DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

TRABAJO DE TITULACIÓN, PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO ELECTRÓNICO EN AUTOMATIZACIÓN Y CONTROL

TEMA: "ESTIMACIÓN DE ODOMETRÍA VISUAL INERCIAL DE UN DISPOSITIVO MÓVIL BASADO EN FILTRO DE KALMAN"

AUTOR:

RODRíGUEZ COELLO, GUILLERMO ANTONIO

DIRECTOR: DR. AGUILAR CASTILLO, WILBERT GEOVANNY

SANGOLQUÍ

2018

DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

CERTIFICACIÓN

Certifico que el trabajo de titulación, "***ESTIMACIÓN DE ODOMETRÍA VISUAL INERCIAL DE UN DISPOSITIVO MÓVIL BASADO EN FILTRO DE KALMAN***" fue realizado por el señor ***GUILLERMO ANTONIO RODRÍGUEZ COELLO*** el mismo que ha sido revisado en su totalidad y analizado por la herramienta de verificación de similitud de contenido; por lo tanto cumple con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecidos por la Universidad de Fuerzas Armadas ESPE, razón por la cual me permito acreditar y autorizar para que lo sustenten públicamente.

Sangolqui, Agosto del 2018

.....

Ing. Wilbert Geovanny Aguilar Catillo PhD.

C.C: 0703844696

DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

AUTORÍA DE RESPONSIBILIDAD

Yo, ***RODRÍGUEZ COELLO, GUILLERMO ANTONIO***, declaro que el contenido, ideas y criterios del trabajo de titulación*:* "***ESTIMACIÓN DE ODOMETRÍA VISUAL INERCIAL DE UN DISPOSITIVO MÓVIL BASADO EN FILTRO DE KALMAN***" es de mi autoría y responsabilidad, cumpliendo con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecidos por la Universidad de Fuerzas Armadas ESPE, respetando los derechos intelectuales de terceros y referenciando las citas bibliográficas.

Consecuentemente el contenido de la investigación mencionada es veraz.

Sangolquí, Agosto de 2018.

.....

Guillermo Antonio Rodríguez Coello

CC: 0503434672

DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

AUTORIZACIÓN

Yo, ***RODRÍGUEZ COELLO, GUILLERMO ANTONIO,*** autorizo a la Universidad de las Fuerzas Armadas ESPE publicar el trabajo de titulación: "***ESTIMACIÓN DE ODOMETRÍA VISUAL INERCIAL DE UN DISPOSITIVO MÓVIL BASADO EN FILTRO DE KALMAN***" en el Repositorio Institucional, cuyo contenido, ideas y criterios son de mi responsabilidad.

Sangolquí, Agosto del 2018

.....

Guillermo Antonio Rodríguez Coello

CC: 0503434672

DEDICATORIA

Dedico este trabajo a mis padres María de los Ángeles y Antonio, a mi hermana Daniela y a mis abuelos Guillermo y Aída, quienes me han apoyado incondicionalmente, tanto económica como moralmente, para poder llegar a ser un profesional y cumplir mis sueños. Aun en los momentos más difíciles han sabido mantener su fuerza y su integridad.

Guillermo Antonio Rodríguez Coello

AGRADECIMIENTO

Agradezco a mis padres, María de los Ángeles y Antonio, por haberme hecho la persona que soy en la actualidad. Todos los logros que tengo en mi vida los debo a ustedes. Desde que nací me han enseñado a ser honrado, a ser respetuoso y a ayudar a los demás, siempre con cariño y con una sonrisa. Gracias a ustedes es que quiero ser cada día mejor.

Agradezco a mi hermana por sus consejos y su apoyo. Eres alguien a quien le cuento todo, una de las personas en las que confío más en esta vida. Gracias por hacerme reír aun en los momentos difíciles que hemos atravesado como familia.

Agradezco a mi abuelo Guillermo, un personaje ejemplar, una persona que ha dedicado su vida al beneficio de los demás. Es la persona más íntegra, honrada, inteligente, racional y solidaria que he conocido en mi vida. No puede imaginarse el orgullo que siento por ser su nieto. Ha sido una de las mayores motivaciones que he tenido en mi vida.

Agradezco a mi tutor de tesis el Doctor Wilbert Aguilar, quien me ha transferido una gran cantidad de conocimiento durante estos últimos años. Gracias a usted sé que con esfuerzo y dedicación cualquier persona puede cumplir sus sueños. Usted no solo ha sido un tutor, también ha sido un mentor.

Finalmente agradezco a mis amigos y compañeros, quienes han sabido soportar mi carácter y me han hecho darme cuenta de lo hermoso que es vivir. Sin su apoyo y preocupación no hubiera logrado alcanzar esta meta.

Guillermo Antonio Rodríguez Coello

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

RESUMEN

En el presente trabajo de investigación se presenta un sistema de odometría usando una cámara monocular y sensores inerciales. Recientes avances en el campo de la odometría visual han producido algoritmos de alta precisión. Sin embargo, debido al costo de los sensores y el equipo de investigación, estos avances son inaccesibles para personas fuera de la comunidad científica. El método utilizado en esta tesis es el Filtro de Kalman Multi-estado Restringido (MSCKF por sus siglas en ingles), el cual es una variante del EKF. Este algoritmo usa una ventana deslizante de poses pasadas de cámara, las cuales son usadas para triangular los puntos de interés observados y obtener una estimación precisa de los mismos en 3D. Este algoritmo fue implementado usando los datos inerciales y visuales de un dispositivo de fácil alcance y bajo costo, un teléfono inteligente. Los sensores inerciales y la cámara de este dispositivo necesitan ser calibrados antes de ser usados en el algoritmo, debido al ruido y los errores en la manufactura de los mismos. Múltiples pruebas fueron realizadas al algoritmo, obteniendo buenos resultados. Una aplicación para Android fue creada para la transmisión de los datos de la cámara y de los sensores inerciales.

PALABRAS CLAVE:

- **ODOMETRÍA**
- **DETECCIÓN Y DESCRIPCIÓN DE PUNTOS DE INTERÉS**
- **ESTIMACIÓN DE MOVIMIENTO**
- **FILTRO DE KALMAN**
- **ANDROID**

ABSTRACT

In this research, an odometry system using a monocular camera and inertial sensors is presented. Recent progress on the field of visual inertial odometry has produced high precision algorithms. However, these advances are inaccessible to people outside the scientific community, because of the cost of the sensors and research equipment. The method used in this thesis is the Multi-State Constrained Kalman Filter, a variation of the EKF. It uses a sliding window of past camera poses, which are used to triangulate the observed features to obtain a precise 3D position estimation of the features. The algorithm was implemented using the visual and inertial data of an easily accessible device, a smartphone. The camera and inertial sensors of the smartphone often need calibration before using them on the algorithm, due to the noise and the manufacturing errors of the sensors. Multiple tests were performed to the algorithm, obtaining excellent results. An Android app was developed to send the camera and inertial sensors data to the computer.

KEYWORDS:

- **ODOMETRY**
- **FEATURE POINTS**
- **MOTION ESTIMATION**
- **KALMAN FILTER**
- **ANDROID**

CAPITULO I

1. INTRODUCCIÓN

1.1 Antecedentes

La aplicación de sistemas basados en visión artificial se ha convertido en un elemento clave en aplicaciones de robótica móvil (Nister, Naroditsky, & Bergen, 2004). Los vehículos inteligentes han visto un gran desarrollo gracias a algoritmos de visión por computador (Matthies, Szeliski, & Kanade, 1988; Usenko, Engel, Stuckler, & Cremers, 2016). Propuestas desarrolladas para usar únicamente una cámara monocular para estimación de pose han sido demostradas en (Newcombe, Lovegrove, & Davison, 2011). Algunas razones para usar una cámara y no otro tipo de sensores disponibles es su reducido costo, peso y consumo energético (Clement, Peretroukhin, Lambert, & Kelly, 2015). Al estar basados únicamente en visión, estos métodos presentan problemas cuando hay movimientos rápidos, particularmente la rotación (Engel, Sturm, & Cremers, 2013), cuando la cámara se está moviendo en su eje focal, y en escenas con pocas esquinas para la detección de puntos de interés (Aguilar, Morales, Ruiz, & Abad, Real-Time Model-Based Video Stabilization for Microaerial Vehicles, 2016). Por esta razón se ha visto la necesidad de fusionar las mediciones realizadas por una Unidad de Medición Inercial (IMU por sus siglas en inglés) con las mediciones visuales.

Aproximaciones a la solución de Odometría Visual Inercial (VIO por sus siglas en inglés) (Li & Mourikis, High-precision, consistent EKF-based visual-inertial odometry, 2013; Bloesch, Omari, Hutter, & Siegwart, 2015), acoplan medidas inerciales con datos visuales y han

demostrado robustez a rotaciones rápidas, pérdida parcial de rastreo (tracking en inglés) y relativamente poco desvío (drift en inglés) con el tiempo. Varios trabajos (Li, Kim, & Mourikis, Real-time motion tracking on a cellphone using inertial sensing and a rolling-shutter camera, 2013; Leutenegger, Lynen, Bosse, Siegwart, & Furgale, 2014) han utilizado el Filtro de Kalman (en inglés Kalman Filter) (Kalman, 1960) para el tracking del movimiento de los datos visuales de la cámara. Permite hacer una intuitiva fusión de los datos de los sensores, y es altamente popular para algoritmos diseñados en plataformas móviles (Aguilar, y otros, On-Board Visual SLAM on a UGV Using a RGB-D Camera, 2017).

Tomando en cuenta estos antecedentes y con el objetivo de superar las dificultades que representa la implementación de estos sistemas con respecto a peso, consumo energético y especialmente costo, se propone en este proyecto de investigación desarrollar un algoritmo y la implementación del sistema de estimación de movimiento mediante una cámara monocular y una IMU, usando las librerías de OpenCV (Open Computer Vision por sus siglas en inglés) (The OpenCV Library, 2018) sobre el Sistema Operativo Robot (ROS por sus siglas en inglés) (ROS Powering the world's robots, 2018), utilizando como herramienta algorítmica el Filtro de Kalman para la fusión de datos.

1.2 Justificación

Estimar el movimiento de un robot en ambientes dinámicos y desconocidos tiene un rol muy importante en la localización y navegación de robots autónomos (Tanskanen, Naegeli, Pollefeys, & Hilliges, 2015; Aguilar & Morales, 3D Environment Mapping Using the Kinect V2 and Path Planning Based on RRT Algorithms, 2016; Aguilar, Morales, Ruiz, & Abad, RRT* GL Based Path Planning for Virtual Aerial Navigation, 2017). Existen múltiples métodos para estimar el movimiento de un robot (Aguilar & Angulo, Real-Time Model-Based Video Stabilization for

Microaerial Vehicles, 2016; Aguilar & Angulo, Robust video stabilization based on motion intention for low-cost micro aerial vehicles, 2014), aunque muchos de ellos tienen un costo de implementación muy elevado debido a los sensores que se usan. Algunos sensores usados para la estimación del movimiento de un robot son los siguientes:

- Sensor RGB-D: posee una cámara RGB y un sensor de proximidad. Produce estimaciones precisas, pero el precio y peso elevados del sensor además del alto consumo de energía hacen que sea inútil en un gran número de aplicaciones robóticas (Huang, y otros, 2016; Aguilar, y otros, Real-Time 3D Modeling with a RGB-D Camera and On-Board Processing, 2017).
- Sensor LIDAR: mide distancias por medio de un láser. Posee gran precisión, pero tiene un alto costo y consumo de energía (Zhang & Singh, 2014).
- Cámara estéreo: Consiste en dos cámaras monoculares a una distancia fija entre sí, simulando la visión estereoscópica de los seres humanos para realizar estimaciones de distancia. Tienen una precisión buena, pero ocupan más espacio y tienen un precio mayor que una cámara monocular convencional (Mouats, Aouf, Sappa, Aguilera, & Toledo, 2015).

En los últimos años, el tema de navegación visual inercial ha recibido una atención considerable en la comunidad robótica (Newcombe, Lovegrove, & Davison, 2011). Avances en la manufactura de sensores inerciales MEMS han hecho posible la construcción de IMUs (Inertial Measurement Unit por sus siglas en inglés) pequeñas, baratas y bastante precisas, adecuadas para la estimación de la pose de sistemas a baja escala, como son los robots móviles y los vehículos aéreos no tripulados (Aguilar, y otros, Visual SLAM with a RGB-D Camera on a Quadrotor UAV Using on-Board Processing, 2017). Estos sistemas frecuentemente son diseñados para ambientes urbanos donde las señales de GPS no son confiables, como por ejemplo dentro de un

edificio o cualquier espacio cubierto. El bajo costo, peso, y consumo de energía de las cámaras digitales las convierten en sensores ideales para complementar a los sistemas de navegación inercial. Un teléfono inteligente reúne estos sensores en un dispositivo compacto, ligero, con procesamiento interno, una batería que brinda portabilidad y distintas formas de trasmitir datos (Shelley, 2014). Es por esto que un teléfono inteligente es el dispositivo perfecto para la estimación de movimiento visual inercial (González, 2015).

Una ventaja importante de utilizar cámaras digitales es que las imágenes son mediciones con un alto contenido de información. Los métodos para la extracción de puntos de interés pueden detectar y rastrear cientos de estos puntos en las imágenes, lo cual puede resultar en excelentes resultados de localización. No obstante, las mediciones no son perfectas. Las cámaras poseen errores en su fabricación, afectando a las mediciones. Ademes, no se puede obtener una correcta estimación de la escala, ya que es difícil estimar profundidades con una cámara monocular. De la misma forma las mediciones de los sensores inerciales sufren de errores, los cuales van creciendo en el tiempo ya que estas mediciones tienen que ser integradas para obtener los valores estimados de velocidad y posición.

Combinando la información de la IMU y de la cámara podemos obtener una estimación más precisa. La información de la cámara puede ser utilizada para corregir los errores producidos durante la integración de las mediciones de la IMU, y al mismo tiempo, la IMU ofrece información sobre la escala, gracias al acelerómetro. Un algoritmo usando frecuentemente en la fusión de datos es el EKF (Extended Kalman Filter), el cual permite obtener la mejor estimación de la pose, y la incertidumbre que se tiene sobre esa pose.

La motivación principal de este proyecto es la realización de un sistema capaz de obtener odometría de manera robusta a bajo costo. Una de las formas de alcanzar esta propuesta es el uso

de una cámara monocular y un sensor IMU para obtener la Odometría Visual Inercial (VIO) de un dispositivo móvil.

El campo de la Odometría Visual Monocular (VO por sus siglas en inglés) ha visto enormes mejoras con respecto a precisión (Engel, Sturm, & Cremers, 2013), robustez y eficiencia y ha ganado bastante popularidad en los últimos años, haciendo de la cámara monocular de los sensores más comunes en la robótica moderna, además de tener un costo bajo y fáciles de manipular (Bloesch, Omari, Hutter, & Siegwart, 2015). Pero esto no significa que este método esté libre de errores. Normalmente, las cámaras son excelentes para identificar y seguir puntos de interés a velocidades bajas, pero con el incremento de la velocidad la precisión disminuye debido a la influencia de la difuminación por movimientos bruscos y el tiempo de muestreo de la cámara (Li & Mourikis, Improving the accuracy of EKF-based visual-inertial odometry, 2012). Además de estos inconvenientes, cambios en la luz y en el clima del ambiente pueden interferir en la interpretación de la información.

Uno de los sensores más comunes que logran resolver los problemas mencionados anteriormente es una IMU, ya que provee datos de posición, velocidad y postura en tiempo real sin tener interferencias por el clima, luz o movimientos bruscos. Una de sus desventajas es que sufre severamente de error acumulativo y puede dar problemas después de un largo tiempo de estimación (Mourikis & Roumeliotis, 2007).

Al fusionar la odometría visual obtenida mediante una cámara monocular y los parámetros de la IMU, se puede obtener una estimación de movimiento en tres dimensiones con mayor precisión (Mourikis & Roumeliotis, 2007). Existen gran cantidad de aplicaciones para esta implementación. Una de ellas es la estimación de la posición de personas en un ambiente cerrado conocido, ya que dispositivos como el Sistema de Posicionamiento Global (GPS por sus siglas en

inglés) no funcionan en estos espacios. Esto es especialmente útil en operaciones militares en donde sea necesario saber la ubicación de cada uno de los individuos que ingresan a un entorno cerrado, de esta forma información sobre la operación en tiempo real estará disponible en todo momento, permitiendo tomar decisiones con mayor eficacia. Esta implementación es también bastante útil en el campo de los vehículos autónomos, sean terrestres, aéreos o submarinos.

El presente proyecto de investigación propone un sistema de odometría visual inercial, basado en el EKF para la fusión de datos de la cámara monocular y la IMU presentes en un teléfono inteligente.

1.3 Alcance del Proyecto

El alcance del presente proyecto será la creación de un algoritmo capaz de realizar la estimación de movimiento de un dispositivo móvil utilizando solamente una cámara monocular y una IMU, basado en el Filtro de Kalman. Este trabajo es de tipo investigación – acción.

El dispositivo será utilizado únicamente para transmitir imágenes y datos de la IMU, las cuales se enviarán a un computador que realice el procesamiento del algoritmo. El computador tendrá como sistema operativo Ubuntu 14.04 y tendrá instalada la versión Indigo de ROS. La escritura del algoritmo se realizará mediante el lenguaje de programación Python, usando las librerías de OpenCV para facilitar la etapa de visión por computador del mismo.

Para la obtención de la odometría visual, se estudiarán e implementarán diferentes algoritmos y técnicas que realicen lo siguiente:

- Detección de puntos de interés
- Descripción de puntos de interés
- Emparejamiento (matching en inglés)

- Desestimación de valores atípicos (outliers en inglés)
- Estimación de la transformación geométrica
- Extracción de los parámetros de movimiento
- Optimización

Después de obtener la odometría visual se extraerán los datos de la IMU. Por medio del Filtro de Kalman se realizará la fusión entre los datos la odometría visual y los datos extraídos de la IMU, teniendo como producto final una estimación de movimiento mucho más robusta.

El sistema contará con una interfaz de usuario para poder visualizar la operación del algoritmo.

1.4 Notación

En la presente tesis se utiliza la siguiente notación:

Notación	**Ejemplo**
Escalares en minúsculas y negrita	$\mathbf{x}$
Vectores en minúsculas y cursiva	x
Matrices en mayúsculas	X
Un punto representa la derivada en tiempo continuo	$\dot{x}$
Un circunflejo representa un valor estimado	$\hat{x}$
Una barra representa un vector o cuaternión unitario	$\bar{q}$
Marcos de referencia en mayúsculas y negrita	B o $\{B\}$
El marco de referencia en la esquina superior izquierda	${}^{G}p$
Rotación del marco de referencia I al marco de referencia G	${}^{G}_{I}p$
El error diferencial se representa con Δ	$\Delta p = p - \hat{p}$

El error en la orientación se representa con δ $\delta\theta$

El error en general se representa con una tilde $\tilde{x} = x - \hat{x}$

1.5 Objetivos

1.5.1 Objetivo General

Desarrollar un sistema de estimación de movimiento basado en una cámara monocular y una IMU, basado en el Filtro de Kalman.

1.5.2 Objetivos Específicos

- Crear una vía de comunicación entre un dispositivo móvil que capture imágenes y datos de la IMU, y un computador que realice el procesamiento del algoritmo usando ROS y OpenCV.
- Implementar un algoritmo que realice la estimación de odometría visual a partir de las imágenes de la cámara monocular.
- Implementar un algoritmo que permita efectuar la fusión de datos entre los datos de la IMU y la odometría visual obtenida para reducir el error de la estimación, basado en el Filtro de Kalman.
- Analizar los resultados de la estimación de movimiento obtenida y compararlos con otras propuestas que se encuentran en la literatura.

CAPITULO II

2. MARCO TEÓRICO

2.1 Unidad de Medición Inercial (IMU)

Una Unidad de Medición Inercial (IMU por sus siglas en inglés) es un dispositivo electrónico compuesto usualmente por acelerómetros, giroscopios y, algunas veces, magnetómetros. Las IMU modernas son baratas y fáciles de producir, lo cual ha hecho que su uso se popularice en el campo de la robótica y en dispositivos como teléfonos inteligentes, vehículos aéreos no tripulados y gafas de realidad virtual.

En la práctica, desafortunadamente, estos sensores no entregan los valores reales de las magnitudes físicas que miden. Imperfecciones y desalineaciones en los componentes de los sensores, así como la calidad del sensor en general, producen pequeñas diferencias entre los valores medidos y los valores verdaderos. Para compensar estas imperfecciones, el modelo usado para describir las mediciones de los sensores tendrá la forma:

$$m_{medida} = m_{real} + b + n \tag{1}$$

Donde ***m*** representa a la magnitud física a ser medida, ya sea aceleración o velocidad angular. El valor de offset ***b*** (bias en inglés) es un valor aleatorio añadido al valor real. Este valor varía cada vez que el sensor es encendido, y puede cambiar lentamente durante su operación. El ruido ***n*** (noise en inglés) es también un valor aleatorio, pero varía cada vez que una nueva medida es realizada. Todos estos valores son tomados en cuenta para obtener una estimación óptima de la

magnitud real m_{real}. En la Fig. 1 ilustra el efecto que tienen el ruido y el bias sobre la señal de la magnitud física medida por el sensor (González, 2015).

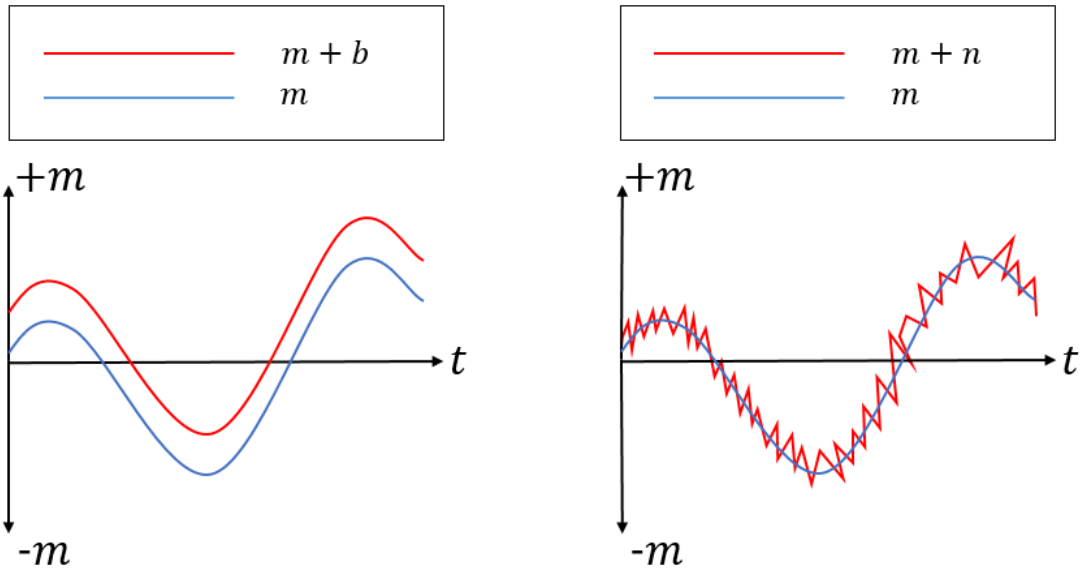

Figura 1. Ejemplos ilustrativos de los tipos de ruido que afectan a un sensor IMU.
Nota. Valor de offset (bias) a la izquierda, ruido (noise) a la derecha.

El acelerómetro y el giroscopio son de principal interés para el desarrollo de esta tesis, debido a que sus propiedades de ruido pueden ser modeladas. El magnetómetro, siendo una brújula que mide la dirección del campo magnético de la tierra en los 3 ejes, es altamente influenciable por campos magnéticos cercanos, haciéndolo difícil de modelar matemáticamente. En las siguientes secciones se explican a detalle los modelos usados para los acelerómetros y giroscopios.

2.1.1 Acelerómetro

Un acelerómetro mide la aceleración de un dispositivo relativo al estado de caída libre. Es por esta razón que el dispositivo en caída libre medirá una aceleración de 0, mientras que un dispositivo en reposo en la superficie de la tierra medirá una aceleración de aproximadamente 1 g ($9.81\, m/s^2$) debido al efecto de la gravedad.

Los acelerómetros reales son construidos con componentes electrónicos haciendo uso de tecnología de Sistemas Micro Electro-Mecánicos (MEMS por sus siglas en ingles). Están compuestos de una masa sujeta a un grupo de fibras flexibles. Como puede observarse en la Fig. 2, una aceleración aplicada a la masa produce un desplazamiento de la misma, lo cual afecta a la capacitancia que existe entre las placas capacitivas, teniendo así un valor medido proporcional a la aceleración en una dirección especifica (Thrun, Burgard, & Fox, 2010).

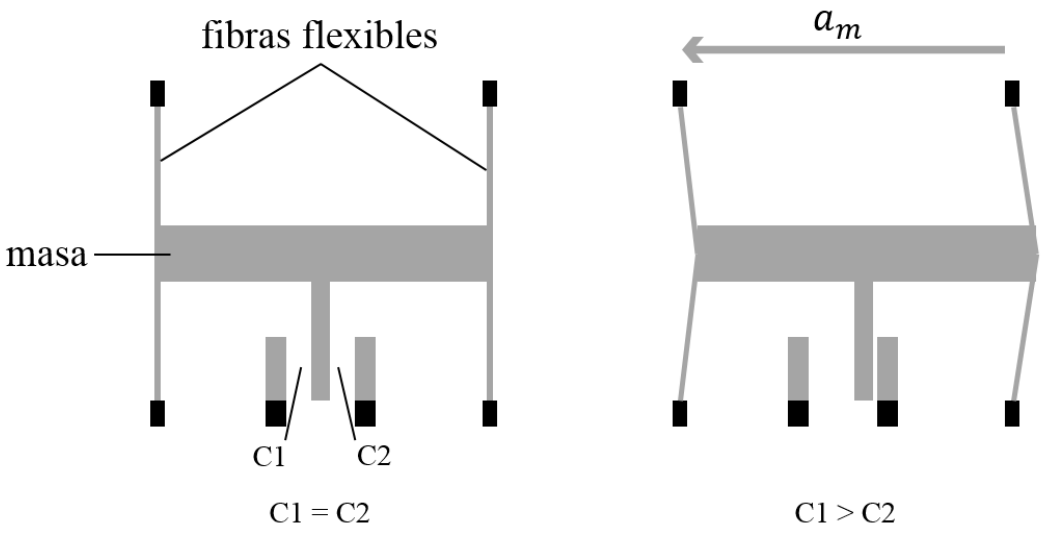

Figura 2. Ejemplo ilustrativo del funcionamiento del acelerómetro.

Cada acelerómetro mide aceleración solamente en un eje, por lo cual se los encuentra comúnmente como un grupo de tres dispositivos ortogonales en un mismo chip MEMS de bajo costo. Un acelerómetro ideal entregaría mediciones de la siguiente manera:

$$^{I}a_m = {}^{I}_{G}R({}^{G}a - {}^{G}g) \tag{2}$$

Donde ${}^{G}a$ es la aceleración real de la IMU con respecto al marco de referencia global y ***{I}*** representa el marco de referencia inercial de la IMU. El marco de referencia global tiene su eje z apuntando hacia arriba de la superficie de la tierra con $g = (0,0,-1)^T$. El subscrito ***m*** de a_m

denota que esa es la magnitud medida. Los acelerómetros de bajo costo, como los que disponen los teléfonos inteligentes, no son ideales y son afectados por ruido y bias, como mencionamos anteriormente. Un modelo más real del acelerómetro es el siguiente:

$$ {}^{I}a_m = {}^{I}_{G}R({}^{G}a - {}^{G}g) + n_a + b_a \tag{3} $$

El ruido n_a es una variable randómica de media cero y normalmente distribuida, $n_a\ N(0, N_a)$. El bias b_a cambia con el tiempo y se lo modela como un proceso de camino aleatorio impulsado por su propio vector de ruido $n_{wa}\ N(0, N_{wa})$. Los acelerómetros también pueden sufrir de errores de desalineación y de escala, pero en esta tesis se omite la explicación de su modelamiento ya que no van a ser parte de la implementación final por motivos de falta de tiempo (Shelley, 2014).

2.1.2 Giroscopio

Los giroscopios miden velocidad angular, ω, con respecto al marco de referencial inercial {I}. Su construcción es similar a la de los acelerómetros MEMS, teniendo tres giroscopios ortogonales en un mismo chip (González, 2015).

En la Fig. 3 se puede apreciar el principio de funcionamiento fundamental de los giroscopios, el cual se basa en inducir una velocidad a una masa en una dirección, y después medir en sentido ortogonal el movimiento causado por la fuerza de Coriolis cuando el dispositivo es rotado en cierta dirección. Ya que la masa no puede ser trasladada lejos del sensor, se usan dispositivos que inducen una oscilación en la masa.

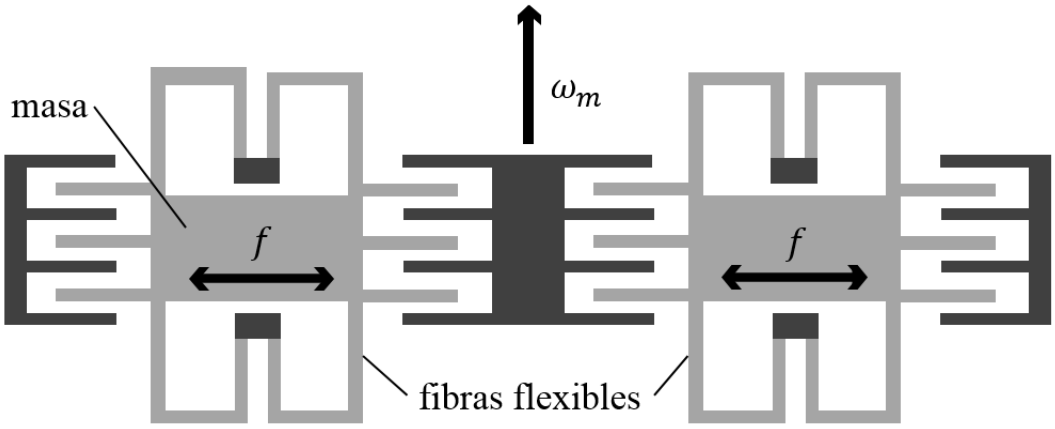

Figura 3. Ejemplo ilustrativo del funcionamiento del giroscopio.

Al igual que el acelerómetro, los giroscopios sufren de ruido, bias, errores de desalineación y errores de escala. Los errores de desalineación y de escala no serán tomados en cuenta en esta tesis. Entonces el modelo del giroscopio será el siguiente:

$$ {}^{I}\omega_m = {}^{I}\omega + n_g + b_g \tag{4}$$

El ruido n_g, al igual que el acelerómetro, es una variable randómica de media cero y normalmente distribuida, $n_g\ N(0, N_g)$. El bias b_a igualmente cambia con el tiempo y se lo modela como un proceso de camino aleatorio impulsado por su propio vector de ruido $n_{wg}\ N(0, N_{wg})$ (Thrun, Burgard, & Fox, 2010).

2.2 Cámara

2.2.1 Modelo de Cámara Pinhole

Las cámaras son sensores que proporcionan una gran cantidad de información sobre el lugar que las rodea. Una cámara mapea puntos desde un entorno 3D a imágenes en 2D, las mismas que pueden ser usadas para inferir localizaciones de objetos en el entorno. Estas localizaciones son las que van ser usadas para la estimación de la pose de la cámara (Shelley, 2014).

Existen varios modelos de cámara con diferentes niveles de complejidad y de precisión (Aguilar, Angulo, & Pardo, Motion intention optimization for multirotor robust video stabilization, 2017). El modelo utilizado en esta tesis es el modelo de cámara pinhole, por su simplicidad matemática (Aguilar & Angulo, Real-time video stabilization without phantom movements for micro aerial vehicles, 2014). Este modelo asume que no existe distorsión geométrica y que la cámara no tiene lente. Aunque este modelo es poco realista, es bastante popular ya que ofrece excelentes resultados en la práctica.

En el modelo de cámara pinhole, la cámara es representada como un punto en el espacio. La luz viaja desde el entorno visible por el hueco de la cámara hacia el plano de la imagen, donde una imagen es proyectada y capturada. Este punto conocido como punto focal. En la Fig. 4 se puede observar que la imagen capturada es una proyección invertida en 2D del entorno 3D capturado por la cámara. Para facilitar el modelamiento de la cámara, se utiliza el plano virtual de la imagen, el cual se encuentra entre el punto focal y la escena (González, 2015).

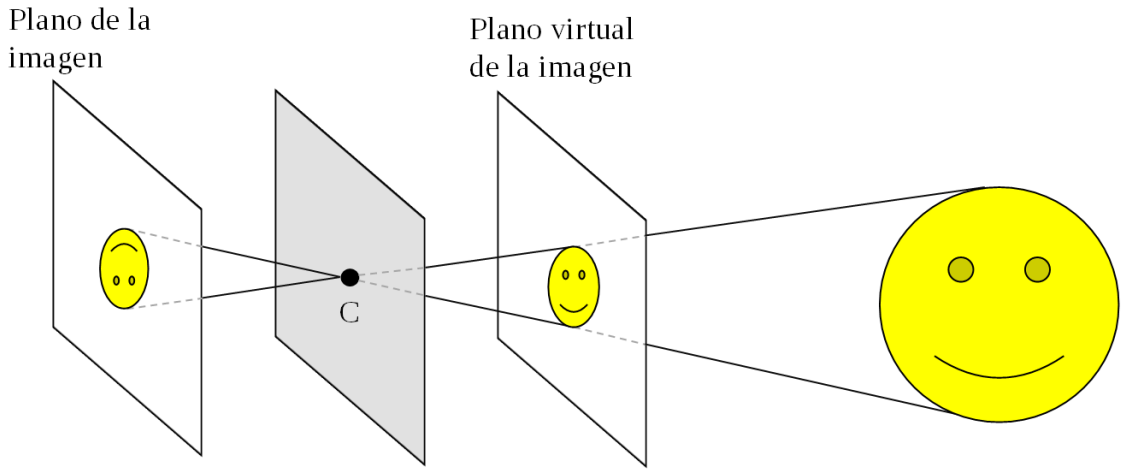

Figura 4. Ilustración del modelo de cámara pinhole.

2.2.2 Proyección en el entorno

Para poder utilizar correctamente la información obtenida de la cámara, es necesario proyectar los puntos 3D de la escena en el plano de la imagen, y que su ubicación sea descrita en pixeles. Mediante el modelo de cámara pinhole se puede encontrar una función sencilla que realice esta proyección. Usando algunos artificios trigonométricos se puede obtener esta función fácilmente.

La distancia más corta entre el punto focal y el plano de la imagen es conocida como la longitud focal f. Como se puede observar en la Fig. 5, utilizando la propiedad de semejanza de triángulos es posible encontrar la ubicación de un objeto en el plano de la imagen (González, 2015).

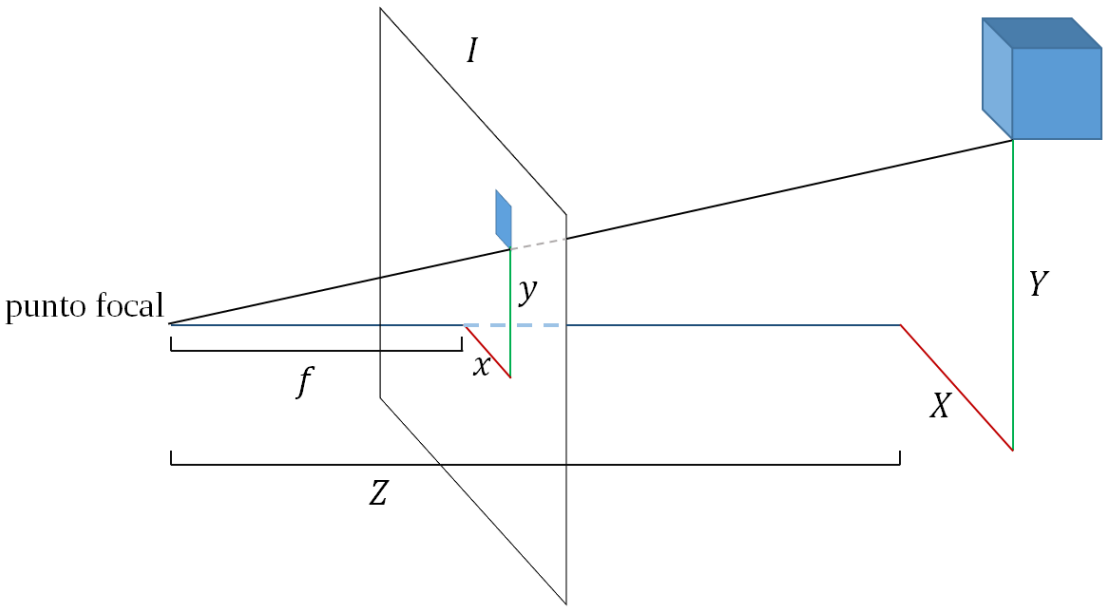

Figura 5. Proyección de un objeto 3D al plano de la imagen en 2D.

Un objeto de la escena proyectado el plano de la imagen de la cámara tiene una longitud focal en metros f, sus coordenadas de imagen $x_{imagen} = (x, y)^T$ y su posición en el espacio es ${}^{C}X = (X, Y, Z)^T$. Las coordenadas de la imagen pueden ser obtenidas aplicando las siguientes fórmulas:

$$x = f\frac{X}{Z} \qquad\qquad x = f\frac{y}{Z} \qquad\qquad (5)$$

Una vez obtenido x_{imagen}, que es la ubicación deseada en el plano de la imagen, sus unidades deben ser convertidas de metros a pixeles. Es necesario conocer el tamaño de los pixeles en metros para realizar esta conversión. Además de sus unidades, las coordenadas de imagen y de pixeles difieren en la ubicación del origen. En la Fig. 6 se puede apreciar que el origen de las coordenadas de pixeles está ubicado en la esquina superior izquierda.

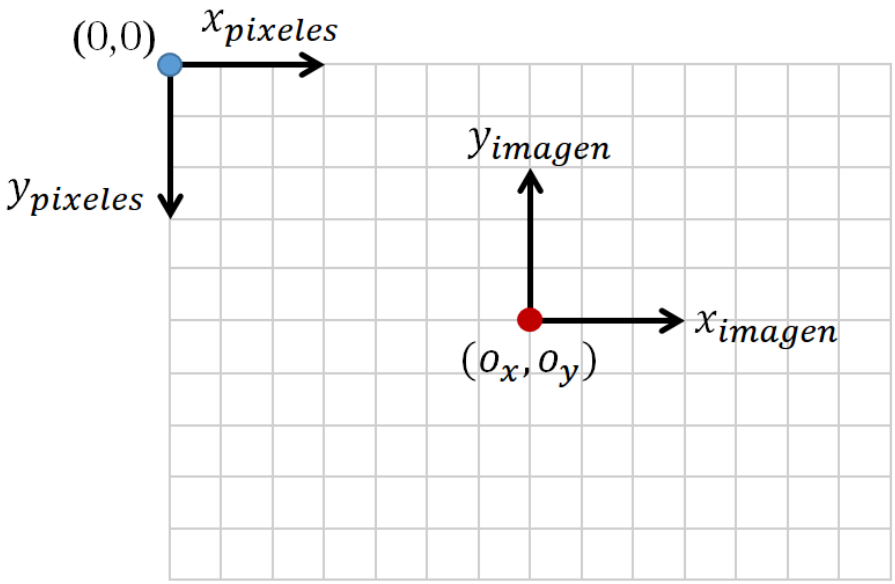

Figura 6. Coordenadas de imagen y coordenadas de pixeles.

Para convertir coordenadas de imagen a coordenadas de pixeles es necesario dividir los valores de las coordenadas para d, el tamaño de los pixeles en metros, y sumar el valor del punto central en coordenadas de pixeles. Ya que el sensor de la cámara es raramente un cuadrado, d va a tener un valor diferente para x y para y. Por simplicidad, definimos $f_x = \frac{f}{d_x}$ y $f_y = \frac{f}{d_y}$, siendo d_x el tamaño de los pixeles en x y d_y el tamaño de los pixeles en y. Entonces, la ubicación de un punto expresado en coordenadas de pixeles puede obtenerse de la siguiente forma:

$$x = f_x \frac{X}{Z} + o_x \qquad y = f_y \frac{Y}{Z} + o_y \tag{6}$$

Con estos valores es posible formar la función básica para la proyección de un objeto en el plano de la imagen de la cámara. Es importante recordar que los valores de o_x, o_y, f_x y f_y están expresados en pixeles.

$$\mathrm{h}\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{bmatrix} o_x \\ o_y \end{bmatrix} + \begin{bmatrix} f_x & 0 \\ 0 & f_y \end{bmatrix} \begin{bmatrix} \frac{X}{Z} \\ \frac{Y}{Z} \end{bmatrix} \tag{7}$$

Los parámetros o_x, o_y, f_x y f_y son comúnmente conocidos como los parámetros intrínsecos de la cámara, y son únicos para cada cámara. En esta tesis, estos parámetros son calculados mediante un programa de calibración escrito en Python (Python Programming Language, 2018) y utilizando las librerías de OpenCV, lo cual se detalla a profundidad en secciones posteriores.

2.2.3 Transformación de marco referencial

Si un punto es expresado en el marco referencial global en lugar del marco referencial de la cámara, debe ser transformado antes de utilizar las ecuaciones antes descritas. Un punto 3D llamado x, visto desde el marco referencial global, puede ser transformado al marco de la cámara de la siguiente manera:

$${}^{C}x = {}^{C}_{G}R({}^{G}x - {}^{G}p_c) \tag{8}$$

Donde ${}^{G}p_c$ es la posición de la cámara en el marco referencial global. Para transformar un punto en 3D en el marco de referencia global, ${}^{G}x$, a coordenadas de pixeles, la siguiente ecuación debe ser aplicada (Shelley, 2014).

$$\begin{bmatrix} u \\ v \end{bmatrix} = \mathrm{h}\left({}^{C}_{G}R({}^{G}x - {}^{G}p_c)\right) \tag{9}$$

Donde u y v representan la ubicaci ón del punto en coordenadas de pixeles.

2.2.4 Triangulación de puntos de interés

En secciones pasadas se discutió como extraer información de objetos del entorno por utilizando una cámara. Para usar efectivamente esta información, es necesario extraer las posiciones 3D de los puntos de interés observados. Esto se puede lograr mediante la triangulación de los puntos de interés (González, 2015).

Para explicar la importancia de triangular puntos de interés, se consideró el siguiente caso. Un punto de interés localizado en ${}^{G}p_f$ ha sido observado desde las ubicaciones de c ámara C_0, C_1, …, C_n y ha sido proyectado a los planos de imagen en los puntos p_0, p_1, …, p_n respectivamente. En un mundo ideal con cámara sin imperfecciones, errores y ruido, una línea trazada desde el centro de cada cámara y que pase por el pixel donde el punto de interés fue observado debería intersecar a ${}^{G}p_f$. En la Fig. 7 se puede observar este caso ideal para dos cámaras.

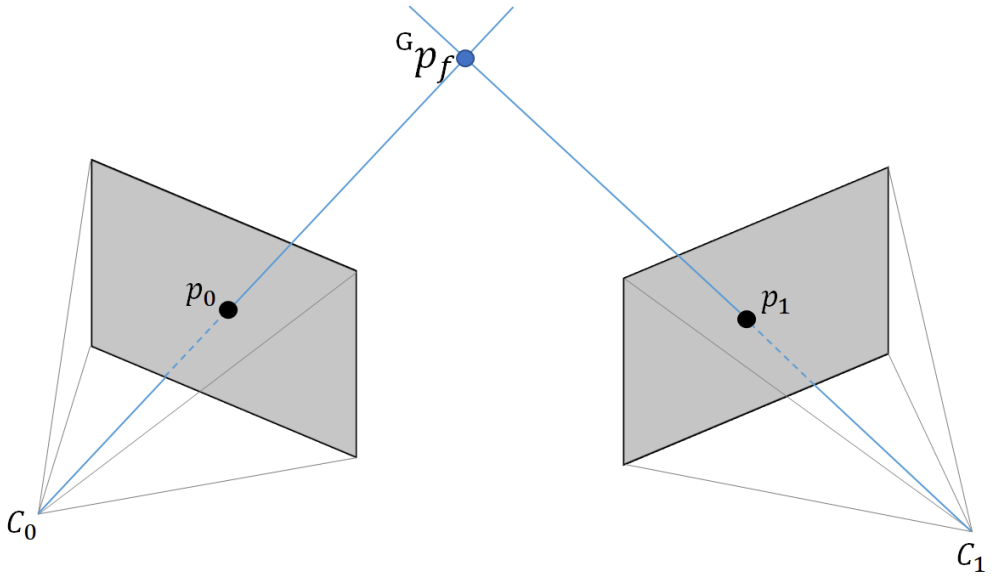

Figura 7. Triangulación ideal.

Sin embargo, en la práctica esta intersección perfecta no es posible debido a que los puntos de interés proyectados difieren de los reales por varias razones. Por ejemplo, las cámaras digitales

representan las imágenes en pixeles, los cuales son valores discretos, haciendo que las proyecciones no coincidan perfectamente con el centro del pixel. Además de esto, imperfecciones no tomadas en cuenta en el modelamiento de la cámara y la presencia de ruido en la imagen afectan a la medición de los puntos de interés. En la Fig. 8 se puede observar la triangulación real del punto ${}^{G}p_f$.

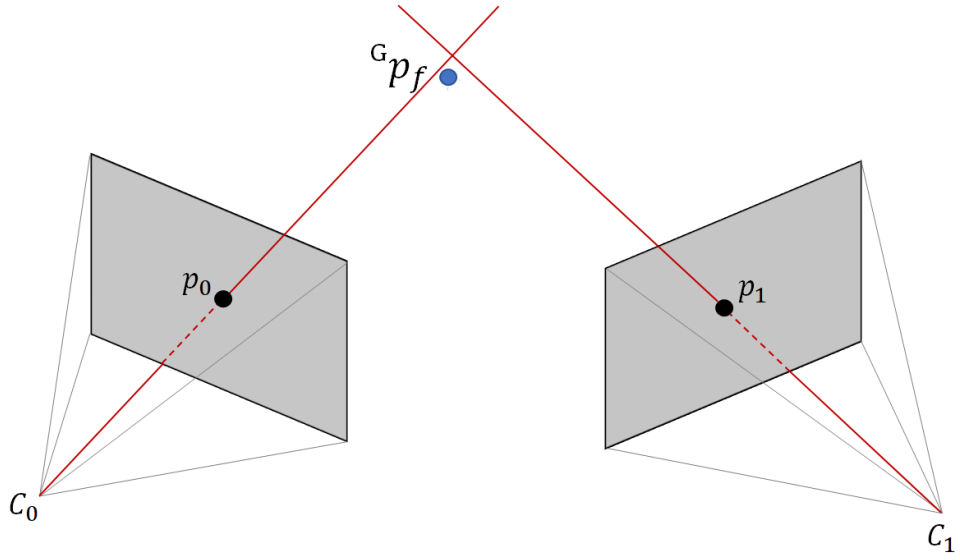

Figura 8. Triangulación real.

El objetivo de la triangulación es encontrar la mejor estimación de la ubicación del punto de interés. Para lograrlo, el problema debe ser formulado como un problema de minimización que puede ser resuelto mediante el algoritmo de Gauss-Newton, el cual es un método usado para la resolución de problemas no lineales de mínimos cuadrados. Para empezar, es necesario definir la función de error **f.**

$$\mathrm{f}_i(\theta) = z_i - h(\theta) \tag{10}$$

Donde θ es el parámetro, z_i es la medición y h la función que proyecta al parámetro al espacio de medición.

El punto de interés ${}^{G}p_f$ es observado por la cámara desde n ubicaciones, como se puede observar en la Fig.8. Dado un marco de referencia cualquiera de la cámara, C_i, la posición de ese punto de referencia en ese marco es ${}^{C_i}p_f = \left({}^{C_i}X, {}^{C_i}Y, {}^{C_i}Z\right)^T$, sabiendo que la cámara utiliza coordenadas de pixeles en la forma $(u_i, v_i)^T$. En este marco, definimos a la medición como $z_i = \mathrm{h}({}^{C_i}X, {}^{C_i}Y, {}^{C_i}Z)$, expresado en coordenadas de pixeles. El marco de referencia de C_0 es el marco de la cámara de donde el punto fue observado por primera vez. La posición del punto en el i-ésimo marco referencial de la cámara se puede obtener de la siguiente forma.

$$ {}^{C_i}p_f = {}^{C_i}_{C_0}R({}^{C_0}p_f - {}^{C_0}p_{C_i}) \tag{11} $$

$$ {}^{C_i}p_f = {}^{C_i}_{C_0}R\,{}^{C_0}p_f + {}^{C_i}p_{C_0} \tag{12} $$

Esta fórmula puede ser reescrita mediante la parametrización de profundidad inversa (Montiel, 2006). Esta parametrización es necesaria para tener estabilidad numérica y evitar mininos locales.

$$ {}^{C_i}p_f = {}^{C_i}_{C_0}R \begin{bmatrix} {}^{C_0}X \\ {}^{C_0}Y \\ {}^{C_0}Z \end{bmatrix} + {}^{C_i}p_{C_0} \tag{13} $$

$$ = {}^{C_0}Z \left({}^{C_i}_{C_0}R \begin{bmatrix} \frac{{}^{C_0}X}{{}^{C_0}Z} \\ \frac{{}^{C_0}Y}{{}^{C_0}Z} \\ 1 \end{bmatrix} + \frac{1}{{}^{C_0}Z}\,{}^{C_i}p_{C_0} \right) \tag{14} $$

$$ = {}^{C_0}Z \left({}^{C_i}_{C_0}R \begin{bmatrix} \alpha \\ \beta \\ 1 \end{bmatrix} + \rho\,{}^{C_i}p_{C_0} \right) \tag{15} $$

$$= {}^{C_0}Z g_i \begin{pmatrix} \alpha \\ \beta \\ \rho \end{pmatrix} \tag{16}$$

Donde g_i es una función tridimensional de α, β y ρ.

$$\alpha = \frac{{}^{C_0}X}{{}^{C_0}Z} \qquad \beta = \frac{{}^{C_0}Y}{{}^{C_0}Z} \qquad \rho = \frac{1}{{}^{C_0}Z} \tag{17}$$

Tomando en cuenta esto, ahora es posible redefinir la ecuación (10):

$$\mathrm{f}_i(\theta) = z_i - \mathrm{h}(g_i(\theta)), \text{ donde } \theta = \begin{pmatrix} \alpha \\ \beta \\ \rho \end{pmatrix} \tag{18}$$

Esta ecuación quiere decir lo siguiente: el punto 3D, parametrizado por θ en el marco referencial de C_0, es transformado al marco de referencia de C_i con g_i y proyectado a coordenadas de pixel por la función h, donde es comparado con la medición en pixeles hecha en ese marco de referencia. El error de la estimación puede ser minimizado actualizando iterativamente los parámetros utilizando el algoritmo de Gauss-Newton.

2.2.4.1 Minimización de Gauss-Newton

El algoritmo de Gauss-Newton (Gauss–Newton algorithm, 2018) es un método iterativo eficiente para encontrar los parámetros que minimizan la suma de cuadrados de una función (Shelley, 2014). Esta función es comúnmente mide el error entre el modelo y la observación.

$$S(\theta) = \sum_{i=1}^{n} \mathrm{f}_i(\theta)^2 \tag{19}$$

Para encontrar a los parámetros que minimizan a S, es necesario considerar una estimación inicial $\theta^{(0)}$. Cada iteración del algoritmo mejora la estimación actual mediante el cálculo del Jacobiano de f (las derivadas parciales de primer orden de f con respecto a los parámetros) y aproximando la función cuadráticamente, con lo cual es más sencillo minimizar la función. Los

parámetros son entonces movidos a este mínimo y el proceso se repite. Para calcular el Jacobiano de f**,** representado por J_f, se utiliza la regla de la cadena y se diferencia la función de proyección.

$$J_f = \frac{\partial f}{\partial \theta} = \frac{\partial h}{\partial g}\frac{\partial g}{\partial \theta} \tag{20}$$

$$\frac{\partial h}{\partial g} = \begin{bmatrix} f_x & 0 \\ 0 & f_y \end{bmatrix} \begin{bmatrix} \frac{d_r}{z} + \frac{\partial d_r}{\partial x}a + \frac{\partial d_t}{\partial x} & \frac{\partial d_r}{\partial y}a + \frac{\partial d_t}{\partial y} & -\frac{d_r}{z}a + \frac{\partial d_r}{\partial z}a + \frac{\partial d_t}{\partial z} \\ \frac{\partial d_r}{\partial x}b + \frac{\partial d_t}{\partial x} & \frac{d_r}{z} + \frac{\partial d_r}{\partial y}b + \frac{\partial d_t}{\partial y} & -\frac{d_r}{z}b + \frac{\partial d_r}{\partial z}b + \frac{\partial d_t}{\partial z} \end{bmatrix} \tag{21}$$

Donde $a = x/z$ y $b = y/z$. Estos parámetros, junto con $(x, y, z)^T$, representan la salida de la función g. Las derivadas parciales internas de d_r y d_t son bastante fáciles de calcular y han sido omitidas por simplicidad. Antes de calcular el Jacobiano de g, necesario reescribir la fórmula para nuestro caso.

$$g_i(\theta) = {}^{C_i}_{C_0}R \begin{bmatrix} \alpha \\ \beta \\ 1 \end{bmatrix} + \rho^{C_i} p_{C_0} \tag{22}$$

$$\frac{\partial g_i}{\partial \theta} = \left[\frac{\partial g_i}{\partial \alpha}, \frac{\partial g_i}{\partial \beta}, \frac{\partial g_i}{\partial \rho}\right] \tag{23}$$

$$= \left[{}^{C_i}_{C_0}R \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, {}^{C_i}_{C_0}R \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, {}^{C_i}p_{C_0}\right] \tag{24}$$

La función f_i y su Jacobiano J_{f_i} son calculados para n poses de la cámara y apilados para crear un vector f de $2n$ x 1 y una matriz J_f de $2n$ x 3. Estas funciones son usadas para actualizar los parámetros.

$$\theta^{(s+1)} = \theta^{(s)} - \left(J_f{}^T J_f\right)^{-1} J_f{}^T \, f(\theta^{(s)}) \tag{25}$$

Este proceso continua hasta que un número máximo de iteraciones ha sido alcanzado, o la función error f decrece hasta llegar a un valor umbral predefinido. Finalmente, la posición global del punto es calculada a partir de los parámetros estimados.

$$^{G}\hat{p}_{f} = \frac{1}{\hat{\rho}}\,{}^{G}_{C_0}R\begin{bmatrix}\alpha\\ \beta\\ 1\end{bmatrix} + {}^{G}p_{C_0} \tag{26}$$

2.2.5 Puntos de interés

En secciones pasadas se discutió como realizar la triangulación de un punto de interés capturado en imágenes a diferentes ubicaciones conocidas de la cámara. En esta sección se explica cómo son encontrados estos puntos de interés, y como son estos puntos relacionados entre imágenes (Aguilar, Verónica, & José, Obstacle Avoidance Based-Visual Navigation for Micro Aerial Vehicles, 2017; Aguilar, Angulo, & Costa-Castello, Autonomous Navigation Control for Quadrotors in Trajectories Tracking, 2017). Esencialmente, lo que se busca en las imágenes son áreas de interés que estén asociadas con objetos 3D fácilmente identificables. La mayoría de los algoritmos de detección de puntos de interés se enfocan en la detección de esquinas, las cuales son fáciles de identificar (Aguilar, Verónica, & José, Obstacle Avoidance for Low-Cost UAVs, 2017; Aguilar, Angulo, Costa, & Molina, 2014). El algoritmo de detección y descripción de puntos de interés usado en esta tesis es el SURF (Bay, Ess, Tuytelaars, & Gool, 2008), y para realizar el rastreo (conocido como tracking en inglés), el algoritmo utilizado fue el de Kanade-Lucas-Tomasi (Tomasi & Kanade, 1993).

2.2.5.1 SURF

SURF (Speeded Up Robust Features por sus siglas en inglés) es un detector y descriptor de puntos de interés (Bay, Ess, Tuytelaars, & Gool, 2008), derivado del famoso algoritmo SIFT (Burger & Burge, 2016). SURF aproxima el Laplaciano de Gaussiano con filtros de caja. La Fig. 9 muestra una demostración de tal aproximación.

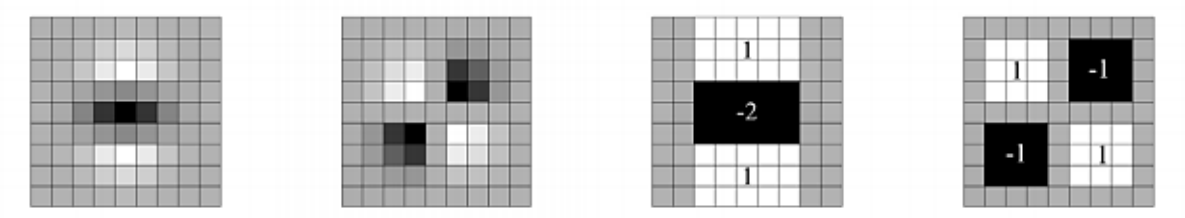

Figura 9. Aproximaciones para SURF.
Fuente: (Bay, Ess, Tuytelaars, & Gool, 2008)

Una gran ventaja de esta aproximación es que la convolución con el filtro de caja puede ser fácilmente calculada con la ayuda de imágenes integrales, y puede ser realizada en paralelo para diferentes escalas. SURF depende del determinante de la matriz Hessiana para la escala y la ubicación.

$$\det(H_{aprox} = D_{xx}D_{yy} - (wD_{xy})^2) \quad (27)$$

Para la asignación de la orientación, SURF usa respuestas de wavelets en direcciones horizontales y verticales para un vecindario de tamaño 6s. Los pesos Gaussianos adecuados son aplicados a esta orientación. Después, estos pesos son graficados en un espacio como el que se puede apreciar en la Fig. 10.

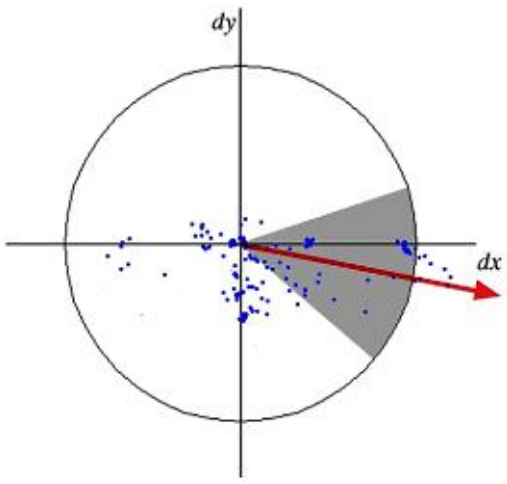

Figura 10. Asignación de la orientación en ventanas de pi/3.
Fuente: (Bay, Ess, Tuytelaars, & Gool, 2008)

La orientación dominante es estimada calculando la suma de todas las respuestas dentro de una ventana de orientación deslizante con un ángulo de 60 grados. La respuesta de wavelet puede ser fácilmente encontrada usando imágenes integrales a cualquier escala. Para muchas aplicaciones, la invariancia a la rotación no es necesaria, por lo cual la orientación encontrada puede ser omitida, haciendo que el proceso sea más rápido.

Para la descripción de puntos de interés, SURF usa respuestas de wavelet en dirección horizontal y vertical. De la misma forma que con la detección de puntos de interés, el uso de imágenes integrales facilita el proceso. Un vecindario de 20sx20s es tomado alrededor del punto de interés, donde s es el tamaño. Este vecindario es dividido en subregiones de 4x4. Para cada subregion, las respuestas de wavelet horizontales y verticales son utilizadas y un vector es formado de la siguiente forma:

$$v = \left(\sum d_x , \sum |d_x|, \sum d_y, \sum |d_y|\right) \tag{28}$$

Esta ecuación es representada como un vector le brinda al descriptor de puntos de interés de SURF un total de 64 dimensiones. Mientras la dimensión sea menor, la velocidad de procesamiento del algoritmo mejora, pero al costo de empeorar la diferenciación entre puntos de interés.

2.2.5.2 Rastreo KLT

El rastreo de puntos de objetos es una de las aplicaciones más importantes del campo de la visión por computador. El rastreador de Kanade-Lucas-Tomasi (KLT) rastrea un mismo punto de interés en diferentes imágenes (Tomasi & Kanade, 1993). Para usar este algoritmo, se deben realizar algunas suposiciones:

- Las intensidades de los pixeles de un objeto no deben cambiar entre cuadros consecutivos.

- Los pixeles vecinos tienen un movimiento similar.

Para entender el algoritmo, es necesario considerando un pixel $I(x, y, t)$ en la primera imagen que se mueve una distancia (dx, dy) en la próxima imagen después de un tiempo dt. Ya que esos pixeles tienen la misma intensidad, se puede asumir lo siguiente:

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{29}$$

Después, aplicando la serie de aproximación de Taylor en el lado derecho de la ecuación, removiendo términos comunes y dividiendo para dt obtenemos la siguiente ecuación:

$$f_x u + f_y v + f_t = 0 \tag{30}$$

Donde:

$$f_x = \frac{\partial f}{\partial x}; \quad f_y = \frac{\partial f}{\partial y} \tag{31}$$

$$u = \frac{\partial x}{\partial t}; \quad v = \frac{\partial y}{\partial t} \tag{32}$$

En estas ecuaciones podemos encontrar f_x y f_y ya que son gradientes de la imagen. Similarmente, f_t es la gradiente con respecto al tiempo. Sin embargo, u, v son desconocidos, entonces la ecuación no puede ser resuelta. Para solventar este problema, el algoritmo toma un área de 3x3 pixeles alrededor del punto de interés. Sabemos que los 9 pixeles tienen el mismo movimiento, por lo que podemos encontrar f_x, f_y, f_t para estos puntos. Ahora el problema se ha vuelto un sistema de 9 ecuaciones con dos incógnitas. La solución puede encontrarse resolviendo el siguiente sistema:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i {f_{x_i}}^2 & \sum_i f_{x_i} f_{y_i} \\ \sum_i f_{x_i} f_{y_i} & \sum_i {f_{y_i}}^2 \end{bmatrix} \begin{bmatrix} -\sum_i f_{x_i} f_{t_i} \\ -\sum_i f_{y_i} f_{t_i} \end{bmatrix} \tag{33}$$

2.2.5.3 Detección de Outliers

Para la detección de outliers (falsos positivos), se utilizó una variante de RANSAC, el MSAC (Torr & Zisserman, 2000). El estimador MSAC (M-estimator Sample Consensus por sus siglas en inglés) es un método usado para la estimación robusta de relaciones de múltiples vistas a partir de correspondencias de puntos. El algoritmo de RANSAC ha probado ser muy exitoso para estimaciones robustas, pero con la función logarítmica de probabilidad robusta negativa –L siendo la cantidad a ser minimizada, es aparente que RANSAC puede ser mejorado aún más. MSAC define a la función de pérdida como:

$$L(e) = \begin{cases} e^2 & |e| < c \\ c^2 & de\ otra\ manera \end{cases} \tag{34}$$

Uno de los problemas de RANSAC es que si el umbral T para considerar inliers es muy alto, entonces la estimación robusta puede ser bastante pobre. MSAC solventa este problema, y sin ningún costo computacional.

2.3 Filtro de Kalman

2.3.1 Filtro de Kalman tradicional

El filtro de Kalman, nombrado así por su creador Rudolf E. Kalman, tiene más de 50 años desde su primera concepción, pero es aun uno de los algoritmos de fusión de datos más comunes en la actualidad (Kalman, 1960). El éxito del filtro de Kalman se debe a su bajo coste computacional y sus características de recursividad (Thrun, Burgard, & Fox, 2010). El filtro de Kalman tiene un amplio rango de aplicaciones, como receptores de GPS, el suavizado del ruido de la señal de un mouse de laptop, estimación de trayectorias de drones (Aguilar W. G., y otros, Pedestrian Detection for UAVs Using Cascade Classifiers and Saliency Maps, 2017; Aguilar W. G., y otros, 2017), el control de temperatura de los motores de cohetes espaciales, detección y

seguimiento de personas (Aguilar W. G., y otros, Real-Time Detection and Simulation of Abnormal Crowd Behavior, 2017; Aguilar W. G., y otros, Cascade Classifiers and Saliency Maps Based People Detection, 2017) y muchas más. Esta tesis se concentra en el uso del filtro de Kalman para la estimación de pose mediante una cámara monocular y un sensor inercial.

El filtro de Kalman está basado en la teoría de probabilidad (The Extended Kalman Filter: An Interactive Tutorial, 2014). Cuando rastreamos la posición de un robot, deseamos saber su posición probable más que su posición explicita, para así mantener un seguimiento de la incertidumbre generada por nuestra estimación, y posteriormente utilizar esta incertidumbre para calcular el siguiente estado. En esta tesis se asume que conceptos de teoría de probabilidad (distribuciones normales, funciones de densidad de probabilidad, esperanza matemática, covarianza, intersecciones, etc.) son dominados por el lector.

El filtro de Kalman estima un proceso usando una forma de control retroalimentado: el filtro estima el estado del proceso en un tiempo $\boldsymbol{t}$ y luego obtiene su retroalimentación en forma de mediciones ruidosas. Las ecuaciones del filtro de Kalman pueden ser divididas en dos diferentes grupos: ecuaciones de predicción y ecuaciones de actualización (también llamada corrección). Las ecuaciones de predicción son responsables por proyectar adelante en el tiempo al estado actual y los errores de las covarianzas estimadas para obtener los estimados necesarios para la siguiente etapa. Las ecuaciones de actualización son responsables por incorporar las mediciones a los estimados obtenidos en la etapa anterior, para obtener un estimado mejorado del estado (Thrun, Burgard, & Fox, 2010). En la Figura 11 se puede apreciar una ilustración del funcionamiento de este ciclo.

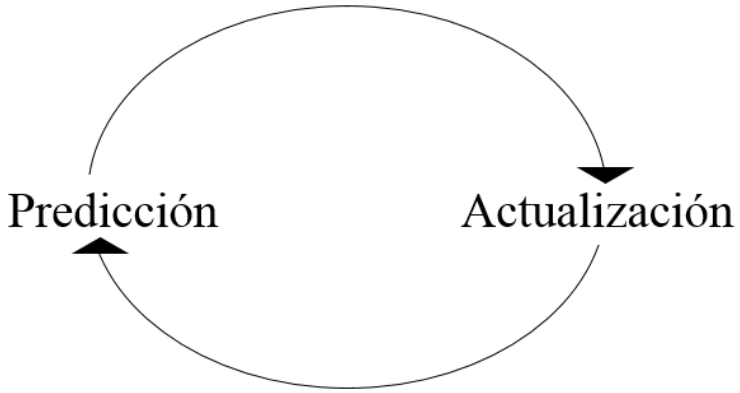

Figura 11. El ciclo del filtro de Kalman.

En la práctica, ningún proceso carece de errores, y si se quiere tener una buena estimación, estos errores deben ser modelados y compensados. Como se puede observar en la Figura 12, se asume que el modelo del sistema y las mediciones realizadas por los sensores son afectadas por ruido (Thrun, Burgard, & Fox, 2010). En ambos casos, se asume un ruido Gaussiano blanco.

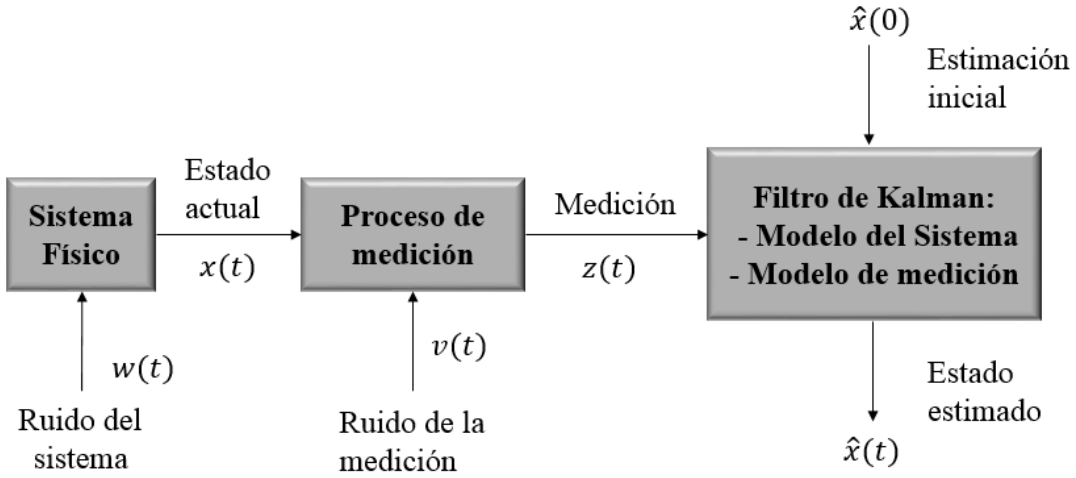

Figura 12. Modelamiento de sistemas en el Filtro de Kalman.

Este filtro es altamente útil, entre otras cosas, para la estimación de pose para sistemas lineares (Shelley, 2014). Si se tiene un conocimiento previo del sistema, podemos calcular la función de

densidad de probabilidad del próximo estado dado el estado actual. La relación lineal entre en próximo estado y el estado actual como:

$$x_{t+1} = A_t x_t \tag{35}$$

Esta información puede ser usada para propagar la función de densidad de probabilidad. Si

$$x_t \sim N(\mu_t, \Sigma_t) \tag{36}$$

Por medio de intersección se obtiene:

$$x_{t+1} \sim N(A_t \mu_t, A_t \Sigma_t A_t^T) \tag{37}$$

Si existe más información sobre la forma en que $\boldsymbol{x}$ cambia, como alguna acción o medida que sea independiente del estado actual, se puede usar esta información para propagar la probabilidad. En algunas aplicaciones del filtro de Kalman, $\boldsymbol{u}$ es conocido como la entrada de control y describe los comandos de movimiento enviados al robot. Sin embargo, es totalmente posible que $\boldsymbol{u}$ represente una medición de un sensor, siempre y cuando este sensor sea independiente de $\boldsymbol{x}$ y tenga una relación lineal con la misma. Cualquier sea el caso, existe un ruido en el modelo. Este ruido incrementa la incertidumbre en la estimación de la pose, y tiene la forma:

$$w_t \sim N(0, Q) \tag{38}$$

A partir de estas consideraciones se obtiene:

$$x_{t+1} = A_t x_t + B_t u_t + w_t \tag{39}$$

Para encontrar a $p(x_{t+1})$, primero se puede usar intersección para encontrar $p(x_{t+1}, x_t)$ y marginalizar x_t.

$$p(x_{t+1}, x_t) = N\left(\begin{bmatrix} \mu_t \\ A_t \mu_t + B_t u_t \end{bmatrix}, \begin{bmatrix} \Sigma_t & \Sigma_t A_t^T \\ A_t \Sigma_t & A_t \Sigma_t A_t^T + Q_t \end{bmatrix}\right) \tag{40}$$

$$x_{t+1} \sim N(A_t \mu_t + B_t u_t, A_t \Sigma_t A_t^T + Q_t) \tag{41}$$

$$x_{t+1} \sim N(\mu_{t+1}, \Sigma_{t+1}) \tag{42}$$

Con el tiempo, el modelo de movimiento (Aguilar, Manosalvas, Guillén, & Collaguazo, 2018; Orbea, y otros, 2017) va a incrementar la incertidumbre del estado estimado indefinidamente. Para reducir la incertidumbre, se necesitan mediciones directas del estado actual. Esta medición, conocida como ***z***, puede ser usada para actualizar la función de densidad de probabilidad de ***x***, asumiendo que sabemos la relación lineal de los valores verdaderos de $z = Cx + v$, donde $v \sim N(0, R)$ es el ruido en la medición. Usamos nuevamente intersección para obtener $p(x_t, z_t)$, y usamos la operación de condicionamiento para encontrar los valores actualizados de la media y la covarianza de x_t.

$$p(x_t, z_t) = N(\begin{bmatrix} \mu_t \\ C_t\mu_t \end{bmatrix}, \begin{bmatrix} \Sigma_t & \Sigma_t C_t^T \\ C_t\Sigma_t & C_t\Sigma_t C_t^T + R_t \end{bmatrix}) \tag{43}$$

$$x_t|_{Z_t = Z_0} \sim N(\mu_t + K_t(Z_0 - C_t\mu_t), \Sigma_t - K_t C\Sigma_t) \tag{44}$$

$$K_t = \Sigma_t C_t^T (C_t\Sigma_t C_t^T + R_t)^{-1} \tag{45}$$

Una suposición importante in la estimación de pose es que cada estado debe ser modelado como si fuera dependiente solamente en el estado previo y la entrada de control, ***u***. Similarmente, observaciones o mediciones son dependientes solamente del estado actual. Si el proceso cumple esta suposición de independencia de estados anteriores, entonces se dice que el proceso cumple con la Propiedad de Markov, la cual es una regla fundamental en los algoritmos de filtrado. La siguiente figura (Figura 13) muestra una ilustración de la propiedad de Markov (Thrun, Burgard, & Fox, 2010).

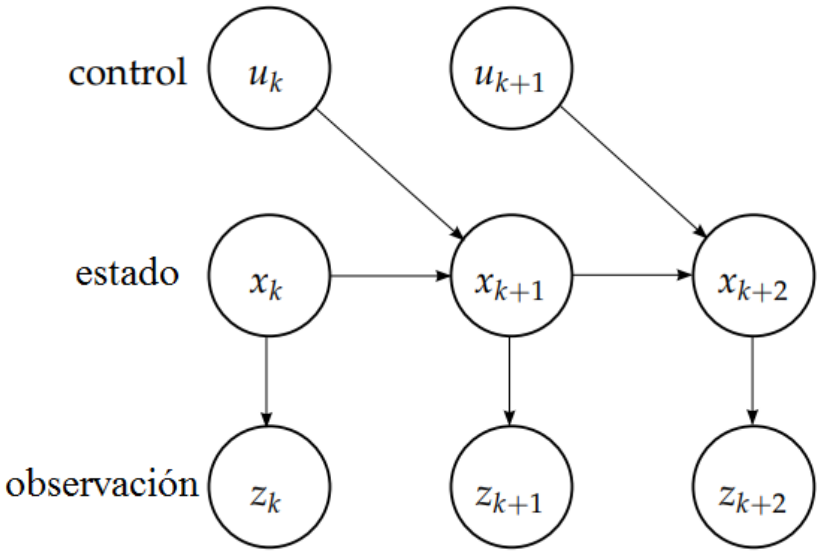

Figura 13. La propiedad de Markov.

2.3.1.1 Resumen del Filtro de Kalman tradicional

Para mantener consistencia en el resto de esta tesis, usamos una notación ligeramente diferente. En lugar de usar μ para la media de la función de densidad de probabilidad del estado actual, usamos el estimado $\hat{x}$. Usamos también la notación $\hat{x}_t^-$, que se denota el estado estimado en el tiempo ***t*** dado un conocimiento previo del proceso anterior a ***t***, y $\hat{x}_t$ es el estimado en el tiempo ***t*** dada la medición z_t.

Estimación inicial

$$x_0 \sim N(\hat{x}_0, \Sigma_0)$$

Predicción

Dado: $x_{t+1} = A_t x_t + B_t u_t + w_t$ donde $w_t \sim N(0, Q)$

$$\hat{x}_t^- \leftarrow A_t \hat{x}_t + B_t u_t$$

$$\Sigma_t^- \leftarrow A_t \Sigma_t A_t^T + Q_t$$

Actualización

Dado: $z_t = C_t x_t + v_t$ donde $v_t \sim N(0, R_t)$

$$\hat{x}_t \leftarrow \hat{x}_t^- + K_t(z_t - C_t \hat{x}_t^-)$$

$$\Sigma_t \leftarrow \Sigma_t^- - K_t C_t \Sigma_t^-$$

$$K_t \leftarrow \Sigma_t^- C_t^T (C_t \Sigma_t^- C_t^T + R_t)^{-1}$$

2.3.2 Filtro de Kalman Extendido

La finalidad de esta tesis es la estimación de la pose de un dispositivo móvil con todo el rango de movimientos posibles. Esto significa tener 3 grados de libertad en posición y 3 en orientación. Estimar esta pose sería imposible con el filtro de Kalman tradicional ya que tiene una dinámica no lineal. Para solventar ese inconveniente, se utiliza una variante de este filtro, llamado el Filtro de Kalman Extendido (EKF) (Shelley, 2014). Las dos principales diferencias están en las ecuaciones del modelo dinámico de movimiento y el modelo del sensor. Estas ecuaciones son las siguientes:

$$x_{t+1} = f(x_t, u_t, w_t) \qquad w_t \sim N(0, Q_t) \tag{46}$$

$$z_t = h(x_t, v_t) \qquad v_t \sim N(0, R_t) \tag{47}$$

Las funciones $\boldsymbol{f}$ y $\boldsymbol{h}$ son funciones no lineales. Debido a sus propiedades no lineales, no es posible encontrar la intersección de x_{t+1} y x_t como en el filtro de Kalman convencional. El EKF soluciona este problema linealizando las funciones $\boldsymbol{f}$ y $\boldsymbol{h}$ cerca del estimado actual $\hat{x}_t$. Una de las formas de lograr linealizar estas funciones es el uso de la Expansión de la Serie de Taylor, la cual afirma que cualquier función $f(x)$ puede ser aproximada a x con la siguiente serie:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)(x-a)^2}{2!} + \frac{f'''(a)(x-a)^3}{3!} + \cdots \tag{48}$$

Los términos en esta serie decrecen en tamaño e importancia mientras la serie se expande, por lo que los términos de alto orden pueden ser ignorados por motivos de aproximación. Normalmente se usan los términos de hasta primer orden solamente, convirtiendo al EKF un estimador de primer orden.

2.3.2.1 Etapa de predicción

Considerando la función $f(x_t, u_t, w_t)$, de la expansión de la Serie de Taylor se obtiene:

$$f(x_t, u_t, w_t) \approx f(\hat{x}_t, u_t) + f'(\hat{x}_t, u_t)\,(x_t - \hat{x}_t) + \cdots \tag{49}$$

$$\approx f(\hat{x}_t, u_t) + F(\hat{x}_t, u_t)\,\tilde{x}_t \tag{50}$$

Donde $\tilde{x}$ es el estado error, la diferencia entre el estado verdadero y el estado estimado, y $\boldsymbol{F}$ es el Jacobiano de $\boldsymbol{f}$. Si $f(x) = (f_1(x), f_2(x), \ldots, f_n(x))^T$ y $x = (x_1, x_2, \ldots, x_n)^T$ entonces:

$$F = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \tag{51}$$

Después de aplicar el Jacobiano, es posible propagar el estado estimado para la etapa de predicción del EKF. Observando las ecuaciones (46) y (50), podemos obtener la estimación del siguiente estado.

$$\hat{x}_{t+1} = f(\hat{x}_t, u_t) \tag{52}$$

El error propagado puede ser representado como una combinación lineal del error anterior y el ruido. Para facilitar la lectura de las ecuaciones, reemplazamos a $F(\hat{x}_t, u_t)$ por F_t.

$$\tilde{x}_{t+1} = F_t \tilde{x}_t + G_t w_t \tag{53}$$

$$\text{con} \quad \tilde{x}_{t+1} = x_{t+1} - \hat{x}_{t+1} \tag{54}$$

La matriz $\boldsymbol{G}$ es el Jacobiano de $\boldsymbol{f}$ con respecto al ruido. Sirve para transformar el vector de ruido a la misma dimensión que el estado error. Usando la definición de covarianza:

$$Cov(x, y) = E[(x - E[x])(y - E(y))] \tag{55}$$

Ahora se puede encontrar la matriz de covarianza propagada en t+1.

$$\Sigma_{t+1} = E[(x_{t+1} - \hat{x}_{t+1})(x_{t+1} - \hat{x}_{t+1})^T] \tag{56}$$

$$= E[\tilde{x}_{t+1}\tilde{x}_{t+1}^T] \tag{57}$$

$$= E[F_t\tilde{x}_t\tilde{x}_t^T F_t^T] + E[G_t w_t w_t^T G_t^T] \tag{58}$$

$$= F_t E[\tilde{x}_t\tilde{x}_t^T]F_t^T + G_t E[w_t w_t^T]G_t^T \tag{59}$$

$$= F_t\Sigma_t F_t^T + G_t Q_t G_t^T \tag{60}$$

2.3.2.2 Filtrado discreto

Todo lo que ha discutido en esta tesis sobre el filtro de Kalman ha sido ha sido basado en filtrado en tiempo continuo, pero las aplicaciones de la vida real utilizan instrumentos que entregan información no lineal y discretizada. Existen algunos métodos para la discretización, y se usan de acuerdo a las especificaciones de la aplicación. El proceso de discretización del estado propagado no debería tener ningún efecto sobre la incertidumbre. Por esto es siempre mejor encontrar una solución analítica, y si eso no es posible la integración numérica es una alternativa. Cualquiera sea el caso, el objetivo es encontrar la matriz Φ que satisfaga a:

$$\tilde{x}_{t+1} = \Phi_t\check{x}_t + wd_t \tag{61}$$

Con ruido discreto $wd_t \sim N(0, Q_t)$. Esto se puede lograr resolviendo la ecuación diferencial $\Phi(t, t_i) = F(t)\Phi(t, t_i)$ con $t \in [t, t + 1]$ con condiciones iniciales $\Phi_t = I$.

La covarianza del ruido también necesita ser discretizada. Esto se lo puede lograr aplicando la siguiente ecuación:

$$Q_d = \int_t^{t+1} \Phi(t + 1, \tau)G(\tau)Q_c G(\tau)^T \Phi(t + 1, \tau)^T d\tau \tag{62}$$

Donde Q_c es la matriz de covarianza del vector de ruido en tiempo continuo (Shelley, 2014).

2.3.2.3 Etapa de actualización

Dada una medición de un sensor $z_t = h(x_t) + v_t$, se puede usar la expansión de la Serie de Taylor para encontrar $h(x_t)$ con el estado estimado actual.

$$h(x_t) \approx h(\hat{x}_t) + H(\hat{x}_t)\,(x_t - \hat{x}_t) + \cdots \tag{63}$$

$$\approx h(\hat{x}_t) + H(\hat{x}_t)\,\tilde{x}_t \tag{64}$$

Donde $\boldsymbol{H}$ es el Jacobiano de $\boldsymbol{h}$ con respecto a $\boldsymbol{x}$. El Jacobiano $\boldsymbol{H}$ puede entonces ser usado en las ecuaciones de actualización del EKF. De la ecuación (64) obtenemos:

$$h(x_t) - h(\hat{x}_t) \approx H_t\tilde{x}_t \tag{65}$$

Y reemplazando $z_t = h(x_t) + v_t$ en (65) se tiene:

$$z_t - h(\hat{x}_t) \approx H_t\tilde{x}_t - v_t \tag{66}$$

$$r_t = H_t\tilde{x}_t + n_t \tag{67}$$

Donde $\boldsymbol{r}$, conocido como vector residual, es la diferencia entre la medida y la hipótesis, y $\boldsymbol{n}$ es el ruido. Esta ecuación es importante, ya que nos indica que el vector residual debe ser una combinación lineal del estado error y el ruido. De la misma manera que en el filtro de Kalman tradicional, procedemos con la modificación de la etapa de actualización:

$$\hat{x}_t \leftarrow \hat{x}_t + K_t(z_t - h(\hat{x}_t)) \tag{68}$$

$$\Sigma_t \leftarrow \Sigma_t - K_t H_t \Sigma_t \tag{69}$$

$$K_t \leftarrow \Sigma_t H_t^T (H_t \Sigma_t H_t^T + R_t)^{-1} \tag{70}$$

El proceso de derivación de las ecuaciones de la etapa de actualización del EKF es similar a la de la etapa de predicción, pero al tener un nivel de complejidad bastante elevado y no ser relevante para esta tesis, se lo ha omitido completamente (Thrun, Burgard, & Fox, 2010).

2.3.2.4 Resumen del Filtro de Kalman Extendido

Las ecuaciones finales del filtro de Kalman Extendido son muy similares a las del filtro de Kalman tradicional. Si las funciones $\boldsymbol{f}$ y $\boldsymbol{h}$ son lineales, el EKF se convierte en un filtro de Kalman tradicional. Las notaciones usadas son las mismas que el filtro de Kalman tradicional.

Estimación inicial

$$x_0 \sim N(\hat{x}_0, \Sigma_0)$$

Predicción

Dado: $x_{t+1} = f(x_t, u_t) + w_t$ donde $w_t \sim N(0, Q_t)$

$$\hat{x}_t^- \leftarrow f(\hat{x}_t, u_t)$$

$$F_t = \frac{\partial f}{\partial x}(x_t, u_t)$$

$$\Sigma_t^- \leftarrow F_t \Sigma_t F_t^T + Q_t$$

Actualización

Dado: $z_t = h(x_t) + v_t$ donde $v_t \sim N(0, R_t)$

$$\hat{x}_t \leftarrow \hat{x}_t^- + K_t(z_t - h(\hat{x}_t^-))$$

$$H_t = \frac{\partial h}{\partial x}(x_t, u_t)$$

$$\Sigma_t \leftarrow \Sigma_t^- - K_t H_t \Sigma_t^-$$

$$K_t \leftarrow \Sigma_t^- H_t^T (H_t \Sigma_t^- H_t^T + R_t)^{-1}$$

2.4 Cuaterniones

Un cuaternión se define de la siguiente manera (Solá, 2017):

$$q = iq_1 + jq_2 + kq_3 + q_4 \tag{71}$$

Los números i, j y k son conocidos como números hipercomplejos.

$$i^2 = -1 \qquad j^2 = -1 \qquad k^2 = -1 \tag{72}$$

$$-ij = ji = k \qquad -ik = kj = i \qquad -ki = ik = j \tag{73}$$

Esta convención es diferente a la convención de cuaterniones original de Hamilton. Esta convención facilita la multiplicación entre cuaterniones.

$$\overset{I}{G}\bar{q} = \overset{I}{C}\bar{q} \otimes \overset{C}{G}\bar{q} \tag{74}$$

El operador $\otimes$ representa la multiplicación de cuaterniones.

$$\bar{q} \otimes \bar{p} = (q_4 + iq_1 + jq_2 + kq_3)(p_4 + ip_1 + jp_2 + kp_3) \tag{75}$$

$$\bar{q} \otimes \bar{p} = \begin{bmatrix} q_4p_1 + q_3p_2 - q_2p_3 + q_1p_4 \\ -q_3p_1 + q_4p_2 + q_1p_3 + q_2p_4 \\ q_2p_1 - q_1p_2 + q_4p_3 + q_3p_4 \\ -q_1p_1 - q_2p_2 - q_3p_3 + q_4p_4 \end{bmatrix} \tag{76}$$

Para transformar el cuaternión unitario en una matriz de rotación, es necesario utilizar la siguiente fórmula:

$$\overset{I}{C}C(\bar{q}) = \begin{bmatrix} -q_1 - q_2 - q_3 + q_4 & 2(q_1q_2 + q_3q_4) & 2(q_1q_3 + q_2q_4) \\ 2(q_1q_2 - q_3q_4) & -{q_1}^2 + {q_2}^2 - {q_3}^2 + {q_4}^2 & 2(q_2q_3 + q_1q_4) \\ 2(q_1q_3 + q_2q_4) & 2(q_2q_3 - q_1q_4) & -{q_1}^2 - {q_2}^2 + {q_3}^2 + {q_4}^2 \end{bmatrix} \tag{77}$$

CAPITULO III

3. SISTEMA DE MEDICIÓN

3.1 Modelamiento de la IMU

Para utilizar correctamente los datos de la IMU en el algoritmo de odometría visual inercial, es necesario calcular los parámetros para modelar el ruido del acelerómetro y del giroscopio (los valores de ruido y de bias) y la transformación que existe entre la cámara y la IMU (rotación y traslación de la IMU con respecto a la cámara). Para esta tesis, todos estos parámetros fueron calculados mediante la herramienta de código abierto Kalibr (Figura 14).

Figura 14. Herramienta Open Source Kalibr.
Fuente: (Kalibr the calibration toolbox, 2018)

Kalibr es un conjunto de librerías y herramientas para la calibración de sistemas de odometría y SLAM que hacen uso de sensores visuales e inerciales. Para usar Kalibr, es necesario que el computador disponga de ROS Indigo y de MATLAB, y que estos programas estén configurados correctamente.

Kalibr utiliza ROSbags (herramienta de ROS para grabar datos de mensajes de ROS, para posteriormente ser reproducidos) para grabar los datos de la IMU y de la cámara, y ser usados

posteriormente en la calibración. Mediante la herramienta bagcreater de Kalibr se puede generar un ROSbag a partir de las imágenes grabadas y los datos de la IMU. La Fig. 15 muestra la organización de los archivos de la IMU y la cámara para el uso de la herramienta bagcreater. Los nombres de las imágenes son las marcas de tiempo (timestamp en inglés) de las mismas, expresadas en formato de tiempo UNIX.

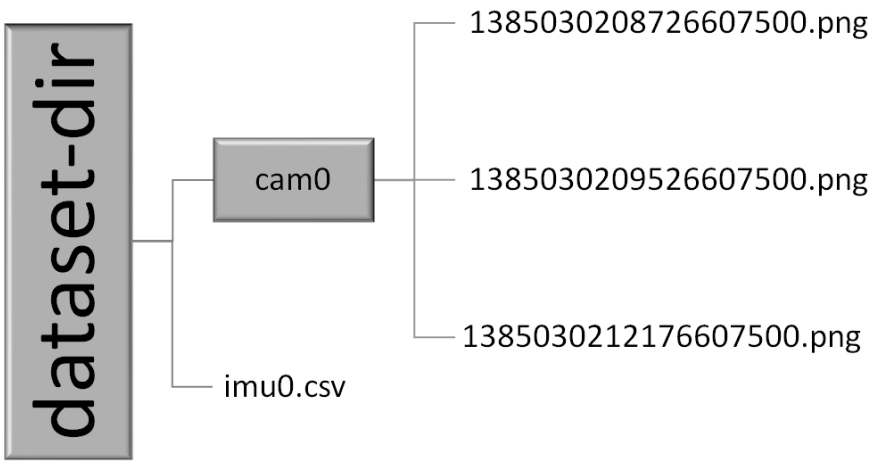

Figura 15. Organización de los archivos para el uso de la herramienta bagcreater.

El archivo imu0.csv usa el formato de la Fig. 16:

```
timestamp,omega_x,omega_y,omega_z,alpha_x,alpha_y,alpha_z
1385030208736607488,0.5,-0.2,-0.1,8.1,-1.9,-3.3
 ...
1386030208736607488,0.5,-0.1,-0.1,8.1,-1.9,-3.3
```

Figura 16. Formato del archivo de la IMU usado por bagcreater.

Donde timestamp representa las marcas de tiempo en que se produjeron los datos, omega representa los valores del giroscopio y alpha los valores del acelerómetro. Al igual que las imágenes, las marcas de tiempo de la IMU son expresadas en formato de tiempo UNIX. Las

unidades de las marcas de tiempo son nanosegundos, los datos del giroscopio están expresados en rad/s y los valores del acelerómetro m/s^2.

Para la creación del ROSbag es necesario ejecutar el comando kalibr_bagcreater --folder dataset-dir --output-bag ejemplo.bag.

3.1.1 Calibración de la IMU

Existen varios métodos para la determinación de los parámetros de ruido de la IMU, pero el más común y más estandarizado es la obtención de la gráfica de desviación estándar de Allan (AD por sus siglas en inglés). Esta grafica es muy útil para la determinación de diferentes procesos randómicos, incluyendo el ruido blanco de la IMU (pendiente de -1/2 en la gráfica AD) y el proceso de camino aleatorio para el bias de la IMU (pendiente de ½ en la gráfica AD). Los valores del ruido blanco del acelerómetro y del giroscopio son representados en Kalibr con σ_a y σ_g respectivamente. Así mismo, los valores de bias son representados mediante σ_{ba} y σ_{bg}.

Los pasos que se deben seguir para obtener los parámetros de ruido de la IMU con Kalibr son los siguientes:

1. Grabar los datos de la IMU en un ROSbag. El teléfono debe estar inmóvil al momento de grabar los datos. Para esta tesis, un ROSbag de aproximadamente 4 horas fue grabado.
2. Convertir el ROSbag en un archivo .mat para ser utilizado en MATLAB. Esto se puede realizar mediante la ejecución del comando rosrun bagconvert bagconvert imu0.bag /imu0, donde imu0.bag es el ROSbag creado en el paso anterior y /imu0 es el tópico en donde se están publicando los datos de la IMU en el ROSbag.
3. Ejecutar uno de los scripts de MATLAB proporcionados por Kalibr. En esta tesis el script seleccionado fue el SCRIPT_allan_matparallel.m, el cual permite estimar la gráfica AD

mediante el uso toolbox de paralelización de MATLAB, reduciendo el tiempo de cálculo del proceso.

Una vez completados los pasos, MATLAB entregara dos graficas AD, una para el acelerómetro y otra para el giroscopio, ambas con los valores de ruido blanco y bias ya calculados. Los resultados obtenidos para esta tesis pueden observarse en la sección de pruebas y resultados.

3.1.2 Calibración IMU-Cámara

Las herramientas para la calibración cámara-IMU permiten obtener los parámetros espaciales y temporales de una cámara con respecto a una IMU intrínsecamente calibrada. Mientras la frecuencia en la que se tomaron los datos sea mayor, los resultados de la calibración van a ser más precisos.

Para el uso de la herramienta para la calibración del sistema cámara-IMU se debe disponer previamente de los siguientes archivos:

- nombre.bag: Este archivo es el ROSbag que contiene los datos de la cámara y de la IMU.
- camchain.yaml: Este archivo contiene los parámetros de calibración de la cámara. El formato que debe tener este archivo es el de la Fig. 17:

```
cam0:
  camera_model: pinhole
  intrinsics: [2199.02681, 2131.74580, 1017.76745, 1486.49311]
  distortion_model: radtan
  distortion_coeffs: [0, 0, 0, 0]
  rostopic: /cam0/image_raw
  resolution: [1920, 1080]
```

Figura 17. Formato del archivo camchain.yaml.

- imu.yaml: Este archivo contiene los parámetros de ruido de la IMU. El formato de este archivo es el de la Fig.18:

```
#Accelerometers
accelerometer_noise_density: 4.854e-02
accelerometer_random_walk:   1.063e-04

#Gyroscopes
gyroscope_noise_density:     2.437e-02
gyroscope_random_walk:       7.045e-05

rostopic:                    /imu0
update_rate:                 20.0
```

Figura 18. Formato del archivo imu.yaml.

- target.yaml: Este archivo contiene las características del patrón de calibración, el mismo que puede observarse en la Fig. 20. El formato del archivo es el de la Fig. 19:

```
target_type: 'aprilgrid'
tagCols: 6
tagRows: 6
tagSize: 0.025
tagSpacing: 0.3
```

Figura 19. Formato del archivo target.yaml.

Los pasos para obtener estos parámetros mediante las herramientas proporcionadas por Kalibr son los siguientes:

1. Grabar los datos de la IMU y de la cámara en un ROSbag. Esta vez es necesario excitar todos los ejes de la IMU, tanto en rotación como en traslación. Es necesario evitar golpes y movimientos bruscos al momento de la calibración.
2. En una carpeta, guardar los archivos previamente especificados y configurados correctamente para la aplicación actual.
3. Ejecutar el comando kalibr_calibrate_imu_camera --bag nombre.bag --cam camchain.yaml --imu imu.yaml --target target.yaml.

Una vez completados estos pasos, el programa devolverá varios archivos y reportes con la información necesaria para realizar la calibración del sistema cámara-IMU. Los resultados obtenidos para esta tesis pueden observarse en la sección de pruebas y resultados.

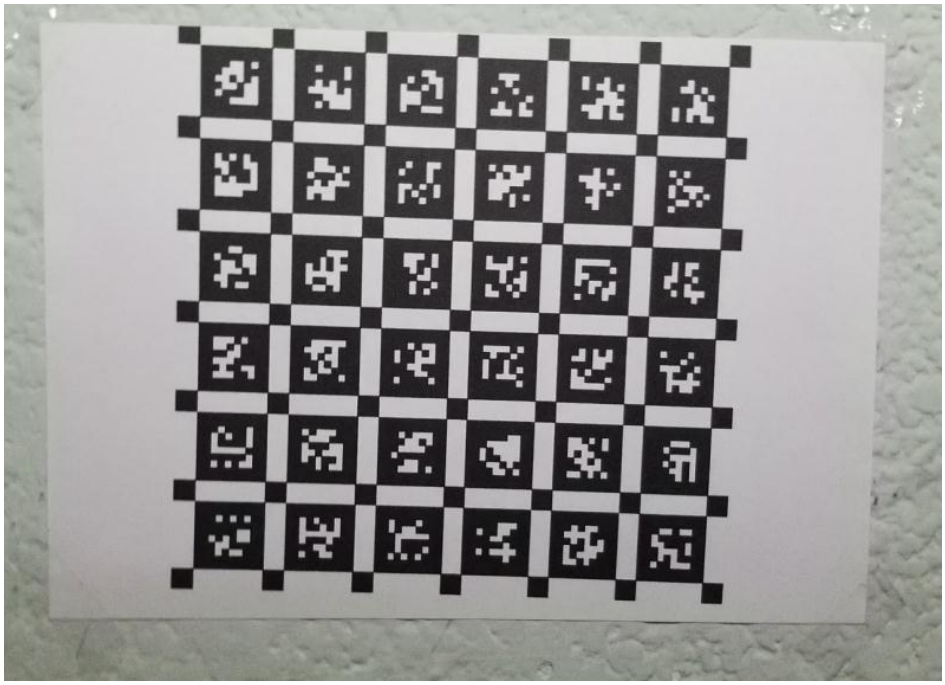

Figura 20. Patrón de calibración del sistema cámara-IMU.
Nota. Este patrón es conocido como Aprilgrid.

3.2 Modelamiento de la cámara

3.2.1 Calibración de la cámara

La calibración de la cámara es necesaria para obtener los parámetros o_x, o_y, f_x y f_y, los cuales son los parámetros intrínsecos de la cámara. Para la obtención de estos parámetros, se implementó un programa en ROS y la función cv2.findChessboardCorners() de OpenCV. Este programa permite calcular los parámetros a partir de varias imágenes tomadas a un patrón de calibración en diferentes ángulos. Este patrón debe ubicarse en una superficie plana y rígida, y no debe ser movido en el proceso de toma de imágenes. En la Fig. 21 se puede apreciar una fotografía de este patrón tomada desde el teléfono.

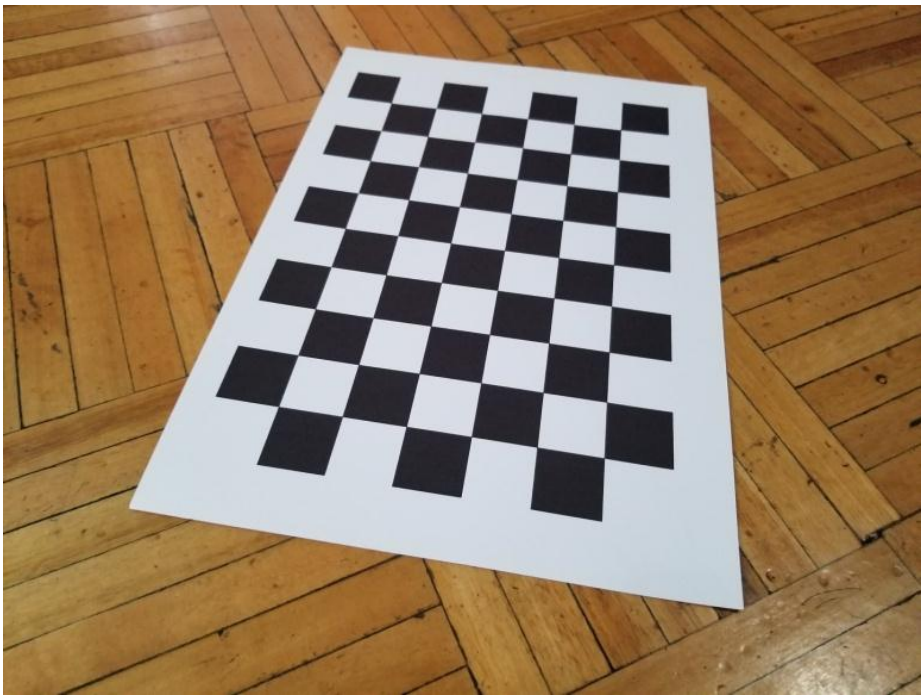

Figura 21. Patrón de prueba para la calibración de los parámetros de la cámara.

Para el funcionamiento correcto del programa, es necesario tener un conjunto de al menos diez imágenes tomadas en diferentes ángulos al patrón de prueba. Estas imágenes deben ser colocadas en una carpeta especificada en el programa. Una vez las imágenes se encuentren en la carpeta correcta, el programa es ejecutado mediante el comando roslaunch cam_matrix run_calib.launch. Dependiendo del número de imágenes y del tamaño y formato de las mismas, el programa puede tardar varios minutos en ejecutarse.

El programa puede también estimar los parámetros de distorsión tangencial y radial de la cámara, pero no es necesario hacerlo ya que estas distorsiones son corregidas previamente por el software del teléfono al momento de tomar una fotografía o video.

3.3 Aplicación en Android

Una de las finalidades de esta tesis es el envío los datos de la cámara y de la IMU de un dispositivo móvil hacia el computador para navegación visual inercial. Para lograrlo, es necesaria

la creación de una aplicación escrita en Java para Android que permita que el dispositivo móvil envíe la información visual obtenida de la cámara y la información inercial obtenida por el sensor IMU. La herramienta utilizada en esta tesis para solventar este problema es ROSJava.

3.3.1 Introducción a ROSJava

ROSJava es una implementación de Java puro en ROS, y uno de sus principales objetivos es el permitir el uso de ROS en Android. Provee un conjunto de librerías que permite a programadores de Java interaccionar con tópicos, servicios y parámetros de ROS. También posee una implementación en Java de roscore, el nodo master de ROS. Parte de ROSJava son las librerías de Android Core, que permiten compilar y generar aplicaciones para Android escritas en Java por medio del terminal de Ubuntu en el computador. En la Fig. 22 se puede observar cómo se realiza la comunicación entre el dispositivo Android y el computador.

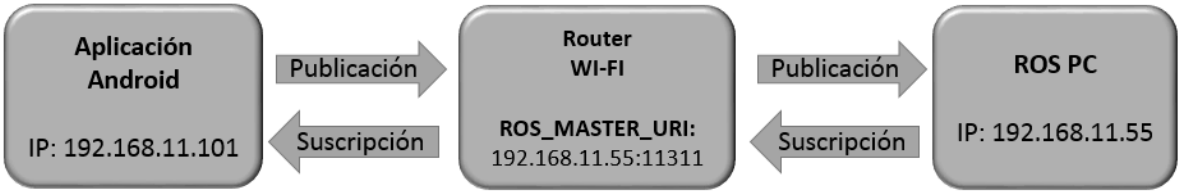

Figura 22. Ilustración de la comunicación entre tópicos de ROS.

Usando las herramientas proporcionadas por ROSJava y Android Core es posible la creación de nodos de ROS desde el dispositivo móvil, sin embargo, esta implementación tiene muchas menos opciones y características que las implementaciones estándar de ROS en C++ o Python.

ROSJava tiene una colección muy amplia de funcionalidades de ROS, pero éstas se encuentran en constante desarrollo. Es por esta razón que la documentación y ejemplos de ROSJava son muy escasos o están en constante cambio.

En las siguientes secciones se explica brevemente cuales son los prerequisitos necesarios y, subsecuentemente, la programación de una aplicación con ROSJava en un computador con Ubuntu 14.04.

3.3.2 Prerrequisitos

Existen varios requerimientos que deben ser cubiertos antes de poder compilar y crear una aplicación en Android con ROSJava. Estas librerías son totalmente gratuitas, y pueden ser utilizadas y modificadas libremente. A continuación, se muestran los requisitos seguidos de una breve descripción de los mismos.

- **ROSJava:** Librerías para la implementación de ROS en Java, optimizadas para ser usadas en Android.
- **Android SDK**: Kit de desarrollo de software (Software Development Kit) para aplicaciones en Android en Ubuntu. Aquí se encuentran todas las herramientas necesarias para crear una aplicación en Java para Android. Se debe elegir la versión de SDK, y esta corresponde a la versión de la aplicación en Android.
- **Android Core**: Paquete para la interfaz entre ROS y Android. Permite usar las herramientas de ROS en la aplicación de Android por medio de las clases de ROSJava y generar el archivo apk.
- **Catkin:** Herramientas y comandos para la manipulación de paquetes de ROS en Ubuntu.

Una vez completados estos requerimientos podemos continuar con el desarrollo de la aplicación con ROSJava.

3.3.3 Estructura de la aplicación

Java es el lenguaje de programación usado en las aplicaciones en Android, aplicando el paradigma de la Programacion Orientada a Objetos (en inglés Object Oriented Programming

(OOP)). Este tipo de programación nos permite acceder a los recursos del dispositivo de manera más ordenada y sencilla que los métodos de programación convencionales.

La aplicación en Android dispone de diferentes tipos de archivos, varios de los cuales tienen que ser correctamente programados y configurados antes de empezar a escribir el programa principal de la aplicación. En la Fig. 23 se muestra la estructura de los archivos de la aplicación.

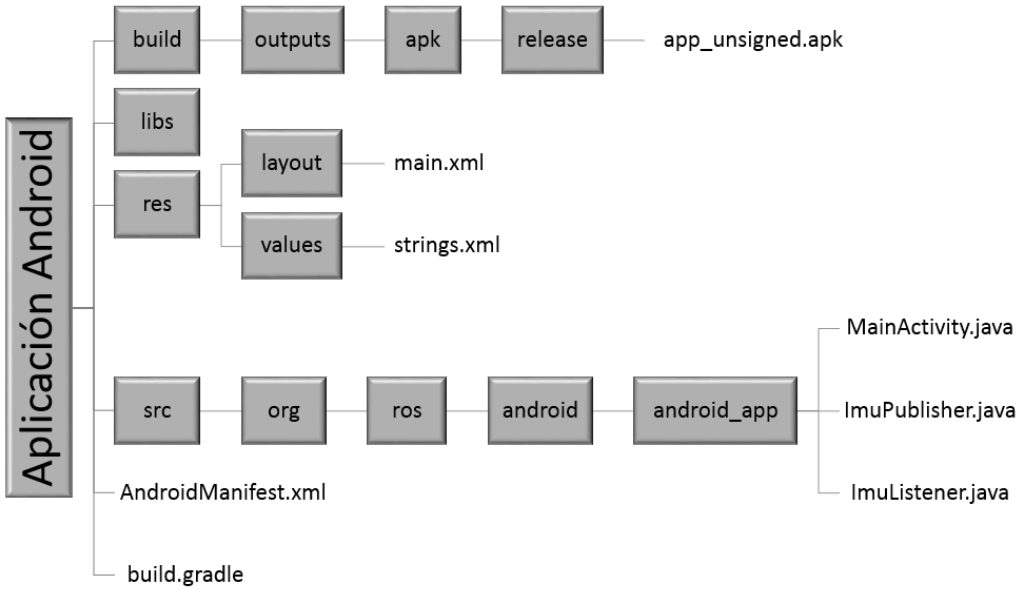

***Figura 23*.** Estructura de los archivos de la aplicación en Android.

3.3.3.1 Archivos de configuración

En los archivos de configuración se definen todas las características necesarias para el correcto funcionamiento y generación de la aplicación. A continuación se muestra una descripción de cada uno de estos archivos, detallando brevemente la función y el lenguaje en el que están escritos los mismos.

- **main.xml:** En este archivo se define el diseño de la interfaz visual de la aplicación. Está escrito en lenguaje XML, muy similar a HTML, que permite configurar fácilmente los

colores, letra y disposición de los elementos en la aplicación. La aplicación desarrollada para esta tesis requiere solamente de una vista mostrando las imágenes transmitidas hacia el computador, haciendo que la programación de este archivo sea muy sencilla. Para el correcto funcionamiento de la aplicación es necesario que la pantalla no se apague durante la ejecución del programa, lo cual puede ser logrardo escribiendo android:keepScreenOn="true" en este archivo.

- **strings.xml:** Este archivo sirve para definir los strings de las frases principales a ser usadas en la aplicación. Son muy útiles si se quiere realizar una aplicación en diferentes idiomas. En la presente tesis es necesario definir solamente el nombre de la aplicación en este archivo. El lenguaje usado es el XML.
- **AndroidManifest.xml:** En este archivo se definen principalmente los permisos que deben ser aceptados por el usuario y el hardware del teléfono a ser usado por la aplicación, además de otras pequeñas configuraciones como la dirección del ícono, la dirección del nombre de la aplicación y la orientación de la pantalla en el teléfono. Aquí también se realizan las configuraciones de algunos de los elementos de ROSJava que van a ser usados en la aplicación. Está escrito en lenguaje XML.
- **build.gradle**: Este archivo especifica la configuración para la generación del archivo de la aplicación. Aquí se definen los repositorios de las librerías, las dependencias a ser usadas por la aplicación, el Id de la aplicación, y las versiones de compilación y del SDK a ser usado. La versión a ser usada por la aplicación de esta tesis es la SDK de la API 27, más conocida como Android 8.1 Oreo. El lenguaje de programación usado en este archivo es Groovy-DSL.

3.3.3.2 Archivos principales de la aplicación

Las aplicaciones en Android son programadas en lenguaje Java, y los recursos del dispositivo pueden ser manipulados mediante las clases propias de Android. Para la aplicación de la presente tesis se crearon dos clases más, una para leer los datos de la IMU y otra para publicar en un nodo de ROS estos datos. Estas clases son llamadas en la clase principal de la aplicación. La aplicación entonces consta de un total de tres archivos programados java, dos de las clases antes mencionadas y uno de la clase principal de la aplicación. A continuación se describe detalladamente cada uno de estos archivos:

ImuListener.java

Este archivo contiene la clase ImuListener. Su función es leer los datos de la IMU. Para lograrlo, es necesario implementar la interfaz de Android SensorEventListener. Esta interfaz es la que permite acceder y manipular los datos de los sensores del teléfono. Todos los métodos de esta interfaz son públicos. Los métodos de la clase ImuListener son los siguientes:

- onSensorChanged(SensorEvent event): Este método público reporta un evento cada vez que hay un cambio en el valor de los sensores a ser leídos. El evento guarda toda la información relevante del sensor en una variable llamada event, y es por medio de los atributos esta variable que podemos acceder a los valores de la IMU y el tiempo entre lecturas del sensor. Este método reporta solamente un evento a la vez, por lo cual es necesario condicionar el tipo de sensor que reportó el evento, lo cual puede lograrse al igualar event.sensor.getType() al sensor del cual deseamos leer los valores.
- publishImu(Publisher<sensor_msgs.Imu> publisher): Una vez que todos los valores de la IMU han sido correctamente leídos, este método guarda estos valores en una variable con un

formato legible para ROS. Una vez publicada esta variable, el método espera a que una nueva lectura de la IMU suceda.

- onResume(): Este método se ejecuta al iniciar o al continuar la aplicación después de una pausa. Aquí es donde se registran los sensores a ser utilizados y la frecuencia a la que van a reportar los valores medidos.
- onPause(): Este método se ejecuta al pausarse la aplicación. Es necesario cancelar el registro de datos de los sensores para que no sean utilizados mientras la aplicación esté en pausa.

ImuPublisher.java

Este archivo contiene la clase ImuPublisher, y es la encargada de publicar los datos de la IMU en un tópico de ROS conectado al nodo master del computador. Esta vez es necesario implementar la interfaz de ROSJava NodeMain. Esta interfaz permite la creación y manipulación de nodos y tópicos en ROS desde el teléfono. Todos los métodos de esta clase son públicos. Los métodos de la clase ImuPublisher son los siguientes:

- onStart(ConnectedNode connectedNode): Este método es ejecutado una vez se inicia la aplicación. Es aquí donde los datos obtenidos mediante un llamado al método publishImu de la clase ImuListener son publicados, usando un bucle cancelable, en un tópico de ROS.
- GraphName getDefaultNodeName(): En este método se define el nombre del nodo visto desde ROS.
- onError(Node node, Throwable throwable): Este método se ejecuta cuando existe un error en la ejecución de la aplicación.
- onShutdown(Node node): Este método es llamado una vez se finalice la aplicación.

MainActivity.java

Este archivo contiene la clase principal del programa, donde se hace un llamado a los métodos de las clases ImuListener e ImuPublisher para publicar en un tópico los valores medidos por la IMU, además de adquirir y publicar imágenes obtenidas de la cámara del teléfono. Para poder utilizar las herramientas de ROSJava, es necesario que la clase principal sea una subclase de la clase RosActivity. Los métodos de la clase MainActivity son los siguientes:

- onCreate(Bundle savedInstanceState): Este método es ejecutado una vez que se inicializa la aplicación. Aquí se vincula el archivo de la interfaz (main.xml) con la aplicación principal. En esta aplicación se muestran las imágenes transmitidas en la pantalla del teléfono.
- init(NodeMainExecutor nodeMainExecutor): Una vez que el teléfono se haya conectado al nodo master de ROS en el computador, este método se ejecuta. Este método tiene varias funciones necesarias para el correcto envido de los datos de la cámara y de la IMU. Aquí se realiza la solicitud de permisos al usuario (desde Android 6.0 los permisos son solicitados mientras la aplicación se ejecuta), apertura y selección de la cámara a ser usada, apagar el balance de blancos automático y la exposición automática de la cámara. Una vez realizadas estas funciones, se configuran los nodos para el envio de los datos de la IMU y las imágenes de la cámara.
- onRequestPermissionsResult(): Este método es el responsable de pedir los permisos de uso de recursos del teléfono al usuario. Se pueden configurar mensajes para informar al usuario de los permisos que este tiene.

3.3.4 Generación del archivo APK

La compilación y generación del archivo apk se realizó mediante el paquete android_core de ROS, utilizando las herramientas y comandos de Catkin. Los archivos de la aplicación deben estar dentro de la carpeta ~/android_core/src/android_core/. Para compilar la aplicación es

necesario dirigirse, por medio de la consola, a la carpeta principal de android_core. Una vez dentro, se puede ejecutar la compilación escribiendo en la consola el comando catkin_make.

Si no existen errores en la programación o en la configuración de la aplicación, en la consola se podrá observar algo similar a la Fig. 24.

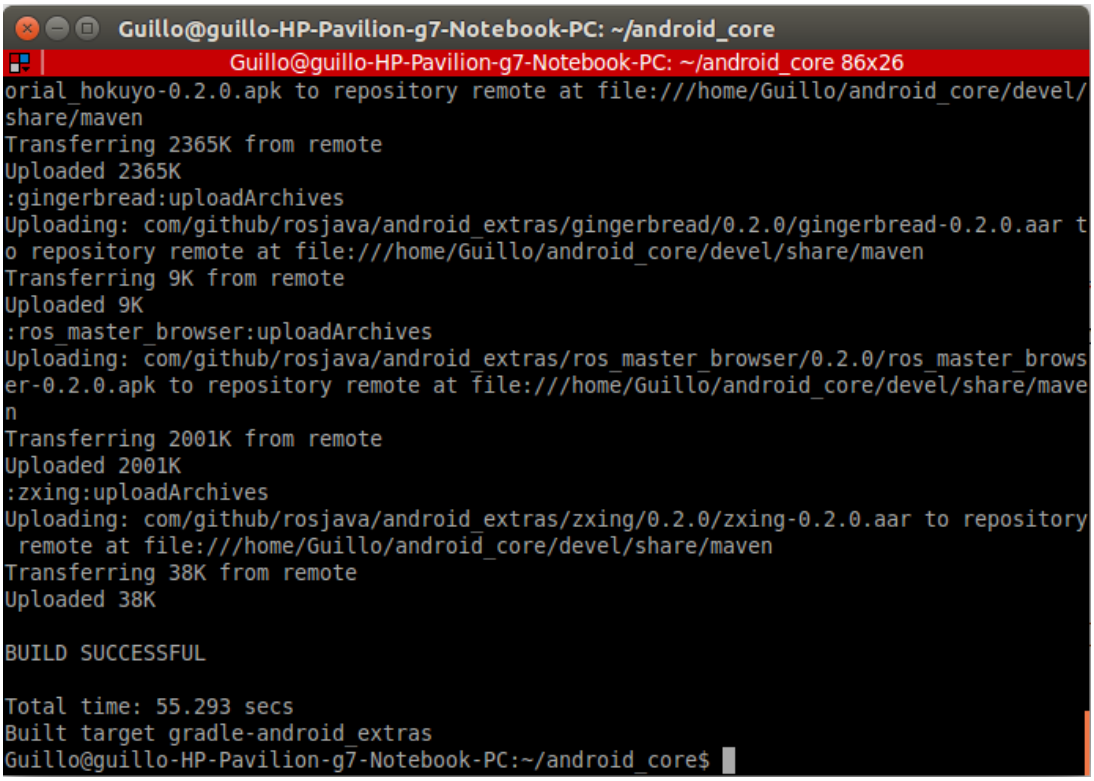

Figura 24. Compilación de la aplicación en consola.

Una vez que la aplicación haya sido compilada con éxito, podemos hacer uso del archivo apk generado. Este archivo se lo puede encontrar dentro de la carpeta del proyecto. Para la aplicación Android de esta tesis, el archivo apk puede ser encontrado en el directorio: ~/android_core/src/android_core/android_cicte_vio/builds/outputs/apk/release/android_cicte_vio-release-unsigned.apk.

Esta aplicación aún no está lista para ser usada. Android requiere que todas sus aplicaciones sean firmadas digitalmente con un certificado y metadatos (como nombre, localización, fecha de creación, etc.). Esta firma sirve como una huella digital que asocia la aplicación con los datos privados del autor. Mediante este proceso, Android se asegura que futuras actualizaciones a la aplicación son auténticas y vienen del autor original. Todas las aplicaciones deben usar esta firma digital para poder ser instaladas y actualizadas por cualquier usuario.

Existen varios métodos para firmar digitalmente una aplicación en Android. La aplicación de esta tesis fue firmada digitalmente mediante la aplicación para Android APK Editor, gratuitamente disponible en PlayStore de Android. En la Fig. 25 se pueden apreciar los pasos necesarios para firmar digitalmente una aplicación mediante APK Editor.

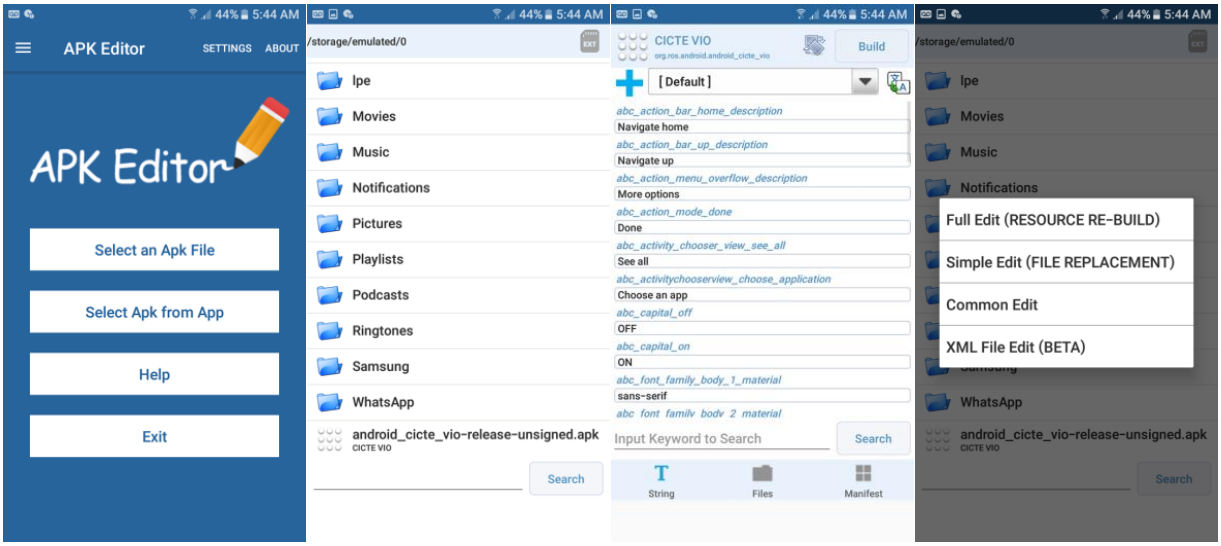

***Figura 25*.** Pasos para el firmado digital de una aplicación usando APK Editor.

3.3.5 Funcionamiento de la aplicación

Antes de ejecutar la aplicación es necesario conectar al teléfono a una red LAN en la que se encuentre el computador que va a recibir los datos de la IMU y de la cámara. El nodo master de ROS debe estar corriendo en el computador, ya que la aplicación debe conectarse a este nodo

para publicar los tópicos en la red. Escribiendo el comando roscore en la consola de Ubuntu se puede ejecutar el nodo master de ROS. Si no hay problemas con la instalación de ROS, algo similar a la Fig. 26 aparecerá en la consola.

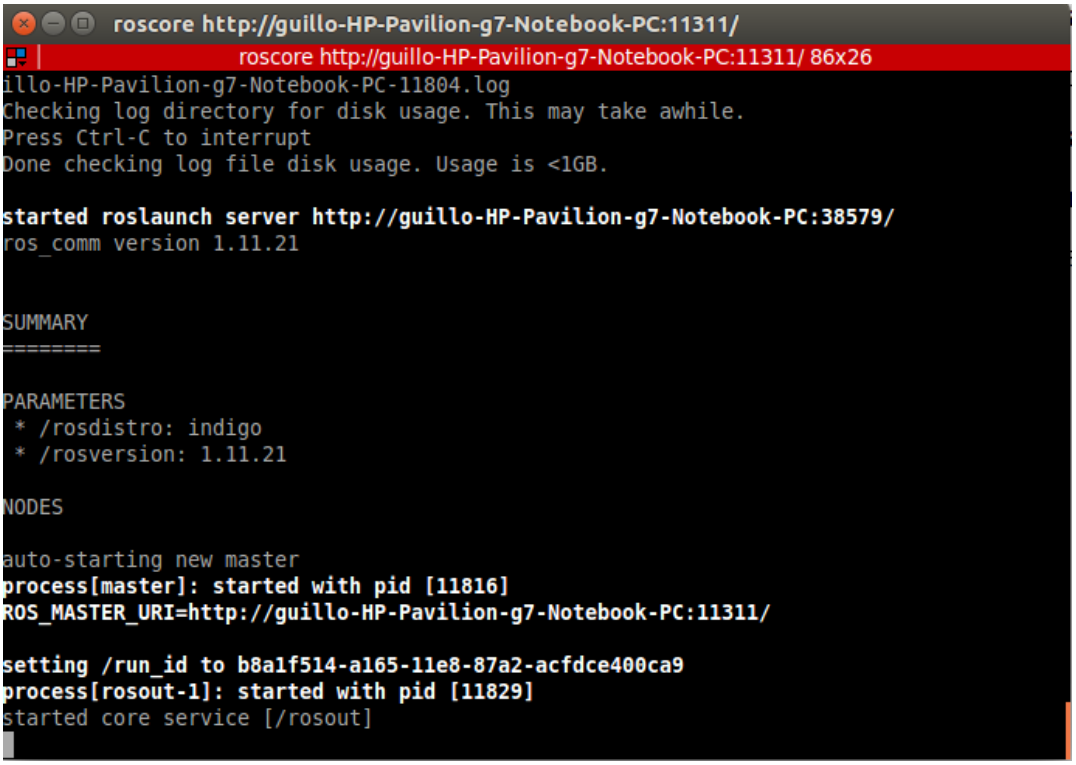

Figura 26. Ejecución del nodo master de ROS en el computador.

Una vez ejecutado el nodo master en el computador, procedemos a lanzar la aplicación en el teléfono. La Fig. 27 muestra la pantalla que se muestra al iniciar la aplicación.

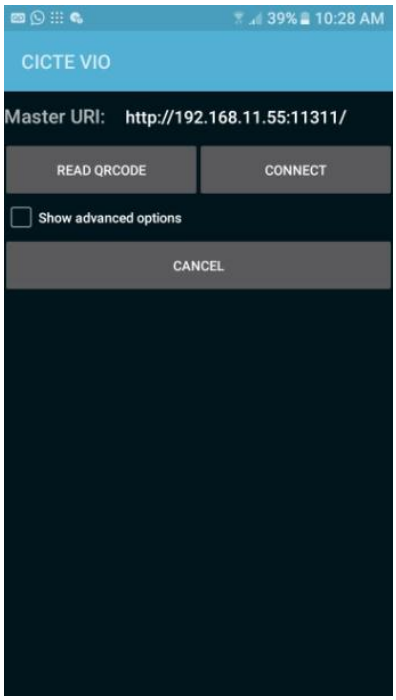

***Figura 27*.** Pantalla principal de la aplicación.

Para empezar a publicar los tópicos de la IMU y de la cámara en el computador, es necesario ingresa el IP del nodo master seguido del puerto que va a ser usado para la conexión (el puerto por defecto es el 11311) en el campo Master URI, y después presionar el botón CONECT. Un mensaje aparecerá en la pantalla, en el cual se le pedirá al usuario que acepte los permisos necesarios para el uso de la cámara y otros recursos del teléfono. Después de aceptar el mensaje, la aplicación empezará a publicar los tópicos de la cámara y de la IMU. Los tópicos que están publicados en ROS pueden ser observados escribiendo el comando rostopic list en la consola de Ubuntu.

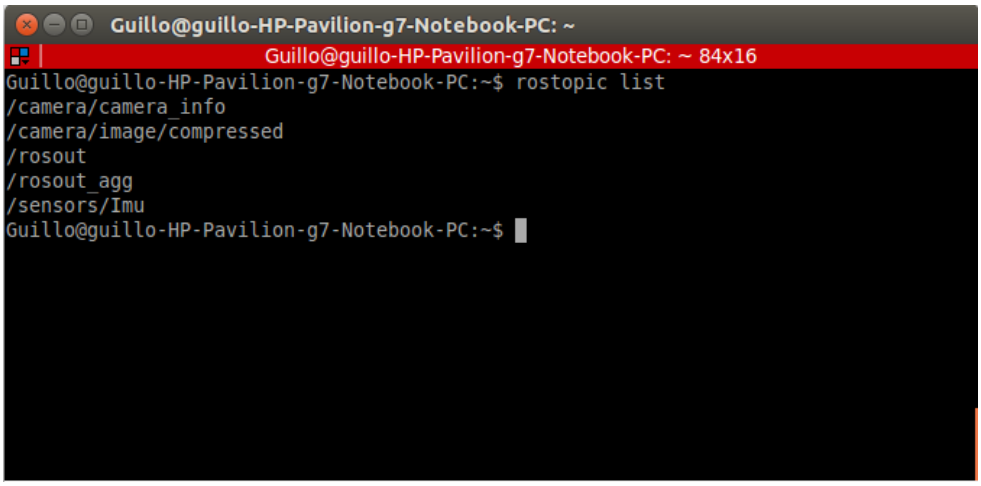

Figura 28. Ejecución del comando rostopic list.

Como se puede observar en la Fig. 28, existen varios tópicos publicados. Los tópicos publicados por la aplicación son /sensors/Imu para los datos de la IMU y /camera/image/compressed para los datos de la cámara.

Mediante el comando rostopic echo /sensors/Imu podemos observar los datos de la IMU del teléfono trasmitidos en tiempo real. En la Fig. 29 se puede apreciar la ejecución de este comando en la consola.

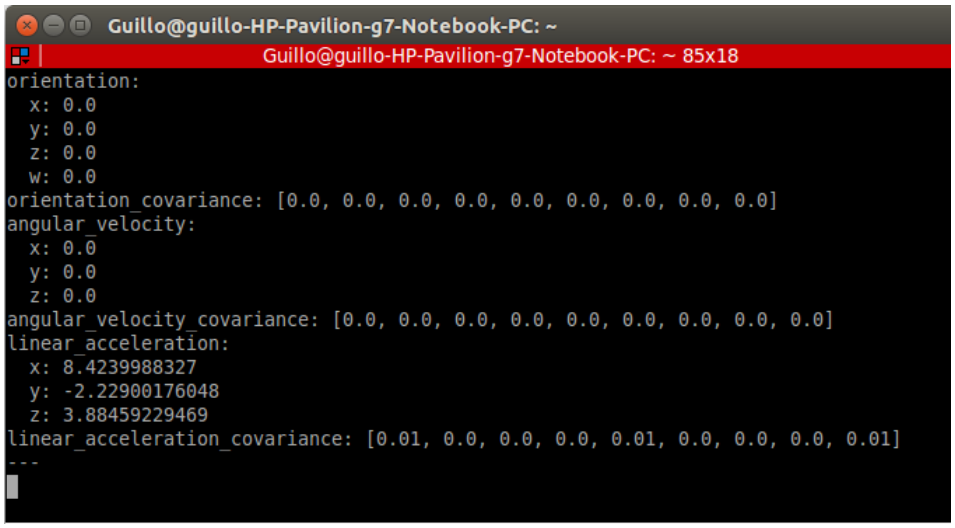

Figura 29. Ejecución del comando rostopic echo /sensors/Imu.

Para observar las imágenes que son publicadas en tiempo real por la cámara del teléfono, es necesario utilizar el comando rosrun image_view image_view image:=/camera/image _image_transport:=compressed. El funcionamiento de este comando puede observarse en la Fig. 30.

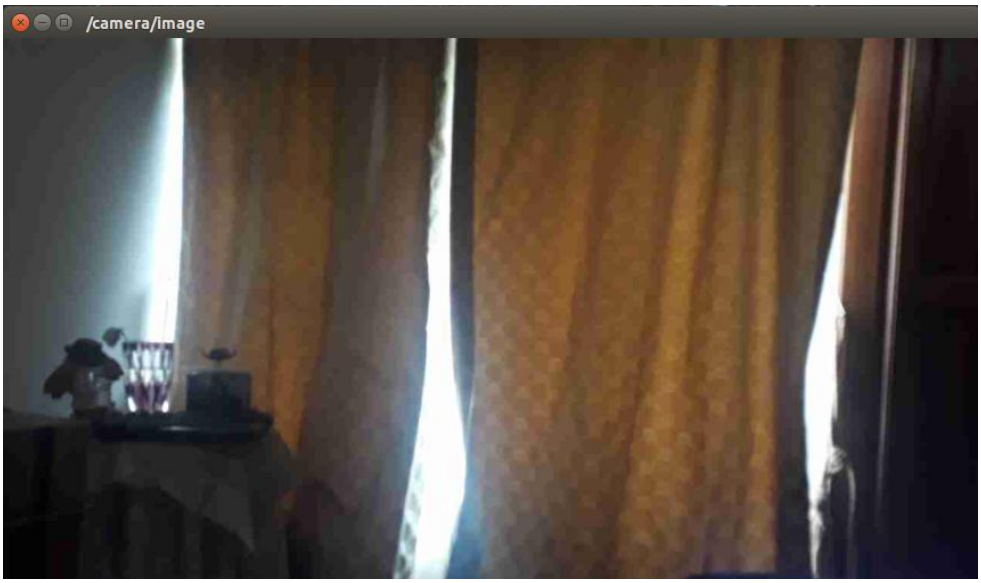

***Figura 30*.** Ejecución del comando de visualización de imágenes de la cámara.

CAPITULO IV

4. ODOMETRÍA VISUAL INERCIAL

4.1 Filtro de Kalman Multi-Estado Restringido

El Filtro de Kalman Multi-Estado Restringido (MSCKF por sus siglas en inglés) es el algoritmo de odometría visual-inercial implementado en esta tesis. Es similar al EKF SLAM asistido inercialmente, pero difieren en que el MSCKF no construye un mapa añadiendo las posiciones de los puntos de interés al vector de estado (González, 2015). Un mapa ofrece información valiosa sobre las correlaciones entre los puntos de interés, sin embargo, procesar esa cantidad de información es bastante costoso computacionalmente.

El MSCKF usa una ventana deslizante de poses pasadas de cámara, las cuales son usadas para triangular los puntos de interés observados sin añadirlos al vector de estado (Mourikis & Roumeliotis, 2007). Después de cada imagen de la cámara es recibida, la pose actual del cuerpo del dispositivo es añadida al vector de estado. Una vez que m número de poses han sido añadidas, la pose más antigua es desechada del vector de estado en de la manera FIFO (el primero en entrar es el primero en salir). En el Algoritmo 1 se puede apreciar una descripción general del algoritmo MSCKF implementado en esta tesis.

Algoritmo 1. Descripción general del MSCKF

if nueva medida de la IMU ***then***

PROPAGAR_ESTADO

end if

if nueva Imagen $\boldsymbol{I_i}$ ***then***

Identificar y rastrear puntos de interés de $\boldsymbol{I_i}$

AUMENTAR_ESTADO

if Puntos de interés rastreados finalizados ***then***

ACTUALIZAR_ESTADO

end if

end if

El MSCKF hace un uso óptimo de las restricciones geométricas que surgen tras observar el mismo punto de interés en diferentes imágenes, sin añadirlas al vector de estado. Esto hace que el costo computacional sea lineal en el número de puntos de interés. Además, los puntos de interés son usados para actualizar el filtro solamente cuando han dejado de ser observados o cuando han sido observados un número m de veces, obteniendo así una estimación muy precisa de la posición en 3D del punto de interés (Li & Mourikis, Vision-aided inertial navigation for resource-constrained systems, 2012). En la Fig. 31 se puede apreciar una ilustración de la ventana deslizante de poses del algoritmo.

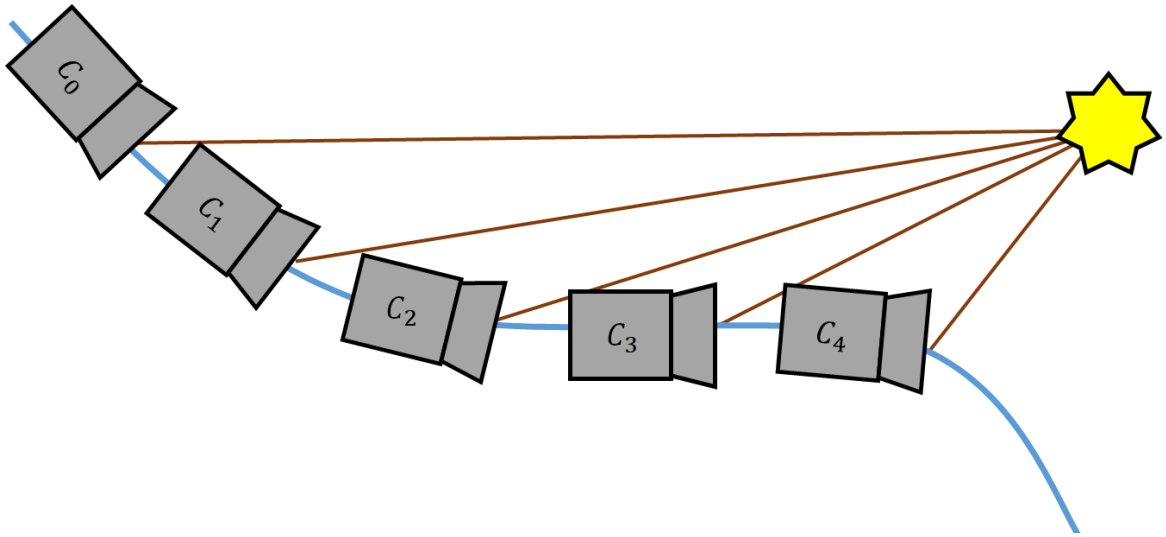

Figura 31. Ventana deslizante de poses de la cámara.

4.1.1 Representación del estado

4.1.1.1 Representación de las rotaciones

La mínima forma de representar una rotación es con tres grados de libertad. Con los ángulos de Euler puede describirse sencillamente una orientación, pero el problema de esta representación es la presencia de singularidades. Esto ocurre cuando los ángulos no están determinados de manera única, resultando en lo que se conoce como gimbal lock. Para prevenir estos problemas, las rotaciones del estado del cuerpo y la rotación de cuerpo-cámara están representadas por cuaterniones (Shelley, 2014).

Un cuaternión es una representación de orientación de cuatro dimensiones que no sufre de singularidades. Una forma popular de ver a los cuaterniones es la representación eje-ángulo (Solá, 2017), en la cual tres de los valores representan un eje en 3D y el cuarto valor es la rotación con respecto a ese eje.

$$\bar{q} = \begin{bmatrix} \vec{q} \\ q_4 \end{bmatrix} = \begin{bmatrix} k_x \sin(\theta/2) \\ k_y \sin(\theta/2) \\ k_z \sin(\theta/2) \\ \cos(\theta/2) \end{bmatrix} \tag{78}$$

Los cuaterniones en esta sección son considerados como cuaterniones unitarios, a menos que se especifique lo contrario. Las matrices de rotación como ${}^{C}_{I}R$ son realmente calculadas desde su cuaternión correspondiente, escrito como ${}^{C}_{I}q$.

4.1.1.2 Marcos de referencia

Para ejecutar correctamente el algoritmo de odometría visual-inercial, es necesario definir tres marcos de referencia. El primer marco de referencia es el de la cámara, C. Este marco se encuentra en la ubicación exacta de la cámara en el dispositivo, en términos de posición y rotación. El segundo marco de referencia es el marco del cuerpo, denotado como I, y se encuentra en la posición exacta de la IMU, con una rotación relativa a la cámara de ${}^{C}_{I}R$, la cual puede ser calculada mediante una calibración IMU-cámara previa, al igual que la posición del cuerpo en el marco referencial de la cámara ${}^{C}p_{I}$. El tercer marco referencial es el marco global G, en el cual el origen es el marco referencial de la primera pose de la cámara C_0. La Fig.32 muestra la ubicación del origen de los marcos referenciales en el dispositivo.

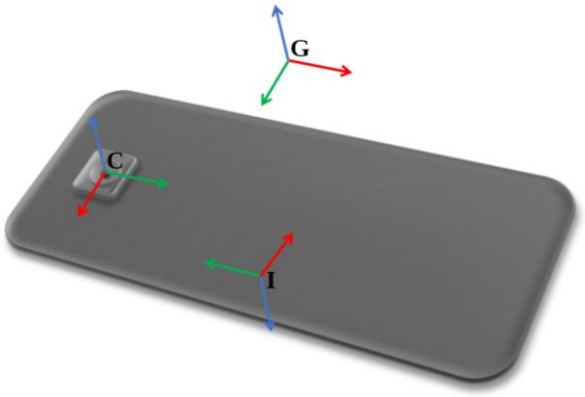

Figura 32. Marcos referenciales ubicados en el dispositivo.

4.1.1.3 Vector de Estado

El vector de estado incluye los parámetros que son constantemente estimados por el filtro. En el caso de esta tesis, el estado nominal completo se divide en dos vectores: el vector de estado de la IMU x_{IMU} y el vector que contiene la ventana deslizante de poses de la cámara (Clement, Peretroukhin, Lambert, & Kelly, 2015). El vector de estado de la IMU contiene los estimados actuales de las magnitudes de la IMU con respecto al marco de referencia global G, junto con los valores de bias del acelerómetro y del giroscopio.

$$x_{IMU} = \begin{bmatrix} {}^{G}q_I^T & b_\omega^T & b_v^T & {}^{G}p_I^T \end{bmatrix}^T \tag{79}$$

Donde ${}^{G}q_I^T$ es el cuaternión unitario que representa la rotación desde el marco de referencia global G hacia el marco de referencia de la IMU, I, b_ω^T es el bias del giroscopio, b_v^T es el bias del acelerómetro y ${}^{G}p_I^T$ es el vector que representa la posición de la IMU en el marco de referencia global.

En un tiempo determinado k, el vector de estado completo del MSCKF contiene el vector de estado estimado actual de la IMU y los estimados de N poses pasadas de la cámara, cada una con siete dimensiones, tres para la posición y cuatro para la rotación. Para que una pose sea considerada en el estado, puntos de interés estimados anteriormente deben ser visibles.

$$\hat{x}_k = \begin{bmatrix} \hat{x}_{IMU}^T & {}^{G}\hat{q}_{C_1}^T & {}^{G}\hat{p}_{C_1}^T & \cdots & {}^{G}\hat{q}_{C_N}^T & {}^{G}\hat{p}_{C_N}^T \end{bmatrix}^T \tag{80}$$

Es importante indicar que el estado de la IMU guarda las magnitudes de la IMU en el marco referencial I, mientras que la ventana deslizante guarda las poses de la cámara en el marco referencial C.

4.1.1.4 Definición del error

En los vectores de la posición, velocidad, o escalares como los parámetros de la IMU el error puede ser modelado sencillamente, solo es necesario obtener la diferencia entre el estado verdadero y el estado estimado.

$$\tilde{p} = p - \hat{p} \tag{81}$$

No obstante, esto no funciona para los cuaterniones. Al ser las rotaciones representadas mediante cuaterniones unitarios, por lo que deben las rotaciones mantener una longitud igual a la unidad, $\|q\| = 1$. Por lo tanto, el error es modelado como la pequeña diferencia en los ángulos en las tres dimensiones, de la forma:

$$\delta q = \hat{q}^{-1} \otimes q \simeq \begin{bmatrix} \frac{1}{2}\delta\theta^T & 1 \end{bmatrix}^T \tag{82}$$

Donde $\otimes$ denota la multiplicación de cuaterniones. Esta representación tiene la ventaja de ser una representación de orientación mínima (3 grados de libertad), además de evitar singularidades encontradas en los ángulos de Euler, ya que los errores en los ángulos son muy insignificantes. Tomando en cuenta este error, podemos definir el estado de error en un instante k.

$$\tilde{x}_k = \begin{bmatrix} \tilde{x}_{IMU}^T & \delta\theta_{C_1}^T & {}^G\tilde{p}_{C_1}^T & \cdots & \delta\theta_{C_N}^T & {}^G\tilde{p}_{C_N}^T \end{bmatrix}^T \tag{83}$$

Donde:

$$\tilde{x}_{IMU} = \begin{bmatrix} \delta\theta_I^T & \tilde{b}_\omega^T & \tilde{b}_v^T & {}^G\tilde{p}_I^T \end{bmatrix}^T \tag{84}$$

Este es el estado de error de 12 dimensiones de la IMU.

Finalmente, es necesario describir la matriz de covarianzas de estado del MSCKF, la cual es la que representa la incertidumbre del estado de error y las covarianzas cruzadas entre todos los

componentes del vector de estado (González, 2015). En la Fig. 33 se puede observar la estructura de la matriz de covarianzas.

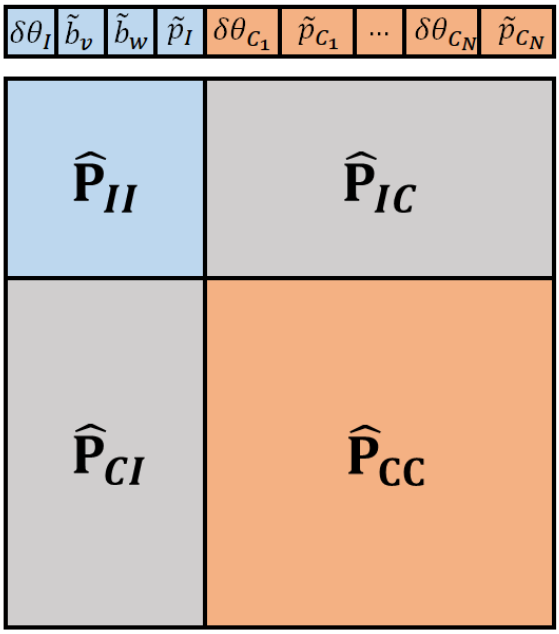

Figura 33. Vector de estado de error y matriz de covarianzas.

La matriz de covarianzas de estado $\widehat{\mathrm{P}}_k$ tiene una dimensión de $(12 + 6N)$ x $(12 + 6N)$. Usando bloques de matrices, la matriz de covarianzas de estado puede ser descrita como:

$$\widehat{\mathrm{P}}_k = \begin{bmatrix} \widehat{\mathrm{P}}_{II} & \widehat{\mathrm{P}}_{IC} \\ \widehat{\mathrm{P}}_{CI} & \widehat{\mathrm{P}}_{CC} \end{bmatrix} \tag{85}$$

Donde $\widehat{\mathrm{P}}_{II}$ es la matriz de covarianza del estado actual de la IMU (con dimensión 12 x 12), $\widehat{\mathrm{P}}_{CC}$ es la matriz de covarianza de la ventana deslizante de poses (con dimensión $6N$ x $6N$), y $\widehat{\mathrm{P}}_{IC}$, $\widehat{\mathrm{P}}_{CI}$ son las covarianzas cruzadas entre el estado actual de la IMU y la ventana deslizante de poses (con dimensión 12 x $6N$ y $6N$ x 12 respectivamente).

4.1.2 Propagación del estado

Cada vez que un dato de la IMU es recibido, la etapa de propagación del estado de la IMU es ejecutada. Consiste básicamente en integrar las magnitudes medidas del vector de estado y

propagar las covarianzas de estado. La evolución del estado de la IMU $\hat{x}_{IMU}$ con respecto al tiempo es descrita por un modelo de movimiento en tiempo continuo (Clement, Peretroukhin, Lambert, & Kelly, 2015):

$$ {}^{I}_{G}\dot{\hat{q}}(t) = \frac{1}{2}\Omega(\hat{\omega}(t)) {}^{I}_{G}\hat{q}(t) \tag{86} $$

$$ {}^{G}\dot{\hat{p}}_{I}(t) = {}^{I}_{G}\hat{R}^{T}\hat{v}(t) \tag{87} $$

$$ \dot{\hat{b}}_{\omega} = 0_{3x1} \tag{88} $$

$$ \dot{\hat{b}}_{v} = 0_{3x1} \tag{89} $$

$$ \hat{v}(t) = v_{m}(t) - \hat{b}_{v} \tag{90} $$

$$ \hat{\omega}(t) = \omega_{m}(t) - \hat{b}_{\omega} \tag{91} $$

$$ \Omega(\hat{\omega}) = \begin{bmatrix} -\hat{\omega}^{\times} & \hat{\omega} \\ -\hat{\omega}^{T} & 0 \end{bmatrix}, \quad \text{donde} \quad \hat{\omega}^{\times} = \begin{bmatrix} 0 & -\hat{\omega}_{3} & \hat{\omega}_{2} \\ \hat{\omega}_{3} & 0 & -\hat{\omega}_{1} \\ -\hat{\omega}_{2} & \hat{\omega}_{1} & 0 \end{bmatrix} \tag{92} $$

Donde ${}^{I}_{G}\hat{R}$ es la matriz de rotación correspondiente a ${}^{I}_{G}\hat{q}$.

Para propagar el modelo de movimiento en tiempo continuo es necesario hacer uso de un algoritmo de integración numérica. En esta tesis, el algoritmo seleccionado para la integración es el método de Euler.

De la misma manera podemos examinar el modelo linealizado en tiempo continuo del estado de error de la IMU:

$$ \dot{\tilde{x}}_{IMU} = F\tilde{x}_{IMU} + Gn_{IMU} \tag{93} $$

Donde los Jacobianos F y G están definidos como:

$$F = \begin{bmatrix} -\widehat{\omega}^{\times} & -1_3 & 0_{3x3} & 0_{3x3} \\ 0_{3x3} & 0_{3x3} & 0_{3x3} & 0_{3x3} \\ 0_{3x3} & 0_{3x3} & 0_{3x3} & 0_{3x3} \\ -{}^{I}_{G}\hat{C}^{T}\hat{v}^{\times} & 0_{3x3} & -{}^{I}_{G}\hat{C}^{T} & 0_{3x3} \end{bmatrix} \tag{94}$$

$$G = \begin{bmatrix} -1_3 & 0_{3x3} & 0_{3x3} & 0_{3x3} \\ 0_{3x3} & 1_3 & 0_{3x3} & 0_{3x3} \\ 0_{3x3} & 0_{3x3} & 0_{3x3} & 1_3 \\ 0_{3x3} & 0_{3x3} & -{}^{I}_{G}\hat{C}^{T} & 0_{3x3} \end{bmatrix} \tag{95}$$

Donde 1_m representa la matriz identidad m-dimensional y $0_{m\,\mathrm{x}\,n}$ representa una matriz de ceros de dimensión m x n. El ruido de proceso de la IMU está definido por:

$$n_{IMU} = \begin{bmatrix} {n_\omega}^T & {n_{b_\omega}}^T & {n_v}^T & {n_{b_v}}^T \end{bmatrix} \tag{96}$$

Y la matriz de ruido Q_I es formada a partir de las varianzas integradas del ruido de la IMU.

4.1.2.1 Propagación de la covarianza de estado

Las estimaciones de las covarianzas de la ventana deslizante de poses y las covarianzas cruzadas entre la IMU y la ventana deslizante de poses, definidas previamente en (85), fueron calculadas de la siguiente manera:

$$\widehat{\mathrm{P}}^{-}_{CC_k} \leftarrow \widehat{\mathrm{P}}_{CC_k} \tag{97}$$

$$\widehat{\mathrm{P}}^{-}_{IC_k} \leftarrow \Phi(t_k + \Delta T, t_k)\widehat{\mathrm{P}}_{IC_k} \tag{98}$$

Donde ΔT es el periodo de muestreo de la IMU.

La matriz de evolución de estado $\Phi(t_k + \Delta T, t_k)$ y la covarianza del estado de la IMU estimada $\widehat{\mathrm{P}}_{IC_k}$ son calculadas de la siguiente forma:

$$\Phi(t_k + \Delta T, t_k) = 1_{12} + \mathrm{F}\Delta T \tag{99}$$

$$\widehat{\mathrm{P}}^{-}_{II_k} \leftarrow \Phi(t_k + \Delta T, t_k)\widehat{\mathrm{P}}^{-}_{II_k}\Phi(t_k + \Delta T, t_k)^T + \mathrm{G}\mathrm{Q}_I\mathrm{G}^T\Delta T \tag{100}$$

Una descripción rápida de la etapa de propagación puede verse en el Algoritmo 2.

Algoritmo 2. Etapa de propagación del estado

function PROPAGAR_ESTADO

Propagar el estado $\hat{x}_{IMU}$

Propagar la covarianza de estado $\widehat{\mathrm{P}}_k$

end

4.1.3 Aumentación del estado

Cuando una nueva imagen está disponible, el vector de estado del MSCKF debe ser aumentado con la pose actual de la cámara. Obtenemos la pose de la cámara aplicando una transformación IMU-cámara ya conocida (${}^{C}_{I}\hat{q},\ {}^{I}p_C$) a una copia de la pose actual de la IMU.

$$ {}^{C_{N+1}}_{G}\hat{q} = {}^{C}_{I}q \otimes {}^{I}_{G}\hat{q}_k \tag{101} $$

$$ {}^{G}\hat{p}_{C_{N+1}} = {}^{G}\hat{p}_I + {}^{I}_{G}\hat{R}_k{}^{T}\ {}^{I}\hat{p}_C \tag{102} $$

Donde ${}^{I}_{G}\hat{R}$ es la matriz de rotación correspondiente a ${}^{I}_{G}\hat{q}_k$.

Asumiendo que el vector de estado del MSCKF ha sido aumentado por N poses de la cámara, la $(N+1)$- ésima pose de la cámara es añadida al vector de estado de acuerdo con:

$$ \hat{x}_k \leftarrow \left[\hat{x}_k{}^{T} \quad {}^{C_{N+1}}_{G}\hat{q}^{T} \quad {}^{G}\hat{p}_{C_{N+1}}{}^{T}\right]^{T} \tag{103} $$

De manera similar, la covarianza de estado del MSCKF es aumentada de acuerdo con:

$$ \widehat{\mathrm{P}}_k \leftarrow \begin{bmatrix} 1_{12+6N} \\ \mathrm{J}_k \end{bmatrix} \widehat{\mathrm{P}}_k \begin{bmatrix} 1_{12+6N} \\ \mathrm{J}_k \end{bmatrix}^{T} \tag{104} $$

Donde el Jacobiano J_k está dado por:

$$ \mathrm{J}_k \leftarrow \begin{bmatrix} {}^{C}_{I}\hat{R}_k & 0_{3x6} & 0_{3x3} & 0_{3x6N} \\ ({}^{I}_{G}\hat{R}_k{}^{T}\ {}^{I}\hat{p}_C)^{\times} & 0_{3x6} & 1_3 & 0_{3x6N} \end{bmatrix} \tag{105} $$

Una descripción rápida de la etapa de aumentación puede verse en el Algoritmo 3.

Algoritmo 3. Etapa de aumentación del estado

function AUMENTAR_ESTADO

Calcular la pose de la cámara ${}^{G}\hat{p}_{C_N}$

if Ventana de poses llena ***then***

Quitar pose más antigua

end if

Añadir la pose al estado $\hat{x}_k$

Aumentar la covarianza de estado $\widehat{\mathrm{P}}_k$

end

4.1.4 Actualización del estado

La actualización del estado es la etapa final del algoritmo MSCKF. Hace uso de las restricciones geométricas de los puntos de interés rastreados para observar y corregir los errores producidos por el sensor de la cámara (Mourikis & Roumeliotis, 2007). La actualización del estado se ejecutará si se produce alguno de los siguientes escenarios (González, 2015):

- Cuando un punto de interés ya no es detectado, lo cual significa que el rastreo de ese punto ha terminado. Esto puede suceder si el punto de interés se ha movido fuera del rango visual de la cámara o ha sido obstruido por algún objeto.
- Cuando el estado está completamente lleno. Cada vez que una nueva imagen llega, la ventana deslizante de poses es aumentada con una copia de la pose actual de la cámara. Sabiendo que la ventana deslizante tiene un número limitado de poses, es necesario deshacerse de las poses antiguas para que haya espacio para las nuevas. Cada vez que esto sucede, un grupo de poses

uniformemente separadas son seleccionadas para realizar una actualización con sus puntos de interés observados, antes de descartarlas.

Sabiendo esto, y después de obtener las ubicaciones estimadas de los puntos de interés a ser usados en la actualización, es necesario aplicar las restricciones de movimiento a la ventana deslizante de poses. Empezamos formando el error de medición correspondiente a una observación $z_i^{(j)}$ del punto de interés f_j de la pose de la cámara C_i.

$$r_i^{(j)} = z_i^{(j)} - \hat{z}_i^{(j)} \tag{106}$$

Donde:

$$z_i^{(j)} = \frac{1}{\hat{Z}_{Ci}^{(j)}} \begin{bmatrix} \hat{X}_{Ci}^{(j)} & \hat{Y}_{Ci}^{(j)} \end{bmatrix} \tag{107}$$

Con:

$${}^{C_i}\hat{p}_{f_j} = \begin{bmatrix} \hat{X}_{Ci}^{(j)} & \hat{X}_{Ci}^{(j)} & \hat{Z}_{Ci}^{(j)} \end{bmatrix}^T \tag{108}$$

$$= {}_{G}^{C_i}\hat{R}({}^{G}\hat{P}_{f_j} - {}^{C_i}\hat{P}_{C_i}) \tag{109}$$

Linealizando la ecuación (106) en los estimados de la pose de la cámara y la ubicación del punto de interés, obtenemos el estimado del error de la medición.

$$r_i^{(j)} \simeq \mathrm{H}_{x,i}^{(j)} \tilde{x}_i + \mathrm{H}_{f,i}^{(j)} \, {}^{G}\tilde{p}_{f_j} + \mathrm{n}_i^{(j)} \tag{110}$$

Donde $\mathrm{H}_{x,i}^{(j)}$ y $\mathrm{H}_{f,i}^{(j)}$ son los Jacobianos de la medición del punto de interés f_j de la pose de la cámara C_i con respecto al vector de estado del MSCKF y la ubicación del punto de interés, respectivamente.

$$\mathrm{H}_{x,i}^{(j)} = \begin{bmatrix} 0 & \mathrm{J}_i^{(j)} & ({}^{C_i}\hat{p}_{f_j})^{\times} & -\mathrm{J}_i^{(j)} \, {}_{G}^{C_i}\hat{R} & 0 \end{bmatrix} \tag{111}$$

$$\mathrm{H}_{f,i}^{(j)} = \mathrm{J}_i^{(j)} \, {}_{G}^{C_i}\hat{R} \tag{112}$$

Es necesario considerar a $\mathrm{n}_i^{(j)}$, que es el ruido Gaussiano de media cero y con una matriz de covarianza $\mathrm{R}_i^{(j)} = diag\{\sigma_{u'}^2, \sigma_{v'}^2\}$, donde$\sigma_{u'}^2$ y $\sigma_{v'}^2$ representan a las varianzas de las mediciones en coordenadas de pixeles del punto de interés que han sido corregidas con los parámetros intrínsecos de la cámara.

Apilando los errores $r_i^{(j)}$, llegamos a una expresión para el vector con dimensión $2M_j$ x 1 del error en la medición del punto de interés f_j con respecto a toda la ventana deslizante de poses de la cámara, donde M_j es el número de poses de la cámara de donde el punto de interés f_j fue observado.

$$r_i^{(j)} = z_i^{(j)} - \hat{z}_i^{(j)} \simeq \mathrm{H}_{x,i}^{(j)} \tilde{x}_i + \mathrm{H}_{f,i}^{(j)} {}^{G}\tilde{p}_{f_j} + \mathrm{n}_i^{(j)} \tag{114}$$

El vector de ruido $\mathrm{n}^{(j)}$ tiene una matriz de covarianzas $\mathrm{R}^{(j)} = diag\{\mathrm{R}_1^{(j)}, \dots, \mathrm{R}_{M_j}^{(j)}\}$

Sin embargo, el MSCKF asume que los errores en la medición son lineales en el vector de estado y que tienen un componente de ruido Gaussiano aditivo de media cero que no está correlacionado al estado. Ya que $\mathrm{H}_f^{(j)} {}^{G}\tilde{p}_{f_j}$ esta correlacionado al estado, $r^{(j)}$ no está en la forma correcta para ser usado en el MSCKF y debe ser modificado para que no se correlacione con el estado. La correlación entre el error en la medición y el vector de estado causa que las estimaciones del filtro se desvíen mucho más de los valores reales que teniendo un error en la medición no correlacionado (Clement, Peretroukhin, Lambert, & Kelly, 2015).

Para transformar a $r^{(j)}$ en una forma que pueda ser usada efectivamente en el MSCKF, es necesario definir una matriz semi-unitaria (matriz no cuadrada en la cual las columnas o filas de la matriz son ortonormales) A en la cual sus columnas forman la base del espacio nulo izquierdo

(vector que multiplicado a $\mathrm{H}_f^{(j)}$ resulta en 0) de $\mathrm{H}_f^{(j)}$, y proyectar $r^{(j)}$ en este espacio nulo para obtener una ecuación de error en la forma correcta.

$$r_o^{(j)} = A^T r^{(j)} \simeq A^T \mathrm{H}_x^{(j)} \tilde{x} + 0 + A^T \mathrm{n}^{(j)} \tag{115}$$

$$= \mathrm{H}_o^{(j)} \tilde{x} + \mathrm{n}_o^{(j)} \tag{116}$$

Ya que $\mathrm{H}_f^{(j)}$ tiene un rango de columna completa (cada una de las columnas de la matriz es linealmente independiente), A tiene una dimensión de $2M_j$ x $(2M_j - 3)$ y $r_o^{(j)}$ tiene una dimensión de $\left(2M_j - 3\right)$ x 1. La matriz de covarianza de $\mathrm{n}_o^{(j)}$ esta dada por $\mathrm{R}_o^{(j)} = A^T \mathrm{R}^{(j)} A$.

Ahora es posible apilar todos los errores $r_o^{(j)}$ para todos los puntos de interés en el grupo actual que llega a:

$$r_o = \mathrm{H}_o \tilde{x} + \mathrm{n}_o \tag{117}$$

La dimensión de este vector puede ser bastante larga en la práctica, así que es necesario usar la descomposición QR () de H_o para reducir la complejidad computacional de la actualización del MSCKF:

$$\mathrm{H}_o = \begin{bmatrix} \mathrm{Q}_1 & \mathrm{Q}_2 \end{bmatrix} \begin{bmatrix} \mathrm{T}_H \\ 0 \end{bmatrix} \tag{118}$$

Donde Q_1 y Q_2 son matrices unitarias y T_H es una matriz triangular superior. Substituyendo este resultado en la ecuación () y multiplicando por $[\mathrm{Q}_1 \quad \mathrm{Q}_2]^T$ obtenemos:

$$\begin{bmatrix} {\mathrm{Q}_1}^T r_o \\ {\mathrm{Q}_2}^T r_o \end{bmatrix} = \begin{bmatrix} \mathrm{T}_H \\ 0 \end{bmatrix} \tilde{x} + \begin{bmatrix} {\mathrm{Q}_1}^T \mathrm{n}_o \\ {\mathrm{Q}_2}^T \mathrm{n}_o \end{bmatrix} \tag{119}$$

Ya que la el valor de ${\mathrm{Q}_2}^T r_o$ es solamente ruido, puede ser despreciado para definir un nuevo término de error para ser usado en la etapa de actualización del MSCKF.

$$r_n = \mathrm{Q_1}^T r_o = \mathrm{T}_H \tilde{x} + \mathrm{Q_1}^T \mathrm{n}_o = \mathrm{T}_H \tilde{x} + \mathrm{n}_n \tag{120}$$

La matriz de covarianza de n_n esta dada por $\mathrm{R}_n = \mathrm{Q_1}^T \mathrm{R}_o^{(j)} \mathrm{Q_1}$.

Finalmente, la ganancia de Kalman y las ecuaciones de actualización pueden ser formuladas para obtener las actualizaciones de las estimaciones del vector de estado y la matriz de covarianza del MSCKF:

$$K = \widehat{\mathrm{P}}_{k+1}^{-} \mathrm{T}_H^T (\mathrm{T}_H \widehat{\mathrm{P}}_{k+1}^{-} \mathrm{T}_H^T + \mathrm{R}_n)^{-1} \tag{121}$$

$$\Delta x_k \leftarrow K r_n \tag{122}$$

$$\widehat{\mathrm{P}}_k = (1_{12+6N} - K\mathrm{T}_H)\widehat{\mathrm{P}}_{k+1}^{-} (1_{12+6N} - K\mathrm{T}_H)^T + K\mathrm{R}_n K^T \tag{123}$$

El Algoritmo 4 muestra una descripción rápida de la etapa de actualización del MSCKF.

Algoritmo 4. Etapa de actualización del estado

function ACTUALIZAR_ESTADO

Triangular la posición de ${}^G\hat{P}_{f_j}$ utilizando Gauss-Newton

for Todas las observaciones $z_i^{(j)}$

Calcular el error en la medición $r_i^{(j)}$

Calcular Jacobianos de la medición del punto de interés f_j

Apilar error en la medición y Jacobianos

end for

Calcular la ganancia de Kalman K

Estimar $\hat{x}_k$ del próximo instante

Estimar $\widehat{\mathrm{P}}_k$ del próximo instante

end

CAPITULO V

5. PRUEBAS Y RESULTADOS

5.1 Hardware del sistema

La presente tesis se ha desarrollado mediante el uso de dos componentes principales de hardware:

- Un teléfono inteligente que posea un sensor IMU de mínimo seis grados de libertad, un sensor de cámara, sistema operativo Android y capacidad de conexión a una red Wi-Fi. Este teléfono es el usado en la transmisión de los datos de la IMU y de la cámara. El teléfono seleccionado para esta tesis es el Samsung Galaxy S8, con 4 GB de RAM y procesador Snapdragon 835 de ocho núcleos (4x2.35 GHz y 4x1.9 GHz), IMU LSM6DSL de seis grados de libertad. En la Fig. 34 se puede observar el teléfono utilizado.

Figura 34. Samsung Galaxy S8.
Fuente: (Samsung Galaxy S8, 2018)

- Un computador con sistema operativo Ubuntu 14.04, el cual va a ser el encargado de la ejecución del algoritmo del MSCKF. El computador utilizado es ASUS ROG G551jw con procesador Intel Core i7 de ocho núcleos de 2.60GHz y 16GB DDR3L de RAM, como puede apreciarse en la Fig. 35. El algoritmo fue implementado en MATLAB 2017b, y por razones de tiempo no pudo ser totalmente portado a Python para ser ejecutado ROS.

Figura 35. ASUS ROG G551jw.
Fuente: *(ASUS ROG G551JW, 2018)*

5.2 Resultados de la calibración de la cámara

Para la calibración de la cámara, un conjunto de 18 imágenes tomadas al patrón de calibración fueron las usadas en el programa para encontrar los parámetros intrínsecos de la cámara. Los resultados arrojados por el programa son los siguientes:

$$\begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2199.02681 & 0 & 1017.76745 \\ 0 & 2131.74580 & 1486.49311 \\ 0 & 0 & 1 \end{bmatrix}$$

5.3 Resultados de la calibración de la IMU

Para la calibración de la IMU del teléfono, un ROSbag de aproximadamente 4 horas fue grabado. La gráfica de Allan para el acelerómetro puede apreciarse en la Fig. 36.

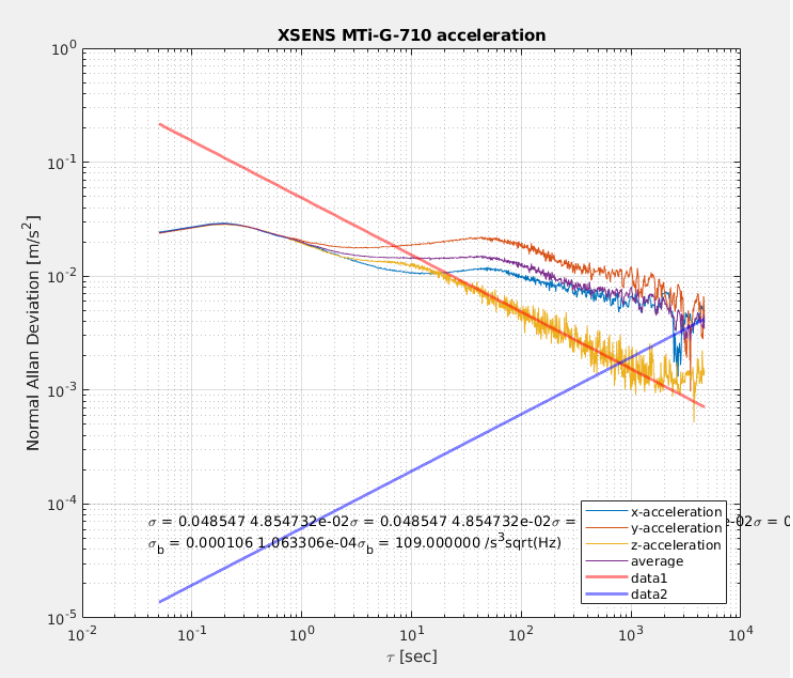

Figura 36. Gráfica AD para el acelerómetro.

Y la gráfica de Allan para el giroscopio puede observarse en la Fig. 37.

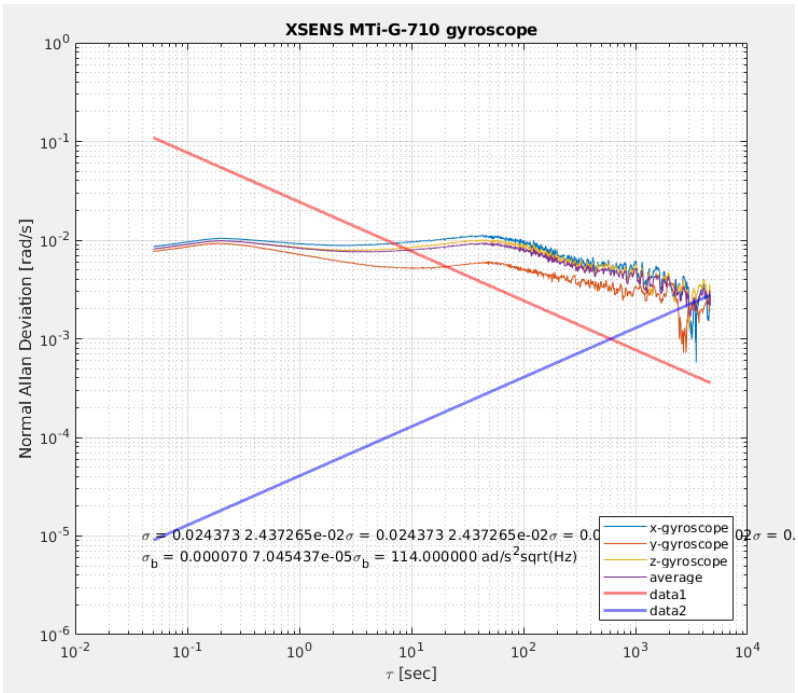

Figura 37. Gráfica AD para el giroscopio.

Los valores de ruido blanco y de bias obtenidos para el acelerómetro y el giroscopio son los siguientes:

$$\sigma_a = 0.0485473\,\frac{m}{s^2}\frac{1}{\sqrt{Hz}}, \quad \sigma_{ba} = 1.063306\text{x}10^{-4}\,\frac{m}{s^3}\frac{1}{\sqrt{Hz}}$$

$$\sigma_g = 0.0253726\,\frac{rad}{s}\frac{1}{\sqrt{Hz}}, \quad \sigma_{bg} = 7.045437\text{x}10^{-5}\,\frac{rad}{s^2}\frac{1}{\sqrt{Hz}}$$

5.4 Resultados de la calibración cámara-IMU

Para la calibración cámara-IMU, se grabaron los datos del teléfono con la cámara apuntando siempre al patrón de calibración. Fue necesario excitar la IMU en todos los ejes, tanto en rotación como en traslación. Los resultados arrojados por el programa fueron los siguientes:

$$T_{cam-IMU} = \begin{bmatrix} 0.605295 & -0.792728 & -0.721037 & -0.000267 \\ -0.789512 & -0.609435 & 0.0725131 & 0.0006942 \\ -0.1014258 & 0.0130349 & -0.9947577 & 0.00220205 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

El programa entrega además una gráfica de los errores de reproyección (distancia entre un punto proyectado y un punto medido) de la cámara. En la Fig. 38 se pueden apreciar estos errores.

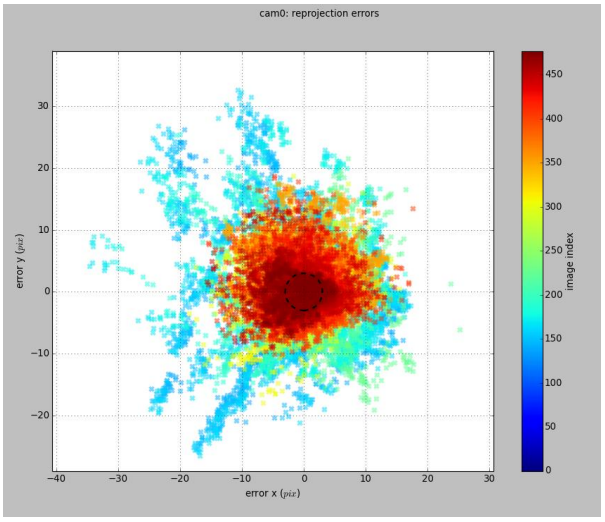

Figura 38. Gráfica de los errores de reproyección de la cámara.

5.5 Resultados del MSCKF con el dataset de KITTI

Para probar el algoritmo, se utilizó el dataset de KITTI, y se implementó el algoritmo del paper de (Clement, Peretroukhin, Lambert, & Kelly, 2015) con algunas modificaciones. El uso de este dataset ofrece múltiples ventajas: Puede hacerse una comparación con los valores reales y obtener un verdadero error en la estimación, todos los parámetros de calibración y posiciones de los elementos están ya incluidos en el dataset, las imágenes ya tienen las correcciones de distorsión aplicadas y los valores de la IMU ya están corregidos para los errores de desalineación y escala. En la Fig. 39 se puede apreciar una gráfica de la estimación de movimiento del MSCKF y la trayectoria real. La trayectoria usada fue el drive 0036 del dataset.

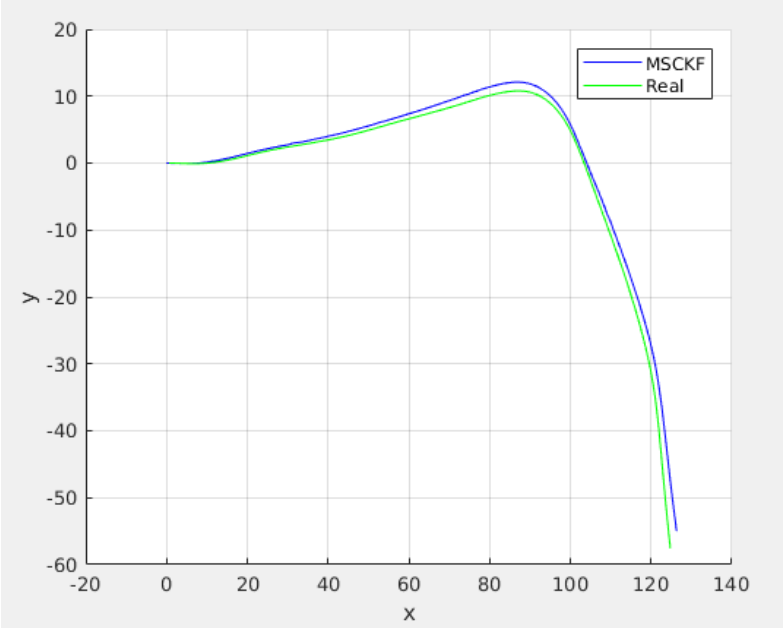

Figura 39. Trayectoria estimada por el MSCKF vs trayectoria real del dataset de KITTI.

5.6 Resultados del MSCKF con los datos del teléfono

Por motivos de tiempo, no se pudo ejecutar el algoritmo en tiempo real. Para hacer uso de los datos del teléfono, se grabaron los datos de la IMU y de la cámara mediante una aplicación en el teléfono. Ya que se modelo el estado con la velocidad en vez de la aceleración, es necesario

preintegrar los valores de la aceleración. Para esto, se creó un programa que utiliza la integración numérica RK4 (Runge Kutta method, 2018). Los videos fueron grabados a 30 cuadros por segundo, y los datos de la IMU a 15Hz, estos datos están sincronizados. Ya que los sensores del teléfono no son tan sensibles como los del dataset de KITTI, además de tener que estimar los valores de calibración de la cámara y de la IMU e introducir error en la estimación al sostener el teléfono en la mano, la estimación del MSCKF no va a ser tan precisa, y va a haber una gran cantidad de ruido introducido. Otro gran problema es la ausencia de los valores reales, lo cual impide que podamos medir el error real de la estimación. En la Fig. 40 se puede apreciar una de las trayectorias estimadas por el MSCKF.

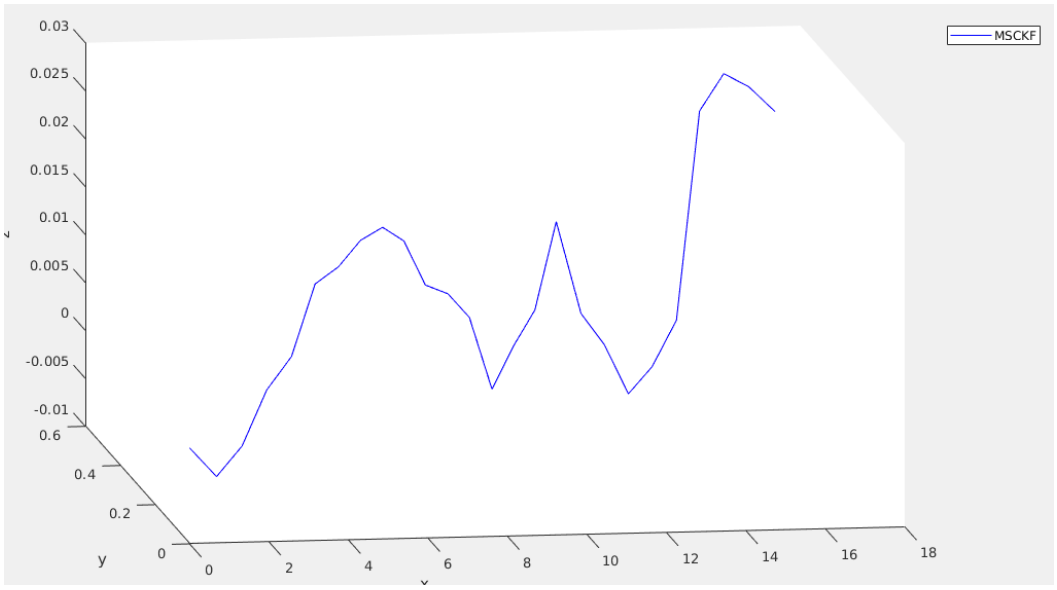

Figura 40. Trayectoria estimada por el MSCKF de los datos del teléfono.

CAPÍTULO VI

6. CONCLUSIONES Y RECOMENDACIONES

6.1 Conclusiones

Se desarrolló satisfactoriamente el sistema de odometría visual inercial en base al algoritmo MSCKF (Filtro de Kalman Multi-Estado Restringido) y con los datos obtenidos de un teléfono inteligente. El algoritmo no pudo ser ejecutado en tiempo real, debido a limitaciones de tiempo. Hasta el momento de escribir esta tesis, la implementación en tiempo real de este algoritmo no ha sido lograda. Esto se debe mayormente a la dificultad de enviar los datos inerciales y visuales del teléfono a ROS. En esta tesis este problema ha sido solventado con éxito, facilitando implementación de este algoritmo en tiempo real para trabajos futuros.

Se logró desarrollar una aplicación para Android capaz de enviar los datos capturados por la cámara y por los sensores inerciales hacia un computador con ROS. Los datos se transmiten en tiempo real a tópicos de ROS. Uno de los mayores retos en esta tesis fue lograr la comunicación entre ROS, Android y OpenCV. Esta comunicación pudo lograrse con éxito.

La aplicación de Android consta de varias funciones que son necesarias para el correcto funcionamiento del algoritmo. Las cámaras de los teléfonos tienen algoritmos que regulan automáticamente el brillo, la exposición a la luz y el foco de la cámara. Es por eso que la aplicación está programada para deshabilitar estos mecanismos automáticos de la cámara.

El algoritmo de MSCKF tiene varias ventajas frente a otros algoritmos: es ligero computacionalmente, es rápido, tiene una alta precisión y tiene una buena respuesta a rotaciones.

La desventaja principal de este algoritmo es su dificultad matemática, lo cual lo hace muy difícil de implementar.

Mediante herramientas externas como Kalibr y OpenCV se pudo calibrar todos los parámetros de ruido de la IMU, los parámetros intrínsecos de la cámara y la transformación que hay entre el marco referencial de la cámara y el de la IMU. Se comprobé que los valores de estos parámetros estos parámetros están dentro de un rango admisible, y fueron usados satisfactoriamente en el algoritmo.

El programa se implementó primeramente en MATLAB, ya que MATLAB tiene todas las herramientas necesarias para procesar los datos y visualizar fácilmente la ejecución del algoritmo. Una vez se pudo ejecutar el algoritmo exitosamente en MATLAB, se planeó portar el algoritmo a Python para ROS, pero por motivos de tiempo no se pudo lograr esto.

Las primeras pruebas del programa fueron realizadas usando el dataset de KITTI, ya que este dataset tiene herramientas para procesar los datos en MATLAB, además de proveer todos los datos de calibración de los sensores. Se obtuvieron resultados excelentes con este dataset.

Una vez se pudo ejecutar el programa con el dataset de KITTI, se procedió a probar el algoritmo con los datos del teléfono inteligente. Se crearon herramientas especiales para procesar los datos del teléfono en MATLAB. Después de varias semanas de prueba y error, se pudo correr el algoritmo con los datos del teléfono. Los resultados obtenidos son muy buenos.

Para la realización de esta tesis tuve que leer varios libros y papers. Aprendí muchos conceptos útiles de matemática que no había aprendido en la universidad, y complemente algunos conceptos que solo había topado superficialmente. Los conocimientos que obtuve en esta tesis me van a servir mucho para proyectos futuros.

6.2 Recomendaciones

El algoritmo de MSCKF tiene una complejidad matemática muy elevada, y para poder entenderlo e implementarlo en código es necesario tener claros conceptos de estadística, física, algebra lineal, calculo, trigonometría, procesos estocásticos, integraciones numéricas y cuaterniones.

Se recomienda comentar el código, tabular y seguir las convenciones para nombrar variables y funciones en el código. Una vez que el código se vuelve grande, aplicar estos protocolos es algo totalmente necesario para evitar errores en el programa.

Al momento de correr el algoritmo, se recomienda no hacer movimientos bruscos o movimientos muy rápidos. El algoritmo no responde bien a este tipo de movimientos.

Un error común al momento de calibrar la cámara o la IMU es el considerar los movimientos al inicio y al final del algoritmo. Es necesario no considerar estos movimientos en el momento de la calibración.

Es necesario que el teléfono inteligente tenga una versión de Android mayor a la 6.0 para su correcto funcionamiento. La aplicación ha sido probada en un teléfono con Android Oreo 8.1.

Al momento de tener el teléfono en la mano para la toma de datos, se introduce un poco de ruido no intencional por el movimiento de la mano. Este ruido es difícil de modelar, por lo que es normal que la gráfica de salida tenga un pequeño ruido.

.

REFERENCIAS BIBLIOGRÁFICAS

Aguilar, W. G., & Angulo, C. (2014). *Real-time video stabilization without phantom movements for micro aerial vehicles.* EURASIP Journal on Image and Video Processing, 1-13.

Aguilar, W. G., & Angulo, C. (2014). *Robust video stabilization based on motion intention for low-cost micro aerial vehicles.* 11th International Multi-Conference on Systems, Signals & Devices (SSD). Barcelona.

Aguilar, W. G., & Angulo, C. (2016). *Real-Time Model-Based Video Stabilization for Microaerial Vehicles.* Neural Processing Letters, 459-477.

Aguilar, W. G., & Morales, S. (2016). *3D Environment Mapping Using the Kinect V2 and Path Planning Based on RRT Algorithms*. Electronics, 70-74.

Aguilar, W. G., Angulo, C., & Costa-Castello, R. (2017). *Autonomous Navigation Control for Quadrotors in Trajectories Tracking.* Lecture Notes in Computer Science, (págs. 287-297).

Aguilar, W. G., Angulo, C., & Pardo, J. A. (2017). *Motion intention optimization for multirotor robust video stabilization.* Chilean Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON). Pucón, Chile.

Aguilar, W. G., Angulo, C., Costa, R., & Molina, L. (2014). *Control autónomo de cuadricópteros para seguimiento de trayectorias.* IX Congreso de Ciencia y Tecnología ESPE. Sangolquí.

Aguilar, W. G., Luna, M. A., Moya, J. F., Abad, V., Ruiz, H., & Parra, H. (2017). *Pedestrian Detection for UAVs Using Cascade Classifiers and Saliency Maps*. Lecture Notes in Computer Science , (págs. 563-574).

Aguilar, W. G., Luna, M. A., Moya, J. F., Luna, M. P., Abad, V., & Ruiz, H. (2017). *Cascade Classifiers and Saliency Maps Based People Detection.* Lecture Notes in Computer Science, (págs. 501-510).

Aguilar, W. G., Luna, M. A., Moya, J. F., Luna, M. P., Abad, V., & Ruiz, H. (2017). *Real-Time Detection and Simulation of Abnormal Crowd Behavior*. Lecture Notes in Computer Science, (págs. 420-428).

Aguilar, W. G., Luna, M., Moya, J., Abad, V., Parra, H., & Ruiz, H. (2017). *Pedestrian Detection for UAVs Using Cascade Classifiers with Meanshift.* IEEE 11th International Conference on Semantic Computing (ICSC). San Diego.

Aguilar, W. G., Manosalvas, J. F., Guillén, J. A., & Collaguazo, B. (2018). *Robust Motion Estimation Based on Multiple Monocular Camera for Indoor Autonomous Navigation of Micro Aerial Vehicle.* International Conference on Augmented Reality, (págs. 547-561).

Aguilar, W. G., Morales, S., Ruiz, H., & Abad, V. (2016). *Real-Time Model-Based Video Stabilization for Microaerial Vehicles.* Neural Processing Letters, 459-477.

Aguilar, W. G., Morales, S., Ruiz, H., & Abad, V. (2017). *RRT* GL Based Path Planning for Virtual Aerial Navigation.* Lecture Notes in Computer Science, 176-184.

Aguilar, W. G., Rodríguez, G., Álvarez, L., Sandoval, S., Quisaguano, F., & Limaico, A. (2017). *On-Board Visual SLAM on a UGV Using a RGB-D Camera.* En Lecture Notes in Computer Science, (págs. 298-308).

Aguilar, W. G., Rodríguez, G., Álvarez, L., Sandoval, S., Quisaguano, F., & Limaico, A. (2017). *Real-Time 3D Modeling with a RGB-D Camera and On-Board Processing.* Lecture Notes in Computer Science, (págs. 410-419).

Aguilar, W. G., Rodríguez, G., Álvarez, L., Sandoval, S., Quisaguano, F., & Limaico, A. (2017). *Visual SLAM with a RGB-D Camera on a Quadrotor UAV Using on-Board Processing.* Lecture Notes in Computer Science, (págs. 596-606).

Aguilar, W. G., Verónica, C., & José, P. (2017). *Obstacle Avoidance Based-Visual Navigation for Micro Aerial Vehicles*. Electronics, 10.

Aguilar, W. G., Verónica, C., & José, P. (2017). *Obstacle Avoidance for Low-Cost UAVs. IEEE 11th International Conference on Semantic Computing (ICSC)*. San Diego.

ASUS ROG G551JW. (2018). Obtenido de ASUS Global: https://www.asus.com/ROG-Republic-Of-Gamers/ROG-G551JW/

Bay, H., Ess, A., Tuytelaars, T., & Gool, L. V. (2008). *Speeded-Up Robust Features (SURF).* Computer Vision and Image Understanding, 346-359.

Bloesch, M., Omari, S., Hutter, M., & Siegwart, R. (2015). *Robust visual inertial odometry using a direct EKF-based approach.* 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).

Burger, W., & Burge, M. J. (2016). *Scale-Invariant Feature Transform (SIFT)*. Texts in Computer Science Digital Image Processing, 609-664.

Clement, L. E., Peretroukhin, V., Lambert, J., & Kelly, J. (2015). *The Battle for Filter Supremacy: A Comparative Study of the Multi-State Constraint Kalman Filter and the Sliding Window Filter*. 2015 12th Conference on Computer and Robot Vision.

Engel, J., Sturm, J., & Cremers, D. (2013). *Semi-dense Visual Odometry for a Monocular Camera*. 2013 IEEE International Conference on Computer Vision.

Gauss–Newton algorithm. (2018). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Gauss–Newton_algorithm

González, F. (2015). *Visual Inertial Odometry for Mobile Robotics.*

Huang, A. S., Bachrach, A., Henry, P., Krainin, M., Maturana, D., Fox, D., & Roy, N. (2016). *Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera.* Springer Tracts in Advanced Robotics Robotics Research, (págs. 235-252).

Kalibr the Calibration Toolbox. (2018). Obtenido de Kalibr: https://github.com/ethz-asl/kalibr

Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. Journal of Basic Engineering, 35.

Leutenegger, S., Lynen, S., Bosse, M., Siegwart, R., & Furgale, P. (2014). *Keyframe-based visual–inertial odometry using nonlinear optimization.* The International Journal of Robotics Research, (págs. 314-334).

Li, M., & Mourikis, A. I. (2012). *Improving the accuracy of EKF-based visual-inertial odometry.* 2012 IEEE International Conference on Robotics and Automation.

Li, M., & Mourikis, A. I. (2012). *Vision-aided inertial navigation for resource-constrained systems.* 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems.

Li, M., & Mourikis, A. I. (2013). *High-precision, consistent EKF-based visual-inertial odometry.* The International Journal of Robotics Research, (págs. 690-711).

Li, M., Kim, B. H., & Mourikis, A. I. (2013). *Real-time motion tracking on a cellphone using inertial sensing and a rolling-shutter camera.* 2013 IEEE International Conference on Robotics and Automation.

Matthies, L., Szeliski, R., & Kanade, T. (1988). *Incremental estimation of dense depth maps from image sequences.* Proceedings CVPR 88: The Computer Society Conference on Computer Vision and Pattern Recognition.

Montiel, J. C. (2006). *Unified Inverse Depth Parametrization for Monocular SLAM.* Robotics: Science and Systems II.

Mouats, T., Aouf, N., Sappa, A. D., Aguilera, C., & Toledo, R. (2015). *Multispectral Stereo Odometry.* IEEE Transactions on Intelligent Transportation Systems, (págs. 1210-1224).

Mourikis, A. I., & Roumeliotis, S. I. (2007). *A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation*. Proceedings 2007 IEEE International Conference on Robotics and Automation.

Newcombe, R. A., Lovegrove, S. J., & Davison, A. J. (2011). *DTAM: Dense tracking and mapping in real-time.* 2011 International Conference on Computer Vision.

Nister, D., Naroditsky, O., & Bergen, J. (2004). *Visual odometry.* Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2004.

Orbea, D., Moposita, J., Aguilar, W. G., Paredes, M., León, G., & Jara-Olmedo, A. (2017). *Math Model of UAV Multi Rotor Prototype with Fixed Wing Aerodynamic Structure for a Flight Simulator*. Lecture Notes in Computer Science, (págs. 199-211).

Python Programming Language. (2018). Obtenido de Python: https://www.python.org/

ROS Powering the world's robots. (2018). Obtenido de ROS: http://www.ros.org/

Runge Kutta method. (2018). Obtenido de Wolfram MathWorld: http://mathworld.wolfram.com/Runge-KuttaMethod.html

Samsung Galaxy S8. (2018). Obtenido de Samsung Store: https://www.samsung.com/us/mobile/phones/galaxy-s/galaxy-s8-64gb--verizon--midnight-black-sm-g950uzkavzw/

Shelley, M. (2014). *Monocular Visual Inertial Odometry.*

Solá, J. (2017). *Quaternion kinematics for the error-state Kalman filter*.

Tanskanen, P., Naegeli, T., Pollefeys, M., & Hilliges, O. (2015). *Semi-direct EKF-based monocular visual-inertial odometry.* 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).

The Extended Kalman Filter: An Interactive Tutorial. (2014). Obtenido de The Extended Kalman Filter: An Interactive Tutorial: https://home.wlu.edu/~levys/kalman_tutorial/

The OpenCV Library. (2018). Obtenido de OpenCV: https://opencv.org/

Thrun, S., Burgard, W., & Fox, D. (2010). *Probabilistic robotics*. Cambridge.

Tomasi, C., & Kanade, T. (1993). *Shape and motion from image streams: A factorization method.* Proceedings of the National Academy of Sciences, 9795-9802.

Torr, P., & Zisserman, A. (2000). *MLESAC: A New Robust Estimator with Application to Estimating Image Geometry*. Computer Vision and Image Understanding, 138-156.

Usenko, V., Engel, J., Stuckler, J., & Cremers, D. (2016). *Direct visual-inertial odometry with stereo cameras.* 2016 IEEE International Conference on Robotics and Automation (ICRA).

Zhang, J., & Singh, S. (2014). *LOAM: Lidar Odometry and Mapping in Real-time.* Robotics: Science and Systems X.