guardar las imágenes para ahorrar memoria.

- **Recuperación de características:**
  Cuando se detecta un bucle, se establece la conexión entre la ventana actual y el candidato de cierre de bucle. Las correspondencias de características son encontradas por el descriptor BRIEF, pero pueden existir valores atípicos. Existen diversos pasos para eliminarlos, y tras este procedimiento si el número de *inliers* es mayor que el límite establecido, se considera un bucle y se realiza la relocalización.

- **Relocalización estrechamente acoplada:**
  En este paso, se alinea la ventana actual con las poses pasadas. La optimización de la ventana deslizante se realiza utilizando medidas de la IMU, medidas visuales y correspondencias del bucle en el cierre de lazo. Cuando se establecen muchos cierres de bucles en la ventana actual, se optimizan las correspondencias de las características de cierre de bucle de todos los frames a la vez [18].

3.5 Optimización de grafo de poses global

Hay que resaltar que la optimización y la relocalización se llevan a cabo en diferentes hilos de forma asíncrona: esto permite utilizar en la relocalización la optimización del grafo de poses en el instante en el que esté disponible.

Gracias a la medida inercial de la gravedad, los ángulos *pitch* y *roll* son totalmente observables. Estos son estados absolutos respecto el marco del mundo, mientras que x,y,z y *yaw* son estimaciones relativas respecto el marco de referencia. La deriva acumulada solo ocurre en estos 4 últimos estados y la optimización sólo se realiza de estos 4 grados de libertad.

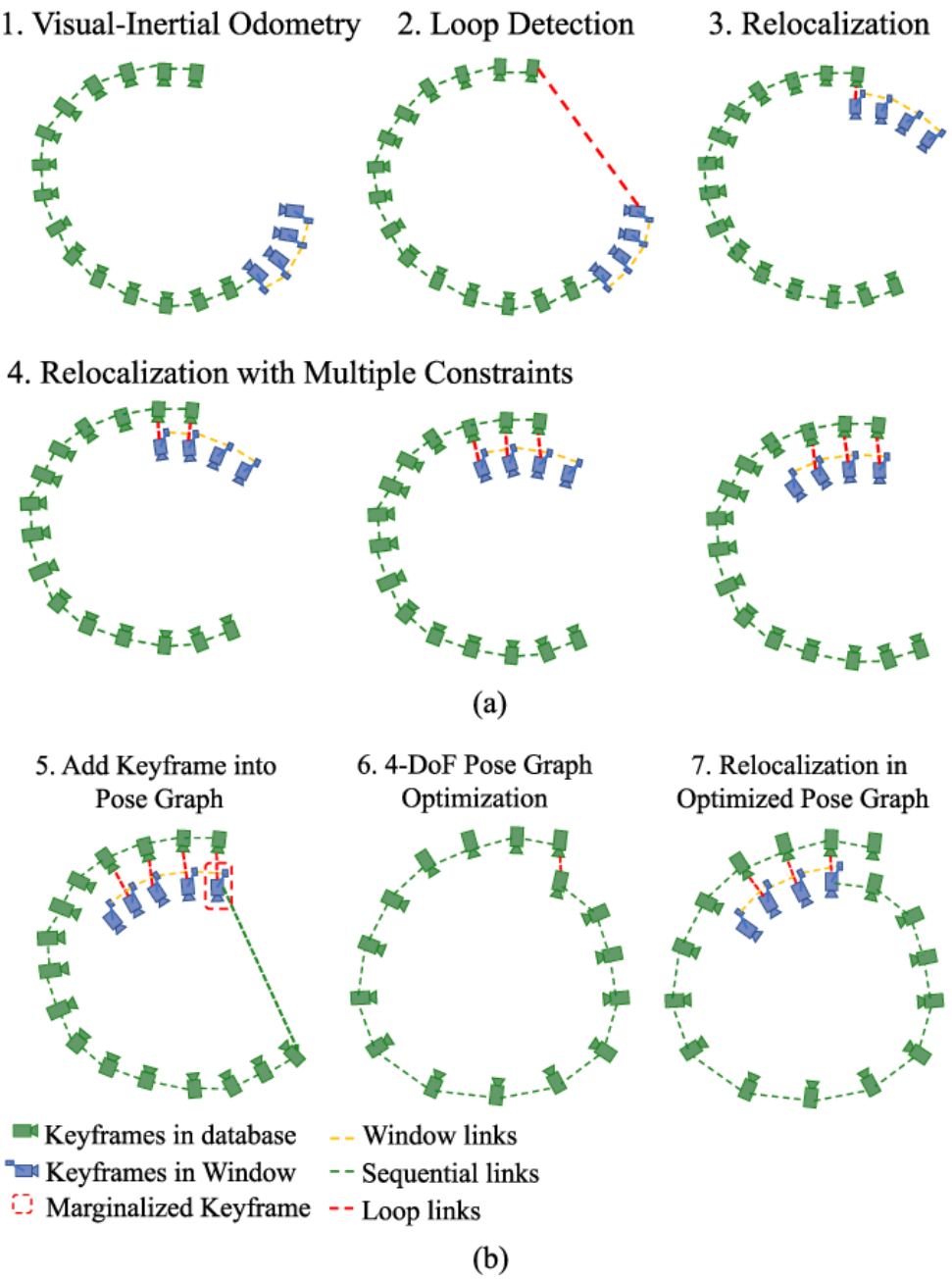


Figura 3.7 Pasos seguidos por el módulo de relocalización.

$$\begin{aligned}\hat{\mathbf{p}}_{ij}^{i} &= \hat{\mathbf{R}}_{i}^{w^{-1}}(\hat{\mathbf{p}}_{j}^{w} - \hat{\mathbf{p}}_{i}^{w}) \\ \hat{\psi}_{ij} &= \hat{\psi}_{j} - \hat{\psi}_{i}.\end{aligned} \tag{3.24}$$

3.5.1 Fusionar grafo de poses

El grafo de poses optimiza el mapa actual y también es capaz de fusionar el mapa actual con un mapa anterior ya construido y guardado. Si hemos cargado un mapa anterior y se detectan conexiones antre ambos, se puede producir una fusión.

El tamaño del grafo de poses puede crecer sin límites cuando la distancia del viaje aumenta,

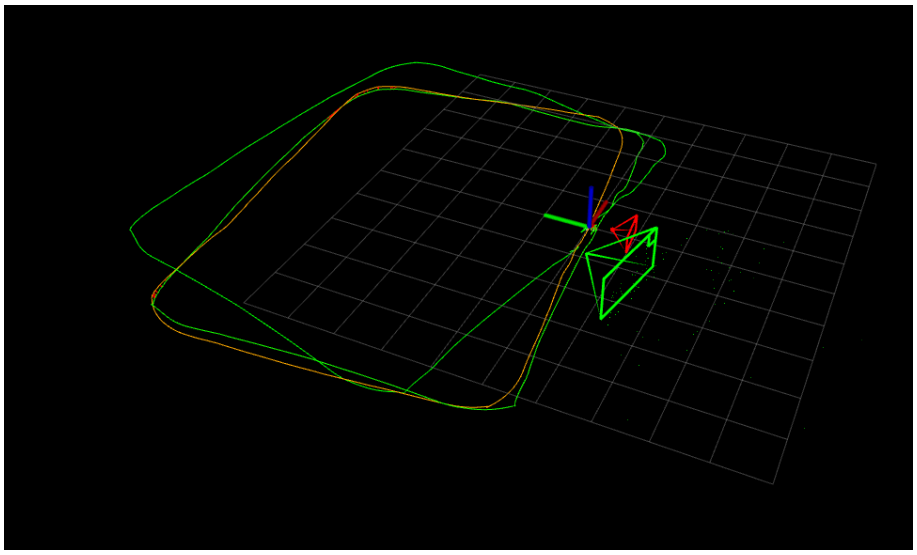


Figura 3.8 Trayectoria estimada del vehículo terrestre con algoritmo VINS-Mono al al fusionarla con una trayectoria guardada de un experimento anterior.

limitando el tiempo real del sistema a largo plazo. Por tanto, se implementa un proceso de reducción de la muestra: todos los *keyframes* con restricciones de cierre de lazo se mantienen, pero sus vecinos que se encuentren muy cerca o tengan orientaciones muy parecidas se eliminan.

4 Herramientas utilizadas

El El algoritmo VINS-Mono se ejecuta en Linux y está totalmente integrado en ROS. En este capítulo se van a recoger todos los programas necesarios en la instalación del algoritmo, y para demostrar su buen funcionamiento.

4.1 Prerrequisitos

Se va a hacer una breve descripción de todas las librerías y programas que es necesario tener antes de proceder a la instalación del paquete VINS-Mono.

- **Ubuntu 16.04**
  Ubuntu es un sistema operativo de software libre y código abierto, basado en GNU/Linux y creado en 2004. Presenta una interfaz fuy fácil de manejar y tiene diferentes tipos de instalación:
  - Partición de disco duro: se instala en el espacio libre, conservando el resto de sistemas instalados.
  - Disco duro total: elimina todas sus particiones y su contenido, creando una sóla partición (en la cual se instalará).
  - Aplicación Windows: se instala como un programa más del sistema operativo, en una máquina virtual.
- **ROS Kinetic**
  Robotic Operating System (ROS) [5] es una colección de *frameworks* para el desarrollo de robots. Aunque ROS no es un sistema operativo como tal, proporciona servicios como la abstracción del hardware, control de dispositivos de bajo nivel, implementación de funcionalidades de uso común, comunicación entre procesos y administración de paquetes.

ROS tiene dos partes básicas: ros y ros-pkg. La primera es la parte del sistema operativo, y la segunda es el conjunto de paquetes aportados por la contribución de usuarios. ROS es software libre y tiene libertad para uso comercial e investigador.

Existen varias distribuciones de ROS, entre las cuales las más destacadas son Melodic, Indigo, Hydro...En este proyecto se ha utilizado ROS Kinetic, versión instalada en Ubuntu 16.04.

- **Ceres Solver**
  Se trata de una librería de C++ de código abierto para modelar y resolver complicados problemas de optimización. Es una biblioteca rica en funciones y de alto rendimiento que se usa desde el año 2010 [6].

- **OpenCV 3.3.1**
  OpenCV es una librería software de código abierto (*open-source*) de visión artificial y *machine learning*. Dispone de infinitud de algoritmos que permiten identificar rostros, reconocer objetos, identificarlos...

  Tiene una licencia BSD que permite utilizar y modificar código. OpenCV está escrito en C++ y tiene interfaces en C++, C, Python, Java y *Matlab inferfaces*. Es una librería multiplatafoma: funciona en Windows, Linux, Android y Mac.

  Es necesario descargar la versión de OpenCV 3.3.1 o mayor.

- **Eigen 3.3.3**
  Eigen [15] es una biblioteca de plantillas de C++ para álgebra lineal: matrices, vectores, solucionadores numéricos y algoritmos relacionados.

4.1.1 Instalar paquete VINS-Mono

Una vez se encuentran instaladas todas las dependencias necesarias y especificadas en el apartado 4.1, se procede a clonar el repositorio y hacer *catkin_make*.

Código 4.1 Comandos necesarios para instalación de paquete VINS-Mono.

```
cd ~/<nombre_ws>/src
git clone https://github.com/HKUST-Aerial-Robotics/VINS-Mono.git
cd ..
catkin_make
source ~/<nombre_ws>/devel/setup.bash
```

Ya está todo preparado para poder lanzar el algoritmo VINS-Mono. A continuación, se va a describir un poco más en detalle las herramientas de ROS y Gazebo, que son las más usadas e influyentes en el desarrollo de este proyecto, tanto para el algoritmo como para la simulación.

4.2 ROS

Como se comentó anteriormente, ROS es una plataforma software para el desarrollo de software específico para robótica. Su filosofía es la de que todos los programas sean de uso libre, reutilizables y escalables.

ROS también proporciona servicios estándares propios de un sistema operativo pero sin serlo, ya que se instala sobre otro, en general Linux.

4.2.1 Conceptos basicos

Nivel del sistema de archivos

- **Paquetes**: Son la unidad principal de organización de software en ROS. Un paquete puede contener nodos, librerías, *datasets*, archivos de configuración. Lo más pequeño que se puede construir es un paquete.
- **Manifiestos del paquete (*package.xml*)**: Proporciona información sobre un paquete: su nombre, versión, descripción, información sobre la licencia, dependencias, y otros datos.
- **Tipo de mensajes**: Permite definir estructuras de datos para mensajes enviados en ROS.
- **Tipos de servicios**: Permite definir la estructura de datos tanto de la petición como de la respuesta de los servicios de ROS.

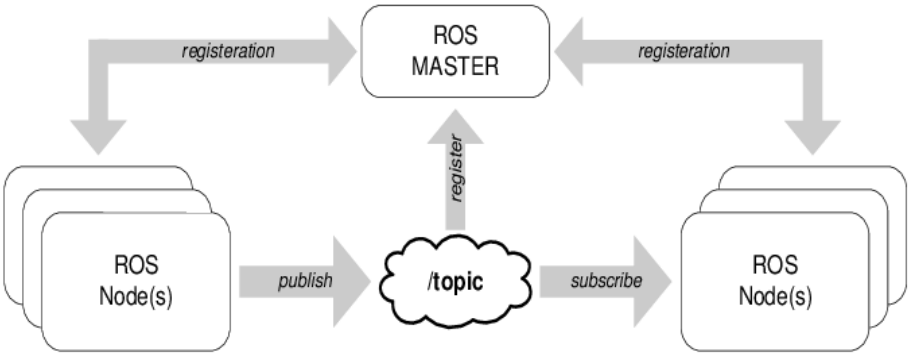


Figura 4.1 Comunicación en el entorno ROS.

Nivel gráfico de computación

- **Nodos:**
  Son archivos ejecutables dentro de un paquete de ROS. Estos realizan funciones concretas: acceso a un sensor, un motor...Un nodo puede publicar o suscribirse a un tópico, y puede utilizar o proporcionar un servicio.
  La principal ventaja es que tienen un diseño modular y son independientes del resto de nodos.
- **Mensajes**:
  Los mensajes son estructuras de datos. Los nodos se comunican entre ellos publicando mensajes en los tópicos.
- **Tópicos**:
  Son usados por los nodos para intercambiar mensajes. Los nodos no conocen realmente con qué nodo se están comunicando: unos se suscriben a los tópicos, y otros publican información en estos. Por tanto, la estructura de comunicación es la siguiente:
  1. Un nodo publica un mensaje en un tópico.
  2. Uno o varios nodos se subscriben a dicho tópico para recibir la información.
- **Servicio:**
  Es un método de comunicación síncrona entre dos nodos. Son definidos por un par de estructuras de datos: una para la petición y otra para la respuesta.

- **Bolsas (*bags*):**
  Son un formato para guardar y reproducir datos de mensajes de ROS. Son un mecanismo importante para guardar datos.
- **ROS Master:**
  Es un servicio de declaración y registro de nodos. Sin el Master, los nodos no podrían encontrarse entre ellos, intercambiar mensajes o invocar servicios.

  El ROS Maestro registra todos los nodos, servicios y tópicos que se encuentran en ejecución. Los nodos se comunican con el Maestro para reportar su información de registro. Los nodos se comunican directamente con otros nodos (el Maestro sólo proporciona información de búsqueda).

4.2.2 Herramientas básicas

Algunos de los comandos más utilizados en ROS son los siguientes:

- ***rosbag***: Es un paquete que proporciona una herramienta para línea de comandos que trabaja con bolsas de datos. Permite guardar información de los tópicos de un robot para reproducirla en otro momento.
- ***roscd***: Permite ir directamente a un paquete sin tener que conocer el directorio donde se encuentre.
- ***roscore***: Debe estar ejecutándose para que los nodos de ROS se puedan comunicar.
- ***rosrun***: Permite ejecutar un ejecutable de cualquier paquete sin tener que hacer *roscd* primero.
- ***roslaunch***: Lanza un conjunto de nodos.

4.2.3 Herramientas gráficas

Se han utilizado algunas herramientas de ROS para facilizar la visualización de las imágenes y de la trayectoria calculada por el algoritmo:

- **RViz:**
  Es una herramienta de visualización en 3D. Proporciona una vista del modelo del robot, captura información de los sensores y reproduce los datos. Puede mostrar datos de cámaras, láseres y dispositivos 2D y 3D.
  En este trabajo, se utiliza para observar la trayectoria estimada por VINS-Mono y, a la vez, los datos de los tópicos de ROS que se desee (las imágenes tomadas por la cámara, o las imágenes una vez se han extraído los puntos característicos...). Todo esto permite observar y estudiar información diferente al mismo tiempo.
- **Rqt_bag:**
  Es una aplicación que se encarga de grabar y gestionar las bolsas. Algunas de las características principales son: la publicación/grabación de mensajes de los tópicos que se indiquen y la reproducción del contenido de las bolsas (tanto datos numéricos como imágenes).
- **Rqt_graph:**
  Esta herramienta muestra un grafo de los procesos que se ejecutan en ROS y sus conexiones.

4.3 Gazebo

Gazebo es un simulador de entornos 3D que permite evaluar el comportamiento de un robot en un mundo virtual. Además, Gazebo se puede sincronizar con ROS de forma que los robots publiquen la información de sus sensores en nodos, o que reciban información como, por ejemplo, órdenes provenientes de un un sistema de control.

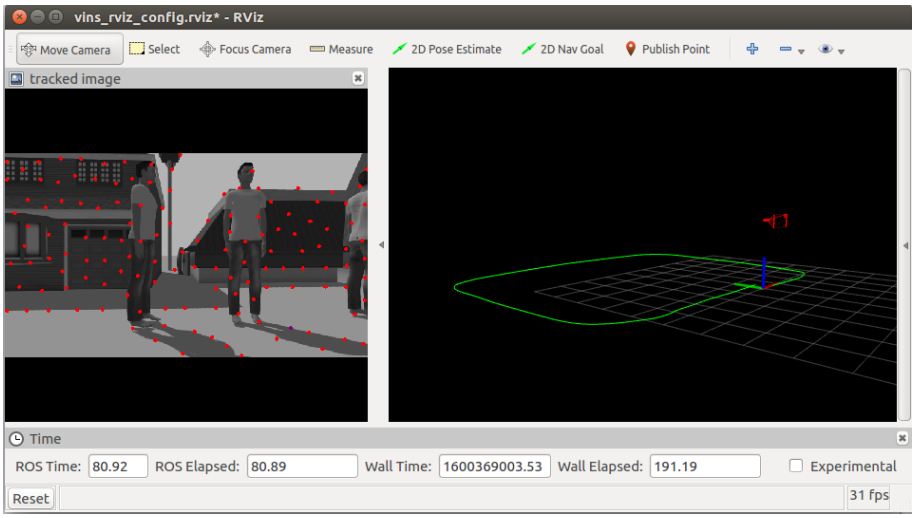


Figura 4.2 Herramienta RViz que muestra la trayectoria seguida por un robot en 3D.

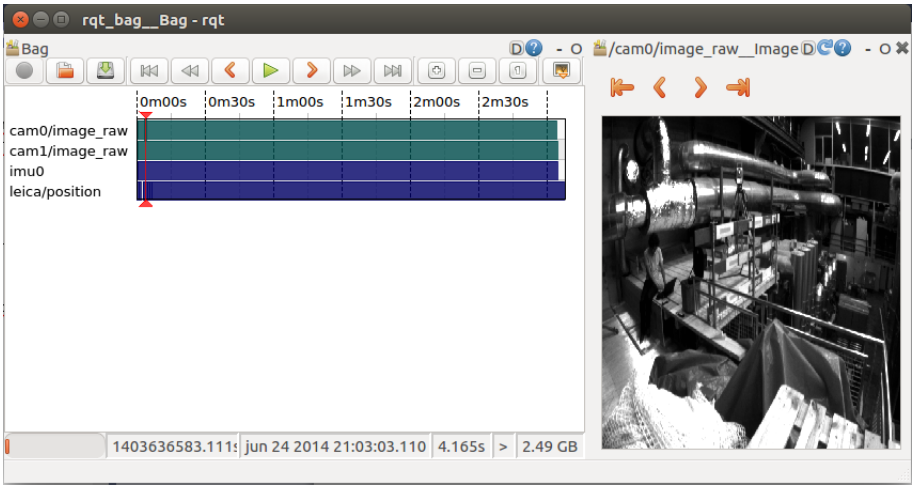


Figura 4.3 Herramienta *rqt_bag* al reproducir una *rosbag* que contiene datos de tipo imagen.

4.3.1 Componentes de Gazebo

Archivos del mundo

Este archivo contiene todos los elementos necesarios del entorno de una simulación, como los objetos estáticos, luces, sensores o robots. Utiliza formato SDF y normalmente tiene extensión *.world*. Gazebo ya tiene ejemplos creados de diferentes mundos.

Código 4.2 Ejemplo de un archivo del mundo.

```
<?xml version="1.0" ?>
<sdf version="1.4">
  <world name="default">
    <include>
      <uri>model://house_1</uri>
      <name>House1_0</name>
      <pose>8.32 4.7 0 0 0 -1.57</pose>
    </include>
```

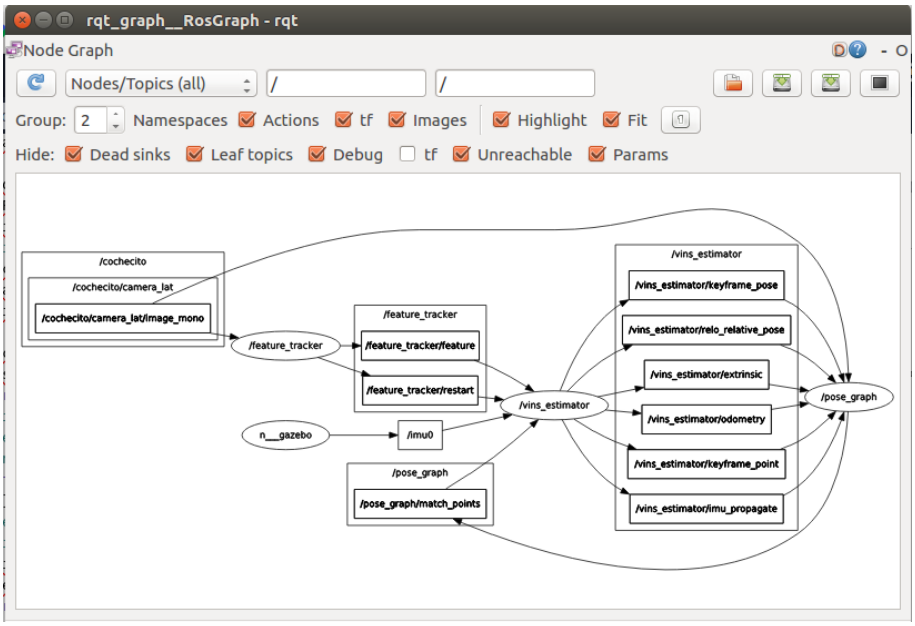
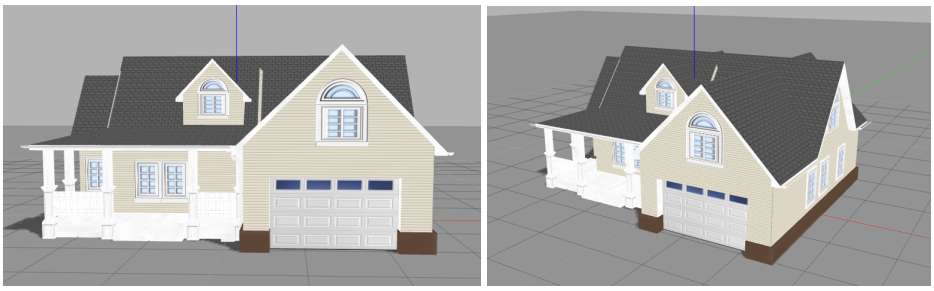


Figura 4.4 Herramienta *rqt_graph* que muestra conexiones entre nodos al ejecutar VINS-Mono.

```
    </world>
</sdf>
```

El código 4.2 es un ejemplo de un archivo con extensión *.world*. En él, se añade un modelo de una casa ya creado de Gazebo denominado *house_1*, y se ha indicado el nombre, la posición y la orientación deseada dentro de la simulación. Esto es lo que se ha hecho para crear el entorno de la simulación de Gazebo del último experimento realizado sobre VINS-Mono: se han añadido diferentes modelos ya creados como carreteras, farolas, semáforos, parques, casas... El código completo del archivo del mundo creado se puede encontrar en el código 7.1.



(a) Vista frontal casa. **(b)** Vista girada casa.

Figura 4.5 Diferentes vistas de la casa ya creada de Gazebo.

Archivos del modelo

Este tipo de archivos también tienen formato SDF. Cada archivo sólo contiene un modelo y una vez creado puede introducirse en el archivo del mundo. Esto hace que el modelo se pueda reutilizar y también simplifica el archivo del mundo. Al igual que existen ya ejemplos de mundos creados en Gazebo, también hay muchos modelos.

Los modelos se crean a partir de articulaciones, enlaces, *plugins*...Es necesario indicar todos estos elementos para poder crear el modelo. El modelo creado en este proyecto ha sido un vehículo de 4 ruedas (figura 4.6). El código completo necesario para su creación se puede ver en el anexo 7.2, donde se pueden observar todos los elementos que se deben especificar. Aquí solo se introduce la sección del código donde se modela la rueda delantera derecha, con el objetivo de explicar una parte del código más detalladamente.

El vehículo está compuesto por 4 ruedas, el chasis, una unidad de medida inercial y 2 cámaras (una frontal y una lateral, que es la usada para tomar las imágenes que se utilizarán en la estimación del algoritmo VINS-Mono). Para cada componente se ha de especificar el nombre, el elemento visual de colisión e inercial, y también la articulación que lo une a otro componente del modelo.

(a) Vista frontal vehículo. **(b)** Vista girada vehículo.

Figura 4.6 Diferentes vistas del vehículo creado en Gazebo.

El siguiente código contiene un ejemplo de los elementos que deben especificarse para cada componente del coche. Este, en concreto, modela la rueda delantera derecha (figura 4.7):

Código 4.3 Fragmento de código del modelo del vehículo. Rueda delantera derecha.

```
<!--Rueda derecha delantera-->
<link name="front_right_wheel">
    <visual>
      <origin rpy="1.57079632679 0 0 " xyz="0 0.15 0"/>
      <geometry>
        <mesh scale ="0.9 0.9 0.9" filename="model://car_wheel/meshes/
            car_wheel.dae"/>
      </geometry>
      <material name="DarkGray"/>
    </visual>
    <collision>
      <origin rpy="1.57079632679 0 0.2 " xyz="0 0.15 0"/>
      <geometry>
        <cylinder length="0.1" radius="0.2"/>
      </geometry>
    </collision>
    <inertial>
      <mass value="${wheel_mass}"/>
      <origin xyz="0 0.08 0"/>
```

```
    <inertia ixx="${0.25*wheel_mass*0.2*0.2}" ixy="0" ixz="0" iyy="$
        {0.5*wheel_mass*0.2*0.2}" iyz="0" izz="${0.25*wheel_mass
        *0.2*0.2}"/>
  </inertial>
</link>

<joint name="front_right_wheel_joint" type="continuous">
  <parent link="chassis"/>
  <child link="front_right_wheel"/>
  <origin rpy="0 0 0" xyz="0.25 0.3 -0.14"/>
  <axis rpy="0 0 0" xyz="0 1 0"/>
  <limit effort="100" velocity="100"/>
  <joint_properties damping="0.0" friction="0.0"/>
</joint>
```

Se pueden observar dos partes totalmente diferenciadas: la rueda y la articulación que une la rueda con el resto del vehículo, en contreto, con el chasis. A continuación, se va a hacer una breve explicación de los elementos del código más importantes:

- **Link:** Aquí se detallan las propiedades físicas del enlace en el modelo.
  - **name**: Se indica el nombre del enlace, el cual debe ser único.
  - **collision**: Se especifican las propiedades de colisión de un enlace. Estas pueden ser diferentes a las propiedades visuales.
  - **visual**: Las propiedades visuales del enlace (la posición y orientación de los ejes del enlace, y la forma del objeto).
    Se puede observar en el código que se utiliza un modelo 3D ya creado llamado *car_wheel.dae* en lugar de usar una figura simple como podría ser un cilindro para imitar la forma de una rueda.
  - **inertial**: Las propiedades inerciales del enlace (la masa, el origen del centro de masas y los elementos de la matriz de inercias).
- **Joint:** Se describe la articulación que va a unir dos componentes del modelo (en este caso se une la rueda y el chasis).
  - **name**: Nombre de la articulación, el cual debe ser único.
  - **type**: Se especifica el tipo de articulación del que se trata. En este caso es de tipo *continuous* porque rota respecto a un único eje con un rango continuo de movimiento.
  - **parent link** y **child link**: Se especifican los componentes que se unen a través de dicha articulación.
  - **axis**: Se indican los parámetros de los ejes de rotación o de los ejes de traslación, según sea una articulación de revolución o prismática. La rueda es una articulación de revolución, y rota alrededor del eje *y*.
  - **origin**: Es el desplazamiento y la rotación de los ejes del *child link* respecto *parent link*. Es decir, los ejes de las ruedas tienen igual rotación que los ejes del chasis pero tiene un cierto desplazamiento.

Existen muchos más elementos que se pueden definir a la hora de crear los componentes de un modelo, pero sólo he hecho un breve inciso sobre aquellos que se han utilizado en el modelo del vehículo y cuyo código se encuentra en su totalidad en 7.2.

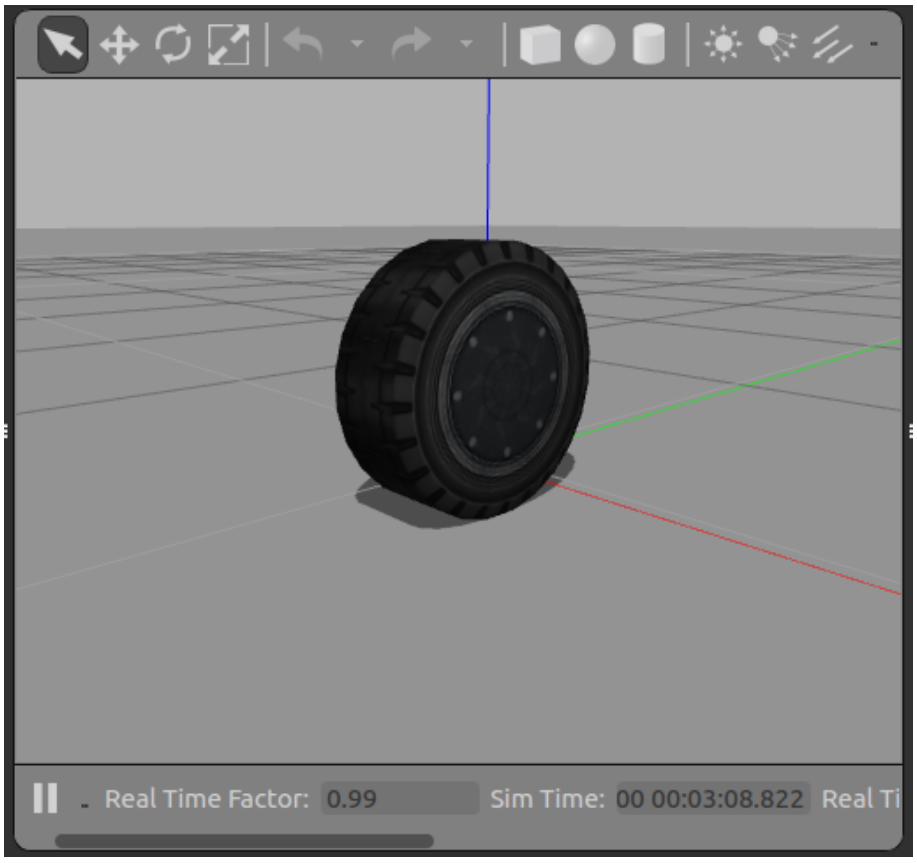


Figura 4.7 Modelo de una rueda de coche ya creado por Gazebo.

Plugins

Los *plugins* proporcionan un mecanismo simple para interactuar con Gazebo. Son muy útiles porque:

- Permiten a los desarrolladores controlar casi cualquier aspecto de Gazebo.
- Pueden ser insertados y eliminados de un sistema en ejecución.
- Son rutinas independientes que se comparten fácilmente.

Existen diferentes tipos: mundo, modelo, sensor, sistema, visual y GUI. Cada tipo es gestionado por un componente diferente de Gazebo y son elegidos según la funcionalidad deseada. En la simulación creada con Gazebo para el experimento de VINS-Mono se han utilizado *plugins* para los sensores que lleva incorporado el vehículo: la IMU y las dos cámaras. En ambas cámaras se ha utilizado el mismo *plugin*, con las mismas caracterísicas.

Código 4.4 Plugin de la cámara.

```
<gazebo reference="camera2_link">
   <sensor type="camera" name="camera_lat">
   <visualize>true</visualize>
     <update_rate>60</update_rate>
     <camera name="camera_lat">
       <horizontal_fov>1.413</horizontal_fov>
       <image>
         <width>640</width>
```

```
          <height>360</height>
          <format>R8G8B8</format>
        </image>

        <clip>
        <near>0.01</near>
            <far>100</far>
        </clip>
      </camera>
      <plugin name="camera_controller" filename="libgazebo_ros_camera.
         so">
      <cameraName>cochecito/camera_lat</cameraName>
        <alwaysOn>true</alwaysOn>
        <updateRate>60</updateRate>
        <imageTopicName>/cochecito/camera_lat/image_raw</imageTopicName
           >
        <cameraInfoTopicName>/cochecito/camera_lat/camera_info</
           cameraInfoTopicName>
        <frameName>cochecito_base_cam_lat</frameName>
      </plugin>
    </sensor>
    <turnGravityOff>true</turnGravityOff>
  </gazebo>
```

En primer lugar, se van a describir los diferentes parámetros que se definen en el fragmento de código anterior, que van a modelar la cámara lateral (es aquella que proporciona las imágenes que posteriormente se procesan para estimar la trayectoria del vehículo). Posteriormente, en el capítulo 5, se verá cómo será necesario ajustar en un fichero de configuración del algoritmo VINS-Mono los parámetros internos relacionados tanto con la cámara como con la IMU. Esto es necesario para que la trayectoria que estima sea buena. También será necesario conocer la posición relativa de un sensor respecto a otro (esto también se explicará más detenidamente en el capítulo 5, de simulaciones). Volviendo al *plugin* de la cámara, se va a describir brevemente el objetivo que persigue el código:

- **reference**: Se indica el enlace en el cual se instala el *plugin*.
- **type**: Tipo de sensor. En este caso, se trata de una cámara pero también puede ser GPS, IMU, LIDAR, altímetro, cámara 3D...
- **name**: Nombre que se le asigna al sensor, el cual debe ser único.
- **visualize**: Indica si el sensor es visualizado en la GUI. Cuando está fijado a *True*, en la simulación de Gazebo se verán las imágenes que capta la cámara. Se puede observar en la figura 4.8.
- **updateRate**: Se especifica la frecuencia a la que el sensor publica la información (en este caso, número de *frames* por segundo).
- **camera**: Se indican los elementos específicos de una cámara:
  - **name**: Nombre de la cámara.
  - **horizontal_fov**: Campo de visión o el ángulo abarcable por el sensor.
  - **image**: Ancho, alto y formato de las imágenes capturadas por la cámara.
  - **clip**: Distancia mínima y máxima a la que la cámara es capaz de captar objetos.

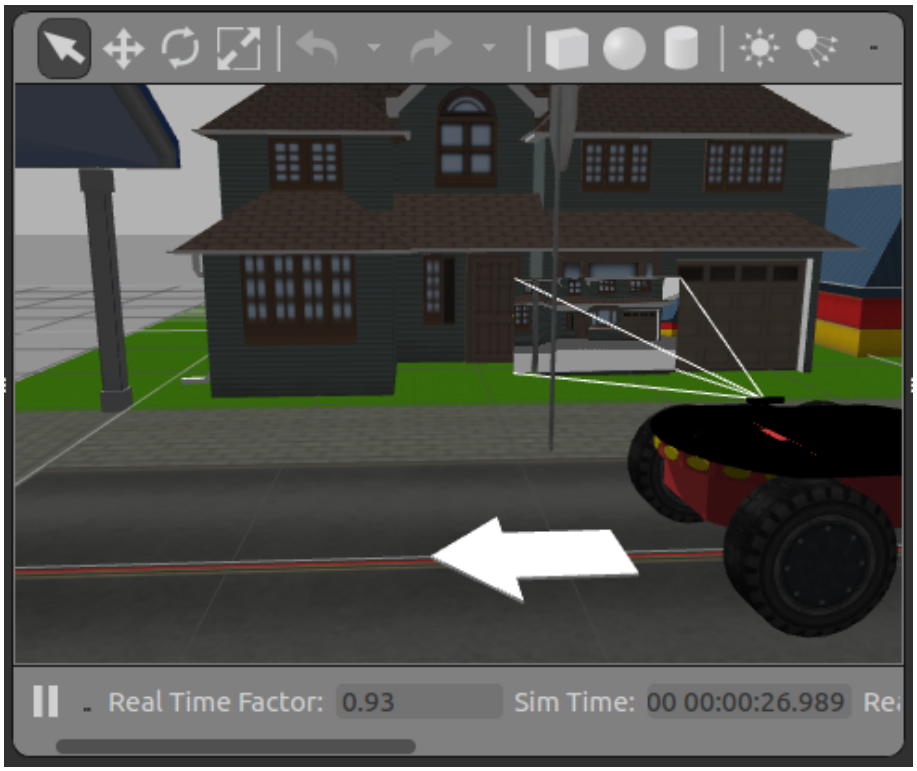


Figura 4.8 Las imágenes captadas por la cámara son visualizadas en la simulación (visualize=True).

- **plugin**
  - **alwaysOn**: Indica si el *plugin* puede activarse o desactivarse en cualquier momento. Al estar a *True*, se ordena que siempre esté activo.
  - **updateRate**: Frecuencia a la que publica la información.
  - **imageTopicName**: Tópico en el que publica la información.
  - **cameraInfoTopicName**: Tópico en el que publica datos sobre la información que publica. En este caso, al tratarse de una cámara los datos que publica son el alto y ancho de las imágenes, y el tipo de formato.

Código 4.5 Plugin de la IMU.

```
<gazebo reference="imu_link">
  <gravity>true</gravity>
  <sensor name="imu_sensor" type="imu">
  <always_on>true</always_on>
  <update_rate>200</update_rate>
  <topic>__default_topic__</topic>
  <plugin filename="libgazebo_ros_imu_sensor.so" name="imu_plugin">
    <topicName>imu0</topicName>
    <bodyName>imu_link</bodyName>
    <updateRateHZ>200.0</updateRateHZ>
    <gaussianNoise>0</gaussianNoise>
    <xyzOffset>0 0 0</xyzOffset>
    <rpyOffset>0 0 0</rpyOffset>
```

```
        <frameName>imu_link</frameName>

        <accelDrift>0.5 0.5 0.5</accelDrift>
        <accelGaussianNoise>0.35 0.35 0.3</accelGaussianNoise>
        <rateDrift>0.0 0.0 0.0</rateDrift>
        <rateGaussianNoise>0.00 0.00 0.00</rateGaussianNoise>
        <headingDrift>0.0</headingDrift>
        <headingGaussianNoise>0.00</headingGaussianNoise>
      </plugin>

    </sensor>
  </gazebo>
```

En el código 4.5, se pueden observar cómo los parámetros relacionados con el sensor IMU son los mismos (*topicname*, *updateRateHZ*...) que en el *plugin* de la cámara. Pero, en cambio, hay otros parámetros del *plugin* que si varían. Se han especificado unos parámetros referidos al ruido de la IMU: la desviación estándar de los ruidos gausianos y la desviación estándar de los errores de deriva. Justamente estos datos son muy importantes porque es necesario conocerlos para posteriormente indicarlos en el fichero de configuración del algoritmo VINS-Mono.

En la figura 4.9 se pueden observar las conexiones existentes en la simulación creada del vehículo terrestre en Gazebo y los tópicos de ROS donde se publica información o se lee. Por un lado, se publica información en los tópicos referidos a ambas cámaras (*/cochecito/camara_lat* y */cochecito/camara_front*) y a la IMU(*/imu0*). Por otro lado, a uno de los tópicos a los que se suscribe es */cmd_vel* (cuya información proviene del nodo con el que se maneja el vehículo y le indica la velocidad y aceleración en cada instante).

Resaltar que aunque el vehículo tenga incorporadas dos cámaras, las imagenes utilizadas posteriormente en el algoritmo VINS-Mono son las captadas por la cámara lateral. La cámara frontal sólo se ha incluido porque es más típico tener una cámara que tome fotos en la parte delantera del vehículo, pero no tiene ninguna función a la hora de realizar la estimación de la trayectoria.

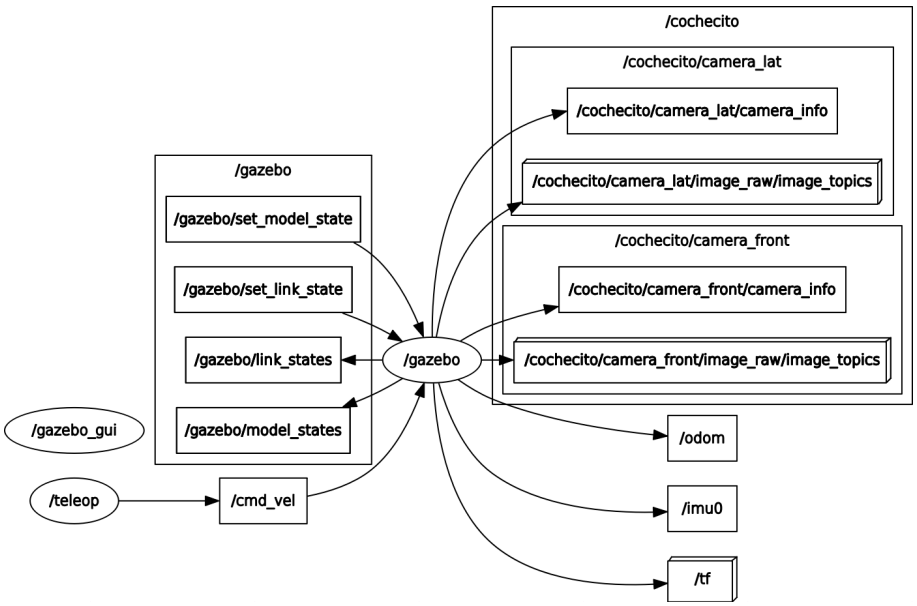


Figura 4.9 Nodos y tópicos creados con la simulación de Gazebo. Imagen tomada a partir de la herramienta *rqt_graph*.

5 Algoritmo VINS-Mono: Simulaciones

Para realizar experimentos del algoritmo VINS-Mono se ha decidido utilizar diferentes entornos y robots. Por un lado, se han utilizado varios datasets públicos de EuRoC MAV [8], cuyas medidas son tomadas por sensores existentes en un dron. Por otro lado, se han realizado un par de simulaciones en Gazebo en tiempo real (una de ellas estima la trayectoria de un coche, mientras que la otra estima la trayectoria de un dron).

5.1 Archivo de configuración de VINS-Mono

Para hacer funcionar el algoritmo con las simulaciones anteriores es necesario modificar un archivo de configuración, como ya se comentó anteriormente. En este se indican algunos datos necesarios para una buena estimación, entre los que se encuentran los parámetros intrínsecos de los sensores (tanto de la IMU como de la cámara), además de los parámetros extrínsecos que reflejan la posición y orientación relativa entre ambos. A parte, también se especifican otros parámetros relacionados con la visualización, con la configuración de cierre de bucle, etc. A continuación, se van a explicar más detenidamente todos estos aspectos, incluyendo distintas partes del código. El código completo se puede encontrar en el anexo 7.4, donde se han incluido los ficheros de configuración de cada una de las simulaciones realizadas. El fichero de configuración utilizado para los experimentos basados en los *datasets* de EuRoC MAV es siempre el mismo porque el *dataset* se obtiene del mismo vehículo aéreo y los parámetros no varían (por tanto, sólo se añade en el anexo el archivo utilizado en el primer *dataset*). La única diferencia se encuentra en el directorio donde se guardan los datos de salida.

En primer lugar, en el codigo 5.1 se indican los tópicos a los que se suscribe el algoritmo para leer la información obtenida de los sensores, tanto de la cámara monocular como de la IMU (en estos tópicos publican las medidas los *datasets* de EuRoC MAV o las simulaciones de Gazebo, según el experimento en el que nos encontremos). Un requisito para el buen funcionamiento de este algoritmo es que la frecuencia a la que publica medidas la cámara y la IMU sea de mínimo de 20Hz y 100Hz, respectivamente. También se indica el directorio de salida donde más adelante se guardará la trayectoria estimada.

Código 5.1 Parámetros comunes.

```
imu_topic: "/imu0"
image_topic: "/cam0/image_raw"
output_path: "/home/mariarod/Escritorio/TFG/experimentos/dataset_EUROC1
    /"
```

Por otro lado, se especifican los parámetros intrínsecos de la cámara: el alto y el ancho de las imágenes, y los parámetros de distorsión y de proyección. Esta información se obtiene de forma diferente según el experimento y se explicará más detalladamente en los posteriores apartados. En las simulaciones a partir de un modelo ya creado se investiga el valor de estos parámetros a partir de datos publicados por los sensores en tópicos en ROS, mientras que en la simulación en la que el vehículo es creado estos parámetros serán conocidos. Esto es debido a que los sensores son diseñados y, por tanto, todos estos parámetros son elegidos personalmente.

Código 5.2 Calibración de la cámara.

```
model_type: PINHOLE
camera_name: camera
image_width: 752
image_height: 480
distortion_parameters:
   k1: -2.917e-01
   k2: 8.228e-02
   p1: 5.333e-05
   p2: -1.578e-04
projection_parameters:
   fx: 4.616e+02
   fy: 4.603e+02
   cx: 3.630e+02
   cy: 2.481e+02
```

Una parte del código que es muy influyente en el buen funcionamiento del algoritmo es la calibración entre la cámara e IMU. El parámetro *estimate_extrinsic* tiene tres posibles valores: 0, 1 o 2. Este valor depende del conocimiento que se posea sobre los parámetros extrínsecos entre la IMU y la cámara. Si se conocen los parámetros exactos y confiamos en la matriz de rotación y el vector de traslación que se indica justo abajo en 5.3, se escribe un 0. En cambio, si tenemos una estimación inicial pero se desea mejorar se escribe un 1. Por último, si no conocemos nada sobre los parámetros extrínsecos entre ambos sensores, se escribe un 2 y, por tanto, el algoritmo se encarga de calcular estos valores al comienzo de la simulación. En las dos últimas opciones propuestas, el algoritmo calcula los parámetros extrínsecos (tomando una matriz como guía o desde cero) y los guarda en un fichero llamado *extrinsic_parameter.csv*, guardado en el directorio de salida indicado en el parámetro *output_path* que aparece en el código 5.1. Una vez se haya producido la calibración, se podrán leer estos valores y utilizarlos en cualquier experimento posterior.

Se debe resaltar que, en el caso de que se estimen los parámetros extrínsecos (segundo y tercer caso), es necesario mover el robot y girarlo en todas las direcciones al comienzo de la estimación para obtener una buena calibración. De esta dependerá la posterior estimación de la trayectoria, ya que si no es correcta la transformación cámara-IMU tampoco lo será la futura trayectoria estimada.

Las matriz de rotación y el vector de traslación que se han comentado se introducen justo abajo:

Código 5.3 Parámetros extrínsecos entre cámara e IMU.

```
estimate_extrinsic: 0

#Rotacion del sistema de referencia de la camara a la IMU
```

```
extrinsicRotation: !!opencv-matrix
   rows: 3
   cols: 3
   dt: d
   data: [0.0148655429818, -0.999880929698, 0.00414029679422,
          0.999557249008, 0.0149672133247, 0.025715529948,
          -0.0257744366974, 0.00375618835797, 0.999660727178]
#Traslacion del sistema de referencia de la camara a la IMU
extrinsicTranslation: !!opencv-matrix
   rows: 3
   cols: 1
   dt: d
   data: [-0.0216401454975,-0.064676986768, 0.00981073058949]
```

Al igual que se han indicado los parámetros intrínsecos de la cámara, es necesario caracterizar los ruidos y derivas de la unidad de medida inercial (o IMU). Los parámetros que aparecen en 5.4 se corresponden con la desviación estándar de la medida del ruido del acelerómetro y del giroscopio, y con la desviación estándar del sesgo del acelerómetro y del giroscopio.

Código 5.4 Parámetros IMU.

```
acc_n: 0.08
gyr_n: 0.004
acc_w: 0.00004
gyr_w: 2.0e-6
g_norm: 9.81007
```

Por último, comentar algunos otros parámetros también modificados en los experimentos realizados. *estimate_td* tiene la funcionalidad de medir el *offset* de tiempo entre la IMU y la cámara (se suele activar cuando el experimento se realiza en tiempo real). *load_previous_pose_graph* lee una trayectoria calculada en algún experimento anterior. Lee esta trayectoria del directorio que es indicado en *pose_graph_save_path*. Para activar la funcionalidad de guardar la trayectoria estimada, se encuentra el parámetro *save_image*.

Una vez explicados los parámetros cuyos valores se deben modificar según el experimento en el que nos encontremos para que la trayectoria estimada por VINS-Mono sea buena, se procede a explicar cada experimento de forma individual: simulación realizada y resultados obtenidos.

5.2 EuRoC MAV *datasets*

En primer lugar, se evaluó el algoritmo VINS-Mono a partir de varios *datasets* públicos denominados EuRoC MAV [8], recogidos de un Vehículo Aéreo Micro (o MAV). En total existen 11 *datasets* diferentes, en los que la dificultad va aumentando gradualmente. Estos abarcan desde vuelos más lentos hasta vuelos con peor iluminación y movimientos más bruscos. Se han utilizado los cuatro primero *datasets* en este trabajo, los cuales han sido creados en un entorno industrial con el objetivo de permitir evaluar algoritmos SLAM en situaciones reales.

Para estos *datasets* se utilizó el helicóptero AscTec "FireFly" (figura 5.1). Este proporciona imágenes provenientes de una cámara estereoscópica (Aptina MT9V034 global shutter, WVGA monochrome, 2×20 FPS), medidas de una IMU (ADIS16448, angular rate and acceleration, 200 Hz)

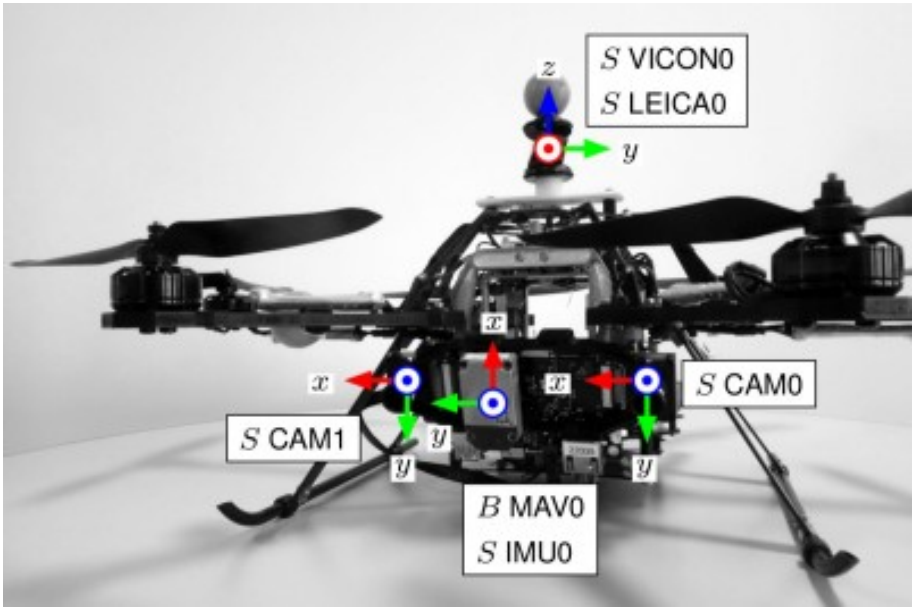


Figura 5.1 Helicóptero de 6 hélices utilizado para la recogida de los datos del *dataset* EuRoC MAV.

y medidas *ground truth* del sensor láser Leica (obtiene la posición real 3D con precisión milimétrica, 20Hz).

Se debe resaltar que las medidas de los sensores proporcionados por estos *datasets* están expresadas respecto el SI (Sistema Internacional). La única excepción es el tiempo, el cual se expresa en nanosegundos. Por otro lado, las medidas de todos los sensores se expresan respecto al marco de referencia de dicho sensor. En los *datasets*, para cada sensor, se proporcionan tanto las medidas como los parámetros extrínsecos (que expresan la posición y orientación del sensor correspondiente respecto al cuerpo).

La alineación de los datos de la estimación obtenida con los datos de la posición 3D real permite evaluar el algoritmo VINS-Mono y realizar comparaciones, las cuales se mostrarán al final de este capítulo. Se va a estimar la trayectoria seguida en varios *datasets* y, posteriormente, se compararán estas estimaciones con las medidas reales para poder evaluar el algoritmo VINS-Mono. También se calculará el error de cada estimación respecto a cada eje (x,y,z) y respecto a los ángulos de Euler (*roll*, *pitch* y *yaw*). Por último, se realiza una comparación de los errores que se han producido en cada *dataset* (cada uno con diferente dificultad).

Tras explicar los datos obtenidos de los *datasets*, comienzo a hablar de la forma en la que se han utilizado para realizar los experimentos del algoritmo VINS-Mono. Aunque estos *datasets* proporcionan imágenes estereoscópicas, como el algoritmo se basa en visión monocular, se utiliza sólo una de las cámaras (en este experimento se va a utilizar, en concreto, la cámara izquierda). Las imágenes tomadas por esta cámara son publicadas en el tópico *\cam0 \image_raw* (figura 5.2). Por otro lado, las medidas de la IMU se publican en el tópico *\imu0*.

Con el objetivo de lanzar el algoritmo VINS-Mono para la estimación de la trayectoria, también es necesario conocer los parámetros intrínsecos y extrínsecos de la cámara y de la IMU, los cuales se descargan junto con el *dataset* de EuRoC MAV. Este tendrá la siguiente forma:

- **<sensor_system_id>**

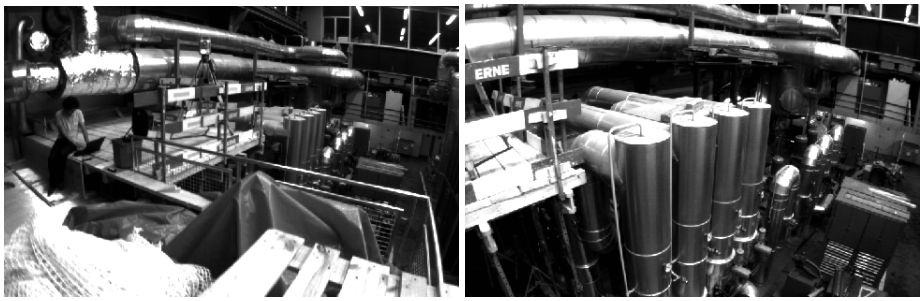


Figura 5.2 Imágenes tomadas por la cámara izquierda del helicóptero en instantes aleatorios.

- **<sensor_id>**
  - **sensor.yaml**
  - **data.csv**
  - **data**

Las medidas de los sensores se obtienen de *data.csv*. Por otro lado, en *sensor.yaml* se guarda los parámetros extrínsecos del sistema de referencia del sensor <*sensor_id*> respecto al sistema de referencia del cuerpo (el cual coincide con el de la IMU). En este archivo también se definen los parámetros intrínsecos de dicho sensor. Por tanto, se conoce la transformación exacta entre la cámara y la IMU, y se fija el valor del parámetro *estimate_extrinsic* a 0.

Una vez se conocen todos los parámetros necesarios y se ha modificado el archivo de configuración, se procede a lanzar el algoritmo. Realmente, aunque haya explicado cómo se obtienen todos los parámetros para incluirlos en el fichero de configuración, esto no es necesario en este experimento. La razón es la siguiente: el archivo que se descarga por defecto al descargar VINS-Mono está preparado para que se pruebe el funcionamiento del algoritmo con uno de estos *datasets* (están indicados los tópicos en los que publican los sensores cámara e IMU, los parámetros extrínsecos que describen la posición y orientación entre la cámara y la IMU, e incluso los parámetros intrínsecos de la cámara y de la IMU). El archivo de configuración utilizado aparece completo en el código 7.4. Este archivo, como ya se ha comentado anteriormente, será común para todos los experimentos realizados con los *datasets* de EuRoC MAV porque estos se obtienen a partir de las medidas del mismo helicóptero y, por tanto, los sensores y sus posiciones en el helicóptero son la misma. Los únicos campos que han sido modificados son los relativos al directorio donde se guardan los resultados obtenidos.

Posteriormente, se lanza el algoritmo y el simulador Rviz. Luego, se lanza la *rosbag* del primer *dataset* de EuRoC MAV (el de dificultad más baja) con el fin de estimar la trayectoria. Las órdenes necesarias aparecen en el código 5.5. La última línea tiene el objetivo de publicar las medidas *ground truth* para poder visualizar tanto la trayectoria estimada por el algoritmo como la trayectoria real.

Código 5.5 Lanzar experimento *dataset* EuRoC MAV.

```
roslaunch vins_estimator euroc.launch
roslaunch vins_estimator vins_rviz.launch
rosbag play MH_01_easy.bag
roslaunch benchmark_publisher publish.launch sequence_name:=MH_01_easy
```

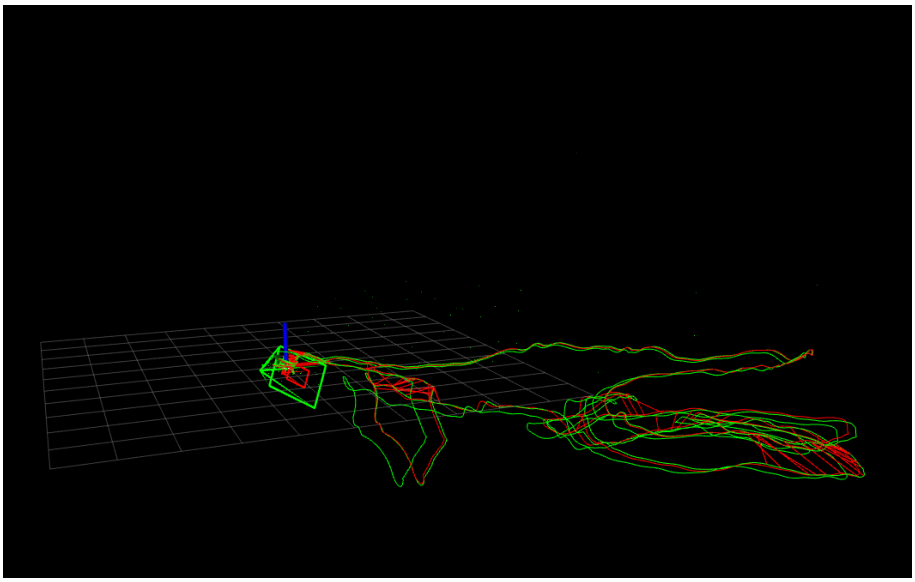


Figura 5.3 Trayectoria estimada VINS-Mono a partir del *dataset* EuRoC MAV de dificultad baja, junto con la trayectoria real (los colores son verde y rojo, respectivamente).

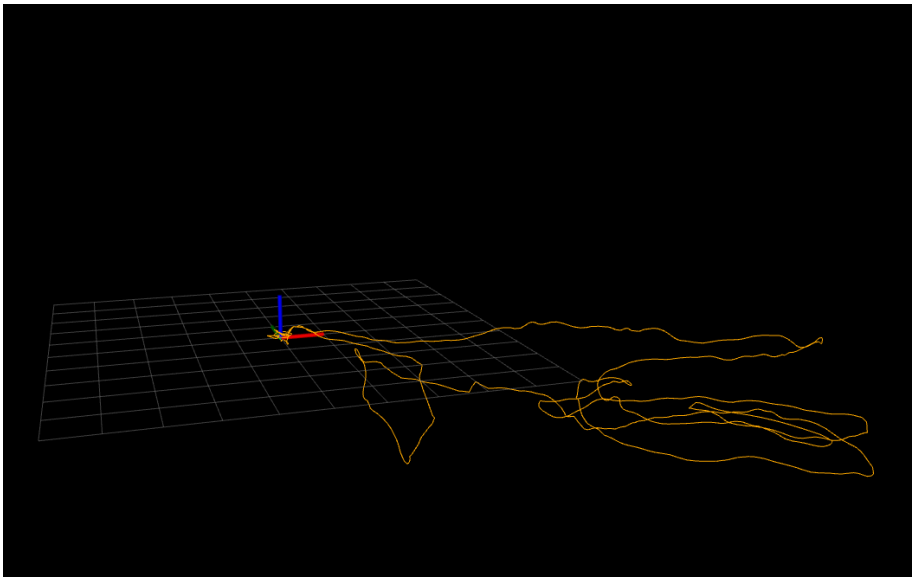


Figura 5.4 Trayectoria guardada VINS-Mono del *dataset* EuRoC MAV de dificultad baja.

Si realizamos esta estimación por primera vez, se puede ver el resultado obtenido en la imagen 5.3. Aquí, se puede observar la trayectoria estimada por el algoritmo, junto con la trayectoria real, la cual se obtiene de las medidas *ground truth*. Se puede guardar esta trayectoria para utilizarla en experimentos posteriores (esto permite optimizar la nueva estimación a a través de su fusión con la estimación ya guardada). La trayectoria guardada se puede observar en la figura 5.4. Por último, si volvemos a lanzar el algoritmo, se obtiene la imagen 5.5, la cual muestra las trayectorias de ambos experimentos lanzados junto con la trayectoria real (la cual se corresponde con la línea roja).

Este mismo procedimiento se sigue para lanzar los cuatro *datasets* con los que se trabaja: *MH_01_easy.bag*, *MH_02_easy.bag*, *MH_03_medium.bag* y *MH_04_difficult.bag*. Los resultados se mostrarán más adelante, cuando se realice la comparación entre los cuatro experimentos. Se puede observar en la figura 5.6 las imágenes que capta la cámara en cada *dataset* para poder observar el

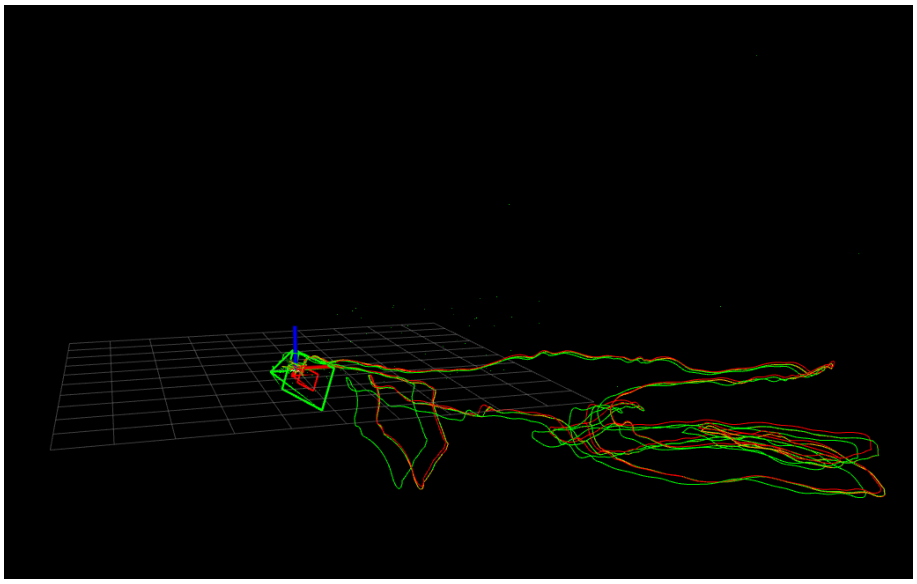


Figura 5.5 Trayectoria estimada VINS-Mono a partir del *dataset* EuRoC MAV de dificultad baja, junto con una trayectoria anterior ya guardada y la trayectoria real (los colores son verde, naranja y rojo, respectivamente.

aumento de la dificultad de las estimaciones.

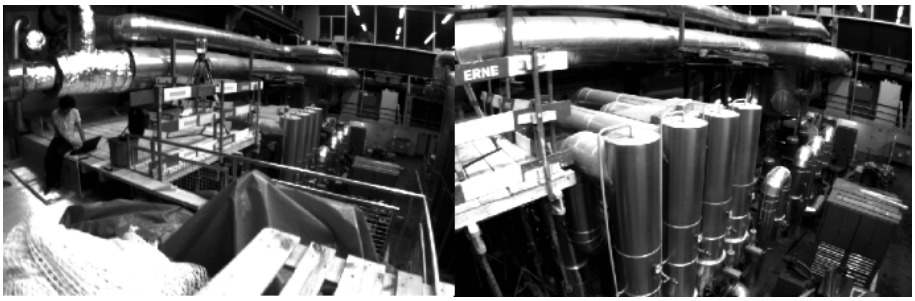
Una vez se estima la trayectoria para cada *dataset* y, a partir de las medidas *ground truth*, se procede a evaluar el algoritmo. Para ello, se va a utilizar un paquete llamado *evo* [14], de Python. El objetivo de este paquete es evaluar y comparar la trayectoria obtenida en algoritmos SLAM. Entre los formatos de trayectoria que soporta, se incluye el *dataset* EuRoC MAV.

Se van a utilizar varias herramientas de este paquete: *evo_traj* y *evo_ape*. La primera sirve para analizar y crear gráficas de una o varias trayectorias conjuntamente, mientras que la segunda métrica calcula el error cometido en la estimación (mínimo, máximo, media, etc.).

En primer lugar, se van a representar la trayectoria estimada y la trayectoria real a partir de *evo_traj*. El problema principal es que la trayectoria guardada por el algoritmo VINS-Mono no se puede utilizar con este paquete directamente porque no tiene formato TUM ni formato de *dataset* de EuRoC MAV. Por tanto, hay que hacer una serie de modificaciones en el algoritmo, en los ficheros donde se guardan los archivos de salida. De esta forma, se consigue que la extensión del archivo obtenido sea *.txt* en lugar de *.csv*. Las modificaciones realizadas se indican a continuación:

Código 5.6 Modificar función *pubOdometry()* en archivo *visualization.cpp*.

```
ofstream foutC(VINS_RESULT_PATH, ios::app);
      foutC.setf(ios::fixed, ios::floatfield);
      foutC.precision(0);
      foutC << header.stamp.toSec() << " ";
      foutC.precision(5);
      foutC << estimator.Ps[WINDOW_SIZE].x() << " "
            << estimator.Ps[WINDOW_SIZE].y() << " "
            << estimator.Ps[WINDOW_SIZE].z() << " "
            << tmp_Q.x() << " "
```



(a) MH_01_easy.

(b) MH_02_easy.

(c) MH_03_medium.

(d) MH_04_difficult.

Figura 5.6 Imágenes utilizadas en los *datasets* EuRoC MAV.

```
            << tmp_Q.y() << " "
            << tmp_Q.z() << " "
            << tmp_Q.w() << endl;
```

Código 5.7 Modificar función *updatePath()* en archivo *pose_graph.cpp*.

```
ofstream loop_path_file(VINS_RESULT_PATH, ios::app);
        loop_path_file.setf(ios::fixed, ios::floatfield);
```

```
            loop_path_file.precision(0);
            loop_path_file << (*it)->time_stamp << " ";
            loop_path_file.precision(5);
            loop_path_file << P.x() << " "
                           << P.y() << " "
                           << P.z() << " "
                           << Q.x() << " "
                           << Q.y() << " "
                           << Q.z() << " "
                           << Q.w() << endl;
```

Código 5.8 Modificar función *addKeyFrame()* en archivo *pose_graph.cpp*.

```
ofstream loop_path_file(VINS_RESULT_PATH, ios::app);
      loop_path_file.setf(ios::fixed, ios::floatfield);
      loop_path_file.precision(0);
      loop_path_file << cur_kf->time_stamp << " ";
      loop_path_file.precision(5);
      loop_path_file << P.x() << " "
                     << P.y() << " "
                     << P.z() << " "
                     << Q.x() << " "
                     << Q.y() << " "
                     << Q.z() << " "
                     << Q.w() << endl;
```

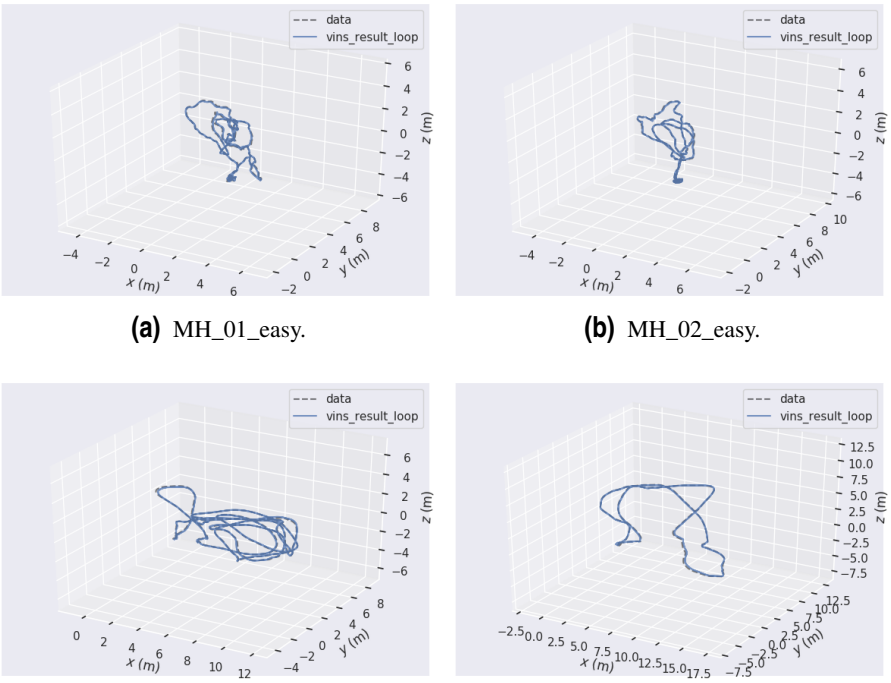
Código 5.9 Modificar función *main()* en archivo *pose_graph_node.cpp*.

```
   VINS_RESULT_PATH = VINS_RESULT_PATH + "/vins_result_loop.txt";
```

Una vez se realizan todas las modificaciones necesarias, se vuelve a lanzar el algoritmo para obtener el archivo de salida *vins_result_loop.txt*. Por tanto, ya se tiene la trayectoria estimada en el formato correcto para evaluar el algoritmo con *evo*. Por otro lado, los datos *ground truth* tienen formato *.csv* y se tienen que transformar a TUM. Para ello, se utiliza la orden mostrada en 5.10. Posteriormente, ya se puede ejecutar la herramienta *evo_traj* para obtener las gráficas comparativas de las trayectorias, de las posiciones respecto los tres ejes, y de los ángulos de Euler. La comparación entre la trayectoria real y estimada se puede observar en la imagen 5.7, en la cual aparece la trayectoria estimada de todos los *datasets*. Las gráficas de las posiciones respecto los tres ejes y respecto los ángulos de navegación son como las mostradas en 5.8 (solo se han añadido las obtenidas a partir del primer *dataset* para mostrar cómo son las gráficas obtenidas). En estas se pueden ver las líneas correspondientes a la estimación y a las medidas *ground_truth* casi superpuestas (es un poco complicado diferenciarlas). Por otro lado, la línea recta que aparece en todas las gráficas sólo tiene la funcion de unir el primer valor con el último valor del experimento (es decir, no tiene nada que ver con la trayectoria real o estimada).

Código 5.10 Transformar medidas *ground truth* a formato TUM.

```
   evo_traj euroc data.csv --save_as_tum
```



(a) MH_01_easy. **(b)** MH_02_easy.

(c) MH_03_medium. **(d)** MH_04_difficult.

Figura 5.7 Gráficas obtenidas a partir de la herramienta *evo_traj*. Contienen la trayectoria estimada y la trayectoria real de los cuatro *datasets* .

Tabla 5.1 Errores calculados a partir de herramienta *evo_ape* (en metros).

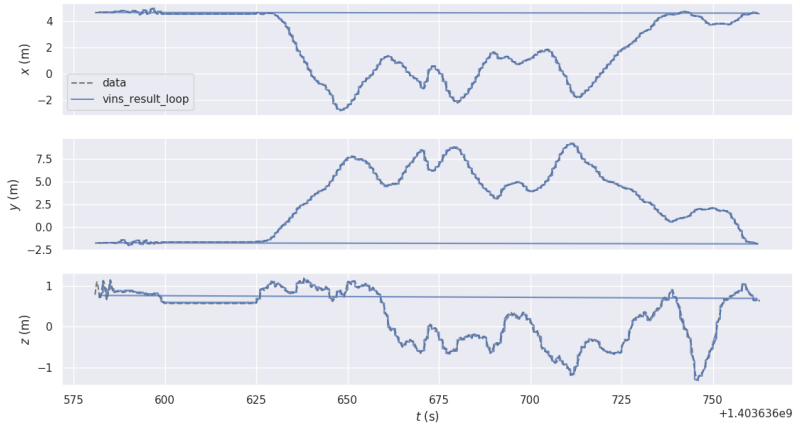
	Max	Mean	Median	Min	RMSE	SSE	Std
MH_01_easy	0.90279	0.20720	0.20457	0.00937	0.22785	94.23017	0.09478
MH_02_easy	0.83487	0.19371	0.13849	0.01478	0.24065	86.64175	0.14279
MH_03_medium	1.30522	0.32323	0.23857	0.02071	0.40105	210.70204	0.23740
MH_04_difficult	1.5956	0.42322	0.39195	0.04747	0.47774	222.52977	0.22162

Código 5.11 Utilizar herramienta *evo_traj*.

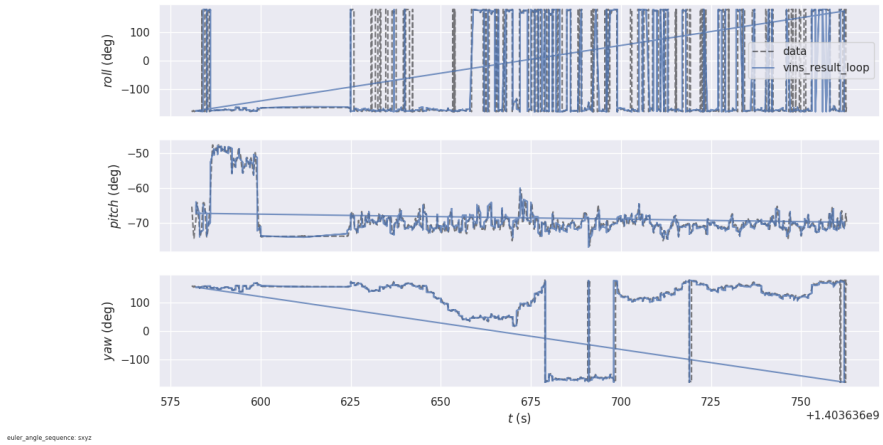
```
evo_traj tum ../../../vins_result_loop.txt --ref=data.tum -p --
   plot_mode=xyz align -correct_scale
```

Pero, para poder comparar la evaluación de VINS-Mono de los cuatro *datasets*, se van a calcular los errores de cada experimento con la herramienta *evo_ape* y se van a mostrar en la tabla 5.1. En esta tabla aparece diferente información sobre el error cometido: error máximo, error mínimo, error medio, la mediana, RMSE (error cuadrático medio), SSE (suma de los cuadrados de los errores), y STD (error estándar de la media).

Voy a explicar a continuación algunos de los errores que aparecen en la tabla y cuyos valores se obtienen con la herramienta *evo_ape*. El RMSE mide la cantidad de error que hay entre dos conjuntos de datos, comparando un valor predicho y un valor observado o conocido. Por otro lado, el SSE también mide la diferencia entre el valor observado y el predicho. Los valores de estos errores aumentan cuando también lo hace la complejidad de los *datasets*, debido a que las estimaciones son menos parecidas a los datos observados. Por último, el error estándar de la media es la desviación estándar de una muestra estadística (es decir, la variación entre la media de la población y la media



(a) Comparación posiciones x, y, z.



(b) Comparación ángulos *roll*, *pitch* y *yaw*.

Figura 5.8 Gráficas obtenidas a partir de la herramienta *evo_traj* (para el *dataset MH_01_easy*).

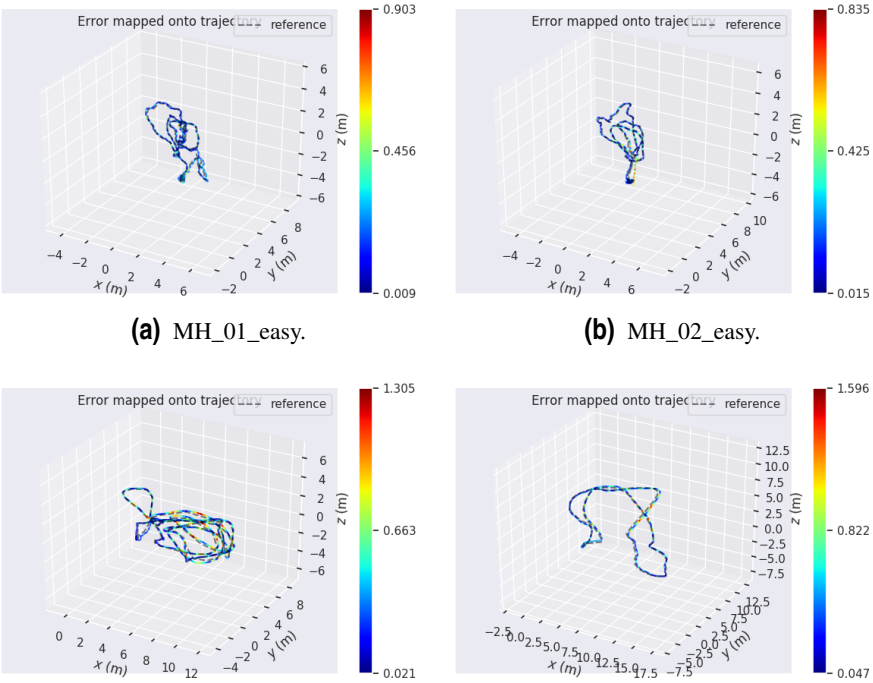
que se considera conocida). Por tanto, se puede observar en la tabla 5.1 que el error producido en la estimación de la trayectoria con VINS-Mono aumenta a medida que aumenta la complejidad de los *datasets*.

También se han obtenido unos mapas de errores para los cuatro *datasets*, los cuales indican en qué zonas de la trayectoria se comete un mayor error en la estimación. Estos se han recogido en la figura 5.9.

Existen más herramientas en el paquete *evo* para la evaluación de algoritmos SLAM, pero he considerado *evo_traj* y *evo_ape* son aquellas que ofrecen una información más descriptiva para poder evaluar VINS-Mono.

5.3 Vehículo aéreo Gazebo

Este experimento se ha realizado a partir de un paquete de ROS llamado *tum_simulator* [1]. Este se utiliza para simular el robot aéreo Ardrone en Gazebo. Por tanto, es otro experimento del algoritmo realizado en un dron, ya que en este tipo de vehículos es donde más se utiliza VINS-Mono debido a que el sistema visuo-inercial utilizado (una cámara monocular e IMU) supone bajo peso y poco



(a) MH_01_easy. **(b)** MH_02_easy.

(c) MH_03_medium. **(d)** MH_04_difficult.

Figura 5.9 Gráficas obtenidas a partir de la herramienta *evo_epo*. Contienen la trayectoria estimada junto con el nivel de error cometido.

Figura 5.10 Ardrone 2.0.

consumo de potencia.

Para controlar el Ardrone 2.0 en Gazebo se ha utilizado un paquete denominado *ardrone_autopilot* [2]. Con este paquete es posible despegar y aterrizar el dron, trasladarlo en todas las direcciones, rotar y cambiar de cámara. En la simulación, debido a las limitaciones de Gazebo, no están modelados todos los sensores del Ardrone 2.0 al 100 %. Están implementadas un par de cámaras (una frontal y otra dirigida hacia el suelo), un sensor de altitud y un sensor IMU.

Para poder ejecutar el algoritmo VINS-Mono y estimar los estados del dron al mismo tiempo que este se controla en la simulación de Gazebo es necesario conocer, en primer lugar, los tópicos donde se publican las medidas de los sensores que nos interesan. Por un lado, las medidas de la IMU son publicadas en *\ardrone \imu*. En cuanto al tópico donde se publican las imágenes captadas por las cámaras, se tratan de dos tópicos diferentes debido a que tiene una cámara frontal y otra dirigida

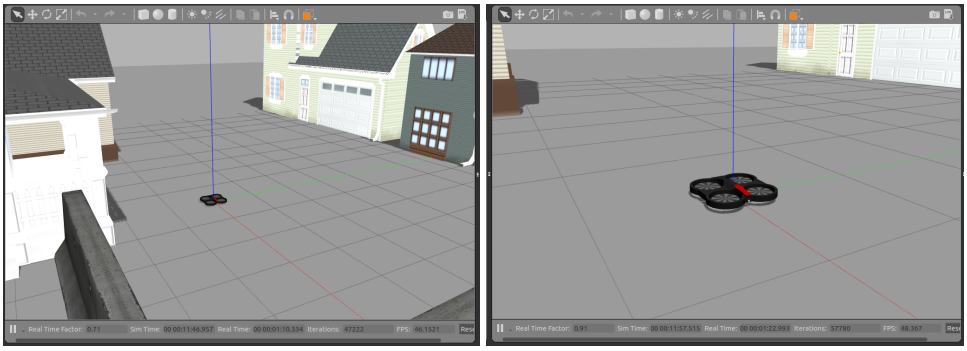


Figura 5.11 Entorno de Ardrone 2.0 en Gazebo.

hacia el suelo. Para el algoritmo VINS-Mono se utiliza la cámara frontal debido a que capta más riqueza de elementos en el entorno y mejorará la estimación. El tópico de dicha cámara es *\ardrone \front \image_mono*.

En este experimento, se va a suponer que la posición y orientación de la cámara respecto la IMU son desconocidas, por lo que el valor del parámetro *estimate_extrinsic* se fija a 2 (realmente se puede obtener a partir de los archivos de configuración descargados con el paquete *tum_simulator*). Esto significa que al comienzo de la simulación se realizará la calibración de los parámetros extrínsecos cámara-IMU y será necesario rotar y desplazar el dron en todas las direcciones durante unos segundos. En lo que respecta a los parámetros internos de la cámara, los cuales son necesarios para modificar el archivo de configuración de VINS-Mono, se pueden consultar en el tópico *\ardrone \front \image_mono \camera_info*. En este tópico, los parámetros son publicados por el simulador de Gazebo, como se puede observar en los mensajes publicados en 5.12. Generalmente la información sobre los parámetros de la cámara se publican en un tópico en el que aparezca *"camera_info"*.

Código 5.12 Mensaje publicado en el tópico \ardrone \front \image_mono \camera_info.

```
header:
  seq: 8726
  stamp:
    secs: 872
    nsecs: 745000000
  frame_id: "ardrone_base_frontcam"
height: 360
width: 640
distortion_model: "plumb_bob"
D: [0.0, 0.0, 0.0, 0.0, 0.0]
K: [374.6706070969281, 0.0, 320.5, 0.0, 374.6706070969281, 180.5, 0.0,
    0.0, 1.0]
R: [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0]
P: [374.6706070969281, 0.0, 320.5, -0.0, 0.0, 374.6706070969281, 180.5,
    0.0, 0.0, 0.0, 1.0, 0.0]
binning_x: 0
binning_y: 0
roi:
  x_offset: 0
  y_offset: 0
```

```
  height: 0
  width: 0
  do_rectify: False
```

Como se puede ver en el código, aparecen todos los parámetros necesarios sobre la cámara que se comentaron anteriormente: alto y ancho de la imagen, parámetros de distorsión y parámetros de proyección.

Otra forma de obtener los parámetros de la cámara es buscar la configuración del sensor dentro de los archivos descargados del paquete de Gazebo. En estos archivos se puede encontrar el entorno que se utiliza, el modelo del dron, los sensores que posee, los *plugins* que utiliza, etc. Es justamente la información de los *plugins* la que nos interesa para conocer los parámetros tanto de la cámara como de la IMU. Esta información se puede obtener de los códigos 5.13 y 5.14, respectivamente.

Código 5.13 Configuración de la cámara en el Ardrone 2.0.

```
<xacro:include filename="$(find cvg_sim_gazebo)/urdf/sensors/
    generic_camera.urdf.xacro" />
    <xacro:generic_camera name="front" sim_name="ardrone" parent="
        base_link" update_rate="60" res_x="640" res_y="360" image_format
        ="R8G8B8" hfov="${81*M_PI/180}">
      <origin xyz="0.21 0.0 0.01" rpy="0 0 0"/>
    </xacro:generic_camera>

    ----------------------------------------

    <plugin name="${name}_camera_controller" filename="
        libgazebo_ros_camera.so">
    <cameraName>/${sim_name}/${name}</cameraName>
          <alwaysOn>true</alwaysOn>
          <updateRate>${update_rate}</updateRate>
          <imageTopicName>/${sim_name}/${name}/image_raw</imageTopicName>
          <cameraInfoTopicName>/${sim_name}/${name}/camera_info</
              cameraInfoTopicName>
          <frameName>ardrone_base_${name}cam</frameName>
    </plugin>
```

Código 5.14 Configuración de la IMU en el Ardrone 2.0.

```
<plugin name="quadrotor_imu_sim" filename="libhector_gazebo_ros_imu.so">
       <alwaysOn>true</alwaysOn>
       <updateRate>100.0</updateRate>
       <bodyName>base_link</bodyName>
       <frameId>ardrone_base_link</frameId>
       <topicName>/ardrone/imu</topicName>
       <rpyOffsets>0 0 0</rpyOffsets> <!-- deprecated -->
       <gaussianNoise>0</gaussianNoise> <!-- deprecated -->
       <accelDrift>0.5 0.5 0.5</accelDrift>
       <accelGaussianNoise>0.35 0.35 0.3</accelGaussianNoise>
       <rateDrift>0.0 0.0 0.0</rateDrift>
```

```
        <rateGaussianNoise>0.00 0.00 0.00</rateGaussianNoise>
        <headingDrift>0.0</headingDrift>
        <headingGaussianNoise>0.00</headingGaussianNoise>
      </plugin>
```

Una vez modificado el archivo de configuración, se realiza la estimación de la trayectoria con el algoritmo VINS-Mono. Para ello, se utilizan los comandos observados en 5.15:

- La primera línea lanza la simulación del dron del paquete Ardrone 2.0.
- La segunda se utiliza para poder controlar el dron a través del teclado y moverlo por el entorno con el paquete descargado de *ardrone_autopilot*.
- La tercera línea se utiliza para modificar el tipo de formato de imagen que tienen las imágenes captadas por la cámara frontal. Este formato es RBG, el cual no es compatible con el algoritmo VINS-Mono. De esta forma, se corrige transformando las imágenes a formato monocromático. Para ello se utiliza el paquete *image_proc* [4].
- Las últimas líneas ya se vieron en el experimento del *dataset* de EuRoC MAV, pues son los comandos necesarios para lanzar el algoritmo VINS-Mono y poder abrir el simulador Rviz (estas líneas son comunes en todos los experimentos).

Código 5.15 Lanzar experiemnto Ardrone 2.0.

```
roslaunch cvg_gazebo ardrone_testword.launch
roslaunch ardrone_autopilot autopilot.launch
ROS_NAMESPACE=ardrone/front rosrun image_proc image_proc

roslaunch vins_estimator euroc.launch
rslaunch vins_estimator vins_rviz.launch
```

Los resultados obtenidos se pueden observar 5.12. Esta imagen se ha tomado del visualizador RViz. Además de la trayectoria estimada del dron, se pueden observar las imágenes que capta la cámara y también las características extraídas de las imágenes. Estas son sólo algunas de las opciones que ofrece RViz, ya que se puede visualizar la información de cualquier tópico de la simulación.

El archivo completo de configuración utilizado para lanzar esta simulación aparece completo en 7.5.

5.4 Vehiculo terrestre Gazebo

El último experimento realizado para mostrar los resultados del algoritmo VINS-Mono tiene como objetivo estimar la trayectoria de un vehículo terrestre, no como en los casos anteriores que se trataban de vehículos aéreos (los cuales son los vehículos más comunes en los que se suele utilizar este algoritmo). Para este experimento no sólo se ha lanzado el algoritmo y se ha modificado el archivo de configuración, sino que se ha construido la simulación desde cero.

Tal y como se comentó en el apartado 4.3, se ha creado tanto el vehículo como el entorno por el que se mueve. En primer lugar, el entorno se ha formado por elementos ya creados por Gazebo: calles, parques, casas, restaurantes, farolas, señales de tráfico...simulando un entorno real. Cuantos más elementos contenga el entorno, mayor será la riqueza de características en las imágenes tomadas

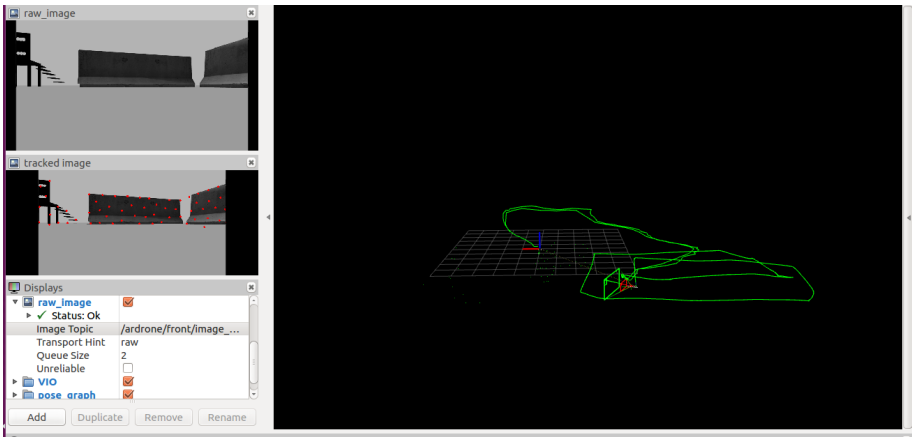


Figura 5.12 Trayectoria obtenida de Ardrone 2.0 a partir del algoritmo VINS-MONO.

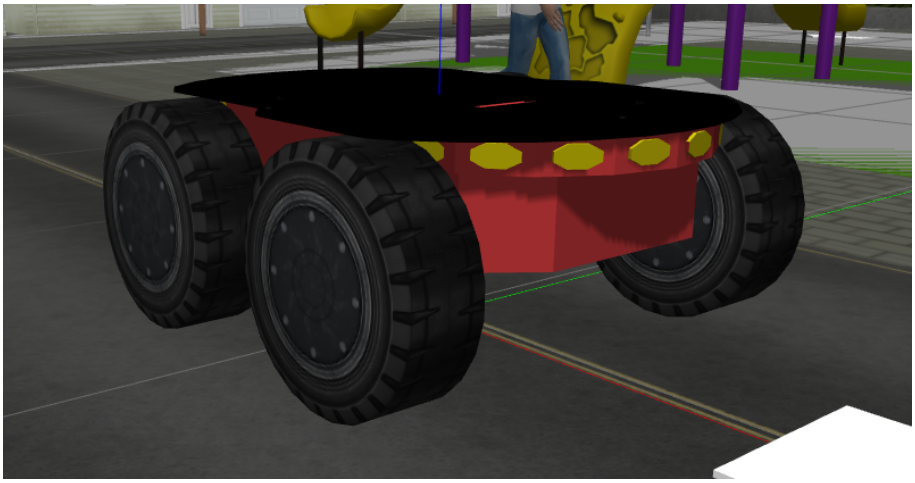


Figura 5.13 Vehículo terrestre modelado en Gazebo.

por las cámaras, y mejor será la estimación. Esta es la razón principal por la que la cámara utilizada para el algortimo VINS-Mono es la cámara lateral y no la cámara frontal: la cámara lateral percibe muchos más elementos del entorno (la cámara frontal percibe sobre todo la carretera, mientras que la lateral capta todos los elementos a ambos lados de la calle).

Además del entorno, también se ha creado el vehículo terrestre. Este está formado por cuatro ruedas y el chasis (figura 5.13). De cada elemento se han indicado sus dimensiones, inercias y otras propiedades (por ejemplo, en las ruedas, también se especifica el coeficiente de rozamiento). Para obtener medidas útiles que puedan utilizarse para la estimación de la trayectoria en VINS-Mono, se han añadido *plugins* para los sensores de las cámaras y de la IMU. El código de estos ya fue explicado en el capítulo 4.3.1.

Además de los *plugins* de la IMU y de las cámaras, se ha añadido un *plugin* para el control del vehículo (código 5.16). Este *plugin* recibe los mensajes de las velocidades lineales respecto x e y, y la velocidad angular respecto z (a través del tópico *cmd_vel*). Por otro lado, publica en el tópico *odom* la posición real (o medidas *ground_truth*) que sigue el vehículo, la cual se utilizará más adelante para calcular el error cometido en la estimación de la trayectoria.

Código 5.16 Plugin de control.

```
<gazebo>
    <plugin name="object_controller" filename="libgazebo_ros_planar_move.
        so">
      <commandTopic>cmd_vel</commandTopic>
      <odometryTopic>odom</odometryTopic>
      <odometryFrame>odom</odometryFrame>
      <odometryRate>50.0</odometryRate>
      <robotBaseFrame>base_footprint</robotBaseFrame>
      <publishTF>false</publishTF>
    </plugin>
</gazebo>
```

Para controlar el vehículo e indicar las velocidades mencionadas anteriormente que necesita el *plugin* de control para mover el vehíchulo se utiliza un paquete de ROS que recibe el nombre de *teleop_twist_keyboard* [3]. Este permite controlar el vehículo a través del teclado (desplazar el vehículo, rotar, aumentar velocidad de desplazamiento y de giro). Este paquete se ejecuta a la vez que se lanza la simulación de Gazebo, sin necesidad de ejecutarlo en un terminal diferente. Esto nos da la libertad de controlar el vehículo en tiempo real.

Una vez creado tanto el entorno como el modelo del vehículo, y ya se están publicando las medidas de la IMU y de la cámara lateral en sus correspondientes tópicos, se procede a modificar el archivo de configuración del algoritmo para realizar la estimación de la trayectoria.

En primer lugar, los tópicos donde se publican las medidas se han elegido al crear el modelo del coche, y se han especificado en el *plugin* del sensor correspondiente. En este caso, los nombres de los tópicos son */imu0* y */cochecito/camera_lat/image_mono*. Por otro lado, como el vehículo ha sido creado desde cero, también se conoce perfectamente la matriz de parámetros extrínsecos entre la cámara (lateral) y el sensor IMU. La posición relativa entre ambos sensores se puede ver en la imagen 5.14. Esta figura se ha obtenido a partir del programa Matlab. A partir de la matriz de parámetros extrínsecos calculada y los ejes de la IMU, se han obtenido los ejes de la cámara lateral. Está claro que si se desease utilizar la cámara frontal, la matriz de parámetros extrínsecos debería ser calculada de nuevo.

La matriz de parámetros extrínsecos dependen tanto de la rotación como de la traslación entre los sensores. Se deben obtener los parámetros extrínsecos de la cámara respecto la IMU y, por tanto, para calcularlos se deben de girar y trasladar los ejes de la IMU hasta que coincidan con la cámara. En el código 5.17 se pueden observar el ángulo que se gira respecto cada eje y la traslación necesaria. El objetivo principal de obtener el gráfico en Matlab es la comprobación de que los parámetros extrínsecos que se han calculado son correctos. Esto se puede comprobar al ver la posición relativa entre la cámara y la IMU con los parámetros calculados. Por último, remarcar que el eje z de la cámara es aquel que apunta hacia las imágenes que se toman.

Código 5.17 Obtención gráfico de la posición relativa entre la IMU y la cámara.

```
theta_ini=180;
phi_ini=0;
alpha_ini=90;

theta=theta_ini*(2*pi/360);%angulo que gira alrededor de z
```

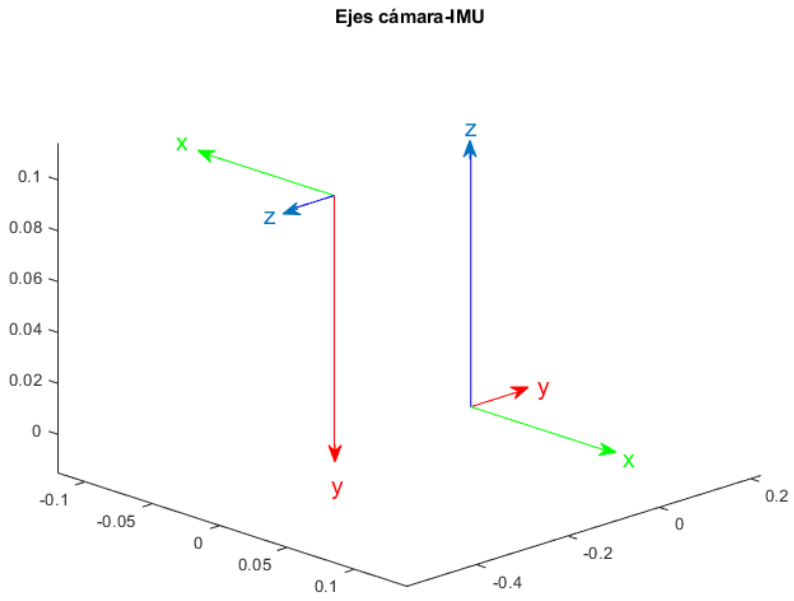


Figura 5.14 Posición relativa entre los sensores cámara-IMU.

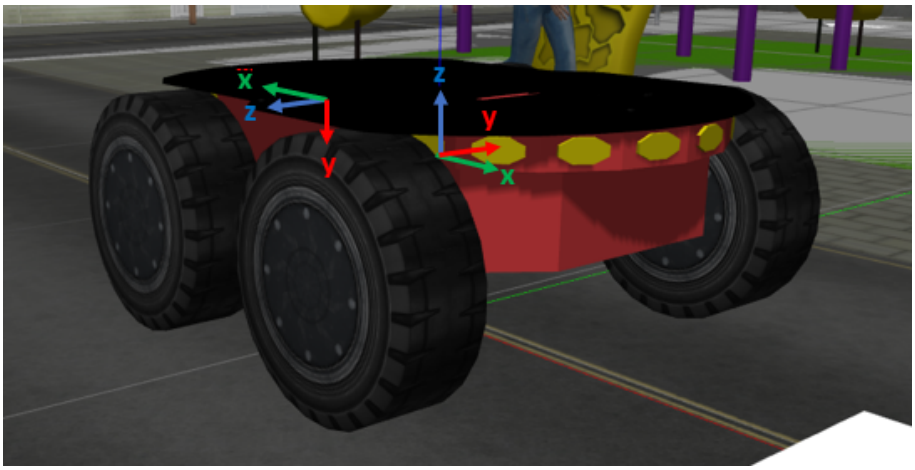
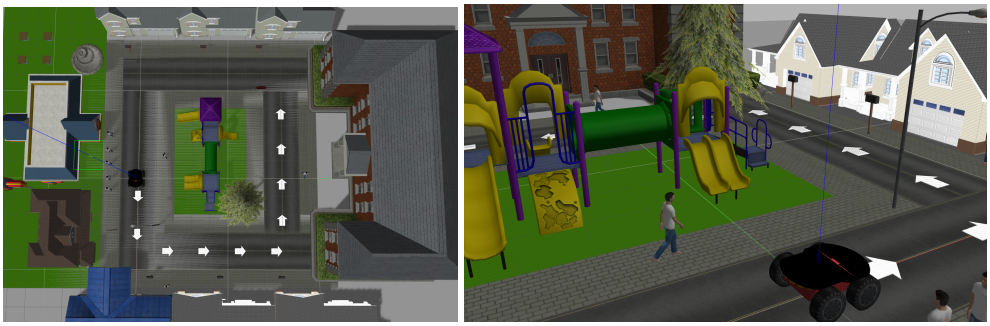


Figura 5.15 Ejes de la cámara e IMU sobre el vehículo.

```
phi=phi_ini*(2*pi/360); %angulo que gira alrededor de y
alpha=alpha_ini*(2*pi/360); %angulo que gira alrededor de x

rotx=[cos(theta) -sin(theta) 0 ;sin(theta) cos(theta) 0; 0 0 1];
roty=[cos(phi) 0 sin(phi); 0 1 0;-sin(phi) 0 cos(phi)];
rotz=[1 0 0;0 cos(alpha) -sin(alpha);0 sin(alpha) cos(alpha)];
Rot=rotz*roty*rotx;
tras=[0; -0.35; 0.1];
Twc=[Rot tras;0 0 0 1];%imu-cam
```

El valor elegido de *estimate_extrinsic* es 0 ya que los parámetros extrínsecos son completamente conocidos. En cuanto a los parámetros intrínsecos tanto de la cámara como de la IMU, son conocidos porque se han diseñado los *plugins* de ambos sensores para la simulación. Por tanto, se modificará



(a) (b)

Figura 5.16 Entorno de simulación creado en Gazebo.

el archivo de configuración siguiendo los parámetros indicados en dichos *plugins*.

Una vez se ha terminado de modificar el archivo anterior, se procede a lanzar el algoritmo. Los comandos utilizados se pueden ver en 5.18. La primera línea se encarga de lanzar la simulación creada del vehículo terrestre. La segunda línea, al igual que en el experiemento anterior, se utiliza para cambiar el formato de imagen RGB a formato monocromático. Las últimas líneas son las dedicadas a lanzar el algoritmo y el visualizador Rviz.

Código 5.18 Lazar experimento vehículo terrestre.

```
roslaunch myrobot3_gazebo myrobot3.launch
ROS_NAMESPACE=cochecito/camera_lat rosrun image_proc image_proc

roslaunch vins_estimator euroc.launch
roslaunch vins_estimator vins_rviz.launch
```

En primer lugar, se espera que la trayectoria estimada tenga forma rectangular debido al entorno de simulación creado en Gazebo (figura 5.16). Por otro lado, al ser un vehículo terrestre, sólo varían las coordenadas x e y de la trayectoria debido a que no varía la altura (sólo se desplaza en el plano XY).

Al realizar el experimento se obtiene una trayectoria como la mostrada en 5.17 (que se puede observar que es una trayectoria bastante rectangular y plana). Al utilizar la función de guardar la trayectoria estimada (indicada a través del parámetro *save_image*), la próxima vez que se realice una estimación con el algoritmo VINS-Mono, se lee la trayectoria que se ha guardado y, a partir de esta, se puede optimizar la nueva trayectoria fusionándola con la que ya se encuentra guardada (tal y como se explicó en el apartado 3.5.1). Este resultado se puede observar en la figura 5.19. En la imagen 5.18 se pueden observar tres trayectorias diferentes: la nueva trayectoria estimada, la trayectoria leída de un experimento anterior y la trayectoria que ha surgido de la fusión de las anteriores. En cambio, en la imagen 5.19, sólo se encuentra la trayectoria guardada del experimento anterior y la nueva trayectoria optimizada del nuevo experimento, observándose que las trayectorias son bastante parecidas. Se ha descartado la trayectoria estimada que difería bastante de la real porque era la que no tenía en cuenta la trayectoria ya guardada.

Es importante resaltar que, en este experimento, como el vehículo es controlado por teclado, la trayectoria que sigue el vehículo en la simulación no es siempre la misma. Es lógico que ambas trayectorias no sean totalmente iguales. No es como el primer experimento en el que siempre se

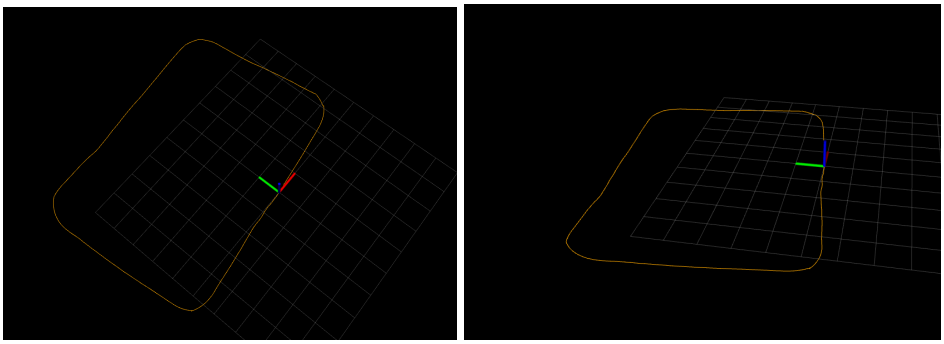


Figura 5.17 Trayectoria estimada del vehículo terrestre con algoritmo VINS-Mono.

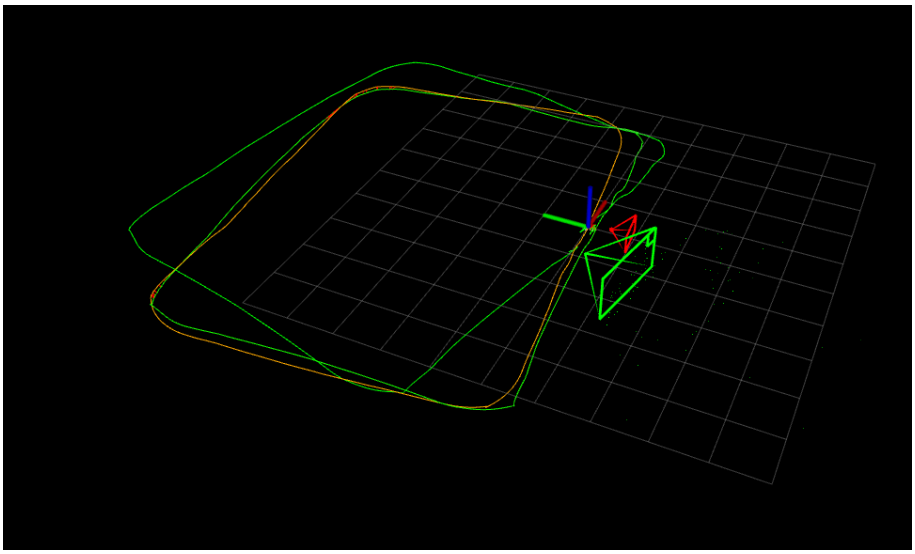


Figura 5.18 Trayectoria estimada del vehículo terrestre con algoritmo VINS-Mono al utilizar una trayectoria guardada de un experimento anterior.

estima la trayectoria con el mismo *dataset* y, por tanto, las trayectorias estimadas deben de ser muy parecidas.

Para evaluar el algoritmo se ha comparado la posición estimada con la posición real que se obtiene de Gazebo (del *plugin* de control). Como se ha definido el *plugin* de forma que las medidas reales obtenidas tengan el mismo sistema de referencia que las medidas estimadas, se pueden comparar los datos directamente sin tener que realizar transformaciones entre los sistemas de referencia.

En primer lugar, se han guardado tanto la posición estimada como la posición 3D real del vehículo durante el experimento en una *rosbag*. Estas medidas se publican en los tópicos */vins_estimator/odometry* y */odom*, respectivamente. El comando necesario será el siguiente:

Código 5.19 Guardar información del experimento en una rosbag.

```
rosbag record /vins_estimator/odometry /odom
```

Una vez se tiene esta información guardada en una *rosbag*, se va tratar en Matlab. En este programa, se van a obtener dos series temporales diferentes: una con la posición estimada y otra con la posición real. De esta forma, se puede evaluar la precisión del algoritmo.

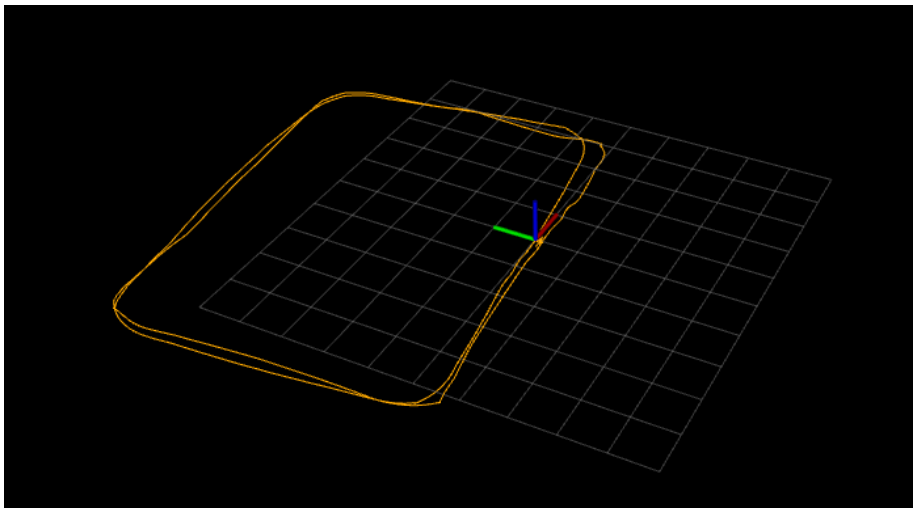


Figura 5.19 Trayectoria final guardada del vehículo terrestre con algoritmo VINS-Mono al utilizar una trayectoria guardada de un experimento anterior.

Código 5.20 Código de Matlab donde se tratan los datos de la rosbag.

```
bag1=rosbag('2020-11-07-11-12-30.bag')
bag1.AvailableTopics; %se pueden vr todos los tópicos guardados en la
    rosbag
start1=bag1.StartTime;
end1=bag1.EndTime;

bag1_odom=select(bag1,'Time',[start1+30 end1],'Topic','/vins_estimator/
    odometry')
bag1_gt=select(bag1,'Time',[start1+30 end1],'Topic','/odom')

%POSICION ESTIMADA Y POSICION REAL EN X
ts_x_odom=timeseries(bag1_odom,'Pose.Pose.Position.X');ts_x_gt=
    timeseries(bag1_gt,'Pose.Pose.Position.X');

%POSICION ESTIMADA Y POSICION REAL EN Y
ts_y_odom=timeseries(bag1_odom,'Pose.Pose.Position.Y');ts_y_gt=
    timeseries(bag1_gt,'Pose.Pose.Position.Y');

%POSICION ESTIMADA Y POSICION REAL EN Z
ts_z_odom=timeseries(bag1_odom,'Pose.Pose.Position.Z');ts_z_gt=
    timeseries(bag1_gt,'Pose.Pose.Position.Z');
```

Una vez se tienen por separado las medidas estimadas y las reales, se van a representar en una gráfica para poder ver la diferencia entre estos datos. Se puede observar en la figura 5.20. Además, para poder evaluar de forma más precisa este algoritmo, también se va a calcular el error en cada instante respecto a los ejes x, y y z, y el error cuadrático. El problema principal es que las medidas estimadas y reales se publican con una frecuencia diferente y, por tanto, las series temporales de cada tipo de medida tiene una longitud diferente. De esta forma es imposible calcular los errores. La solución elegida ha sido interpolar el vector de las medidas reales para que tenga el mismo tamaño que el de las medidas estimadas y, de esta forma, calcular los errores antes mencionados (y loS cuales se pueden observar en las figuras 5.21 y 5.22). El código completo utilizado en Matlab para

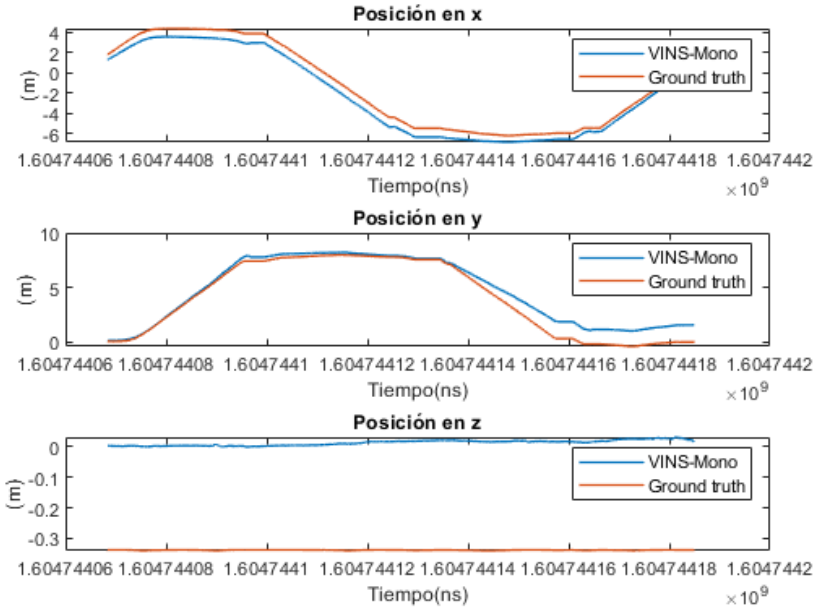


Figura 5.20 Comparación entre la trayectoria estimada y la trayectoria real del vehículo terrestre de Gazebo.

obtener todas las gráficas respecto a este experimento se puede ver en 7.8.

Código 5.21 Código de Matlab donde se tratan los datos de la rosbag.

```
odom_x=ts_x_odom.Data;odom_y=ts_y_odom.Data;odom_z=ts_z_odom.Data;%size
    es 838
odom_t=ts_x_odom.Time;
gt_x=ts_x_gt.Data;gt_y=ts_y_gt.Data;gt_z=ts_z_gt.Data;%size es 1596
gt_t=ts_x_gt.Time;

%interpolar gt para que tenga los mismos datos que odom
gt_x1=interp1(gt_t,gt_x,odom_t);gt_y1=interp1(gt_t,gt_y,odom_t);gt_z1=
    interp1(gt_t,gt_z,odom_t);

% Calculo error respecto cada eje
error_x=abs(odom_x-gt_x1);error_y=abs(odom_y-gt_y1);error_z=abs(odom_z-
    gt_z1);

%% Calcular error cuadratico
error_cuad=sqrt(error_x.^2+error_y.^2+error_z.^2);
```

Se puede observar que el eje en el cual se comete un error más chico es el eje z (lo cual es normal porque el vehículo no se mueve respecto a este eje y dicho error es prácticamente constante a lo largo de todo el experimento). Por el contrario, donde más error se produce es en el eje y, sobre todo en la segunda parte de la simulación. Esto se puede comprobar tanto en la gráfica del error respecto al eje y, como en la gráfica del error cuadrático. Esta última gráfica tiene en cuenta los errores respecto a los tres ejes, y se puede comprobar un gran aumento del error en la segunda

Figura 5.21 Errores respecto a los ejes x, y, z.

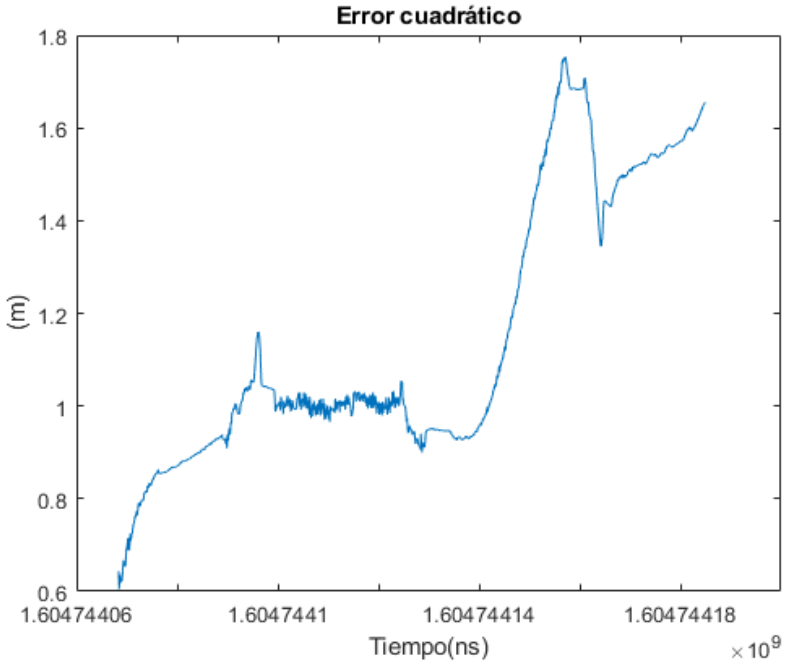


Figura 5.22 Error cuadrático de la estimación de la trayectoria del vehículo terrestre.

mitad. Este es debido al error respecto y (se puede ver que tiene casi el mismo valor, por lo que el resto de errores es despreciable frente el error de y).

6 Conclusión

El problema de VINS ha conseguido bastantes progresos durante los últimos 20 años. El objetivo principal que VINS persigue es conseguir la mayor precisión posible en sus estimaciones.

Se debe resaltar la importancia de VINS en los robots autónomos (se necesita conocer la localización de estos robots en todo momento). Para todas sus aplicaciones, es necesario conocer dónde se encuentran para poder saber cuál será el siguiente movimiento necesario. Además, una ventaja del algoritmo VINS-Mono es que funciona en lugares donde las medidas GPS son inaccesibles.

Como se ha comentado a lo largo de este trabajo, VINS-Mono es un estimador de estados 6 DOF visuo-inercial, el cual está basado en el método de optimización (consiguen mayor precisión y requieren menos memoria). Este algoritmo realiza la estimación de la trayectoria a partir una cámara monocular y una IMU. Para evaluar este algoritmo se han realizado diferentes experimentos y se han mostrado tanto las estimaciones obtenidas como los errores producidos en estas.

Los experimentos realizados han sido variados para poder evaluar el algoritmo. Por un lado, se ha probado VINS-Mono en vehículos aéreos (se corresponde al experimento del *dataset* EuRoC MAV y al experimento del Ardrone 2.0), y en vehículos terrestres (se corresponde con la simulación de Gazebo creada desde cero). Además, también se ha probado tanto en tiempo real como a partir de una bolsa de imágenes (o *rosbag*). De cada experimento se han calculado los errores para poder ver la calidad de la estimación.

Gracias a los experimentos realizados con los *datasets* EuRoC MAV se ha podido evaluar el algortimo en situaciones con diferente complejidad. Se ha observado que la trayectoria estimada por VINS-Mono sufre mayor error cuando la dificultad aumenta, pero igualmente obtiene muy buenos resultados.

Se van a remarcar algunos de los aspectos más importantes a tener en cuenta para que VINS-Mono realice una buena estimación de la trayectoria. Por un lado, se encuentra la calibración de los parámetros extrínsecos IMU-cámara (de todas formas, si no se tiene esta información, el algoritmo tiene la ventaja de poder calcularlos al comienzo del experimento y guardar esos valores para experimentos futuros). Por otro lado, también es muy importante la riqueza en texturas y elementos del entorno. El algoritmo es capaz de detectar escenas repetidas gracias a la observación de características comunes entre las imágenes. Por tanto, si las imágenes captadas por las cámaras tienen bastantes características a extraer, será mucho más fácil encontrar similitudes entre imágenes. Otro aspecto importante es la frecuencia mínima a la que deben publicar las medidas la IMU y la cámara, la cual es 100 Hz y 20Hz, respectivamente (esto es un requisito fundamental).

Aunque los algoritmos VINS han alcanzado bastante madurez, todavía existen muchos frentes abiertos para futuras investigaciones. A veces se pueden dar condiciones del entorno difíciles o degeneradas. Una de las posibles mejoras para la estimación puede ser la inclusión de más sensores con el fin de minimizar los errores, ó con el fin de mejorar el funcionamiento en entornos y situaciones donde la combinación cámara-IMU no dan una solución óptima.

7 Anexo

7.1 *myrobot.world*

Código 7.1 Archivo del mundo.

```
<?xml version="1.0" ?>
<sdf version="1.4">
  <world name="default">
    <include>
      <uri>model://ground_plane</uri>
    </include>
    <include>
      <uri>model://sun</uri>
    </include>


<!--CARRETERAS-->
    <road name="my_road1">
    <width>2</width>
    <point>-5 0 0</point>
    <point>5 0 0</point>
    </road>

    <road name="my_road2">
    <width>2</width>
    <point>4 1 0</point>
    <point>4 11 0</point>
    </road>
    <road name="my_road3">
    <width>2</width>
    <point>-6 -1 0</point>
    <point>-6 11 0</point>
    </road>
    <road name="my_road4">
    <width>2</width>
    <point>-5 8 0</point>
```

```
    <point>3 8 0</point>
    </road>

<!--ACERAS-->
    <road name="Pedestrian1">
      <width>1</width>
      <point>-5 1.5 0</point>
      <point>3 1.5 0</point>
      <material>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/Pedestrian</name>
        </script>
      </material>
    </road>

    <road name="Pedestrian2">
      <width>1</width>
      <point>-8 -1.5 0</point>
      <point>5 -1.5 0</point>
      <material>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/Pedestrian</name>
        </script>
      </material>
    </road>

    <road name="Pedestrian3">
      <width>1</width>
      <point>2.5 1.5 0</point>
      <point>2.5 11 0</point>
      <material>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/Pedestrian</name>
        </script>
      </material>
    </road>

    <road name="Pedestrian5">
      <width>1</width>
      <point>5.5 -2 0</point>
      <point>5.5 11 0</point>
      <material>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/Pedestrian</name>
        </script>
      </material>
```

```
    </road>

    <road name="Pedestrian6">
      <width>1</width>
      <point>-4.5 2 0</point>
      <point>-4.5 11 0</point>
      <material>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/Pedestrian</name>
        </script>
      </material>
    </road>

    <road name="Pedestrian7">
      <width>1</width>
      <point>-7.5 -2 0</point>
      <point>-7.5 11 0</point>
      <material>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/Pedestrian</name>
        </script>
      </material>
    </road>

    <road name="Pedestrian8">
      <width>1</width>
      <point>-4 6.5 0</point>
      <point>2 6.5 0</point>
      <material>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/Pedestrian</name>
        </script>
      </material>
    </road>

    <road name="Pedestrian9">
      <width>1</width>
      <point>-4 9.5 0</point>
      <point>2 9.5 0</point>
      <material>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/Pedestrian</name>
        </script>
      </material>
    </road>
```

```
<road name="grass0"><!--cesped columpio-->
      <width>4</width>
      <point>-4 4 0</point>
      <point>2 4 0</point>
      <material>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/Grass</name>
        </script>
      </material>
    </road>
<road name="grass1"><!--cesped fast food-->
      <width>6</width>
      <point>-10 -5 0</point>
      <point>6 -5 0</point>
      <material>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/Grass</name>
        </script>
      </material>
    </road>

<!--flechas-->
  <include>
      <uri>model://arrow_red</uri>
      <name>arrow_0</name>
      <pose>1 0 0.01 -2.5 -1.57 0.94</pose>
    </include>
  <include>
      <uri>model://arrow_red</uri>
      <name>arrow_1</name>
      <pose>3.10 0 0.01 -2.5 -1.57 0.94</pose>
    </include>
  <include>
      <uri>model://arrow_red</uri>
      <name>arrow_2</name>
      <pose>4 1.7 0.01 0 -1.57 0.0</pose>
    </include>
  <include>
      <uri>model://arrow_red</uri>
      <name>arrow_3</name>
      <pose>4 3.68 0.01 0 -1.57 0.0</pose>
    </include>
  <include>
      <uri>model://arrow_red</uri>
      <name>arrow_4</name>
      <pose>4 5.68 0.01 0 -1.57 0.0</pose>
    </include>
  <include>
```

```
      <uri>model://arrow_red</uri>
      <name>arrow_5</name>
      <pose>4 7.68 0.01 0 -1.57 0.0</pose>
    </include>
  <include>
      <uri>model://arrow_red</uri>
      <name>arrow_6</name>
      <pose>2.31 8 0 1.57 -1.57 0.0</pose>
    </include>
  <include>
      <uri>model://arrow_red</uri>
      <name>arrow_7</name>
      <pose>0.31 8 0 1.57 -1.57 0.0</pose>
    </include>
  <include>
      <uri>model://arrow_red</uri>
      <name>arrow_8</name>
      <pose>-1.66 8 0 1.57 -1.57 0.0</pose>
    </include>
  <include>
      <uri>model://arrow_red</uri>
      <name>arrow_9</name>
      <pose>-3.66 8 0 1.57 -1.57 0.0</pose>
    </include>

<!--CASAS-->
    <include>
      <uri>model://house_1</uri><!--se han modificado los .dae para que
          las casas sean mas pequenas-->
      <name>House1_0</name>
      <pose>8.32 4.7 0 0 0 -1.57</pose>
    </include>

    <include>
      <uri>model://house_1</uri>
      <name>House1_1</name>
      <pose>8.32 9.95 0 0 0 -1.57</pose>
    </include>

  <include>
      <uri>model://house_3</uri>
      <name>House3_0</name>
      <pose>-8.92 0.78 0 0 0 1.57</pose>
    </include>
  <include>
      <uri>model://house_3</uri>
      <name>House3_1</name>
      <pose>-8.92 5 0 0 0 1.57</pose>
    </include>
  <include>
```

```
      <uri>model://house_3</uri>
      <name>House3_2</name>
      <pose>-8.76 9.23 0 0 0 1.57</pose>
    </include>

  <include>
      <uri>model://house_2</uri>
      <name>House2_0</name>
      <pose>2.38 -4 0 0 0 0</pose>
    </include>

     <include>
      <uri>model://playground</uri>
      <name>playground</name>
      <pose>-1.06 3.84 0 0 0 0</pose>
    </include>

     <include>
      <uri>model://fountain</uri>
      <name>fountain</name>
      <pose>-6.4 -2.7 0 0 0 0</pose>
    </include>

     <include>
      <uri>model://post_office</uri>
      <name>post_office</name>
      <pose>7.27 -0.55 0 0 0 1.57</pose>
    </include>

     <include>
      <uri>model://school</uri>
      <name>school</name>
      <pose>-1.02 14 0 0 0 0</pose>
    </include>

<!--RESTAURANTE COMIDA RAPIDA Y MESAS-->
     <include>
      <uri>model://fast_food</uri>
      <name>fast_food</name>
      <pose>-2.79 -4.69 0 0 0 0</pose>
    </include>
     <include>
      <uri>model://cafe_table</uri>
      <name>cafe_table_0</name>
      <pose>-6.69 -5.29 0 0 0 0</pose>
    </include>
     <include>
      <uri>model://cafe_table</uri>
      <name>cafe_table_1</name>
      <pose>-8.08 -5.29 0 0 0 0</pose>
```

```
    </include>
     <include>
      <uri>model://cafe_table</uri>
      <name>cafe_table_2</name>
      <pose>-8.08 -6.68 0 0 0 0</pose>
    </include>
     <include>
      <uri>model://cafe_table</uri>
      <name>cafe_table_3</name>
      <pose>-6.69 -6.68 0 0 0 0</pose>
    </include>
     <include>
      <uri>model://person_standing</uri>
      <name>person_standing_0</name>
      <pose>5.81 1.74 0 0 0 -1.57</pose>
    </include>

<!--PERSONAS ESPERANDO COMIDA RAPIDA-->
     <include>
      <uri>model://person_standing</uri>
      <name>person_standing_1</name>
      <pose>-2.44 -1.48 0 0 0 0</pose>
    </include>
     <include>
      <uri>model://person_standing</uri>
      <name>person_standing_2</name>
      <pose>-1.48 -1.56 0 0 0 -1.57</pose>
    </include>
     <include>
      <uri>model://person_standing</uri>
      <name>person_standing_3</name>
      <pose>-0.77 -1.57 0 0 0 -1.57</pose>
    </include>
     <include>
      <uri>model://person_standing</uri>
      <name>person_standing_4</name>
      <pose>-0 -1.58 0 0 0 3.11</pose>
    </include>
     <include>
      <uri>model://person_standing</uri>
      <name>person_standing_5</name>
      <pose>0.66 -1.54 0 0 0 -1.82</pose>
    </include>
     <include>
      <uri>model://person_walking</uri>
      <name>person_walking_0</name>
      <pose>-1.22 1.51 0 0 0 1.57</pose>
    </include>
    <include>
      <uri>model://person_walking</uri>
```

```
      <name>person_walking_1</name>
      <pose>-0.22 9.2 0 0 0 -1.57</pose>
    </include>

  <include>
      <uri>model://stop_sign</uri>
      <name>stop_sign_0</name>
      <pose>-4.78 6.75 0 0 0 1.57</pose>
    </include>
  <include>
      <uri>model://stop_sign</uri>
      <name>stop_sign_1</name>
      <pose>2.83 -1.1 0 0 0 -1.57</pose>
    </include>

  <include>
      <uri>model://pine_tree</uri>
      <name>pine_tree_0</name>
      <pose>1.38 5.69 0 0 0 0</pose>
    </include>

  <include>
      <uri>model://lamp_post</uri>
      <name>lamp_post_0</name>
      <pose>2.56 1.2 0 0 0 0</pose>
    </include>

<!--BUZONES CORREO-->
  <include>
      <uri>model://mailbox</uri>
      <name>mailpost_0</name>
      <pose>5.3 0.86 0 0 0 0</pose>
    </include>
  <include>
      <uri>model://mailbox</uri>
      <name>mailpost_1</name>
      <pose>5.3 4.56 0 0 0 0</pose>
    </include>
  <include>
      <uri>model://mailbox</uri>
      <name>mailpost_2</name>
      <pose>5.3 7.8 0 0 0 0</pose>
    </include>
  <include>
      <uri>model://mailbox</uri>
      <name>mailpost_3</name>
      <pose>-7.3 -0.6 0 0 0 0</pose>
    </include>
  <include>
      <uri>model://mailbox</uri>
```

```
      <name>mailpost_4</name>
      <pose>-7.3 3.6 0 0 0 0</pose>
    </include>
  <include>
      <uri>model://mailbox</uri>
      <name>mailpost_5</name>
      <pose>-7.3 6.7 0 0 0 0</pose>
    </include>

  </world>
</sdf>
```

7.2 *myrobot.xacro*

Código 7.2 Creación del modelo del vehículo terrestre.

```
<?xml version="1.0"?>
<!-- Revolute-Revolute Manipulator -->
<robot name="my_robot3" xmlns:xacro="http://www.ros.org/wiki/xacro">

  <!-- Constants for robot dimensions -->
  <xacro:property name="PI" value="3.1415926535897931"/>
  <xacro:property name="chassis_mass" value="4" /> <!-- arbitrary value
     for mass -->
  <xacro:property name="offset" value="0.15" />
  <xacro:property name="width" value="0.6" /> <!-- Square dimensions (
     widthxwidth) of beams -->
  <xacro:property name="height" value="0.17" /> <!-- Link 1 -->
  <xacro:property name="depth" value="1" /> <!-- Link 2 -->
  <xacro:property name="wheel_mass" value="1.5" />
  <xacro:property name="camera_link" value="0.05" /> <!-- Size of square
      'camera' box -->
  <xacro:property name="imu_mass" value="0.01" />
  <xacro:property name="imu_link" value="0.02" /> <!-- Size of square '
     camera' box -->
  <xacro:property name="axel_offset" value="0.05" /> <!-- Space btw top
     of beam and the each j-->

  <!-- Import all Gazebo-customization elements, including Gazebo colors
      -->
  <xacro:include filename="$(find myrobot3_description)/urdf/myrobot3.
     gazebo" />
  <!-- Import Rviz colors -->
  <xacro:include filename="$(find myrobot3_description)/urdf/materials.
     xacro" />

  <!--BASE-->
  <link name="base">
```

```
  <inertial>
    <mass value="0.0001"/>
    <origin xyz="0 0 0"/>
    <inertia ixx="0.0001" ixy="0.0" ixz="0.0" iyy="0.0001" iyz="0.0"
       izz="0.0001"/>
  </inertial>
  <visual>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry>
      <box size="0.001 0.001 0.001"/>
    </geometry>
  </visual>
</link>

<joint name="base_chassis_joint" type="fixed">
  <origin rpy="0 0 0" xyz="0 0 0.34"/>
  <parent link="base"/>
  <child link="chassis"/>
</joint>

  <!--CHASSIS -->
<link name="chassis">
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/><!--Aqui indico donde esta el
       chassis del coche-->
    <geometry>
<box size="${depth} ${width} ${height}"/>
    </geometry>
  </collision>

  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
<!--<box size="${depth} ${width} ${height}"/>-->
<mesh scale="2 2 1" filename="package://myrobot3_description/meshes/
    chassis.dae"/>
    </geometry>
  </visual>

  <inertial>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <mass value="${chassis_mass}"/>
<!--0.16 valen todas las inercas q no son cero-->
    <inertia
  ixx="${1/12*chassis_mass*(width*width+height*height)}" ixy="0.0" ixz
     ="0.0"
  iyy="${1/12*chassis_mass*(depth*depth+height*height)}" iyz="0.0"
  izz="${1/12*chassis_mass*(depth*depth+width*width)}"/>
  </inertial>
```

```
  </link>


<!--Para calcular la inercia de las ruedas en las 3 ruedas hay que tener
     en cuenta de que tienen forma de discos y ademas los ejes de
    simetria son los ejes x e y-->

<!--Rueda derecha delantera-->
<link name="front_right_wheel">
    <visual>
      <origin rpy="1.57079632679 0 0 " xyz="0 0.15 0"/>
      <geometry>

  <mesh scale ="0.9 0.9 0.9" filename="model://car_wheel/meshes/
      car_wheel.dae"/>
      </geometry>
      <material name="DarkGray"/>
    </visual>
    <collision>
      <origin rpy="1.57079632679 0 0.2 " xyz="0 0.15 0"/>
      <geometry>
        <cylinder length="0.1" radius="0.2"/>
      </geometry>
    </collision>
    <inertial>
      <mass value="${wheel_mass}"/>
      <origin xyz="0 0.08 0"/>
     <inertia ixx="${0.25*wheel_mass*0.2*0.2}" ixy="0" ixz="0" iyy="$
         {0.5*wheel_mass*0.2*0.2}" iyz="0" izz="${0.25*wheel_mass
         *0.2*0.2}"/>
    </inertial>
  </link>

  <joint name="front_right_wheel_joint" type="continuous">
    <parent link="chassis"/>
    <child link="front_right_wheel"/>
    <origin rpy="0 0 0" xyz="0.25 0.3 -0.14"/>
    <axis rpy="0 0 0" xyz="0 1 0"/>
    <limit effort="100" velocity="100"/>
    <joint_properties damping="0.0" friction="0.0"/>
  </joint>

<!--Rueda izquierda delantera-->
  <link name="front_left_wheel">
    <visual>
      <origin rpy="1.57079632679 0 0 " xyz="0 0 0"/>
  <geometry>

  <mesh scale ="0.9 0.9 0.9" filename="model://car_wheel/meshes/
      car_wheel.dae"/>
```

```
      </geometry>
    </visual>
    <collision>
      <origin rpy="1.57079632679 0 0 " xyz="0 0 0"/>
      <geometry>
        <cylinder length="0.1" radius="0.2"/>
      </geometry>
    </collision>
    <inertial>
      <mass value="${wheel_mass}"/>
      <origin xyz="0 -0.08 0"/>
      <inertia ixx="${0.25*wheel_mass*0.2*0.2}" ixy="0" ixz="0" iyy="$
          {0.5*wheel_mass*0.2*0.2}" iyz="0" izz="${0.25*wheel_mass
          *0.2*0.2}"/>
    </inertial>
  </link>

  <joint name="front_left_wheel_joint" type="continuous">
    <parent link="chassis"/>
    <child link="front_left_wheel"/>
    <origin rpy="0 0 0" xyz="0.25 -0.3 -0.14"/> <!--aqui se indica donde
        se add la rueda-->
    <axis rpy="0 0" xyz="0 1 0"/>
    <limit effort="100" velocity="100"/>
    <joint_properties damping="0.0" friction="0.0"/>
  </joint>

<!--Rueda derecha trasera-->
<link name="back_right_wheel">
    <visual>
      <origin rpy="1.57079632679 0 0 " xyz="0 0.15 0"/>
      <geometry>
        <mesh scale ="0.9 0.9 0.9" filename="model://car_wheel/meshes/
            car_wheel.dae"/>
      </geometry>
      <material name="DarkGray"/>
    </visual>
    <collision>
      <origin rpy="1.57079632679 0 0 " xyz="0 0.15 0"/>
      <geometry>
        <cylinder length="0.1" radius="0.2"/>
      </geometry>
    </collision>
    <inertial>
      <mass value="${wheel_mass}"/>
      <origin xyz="0 0.08 0"/>
      <inertia ixx="${0.25*wheel_mass*0.2*0.2}" ixy="0" ixz="0" iyy="$
          {0.5*wheel_mass*0.2*0.2}" iyz="0" izz="${0.25*wheel_mass
          *0.2*0.2}"/>
    </inertial>
```

```
  </link>

  <joint name="back_right_wheel_joint" type="continuous">
    <parent link="chassis"/>
    <child link="back_right_wheel"/>
    <origin rpy="0 0 0" xyz="-0.25 0.3 -0.14"/>
    <axis rpy="0 0 0" xyz="0 1 0"/>
    <limit effort="100" velocity="100"/>
    <joint_properties damping="0.0" friction="0.0"/>
  </joint>

<!--Rueda izquierda trasera-->
  <link name="back_left_wheel">
    <visual>
      <origin rpy="1.57079632679 0 0 " xyz="0 0 0"/>
      <geometry>
        <mesh scale ="0.9 0.9 0.9" filename="model://car_wheel/meshes/
            car_wheel.dae"/>
      </geometry>
      <material name="DarkGray"/>
    </visual>
    <collision>
      <origin rpy="1.57079632679 0 0 " xyz="0 0 0"/>
      <geometry>
        <cylinder length="0.1" radius="0.2"/>
      </geometry>
    </collision>
    <inertial>
      <mass value="${wheel_mass}"/>
      <origin xyz="0 -0.08 0"/>
      <inertia ixx="${0.25*wheel_mass*0.2*0.2}" ixy="0" ixz="0" iyy="$
          {0.5*wheel_mass*0.2*0.2}" iyz="0" izz="${0.25*wheel_mass
          *0.2*0.2}"/>
    </inertial>
  </link>

  <joint name="back_left_wheel_joint" type="continuous">
    <parent link="chassis"/>
    <child link="back_left_wheel"/>
    <origin rpy="0 0 0" xyz="-0.25 -0.3 -0.14"/> <!--aqui se indica
        donde se add la rueda-->
    <axis rpy="0 0" xyz="0 1 0"/>
    <limit effort="100" velocity="100"/>
    <joint_properties damping="0.0" friction="0.0"/>
  </joint>


  <!-- Camera -->
 <joint name="camera_joint" type="fixed">
    <axis xyz="0 1 0" />
```

```
    <origin xyz="0.4 0 ${height/2}" rpy="0 0 0"/><!-- he estado
       cambiando el valor de xyz-->
    <parent link="chassis"/>
    <child link="camera_link"/>
  </joint>

  <link name="camera_link">
    <collision>
      <origin xyz="0 0 0.0" rpy="0 0 0"/>
      <geometry>
      <box size="${camera_link} ${camera_link} ${camera_link}"/>
      </geometry>
    </collision>

    <visual>
      <origin xyz="0 0 0.0" rpy="0 0 0"/>
      <geometry>
     <mesh scale ="0.5 0.5 0.5" filename="model://kinect/meshes/kinect.
        dae"/>
      </geometry>
      <material name="black"/>
    </visual>

    <inertial>
      <mass value="1e-5" />
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0" izz="1e-6"
         />
    </inertial>
</link>


  <!-- Camera lateral-->
 <joint name="camera2_joint" type="fixed">
    <axis xyz="0 1 0" />
    <origin xyz="0 -0.35 ${height/2}" rpy="0 0 -1.57"/><!-- se pone en
       un lado del coche-->
    <parent link="chassis"/>
    <child link="camera2_link"/>
  </joint>

  <link name="camera2_link">
    <collision>
      <origin xyz="0 0 0.0" rpy="0 0 0"/>
      <geometry>
      <box size="${camera_link} ${camera_link+0.1} ${camera_link}"/>
      </geometry>
    </collision>

    <visual>
```

```
      <origin xyz="0 0 0.0" rpy="0 0 0"/>
      <geometry>
  <box size="${camera_link} ${camera_link+0.1} ${camera_link}"/>
    <!-- <mesh scale ="0.5 0.5 0.5" filename="model://kinect/meshes/
       kinect.dae"/>-->
      </geometry>
      <material name="black"/>
    </visual>

    <inertial>
      <mass value="1e-5" />
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0" izz="1e-6"
          />
    </inertial>
  </link>



<!--IMU-->
 <joint name="imu_joint" type="fixed">
    <axis xyz="1 0 0"/> <!-- 0 1 0 -->
    <origin xyz="0 0 0"/>
    <parent link="chassis"/>
    <child link="imu_link"/>
  </joint>

<link name="imu_link">
  <inertial>
    <mass value="0.001"/>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <inertia ixx="0.0001" ixy="0" ixz="0" iyy="0.0001" iyz="0" izz
       ="0.0001"/>
  </inertial>

  <visual>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry>
      <box size="0.1 0.1 0.1"/>
    </geometry>
  </visual>

  <collision>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry>
      <box size="0.1 0.1 0.1"/>
    </geometry>
  </collision>
</link>
```

```
</robot>
```

7.3 *myrobot.gazebo*

Código 7.3 Archivo que contiene los *plugins* del vehículo terrestre.

```
<?xml version="1.0"?>
<robot>

  <gazebo reference="base">
    <turnGravityOff>false</turnGravityOff>
  </gazebo>

<!--Rueda derecha delantera-->
  <gazebo reference="front_right_wheel">
    <mu1 value="1.0"/>
    <mu2 value="1.0"/>
    <kp value="10000000.0"/>
    <kd value="1.0"/>
    <fdir1 value="1 0 0"/>
    <!--<material>Gazebo/Grey</material>-->
    <turnGravityOff>false</turnGravityOff>
  </gazebo>


<!--Rueda izquierda delantera-->
  <gazebo reference="front_left_wheel">
    <mu1 value="1.0"/>
    <mu2 value="1.0"/>
    <kp value="10000000.0"/>
    <kd value="1.0"/>
    <fdir1 value="1 0 0"/>
   <!-- <material>Gazebo/Grey</material>-->
    <turnGravityOff>false</turnGravityOff>
  </gazebo>

<!--Rueda derecha trasera-->
  <gazebo reference="back_right_wheel">
    <mu1 value="1.0"/>
    <mu2 value="1.0"/>
    <kp value="10000000.0"/>
    <kd value="1.0"/>
    <fdir1 value="1 0 0"/>
    <!--<material>Gazebo/Grey</material>-->
    <turnGravityOff>false</turnGravityOff>
  </gazebo>

<!--Rueda izquierda trasera-->
```

```
<gazebo reference="back_left_wheel">
  <mu1 value="1.0"/>
  <mu2 value="1.0"/>
  <kp value="10000000.0"/>
  <kd value="1.0"/>
  <fdir1 value="1 0 0"/>
 <!-- <material>Gazebo/Grey</material>-->
  <turnGravityOff>false</turnGravityOff>
</gazebo>

<!-- camera delantera PLUGIN-->
<gazebo reference="camera_link">
  <sensor name="camera" type="depth">
      <update_rate>30</update_rate><!--tiene que ser 20fps como min-->
      <camera>
        <horizontal_fov>1.047198</horizontal_fov>
        <image>
          <width>752</width><!--<width>640</width>--><!--cambiado para
              ue coincida con datos del euroc_config-->
          <height>480</height>
          <format>R8G8B8</format>
        </image>
        <clip>
          <near>0.05</near>
          <far>3</far>
        </clip>
      </camera>
<plugin name="camera_plugin" filename="libgazebo_ros_openni_kinect.so
    ">
        <baseline>0.2</baseline>
        <alwaysOn>true</alwaysOn>
        <!-- Keep this zero, update_rate in the parent <sensor> tag
          will control the frame rate. -->
        <updateRate>0.0</updateRate><!--numero de imagenes que toma la
            camara por segundo(si esta a cero la freq es a misma que la
             del parent sensor-->
        <cameraName>camera_ir</cameraName>
        <imageTopicName>/cam0/color/image_raw</imageTopicName>
        <cameraInfoTopicName>/cam0/color/camera_info</
            cameraInfoTopicName>
        <depthImageTopicName>/cam0/depth/image_raw</depthImageTopicName
            >
        <depthImageCameraInfoTopicName>/cam0/depth/camera_info</
            depthImageCameraInfoTopicName>
        <pointCloudTopicName>/cam0/depth/points</pointCloudTopicName>
        <frameName>camera_link</frameName>
        <pointCloudCutoff>0.5</pointCloudCutoff><!--min y max distancia
             entre puntos-->
        <pointCloudCutoffMax>3.0</pointCloudCutoffMax>
        <Cx>0</Cx>
```

```
          <Cy>0</Cy>
          <focalLength>0</focalLength>
          <hackBaseline>0</hackBaseline>

        </plugin>
      </sensor>
  </gazebo>

  <!-- camera lateral PLUGIN-->

   <gazebo reference="camera2_link">
      <sensor type="camera" name="camera_lat">
  <visualize>true</visualize>
        <update_rate>60</update_rate>
        <camera name="camera_lat">
          <horizontal_fov>1.413</horizontal_fov>
          <image>
            <width>640</width>
            <height>360</height>
            <format>R8G8B8</format>
          </image>

          <clip>
          <near>0.01</near>
              <far>100</far>
          </clip>
        </camera>
        <plugin name="camera_controller" filename="libgazebo_ros_camera.
           so">
    <cameraName>cochecito/camera_lat</cameraName>
          <alwaysOn>true</alwaysOn>
          <updateRate>60</updateRate>
          <imageTopicName>/cochecito/camera_lat/image_raw</imageTopicName
              >
          <cameraInfoTopicName>/cochecito/camera_lat/camera_info</
              cameraInfoTopicName>
          <frameName>cochecito_base_cam_lat</frameName>
        </plugin>
      </sensor>
      <turnGravityOff>true</turnGravityOff>
    </gazebo>-->


<!--IMU PLUGIN-->
   <gazebo reference="imu_link">
    <gravity>true</gravity>
    <sensor name="imu_sensor" type="imu">
      <always_on>true</always_on>
      <update_rate>200</update_rate>
      <topic>__default_topic__</topic>
```

```
    <plugin filename="libgazebo_ros_imu_sensor.so" name="imu_plugin">
      <topicName>imu0</topicName>
      <bodyName>imu_link</bodyName>
      <updateRateHZ>200.0</updateRateHZ>
      <gaussianNoise>0</gaussianNoise>
      <xyzOffset>0 0 0</xyzOffset>
      <rpyOffset>0 0 0</rpyOffset>
      <frameName>imu_link</frameName>

      <accelDrift>0.5 0.5 0.5</accelDrift>
      <accelGaussianNoise>0.35 0.35 0.3</accelGaussianNoise>
      <rateDrift>0.0 0.0 0.0</rateDrift>
      <rateGaussianNoise>0.00 0.00 0.00</rateGaussianNoise>
      <headingDrift>0.0</headingDrift>
      <headingGaussianNoise>0.00</headingGaussianNoise>

    </plugin>

  </sensor>
 </gazebo>

 <!--CONTROL-->
 <gazebo>
  <plugin name="object_controller" filename="libgazebo_ros_planar_move.
      so">
    <commandTopic>cmd_vel</commandTopic>
    <odometryTopic>odom</odometryTopic>
    <odometryFrame>odom</odometryFrame>
    <odometryRate>50.0</odometryRate>
    <robotBaseFrame>base_footprint</robotBaseFrame>
    <publishTF>false</publishTF>
  </plugin>
 </gazebo>
```

7.4 Archivo de configuración *euroc_config.yaml* del algoritmo VINS-Mono

7.4.1 Experimento EuRoC MAV Dataset

Código 7.4 *euroc_config.yaml* para el experimento de EuRoC MAV *Dataset*.

```
%YAML:1.0

#common parameters
imu_topic: "/imu0"
image_topic: "/cam0/image_raw"
output_path: "/home/mariarod/Escritorio/TFG/experimentos/dataset_EUROC4
   /"
```

```
#camera calibration
model_type: PINHOLE
camera_name: camera
image_width: 752
image_height: 480
distortion_parameters:
   k1: -2.917e-01
   k2: 8.228e-02
   p1: 5.333e-05
   p2: -1.578e-04
projection_parameters:
   fx: 4.616e+02
   fy: 4.603e+02
   cx: 3.630e+02
   cy: 2.481e+02

# Extrinsic parameter between IMU and Camera.
estimate_extrinsic: 0 # 0 Have an accurate extrinsic parameters. We will
     trust the following imu^R_cam, imu^T_cam, don't change it.
                     # 1  Have an initial guess about extrinsic
                         parameters. We will optimize around your
                         initial guess.
                     # 2  Don't know anything about extrinsic parameters
                         . You don't need to give R,T. We will try to
                         calibrate it. Do some rotation movement at
                         beginning.
#If you choose 0 or 1, you should write down the following matrix.
#Rotation from camera frame to imu frame, imu^R_cam
extrinsicRotation: !!opencv-matrix
   rows: 3
   cols: 3
   dt: d
   data: [0.0148655429818, -0.999880929698, 0.00414029679422,
          0.999557249008, 0.0149672133247, 0.025715529948,
          -0.0257744366974, 0.00375618835797, 0.999660727178]
#Translation from camera frame to imu frame, imu^T_cam
extrinsicTranslation: !!opencv-matrix
   rows: 3
   cols: 1
   dt: d
   data: [-0.0216401454975,-0.064676986768, 0.00981073058949]

#feature traker paprameters
max_cnt: 150          # max feature number in feature tracking
min_dist: 30          # min distance between two features
freq: 10              # frequence (Hz) of publish tracking result. At
    least 10Hz for good estimation. If set 0, the frequence will be same
     as raw image
F_threshold: 1.0      # ransac threshold (pixel)
show_track: 1         # publish tracking image as topic
```

```
equalize: 1           # if image is too dark or light, trun on equalize
    to find enough features
fisheye: 0            # if using fisheye, trun on it. A circle mask will
     be loaded to remove edge noisy points

#optimization parameters
max_solver_time: 0.04 # max solver itration time (ms), to guarantee real
     time
max_num_iterations: 8 # max solver itrations, to guarantee real time
keyframe_parallax: 10.0 # keyframe selection threshold (pixel)

#imu parameters     The more accurate parameters you provide, the better
     performance
acc_n: 0.08        # accelerometer measurement noise standard deviation.
     #0.2 0.04
gyr_n: 0.004       # gyroscope measurement noise standard deviation.
    #0.05 0.004
acc_w: 0.00004       # accelerometer bias random work noise standard
    deviation. #0.02
gyr_w: 2.0e-6     # gyroscope bias random work noise standard deviation.
     #4.0e-5
g_norm: 9.81007   # gravity magnitude

#loop closure parameters
loop_closure: 1                 # start loop closure
load_previous_pose_graph: 1     # load and reuse previous pose graph;
    load from 'pose_graph_save_path'
fast_relocalization: 1          # useful in real-time and large project
pose_graph_save_path: "/home/mariarod/Escritorio/TFG/experimentos/
    dataset_EUROC4/" # save and load path

#unsynchronization parameters
estimate_td: 0                  # online estimate time offset between
    camera and imu
td: 0.0                         # initial value of time offset. unit: s.
     readed image clock + td = real image clock (IMU clock)

#rolling shutter parameters
rolling_shutter: 0              # 0: global shutter camera, 1: rolling
    shutter camera
rolling_shutter_tr: 0           # unit: s. rolling shutter read out
    time per frame (from data sheet).

#visualization parameters
save_image: 1                # save image in pose graph for
    visualization prupose; you can close this function by setting 0
visualize_imu_forward: 0     # output imu forward propogation to achieve
     low latency and high frequence results
visualize_camera_size: 0.4   # size of camera marker in RVIZ
```

7.4.2 Experimento vehículo aéreo Gazebo

Código 7.5 *euroc_config.yaml* para el experimento del Ardrone 2.0 en Gazebo.

```
%YAML:1.0

#common parameters
imu_topic: "/ardrone/imu"
image_topic: "/ardrone/front/image_mono"
output_path: "/home/mariarod/Escritorio/TFG/experimentos/ardrone"

#camera calibration
model_type: PINHOLE
camera_name: camera
image_width: 640
image_height: 360
distortion_parameters:
   k1: 0
   k2: 0
   p1: 0
   p2: 0
projection_parameters:
   fx: 3.7467e+02
   fy: 3.7467e+02
   cx: 3.205e+02
   cy: 1.805e+02

# Extrinsic parameter between IMU and Camera.
estimate_extrinsic: 2 # 0 Have an accurate extrinsic parameters. We will
     trust the following imu^R_cam, imu^T_cam, don't change it.
                     # 1  Have an initial guess about extrinsic
                         parameters. We will optimize around your
                         initial guess.
                     # 2  Don't know anything about extrinsic parameters
                         . You don't need to give R,T. We will try to
                         calibrate it. Do some rotation movement at
                         beginning.
#If you choose 0 or 1, you should write down the following matrix.
#Rotation from camera frame to imu frame, imu^R_cam
extrinsicRotation: !!opencv-matrix
   rows: 3
   cols: 3
   dt: d
   data: [0.0148655429818, -0.999880929698, 0.00414029679422,
          0.999557249008, 0.0149672133247, 0.025715529948,
          -0.0257744366974, 0.00375618835797, 0.999660727178]
#Translation from camera frame to imu frame, imu^T_cam
extrinsicTranslation: !!opencv-matrix
   rows: 3
```

```
   cols: 1
   dt: d
   data: [-0.0216401454975,-0.064676986768, 0.00981073058949]

#feature traker paprameters
max_cnt: 150          # max feature number in feature tracking
min_dist: 30          # min distance between two features
freq: 10              # frequence (Hz) of publish tracking result. At
    least 10Hz for good estimation. If set 0, the frequence will be same
     as raw image
F_threshold: 1.0      # ransac threshold (pixel)
show_track: 1         # publish tracking image as topic
equalize: 1           # if image is too dark or light, trun on equalize
    to find enough features
fisheye: 0            # if using fisheye, trun on it. A circle mask will
     be loaded to remove edge noisy points

#optimization parameters
max_solver_time: 0.04 # max solver itration time (ms), to guarantee real
     time
max_num_iterations: 8 # max solver itrations, to guarantee real time
keyframe_parallax: 10.0 # keyframe selection threshold (pixel)

#imu parameters     The more accurate parameters you provide, the better
     performance
acc_n: 0.08        # accelerometer measurement noise standard deviation.
     #0.2 0.04
gyr_n: 0.004       # gyroscope measurement noise standard deviation.
    #0.05 0.004
acc_w: 0.00004       # accelerometer bias random work noise standard
    deviation. #0.02
gyr_w: 2.0e-6     # gyroscope bias random work noise standard deviation.
     #4.0e-5
g_norm: 9.81007   # gravity magnitude

#loop closure parameters
loop_closure: 1                 # start loop closure
load_previous_pose_graph: 1     # load and reuse previous pose graph;
    load from 'pose_graph_save_path'
fast_relocalization: 0          # useful in real-time and large project
pose_graph_save_path: "/home/mariarod/Escritorio/TFG/experimentos/
    ardrone" # save and load path

#unsynchronization parameters
estimate_td: 1                   # online estimate time offset between
    camera and imu
td: 0.0                          # initial value of time offset. unit: s.
     readed image clock + td = real image clock (IMU clock)

#rolling shutter parameters
```

```
rolling_shutter: 0              # 0: global shutter camera, 1: rolling
    shutter camera
rolling_shutter_tr: 0           # unit: s. rolling shutter read out
    time per frame (from data sheet).

#visualization parameters
save_image: 1                # save image in pose graph for
    visualization prupose; you can close this function by setting 0
visualize_imu_forward: 0     # output imu forward propogation to achieve
     low latency and high frequence results
visualize_camera_size: 0.4   # size of camera marker in RVIZ
```

7.4.3 Experimento vehículo terrestre Gazebo

Código 7.6 *euroc_config.yaml* para el experimento del vehículo terrestre en Gazebo.

```
%YAML:1.0

#common parameters
imu_topic: "/imu0"
image_topic: "/cochecito/camera_lat/image_mono"
output_path: "/home/mariarod/Escritorio/TFG/experimentos_coche"

#camera calibration
model_type: PINHOLE
camera_name: camera
image_width: 640
image_height: 360
distortion_parameters:
   k1: 0
   k2: 0
   p1: 0
   p2: 0
projection_parameters:
   fx: 3.7467e+02
   fy: 3.7467e+02
   cx: 3.205e+02
   cy: 1.805e+02

# Extrinsic parameter between IMU and Camera.
estimate_extrinsic: 0 # 0 Have an accurate extrinsic parameters. We will
     trust the following imu^R_cam, imu^T_cam, don't change it.
                     # 1  Have an initial guess about extrinsic
                         parameters. We will optimize around your
                         initial guess.
                     # 2  Don't know anything about extrinsic parameters
                         . You don't need to give R,T. We will try to
                         calibrate it. Do some rotation movement at
                         beginning.
```

```
#If you choose 0 or 1, you should write down the following matrix.
#Rotation from camera frame to imu frame, imu^R_cam
extrinsicRotation: !!opencv-matrix
   rows: 3
   cols: 3
   dt: d
   data: [-1, 0, 0,
          0, 0, -1,
          0, -1, 0]
  # data: [-0.869515, 0.0172587, 0.493606,
  #        -0.493865, -0.0172721, -0.869367,
  #        -0.00647855, -0.999702, 0.023548]
#Translation from camera frame to imu frame, imu^T_cam
extrinsicTranslation: !!opencv-matrix
   rows: 3
   cols: 1
   dt: d
   data: [0,-0.35, 0.1]
   #data: [-0.0216401454975,-0.064676986768, 0.00981073058949]

#feature traker paprameters
max_cnt: 150          # max feature number in feature tracking
min_dist: 30          # min distance between two features
freq: 10              # frequence (Hz) of publish tracking result. At
    least 10Hz for good estimation. If set 0, the frequence will be same
     as raw image
F_threshold: 1.0      # ransac threshold (pixel)
show_track: 1         # publish tracking image as topic
equalize: 1           # if image is too dark or light, trun on equalize
    to find enough features
fisheye: 0            # if using fisheye, trun on it. A circle mask will
     be loaded to remove edge noisy points

#optimization parameters
max_solver_time: 0.04 # max solver itration time (ms), to guarantee real
     time
max_num_iterations: 8 # max solver itrations, to guarantee real time
keyframe_parallax: 10.0 # keyframe selection threshold (pixel)

#imu parameters     The more accurate parameters you provide, the better
     performance
acc_n: 0.08        # accelerometer measurement noise standard deviation.
     #0.2 0.04
gyr_n: 0.004       # gyroscope measurement noise standard deviation.
    #0.05 0.004
acc_w: 0.00004       # accelerometer bias random work noise standard
    deviation. #0.02
gyr_w: 2.0e-6     # gyroscope bias random work noise standard deviation.
     #4.0e-5
g_norm: 9.81007   # gravity magnitude
```

```
#loop closure parameters
loop_closure: 1                 # start loop closure
load_previous_pose_graph: 1     # load and reuse previous pose graph;
    load from 'pose_graph_save_path'
fast_relocalization: 1          # useful in real-time and large project
pose_graph_save_path: "/home/mariarod/Escritorio/TFG/experimentos_coche"
     # save and load path

#unsynchronization parameters
estimate_td: 1                   # online estimate time offset between
    camera and imu
td: 0.0                          # initial value of time offset. unit: s.
     readed image clock + td = real image clock (IMU clock)

#rolling shutter parameters
rolling_shutter: 0               # 0: global shutter camera, 1: rolling
    shutter camera
rolling_shutter_tr: 0            # unit: s. rolling shutter read out
    time per frame (from data sheet).

#visualization parameters
save_image: 1                # save image in pose graph for
    visualization prupose; you can close this function by setting 0
visualize_imu_forward: 0     # output imu forward propogation to achieve
     low latency and high frequence results
visualize_camera_size: 0.4   # size of camera marker in RVIZ
```

7.5 Código de Matlab para obtener la representación de la posición y orientación relativa entre la cámara y la IMU

Código 7.7 Código Matlab para representar la disposición entre dos sensores.

```
clear all;

theta_ini=180;
phi_ini=0;
alpha_ini=90;

theta=theta_ini*(2*pi/360);%angulo que gira alrededor de z
phi=phi_ini*(2*pi/360); %angulo que gira alrededor de y
alpha=alpha_ini*(2*pi/360); %angulo que gira alrededor de x

rotx=[cos(theta) -sin(theta) 0 ;sin(theta) cos(theta) 0; 0 0 1];
roty=[cos(phi) 0 sin(phi); 0 1 0;-sin(phi) 0 cos(phi)];
rotz=[1 0 0;0 cos(alpha) -sin(alpha);0 sin(alpha) cos(alpha)];
Rot=rotz*roty*rotx;%primero se rota respecto de x, luego respecto de y,
    y por ultimo, respecto z
```

```
tras=[0; -0.3; 0.1];
Twc=[Rot tras;0 0 0 1];%imu-cam
Tcw=inv(Twc);%cam-imu

%Calculo ejes de la camara
origen_C=[0 0 0 1]';%origen coordenadas de C
finx_C=[0.1 0 0 1]';%Eje x respecto de C
finy_C=[0 0.1 0 1]';%Eje y respecto de C
finz_C=[0 0 0.1 1]';%Eje z respecto de C
origen_Cw=Twc*origen_C;
finx_Cw=Twc*finx_C; %Eje x respecto de W
finy_Cw=Twc*finy_C; %Eje y respecto de W
finz_Cw=Twc*finz_C; %Eje z respecto de W

figure()

line([origen_C(1),finx_C(1)],[origen_C(2),finx_C(2)],[origen_C(3),finx_C
    (3)],'Color','g')%ejex
line([origen_C(1),finy_C(1)],[origen_C(2),finy_C(2)],[origen_C(3),finy_C
    (3)],'Color','r')%ejey
line([origen_C(1),finz_C(1)],[origen_C(2),finz_C(2)],[origen_C(3),finz_C
    (3)],'Color','b') %ejez

line([origen_Cw(1),finx_Cw(1)],[origen_Cw(2),finx_Cw(2)],[origen_Cw(3),
    finx_Cw(3)],'Color','g')%ejex
line([origen_Cw(1),finy_Cw(1)],[origen_Cw(2),finy_Cw(2)],[origen_Cw(3),
    finy_Cw(3)],'Color','r')%ejey
line([origen_Cw(1),finz_Cw(1)],[origen_Cw(2),finz_Cw(2)],[origen_Cw(3),
    finz_Cw(3)],'Color','b') %ejez
```

7.6 Código de Matlab para calcular errores en la estimación de la trayectoria del experimento del vehículo terrestre

Código 7.8 Código Matlab para evaluar VINS-Mono en el experimento del vehículo terrestre en Gazebo.

```
close all; clear all;

bag1=rosbag('2020-11-07-11-12-30.bag')
bag1.AvailableTopics; %se pueden vr todos los tópicos guardados en la
    rosbag
start1=bag1.StartTime;
end1=bag1.EndTime;

bag1_odom=select(bag1,'Time',[start1+30 end1],'Topic','/vins_estimator/
    odometry')
bag1_gt=select(bag1,'Time',[start1+30 end1],'Topic','/ground_truth')
```

```
%POSICION ESTIMADA EN X
ts_x_odom=timeseries(bag1_odom,'Pose.Pose.Position.X');ts_x_gt=
    timeseries(bag1_gt,'Pose.Pose.Position.X');

%POSICION ESTIMADA EN Y
ts_y_odom=timeseries(bag1_odom,'Pose.Pose.Position.Y');ts_y_gt=
    timeseries(bag1_gt,'Pose.Pose.Position.Y');

%POSICION ESTIMADA EN Z
ts_z_odom=timeseries(bag1_odom,'Pose.Pose.Position.Z');ts_z_gt=
    timeseries(bag1_gt,'Pose.Pose.Position.Z');

%% Representar las gráficas de odometría y leica sin transformaciones
figure()
subplot(3,1,1);plot(ts_x_odom, 'LineWidth', 1);hold on;plot(ts_x_gt, '
    LineWidth', 1);
title("Posición en x");
ylabel("(m)");
xlabel("Tiempo(ns)");
legend('VINS-Mono','Ground truth');
subplot(3,1,2);plot(ts_y_odom, 'LineWidth', 1);hold on;plot(ts_y_gt, '
    LineWidth', 1);
title("Posición en y");
ylabel("(m)");
xlabel("Tiempo(ns)");
legend('VINS-Mono','Ground truth');
subplot(3,1,3);plot(ts_z_odom, 'LineWidth', 1);hold on;plot(ts_z_gt, '
    LineWidth', 1);
title("Posición en z");
ylabel("(m)");
xlabel("Tiempo(ns)");
legend('VINS-Mono','Ground truth');

%% Probando a interpolar para que tengan igual longitud
odom_x=ts_x_odom.Data;odom_y=ts_y_odom.Data;odom_z=ts_z_odom.Data;%size
    es 838
odom_t=ts_x_odom.Time;
gt_x=ts_x_gt.Data;gt_y=ts_y_gt.Data;gt_z=ts_z_gt.Data;%size es 1596
gt_t=ts_x_gt.Time;

%interpolar gt para que tenga los mismos datos que odom
gt_x1=interp1(gt_t,gt_x,odom_t);gt_y1=interp1(gt_t,gt_y,odom_t);gt_z1=
    interp1(gt_t,gt_z,odom_t);

% Calculo error respecto cada eje
error_x=abs(odom_x-gt_x1);error_y=abs(odom_y-gt_y1);error_z=abs(odom_z-
    gt_z1);

figure()
subplot(3,1,1);plot(odom_t,error_x);
```

```
title("Error respecto eje x");
ylabel("Error(m)");
xlabel("Tiempo(ns)");
subplot(3,1,2);plot(odom_t,error_y);
title("Error respecto eje y");
ylabel("Error(m)");
xlabel("Tiempo(ns)");
subplot(3,1,3);plot(odom_t,error_z);
title("Error respecto eje z");
ylabel("Error(m)");
xlabel("Tiempo(ns)");

%% Calcular error cuadratico
error_cuad=sqrt(error_x.^2+error_y.^2+error_z.^2);
figure()
plot(odom_t,error_cuad);
title('Error cuadrático');
ylabel('(m)');
xlabel('Tiempo(ns)');
```

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] *Paquete ardrone*, *http://wiki.ros.org/tum_simulator*.

[2] *Paquete control ardrone*, *https://github.com/Barahlush/ardrone_autopilot*.

[3] *Paquete control vehiculo terrestre*, *http://wiki.ros.org/teleop_twist_keyboard*.

[4] *Paquete image proc*, *http://wiki.ros.org/image_proc*.

[5] *Ros wiki*, *http://wiki.ros.org/es*.

[6] Sameer Agarwal, Keir Mierle, and Others, *Ceres solver*, *http://ceres-solver.org*.

[7] J.-Y. Bouguet, *Pyramidal implementation of the lucas kanade feature tracker*, 1999.

[8] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart, *The euroc micro aerial vehicle datasets*, The International Journal of Robotics Research (2016).

[9] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua, *Brief: Binary robust independent elementary features*, vol. 6314, 09 2010, pp. 778–792.

[10] Ondřej Chum, Jiří Matas, and Josef Kittler, *Locally optimized ransac*, Joint Pattern Recognition Symposium, Springer, 2003, pp. 236–243.

[11] Jeffrey Delmerico and Davide Scaramuzza, *A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots*, 2018 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2018, pp. 2502–2509.

[12] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza, *Imu preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation*, Georgia Institute of Technology, 2015.

[13] Dorian Gálvez-López and J. D. Tardós, *Bags of binary words for fast place recognition in image sequences*, IEEE Transactions on Robotics **28** (2012), no. 5, 1188–1197.

[14] Michael Grupp, *evo: Python package for the evaluation of odometry and slam.*, *https://github.com/MichaelGrupp/evo*, 2017.

[15] Gaël Guennebaud, Benoît Jacob, et al., *Eigen v3*, http://eigen.tuxfamily.org, 2010.

[16] Todd Lupton and Salah Sukkarieh, *Visual-inertial-aided navigation for high-dynamic motion in built environments without initial conditions*, IEEE Transactions on Robotics **28** (2011), no. 1, 61–76.

[17] Mrinal K Paul, Kejian Wu, Joel A Hesch, Esha D Nerurkar, and Stergios I Roumeliotis, *A comparative analysis of tightly-coupled monocular, binocular, and stereo vins*, 2017 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2017, pp. 165–172.

[18] Tong Qin, Peiliang Li, and Shaojie Shen, *Vins-mono: A robust and versatile monocular visual-inertial state estimator*, IEEE Transactions on Robotics **34** (2018), no. 4, 1004–1020.

[19] Tong Qin, Jie Pan, Shaozu Cao, and Shaojie Shen, *A general optimization-based framework for local odometry estimation with multiple sensors*, arXiv preprint arXiv:1901.03638 (2019).

[20] Shimon Ullman, *The interpretation of structure from motion*, Proceedings of the Royal Society of London. Series B. Biological Sciences **203** (1979), no. 1153, 405–426.

[21] Nam Van Dinh and Gon-Woo Kim, *Multi-sensor fusion towards vins: A concise tutorial, survey, framework and challenges*, 2020 IEEE International Conference on Big Data and Smart Computing (BigComp), IEEE, 2020, pp. 459–462.

[22] Y. Yang, J. Maley, and G. Huang, *Null-space-based marginalization: Analysis and algorithm*, 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017, pp. 6749–6755.

Índice alfabético