Universidad Nacional de Córdoba

Trabajo Especial

LOCALIZACIÓN VISUAL-INERCIAL EN TIEMPO REAL PARA APLICACIONES DE XR

Autor:
Mateo de Mayo

Director:
Dr. Nicolás Wolovick

Licenciatura en Ciencias de la Computación

Universidad Nacional de Córdoba
Facultad de Matemática, Astronomía, Física y Computación

Marzo 2022

Mateo de Mayo: *Localización visual-inercial en tiempo real para aplicaciones de XR*. Extensión del runtime OpenXR de código libre Monado con sistemas de VIO y VI-SLAM. Marzo 2022.

RESUMEN

Las aplicaciones de realidad virtual (VR), aumentada (AR) y sus derivadas, englobadas dentro del término XR, necesitan métodos para localizar y entender los movimientos realizados por el usuario y así poder actualizar la simulación que ejecutan de manera acorde. El problema de localización ha sido uno de los mayores desafíos a la hora de producir dispositivos de realidad virtual para electrónica de consumo en los últimos años. Afortunadamente, han ocurrido grandes avances en la localización visual-inercial mediante cámaras y sensores inerciales; dados principalmente gracias a su continua mejora y abaratamiento relacionados con su producción masiva para telefonía móvil. A pesar de esto, sistemas de este tipo que puedan ser utilizados para XR se desarrollan mayormente como parte de productos privativos. En este trabajo, se estudian sistemas de localización visual-inercial provenientes de la academia y se detallan las dificultades propias de integrarlos y adaptarlos para aplicaciones de XR. En particular se los integra con Monado, la implementación de código libre del estándar OpenXR. De esta forma, se presenta una de las primeras soluciones completamente libres que permite localizar dispositivos XR con este tipo de métodos visual-inerciales en sistemas operativos basados en GNU/Linux.

ABSTRACT

Applications for virtual reality, augmented reality, and their derivatives, all of them encompassed in the term XR, need ways to keep track of motion in the physical world to be able to reflect these movements accordingly in the simulated world. Tracking has been one of the main challenges in bringing VR devices into mainstream consumer electronics in recent years. Fortunately, there has been breakthroughs in visual-inertial tracking with cameras and inertial sensors due to their continuous improvement and reduced costs thanks to the massive production tied to the mobile phone industry. However, systems that perform this kind of tracking are developed for mostly privative products. In this work, we study visual-inertial tracking systems that come from academia, and detail the difficulties that come from integrating and adapting them for XR applications. In particular, we integrate them to Monado, the open-source implementation of the OpenXR standard. Therefore, we present one of the first completely open solutions that integrates this kind of tracking for XR applications and devices on GNU/Linux-based operating systems.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE FRAGMENTOS

Parte I

PRELIMINARES

Esta primera parte está enfocada a intentar entender el problema encarado en este trabajo. Además de una *introducción* general a algunos de los conceptos de XR y SLAM, veremos los *fundamentos* que serán utilizados de referencia a lo largo del escrito.

1 INTRODUCCIÓN

1.1 LOCALIZACIÓN EN XR

Los sistemas de *realidad virtual* y *realidad aumentada*, o *VR* y *AR* respectivamente por sus siglas en inglés, intentan valerse de nuestras formas usuales de interactuar con el mundo que nos rodea, para así generar simulaciones con características muy particulares y difíciles de obtener de otra forma.

En el caso de VR uno usualmente tiene cascos con pantallas o, en inglés, *head mounted displays* (*HMD*), y algún tipo de controles manejados con ambas manos. Estos permiten aplicaciones con un alto nivel de inmersión. La característica de poder adentrarse profundamente en la simulación son deseadas para fines que varían desde el entretenimiento, generalmente en forma de videojuegos, hasta el entrenamiento para situaciones críticas, como lo pueden ser operaciones médicas o la navegación de aeronaves. Por el lado de AR, se suele pensar en dispositivos que permiten agregar o "aumentar" lo que uno ve con información adicional. Estos pueden ser celulares con cámaras o incluso lentes con pantallas transparentes que superponen información 2D o modelos 3D sobre la escena real. Además existen otros términos como *realidad mixta* (*MR*) que intentan hacer referencia a los distintos matices en los que la realidad puede ser combinada con la simulación; es común utilizar el término *XR* para englobar a este espectro de sistemas, a veces también referido como *realidad extendida*. En general, todos ellos comparten problemas similares que son propios de la percepción humana y están relacionados con nuestra psicología y fisiología [1].

Para producir experiencias de XR, es necesario coordinar un gran número de sistemas internos que se encuentran a lo largo del hardware y del software para XR. Uno de los problemas fundamentales y particularmente complicados en XR es el de la *localización* o *tracking*; esta es la capacidad del sistema de identificar la posición y orientación del dispositivo XR, su *pose*, en el entorno para poder reflejarla en la experiencia simulada. En este trabajo nos enfocaremos en el problema de localización. Para una lectura más comprensiva sobre el resto de los problemas a los que XR se enfrenta, y en particular VR, se recomienda el trabajo de LaValle [1].

Hay muchas formas de encarar el problema de localización y en general todas requieren del uso inteligente de sensores físicos, ya sean mecánicos, inerciales, magnéticos, ópticos, o incluso acústicos [2]. Todos estos métodos de tracking comparten un problema en común: los sensores son imperfectos y sus mediciones son *ruidosas*. Tomemos por

[1] *Al considerar sistemas de VR deben tenerse en cuenta, además de los desafíos de ingeniería, los problemas propios de comprender como nuestra percepción afecta lo que experimentamos. Lograr entender estos procesos cognitivos lo suficiente como para valernos de ellos a la hora de diseñar sistemas de XR es fundamental para obtener experiencias inmersivas. Esto puede pensarse como un trabajo de "ingeniería inversa" sobre nuestra propia biología* [1].

ejemplo uno de los paquetes de sensores más comúnmente utilizados en esta área, una *unidad de medición inercial* o *IMU* por sus siglas en inglés. Estas integran *giroscopios* para la medición de velocidad angular, *acelerómetros* para la aceleración lineal, y algunas veces también incluyen *magnetómetros*. En teoría, si sus mediciones fueran perfectas, una IMU debería proveer suficiente información para determinar la pose en el espacio de un dispositivo que la contenga. Pero incluso las IMU de mayor calidad acumulan tanto ruido que la integración de sus mediciones durante cortos períodos de tiempo devuelve poses que tienen un error o *drift* de cientos de metros. Valores aproximados para distintas categorías de IMU pueden verse en la Tabla 1.1 [3, secc 3.3]; las IMU utilizadas en dispositivos XR entran en la categoría para *consumidor* final listada en la tabla. Para contrarrestar la naturaleza imperfecta de sensores físicos como estos, los enfoques más exitosos de localización emplean una combinación de múltiples sensores junto a algoritmos de fusión ingeniosos capaces de integrar tipos de muestras tremendamente distintos en una estimación de la pose que es suficientemente buena.

Tabla 1.1: Error acumulado por distintas IMU

Categoría / Tiempo	**1 s**	**10 s**	**60 s**	**10 min**	**1 hr**
Consumidor	6 cm	6.5 m	400 m	200 km	39.000 km
Industrial	6 mm	0.7 m	40 m	20 km	3.900 km
Táctico	1 mm	8 cm	5 m	2 km	400 km
Navegación	<1 mm	1 mm	50 cm	100 m	10 km

En años recientes, la llamada localización *visual-inercial* (*VI*) ha cobrado gran popularidad en el ecosistema de XR. Al emplear cámaras, usualmente al menos dos, junto a una IMU dentro de un dispositivo XR (p. ej. un HMD o un celular). La localización VI estima la pose del agente haciendo que el mismo dispositivo "mire" hacia su entorno para ganar entendimiento del mismo. Uno de los principales beneficios de este tipo de tracking es que tener todos los sensores empaquetados dentro del dispositivo resulta muy conveniente para el usuario, ya que no necesita colocar ni configurar ningún tipo de sensor externo en su entorno. Este tipo de localización para aplicaciones de XR se consideraba inviable hasta hace un poco más de una década, pero avances en la capacidad de cómputo y en las técnicas de fusión han hecho posible que este ya no sea el caso.

Ya existen dispositivos XR que emplean localización visual-inercial de forma exitosa en productos comerciales como el *Meta Quest*[2], los

2 https://www.oculus.com/quest-2/

cascos *Windows Mixed Reality*[3] o incluso los SDK *ARCore*[4] y *ARKit*[5] utilizados en celulares inteligentes. No obstante, todas estas soluciones son propietarias, por lo tanto no hay manera de mejorarlos, reusarlos en nuevos proyectos o productos, o incluso aprender de su implementación a menos que se adquieran licencias especiales de sus fabricantes.

1.2 SLAM Y VIO PARA LOCALIZACIÓN VISUAL-INERCIAL

Afortunadamente, el área académica en navegación visual-inercial ha experimentado un gran desarrollo en estas últimas décadas. No solo es un problema central en áreas como las de robótica, sino que además es atractiva por combinar una diversa cantidad de áreas como visión por computadora, fusión de sensores, optimización, estimación probabilística, entre otras. Esta ola de investigación ha dado lugar a una gran cantidad de implementaciones de software libre, con distintos grados de rendimiento, robustez, precisión, aplicaciones, facilidad de uso, entre otras propiedades de interés. Recursos como OpenSLAM[6] y estudios como los de las referencias [4] o [5] pueden listar docenas de sistemas disponibles para considerar. Más aún, cada año nuevos sistemas aparecen mientras otros dejan de ser mantenidos. Seguir los avances del área puede ser desafiante, pero esto es una consecuencia de su desarrollo tan activo.

Específicamente, en este trabajo trataremos con sistemas de *localización y mapeo simultáneo* (*SLAM*) mediante sensores visuales-inerciales (VI-SLAM). Como su nombre lo indica, SLAM intenta construir un mapa del entorno en el que el agente XR se encuentra y localizarlo en el mismo de forma simultánea; usualmente, sin contar con información a priori sobre su pose ni el entorno en el que se encuentra. Hay múltiples maneras de implementar SLAM y VI-SLAM, pero los sistemas en los que nos concentraremos en este trabajo usan mediciones de la IMU a altas frecuencias (p. ej. 200 Hz) que miden el *"movimiento interno"* que el agente experimenta, también conocidas como mediciones *propioceptivas*, junto a muestras más lentas (p. ej. 20 Hz) de cámaras, usualmente un par de ellas, que dan información acerca de como el entorno está cambiando alrededor del agente cuando este se mueve. Estas son llamadas mediciones *exteroceptivas* y ayudan a corregir las mediciones ruidosas de la IMU.

El mapa que se forma en VI-SLAM suele crearse a partir de *puntos de interés* (*landmarks*) en la escena que fueron triangulados y detectados durante múltiples muestras como *esquinas* o *features*[7] en las imágenes de las cámaras. En el mejor de los casos el mapa formado en SLAM y las actualizaciones que este sufre pueden durar la corrida entera o

3 https://www.microsoft.com/en-us/mixed-reality/windows-mixed-reality
4 https://developers.google.com/ar
5 https://developer.apple.com/augmented-reality/arkit/
6 http://openslam.org

[7] *La terminología para definir puntos de interés es un poco confusa. Intentaremos esclarecer en capítulos posteriores el significado de términos como keypoint, landmark, feature, corner, etc. que algunas veces pueden parecer intercambiables en la literatura de visión por computadora.*

incluso ser guardados en almacenamiento persistente, y esto ayuda al dispositivo a entender cuando está viendo un lugar que ya vio anteriormente. Sin embargo, hay soluciones más sencillas que solo mantienen el mapa por una corta ventana de tiempo. A estos sistemas se los suele denominar de *odometría visual-inercial* (*VIO*), y suelen presentar mejor desempeño que soluciones completas de SLAM a costa de la precisión en la trayectoria estimada.

1.3 INTEGRACIÓN EN MONADO

Monado[8] es una implementación de código libre del estándar abierto *OpenXR* [6] que intenta unificar la forma en la que las aplicaciones interactúan con el hardware XR. Si bien entenderemos mejor estos términos más adelante en la Sección 4.1, basta con saber que OpenXR es una especificación desarrollada por el *Khronos Group*[9], un consorcio abierto sin fines de lucro integrado por distintos participantes de la industria. Monado fue desarrollado y es mantenido por *Collabora*[10], una consultora especializada en proyectos de código abierto, la cual además participa activamente en la especificación de OpenXR desde sus inicios. Este trabajo fue realizado en el marco de una pasantía con Collabora.

Monado, además de implementar la especificación de OpenXR, provee varias herramientas y funcionalidades para XR, además de controladores para dispositivos populares. El proyecto, antes de este trabajo, contaba con un puñado de métodos de tracking, pero la localización visual-inercial por SLAM o VIO era una característica faltante.

Este trabajo, entonces, resultó en la integración de tres sistemas de SLAM/VIO de código libre con Monado. En particular, los sistemas integrados fueron Kimera-VIO [7], ORB-SLAM3 [8] y Basalt [9]. Además, se adaptaron controladores para poder utilizar dispositivos con este tipo de tracking en Monado, incluyendo cascos de la plataforma Windows Mixed Reality. Estos, según el mejor entendimiento del autor, son ahora los primeros cascos comerciales capaces de ser localizados por SLAM/VIO en una plataforma basada completamente en código libre. Se puede ver un video demostrando el tracking funcionando en la URL referenciada al pie de página [11].

8 https://monado.dev
9 https://www.khronos.org
10 https://www.collabora.com
11 https://youtu.be/g1o2xADr5Fw

1.4 ESTRUCTURA DE LA TESIS

En la primera parte de este trabajo veremos algunos conceptos fundamentales para entender el funcionamiento de los sistemas de SLAM y VIO. Veremos representaciones usuales de la pose de un cuerpo en el espacio y de las transformaciones que se le pueden aplicar al mismo. Esto incluye conceptos como *ángulos Euler*, *cuaterniones* y algunas ideas básicas de *grupos de Lie*. En el capítulo siguiente desarrollaremos el método de *optimización no lineal* de *Gauss Newton*. Este tipo de optimización es el núcleo computacional de los sistemas modernos, ya que es utilizada para realizar la fusión de muestras de sensores maximizando la verosimilitud de un estimador estadístico. Además, se utiliza este tipo de optimización para resolver varios otros subproblemas como lo son la calibración de sensores, la proyección y reproyección de puntos hacia las cámaras y viceversa, la creación y el mantenimiento del mapa del entorno, entre otros.

La siguiente parte del trabajo nos enfocaremos en una implementación particular de uno de estos sistemas: Basalt. Esto permitirá introducir y ver muchos de los conceptos y algoritmia que se utilizan de forma concreta y contextualizadas. Veremos cómo Basalt utiliza algoritmos de *optical flow* para hacer el seguimiento de *features*, cómo decide cuáles cuadros serán *keyframes*, de qué forma utiliza y *preintegra* las muestras de la IMU, la gestión de las *landmarks* que realiza y finalmente cómo construye la función de *error* a optimizar.

En la tercera parte presentaremos los detalles de la integración con Monado. Veremos las decisiones tomadas a la hora de diseñar la interfaz fundamental de la integración con este tipo de sistemas. Hablaremos algunos problemas particulares que debieron resolverse para aplicarlos a XR. Además hablaremos de los controladores de dispositivos que se extendieron y los cuidados que hubo que tener para generar datos aceptables para SLAM.

Para cerrar el trabajo, en la cuarta parte presentamos algunos resultados cualitativos del rendimiento y la precisión de los sistemas integrados. Finalmente cerramos con algunas conclusiones y líneas de trabajo a considerar para el futuro. En el Apéndice A se encuentran listados varios enlaces a los repositorios, contribuciones y discusiones que surgieron como producto de este trabajo.

2

FUNDAMENTOS

2.1 TRANSFORMACIONES

Usualmente necesitaremos manipular y referirnos a transformaciones tridimensionales entre distintos sistemas de referencias como se muestra en la Figura 2.1. Estos sistemas surgen de las entidades que forman parte de nuestro proceso de optimización en SLAM. Algunos ejemplos de transformaciones en los que podríamos estar interesados son las que describen que transformación es necesaria realizar sobre la cámara izquierda para llegar a la cámara derecha de nuestro dispositivo, o la transformación que le ocurrió al agente localizado entre el instante anterior y el actual.

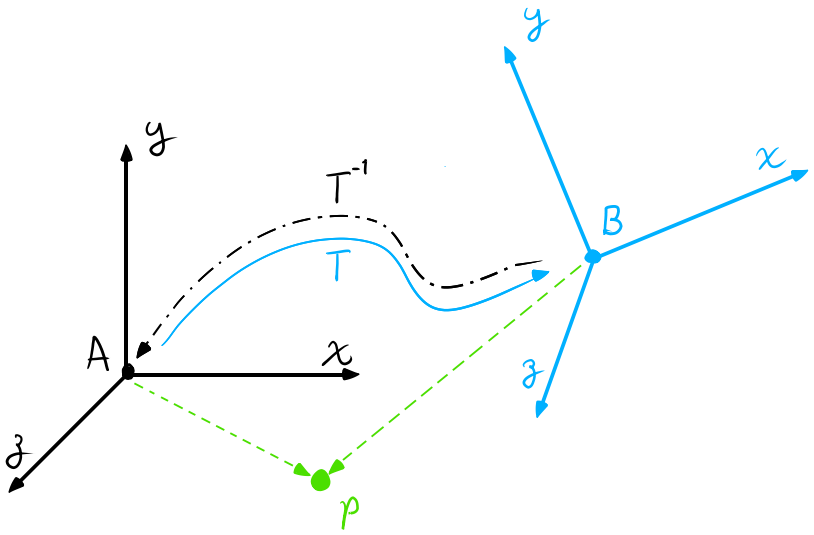


Figura 2.1: Intuición de lo que representa una transformación T que rota y traslada el sistema de referencia A hacia B. En general estas transformaciones tendrán una inversa T^{-1} que puede deshacer la acción original. Notar que un punto en el espacio $p \in \mathbb{R}^3$ tiene diferentes coordenadas dependiendo de que sistema se tome como referencia.

El tipo de transformación en el que estamos interesados son los *movimientos de cuerpo rígido*[1]. Estos pueden describirse mediante *traslaciones* y *rotaciones*. Además, también nos gustaría poder expresar la *ubicación* y *orientación* de las entidades en nuestro sistema. Estas dos características conforman la *pose* en el espacio de tal entidad. Es posible describir este concepto como una transformación aplicada sobre un sistema de

[1] *No profundizaremos en la definición formal de este concepto, pero intuitivamente, son transformaciones que al aplicarlas sobre un conjunto de puntos, preservan su volumen. Esto implica que preservan la norma y el producto cruz.*

referencia global fijo. Por ejemplo en la Figura 2.1 si pensamos en A como este origen global, tenemos entonces que T está describiendo la pose de B con respecto a A. A continuación, desarrollaremos algunas definiciones que nos ayudarán a describir la idea de transformación más formalmente, y utilizaremos este mismo concepto para identificar las poses de nuestras entidades.

Vale la pena aclarar que desarrollaremos una teoría suficientemente genérica como para aplicar en $n = 2$ y $n = 3$ dimensiones. En el caso bidimensional basta con tres variables independientes, también llamadas grados de libertad o *degrees of freedom* (*DoF*), para describir completamente una transformación: dos para la traslación y uno para la rotación. Por otro lado, en el caso tridimensional se necesitarán seis grados de libertad: tres y tres. Veremos al final de la sección que terminaremos usando representaciones sobre-determinadas, con más de tres o seis variables, ya que nos facilitarán su manipulación.

2.1.1 *Preliminares del álgebra lineal*

Comenzaremos construyendo sobre algunas ideas básicas del álgebra lineal.

Definición 2.1. *Una transformación lineal L entre dos espacios vectoriales V, W es una función $L : V \to W$ tal que:*

- $L(x + y) = L(x) + L(y) \quad \forall x, y \in V$
- $L(ax) = aL(x) \quad \forall a \in \mathbb{R}$

Propiedad 2.2 (Matriz de una transformación)**.** *Si $V \subseteq \mathbb{R}^n$ tiene base canónica $e_1, ..., e_n$ con $e_i \in \mathbb{R}^m$ tenemos que la matriz $A = [L(e_1), ..., L(e_n)] \in \mathbb{R}^{m \times n}$ cumple $Ax = L(x) \quad \forall x \in V$*

Propiedad 2.3 (Matrices cuadradas)**.** *El conjunto de matrices en $\mathbb{R}^{n \times n}$, con las operaciones de adición y multiplicación matricial usuales, forman un anillo[2] sobre el cuerpo de los reales. En particular, es un conjunto que se encuentra cerrado bajo estas operaciones.*

Propiedad 2.4. *Sean $a, b \in \mathbb{R}^n$, tenemos que $\|a + b\|^2 = \|a\|^2 + \|b\|^2 + 2a^T b$.*

Demostración. Desarrollemos:

$$\begin{aligned} \|a + b\|^2 &= (a_1 + b_1)^2 + \cdots + (a_n + b_n)^2 && (2.1)\\ &= (a_1^2 + b_1^2 + 2a_1 b_1) + ... + (a_n^2 + b_n^2 + 2a_n b_n) && (2.2)\\ &= \|a\|^2 + \|b\|^2 + 2\langle a, b\rangle && (2.3)\\ &= \|a\|^2 + \|b\|^2 + 2a^T b && (2.4) \end{aligned}$$

□

2 https://en.wikipedia.org/wiki/Ring_(mathematics)#Definition

2.1.2 *Representación de rotaciones*

Para las rotaciones, y orientaciones, existen múltiples representaciones válidas, cada una con sus ventajas y desventajas. Veremos algunas que han sido utilizadas en este trabajo.

2.1.2.1 *Ejes de rotación*

Cuando pensamos en la orientación de un cuerpo o en una rotación a aplicarle podemos pensar respecto a los ejes locales del mismo. Se suelen utilizar los términos *alabeo, cabeceo y guiñada* o *roll, pitch y yaw*, comunes en áreas como la aeronáutica, para hablar sobre rotaciones que ocurren en estos ejes. Ver Figura 2.2.

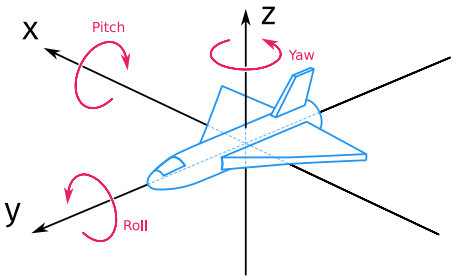


Figura 2.2: Rotaciones y ejes homónimos de alabeo (roll), cabeceo (pitch) y guiñada (yaw).

2.1.2.2 *Ángulos Euler*

La representación por ángulos Euler utiliza un vector $[x, y, z]^T \in \mathbb{R}^3$ en donde cada componente representa el ángulo de rotación que aplicar alrededor de tres ejes seleccionados por convención. Es decir, el vector describe tres rotaciones que aplicar como se ejemplifica en la Figura 2.3. Esta representación tiene la principal ventaja de resultar intuitiva cuando solo se necesita describir una rotación, pero se vuelve inconveniente en otros contextos que necesitaremos.

Uno de sus problemas es que existen docenas de convenciones válidas que varían en: la elección de los tres ejes, orden de los mismos, aplicación global o local en cada uno. Aunque en la práctica solo se utiliza un puñado de ellas, se puede volver fácilmente un punto de confusión[3]. El otro problema fundamental es el llamado *gimbal lock*[4] en donde la combinación de ciertas rotaciones puede causar la pérdida de un grado de libertad y describir nuevas rotaciones arbitrarias se hace imposible a partir de ese punto. Otros problemas de los ángulos Euler se detallan en Shoemake [10].

En general, evitaremos esta representación.

[3] *En la práctica, la manipulación de rotaciones y sistemas de coordenadas son fuentes usuales de muchos dolores de cabeza. Usualmente producto del uso no documentado de distintas convenciones.*

4 https://en.wikipedia.org/wiki/Gimbal_lock

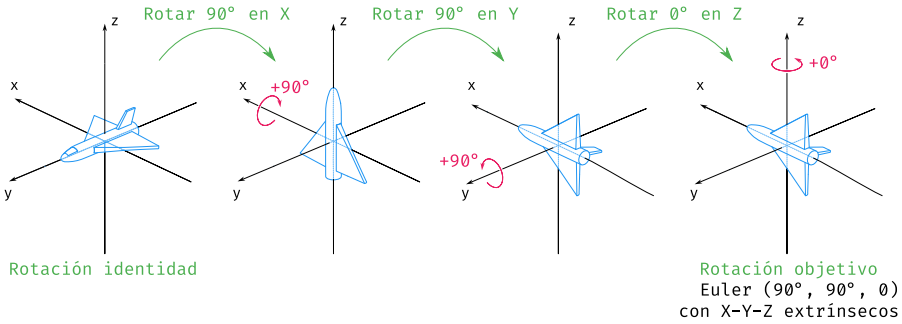


Figura 2.3: Ejemplo de una rotación objetivo representada en ángulos Euler. La construimos aplicando una secuencia de rotaciones sobre los ejes X, Y, y finalmente Z. Como utilizamos rotaciones alrededor de ejes fijos respecto a la orientación inicial de referencia, decimos que estamos usando la convención de ángulos Euler *X-Y-Z extrínseca*. Además notar que también fue necesario elegir cuales rotaciones se consideraron positivas en cada eje. En este caso utilizamos la llamada *regla de la mano derecha*.

2.1.2.3 *Ángulo axial*

Toda rotación puede representarse con un ángulo ω y un eje axial, representado por un vector unitario $a \in \mathbb{R}^3$, sobre el cual rotar ω radianes. La representación de ángulo axial de esta rotación queda definida por el vector $v = \omega a$; ver Figura 2.4. Tenemos entonces que $a = \frac{v}{\|v\|}$ y $\omega = \|v\|$.

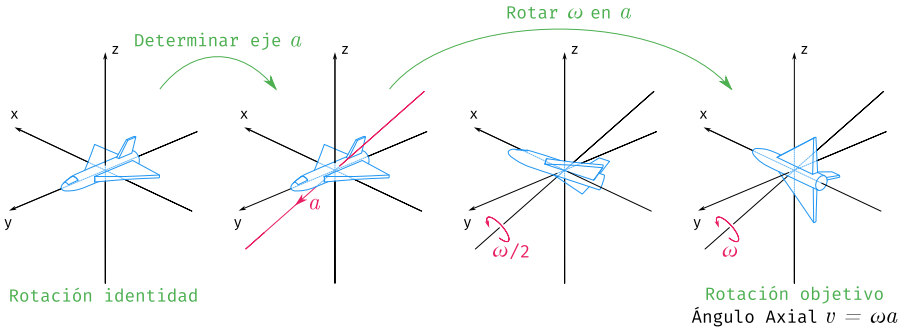


Figura 2.4: Representación de rotación por ángulo axial. Se puede pensar al eje a como si atravesara o espetara el cuerpo y luego se rotara en ω radianes a su alrededor.

Esta representación surge naturalmente del uso de giroscopios. Estos módulos, internamente se componen de tres sensores capaces de reportar cada uno la velocidad angular sobre un eje y se los ubica sobre tres ejes ortogonales. De esta forma, la velocidad que reportan queda determinada por la cantidad de rotación que ocurrió en cada eje durante el instante de la muestra capturada. Si se ven los valores de los tres sensores como componentes de un vector tridimensional, este coincide con la representación angular axial v de la rotación capturada.

Esta representación se utilizará para el almacenamiento de valores de velocidad angular provistos por muestras de giroscopio, ya que no es necesaria ningún tipo de conversión. Además, veremos más abajo que esta forma de describir rotaciones surge naturalmente en el estudio de otras representaciones.

2.1.2.4 *Cuaterniones unitarios*

Los cuaterniones están definidos sobre un álgebra $\mathbb{H}$. Son una construcción que extiende a $\mathbb{R}$ con tres números imaginarios i, j y k que satisfacen las siguientes propiedades:

$$i^2 = j^2 = k^2 = -1 \tag{2.5}$$
$$i = jk, \quad -i = kj \tag{2.6}$$
$$k = ij, \quad -k = ji \tag{2.7}$$
$$j = ki, \quad -j = ik \tag{2.8}$$

Un cuaternión $q \in \mathbb{H}$ tiene la forma

$$q = q_w + q_x i + q_y j + q_z k \quad \text{con } q_w, q_x, q_y, q_z \in \mathbb{R} \tag{2.9}$$

El producto y la adición se extienden de $\mathbb{R}$ naturalmente incluyendo las propiedades de las partes imaginarias listadas en Ecuaciones 2.5 a 2.8

Nos interesarán tres operadores de los cuaterniones:

- Conjugado: $q^* = q_w - (q_x i + q_y j + q_z k)$
- Norma: $\|q\| = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2} = \sqrt{qq^*}$
- Inverso: $q^{-1} = \frac{q^*}{\|q\|^2}$

Representaremos rotaciones y orientaciones con **cuaterniones unitarios**, es decir cuaterniones q tal que $\|q\| = 1$. Más aún cualquier cuaternión q' puede hacerse unitario dividiéndolo por su norma, o sea $q = q'/\|q'\|$ tiene norma 1.

Una rotación representada en **ángulos axiales** por el vector unitario $a = [x, y, z]^T$ y ángulo ω se representa con el siguiente cuaternión unitario q:

$$q = cos\frac{\omega}{2} + x\, sin\frac{\omega}{2}\, i + y\, sin\frac{\omega}{2}\, j + z\, sin\frac{\omega}{2}\, k \tag{2.10}$$

Más aún, todo cuaternión unitario puede expresarse de esa forma.

Para **rotar un vector** $v = [x, y, z]^T \in \mathbb{R}^3$ con una rotación representada por el cuaternión q basta con definir el cuaternión $p = 0 + xi + yj + zk$ y computar p' de la siguiente manera:

$$p' = qpq^{-1} = 0 + p'_x i + p'_y j + p'_z k \tag{2.11}$$

El cuaternión resultante p' tendrá parte real nula y el vector $v' = [p'_x, p'_y, p'_z]^T \in \mathbb{R}^3$ es el vector v rotado por q.

La **composición de rotaciones** queda bien definida por la multiplicación de cuaterniones. Es decir, dados los cuaterniones q y p, el cuaternión pq describe la rotación que se produce al aplicar primero q y luego p.

La **rotación identidad** o neutra, una rotación que "no rota", coincide con $1 \in \mathbb{R}$, es decir no tiene parte imaginaria, es $1 + 0i + 0j + 0k$.

El **inverso multiplicativo** de un cuaternión q^{-1} al componerlo con q, como es de esperarse, produce la identidad, o sea:

$$q^{-1}q = 1 \tag{2.12}$$

Muchas veces necesitaremos computar el **delta de rotación** que lleva de una orientación p a otra q. Esto es similar a lo que obtenemos cuando restamos vectores, así que sobrecargaremos el operador de resta de cuaterniones según el contexto de la siguiente manera:

$$q - p = p^{-1}q \tag{2.13}$$

Será también útil poder interpolar entre dos orientaciones p y q con un factor $t \in [0, 1]$ para conseguir orientaciones intermedias de p a q. La interpolación lineal *lerp* genera cuaterniones no unitarios, y por esto utilizaremos en su lugar la operación de **interpolación esférica** *slerp* presentada en Shoemake [10] y definida de la siguiente manera:

$$slerp(p, q, t) = p(p^{-1}q)^t \tag{2.14}$$

Notar que *no* definimos como elevar un cuaternión con un exponente $t \in \mathbb{R}$. Para esto necesitaríamos definir los llamados mapas *exponenciales* y *logarítimcos* que funcionan esencialmente como las operaciones usuales *exp* y *log* pero para cuaterniones. En este trabajo sólo haremos ese desarrollo sobre matrices de rotación en la sección siguiente. Mientras tanto invitamos al lector curioso en este desarrollo para cuaterniones a consultar el trabajo de Grassia [11] y el de Shoemake [10].

2.1.2.5 *Matrices de rotación*

Las rotaciones pueden representarse también como una matriz $R \in \mathbb{R}^{3x3}$ con ciertas restricciones.

Definición 2.5. *Sean $v, w \in \mathbb{R}^3$, decimos que v y w son ortonormales si $\|v\| = \|w\| = 1$ y son ortogonales, es decir $\langle v, w \rangle = 0$*

Sean $R^{(1)}, R^{(2)}, R^{(3)} \in \mathbb{R}^3$ las columnas de R.

Definición 2.6. *R se dice ortogonal u ortonormal si sus columnas son ortonormales de a pares, es decir, $\langle R^{(i)}, R^{(j)} \rangle = 0$ y $\|R^{(i)}\| = 1$ para $i \neq j;\ i, j \in \{1, 2, 3\}$.*

Las matrices ortonormales pueden tener determinante ± 1. Llamaremos **matrices de rotación** solo a las R tal que $det(R) = +1$ [5] [6]. El conjunto de estas matrices se denomina $SO(3)$ y veremos en la sección siguiente el por qué de este nombre. Estas matrices presentan propiedades agradables para ser manipuladas como rotaciones en el álgebra matricial usual:

- R es la transformación lineal que rota vectores $v \in \mathbb{R}^3$, es decir Rv es el vector v rotado por la rotación representada en R.
- La composición de rotaciones $R, S \in SO(3)$ queda representada con la multiplicación de matrices usual RS.
- La inversa de R es $R^{-1} = R^T$, más aún esta propiedad define a las matrices ortogonales.

[5] *El hecho de que el determinante sea $+1$, intuitivamente, quiere decir que la transformación lineal de R no escala los vectores en $\mathbb{R}^3$, o sea preserva volúmenes.*

[6] *Las transformaciones representadas por R con $det(R) = -1$ suelen llamarse rotaciones impropias o rotorreflexiones, ya que además de rotar, permiten la reflexión de vectores en $\mathbb{R}^3$.*

Demostración. Tenemos por la ortonormalidad de las columnas de R y definición de R^T que

$$R^T R = I_{3\times 3} \tag{2.15}$$

$$\Leftrightarrow \begin{bmatrix} \langle R^{(1)}, R^{(1)} \rangle & \langle R^{(1)}, R^{(2)} \rangle & \langle R^{(1)}, R^{(3)} \rangle \\ \langle R^{(2)}, R^{(1)} \rangle & \langle R^{(2)}, R^{(2)} \rangle & \langle R^{(2)}, R^{(3)} \rangle \\ \langle R^{(3)}, R^{(1)} \rangle & \langle R^{(3)}, R^{(2)} \rangle & \langle R^{(3)}, R^{(3)} \rangle \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.16}$$

$$\Leftrightarrow \|R^{(i)}\| = 1 \text{ y } \langle R^{(i)}, R^{(j)} \rangle = 0 \quad \forall i \neq j \in \{1,2,3\} \tag{2.17}$$

$$\Leftrightarrow \text{R es ortogonal} \tag{2.18}$$

□

2.1.3 *Representación de transformaciones*

Tener rotaciones expresadas como matrices, o equivalentemente como transformaciones lineales, prueba ser muy útil en la práctica, ya que podemos recurrir al arsenal de funcionalidad preexistente para estas estructuras. A continuación, desarrollaremos una serie de definiciones que nos permitirán extender esta idea de utilizar matrices para cubrir el concepto de transformación en su totalidad, con traslaciones incluidas.

2.1.3.1 *Grupos*

Vimos en la Sección 2.1.1 que $\mathbb{R}^{n\times n}$ forma un anillo con multiplicación y suma cerradas bajo $\mathbb{R}$. Además, mostramos que las matrices en $\mathbb{R}^{n\times n}$ corresponden a transformaciones lineales. Veremos ahora una serie de entidades que restringen a las transformaciones en $\mathbb{R}^{n\times n}$ con el objetivo de llegar a describir las transformaciones de cuerpo rígido sobre $\mathbb{R}^3$ que nos interesan.

Definición 2.7. *Un grupo es un conjunto G con una operación binaria $\circ : G \times G \to G$ tal que $\forall a, b, c \in G$:*

1. *Es cerrada: $a \circ b \in G$*
2. *Es asociativa: $(a \circ b) \circ c = a \circ (b \circ c)$*
3. *Tiene neutro: $\exists!\, e \in G : e \circ a = a \circ e = a$*
4. *Tiene inverso: $\exists a^{-1} \in G : a \circ a^{-1} = a^{-1} \circ a = e$*

Es posible empezar a intuir que *tanto las rotaciones como las transformaciones* que describimos al principio del capítulo *coinciden con esta idea de grupo*. "Mezclar" transformaciones debería dar como resultado otra transformación (cerrada) y no debería importar el orden de mezcla (asociativa). Debería existir una transformación neutra que no haga nada, y una transformación inversa que deshagan la transformación original. Si además pudiésemos describir estas transformaciones con matrices, seríamos capaces de recurrir a todas las herramientas preexistentes. Veamos como hacer eso ahora.

Definición 2.8. *El conjunto de matrices invertibles en $\mathbb{R}^{n\times n}$ con la operación de multiplicación matricial forman un grupo denominado Grupo Lineal General $GL(n)$. Es decir:*

$$GL(n) = \{A \in \mathbb{R}^{n\times n} : det(A) \neq 0\} \tag{2.19}$$

Definición 2.9. *El subconjunto de matrices de $GL(n)$ con determinante 1 se denomina el Grupo Lineal Especial $SL(n)$. Es decir:*

$$SL(n) = \{A \in GL(n) : det(A) = 1\} \tag{2.20}$$

Observación 2.10. *$A \in SL(n) \Rightarrow A^{-1} \in SL(n)$ ya que $det(A^{-1}) = \frac{1}{det(A)}$.*

Definición 2.11. *Un grupo G tiene una representación matricial si existe un homomorfismo inyectivo $F : G \to GL(n)$. Es decir, una función inyectiva para la cual la multiplicación matricial preserva la estructura del operador $\circ$. En símbolos, $\forall a, b \in G$:*

$$F(e) = I_{n\times n} \tag{2.21}$$
$$F(a \circ b) = F(a)F(b) \tag{2.22}$$

Definición 2.12. *El subconjunto de matrices ortogonales de $GL(n)$ se denomina el Grupo Ortogonal $O(n)$. Es decir:*

$$O(n) = \{R \in GL(n) : R^T R = I\} \tag{2.23}$$

Propiedad 2.13. *Una matriz ortogonal $R \in \mathbb{R}^{n\times n}$ preserva el producto interno.*

Demostración. $\forall x, y \in \mathbb{R}^n$ tenemos:

$$\langle Rx, Ry \rangle = (Rx)^T Ry = x^T R^T Ry = x^T y = \langle x, y \rangle \tag{2.24}$$

□

Propiedad 2.14. *Una matriz ortogonal $R \in O(n)$ tiene $det(R) = \pm 1$*

Demostración.

$$1 = det(I) = det(R^T R) = det(R^T) det(R) = det(R)^2 \tag{2.25}$$

$$\Leftrightarrow det(R) = \pm 1 \tag{2.26}$$

□

Definición 2.15. *El subconjunto de matrices ortogonales de $O(n)$ con determinante $+1$ se denomina el Grupo Ortogonal Especial $SO(n)$. Es decir:*

$$SO(n) = \{R \in O(n) : det(R) = +1\} = O(n) \cap SL(n) \tag{2.27}$$

Como vimos en la Sección 2.1.2.5, este grupo es el que define a las **matrices de rotación**. Y se corresponde a la representación matricial que presentamos allí. Continuemos ahora con las transformaciones.

2.1.3.2 *Transformaciones como grupos*

Definición 2.16. *Una transformación lineal afín $L : \mathbb{R}^n \to \mathbb{R}^n$ es tal que existe $B \in GL(n)$ y $b \in \mathbb{R}^n$ que determinan $L(x) = Bx + b$.*

Definición 2.17. *El grupo de tales transformaciones se denomina Grupo Afín de dimensión n $A(n)$.*

En general, una transformación afín $L(x) = Bx + b \in A(n)$ no es una transformación lineal a menos que $b = 0$. Introduciremos las llamadas *coordenadas homogéneas* para expresar transformaciones afines en $A(n)$ como transformaciones lineales en $GL(n+1)$. Extenderemos L de la siguiente manera:

$$L' : \mathbb{R}^{n+1} \to \mathbb{R}^{n+1} \tag{2.28}$$

$$L'\left(\begin{bmatrix} x \\ 1 \end{bmatrix}\right) = \begin{bmatrix} B & b \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} = \begin{bmatrix} Bx + b \\ 1 \end{bmatrix} = \begin{bmatrix} L(x) \\ 1 \end{bmatrix} \tag{2.29}$$

Notar que hay un isomorfismo entre L y L', ya que 0 y 1 son constantes y por esto diremos que son la misma transformación. La matriz $\begin{bmatrix} B & b \\ 0 & 1 \end{bmatrix}$ se dice una matriz afín y pertenece a $GL(n+1)$. Tenemos entonces que las matrices afines forman un subgrupo de $GL(n+1)$

Definición 2.18. *Una transformación euclídea $L : \mathbb{R}^n \to \mathbb{R}^n$ se define con una matriz ortogonal $R \in O(n)$ y un vector $t \in \mathbb{R}^n$ como $L(x) = Rx + t$.*

Definición 2.19. *El grupo de tales transformaciones se denomina Grupo Euclídeo $E(n)$ y es un subgrupo de $A(n)$. Es decir*

$$E(n) = \left\{ \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)} : R \in O(n), t \in \mathbb{R}^n \right\} \tag{2.30}$$

Definición 2.20. *El subconjunto de transformaciones euclídeas con $R \in SO(n)$ se denomina el Grupo Euclídeo Especial $SE(n)$, es decir:*

$$E(n) = \left\{ \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)} : R \in SO(n), t \in \mathbb{R}^n \right\} \tag{2.31}$$

Y es este el grupo que buscábamos, ya que $SE(3)$ es capaz de representar las transformaciones de cuerpo rígido en $\mathbb{R}^3$ que necesitábamos. Tenemos entonces que representaremos rotaciones con $R \in SO(3)$, osea matrices cuadradas 3×3; y representaremos transformaciones con $T \in SE(3)$, osea matrices cuadradas 4×4. Estos grupos también funcionan con dos dimensiones en $\mathbb{R}^2$ de la misma manera para $SO(2)$ y $SE(2)$. Finalmente, se pueden visualizar los conjuntos definidos en esta sección en la Figura 2.5.

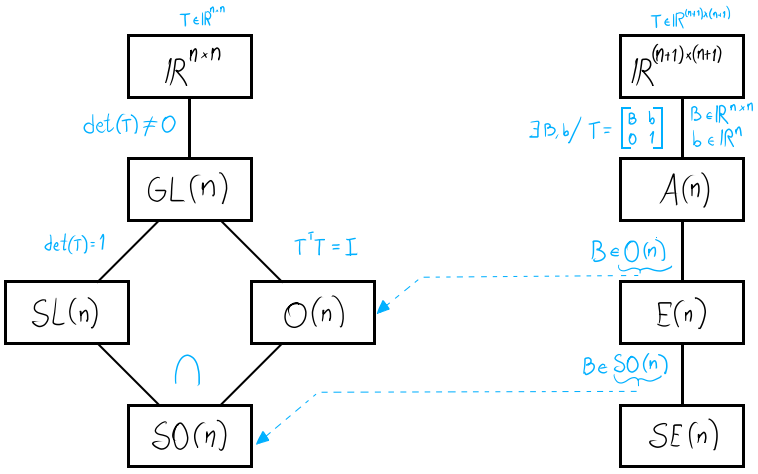


Figura 2.5: Diagramas de Hasse de los grupos desarrollados con el orden parcial $\subset$ (*"es subconjunto propio de"*).

2.1.4 *Representación de infinitesimales*

Ahora nos gustaría entender como tratar cambios infinitesimales en las rotaciones de $SO(3)$ y las transformaciones de $SE(3)$. Esto será necesario a la hora de aplicar algoritmos de optimización que dependen de las derivadas de estas operaciones.

2.1.4.1 *Matrices antisimétricas*

Definición 2.21 (Producto cruz)**.** *Dados $v, w \in \mathbb{R}^3$ el producto cruz de v y w es un vector ortogonal a ambos tal que:*

$$v \times w = \begin{bmatrix} v_y w_z - v_z w_y \\ v_z w_x - v_x w_z \\ v_x w_y - v_y w_x \end{bmatrix} \in \mathbb{R}^3 \tag{2.32}$$

Definición 2.22 (Matriz antismétrica)**.** *$R \in \mathbb{R}^{3\times3}$ se dice antisimétrica si $R = -R^T$*

Definición 2.23. *Definimos el operador $\hat{\cdot} : \mathbb{R}^3 \to \mathbb{R}^{3\times3}$ que devuelve la siguiente matriz antisimétrica:*

$$\hat{v} = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix} \in \mathbb{R}^{3\times3} \tag{2.33}$$

La construcción del operador $\hat{v}$ está diseñada para tener la siguiente propiedad:

Propiedad 2.24. $\hat{v}w = v \times w$

Además, por la definición de matriz antisimétrica es sencillo ver que:

Propiedad 2.25. *Toda matriz antisimétrica R está unívocamente definida por un vector $v \in \mathbb{R}^3$ tal que $R = \hat{v}$*

Definición 2.26. *El conjunto de todas las matrices antisimétricas en $\mathbb{R}^3$ se denomina Álgebra de Lie Ortogonal Especial $\mathfrak{so}(3)$, es decir:*

$$\mathfrak{so}(3) = \{\hat{v} \in \mathbb{R}^{3\times3} : v \in \mathbb{R}^3\} \tag{2.34}$$

Veremos a continuación cuál es su relación con $SO(3)$ y el por qué de su nombre.

2.1.4.2 *Rotaciones infinitesimales*

Consideremos una familia de rotaciones $R(t) \in SO(3)$ con $t \in \mathbb{R}$ que describen una rotación continua aplicada sobre un punto $X(0) \in \mathbb{R}^3$ hacia otro $X(t)$, es decir:

$$R(0) = I \tag{2.35}$$

$$X(t) = R(t)X(0) \tag{2.36}$$

Notación 2.27. *El parámetro t lo consideraremos implícito en algunas ocasiones, o sea $R = R(t)$. Además, utilizaremos la notación $\dot{R} = \frac{dR}{dt}$.*

Como $RR^T = I$ tenemos que

$$0 = \frac{d}{dt}I = \frac{d}{dt}(RR^T) = \dot{R}R^T + R\dot{R}^T \tag{2.37}$$

$$\Rightarrow \dot{R}R^T = -R\dot{R}^T \tag{2.38}$$

O sea que $\dot{R}R^T \in \mathfrak{so}(3)$. Por la Propiedad 2.25 tenemos que existe un vector único $v(t) \in \mathbb{R}^3$ tal que

$$\dot{R}(t)R(t)^T = \hat{v}(t) \Leftrightarrow \dot{R}(t) = \hat{v}(t)R(t) \tag{2.39}$$

Y como $R(0) = I$, entonces $\dot{R}(0) = \hat{v}(0)$. Por lo tanto, tenemos que la matriz antisimétrica $\hat{v}(0)$ nos da la aproximacíon de primer orden de una rotación:

$$R(dt) = R(0) + (dR)(0) = I + \hat{v}(0)dt \tag{2.40}$$

Notar que si pensamos a t en términos de tiempo, la Ecuación 2.40 deja ver a $\hat{v}$ como una matriz que describe la velocidad de la rotación.

Tenemos entonces que el efecto de una rotación infinitesimal en $SO(3)$ puede ser aproximado por matrices en $\mathfrak{so}(3)$. Es necesario mencionar que $SO(3)$ es lo que se denomina un *grupo de Lie* [7] mientras que $\mathfrak{so}(3)$ es su correspondiente *álgebra de Lie* [8]. No necesitaremos adentrarnos en estos conceptos, pero es común encontrarlos en la literatura y gran parte del desarrollo que estamos realizando está ligado a ellos.

Luego de haber presentado estas ideas fundamentales, a partir de aquí procederemos a dar una *vista general* de la definición de dos operadores, *exp* y *log* que nos permiten pasar del grupo al álgebra de Lie y viceversa. Más adelante, también veremos rápidamente como estos conceptos se traducen de forma muy similar para las transformaciones en $\mathbb{R}^3$. Al final del capítulo listaremos algunas referencias para el lector que quiera profundizar en los conceptos y las derivaciones.

[7] *Un grupo de Lie es una "variedad diferenciable" en el cual la operación del grupo, y su inversa, también son diferenciables. Intuitivamente, esto significa que la aplicación de rotaciones o transformaciones es "suave" o continua y esto nos permite trabajar con el concepto de límite y derivadas.*

[8] *Un grupo de Lie tiene un álgebra de Lie relacionada. Esta última es el "espacio tangente" en la identidad I del grupo (en particular, de la variedad). Intuitivamente, este puede pensarse como el espacio de todas las posibles velocidades alrededor de I. Esto es precisamente lo que desarrollamos al calcular $R(dt)$ en la Ecuación 2.40.*

Vimos en el desarrollo anterior que existe un vector $\hat{v}(t) \in \mathfrak{so}(3)$ que actúa como un término de velocidad rotacional. Si asumimos velocidad constante, es decir $\hat{v}$ constante, nos interesaría saber cual es la rotación total luego de rotar desde $R(0) = I$ con esta velocidad durante un tiempo t. En otras palabras, queremos computar $R(t)$ dado $\hat{v}$. Teniendo en cuenta el desarrollo anterior basta con plantear el siguiente sistema de ecuaciones diferenciales:

$$\begin{cases} \dot{R}(t) = \hat{v}R(t) \\ R(0) = I \end{cases} \tag{2.41}$$

Es posible ver que este sistema tiene como solución la siguiente expresión.

$$R(t) = exp(\hat{v}t) = \sum_{n=0}^{\infty} \frac{(\hat{v}t)^n}{n!} \tag{2.42}$$

Notar que los exponentes de la Ecuación 2.42 son respecto a la multiplicación matricial.

Esta rotación se corresponde con, dado $w = vt \in \mathbb{R}^3$, la rotación de $\omega = \|w\|$ radianes alrededor del eje dado por el vector unitario $a = w/\|w\|$. Dicho de otro modo, *la representación ángulo-axial $w = \omega a$ puede ser convertida en la matriz de rotación R equivalente mediante el operador exp con $R = exp(\hat{w})$*. Más aún, $\mathfrak{so}(3)$ contiene todos estos vectores ángulo-axiales.

El operador $exp : \mathfrak{so}(3) \to SO(3)$ se denomina *mapa o aplicación exponencial* y su inversa es el *mapa logarítmico* $log : SO(3) \to \mathfrak{so}(3)$. Este,

dado una matriz de rotación $R \in SO(3)$ devuelve $\hat{w} \in \mathfrak{so}(3)$ tal que $R = exp(\hat{w})$ computando w de la siguiente manera [12]:

$$\|w\| = cos^{-1}\left(\frac{traza(R) - 1}{2}\right) \tag{2.43}$$

$$w = \frac{\|w\|}{2sin(\|w\|)}\begin{bmatrix} R_{3,2} - R_{2,3} \\ R_{1,3} - R_{3,1} \\ R_{2,1} - R_{1,2} \end{bmatrix} \tag{2.44}$$

Notar que definimos *exp* en la Ecuación 2.42 como una suma infinita mientras que *log* pudo definirse con una expresión cerrada en las Ecuaciones 2.43 a 2.44. Existe la *fórmula de Rodrigues* que permite expresar también *exp* de forma cerrada:

$$exp(\hat{w}) = I + \frac{sin(\|w\|)}{\|w\|}\hat{w} + \frac{1 - cos(\|w\|)}{\|w\|^2}\hat{w}^2 \tag{2.45}$$

2.1.4.3 *Transformaciones infinitesimales*

Para las transformaciones en $SE(3)$ tenemos un desarrollo muy similar al de las rotaciones en $SO(3)$. Consideremos una familia de transformaciones $T(t) \in SE(3)$ compuestas por rotaciones $R(t) \in SO(3)$ y traslaciones $b(t) \in \mathbb{R}^3$ con $t \in \mathbb{R}$ que representan una transformación continua aplicada al punto $X(0) \in \mathbb{R}^3$ con $T(0) = I$. Es decir,

$$X(t) = T(t)X(0) \tag{2.46}$$

$$T(t) = \begin{bmatrix} R(t) & b(t) \\ 0 & 1 \end{bmatrix} \tag{2.47}$$

Considerando que esta vez la inversa de T es T^{-1} y no su transpuesta como era el caso con las rotaciones. Podemos aplicar un desarrollo similar al de la sección anterior y llegar a que:

$$\dot{T}T^{-1} = \begin{bmatrix} \dot{R}R^T & \dot{b} - \dot{R}R^T b \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{4\times4} \tag{2.48}$$

De vuelta, $\dot{R}R^T$ corresponde a alguna matriz antisimétrica $\hat{v} \in \mathfrak{so}(3)$. Definiendo un vector $y(t) = \dot{b}(t) - \hat{v}(t)b(t)$ podemos reescribir la Ecuación 2.48 e introducir el concepto de *giro o twist* $\hat{\xi}(t)$:

$$\hat{\xi}(t) = \dot{T}(t)T^{-1}(t) = \begin{bmatrix} \hat{v}(t) & y(t) \\ 0 & 0 \end{bmatrix} \tag{2.49}$$

La matriz de giro $\hat{\xi}$ pertenece al álgebra de Lie $\mathfrak{se}(3)$ del grupo de Lie $SE(3)$ y puede ser parametrizada por las coordenadas de giro $\xi \in \mathbb{R}^6$.

Para esto se utiliza un operador *hat* $\cdot^\wedge$ y su inversa *vee* $\cdot^\vee$ de la siguiente manera:

$$\hat{\xi} = \begin{bmatrix} y \\ v \end{bmatrix}^\wedge = \begin{bmatrix} \hat{v} & y \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{4\times 4} \tag{2.50}$$

$$\begin{bmatrix} \hat{v} & y \\ 0 & 0 \end{bmatrix}^\vee = \begin{bmatrix} y \\ v \end{bmatrix} = \xi \in \mathbb{R}^6 \tag{2.51}$$

Es decir, podemos codificar el cambio infinitesimal de una transformación en un vector de giro ξ de seis dimensiones en donde:

- Los últimos tres componentes dados por v son la representación **ángulo-axial** de la velocidad rotacional.
- Los primeros tres componentes dados por y describen el cambio de traslación teniendo en cuenta esta velocidad rotacional instantánea $\hat{v}$.

Finalmente, tenemos el mapa exponencial y mapa logarítmico entre $SE(3)$ y $\mathfrak{se}(3)$. Similar a $SO(3)$, cualquier transformación $T \in SE(3)$ va a poder ser representada por un vector de giro ξ con $T = exp(\hat{\xi})$.

Existen expresiones cerradas para ambos $exp : \mathfrak{se}(3) \to SE(3)$ y $log : SE(3) \to \mathfrak{se}(3)$.

Para $exp(\hat{\xi})$ con $\xi = [y, v]^T$ tenemos:

$$exp(\hat{\xi}) = \begin{bmatrix} exp(\hat{v}) & Jy \\ 0 & 1 \end{bmatrix} \tag{2.52}$$

Con J el llamado *jacobiano izquierdo* de $SO(3)$ que se puede computar con el ángulo ω de v, es decir con $\omega = \|v\|$ de esta manera [12]:

$$J = I + \frac{1 - cos(\omega)}{\omega^2}\hat{v} + \frac{\omega - sin(\omega)}{\omega^3}\hat{v}^2. \tag{2.53}$$

Para $log(T)$ con $T = \begin{bmatrix} R & b \\ 0 & 1 \end{bmatrix} \in SE(3)$ tenemos:

$$log(T) = \begin{bmatrix} log(R)^\vee \\ J^{-1}b \end{bmatrix}^\wedge \tag{2.54}$$

Con el jacobiano inverso J^{-1} dado por [12]:

$$J^{-1} = I - \frac{1}{2}\hat{v} + \left(\frac{1}{\omega^2} - \frac{1 + cos(\omega)}{2\omega sin(\omega)}\right)\hat{v}^2 \tag{2.55}$$

2.1.4.4 *Cierre y literatura recomendada*

Tenemos ahora una mirada suficientemente formal como para ser capaces de utilizar y comprender las herramientas usualmente utilizadas en sistemas que modelan rotaciones y transformaciones en dos y tres dimensiones por computadora. Nos tomamos el trabajo de entender varias formalizaciones para las rotaciones, ya que todas ellas aparecen de una u otra forma en el pipeline que se desarrolla en este trabajo. Las formalizaciones infinitesimales que hemos presentado en la última subsección, y sus representaciones en el álgebra de Lie, permiten además modelar los estados de las entidades que un algoritmo de SLAM necesita describir de una forma particularmente elegante. Esto es así porque permiten la aplicación de algoritmos de optimización, como los que veremos en el próximo capítulo, *sin restricciones*[9].

[9] *Representaciones de cuaterniones o matriciales requieren mantener los errores numéricos a raya mediante la constante renormalización de sus valores. Las representaciones dadas en las álgebras de Lie no tienen este problema.*

Respecto a la última sección de infinitesimales, se pueden encontrar algunos de estos conceptos explorados en mayor profundidad en Barfoot [13] cap. 7. Una excelente introducción a los grupos de Lie con una teoría reducida enfocada en aplicaciones de robótica es presentada en Solà y col. [14]. Finalmente, gran parte de las derivaciones de las expresiones que se vieron se encuentran detalladas en Eade [12].

2.2 OPTIMIZACIÓN POR CUADRADOS MÍNIMOS

Como se verá más adelante, muchos de los problemas fundamentales de SLAM son problemas de optimización. Querremos minimizar una *función de error* o *energía* no-lineal que integre las múltiples mediciones que contienen información conflictiva, ruido, outliers, y demás problemáticas propias de la toma de datos mediante sensores físicos. Se utiliza como algoritmo principal para la optimización de estos sistemas el algoritmo de Gauss-Newton, y es el que explicaremos en esta sección. Cabe aclarar que existen algoritmos ligeramente más sofisticados como *Levenberg-Marquardt* o el *método dogleg* que pueden aplicarse; no nos explayaremos en ellos en este trabajo, pero serán mencionados cuando sea pertinente.

2.2.1 *Cuadrados mínimos lineales*

Comencemos con el caso lineal que será necesario para luego aproximar el no-lineal.

Querremos encontrar algún punto $x \in \mathbb{R}^n$ que cumpla una serie de m restricciones **lineales** $f_i(x) = b_i$ con $f_i \in \mathbb{R}^n \to \mathbb{R}$ combinación lineal de los componentes de x; $b \in \mathbb{R}^m$ y con $i = 1, ..., m$. Además definimos $f(x) = [f_1(x) \dots f_m(x)]^T$ y al ser f_i combinaciones lineales de x, tenemos que existe una transformación lineal $A \in \mathbb{R}^{m \times n}$ tal que $f(x) = Ax$. De esta forma podemos replantear nuestras ecuaciones como el sistema $Ax = b$.

Será usual que $m > n$ y que A tenga columnas linealmente independientes así que solo consideraremos este caso. Al tener más ecuaciones que incógnitas tenemos un sistema *sobre-determinado* en el cual, para valores usuales de b, no tendremos solución. Buscaremos entonces el x que "mejor" cumpla con estas restricciones en algún sentido de la palabra. Utilizaremos el criterio de minimizar los *residuales* $r_i(x) = f_i(x) - b_i$ y definimos al vector residual como $r(x) = [r_1(x) \dots r_m(x)]^T$. Querremos encontrar un x que minimice la siguiente función de *error* o *energía* $E(x)$ basada en los residuales.

$$E(x) = \sum_{i=1}^{m} r_i(x)^2 = \|r(x)\|^2 \tag{2.56}$$

Notar que la definición de $E(x)$ coincide con la de la *norma euclídea* al cuadrado de $r(x)$. Encontrar x que minimice $E(x)$ con residuales lineales es el *problema de los cuadrados mínimos lineales*; a x le decimos *solución* de tal problema. Notar que $r_i = (Ax - b)_i$, o sea el componente i del vector $Ax - b \in \mathbb{R}^m$, y por ende $r(x) = Ax - b$.

$$E(x) = \sum_{i=1}^{m} (Ax - b)_i^2 = \|Ax - b\|^2 \tag{2.57}$$

Lo que queremos minimizar es entonces equivalente a la norma cuadrada del vector residual $Ax - b$. Introduciremos a continuación una serie de conceptos y teoremas necesarios para poder presentar una forma sucinta de minimizar esta norma.

Aceptaremos el siguiente teorema sin demostración (ver Shores [15], teoremas 2.7 y 3.7).

Teorema 2.28. *Son equivalentes:*

1. *Las columnas de A son linealmente independientes.*
2. *A es invertible.*
3. *$Ax = 0 \Leftrightarrow x = 0$.*

Otras definiciones y resultados que necesitaremos a continuación.

Definición 2.29. *La matriz de Gram de A es $A^TA \in \mathbb{R}^{n \times n}$.*

Teorema 2.30. *La matriz de Gram de A es invertible si y solo si A tiene columnas linealmente independientes.*

Demostración. Por Teorema 2.28 (3) A tiene columnas linealmente independientes si y solo si $x = 0 \Leftrightarrow Ax = 0$, entonces nos basta con ver que $Ax = 0 \Leftrightarrow A^TAx = 0$.

- $\Rightarrow$) Si $Ax = 0$ entonces $A^TAx = A^T(Ax) = A^T0 = 0$
- $\Leftarrow$) Si $A^TAx = 0$ entonces $0 = x^TA^TAx = (x^TA^T)Ax = (Ax)^T(Ax) = \|Ax\|^2 = 0$ y esta norma es 0 si y solo si $Ax = 0$

□

Definición 2.31. *Sea A con columnas linealmente independientes. La matriz pseudo-inversa de A es $A^\dagger = (A^TA)^{-1}A^T$*

Notar que $(A^TA)^{-1}$ existe por Teorema 2.30.

Con estas herramientas, estamos en posición de presentar la solución directa al problema de los cuadrados mínimos lineales.

Teorema 2.32. *$\hat{x} = A^\dagger b$ es **la** solución del problema de los cuadrados mínimos lineales. Es decir $\forall x \in \mathbb{R}^n : E(\hat{x}) = \|A\hat{x} - b\|^2 \leq \|Ax - b\|^2 = E(x)$.*

Demostración. Tenemos primero que

$$\hat{x} = A^\dagger b \tag{2.58}$$
$$\Leftrightarrow \hat{x} = (A^TA)^{-1}A^Tb \tag{2.59}$$
$$\Leftrightarrow A^TA\hat{x} = A^Tb \tag{2.60}$$
$$\Leftrightarrow A^T(A\hat{x} - b) = 0 \tag{2.61}$$

Las últimas dos ecuaciones reciben nombres particulares [10] [11]. Veamos ahora para un $x \in \mathbb{R}^n$ cualquiera.

$$\|Ax - b\|^2 = \|(Ax - A\hat{x}) + (A\hat{x} - b)\|^2 \tag{2.62}$$

Por Propiedad 2.4:

$$= \|A(x - \hat{x})\|^2 + \|A\hat{x} - b\|^2 + 2(A(x - \hat{x}))^T(A\hat{x} - b) \tag{2.63}$$
$$= \|A(x - \hat{x})\|^2 + \|A\hat{x} - b\|^2 + 2(x - \hat{x})^TA^T(A\hat{x} - b) \tag{2.64}$$

Por Ecuación 2.61:

$$= \|A(x - \hat{x})\|^2 + \|A\hat{x} - b\|^2 \tag{2.65}$$
$$\therefore \quad \|Ax - b\|^2 \geq \|A\hat{x} - b\|^2 \tag{2.66}$$

Más aún, esta solución es única, ya que la igualdad en la última ecuación solo se da si $\|A(x - \hat{x})\|^2 = 0$, y como A es no-singular, esto solo pasa cuando $x = \hat{x}$. □

[10] *La expresión $A^TAx = A^Tb$ suele ser referida como las "ecuaciones normales". La derivación de la solución de cuadrados mínimos presentada en este trabajo no es la única. De hecho, es común encarar el problema mediante cálculo diferencial. En ese contexto, las ecuaciones normales surgen naturalmente a la hora de optimizar $E(x)$ igualando su gradiente a 0. Para una demostración mediante cálculo referirse a Nocedal y Wright [16], cap. 10.*

[11] *La expresión $A^T(A\hat{x} - b) = 0$ es interesante, ya que muestra que las columnas de A son ortogonales al residual óptimo $A\hat{x} - b$ al ser su producto interno 0. Esto suele llamarse el "principio de ortogonalidad".*

2.2.2 *Cuadrados mínimos no lineales*

Generalizaremos el problema anterior para que también considere funciones no lineales. Reutilizaremos la notación introducida en la sección anterior.

En este caso permitimos ahora que las f_i sean no lineales, aunque requerimos que continúen siendo diferenciables. Además, ignoraremos el vector b haciendo que $f(x) = r(x)$. Es decir querremos que $f(x) = 0$ en lugar de $f(x) = b$. En el caso de necesitar la segunda restricción, podemos construir nuevas funciones $\tilde{f}_i(x) = f_i(x) - b_i$ y utilizar estas en su lugar. Nuestra función de error es entonces:

$$E(x) = \sum_{i=1}^{m} f_i(x)^2 = \|f(x)\|^2 \tag{2.67}$$

Encontrar x solución que minimice tal error es el *problema de los cuadrados mínimos no lineales*. En este caso, al no ser necesario que los residuales sean *funciones afines* (lineales más un escalar), no podemos dar una matriz A y utilizar ideas como las de la pseudo inversa $A^\dagger$. Más aún, en el caso lineal, si bien no lo mostramos, se cumple que la solución no solo es única, sino que es el único punto con gradiente $\nabla f(x) = 0$, es decir, el único punto optimizador, es un mínimo global. Para una demostración de esto, se utilizan argumentos de *convexidad* sobre $E(x)$; referirse a Nocedal y Wright [16], cap. 10. En el caso no lineal sin embargo, no tenemos ninguna de estas garantías, pueden existir infinitos máximos y mínimos globales, locales y cualquier combinación de estos.

Lo que se hace entonces es utilizar algoritmos heurísticos; en nuestro caso el algoritmo de *Gauss-Newton* que busca un mínimo de forma iterativa comenzando desde un punto que, si se encuentra lo suficientemente cerca a la solución, convergerá a ella. Como veremos más adelante, este requerimiento de proveer un punto inicial adecuado es muy razonable en sistemas de SLAM/VIO, ya que usualmente contaremos con dicha información.

Gauss-Newton genera la secuencia de puntos $x^{(1)}$, $x^{(2)}$, ... al ser un algoritmo iterativo. Es posible juzgar cada iteración k evaluando $E(x^{(k)}) = \|f(x^{(k)})\|^2$. Como es usual existen varios criterios de terminación apropiados: se alcanza un cierto número fijo de iteraciones k^{max}, el error del último iterando $E(x^{(k)})$ es suficientemente cercano a 0, o las últimas dos iteraciones tienen un error muy similar a los fines prácticos $E(x^{(k)}) \sim E(x^{(k+1)})$.

En cada iteración k el algoritmo cuenta con dos etapas. La etapa de **linealización** aproxima a $f(x)$ linealmente con su expansión de Taylor de primer orden centrada en el iterando actual $x^{(k)}$; llamaremos a esta aproximación $f^{(k)}(x)$. En la etapa de **actualización** utilizamos el hecho de que $f^{(k)}(x)$ es una función afín para encontrar una solución al problema de cuadrados mínimos *lineales* sobre ella. Esta solución será el valor de nuestro próximo iterando $x^{(k+1)}$ y, por ende, la estimación que damos como solución del problema no-lineal. Es decir, como sabemos que $x^{(k+1)}$ minimiza a $f^{(k)}$, esperaríamos que sea una buena aproximación al mínimo de f sabiendo que comenzamos el algoritmo cerca de su mínimo.

Linealización. Definimos $f^{(k)}(x)$ como la expansión de Taylor de primer orden de $f(x)$ centrada en $x^{(k)}$. Es decir:

$$f^{(k)}(x) = f(x^{(k)}) + A(x^{(k)})(x - x^{(k)}) \tag{2.68}$$

con $A : \mathbb{R}^n \to \mathbb{R}^{m \times n}$ la matriz jacobiana de f.

$$A = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \tag{2.69}$$

Notar que $f^{(k)}$ es afín; una transformación lineal $A(x^{(k)})$ aplicada sobre x, más un vector. Reacomodemos los términos para dejarlo explícito e introduzcamos la notación $A_k = A(x^{(k)})$.

$$f^{(k)}(x) = f(x^{(k)}) + A_k(x - x^{(k)}) \tag{2.70}$$
$$f^{(k)}(x) = A_k x - (A_k x^{(k)} - f(x^{(k)})) \tag{2.71}$$

Actualización. Teniendo en mente que queremos encontrar el mínimo de $\|f(x)\|^2$, lo aproximamos con el mínimo (y próximo iterando) $x^{(k+1)}$ de $\|f^{(k)}(x)\|^2$ pero como $f^{(k)}$ es afín, esto se reduce a buscar la solución del problema de cuadrados mínimos *lineales*. Sabemos por Teorema 2.32 la solución exacta para este caso con base en la matriz pseudo-inversa $A_k^\dagger = (A_k^T A_k)^{-1} A_k^T$.

$$\|f^{(k)}(x)\|^2 = \|A_k x - (A_k x^{(k)} - f(x^{(k)}))\|^2 \tag{2.72}$$

Se minimiza por Teorema 2.32 cuando

$$x = A_k^\dagger (A_k x^{(k)} - f(x^{(k)})) \tag{2.73}$$
$$= A_k^\dagger A_k x^{(k)} - A_k^\dagger f(x^{(k)}) \tag{2.74}$$
$$= x^{(k)} - A_k^\dagger f(x^{(k)}) \tag{2.75}$$

Entonces el algoritmo de Gauss-Newton queda definido de la siguiente manera:

Algoritmo 1 Gauss-Newton para cuadrados mínimos no lineales

Dada $f : \mathbb{R}^n \to \mathbb{R}^m$ diferenciable y un punto inicial $x^{(1)}$. Con $k = 1, 2, ..., k^{max}$.

1. Linealizar f alrededor de $x^{(k)}$ computando el jacobiano A_k:

$$f^{(k)}(x) = f(x^{(k)}) + A_k(x - x^{(k)}) \tag{2.76}$$

1. Actualizar el iterador a $x^{(k+1)}$ con el minimizador de $\|f^{(k)}(x)\|^2$ descripto en el Teorema 2.32. Se necesitará computar $A_k^\dagger$ con la inversa de la matriz de Gram de A_k:

$$x^{(k+1)} = x^{(k)} - A_k^\dagger f(x^{(k)}) = x^{(k)} - (A_k^T A_k)^{-1} A_k^T f(x^{(k)}) \tag{2.77}$$

Gauss-Newton y minimización lineal con la pseudo-inversa serán de gran utilidad para expresar numerosos tipos de problemas de optimización. Se utilizará para problemas como ajustar la posición de un punto de interés tridimensional que es observado por múltiples cámaras; desproyectar puntos 2D de cámaras hacia puntos 3D en la escena cuando los modelos de proyección no tienen expresiones cerradas para su inversa; incluso veremos que la optimización central de los sistemas de SLAM/VIO que combina todas las mediciones necesarias para generar las estimaciones de poses, usualmente llamada *bundle adjustment*,

suele expresarse y resolverse como una minimización de cuadrados no lineales. El desarrollo de esta sección está basada en Nocedal y Wright [16], cap. 1, 2 y 10 y Boyd [17], cap. 11, 12, 15 y 18; en esos trabajos se puede encontrar derivaciones alternativas e información de otras técnicas más sofisticadas que Gauss-Newton como Levenberg-Marquardt o el método dogleg.

Parte II

IMPLEMENTACIÓN DE UN SISTEMA

En esta parte contextualizaremos los distintos sistemas estudiados y profundizaremos en uno de ellos: *Basalt*. El estudio detallado de una implementación será particularmente esclarecedor al hilar sin generalizaciones multitud de métodos y algoritmos utilizados en contextos concretos y con objetivos bien definidos.

3

BASALT

3.1 PRELIMINARES

La decisión sobre cuál sistema de localización visual-inercial integrar con Monado no fue sencilla. Fue necesario descartar docenas de implementaciones e incluso así, no fue trivial entender si los sistemas elegidos finalmente resultaron los más adecuados para el contexto de XR. Cada implementación presentaba ventajas y desventajas, aunque es usual que las respectivas publicaciones se concentren en destacar solo las métricas favorables. Más aún, es necesario conocimiento experto para comprender cómo la elección de ciertos fundamentos teóricos, técnicas algorítmicas, decisiones arquitecturales o de tecnología afectan a la calidad del tracking dedicado a XR. Gran parte de este trabajo se basó en el estudio de los conceptos necesarios para poder tomar este tipo de decisiones.

A grandes rasgos y sin un orden en particular, las propiedades deseables que se consideraron a la hora de elegir sistemas fueron:

1. Versatilidad en la configuración sensores: Monado necesita soportar una gran variedad de dispositivos, es por esto que se prefirieron sistemas que soporten la mayor cantidad de combinaciones y tipos de sensores. En el mejor de los casos, Monado debería ser capaz de localizar desde cascos con cámaras estéreo y una IMU, hasta celulares con una única cámara y sin giroscopio.

2. Licencia permisiva: La licencia y filosofía de Monado da gran libertad al programador que lo utilice de hacer lo que desee con su código fuente. Esto incluye poder utilizarlo en proyectos en dónde se prefiere no distribuir dicho código. Licencias de código libre "virales" como la GPL [18] no permiten esto y enlazar Monado a sistemas con este tipo de licencias contagiaría a los proyectos que dependen de Monado quitándoles la libertad de decidir no publicar su código.

3. Desarrolladores activos: Estos sistemas suelen surgir de grupos de investigación y es muy común que se abandonen luego de que el trabajo sea publicado. Será necesario encontrar, dentro de lo posible, sistemas con un desarrollo activo, con mantenedores accesibles y que, en el mejor de los casos, acepten contribuciones a su proyecto.

4. Estabilidad, facilidad de instalación y buenas prácticas de desarrollo: Los sistemas de SLAM son muy complejos y es usual que

se optimicen para funcionar únicamente en los conjuntos de datos de prueba dejando de lado estas características que son fundamentales a la hora de querer brindar un sistema para ser utilizado por usuarios finales.

5. Rendimiento: El área de XR tiende a buscar la reducción del tamaño de los dispositivos para mejorar su ergonomía y practicidad. Un sistema de tracking debe ser capaz de operar en contextos con recursos acotados, debería utilizar poca memoria, poca energía y ser capaz de estimar poses a altas frecuencias y utilizando poca capacidad de cómputo.

6. Precisión: Por último y no menos importante, queremos que la precisión de la localización sea adecuada. El nivel de precisión requerido dependerá del tipo de aplicación. Es común que se necesite precisión submilimétrica en contextos de VR donde la simulación cubre completamente el campo de visión del usuario, y fallos en el tracking puedan inducir mareos. Mientras que para otros contextos como AR realizado por un celular, no es tan vital contar con ese nivel de exactitud.

3.1.1 *Sistemas integrados*

En este trabajo se integraron con Monado tres sistemas de código libre distintos, primero *Kimera-VIO* [7], luego *ORB-SLAM3* [8] y finalmente *Basalt* [9]. Los clones de estos sistemas modificados para la integración pueden verse en las Contribuciones 2 a 4.

Kimera-VIO[1] es una implementación desarrollada en el Instituto de Tecnología de Massachusetts (MIT) con una licencia permisiva BSD-2 [19]. Fue publicada originalmente en octubre de 2019 y su última actualización significativa[2], en dónde se introduce la posibilidad utilizar una única cámara, ocurrió en abril de 2021. Kimera resultó inicialmente atractivo por su licencia y su promesa de correr en tiempo real. Además de esto posee funcionalidades muy interesantes para XR como la reconstrucción y análisis semántico del entorno en el que el dispositivo se encuentra. Desgraciadamente, no logró ser adecuado mostrando grandes dificultades a la hora de correrlo en sensores distintos a los presentados en su publicación. Más aún el término "tiempo real" en el contexto de la publicación es ambiguo, ya que está lejos de ser capaz de ejecutarse a frecuencias adecuadas para XR y es más adecuado para uso en robots con frecuencias de menos de 10 Hz [7, Fig. 5].

ORB-SLAM3[3] es desarrollado por la Universidad de Zaragoza con una licencia viral GPL-3.0 [18]. Fue publicado inicialmente en julio de

1 https://github.com/MIT-SPARK/Kimera-VIO
2 https://github.com/MIT-SPARK/Kimera-VIO/pull/152
3 https://github.com/UZ-SLAMLab/ORB_SLAM3

2020 y su última actualización significativa[4] ocurrió en diciembre de 2021. ORB-SLAM3 es la tercera iteración de una línea de sistemas que dominan las tablas comparativas de SLAM desde hace unos cuantos años y es por esto que se implementó incluso aunque no posea una licencia permisiva. El sistema presenta varios métodos novedosos que mejoran la precisión del tracking mientras que muestra ser capaz de estimar poses a unos 20 Hz o 30 Hz [8, Tabla 6]. El sistema es el más versátil del campo permitiendo ser ejecutado en configuraciones con una cámara (monocular) o dos (estéreo), con o sin uso de la IMU e incluso con cámaras de profundidad. Además, soporta la reconstrucción de múltiples mapas y la capacidad de interconectarlos o incluso guardarlos en almacenamiento persistente para ser reutilizados en sesiones de tracking posteriores. Una desventaja es que la trayectoria que se da al construir el mapa es particularmente ruidosa y difícil de utilizar en XR. Se plantea como trabajo a futuro investigar más acerca de la funcionalidad de reutilización del mapa y como funciona el tracking con mapas ya construidos.

Finalmente, Basalt[5] es desarrollado por el Instituto Técnico de Múnich con una licencia permisiva BSD-3 [20]. Fue publicado originalmente en abril de 2019 y su última actualización significativa[6] [21] ocurrió en octubre de 2021. Basalt solo es capaz de correr en tiempo real su sistema de VIO necesitando de una pasada offline de su *"mapper"* para lograr un mapa consistente. A pesar de esto, mostró ser sorprendentemente preciso y tener un gran desempeño. En particular, es un sistema que puede correr fácilmente a 60 Hz consumiendo cuadros a mayores resoluciones que las probadas en ORB-SLAM3 y Kimera y duplicando las frecuencias de muestreo a 60 fps. Esta capacidad de soportar mayor cantidad de muestras aumenta significativamente la precisión de la trayectoria a pesar de que no sea un sistema de SLAM completo. Además de esto, el sistema es notablemente más sencillo de compilar al tener un buen manejo de sus dependencias, posee mejores prácticas de ingeniería de software y es en general mucho más estable logrando fácilmente sesiones de tracking ininterrumpidas a diferencia de los sistemas anteriores.

Los tres sistemas fueron integrados en Monado con distintos niveles de éxito, pueden verse demostraciones de cómo funcionan en los videos referenciados al pie de página [7] [8] [9]. Más adelante, en el Capítulo 6, se verán resultados y distintas métricas comparativas entre los sistemas.

Las terminologías y jerga del área de SLAM/VIO pueden resultar abrumadoras, pero se espera que introduciéndolas en el contexto de

4 https://github.com/UZ-SLAMLab/ORB_SLAM3/releases/tag/v1.0-release
5 https://gitlab.com/VladyslavUsenko/basalt
6 https://gitlab.com/VladyslavUsenko/basalt/-/commit/24325f2a
7 Kimera-VIO con Monado: https://youtu.be/gxu3Ve8VCnI
8 ORB-SLAM3 con Monado: https://youtu.be/kJwWY973b10
9 Basalt con Monado: https://youtu.be/ajuqQ7E1MFw

una implementación concreta se puedan entender mejor los problemas que los sistemas, en general, tienen que resolver. Por todas las razones mencionadas anteriormente, Basalt es actualmente el sistema de preferencia para ser utilizado con Monado y, si bien se estudió el código fuentes de los tres sistemas, en esta parte del trabajo que sigue a continuación *nos vamos a enfocar en profundizar en la implementación de Basalt*.

3.1.2 *Problemáticas de un sistema*

Un problema central en este tipo de sistemas es el de poder generar un mapa y una trayectoria que sean *globalmente consistentes*. Con esto nos referimos a que nuevas mediciones tengan en cuenta todas las mediciones anteriores en el sistema. Una forma ingenua de encarar esto, sería realizando *bundle adjustment*[10] sobre todas las imágenes capturadas a lo largo de una corrida, integrando además las mediciones provenientes de la IMU. Desafortunadamente, este método excede rápidamente cualquier capacidad de cómputo de la que dispongamos, y aún más teniendo en cuenta que nuestro objetivo es localizar en tiempo real al dispositivo de XR.

Por esta razón, es usual recurrir a distintas formas de reducir la complejidad del problema. Para realizar *odometría visual-inercial* (*VIO*), es común que se ejecute la función de optimización sobre una *ventana local* de cuadros y muestras recientemente capturadas, ignorando muestras históricas y acumulando error en las estimaciones a lo largo del tiempo. Además, esta mirada tiene la desventaja adicional de que una porción significativa de los fotogramas capturados podrían tener posiciones similares que no añadirían demasiada información al estimador, o incluso que algunos fotogramas puedan ser de baja calidad por contener *motion blur* u otro tipo de anomalías. Por otro lado, soluciones que intentan hacer *mapeo visual-inercial* realizan el bundle adjustment sin utilizar todas las imágenes capturadas, sino que se limitan a la utilización de algunos fotogramas clave, o *keyframes* elegidos mediante criterios que priorizan cuadros nítidos y con distancias (*baselines*) prudenciales entre ellos.

Como las muestras de IMU vienen a mayor frecuencia que las de la cámara, es común que estas se *preintegren* de forma tal de combinar muestras simultáneas entre dos keyframes en una única entrada del optimizador. Sin embargo, un problema en el que esta integración incurre, es que las mediciones de las IMU son altamente ruidosas, y acumularlas durante tiempos prolongados acumula también cantidades significativas de error. Este factor nos limita el tiempo que puede transcurrir entre dos keyframes; como ejemplo en Mur-Artal y Tardos [23] se habla de keyframes que no pueden tener más de medio segundo entre sí. Además, tener keyframes a muy bajas frecuencias afecta la calidad de las estimaciones de velocidad y *biases*; estos últimos son offsets de medición inherentemente variables de los acelerómetros y

[10] *Introdujimos el término bundle adjustment (BA) en la Sección 2.2.2 en el contexto de cuadrados mínimos. Este se refiere al refinamiento simultáneo de un conjunto de poses de cámara, o vistas, y puntos 3D en el mapa que han sido observados por estas vistas. Así, se intenta reducir el llamado "error de reproyección" del conjunto actualizando tanto las poses de las cámaras como la posición de los puntos observados. Este error hace referencia a la distancia entre las posiciones de los puntos y en dónde las vistas esperan que estos se encuentren* [22].

giroscopios a los que es necesario reestimar de forma constante para compensar por ellos en la medición final.

3.1.3 *Propuesta de Basalt*

La novedad de Basalt [9] es que formula el mapeo visual-inercial como un problema de bundle adjustment y utiliza, de una forma específica, todas mediciones visuales e inerciales a altas frecuencias. Usa un *grafo de factores*[11] [12] de forma similar a otros sistemas, también llamado *grafo de poses* en este contexto por contener poses a estimar como nodos. En lugar de utilizar todos los fotogramas se propone realizar la optimización en dos capas. La capa de VIO, emplea un sistema de odometría visual-inercial, que ya de por sí supera a otros sistemas del mismo tipo, proveyendo estimaciones de movimiento a la misma frecuencia que el sensor de la cámara provee imágenes. Luego, se seleccionan keyframes y se agregan *factores no-lineales* entre estos que estiman la diferencia de posición relativa. Estos dos factores, keyframes y poses relativas, se utilizan en la capa de bundle-adjustment global.

La capa de VIO, detecta *features*[13] que son rápidas y buenas para seguir durante varios cuadros (esto es el *optical flow* que veremos en la sección Sección 3.2.1), mientras que en la capa de mapeo se usan features adecuadas que son indiferentes a las condiciones de luz o al punto de vista de la cámara[14]. De esta forma tenemos un sistema que es capaz de utilizar las mediciones a alta frecuencias de los sensores y al mismo tiempo tiene la capacidad de detectar cuando se está en ubicaciones ya visitadas, obteniendo así un mapa que es globalmente consistente. Además, el problema de optimización se reduce, ya que a diferencia de otros sistemas, no es necesario estimar velocidades ni biases (de la IMU).

[11] *Los grafos de factores son una muy buena forma de representar problemas con muchas variables aleatorias interdependientes y muestras que las relacionan (factores). En general, el uso de estos grafos trae beneficios computacionales interesantes y son de gran importancia para el área de SLAM. Sin embargo, no nos adentraremos demasiado en el tema en este trabajo y dirigimos al lector interesado a Dellaert y Kaess [24].*

[13] *Las features son puntos de interés relevados en una imagen. Son estos los puntos que triangularemos en el bundle adjustment.*

[14] *Esto es fundamental para entender cuándo el dispositivo está visitando un lugar por el que ya pasó. Esto se denomina "loop closing".*

3.2 IMPLEMENTACIÓN

A continuación describiremos la arquitectura e implementación de Basalt de una manera más detallada. Estas secciones surgen directamente de la lectura del código fuente del sistema e intentan proveer detalles más bien pragmáticos que se encuentran en el mismo, pero que pueden quedar escondidos en las publicaciones de más alto nivel que presentan estos sistemas. A su vez, se toman ciertas licencias literarias que deberían ayudar al entendimiento y que no son posibles a la hora de escribir código.

Cómo vimos anteriormente, el funcionamiento de Basalt se divide en dos etapas. La primera etapa de odometría visual-inercial (VIO), en el cual se emplea un sistema de VIO que supera a sistemas equivalentes

12 Artículo introductorio a los grafos de factores: https://gtsam.org/2020/06/01/factor-graphs.html

de vanguardia. La segunda etapa de mapeo visual-inercial (VIM), toma keyframes producidos por la capa de VIO y ejecuta un algoritmo de bundle adjustment para obtener un mapa global consistente. Estas dos capas son completamente independientes. En una corrida usual, se ejecuta inicialmente el sistema de VIO y es este el que decide y almacena persistentemente qué cuadros y con qué información el sistema de VIM, de ejecutarse, debería utilizar al realizar el proceso de bundle adjustment.

Esto significa que, por defecto, no contamos con la capacidad de utilizar el VIM en tiempo real para XR, solo el VIO. Por ende solo este fue integrado con Monado. Se plantea como trabajo a futuro la paralelización del VIM en un hilo separado para poder correrlo en tiempo real; ver Discusión 18. Exploraremos entonces, en esta parte del trabajo, los componentes fundamentales de la capa de VIO: *optical flow*, *bundle adjustment visual-inercial* y finalmente el proceso de *optimización y de marginalización parcial*.

3.2.1 *Optical flow*

El módulo de VIO toma dos tipos de entrada, una de ellas son las muestras raw de la IMU; y la otra, contra intuitivamente, no son las imágenes raw provenientes de las cámaras, sino que son los *keypoints* resultantes de ellas. Estos son la posición en dos dimensiones sobre el plano de la imagen de las *features* detectadas. Las features a su vez son la representación de los puntos de interés o *landmarks* de la escena tridimensional proyectados sobre las imágenes. El proceso de detectar features, computar su transformación entre distintos cuadros, y producir los keypoints de entrada para el módulo de VIO, está a cargo del módulo de *optical flow* (o *flujo óptico*). Cabe aclarar que optical flow es el nombre que recibe tanto el campo vectorial que representa el movimiento aparente de puntos entre dos imágenes, como el proceso de estimarlo. Este puede ser denso, si se considera el flujo de todos los píxeles, o no (*sparse*) si solo se computa el flujo de algunos keypoints como es el caso que veremos.

El módulo de optical flow corre en un hilo separado y es por donde las muestras del par de cámaras estéreo ingresan al pipeline de Basalt. Inicialmente se genera una representación piramidal de las imágenes, o también llamada de *mipmaps*, esta es una forma tradicional [25] de almacenar una imagen en memoria junto a versiones reescaladas de la misma como se ve en la Figura 3.1. Los mipmaps tienen múltiples utilidades en computación gráfica (p. ej. *filtrado trilineal*, *LODs*, reducción de *patrones moiré*) pero en el caso de Basalt serán utilizados para darle robustez al algoritmo de seguimiento de features o *feature tracking*.

Posteriormente se realiza la detección de features nuevas sobre las imágenes utilizando el algoritmo *FAST* [27] para detección de esquinas, o puntos sobresalientes de la imagen en general (keypoints), im-

Figura 3.1: Representación piramidal (mipmaps) de un cuadro del conjunto de datos estándar EuRoC [26].

plementado sobre *OpenCV*[15]. Resulta importante aclarar que Basalt es uno de los sistemas que menos depende de OpenCV, siendo `cv::FAST` el único algoritmo de la biblioteca en uso durante una ejecución usual. El proyecto tiende a reimplementar muchas de las técnicas y algoritmia de forma especializada y, como veremos en otros módulos, otras tareas razonablemente complejas como la optimización de grafos de poses se implementan también dentro del proyecto y sin recurrir a bibliotecas externas. Esta es una de las varias razones por las que el sistema logra tan buen rendimiento, ya que estas bibliotecas suelen necesitar de campos y comprobaciones generales del problema que intenta solucionar, mientras que Basalt puede prescindir de todas las que no apliquen a VIO. Aunque, también es cierto que estas reimplementaciones añaden complejidad al sistema.

[15] *OpenCV es una de las bibliotecas de visión por computadora más populares y utilizadas en este tipo de sistemas. Combina múltiples algoritmos y presenta una licencia permisiva Apache 2 [28] (prev. BSD-3 [20]).*

Siguiendo con la detección de features, una heurística particular de Basalt es la división del cuadro completo en celdas de, por defecto, 50 por 50 píxeles en donde se detectan los nuevos puntos de interés. Por celda solo se conserva la feature de mejor calidad o con mejor *respuesta* (*response*), aunque se puede configurar para detectar más de una. Siempre que la celda tenga alguna localizada de cuadros anteriores, no se intenta detectar nuevas. Esto contrasta con sistemas como Kimera-VIO que corren la detección FAST sobre el cuadro entero y evitan la redetección mediante el uso de *máscaras* que le instruyen al algoritmo a obviar esas secciones. Desafortunadamente la construcción de tales máscaras suele ser costosa y la heurística de Basalt, a pesar de desperdiciar espacio por no permitir la detección de nuevas características entre celdas, es más eficiente, ya que en situaciones comunes se logran detectar una cantidad razonable de features sin problemas. Esta detección se realiza únicamente sobre la primera cámara, usualmente la izquierda, mientras que en la otra cámara se reutiliza el método de seguimiento de keypoints que se describe a continuación.

En cada instante de tiempo que entran un nuevo par de imágenes se tiene acceso a toda la información recolectada del instante anterior,

en particular a sus keypoints. Una suposición razonable es que las imágenes correspondientes a este nuevo instante van a compartir muchos de los keypoints con las imágenes anteriores y en posiciones similares. Con esa suposición Basalt logra ahorrarse tener que volver a detectar features de la imagen con FAST y en cambio el problema se transforma en, dado una imagen anterior (inicial), sus keypoints y una imagen nueva (objetivo), estimar donde ocurren esos mismos keypoints en la imagen nueva. Para esto, por cada keypoint anterior, se genera un parche Ω alrededor de su ubicación de, por defecto, 52 puntos como se ve en el ejemplo de la Figura 3.2.

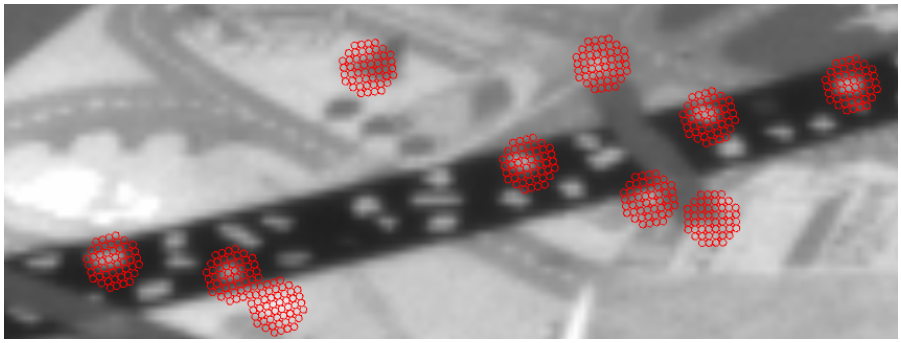


Figura 3.2: Ejemplos de los parches de 52 puntos considerados para computar el optical flow de distintos keypoints de una imagen.

Considerando entonces que este parche debería estar en la imagen nueva en coordenadas cercanas a las del keypoint anterior, queremos encontrar la transformación $\mathbf{T} \in SE(2)$ que le ocurrió al parche, y por ende al nuevo keypoint que se encontraría en el centro de este nuevo parche. Basalt emplea entonces optimización por cuadrados mínimos mediante el algoritmo iterativo de Gauss-Newton para encontrar $\mathbf{T}$ utilizando un residual r con:

$$r_i = \frac{I_{t+1}(\mathbf{T}\mathbf{x}_i)}{\overline{I_{t+1}}} - \frac{I_t(\mathbf{x}_i)}{\overline{I_t}} \quad \forall \mathbf{x}_i \in \Omega$$

Y aquí siendo $I_t(\mathbf{x})$ la intensidad de la imagen anterior en el pixel ubicado en las coordenadas $\mathbf{x}$, análogamente $I_{t+1}(\mathbf{x})$ para la imagen objetivo; y siendo $\overline{I_t}$ la intensidad media del parche Ω en la imagen inicial, análogamente $\overline{I_{t+1}}$ para la imagen objetivo y el parche transformado $\mathbf{T}\Omega$. Notar que al normalizar las intensidades obtenemos un valor que es invariante incluso ante cambios de iluminación.

Los detalles del cálculo de gradientes y jacobianos están basados en el método de Lucas y Kanade [29] para tracking de features (*KLT*). El uso adicional de mipmaps sobre KLT fue originalmente expuesto en Bouguet [30].

Para asegurar que la estimación fue exitosa se invierte el problema y se intenta trackear desde la imagen nueva hacia la inicial y, si el resultado está muy alejado de la posición inicial, el nuevo keypoint se

considera inválido (un *outlier*) y se lo descarta. Otro detalle a aclarar es que, recordando que la detección costosa de features con FAST solo ocurría en las imágenes de una de las cámaras, es posible ahora entender que las features en la segunda cámara pueden ser "detectadas" con este método menos costoso. Es decir, simplemente se computa el optical flow desde la imagen de la cámara izquierda a la de la cámara derecha en el mismo instante de tiempo.

Finalmente, el último de los pasos que ocurre cuando el módulo de optical flow procesa un cuadro, es el de filtrado de keypoints, en el cual se desproyectan los keypoints a posiciones en la escena tridimensional y en caso de que el error de reproyección supere cierto umbral, estos keypoints serán descartados por considerarse outliers.

3.2.2 *Procesamiento de muestras*

En un hilo separado al módulo de optical flow, corre el estimador de VIO encargado de realizar el bundle adjustment sobre los cuadros y muestras de la IMU recientes para estimar la pose. Este toma como entrada las muestras de la IMU junto a los keypoints 2D detectados para cada imagen, o sea la salida del módulo de optical flow. Este módulo es el que efectivamente realizará la integración y optimización con toda la información recibida y producirá como salida en una cola, la estimación de los estados del agente a localizar.

3.2.2.1 *Inicialización y preintegración*

Para comenzar, el hilo de procesamiento de este módulo espera a que la primera muestra de la IMU arribe. Estas muestras son recibidas de forma raw y antes de tratarlas, Basalt utiliza parámetros de calibración estáticos provistos por archivos de configuración para corregirlas. La corrección ocurre con base al modelo de calibración de IMU explicado en Barfoot [13] secc. 6.4.4 sin considerar todavía los parámetros de los procesos aleatorios detallados en la misma. Una particularidad de Basalt, es que el acelerómetro es utilizado como origen del agente localizado y, fundándose en esto, se fija su orientación. Es decir, no se aplica ningún tipo de corrección de orientación al calibrar las muestras del acelerómetro. Esto hace que la matriz de alineamiento para el acelerómetro tenga ceros en su triángulo superior (ver Schubert y col. [31] secc. IV.B y Discusión 20).

Luego de recibir esta primer muestra de la IMU se comienza la ejecución del bucle principal, el cual espera indefinidamente por resultados encolados por el módulo de optical flow para realizar una iteración. El primer par de muestras de cámaras junto a la primera muestra de la IMU posterior al par estéreo son utilizados para inicializar el estado del agente en el primer cuadro. Esto es ya que a cada cuadro se le asigna un estado que se compone de la posición, orientación, veloci-

dad y biases del giroscopio y acelerómetro que se estimaron para tal cuadro. Notar que en Basalt, hablar de cuadros es equivalente a hablar de instantes de tiempo, ya que los únicos puntos en el tiempo para los cuales el sistema produce una pose estimada son las timestamps del par estéreo de imágenes recibidas.

Para inicializar el primer estado se toman varias suposiciones. En particular, se asume que el dispositivo comienza en la posición $(0, 0, 0)$ y sin aceleración ni velocidad, esto permite utilizar el vector de aceleración reportado por la muestra del acelerómetro como el vector de gravedad y computar así la inclinación del agente. Notar que esta inclinación no es capaz de informar la orientación de forma completa al no poder contemplar uno de los ejes de rotación del cuerpo. Por esta razón es recomendable iniciar la corrida con el agente rotado con su eje vertical paralelo al vector gravedad, esto hará que Basalt compute la orientación identidad. Tal rotación suele corresponder con la posición de reposo pensada por el fabricante, o al menos este ha sido el caso en los dispositivos utilizados en este trabajo. Además, es conveniente posicionar el agente mirando hacia "adelante" ajustando el *yaw* [16], como el usuario considere apropiado según su entorno.

En instantes posteriores, o sea al recibir nuevas imágenes, se realiza la llamada preintegración de muestras consecutivas de la IMU. Considerando que estas muestras arriban a mayores frecuencias que las de las cámaras, preintegrarlas es un proceso que intenta resumir las muestras entre los cuadros a una única pseudo-muestra que sucede en los mismos instantes de tiempo que los cuadros como se ve en el ejemplo de la Figura 3.3.

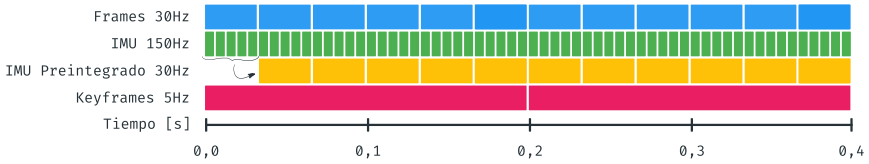


Figura 3.3: Frecuencia de distintos eventos para un ejemplo con cámaras a 30 fps y muestras de la IMU a 150 Hz.

El proceso de preintegración es el siguiente. Dado el cuadro previo i con timestamp t_i y el cuadro posterior j con timestamp t_j, se intenta computar una pseudo-muestra $\Delta\mathbf{s} = (\Delta\mathbf{R}, \Delta\mathbf{v}, \Delta\mathbf{p}) \in SO(3) \times \mathbb{R}^3 \times \mathbb{R}^3$ que representa cambios de orientación, velocidad y posición respectivamente según las mediciones de la IMU que ocurrieron desde t_i hasta t_j. Para cada timestamp t de la IMU tal que $t_i < t \leq t_j$ tenemos la muestra de aceleración lineal $\mathbf{a}_t$ y de velocidad angular ω_t. Definimos

16 Los términos *roll*, *pitch* y *yaw* provenientes de la aeronáutica son muy utilizados para hablar de la orientación de un cuerpo: https://en.wikipedia.org/wiki/Aircraft_principal_axes

entonces de forma recursiva la pseudo-muestra $\Delta\mathbf{s}$ de la siguiente manera:

$$(\Delta\mathbf{R}_{t_i}, \Delta\mathbf{v}_{t_i}, \Delta\mathbf{p}_{t_i}) := (\mathbf{I}, \mathbf{0}, \mathbf{0}) \tag{3.1}$$
$$\Delta\mathbf{R}_{t+1} := \Delta\mathbf{R}_t exp(\omega_{t+1}\Delta t) \tag{3.2}$$
$$\Delta\mathbf{v}_{t+1} := \Delta\mathbf{v}_t + \Delta\mathbf{R}_t\mathbf{a}_{t+1}\Delta t \tag{3.3}$$
$$\Delta\mathbf{p}_{t+1} := \Delta\mathbf{p}_t + \Delta\mathbf{v}_t\Delta t \tag{3.4}$$
$$\Delta\mathbf{s}_t := (\Delta\mathbf{R}_t, \Delta\mathbf{v}_t, \Delta\mathbf{p}_t) \tag{3.5}$$
$$\Delta\mathbf{s} := \Delta\mathbf{s}_{t_j} \tag{3.6}$$

Es destacable mencionar que este tipo de preintegración es también utilizado por los otros sistemas estudiados Kimera y ORB-SLAM3. La ventaja que presenta es que sus características son bien conocidas gracias al trabajo de Forster y col. [32] y las expresiones necesarias para el cómputo de residuales, como sus jacobianos, son cerradas y fueron derivadas de forma ejemplar en dicho trabajo.

Entonces, con esta muestra preintegrada junto a los datos del nuevo cuadro (sus keypoints), se procede a la etapa de `measure` del módulo de VIO. Aquí lo primero que se hace es predecir que el estado de este nuevo instante estará basado en el estado del instante anterior más la adición de las muestras preintegradas de la IMU. El resto de la etapa de `measure` se basa en el manejo y actualización de la base de datos de los puntos de interés en 3D, o *landmarks*, y sus *observaciones*, junto a algo que, en Basalt, está fuertemente ligado: la toma de cuadros clave, o *keyframes*.

3.2.2.2 *Base de datos de landmarks*

Recordemos que el módulo de optical flow encuentra keypoints en cada cuadro, esto es, una landmark o punto de interés en la escena 3D proyectada sobre el plano de la imagen 2D. Más aún este módulo era capaz de hacer el seguimiento de keypoints similares mediante optical flow, es decir, de keypoints que observan la misma landmark. Parte de nuestro objetivo entonces será triangular las posiciones de estas landmarks considerando las observaciones tomadas. Consideremos además la naturaleza altamente ruidosa de estas observaciones, con landmarks que aparecen y desaparecen de la visión de los cuadros por múltiples razones como: ser ocluidas por objetos en el entorno, ser distorsionadas por el ángulo del observador, distorsiones inherentes de los sensores ópticos[17] como el motion blur, la sobreexposición, el ruido introducido por la ganancia del amplificador de señal digital, o simplemente porque dejan de estar en el campo de visión de las cámaras. Por estas razones entonces, será fundamental la correcta gestión de la información de las landmarks y sus observaciones. En Basalt, la clase que se encarga de esto es la `LandmarkDatabase` con la estructura como se define en el Fragmento 3.1.

[17] *Detallaremos en la Sección 5.1 este tipo de distorsiones.*

Fragmento 3.1: Estructuras de Basalt

```
class LandmarkDatabase {
  map<LandmarkId, Landmark> landmarks;
  map<FrameId, map<FrameId, Keypoint>> observations;
}

class Landmark {
 FrameId keyframe;
 Vector2 direction;
 double inverse_distance;
}

class Keypoint {
  LandmarkId landmark_id;
  Vector2 position;
}
```

En la implementación de este Módulo, los términos de keypoint y landmark tienden a utilizarse de forma intercambiable, significando keypoint algo distinto a lo que era en el módulo de optical flow. Para evitar confusión, se han diferenciado los términos de manera explícita en el pseudo código anterior. `Landmark` se utilizará para referirse al punto de interés en la escena tridimensional, mientras que `Keypoint` será un punto 2D que observa la proyección de una landmark definida por `landmark_id`. Entonces la `LandmarkDatabase` es simplemente una colección de todas las `landmarks` de interés y de los keypoints que la observaron en `observations`.

Una `Landmark` surge de un keypoint observado en el módulo de optical flow en algún cuadro y comparte el mismo identificador. Explicaremos los campos `direction` e `inverse_distance` que determinan la posición en 3D de la misma más adelante. Un cuadro se identifica tanto por la timestamp en el que fue tomado como por cuál de las dos cámaras lo tomó; `FrameId` tiene esta información. Una landmark existe en referencia a un cuadro, y un cuadro que es referenciado por landmarks debe ser un keyframe por la implementación; decimos que el keyframe aloja estas landmarks. Una observación en `observations` tiene entonces un mapeo de keyframes a los `Keypoints` que observan a las landmarks alojadas en el keyframe. Un `Keypoint` es reconocido en un cuadro particular, y solo tiene sentido como coordenadas en ese cuadro, es esto lo que representa el segundo `FrameId` de la definición de `observations`.

Habiendo dicho esto, al recibir un nuevo cuadro, `measure` simplemente recorre todas las observaciones o `Keypoint`s que este trae y se añaden a la base de datos las observaciones de landmarks ya existente en la misma. Observaciones de landmarks no registradas en la base de datos se guardan para poder determinar si el módulo amerita la toma de un nuevo keyframe.

3.2.2.3 *Keyframes*

En contraste con sistemas como Kimera y ORB-SLAM3 que tienen condiciones más intrincadas, en Basalt, la heurística para decidir si el cuadro actual será un keyframe es muy sencilla: si *más del 30 % de las observaciones del cuadro actual corresponden a landmarks no registradas y han pasado más de, por defecto, 5 cuadros consecutivos que no fueron keyframes*, el cuadro actual será tomado como un keyframe. La toma de keyframes es lo que registra nuevas landmarks a la base de datos y realiza la estimación inicial de su posición.

En primer lugar, todas las observaciones *"desconectadas"* correspondientes al 30 % o más anterior que no correspondían a ninguna landmark, intentarán ser registradas en la base de datos. Para esto es necesario poder encontrar una segunda observación con la que realizar la triangulación y poder así estimar la posición tridimensional de la landmark al registrarla.

Se tiene entonces que para cada observación desconectada $\mathbf{p}_h \in \mathbb{R}^2$ producida por la cámara h del keyframe se recorrerán todos los cuadros desde el último keyframe, buscando por una segunda observación $\mathbf{p}_t \in \mathbb{R}^2$ de esta landmark producida en alguna cámara t. En caso de encontrarla se continúa de la siguiente manera:

- La cámara h del keyframe tiene una pose estimada para la IMU en esa timestamp que denominaremos $\mathbf{T}_{i_h} \in SE(3)$. Similarmente tendremos $\mathbf{T}_{i_t}$ para la cámara t.
- A su vez como los parámetros de calibración son estáticos y conocidos, tenemos la función de proyección π_h, sus parámetros intrínsecos $\mathbf{i}_h$ y la pose relativa $\mathbf{T}_{i_h c_h} \in SE(3)$ de la cámara h respecto a la IMU. Similarmente con π_t, $\mathbf{i}_t$ y $\mathbf{T}_{i_t c_t}$ para la cámara t.
- Podemos ahora desproyectar $\mathbf{p}_h$ y $\mathbf{p}_t$ a sus respectivas proyecciones (rayos) estimados $\mathbf{p}'_h := \pi_h^{-1}(\mathbf{p}_h, \mathbf{i}_h)$ y $\mathbf{p}'_t := \pi_t^{-1}(\mathbf{p}_t, \mathbf{i}_t)$ con $\mathbf{p}'_h, \mathbf{p}'_t \in \mathbb{R}^3$ en coordenadas homogéneas.
- Computamos ahora la transformación de la IMU de h a la IMU de t dada por $\mathbf{T}_{i_h i_t} := \mathbf{T}_{i_h}^{-1} \mathbf{T}_{i_t}$
- Luego podemos tener la transformación de la cámara h a t con $\mathbf{T}_{c_h c_t} := \mathbf{T}_{i_h c_h}^{-1} \mathbf{T}_{i_h i_t} \mathbf{T}_{i_t c_t}$
- Finalmente con estos tres datos $\mathbf{p}'_h$, $\mathbf{p}'_t$ y $\mathbf{T}_{c_h c_t}$ es posible triangular el punto 3D resultante de la forma descrita en Hartley y Zisserman [22] Cap. 12. Cabe aclarar que se necesitará utilizar cuadrados mínimos lineales para obtener el punto 3D final ya que los rayos proyectados provienen de mediciones ruidosas y usualmente no se alinearan de forma perfecta. Obtendremos el punto de la escena en coordenadas homogéneas en $\mathbb{R}^4$ con el cuarto componente representando el inverso de la distancia entre la cámara

y el punto de interés, mientras que los tres primeros componentes corresponden al *bearing vector*, es decir, un vector unitario que determina la dirección desde la cámara hacia la landmark.

Un detalle a considerar es que Basalt comprueba que la distancia relativa entre los dos cuadros, la *baseline*, sea lo suficientemente grande para considerar la triangulación exitosa, de lo contrario se busca un nuevo cuadro para comparar con el keyframe. Esta comprobación se reduce a revisar que la norma del vector de traslación contenido en $\mathbf{T}_{c_h c_t}$ sea de, por defecto, más de 5 cm.

Como se ve en la definición de `Landmark`, la posición 3D de estos puntos de interés no se almacena exactamente de la misma forma que la triangulación los produce. En particular, no se almacena el bearing vector directamente, sino que se utiliza un punto 2D más compacto `direction` que lo codifica (para esto se utiliza una *proyección estereográfica* como se explica en la Figura 3.4) junto a `inverse_distance`, la distancia inversa a este punto producto de la triangulación, de esta forma la posición de la landmark queda ligada al keyframe que la aloja.

Si todos los procedimientos relacionados con la triangulación de estos dos cuadros fueron correctos, se almacena la landmark nueva en la base de datos. De haber otras observaciones de esta landmark no se utilizan todavía para añadir información a su posición, sino que simplemente se añaden las observaciones para uso futuro.

3.2.3 *Optimización y marginalización*

Finalmente, la optimización central o *bundle adjustment* que ocurre en Basalt se centra en minimizar con cuadrados mínimos una función de error que combina residuales introducidos por las observaciones de las landmarks y la preintegración de la IMU. En la versión original se aplicaba Gauss Newton, pero luego de la actualización introducida en Demmel y col. [21] se utiliza Levenberg-Marquardt como algoritmo de minimización por defecto.

No nos adentraremos en los detalles de implementación de estos métodos por su complejidad, pero basta con aclarar que una buena parte de esta se debe al el cómputo explícito de los jacobianos para la linealización. En la práctica existen formas de calcular estos jacobianos con técnicas de diferenciación automática en tiempo de compilación como lo hace el optimizador Ceres[18] pero estas pueden incurrir en algunas pérdidas de rendimiento[19].

Vale la pena aclarar, que la minimización de la función de error es equivalente a realizar una estimación *máxima a posteriori* (*MAP*) [8] que a su vez se desprende de ideas similares a las encontradas en los

18 http://ceres-solver.org/nnls_solving.html
19 http://ceres-solver.org/analytical_derivatives.html#when-should-you-use-analytical-derivatives

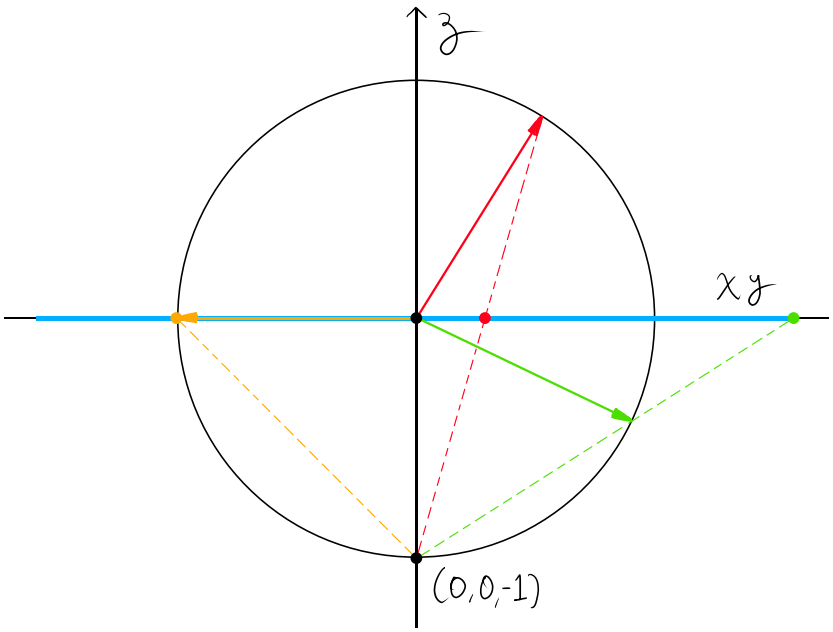


Figura 3.4: Interpretación geométrica de la proyección estereográfica utilizada para representar bearing vectors. Las coordenadas definidas por la propiedad `Vector2 direction` en `Landmark` definen un punto en el plano XY ($Z = 0$) mostrado en azul. Para obtener el vector unitario correspondiente, se traza una línea desde el punto $(0,\ 0,\ -1)^T$ hacia `direction` en el plano XY. El vector en el que esta línea interseca a la esfera unitaria será el bearing vector codificado. Se muestran tres ejemplos en rojo, verde y amarillo, con líneas punteadas que representan las líneas trazadas y flechas representando los bearing vectors obtenidos.

estimadores de *máxima verosimilitud*[20] pero con probabilidades condicionales de por medio.

20 https://en.wikipedia.org/wiki/Maximum_a_posteriori_estimation

Parte III

CONTRIBUCIONES

Describiremos algunas de las contribuciones clave que fueron producto de este trabajo. En el proceso se obtendrá un mejor entendimiento de los problemas comunes de implementación por los que estos sistemas se ven afectados. Además, veremos las soluciones que se les ha dado a otras problemáticas que son propias de la localización en tiempo real aplicada a XR.

4

TRACKING POR SLAM PARA MONADO

4.1 CONTEXTO

Por su naturaleza, el área de XR involucra una gran cantidad de partes interconectadas y de dispositivos muy diversos con configuraciones difíciles de generalizar, y más aún de predecir. Por esta razón hasta hace muy poco tiempo no existían estándares razonables en el área lo cual agravaba la situación con un ecosistema altamente fragmentado en soluciones propietarias que causaban grandes problemas a los desarrolladores de aplicaciones finales. En el mejor de los casos, la carga de soportar los distintos SDK propietarios recaía sobre frameworks como *WebXR*[1] o motores de juegos (p. ej. *Unreal Engine*, *Unity*, *Godot*[2]) y esto forzaba a los desarrolladores a elegir alguna de estas soluciones para realizar su aplicación de XR. En caso de no querer hacerlo, se vería obligado a realizar un esfuerzo significativo para portar su aplicación a cada uno de estos SDK, y eso sin considerar el manejo de características especiales que algunas plataformas exponen y otras no. Este escenario se puede ver en la Figura 4.1.

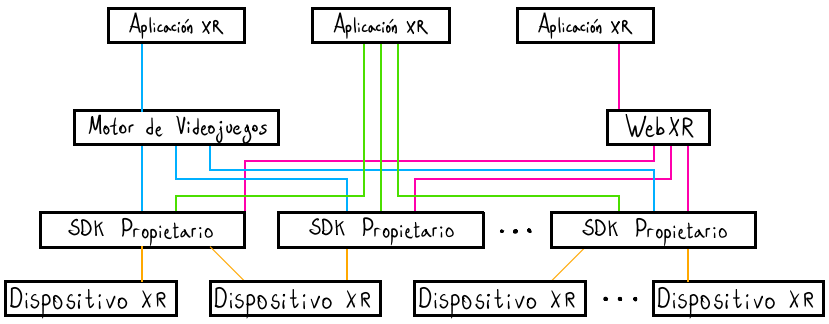


Figura 4.1: Antes de OpenXR las aplicaciones y motores necesitaban código propietario separado para cada SDK de los dispositivos que quisieran soportar.

Luego de unos años de sufrir esta fragmentación, en julio de 2019 se presenta la primera versión de *OpenXR* [6] de la mano del *Khronos Group*[3]. Este es un consorcio abierto y sin fines de lucro compuesto de, a la fecha, 170 organizaciones que desarrolla estándares en distintas áreas de la industria como computación gráfica (*OpenGL*, *Vulkan*),

1 https://www.w3.org/TR/webxr
2 https://www.unrealengine.com, https://unity.com y https://godotengine.org
3 https://www.khronos.org

computación paralela (*OpenCL*, *SYCL*) y, ahora con OpenXR, realidad virtual y aumentada entre otras. OpenXR provee una API estandarizada con soporte para extensiones que permiten añadir características peculiares de ser necesarias por algún fabricante en particular. El estándar ha tenido un gran éxito al haber sido adoptado por una gran cantidad de compañías [4] como reemplazo a sus antiguos SDK propietarios. De esta forma, los motores de juego y desarrolladores solo necesitan interactuar con una única API que además les permite aprovechar cualquier característica especial ofrecida por alguna extensión. Se puede ver la simplificación del esquema de integración con OpenXR en la Figura 4.2.

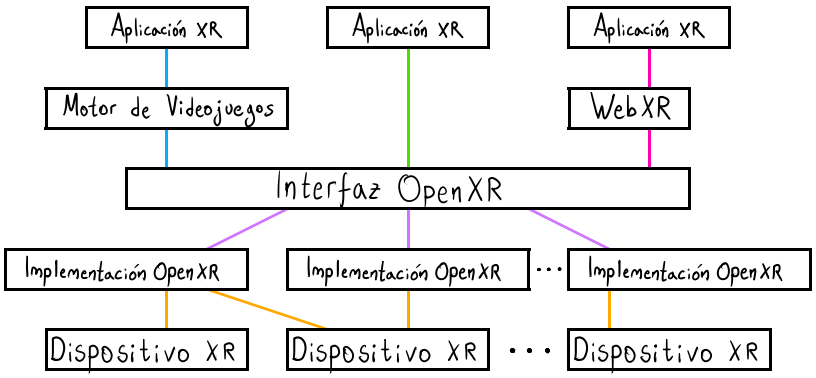


Figura 4.2: OpenXR provee una única interfaz (API) multiplataforma de alta performance entre las aplicaciones y todos los dispositivos compatibles. Esto es una mejora en comparación a la situación presentada en la Figura 4.1.

OpenXR es exclusivamente la especificación [6] de una API y por lo tanto requiere una implementación, o *runtime*, sobre el que ejecutarse. Las implementaciones son provistas por los distintos fabricantes interesados en soportar el estándar, en la imagen referenciada en la nota al pie [5] se pueden ver algunas de las implementaciones ya desarrolladas. En este trabajo nos enfocaremos en una de ellas, *Monado*. Monado es un runtime de la especificación OpenXR de código abierto, por ahora el único con esta característica, licenciado bajo la *Boost Software License 1.0* [33]. La plataforma principal sobre la que Monado corre y se desarrolla es GNU/Linux, pero es capaz de ser ejecutarse en otras como Android y Windows. Su desarrollo está soportado por *Collabora Ltd.*, quien es parte del grupo de trabajo de OpenXR desde sus inicios. Las contribuciones a Monado listadas en esta sección fueron

4 Compañías respaldando públicamente el estándar OpenXR: https://www.khronos.org/assets/uploads/apis/2019-openxr-logo-field_1_15.jpg

5 Algunas de las compañías que implementan un runtime de OpenXR: https://www.khronos.org/assets/uploads/apis/OpenXR-After_3.png.

realizadas durante una pasantía de seis meses realizada por el autor en Collabora.

Además de proveer una implementación de OpenXR, Monado es altamente modular e implementa distintos componentes reutilizables para XR como un compositor especializado para realidad virtual; controladores para una gran variedad de dispositivos, incluyendo hardware propietario y de consumo masivo sobre los que la comunidad ha realizado ingeniería inversa para poder utilizar; herramientas varias de calibración y configuración de hardware; así como también distintos sistemas de fusión de sensores para tracking; recientemente incluso incorpora un módulo de localización de manos mediante visión por computadora y aprendizaje automático.

Una característica faltante en Monado era la posibilidad de realizar localización visual-inercial mediante sistemas de SLAM/VIO. Este tipo de tracking ha cobrado gran popularidad en los últimos años por resultar sumamente convenientes al no requerir sensores externos al dispositivo de XR. Sistemas de este tipo son empleados en productos como el *Meta Quest*, los cascos *Windows Mixed Reality* o incluso los SDK *ARCore* y *ARKit* presentes en dispositivos móviles. Desafortunadamente, todas estas soluciones son privativas y, por lo tanto, no es posible obtener acceso a sus códigos fuentes para reusarlos, modificarlos o simplemente estudiarlos sin obtener licencias especiales de sus fabricantes. Más aún, existen compañías que se especializan en desarrollar soluciones comerciales de SLAM como *SLAMCore*[6], *Arcturus*[7] y *Spectacular AI*[8] entre otras.

Este trabajo se concentró entonces en el estudio de implementaciones de código abierto de sistemas de tracking visual-inercial (ya sea mediante SLAM o solamente VIO) y en la integración de estos sobre Monado. Se necesitó armar la infraestructura para soportar una interacción modular con los sistemas mediante el desarrollo de interfaces, herramientas, controladores, y mejoras principalmente en Monado, pero también en los sistemas a integrar o en sus *forks* (clones específicos de las implementaciones de SLAM/VIO para uso en Monado).

4.2 MONADO

En este capítulo explicaremos los distintos módulos y funcionalidad implementada en Monado para permitirle proveer tracking mediante SLAM/VIO a distintos controladores de dispositivos y de esta forma, a las aplicaciones OpenXR que utilizan estos dispositivos como medios de interacción y corren sobre Monado. Cabe aclarar que a partir de ahora usaremos de forma intercambiable los términos SLAM y VIO, ya que, a pesar de ser distintos, para los fines prácticos de la imple-

6 https://www.slamcore.com/
7 https://arcturus.industries/
8 https://www.spectacularai.com/

mentación son equivalentes: *piezas de software que consumen imágenes y muestras de IMU y devuelven poses estimadas como resultado*. Es por esto que hablaremos de un localizador SLAM o *SLAM tracker* que se ha implementado para referirnos a una única funcionalidad que soporta tanto sistemas externos de SLAM y como de VIO.

Antes de comenzar, vale la pena entender la arquitectura general de Monado como se muestra en la Figura 4.3. Una aplicación que hace uso de OpenXR mediante Monado puede ser corrida en dos modalidades dependiendo de como se compiló el runtime. De la forma tradicional, se compila como una biblioteca de manera **autónoma**, es decir que cuando la aplicación intenta cargar dinámicamente alguna implementación de OpenXR provista por el sistema, se enlaza la biblioteca compartida, o *shared library*, de Monado, y al interactuar con cualquier interfaz a OpenXR, se correría tal procedimiento en el mismo hilo que la aplicación consumidora. Esta modalidad puede tener ciertos beneficios para depuración de código, pero el modo más usual de compilar y correr Monado es mediante *Inter-Process Communication* o *IPC*. De esta manera el runtime se corre en un proceso independiente que debe ser lanzado con anterioridad a cualquier aplicación OpenXR. Lo que se termina enlazando a la aplicación final es simplemente un cliente IPC que es capaz de comunicarse con Monado. Esto ofrece ventajas como la posibilidad de correr múltiples aplicaciones sobre la misma instancia del runtime. Además, se quita la necesidad de ejecutar ambos procesos sobre el mismo nodo de cómputo, lo cual facilita la posibilidad de tener realidad virtual renderizada en la nube.

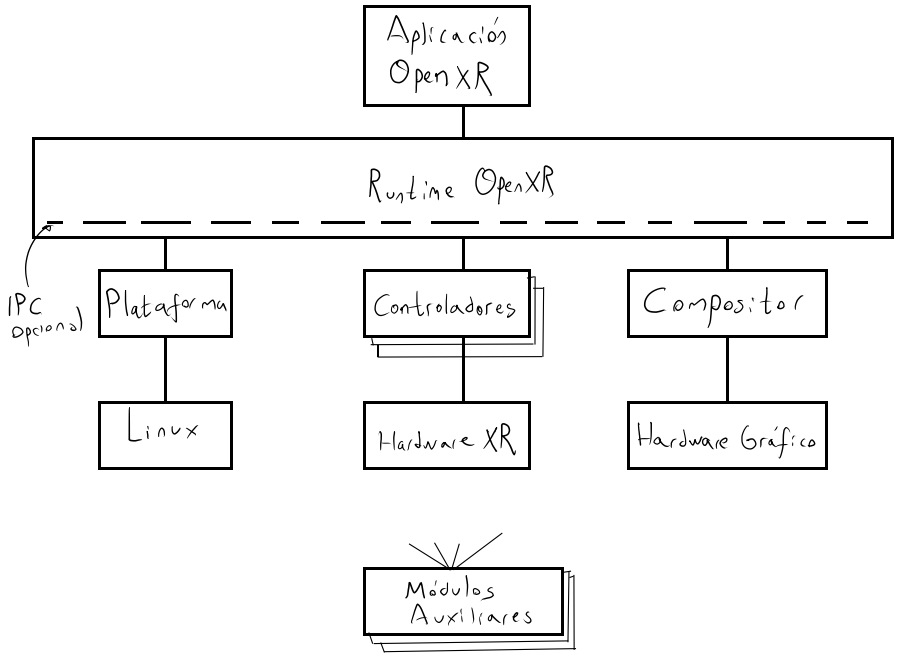


Figura 4.3: Arquitectura de Monado.

No profundizaremos en los aspectos de comunicación con la *plataforma* ni del *compositor* de Monado, pero es bueno saber que estos son partes significativas del runtime. Son los *controladores* (o *drivers*) específicos en Monado los que interactúan con el hardware especializado como cascos, controles, cámaras, entre otros dispositivos que quieran utilizarse para la experiencia de XR. Sensores como las IMU y cámaras sobre los que adquirir las muestras necesitarán ser implementados en este nivel. Finalmente, hay una gran variedad de módulos auxiliares reutilizables que exponen funcionalidades matemáticas, de interacción con el sistema operativo, con OpenGL o Vulkan, entre otras. Es en estos módulos auxiliares en donde encontramos componentes relacionados exclusivamente al problema tracking y a tipos específicos de tracking. Se implementan herramientas de calibración de cámaras, de depuración, filtros de distinto tipo (p. ej. *Kalman* y *low-pass*), entre otra algoritmia genérica de fusión de sensores. Conforman este módulo también sistemas completos de tracking para *PlayStation Move*, *PlayStation VR*, y tracking de manos en general. Es aquí entonces en donde se comienza la implementación del SLAM tracker presentado en este trabajo.

La implementación de un pipeline en Monado que permita la comunicación entre dispositivos, sistemas de SLAM y la aplicación OpenXR requirió desarrollar la infraestructura y herramientas necesarias dentro de Monado. El pipeline en cuestión está esquematizado en la Figura 4.4. Este, requiere un gran nivel de modularidad, ya que sus componentes fundamentales necesitan poder ser intercambiables: dispositivos, aplicaciones y el sistema que provee el SLAM en sí. El resto de esta subsección está dedicada a explicar la infraestructura y los distintos componentes que se necesitaron implementar y adaptar para obtener un pipeline de SLAM modular corriendo en Monado como se muestra en la Figura 4.4.

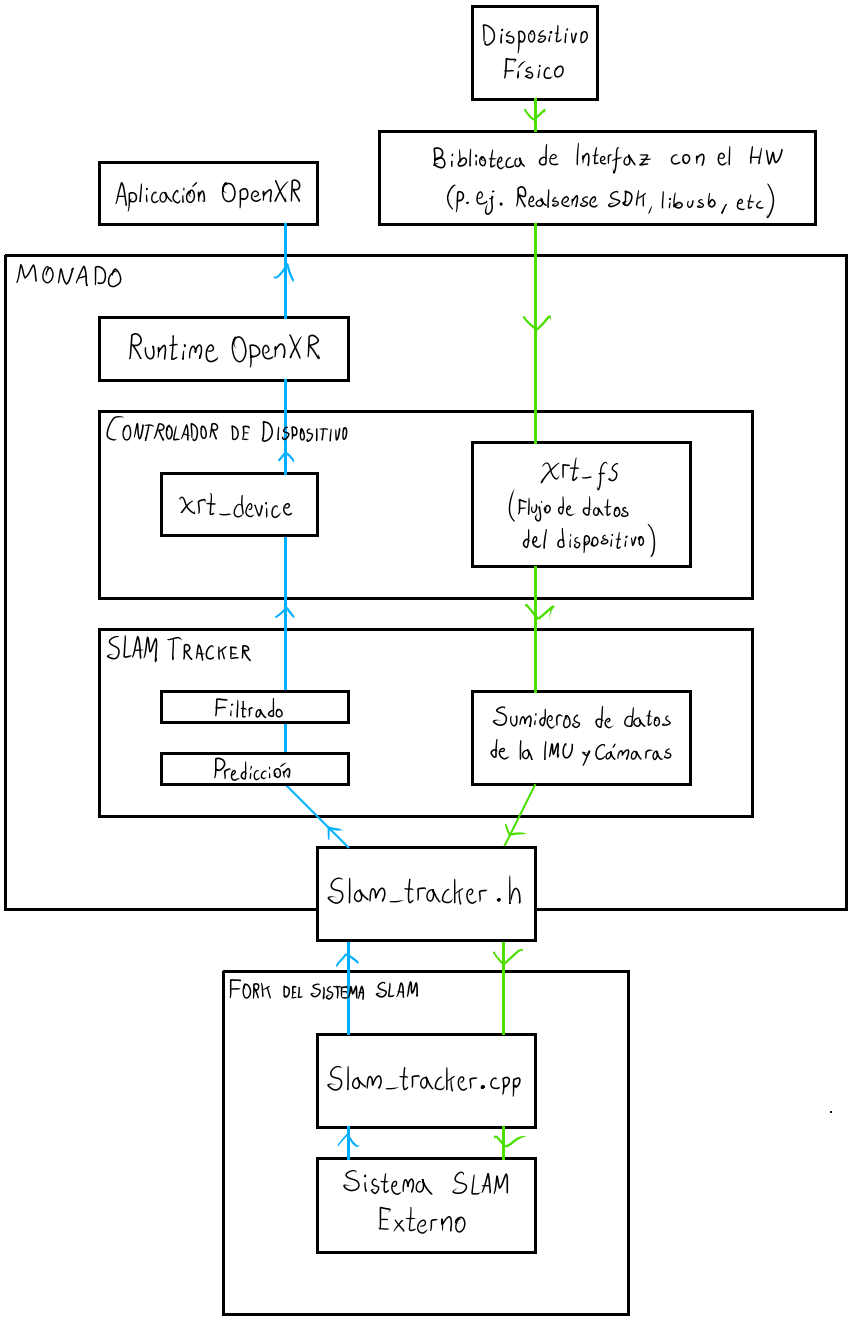


Figura 4.4: Diagrama esquemático de como ocurre el flujo de los datos desde que se generan las muestras en los dispositivos XR hasta que la aplicación OpenXR obtiene una pose utilizable. Notar que las flechas esconden múltiples hilos de ejecución, colas de procesamiento, y desfasajes temporales con los que hay que lidiar en la implementación.

4.3 INTERFAZ EXTERNA

Desde un principio se entendió que se necesitaría utilizar sistemas ya desarrollados como punto de partida. Estos sistemas son complejos y suelen utilizar conceptos teóricos de significativa profundidad, por lo que su creación suele estar limitado a grupos de investigación expertos que toman gran tiempo de desarrollo. Los tres sistemas estudiados por ejemplo, promedian las 25.000 líneas de código (o 25 *KLOC*) cada uno.

Ahora bien, en muchos casos, haber intentado integrar el código del sistema directamente dentro de un componente de Monado no era una opción. Dejando de lado las dificultades técnicas, los problemas de compatibilidad de licencia fueron de particular interés. La gran mayoría de sistemas SLAM producidos en la academia son liberados bajo licencias abiertas "virales" como *GPL* [18] que obligan a desarrolladores que utilizan código del sistema a liberar y licenciar su código de la misma manera. Esto contrasta con la licencia abierta y permisiva de Monado, la *BSL-1.0* [33], que no impone restricciones sobre como los usuarios deben licenciar su código.

Estas fueron algunas razones para intentar desacoplar el sistema a utilizar lo más que se pueda de Monado. Además, considerando la naturaleza experimental de este trabajo, la posibilidad de que más de un sistema necesitase ser integrado era razonable.

Monado está desarrollado principalmente en C, pero gran parte de su código de tracking está implementado en C++ al igual que todos los sistemas de SLAM contemplados. Adicionalmente tanto Monado como estos sistemas suelen hacer un uso extensivo de la biblioteca *OpenCV*, y en particular su clase contenedora de imágenes y matrices `cv::Mat`. Es por esto que se terminó optando por el uso de un archivo *header* C++, en el cual se declara la clase `slam_tracker`[9] que será utilizada por Monado como punto de comunicación con sistemas de SLAM arbitrarios y se utilizan `cv::Mat` como contenedor de imágenes. Luego de varias iteraciones de diseño, la clase `slam_tracker` tiene una interfaz que, quitando detalles de tipos de C++, se puede resumir en algo como lo que se muestra en el Fragmento 4.1.

9 https://gitlab.freedesktop.org/monado/monado/-/blob/2d9c1b2b11373f707b990e5b8a28b15bc1454b83/src/external/slam_tracker/slam_tracker.hpp#L95-167

Fragmento 4.1: Interfaz a implementar por sistemas de SLAM

```cpp
class slam_tracker {
public:
  // (1) Constructor y funciones de inicio/fin
  slam_tracker(string config_file);
  void start();
  void stop();

  // (2) Métodos principales de la interfaz
  void push_imu_sample(timestamp t, vec3 accelerometer, vec3 gyroscope);
  void push_frame(timestamp t, cv::Mat frame, bool is_left);
  bool try_dequeue_pose(timestamp &t, vec3 &position, quat &rotation);

  // (3) Características dinámicas opcionales
  bool supports_feature(int feature_id);
  void* use_feature(int feature_id, void* params);

private:
  // (4) Puntero a la implementación (patrón PIMPL)
  void* impl;
}
```

Este header está presente en Monado, pero su implementación no. Esta debe ser provista por el sistema externo, lo cual implica tener que mantener una copia, o *fork*, levemente modificado de los distintos sistemas que se quieran utilizar, ver Figura 4.5.

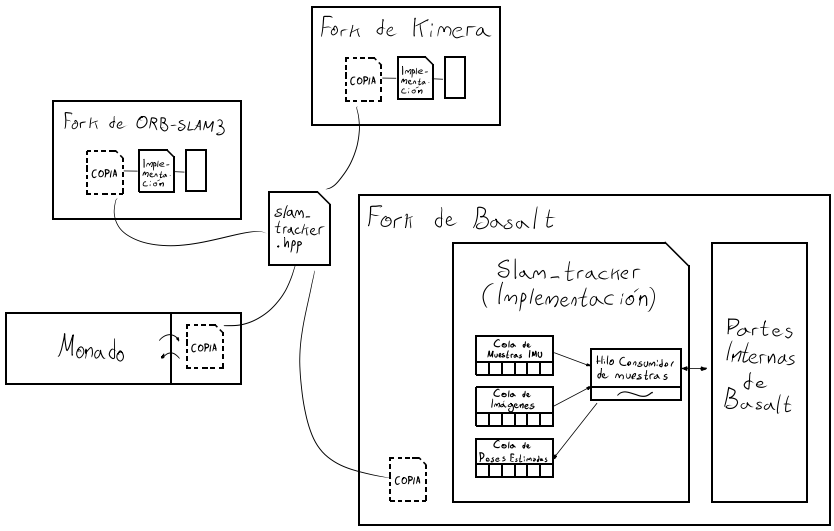


Figura 4.5: Interacción entre Monado y sistemas SLAM mediante la interfaz en C++. Enlaces a estos forks pueden verse en las Contribuciones 2 a 4.

La versión actual de esta clase es el resultado de varias iteraciones y generaliza adecuadamente los tres sistemas en uso. Algunas consideraciones de los puntos marcados en el código:

1. El parámetro `config_file` del constructor es necesario, ya que todos los sistemas con los que se trató requieren proveer información de calibración y puesta a punto de parámetros previo a la corrida mediante un archivo de configuración. Además estos sistemas suelen tener etapas de creación e inicialización de recursos, así como de liberación de los mismos que quedan representados en el par de métodos `start()`/`stop()`.

2. Los sistemas corren en hilos separados de Monado y es por esto que es fundamental implementar colas concurrentes a la hora de intercambiar datos. Monado ingresa muestras mediante los métodos `push_imu_sample` y `push_frame` mientras que sondea si hay poses ya estimadas por el sistema y las obtiene mediante `try_dequeue_pose`.

3. Algo que surge del desarrollo de una interfaz a un tipo de sistemas que aún se desconocen, es que va a haber varios cambios en la misma durante su creación. Si además esta interfaz es compartida por múltiples sistemas y repositorios, mantener todas las versiones sincronizadas se vuelve insostenible. Una forma de aliviar este problema fue la implementación de características dinámicas en la Contribución 11. En ellas, Monado evalúa si el sistema implementa alguna característica específica en tiempo de ejecución antes de utilizarla. Uno de sus usos, fue la automatización del envío de datos de calibración sin pasar por el archivo `config_file`. La forma de añadir nuevas características de este tipo se debe reservar un entero `feature_id` que la identifique en una nueva versión del header `slam_tracker` de Monado y del fork que la va a implementar. En Monado tenemos cuidado de solo utilizarla si el sistema la reporta como disponible en tiempo de ejecución, ya que otros forks podrían no implementarla. De esta forma permitimos la extensión de sistemas específicos sin tener que adaptar la versión de la interfaz en todos ellos.

4. El miembro `impl` es una forma de ligar la definición compacta de `slam_tracker` con una clase que implementa el estado y métodos privados necesarios para proveer la funcionalidad requerida. Este patrón de desarrollo es usualmente conocido como *pointer to implementation* (o *PIMPL*), a `impl` se lo denomina un *puntero opaco* [34].

Esta interfaz no es perfecta: no contempla magnetómetros, asume una configuración de a lo sumo dos cámaras, asume que el sistema utiliza OpenCV y es difícil de extender con cambios no contemplados por el concepto de características dinámicas. A pesar de estos problemas, ha sido suficientemente buena para generalizar todos los sistemas propuestos y correrlos con una performance adecuada [10].

[10] *ILLIXR [35] es un proyecto de XR de código libre desarrollado por la Universidad de Illinois, que también formó recientemente el Consorcio ILLIXR para intentar mejorar el ecosistema de código libre para XR. Ha habido discusiones entre ILLIXR y Monado para plantear una interfaz a sistemas de SLAM que le sirva a ambos proyectos y que quizás en un futuro pueda transformarse en un estándar que los sistemas implementen por elección propia. Para esta nueva interfaz, se tomarán ideas basadas en la experiencia obtenida desarrollando el header* `slam_tracker`.

4.4 IMPLEMENTACIONES DE LA INTERFAZ

A la hora de implementar la interfaz `slam_tracker` se documentan los tres métodos principales `push_imu_sample`, `push_frame` y `try_dequeue_pose` con las precondiciones que el usuario de la interfaz, Monado en este caso, y su implementación deben cumplir:

1. Debe haber un único hilo productor llamando a los métodos `push_*`
2. Las muestras deben tener marcas de tiempo (*timestamps*) monótonas crecientes.
3. La implementación de los métodos `push_*` no debe ser bloqueante.
4. Del punto anterior se desprende que las muestras deben ser procesadas en un hilo consumidor separado.
5. Las muestras de imágenes estéreo deben tener la misma timestamp.
6. Las muestras de imágenes estéreo deben ser enviadas de forma intercalada en orden izquierda-derecha.
7. Debe haber un único hilo consumidor llamando a `try_dequeue_pose`.

Estas condiciones fueron desarrollándose mientras la interfaz evolucionaba y nuevos sistemas se iban adaptando. Intentan ser indicaciones que evitarían implementaciones deficientes mientras que son lo suficientemente generales como para que todas las soluciones estudiadas puedan cumplirlas.

La precondición **1** para el usuario le permite a la implementación utilizar colas enfocadas al caso de un único productor, de las cuales puede obtenerse mayor rendimiento comparado a las versiones que no tienen esta limitación. Más aún, se suelen utilizar colas libres de *locks* (primitiva de sincronización, también conocido como *mutex*). El punto **2** facilita el trabajo a la implementación mientras que suele ser trivial de cumplir para el usuario, ya que los dispositivos tienden a reportar las muestras del mismo tipo en orden temporal creciente. El punto **3** obliga a la implementación de los métodos `push_*` a terminar lo antes posible sin realizar ningún tipo de procesamiento en esas funciones. Esto es fundamental, ya que Monado también recibe muestras de dispositivos mediante colas y simplemente las redirige al `slam_tracker`, y demorarse en estos métodos hace que las colas internas de Monado se llenen y muestras se pierdan. El inciso **4** aclara en dónde se procesarían las muestras; dado que no pueden procesarse en los métodos `push_*` se necesita un hilo consumidor que esté esperando por muestras nuevas

en las distintas colas de la implementación para procesar. El punto **5** es un requerimiento razonable de algunos sistemas, ya que es común que cámaras estéreo pensadas para SLAM tengan sincronización por hardware de sus timestamps. El punto **6** facilita algunos procedimientos y chequeos en la implementación. Y finalmente, la precondición **7** es similar al punto 1 en donde se le permite a la implementación utilizar colas de un único consumidor que sean de mayor rendimiento comparado a otras colas concurrentes.

La interfaz `slam_tracker` se implementó en los tres forks presentados en este trabajo; Kimera, ORB-SLAM3 y Basalt. La idea general de las implementaciones es similar: permitir a Monado, el usuario de la interfaz, utilizar el sistema de SLAM/VIO realizando las conversiones adecuadas para poder comunicarse con la solución subyacente. Sin embargo hay algunas consideraciones a remarcar en cada versión.

Respecto al manejo de muestras entrantes, Kimera y Basalt encolan las muestras directamente a las colas de entrada de sus respectivos pipelines que corren en hilos separados. Para ORB-SLAM3, al ser necesario ejecutar explícitamente el paso de estimación con el par de imágenes estéreo y muestras IMU nuevas, se necesitó implementar colas dedicadas para muestras de cámara izquierda, derecha e IMU, junto a un hilo consumidor que ejecuta la estimación cuando llegan cuadros nuevos. Cabe mencionar que Basalt no utiliza el tipo `cv::Mat` de OpenCV como contenedor para sus imágenes a diferencia de los otros sistemas. Más aún, espera imágenes con intensidad de píxeles de 16 bits para soportar conjuntos de datos como TUM-VI que provee imágenes de este tipo. Sin embargo, las imágenes monocromáticas usadas comúnmente en estas aplicaciones suelen utilizar un rango de 8 bits y por ende los dispositivos generan imágenes de este tipo. Al no haber una forma sencilla de utilizar 8 bits en Basalt, es necesario realizar una copia a un tipo de imagen de 16 bits cada vez que se presentan nuevos cuadros. Este es un punto a mejorar en el futuro sobre Basalt que no se trató de resolver en este trabajo, ya que no se encontraron problemas de performance incluso con esta copia innecesaria.

Respecto a la cola de poses estimadas, Kimera implementa un sistema de *callbacks* que llaman a una función encargada de encolar los resultados del pipeline tan pronto como estén listos. Basalt por otra parte, expone una cola que se va llenando con los resultados de la estimación; la implementación de `slam_tracker` la utiliza directamente, ya que también expone una interfaz de cola mediante el método `try_dequeue_pose`. ORB-SLAM3 difiere de los dos anteriores ya que el procesamiento sucede en una llamada a una función bloqueante que al terminar devuelve la pose estimada. Por lo tanto, nuestra implementación se encarga de armar la maquinaria necesaria de hilos productores, hilos consumidores y colas de entrada y salida para hacer que Monado no sea bloqueado al enviar nuevas muestras de los dispositivos.

Respecto al archivo de configuración referenciado por el parámetro `config_file`, Basalt por defecto utiliza dos archivos, uno para parámetros de calibración y el otro para configuraciones del sistema en sí; se necesitó entonces crear un tipo de archivo nuevo que referencie a estos dos y sea utilizado como `config_file`. La misma idea se utilizó para Kimera que también posee múltiples archivos de configuración. ORB-SLAM3 por su parte utiliza un único archivo por defecto y no necesito de mayores cambios.

Respecto a las interfaces gráficas, todos estos sistemas presentan la posibilidad de visualizar, al menos, la trayectoria estimada junto a las features detectadas en los cuadros entrantes como se muestra en la Figura 4.6. En todos los casos, siempre se intenta permitir la posibilidad de utilizar estas herramientas de visualización de forma opcional al utilizar la clase `slam_tracker`. Para Kimera y ORB-SLAM3 las interfaces funcionan automáticamente al habilitarlas en los archivos de configuración, mientras que Basalt utiliza sus herramientas de visualización de manera más ad hoc y por lo tanto fue necesario implementar una clase de visualización dedicada para el `slam_tracker`.

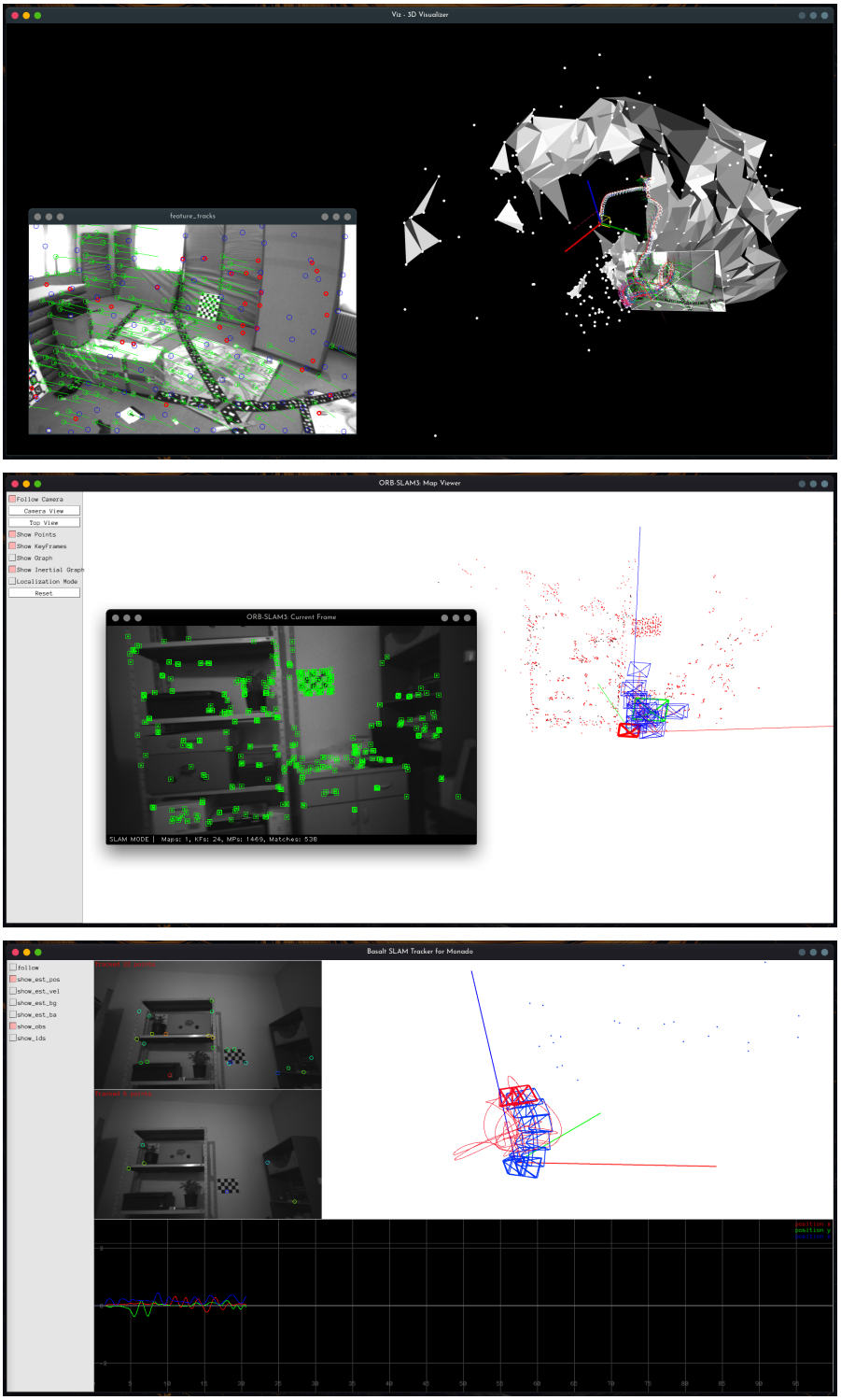


Figura 4.6: Las distintas interfaces gráficas y formas de visualizar presentadas por cada uno de los sistemas adaptados. De arriba a abajo: Kimera, ORB-SLAM3 y Basalt.

4.5 CLASE ADAPTADORA

Para interactuar con la interfaz que detallamos en la sección anterior, se implementa en Monado una clase adaptadora en el módulo auxiliar de tracking (recordar Figura 4.3) y que sirve de nexo entre los controladores de dispositivos de hardware y la interfaz `slam_tracker`. Este adaptador recibe el nombre, un poco confuso, de `TrackerSlam` siguiendo las convenciones de Monado para con los trackers ya existentes.

El funcionamiento de `TrackerSlam` es sencillo, los controladores que quieran ser localizados por SLAM y puedan proveer imágenes y muestras de IMU deben instanciar este adaptador e inicializarlo. Por detrás, esto simplemente llama a los métodos adecuados de la interfaz `slam_tracker` con el sistema externo. Ahora bien, se aprovecha esta clase adaptadora[11] para proveer dos funcionalidades fundamentales que escapan al alcance de los sistemas de SLAM/VIO y serán explicados a continuación: **predicción** y **filtrado** de poses.

[11] *Es debatible si el añadido de estas funcionalidades haría que* `TrackerSlam` *deje de ser considerada una clase adaptadora, ya que como veremos, ambas pueden ser deshabilitadas en tiempo de ejecución.*

4.5.1 *Predicción de poses*

4.5.1.1 *El problema*

Las aplicaciones XR requieren poder localizar constantemente a los distintos dispositivos de entrada y salida soportados que son utilizados por el usuario. En la especificación de OpenXR [6] las dos principales funciones que le permiten a la aplicación pedirle al runtime las poses de estos dispositivos son `xrLocateSpace` y `xrLocateViews`. La primera se utiliza para solicitar poses de dispositivos "comunes" (suelen ser distintos tipos de mandos) mientras que la segunda es un poco más compleja, ya que concierne a la ubicación de las pantallas que renderizan la escena (p. ej. la ubicación de las pantallas de un casco VR). Para nuestro propósito, podemos resumir las signaturas de ambas funciones a:

```
1 XrPosef xrLocateSpace(XrSpace id, XrTime time);
2 XrPosef xrLocateViews(XrSpace id, XrTime time);
```

Ambas devuelven una pose a un tiempo `time` para el "*espacio*" identificado por `id`. Un espacio o *espacio de referencia* es un término utilizado en la especificación para diferenciar cualquier punto que nos interese trackear desde la aplicación y que en definitiva identifica a un sistema de referencia inercial con rotaciones.

Para obtener estos espacios, la aplicación solicita las características que desea. Si se solicitara el espacio de un control o mando el runtime, Monado en este caso, intentará conseguir el más adecuado dentro de los disponibles en el sistema. Entonces, la aplicación OpenXR es indiferente a qué dispositivos están siendo utilizados, ni siquiera se asumen que estos espacios sean dispositivos, podrían ser cualquier otro objeto de interés que está siendo localizado por mecanismos externos (por

visión por computadora por ejemplo) o incluso dispositivos emulados por software[12]. Los espacios que aplican a nuestro caso son aquellos que representaran dispositivos que posean sensores IMU y cámaras que puedan utilizarse en nuestros sistemas de SLAM/VIO.

Otro importante aspecto a considerar es que el punto en el tiempo `time` para la cual la pose debe ser estimada es provisto por el usuario, y como tal, resulta arbitrario para el runtime. Sin embargo los sistemas de SLAM/VIO suelen ser sistemas de tiempo discreto, es decir solo dan estimaciones para puntos en el tiempo para los cuales tienen muestras. En el mejor de los casos esto implica que pueden dar una estimación por cada muestra de IMU, las cuales vienen a altas frecuencias (p. ej. 200hz); Kimera cumple con esto. En el caso más usual sin embargo, y el que más nos afecta en este trabajo, las estimaciones vienen a la misma frecuencia que los cuadros de las cámaras. Esta frecuencia suele ser al menos un orden de magnitud menor (p. ej. 20hz); ORB-SLAM3 y Basalt funcionan de esta manera.

Para complicar más las cosas, los tiempos en los que el programador solicita las poses suelen estar ligados a momentos en los cuales hay que renderizar un nuevo cuadro en la pantalla del usuario. Al ser este un proceso que (en el caso usual) ocurre enteramente en el mismo nodo de cómputo, suelen ser tiempos muy cercanos al presente. Para el pipeline de SLAM sin embargo, siempre nos encontramos levemente en el pasado por tener que lidiar con las demoras inherentes a la captura de muestras, las transmisiones de datos y los tiempos de cómputo significativos de los sistemas de SLAM.

[12] *Una de las primeras contribuciones realizadas para familiarizarse con el código fuente de Monado fue la implementación del controlador `qwerty` que le permite a los usuarios emular de forma modular un casco y/o mandos mediante teclado y ratón. Ver Contribución 6.*

4.5.1.2 *Análisis de un ejemplo*

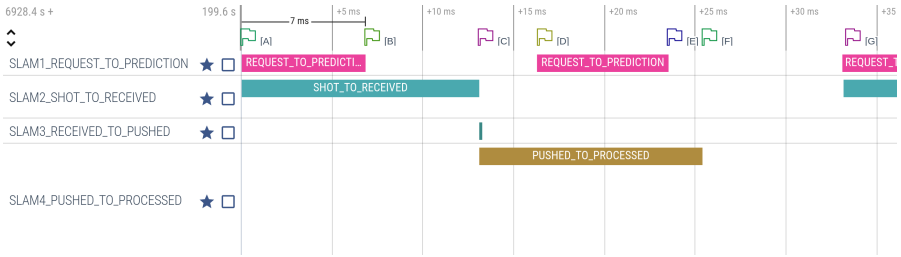


Figura 4.7: Línea de tiempo con timestamps normalizadas. Las barras representan las siguientes duraciones: `REQUEST_TO_PREDICTION`: Del momento en que una predicción es solicitada hasta la timestamp de la predicción. `SHOT_TO_RECEIVED`: Captura de imagen en el dispositivo hasta su recepción en Monado. `RECEIVED_TO_PUSHED`: Transferencia de Monado a Basalt. `PUSHED_TO_PROCESSED`: Cómputo de la estimación de pose.

En la Figura 4.7 se puede apreciar una captura de pantalla de la interfaz de *Perfetto*[13] que, en conjunto con *Percetto*[14], son las herramientas de medición de tiempos preferidas para Monado. Esta captura es sobre una corrida en tiempo real con Monado, Basalt y una cámara RealSense D455 con imágenes estéreo de resolución 640x480 a 30 cuadros por segundo. La figura muestra un tramo de unos 35 ms con la particularidad de que el tiempo en el que el usuario pide una predicción y el par estéreo de imágenes son capturadas por la cámara coinciden en `[A]`. Como la tarea de renderizado para esta captura ocurría a unos 60 cuadros por segundo, tenemos que aproximadamente dos predicciones (en `[A]`, `[D]`) son solicitadas entre cada muestra de la cámara (`[A]` y `[G]`).

A tiempo `[C]` las imágenes llegan al host luego de haber sido transferidas por un cable USB 3.2 con una demora de unos 13,5 ms representada por la barra `SHOT_TO_RECEIVED`. En ese momento ocurre una pequeña copia de Monado hacia Basalt `RECEIVED_TO_PUSHED`[15], y luego de unos 12 ms representados por `PUSHED_TO_PROCESSED`, a tiempo `[F]`, la pose estimada para el tiempo `[A]` está computada. Es decir, tenemos una demora de unos 25,5 ms desde que la muestra es capturada hasta que el sistema de SLAM/VIO es capaz de estimar la pose correspondiente al momento de captura de la muestra. Cabe aclarar que estos tiempos son muy variables incluso en la misma corrida, al depender de la calidad de la conexión USB, el sistema utilizado, y las propias muestras que pueden complicar las iteraciones de los algoritmos de optimización que ocurren en el sistema.

[15] *Esta copia es necesaria por un detalle en la implementación del* `slam_tracker` *para Basalt. Si bien es solucionable, como se ve en el gráfico, no afecta significativamente al rendimiento del pipeline*

Por otro lado, tenemos que a tiempo `[A]`, la aplicación OpenXR solicita a Monado una predicción de a dónde el runtime piensa que el dispositivo se va a encontrar 7 ms en el futuro, en `[B]`. Cabe aclarar que la solicitud debe ser respondida de forma inmediata y la barra `REQUEST_TO_PREDICTION` no implica ninguna espera hasta `[B]`, es solo una forma de visualizar los 7 ms. Notar que en ese punto, todavía faltan 25,5 ms para tener la predicción de Basalt para `[A]`, más aún, una predicción dada por Basalt para el futuro `[B]` ni siquiera existirá, ya que el sistema solo estima poses para los tiempos de las muestras, es decir la próxima estimación correspondería al tiempo `[G]`. Además de esto, tenemos que mientras todavía se está procesando la muestra, otra petición a tiempo `[D]` para `[E]` debe ser respondida.

En conclusión, tenemos un **desfasaje temporal** que hace que las poses estimadas siempre estén levemente en el pasado; en el tramo seleccionado fue de 25,5 ms (más los 7 ms de predicción), pero es variable durante la corrida. Además, las poses se estiman para **puntos discretos** de tiempo, mientras que el usuario puede pedir una predicción para cualquier punto arbitrario. Entonces, si queremos ser capaces de proveer al usuario una pose para el tiempo solicitado, necesitemos imple-

13 Biblioteca de perfilación Perfetto: `https://perfetto.dev`
14 Wrapper de Perfetto para C: `https://github.com/olvaffe/percetto`

mentar formas de interpolar y ***predecir*** la ubicación de un espacio. Es decir, no utilizamos las poses devueltas por el sistema de SLAM/VIO de forma directa, sino como parte fundamental de un procedimiento constante de predicción que se realiza del lado de Monado. En este se intenta utilizar todos los datos disponibles en el runtime para proveer la pose más razonable en el punto de tiempo requerido.

4.5.1.3 *Implementación*

Se encara la solución a estos problemas de manera progresiva y haciendo uso de algunas de las herramientas e ideas ya presentes en Monado.

Dentro de Monado, el concepto de espacio de referencia o `XrSpace` de de OpenXR, es nombrado como una *relación espacial* y se representa con un `struct` muy similar al siguiente.

```
struct xrt_space_relation {
  struct vec3 position;
  struct quat orientation;
  struct vec3 linear_velocity;
  struct vec3 angular_velocity;
};
```

En él, no solo se guarda la información de la pose (`position` y `orientation`) sino que además tenemos el estado de la velocidad lineal y angular de este espacio. Esto es muy útil, ya que si sabemos que a tiempo t_1 tenemos cierto espacio con pose $T_1 \in SE(3)$ y velocidades $v_1, \omega_1 \in \mathbb{R}^3$, podemos predecir que a un tiempo $t_2 = t_1 + \Delta t$ tendremos el espacio con pose $T_2 = \Delta T \; T_1$ con

$$\Delta T = \begin{bmatrix} exp(\Delta t \; \hat{\omega}) & \Delta t \; v \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4x4} \tag{4.1}$$

Monado provee varias herramientas que facilitan tareas que suelen ser recurrentes en diversos sistemas de tracking. La tarea de estimar espacios futuros basándose en uno dado con sus velocidades es una de estas funcionalidades ya incluidas. Más aún, Monado implementa una estructura de datos que permite almacenar un "historial" de estos espacios en una cola circular y generar las interpolaciones y extrapolaciones, tanto a futuro como a pasado, necesarias para cualquier timestamp requerida por un usuario. Para las interpolaciones se utiliza una simple interpolación lineal[16] o *lerp* de a trozos entre cada par de espacios del historial. Para extrapolar hacia el futuro, se usa la pose y velocidad almacenada en el espacio más reciente del historial para realizar el cómputo con ΔT como se definió en la Ecuación 4.1. Simétricamente para extrapolar hacia el pasado lejano[17], o sea fuera del registro del

16 En el caso de la orientación es la interpolación esférica lineal *slerp* que definimos en la Ecuación 2.14

17 *La especificación de OpenXR tiene una sección dedicada a las restricciones y condiciones a los que el runtime está sujeto respecto a solicitudes en el pasado y en el futuro por parte del usuario. Ver* [6], *secc. 2.14.*

historial, se utilizará el espacio más antiguo almacenado y ΔT^{-1}.

Sería razonable, como primera aproximación a nuestro problema, utilizar este historial de espacios. Esto nos garantiza que podamos proveerle al usuario una pose para cualquier punto arbitrario en el tiempo que solicite, basándonos en las estimaciones del sistema de SLAM que asumimos son precisas. Sin embargo, un leve problema que surge es que no existe ningún requerimiento para los sistemas sobre la estimación de velocidades, ya que no todos estiman esta variable. La interfaz `slam_tracker` solo garantizan la estimación de la posición y orientación del dispositivo como puede verse en su definición en el Fragmento 4.1. Lo que haremos para solventar esto es computar las velocidades con base a los pares de poses adyacentes que tengamos en el historial. Estas poses tienen su timestamp correspondiente, entonces es sencillo computar la diferencia entre las mismas respecto a la unidad de tiempo, dando como resultado una estimación de la velocidad del espacio. La Figura 4.8 muestra como funcionaría este tipo de predicción en un ejemplo simplificado en 2D y en el que asumimos que las poses estimadas por el sistema de SLAM/VIO coinciden perfectamente con la trayectoria real del dispositivo.

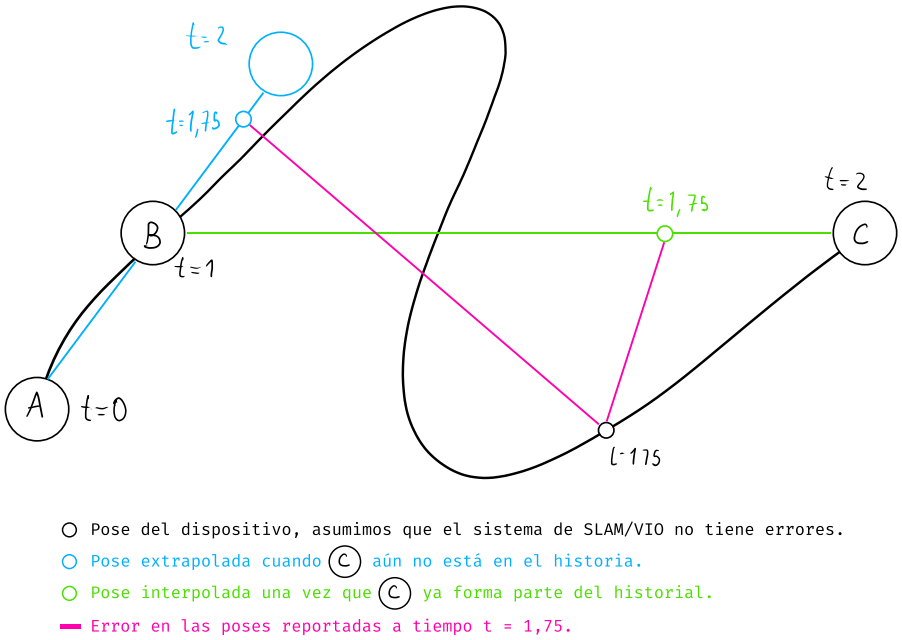


Figura 4.8: Ejemplo de predicción con el historial de espacios. Se asume que las poses estimadas por el sistema de SLAM son perfectas por simplicidad.

Esto es una buena primera solución al problema, y es la opción más básica que la clase adaptadora `TrackerSlam` le ofrece a los usuarios de Monado. Desafortunadamente, si vemos el ejemplo estudiado en la Figura 4.7 notaremos que la frecuencia de poses que se computan es muy baja en comparación a la cantidad de veces que la aplicación OpenXR

requiere una nueva prdicción. En el ejemplo se tenían muestras (y por ende estimaciones) a 30 cuadros por segundo mientras que se renderizaba a 60. En ese ejemplo se está utilizando un monitor de computadora estándar como pantalla, pero en cascos de realidad virtual sin embargo, las frecuencias de renderizado alcanzan fácilmente los 90 o 120 cuadros por segundo, haciendo que la cantidad de predicciones que hacemos entre estimación y estimación crezca. Esto empeora la calidad de las predicciones significativamente, generando movimientos imprecisos y ruidosos, y agravando los efectos de *motion sickness* o *cinetosis*[18] que estas experiencias pueden producir.

Para mejorar nuestras predicciones, visto el problema de que las estimaciones computadas se encuentran muy espaciadas, respecto a las peticiones de predicción, vamos a utilizar las muestras de IMU. Estas vienen usualmente a frecuencias mucho mayores que las de renderizado; 250 Hz en el caso del ejemplo estudiado en la Figura 4.7. Además, a pesar de sufrir de severos problemas de drift, al utilizarlas en ventanas cortas de tiempo (unos pocos milisegundos), estos se ven reducidos en gran medida y la odometría que sus sensores proveen resulta suficientemente precisa.

La clase adaptadora tiene acceso a estas muestras, ya que las intercepta para redirigirlas hacia los sistemas de SLAM. Utilizaremos un concepto similar al de la pre-integración de muestras de IMU explorado en la Sección 3.2.2.1 con algunas ideas relacionadas con las ecuaciones presentadas a partir de la Ecuación 3.1. Consideremos que no vamos a querer interferir de ninguna manera con las estimaciones generadas por los sistemas de SLAM, ya que no querremos distorsionar dichas poses. El proceso de pre-integración correrá de forma completamente aislada de los sistemas. Más aún, será un proceso mayormente efímero que solo tendrá consecuencias sobre puntos de tiempos para los cuales el sistema de SLAM todavía no tenga una estimación posterior. En particular, nos limitaremos a refinar las estimaciones de la velocidad lineal y angular, dejando que las herramientas de predicción de Monado que se basan en el último espacio del historial, computen la pose adecuada para la timestamp requerida. Acumularemos promedios de las mediciones recientes para reducir el impacto del ruido presente en las muestras de la IMU.

Tenemos entonces que el algoritmo utilizado finalmente es similar al pseudo código que se presenta a continuación en el Fragmento 4.2. Cabe aclarar que la función `predict_pose(t)` es llamada cuando el usuario quiere una predicción a tiempo `t`. Además las herramientas de Monado son representadas por `relation_history` (el historial de relaciones) y `predict_from_space` (la función de predicción en base a un espacio).

18 https://es.wikipedia.org/wiki/Cinetosis

Fragmento 4.2: Predicción de poses en Monado

```
struct xrt_space_relation predict_pose(timestamp t) {
   if (relation_history.is_empty()) return {0};

   // Espacio más reciente estimado con SLAM/VIO
   struct xrt_space_relation r = relation_history.get_latest();
   timestamp rt = timestamp_of(r);

   // Variables configuradas por el usuario en tiempo de ejecución
   bool pred_on = /* predicción habilitada por usuario? */
   bool gyr_on = /* giroscopio habilitado por usuario? */
   bool acc_on = /* acelerómetro habilitado por usuario? */
   bool imu_on = gyr_on or acc_on;

   // Flujo condicional de la predicción según la configuración
   if (not pred_on) return r;
   if (not imu_on or t ⩽ rt) relation_history.predict(t);
   if (gyr_on) {
      vec3 avg_gyr = gyr_average_between(rt, t);
      vec3 world_gyr = rotate_angular_velocity(r.orientation, avg_gyr);
      r.angular_velocity = world_gyr;
   }
   if (acc_on) {
      vec3 avg_acc = acc_average_between(rt, t);
      vec3 world_acc = rotate_linear_acceration(r.orientation, avg_acc);
      world_acc += gravity_vector;
      double dt = last_imu_timestamp - rt;
      r.linear_velocity += world_acc * dt;
   }

   return predict_from_space(r, t);
}
```

Es interesante notar la naturaleza modular del algoritmo representada por las condiciones de los `if`, en donde distintos componentes que proveen información a la predicción pueden deshabilitarse. Notar que la velocidad angular provista por el giroscopio es local y necesita ser ajustada a coordenadas globales; esto se realiza con la orientación estimada en `r`, la pose más reciente computada por el sistema de SLAM. Similarmente el acelerómetro debe ser corregido, y a este además se le suma una corrección con el vector de la gravedad[19]. A diferencia del giroscopio, el acelerómetro solo nos puede proveer información sobre los cambios de velocidad, y no sobre la velocidad inicial; para esta usamos la dada por `r` que es computada con la diferencia de las dos poses en `relation_history` más recientes.

19 El vector de gravedad puede computarse dinámicamente con la IMU, detectando momentos en los que el dispositivo está quieto y registrando el vector medido por el acelerómetro.

Mostramos en la Figura 4.9 un ejemplo simplificado del algoritmo implementado utilizando esta idea de promediar muestras de odometría de la IMU para predecir puntos de tiempo posteriores a B cuando C todavía no pertenece al historial de espacios. En el ejemplo se muestra como iría actualizándose el vector promediado (en verde) al integrar las tres muestras (en azul) que ocurren entre $t = 1$ y $t = 2$. Además, se muestra como estos vectores promedios se utilizan para extrapolar linealmente para los tiempos requeridos por un usuario $t \in \{1,45; 1,75; 1,95\}$ (en anaranjado). Por simplicidad, el ejemplo asume que tanto las muestras de odometría de la IMU a tiempos $t \in \{1,3; 1,6; 1,9\}$ como las estimaciones B y C coinciden perfectamente con la trayectoria real del dispositivo.

Para contrastar esto con el caso en el que se ignoran las muestras de la IMU y solo se utiliza el historial de poses, se muestra en la Figura 4.10 los errores de esta predicción para el mismo escenario.

Finalmente, cabe aclarar que la trayectoria presentada en estas figuras es particularmente desafiante. Considerando que estos dispositivos XR están usualmente sujetos a partes como la cabeza o las manos del usuario, es improbable encontrar trayectorias demasiado abruptas entre los tiempos de estimación de las poses de SLAM. Estos, al coincidir con la frecuencia de muestreo de las cámaras, suelen ser menores a 50ms (p.ej. a 30 cuadros por segundo, tenemos ~33ms entre pose y pose). Por otro lado, es usual tener una mayor cantidad de muestras de IMU entre estimaciones y no solo las 3 que se muestran en las figuras (p.ej. a 30 cuadros por segundo con la IMU a 250 hz, tenemos unas ~8 muestras de IMU entre cada par de cuadros consecutivos), lo cual mejora la precisión de la predicción aún más. Este algoritmo de predicción fue implementado en las Contribuciones 14 y 15

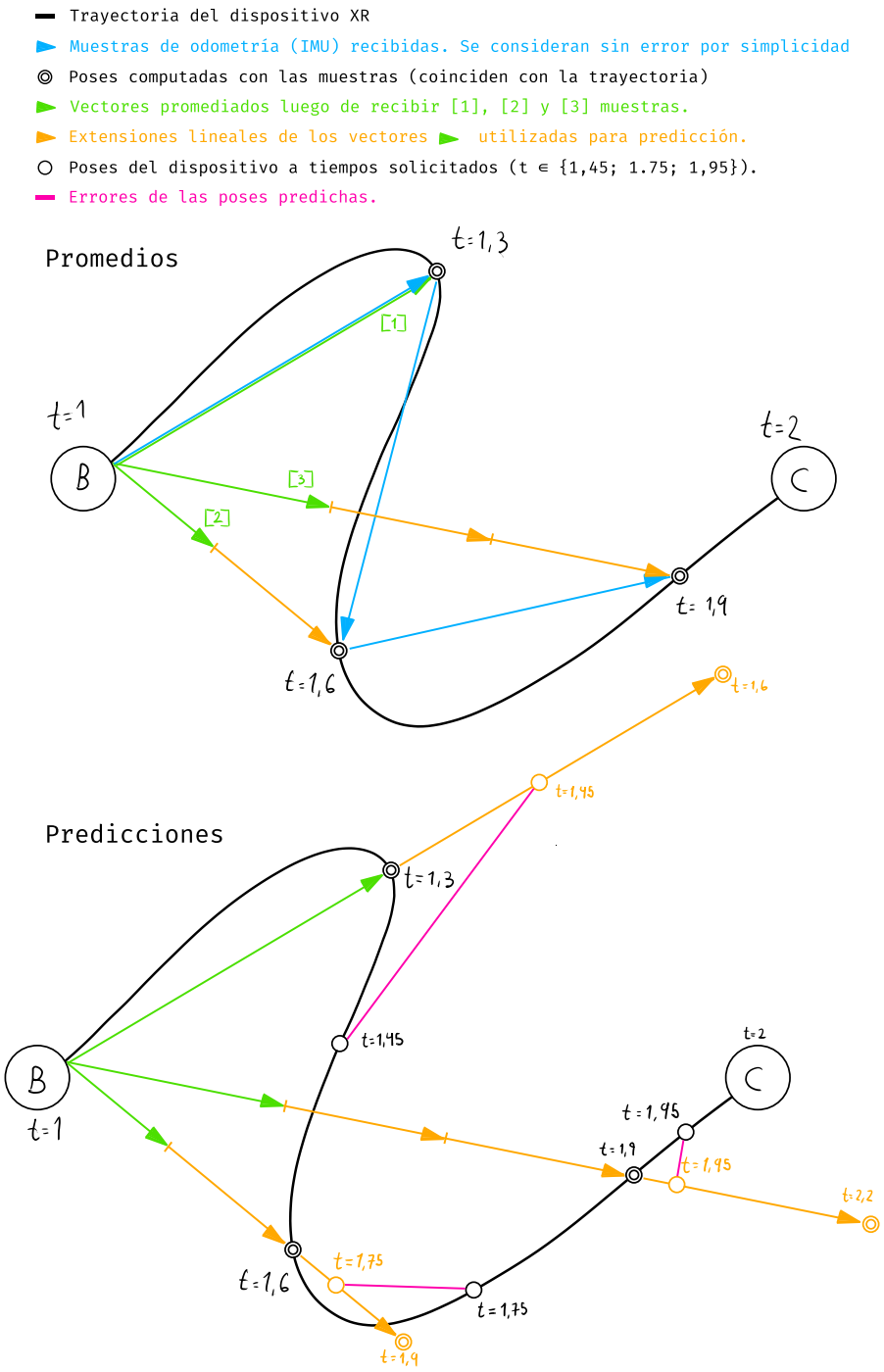
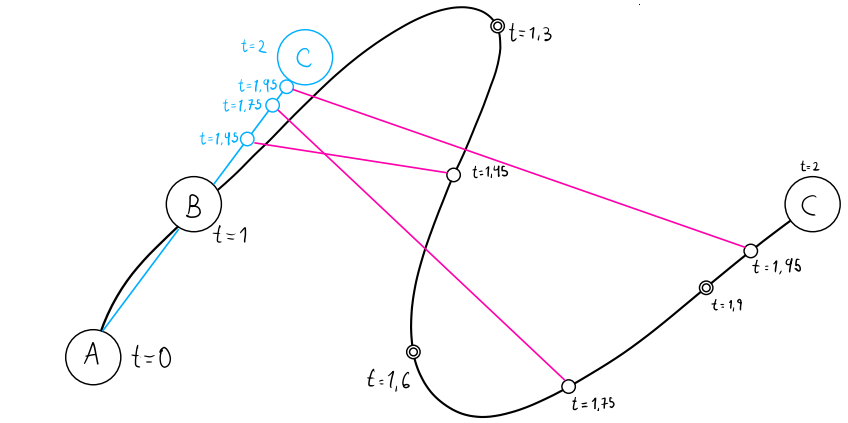


Figura 4.9: Ejemplo de predicción para tiempos $t \in \{1,45; 1,75; 1,95\}$ utilizando la idea de promediar muestras de la IMU posteriores a la pose más reciente del historial (`B` en este caso, se considera que `C` no pertenece al historial aún). Se asume que las poses estimadas por el sistema de SLAM y las muestras de la IMU son perfectas por simplicidad.



Trayectoria del dispositivo XR.
Poses del dispositivo a tiempos solicitados (t ∈ {1,45; 1,75; 1,95}).
Poses predichas utilizando solo el historial de espacios (con últimas muestras A y B).
Errores de las poses predichas.

Figura 4.10: Ejemplo de predicción para tiempos $t \in \{1,45; 1,75; 1,95\}$ ignorando las muestras de IMU y utilizando unicamente el historial de poses con A y B como últimas muestras (esto es antes de que llegue C). De vuelta, asumimos que las estimaciones y no contienen error por simplicidad.

4.5.2 *Filtrado de poses*

Continuando con la descripción de la funcionalidad presente en la clase adaptadora `TrackerSlam` y luego de haber presentado el método de predicción que se emplea en Monado, veremos ahora la siguiente funcionalidad de `TrackerSlam` implementada en la Contribución 15: el **filtrado** de poses.

También llamado *smoothing*, alisado o suavizado, el filtro de señales, curvas o, en este caso poses, es una forma de minimizar o *filtrar* el ruido presente en alguna fuente de información y suavizar los cambios abruptos en las poses. Lo que sucede en nuestro caso es que por la naturaleza del problema en cuestión y por algunas de las decisiones tomadas, se pueden notar la presencia de micro movimientos de alta frecuencia que existen en la trayectoria estimada. Estas alteraciones no son más que ruido proveniente de distintos factores.

Por un lado, los sistemas de SLAM/VIO en el área no necesariamente priorizan el caso de uso de XR y suelen estar mayormente interesados en aplicaciones de robótica. En este campo, la "sensación" que produciría el tracking no es un factor de importancia comparado a la precisión absoluta del mismo. En particular, las métricas más utilizadas en estos trabajos son las que evalúan el error *absoluto* de la trayectoria en su totalidad estimada sobre conjuntos de datos estándar. Es intuitivo pensar que estas métricas, a diferencias de sus versiones *relativas*, incentivan correcciones abruptas en la trayectoria cuando los sistemas determinan que su pose actual no es la mejor posible (p. ej. en momentos de loop closure). El trabajo de Zhang y Scaramuzza [36] es una buena explicación de estas métricas y de la forma de evaluar y reportar el rendimiento de estos sistemas.

De la misma forma, minimizar el ruido en la trayectoria no es uno de los objetivos usualmente considerados, ya que la aplicación artificial de filtros que suavicen la trayectoria podría tender a empeorar el puntaje obtenido en estos conjuntos de datos. Todo esto hace que muchas veces las trayectorias computadas por los sistemas en condiciones no óptimas tiendan a presentar una gran cantidad de movimientos bruscos. Estos pueden resultar particularmente notables en sistemas VR en dónde las pantallas ocupan todo el campo de visión del usuario; movimientos de este tipo pueden fácilmente inducir cinetosis.

Por sobre esto, el método de predicción presentado en la sección anterior en la Figura 4.9 presenta problemas que no fueron tratados. En particular, la predicción que realizamos está basada en una única pose del sistema SLAM y las muestras nueva de la IMU, pero no comparte ningún otro estado en común. Entonces podría esperarse que la diferencia entre las poses que se fueron prediciendo con la nueva pose que el sistema estimará sea significativa. Más aún, en la realidad el error acumulado por la IMU, a pesar de verse limitado a un intervalo corto de unas decenas de milisegundos, es un término más que afectaría

esta diferencia. Todo esto puede verse en el error marcado en la Figura 4.11. Este tipo de desfasaje sería reintroducido de forma repetitiva afectando las predicciones cada vez que una nueva pose es estimada, causando así constantes micro saltos (ruido), especialmente en los momentos en los que la pose estimada por SLAM y la pose predicha por nuestro método difieran significativamente.

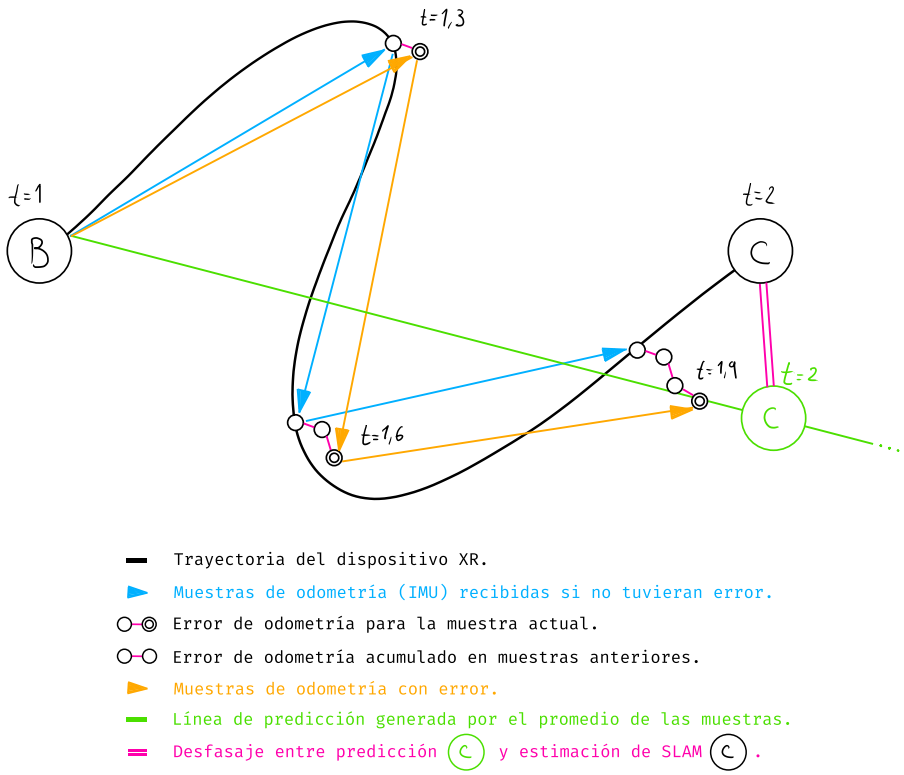


Figura 4.11: Desfasaje esperado entre el método de predicción por muestras promediadas y la pose `C` estimada de SLAM cuando `C` aún no pertenece al historial de espacios. Se muestra la incidencia de la acumulación de errores de la IMU. Este desfasaje causará micro saltos que serían disruptivos en una experiencia de VR.

Es entonces por estas razones, que se implementaron tres métodos sencillos de filtrado combinables para suavizar las trayectorias estimadas y que intentan minimizar los problemas expuestos. Se deja como trabajo futuro el uso de filtros más elaborados como los filtros Kalman [37, ap. B]. Cabe aclarar que en general, el uso de filtros también presenta una desventaja fundamental. Esta es, la introducción de latencia artificial al sistema, ya que para poder suavizar este ruido, los filtros tienden a reducir cambios abruptos en la trayectoria, incluso cuando estos sean realmente los movimientos que el usuario realizó físicamente.

Presentaremos tres filtros, con el más sofisticado basado en Casiez y col. [38]. Además, dicho trabajo cubre el resto de filtros presentados,

muestra gráficas comparativas, y es también una buena lectura para contextualizar el uso de filtros para tracking de movimientos humanos. Los filtros presentados aquí y en el trabajo mencionado pueden visualizarse interactivamente en la aplicación web referenciada al pie de página[20].

El funcionamiento esquemático de un filtro es muy sencillo: es un contenedor de **estado**, con un método de **actualización** que recibe un nuevo **dato** con una **timestamp** posterior a las provistas hasta el momento; con esto, el filtro computa el valor de la **señal filtrada** para una timestamp requerida. En nuestro caso, esta timestamp coincidirá con la del dato de entrada. En Monado, ubicaremos el filtro al final del pipeline de `TrackerSlam`, justo antes de devolverle la pose a la aplicación OpenXR, es decir luego de haber pasado por el procedimiento de predicción. Los filtros habilitados interceptan la pose con sus métodos de actualización respectivos y devuelven en su lugar una nueva pose filtrada con la misma timestamp. Los tres filtros se encuentran uno detrás del otro y son combinables.

Expresado más formalmente, lo que tendremos es un conjunto de poses $X_0, ..., X_k \in \mathbb{R}^7$ que ocurren a tiempos $t_0 < ... < t_k$ respectivamente. Para $i = 0, ..., k$ definimos $X_i = (p_i, q_i)$ con $p_i \in \mathbb{R}^3$ definido por los primeros tres componentes de X_i para representar la posición, mientras que definimos el cuaternión q_i para representar la orientación con coeficientes dados por los últimos cuatro componentes. Tenemos además las versiones filtradas de las poses anteriores $\hat{X}_0, ..., \hat{X}_{k-1}$ y querremos ahora computar la versión filtrada de la pose actual $\hat{X}_k$.

4.5.2.1 *Media móvil*

El primer filtro es una *media móvil*. El estado de este filtro consiste de un historial de poses de los últimos m segundos. El método de actualización registra la pose de entrada y su timestamp en este historial interno. Para computar la pose de salida en esa timestamp, se toma el promedio de las poses de los últimos $w < m$ segundos (por defecto 66 ms). El parámetro w es configurable por el usuario.

Cabe aclarar, que si bien la forma de calcular el promedio para las posiciones p_i debería resultar clara, promediar las orientaciones q_i expresadas mediante cuaterniones no es trivial. En este caso, nos aprovecharemos del hecho de que w suele ser pequeño y por ende contiene orientaciones que no cambian significativamente. Esto nos permite utilizar el resultado expuesto por Gramkow [39] que muestra que, para rotaciones de menos de 40 grados, calcular la media usual (y posteriormente normalizarla) es una muy buena aproximación con un error de menos del 1 %.

20 https://cristal.univ-lille.fr/~casiez/1euro/InteractiveDemo/.

Tenemos entonces que el filtro queda definido por la siguiente ecuación.

$$\hat{X}_k = \frac{1}{|W|} \sum_{i \in W} X_i \quad \text{con} \tag{4.2}$$

$$W = \{i : t_k - w \leq t_i \leq t_k\} \quad \text{(ventana del filtro)} \tag{4.3}$$

Notar que podemos definir el concepto de sumatoria componente a componente en $\mathbb{R}^7$ gracias a la aproximación de los cuaterniones mencionada anteriormente.

4.5.2.2 *Suavizado exponencial*

Este filtro codifica en un único valor los datos históricos e integra nuevos datos con una intensidad dada por un *factor de suavizado* $\alpha \in [0, 1]$ configurable por el usuario (0,1 por defecto). En el caso de tener un único escalar x_k que filtrar, el suavizado exponencial queda definido de esta forma:

$$\hat{x}_0 = x_0 \tag{4.4}$$

$$\hat{x}_k = \alpha x_k + (1 - \alpha)\hat{x}_{k-1} \tag{4.5}$$

Reformulemos la Ecuación 4.5 de la siguiente manera:

$$\hat{x}_k = \alpha x_k + (1 - \alpha)\hat{x}_{k-1} \tag{4.6}$$

$$= \alpha x_k + \hat{x}_{k-1} - \alpha \hat{x}_{k-1} \tag{4.7}$$

$$= \hat{x}_{k-1} + \alpha(x_k - \hat{x}_{k-1}) \tag{4.8}$$

Esto nos deja ver el paso de actualización del filtro como una interpolación (lineal) de $\hat{x}_{k-1}$ hacia x_k con paso α. Utilizaremos esta idea para interpolar esféricamente la orientación de $\hat{X}_k$. Tenemos entonces que el paso de actualización para este filtro queda definido como:

$$\hat{X}_k = (\hat{p}_k, \hat{q}_k) \tag{4.9}$$

$$\hat{p}_k = lerp(\hat{p}_{k-1}, p_k, \alpha) = \hat{p}_{k-1} + \alpha(p_k - \hat{p}_{k-1}) \tag{4.10}$$

$$\hat{q}_k = slerp(\hat{q}_{k-1}, q_k, \alpha) = \hat{q}_{k-1}(\hat{q}_{k-1}^{-1} q_k)^{\alpha} \tag{4.11}$$

4.5.2.3 *Filtro 1€*

El filtro 1€ [38] se basa en el suavizado exponencial, pero utiliza un factor α dinámico que se adapta automáticamente con base a la tasa de cambio de la señal. Reusamos también la idea de interpolar esféricamente para la orientación presentada en el filtro anterior. El filtro queda definido para la posición p_k de la siguiente manera:

$$\hat{p}_0 = p_0 \tag{4.12}$$

$$\hat{p}_k = lerp(\hat{p}_{k-1}, p_k, \alpha) \tag{4.13}$$

Con α que se adapta con la velocidad de la señal:

$$\alpha = \frac{1}{1 + \frac{\tau}{\Delta t_k}} \tag{4.14}$$

$$\Delta t_k = t_k - t_{k-1} \tag{4.15}$$

$$\tau = \frac{1}{2\pi f_C} \tag{4.16}$$

A continuación, f_C es la llamada *frecuencia de corte* [21] y posee un mínimo ajustable por el usuario $f_{C_{min}}$ y un parámetro de intensidad de actualización β también configurable [22].

$$f_C = f_{C_{min}} + \beta|\hat{\dot{p}}_k| \tag{4.17}$$

La velocidad de la señal es a su vez ajustada con otro filtro de suavizado exponencial con factor de suavizado fijo f_{C_d}, también configurable por el usuario.

$$\hat{\dot{p}}_0 = 0 \tag{4.18}$$

$$\hat{\dot{p}}_k = lerp(\hat{\dot{p}}_{k-1}, \dot{p}_k, f_{C_d}) \tag{4.19}$$

Con velocidades instantáneas.

$$\dot{p}_0 = 0 \tag{4.20}$$

$$\dot{p}_k = \frac{p_k - \hat{p}_{k-1}}{\Delta t_k} \tag{4.21}$$

La versión del filtro utilizada para la orientación q_k es análoga a la definición anterior con algunas aclaraciones:

- En lugar de *lerp* se utiliza *slerp*.
- Sobrecargamos el operador de resta de cuaterniones de la siguiente forma: $q_a - q_b = q_b^{-1} q_a$.
- La norma de un cuaternión es equivalente a la norma euclídea en $\mathbb{R}^4$, es decir $|q| = \sqrt{q_x^2 + q_y^2 + q_z^2 + q_w^2}$.

[21] *Algunos lectores reconocerán que el término "frecuencia de corte" proviene de los filtros low-pass y efectivamente, notarán que el filtro 1€ es de este tipo con la particularidad de tener una frecuencia de corte dinámica.*

[22] *Frecuencias de corte bajas reducen el ruido de las poses a costa de aumentar la latencia [40]. La forma en la que f_c es definida resulta en valores altos (y por ende con baja latencia) cuando se presentan cambios significativos en las poses, mientras que para movimientos más suaves se reduce f_c para a su vez disminuir el ruido.*

4.6 RECAPITULANDO

Con la Figura 4.4 vista al principio de la sección en mente, hemos visto que se ha implementado una clase adaptadora `TrackerSlam` en Monado que funciona como punto central de comunicación con los sistemas de SLAM/VIO integrados. Esta clase además implementa funcionalidades de predicción y filtrado de poses que son configurables en runtime.

Los controladores de dispositivos físicos serán los encargados de instanciar y comunicarse con esta clase. Veremos en la siguiente sección los dos controladores (y por ende dispositivos) que necesitaron ser expandidos para este trabajo y que constituyen una contribución importante al runtime.

5

CONTROLADORES EN MONADO

En Monado, la interacción con la gran variedad de dispositivos que el runtime soporta es realizada mediante *drivers* o *controladores*. Estos, como se mostró en la Figura 4.4, le permiten a Monado interactuar con sistemas XR físicos mediante abstracciones derivadas de los requisitos de OpenXR. Un sistema XR en este contexto hace referencia a un conjunto de dispositivos XR. Un dispositivo XR, en forma intuitiva, es algún tipo de hardware que permite la entrada y/o salida de información para intercambio con aplicaciones de XR. Un caso paradigmático de un sistema XR podría considerarse al conjunto de casco y un par de mandos provistos por un fabricante.

El concepto que se termina implementando en Monado es un poco más general, ya que además de sistemas físicos, soporta sistemas simulados que proveen distintas funcionalidades. Entre estas se incluyen la capacidad de conectar dispositivos de forma remota, emulación de dispositivos con teclado y ratón (Contribución 6) o mediante otros dispositivos como placas Arduino[1]. En este trabajo se diferenció el concepto de *fuente de datos* del de *dispositivo* ya que es en definitiva esto en lo que estaremos interesados para SLAM, obtener fuentes de datos de IMU y cámaras.

Se utilizaron los dos dispositivos que se muestran en la Figura 5.1 como principales fuentes de datos para SLAM. A la derecha de la imagen tenemos una cámara de profundidad *Intel RealSense D455*[2] mientras que a izquierda tenemos un casco *Samsung Odyssey+*[3]. La línea de cámaras y módulos RealSense de Intel se enfoca en aplicaciones de robótica y visión por computadora, presentan distintos modelos con múltiples sensores especializados; en nuestro caso nos limitaremos a utilizar su IMU y cámaras estéreo. Estos vienen precalibrados, y además se tiene un SDK[4] de código abierto en C/C++, con *bindings* para otros lenguajes, que facilita la obtención y manipulación de datos. En contraste con esto, el casco de Samsung es un casco ligado a la plataforma privativa *Windows Mixed Reality* (*WMR*)[5] que solo es soportada en sistemas operativos Windows[6]. WMR incluye algoritmos propietarios de tracking por SLAM desarrollados por Microsoft.

Mientras que la cámara D455 funcionó como un dispositivo sumamente versátil para el prototipado y experimentación con sistemas de

1 https://www.arduino.cc/
2 https://www.intelrealsense.com/depth-camera-d455
3 https://www.samsung.com/hk_en/news/product/reality-headset-hmd-odyssey-plus
4 https://github.com/IntelRealSense/librealsense
5 https://www.microsoft.com/en-us/mixed-reality/windows-mixed-reality
6 https://www.microsoft.com/en-us/windows/

Figura 5.1: Dispositivos XR utilizados en este trabajo. A izquierda un casco Samsung Odyssey+ y a derecha una cámara Intel RealSense D455.

SLAM, el Odyssey+ presenta serios desafíos que requirieron trabajo con la comunidad y métodos de ingeniería inversa para poder obtener acceso a las fuentes de datos necesarias para el tracking por SLAM. Cabe aclarar, que anterior a este trabajo, y según mi mejor entendimiento, no existía forma de utilizar este tipo de cascos con tracking basado en SLAM/VIO para correr aplicaciones OpenXR sobre sistemas operativos basados en GNU/Linux [7] [8].

[7] *A pesar de que aún queda mucho por hacer para que el tracking presentado llegue a niveles de calidad comparables a la versión privativa de WMR, la contribución presentada deja asentada en upstream una infraestructura sobre la que extender y mejorar el ecosistema de VR para GNU/Linux.*

[8] *Cuando hablamos de GNU/Linux nos referimos a distribuciones enfocadas a computadoras personales como Ubuntu o Manjaro. Técnicamente, los dispositivos autónomos Oculus Quest de Meta (y otros), corren sobre sistemas operativos basados en Android, que a su vez está basado en GNU/Linux.*

5.1 CARACTERÍSTICAS DE LOS DATOS

Lo primero que se necesita para poder utilizar estos dispositivos para SLAM es conseguir el acceso a los datos que generan; o sea los flujos de imágenes y muestras de IMU. La forma y protocolos necesarios para comunicarse con estos dispositivos se realiza de maneras específicas para cada uno y como veremos en las secciones dedicadas, este es uno de los trabajos fundamentales de un controlador. Además, un controlador tiene que ser capaz, no solo de obtener esta información sino que también de redirigirla a los módulos adecuados de Monado. En la implementación de estas ideas surgen naturalmente los conceptos de *fuentes* (*sources*) y *sumideros* (*sinks*) de datos. En el runtime, estos conceptos existían exclusivamente para el tratado de imágenes, y fueron extendidos para también soportar el flujo de muestras de IMU. Los dispositivos entonces funcionan como fuentes de datos, que instancian un

`TrackerSlam` el cual les provee sumideros a los que redirigir las muestras de sus sensores.

A la hora de implementar un controlador para una familia de dispositivos enfocado a SLAM, hay una serie de cuestiones a tener en cuenta. Desde el punto de vista de sistemas de SLAM/VIO, lo único que nos interesa es un flujo adecuado de imágenes de cámaras y muestras de IMU. Es en la definición de *adecuado* en donde se esconden varios detalles importantes. Intentaremos clarificar los las características que deben considerarse en las muestras para que los sistemas de SLAM puedan utilizarlas apropiadamente:

5.1.1 *Calibración de parámetros intrínsecos*

En primer lugar, es necesario acceder a la información de calibración de los sensores en cuestión, para poder comunicársela de alguna forma a los sistemas de SLAM. Esto incluye la calibración para los cuatro sensores usuales: giroscopio, acelerómetro y el par de cámaras estéreo. La información de calibración describe valores que toman los parámetros *intrínsecos* (propios del sensor) de un modelo de calibración especificado. Es usual que, para evitarle el proceso de calibración al usuario de los sensores, estos provean valores de fábrica. La precisión de tales valores ha probado ser de suficiente calidad para los sistemas de SLAM integrados en este trabajo, pero es necesario aclarar que los valores reales irán cambiando con el tiempo y el desgaste, haciendo que la recalibración sea un punto importante en el uso de estos dispositivos. Monado provee algunas herramientas básicas de calibración para unos pocos modelos de cámara, pero será necesario profundizar en este aspecto como trabajo a futuro.

Habiendo dicho esto, vale aclarar que existen sistemas como OpenVINS [41] que son capaces de realizar el proceso de calibración de forma *online* durante corridas normales. Esto es una característica realmente importante que ninguno de los tres sistemas integrados provee. Todos ellos asumen parámetros de calibración fijos que deben ser provistos previos a la ejecución.

Como intentamos evitarle la calibración manual al usuario, querremos ser capaces de utilizar los valores de fábrica. Para esto es vital que los sistemas de SLAM soporten los mismos modelos de calibración para los que el fabricante especificó los valores[9]. De lo contrario se necesitarán correcciones que usualmente terminan siendo mucho más costosas que si los sistemas soportaran los modelos nativamente. Ejemplos de esto son la *desdistorsión*[10] y *rectificación*[11] de imágenes sobre la cual no nos explayaremos aquí, pero basta con entender que es

[9] *Mientras que en la práctica es conveniente recalibrar con modelos soportados, podría resultar razonable estudiar expresiones analíticas que "traduzcan", en alguna forma significativa, parámetros de un modelo a otro.*

10 Ver `initUndistortRectifyMap` en `https://docs.opencv.org/3.4/da/d54/group__imgproc__transform.html`

11 Ver `stereoRectify` en `https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html`

una transformación costosa aplicada sobre todos los píxeles de las imágenes para "normalizarlas". Esta normalización facilita su uso cuando los sistemas no implementan los modelos de calibración en los que el fabricante comparte los parámetros del dispositivo.

5.1.2 *Calibración de parámetros extrínsecos*

El conjunto de sensores a utilizar: un par de cámaras estéreo y una IMU con acelerómetro y giroscopio, deben estar sujetos a un cuerpo rígido. Es decir, las transformaciones en $SE3$ que describen como alterar la pose de un sensor a otro se mantienen constantes sin importar los movimientos del dispositivo en conjunto. Los valores que describen estas transformaciones relativas son lo que se denominan *parámetros extrínsecos* de la calibración en contraste con los íntrinsecos. Los tres sistemas integrados requieren estos valores antes de comenzar la corrida.

5.1.3 *Sincronización temporal*

Otro problema escencial que debe solucionarse es el de la sincronización de timestamps internas. Cualquier dispositivo que quiera utilizarse para SLAM, debería tener las muestras de todos sus sensores sincronizadas por hardware. Es decir, con timestamps que reflejen lo más precisamente posible los tiempos internos en los que las muestras se tomaron.

Siguiendo con esta línea de problemas de marcas de tiempo, necesitamos un mecanismo de sincronización de timestamps con el host que traiga los relojes del dispositivo y los del host, el cual está corriendo Monado, al mismo *dominio temporal*. Es decir, que una timestamp en uno, represente el mismo momento de tiempo en el otro. Como las aplicaciones OpenXR utilizan el tiempo del host al solicitar predicciones de poses, siempre intentaremos trabajar en ese dominio. Veremos que esta sincronización no es trivial y es una fuente de errores común.

5.1.4 *Muestras de IMU unificadas*

Entrando a problemas que afectan a sensores particulares, todos los sistemas de SLAM estudiados esperan muestras de IMU unificadas que combinen en una única timestamp los valores del acelerómetro y los del giroscopio. En el caso de tener sensores con frecuencias diferentes o timestamps que no coinciden (pero siempre sincronizadas), será necesario realizar algun tipo de transformacion sobre las muestras para unifcarlas.

5.1.5 *Obturador*

Por el lado de los sensores de la cámara, será importante que el obturador o *shutter*[12] de las cámaras sea un *global shutter*, es decir que todos los píxeles sean capturados en el mismo instante. Por el contrario, las cámaras que se utilizan habitualmente en dispositivos de consumo como teléfonos inteligentes, presentan un *rolling shutter* en dónde los píxeles son capturados en un "barrido", fila por fila. Esto genera distorsiones[13] significativas en presencia de movimientos suficientemente rápidos como se ve en la Figura 5.2. Estas distorsiones afectan negativamente la capacidad de los algoritmos de SLAM para reconocer y trackear features de forma estable.

Figura 5.2: Ejemplo de la distorsión que se genera en cámaras con rolling shutter que capturan la imagen barriendo por filas de píxeles. Esto genera distorsiones en los objetos rígidos que son particularmente notables en presencia de movimientos de alta velocidad como los de la hélice. Esto no sucede con cámaras con global shutter que capturan todos los píxeles en el mismo momento.

[12] *El término obturador proviene de los dispositivos mecánicos que usan los sensores ópticos tradicionales. En el contexto analógico, un obturador es una pieza mecánica en movimiento que controla el tiempo durante el cual la película fotográfica es expuesta a la luz de la escena. Para las cámaras que usaremos, el proceso de control de exposición se realiza con interrupciones digitales.*

[13] *Esta distorsión es homónima al tipo de cámara y se denomina el efecto de rolling shutter.*

5.1.6 *Exposición y ganancia*

Otro aspecto muy importante a la hora de presentar muestras de imágenes a sistemas de SLAM es el de utilizar valores adecuados de *exposición* (*exposure*) y *ganancia* (*gain*). La exposición o exposure es la cantidad de tiempo que el obturador de la cámara habilita la entrada de luz a los sensores ópticos por cada cuadro. Por otro lado, la ganancia controla el nivel de amplificación, usualmente digital, que ocurrirá sobre la señal original. El control de estos parámetros, y el de la iluminación del entorno, es vital para asegurar imágenes que posean un brillo adecuado. Imágenes muy oscuras pierden detalles, y por ende la posibilidad de generar features. Imágenes *sobreexpuestas* que tienen de-

masiada iluminación en el entorno y valores de exposición y ganancia altos, presentan el mismo problema al saturar los receptores ópticos, causando que porciones significativas de la imagen se transformen en manchas blancas. Estos problemas pueden verse en la Figura 5.3. Además, los valores de exposición y ganancia afectan el *ruido* y el *motion blur*, esto es la difuminación que se genera por movimientos rápidos en la imagen[14]; ejemplos esto se muestran en la Figura 5.4.

[14] *El motion blur ocurre cuando los movimientos que se realizan modifican sustancialmente la imagen a la que los sensores ópticos están expuestos mientras el obturador sigue abierto.*

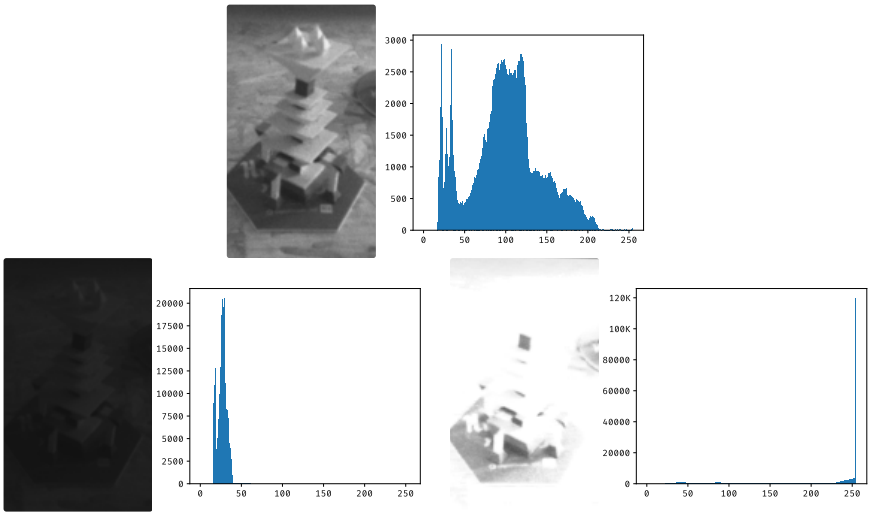


Figura 5.3: Arriba: imagen con valores de exposición y ganancia adecuados a las condiciones de iluminación de la toma. Izquierda: imagen oscura. Derecha: imagen sobreexpuesta. Los histogramas muestran la cantidad de píxeles que toman los valores del eje X de 0 a 255. En las imágenes de abajo hay pérdida de información que se identifica cuando los histogramas se concentran en alguno de los extremos.

Un último problema a considerar que se presenta por el parámetro de exposición, es el llamado *parpadeo* o *flicker*. Este ocurre en entornos iluminados por lámparas artificiales. Estas presentan oscilaciones de intensidad a altas frecuencias que no son visibles a simple vista, pero que si esas frecuencias no están alineadas con las de la exposición de las cámaras, producen una secuencia de imágenes con niveles de brillo intermitentes. Este punto debe tenerse en cuenta, ya que no todos los sistemas son capaces de trackear features de forma eficiente cuando estas cambian las intensidades de sus píxeles.

Para evitar este abanico de problemas, se emplean algoritmos de *ajuste automático* de exposición y ganancia. Estos intentan balancear los valores de estos parámetros en tiempo real; usualmente con técnicas de análisis de histogramas. Hay que ser cuidadoso de que los sistemas empleados soporten tales cambios en la naturaleza de los datos introducidos. En este trabajo se evaluaron algunas posibilidades para emplear algoritmos de este tipo pero ninguna se desarrolló lo suficiente como para realizar una contribución a upstream. Implementar ideas como las propuestas por Zhang y col. [42] es una tarea pendiente como traba-

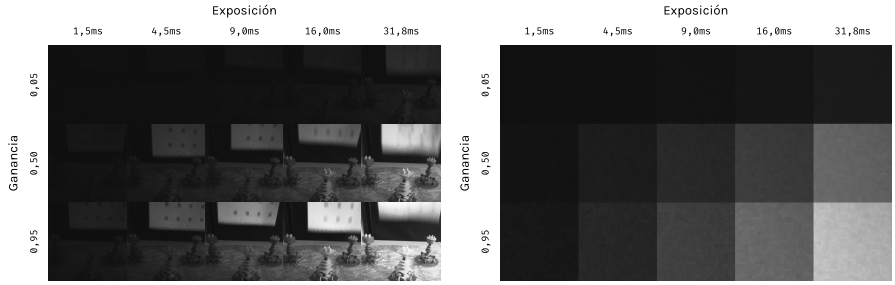


Figura 5.4: Efectos de la exposición y ganancia sobre el ruido y motion blur de la imagen. El ejemplo fue tomados con la cámara D455 con distintas configuracions de exposición y ganancia. A izquierda se observan múltiples capturas de la misma región de 256 píxeles cuadrados que tiene un objeto en movimiento. A derecha tenemos una porción más pequeña de 32 píxeles cuadrados provenientes de las mismas imágenes. Se puede observar en ambas grillas que aumentar ambos valores también incrementa el brillo de la imagen. Por su parte en la figura derecha podemos ver que aumentar la ganancia incrementa el ruido, mientras que en la figura izquierda se observa que valores altos de exposición incrementan el motion blur del objeto en movimiento.

jo a futuro. Por ahora, los controladores recaen en ajustes manuales en los cuales se usa la heurística de que si la imagen se ve razonablemente bien a los ojos del usuario, entonces esta es probablemente adecuada para el sistema de SLAM.

5.1.7 *Comunicación con el dispositivo*

La interfaz de comunicación con el hardware será diferente en cada controlador, pero será usual tener que tratar con hilos consumidores y callbacks asíncronos. En estos habrá colas de datos que deberemos evitar saturar con procesamientos bloqueantes. Para el caso particular del manejo de imágenes, al ser estos recursos de gran tamaño, Monado utiliza mecanismos de *reference counting* mediante su estructura `xrt_frame` para la gestión de memoria. Además, como se mencionó anteriormente, `slam_tracker` utiliza la estructura de datos `cv::Mat` de OpenCV como contenedora de imágenes la cual también provee conteo de referencias. Finalmente, las interfaces con hardware mediante bibliotecas específicas puede traer un contenedor extra de conteo de referencias. Esto puede dar lugar a muchos problemas con el manejo de la memoria y especial cuidado debe tenerse a la hora de adquirir y liberar estos recursos.

5.1.8 *Configuraciones de captura*

Los sensores suelen venir con distintos modos de captura de muestras. Algunos habilitan mayores frecuencias a costa de menor precisión, o mayor precisión a costa de un mayor consumo energético, y otras soluciones de compromiso similares. La capacidad de acceso a la configuración de estos sensores dependerá principalmente de lo que los fabricantes del dispositivo decidan exponer al programador. En caso de existir más de una forma de captura, será necesario poder seleccionar en el controlador la configuración deseada.

5.1.9 *Sistemas de coordenadas*

Finalmente, el último punto que traeremos a atención, es la convivencia de múltiples sistemas de coordenadas que deben unificarse: el de la IMU, el de la implementación de SLAM y el de Monado.

Por un lado, será necesario en algunos casos alinear las muestras de la IMU al sistema de coordenadas de la implementación de SLAM antes de transferirlas. De lo contrario, se pueden presentar situaciones como una implementación que considera que está moviéndose hacia adelante por las muestras ópticas, mientras que las muestras de la IMU le indican que está yendo hacia la derecha; esto causa inconsistencias que suelen terminar en divergencias de los algoritmos de optimización. Además, es usual que también se necesite aplicar un cambio de coordenadas a las poses que el sistema le devuelve a Monado.

5.2 REALSENSE

Estamos ahora en condiciones de entender las contribuciones a controladores realizadas en este trabajo. Comencemos por las del controlador para dispositivos RealSense.

Para soportar la cámara D455, se extendió significativamente en Monado el controlador de dispositivos RealSense en la Contribución 9. Hasta el momento, la única cámara de esta línea soportada por Monado era la T265[15]. Esta cámara es curiosa, ya que presenta un algoritmo de SLAM privativo que corre dentro del dispositivo sin necesidad de interactuar con el host. Este controlador se encargaba únicamente de inicializar el módulo interno de SLAM de la cámara y obtener las poses computadas por la misma para uso en las aplicaciones OpenXR. Uno de sus usuarios clave era el casco libre del proyecto North Star[16] que las utilizaba, en iteraciones anteriores, como principal forma de tracking.

15 https://www.intelrealsense.com/tracking-camera-t265/

16 El proyecto North Star de UltraLeap (prev. LeapMotion) es un casco AR "DIY" que puede fabricarse con piezas impresas en 3D y la compra de algunos componentes. https://developer.leapmotion.com/northstar

En la web pueden encontrarse imágenes[17] que muestran estos cascos con la cámara T265 sujeta en su parte superior.

Al no haber ningún tipo de manejo de las imágenes o muestras de IMU provistas por la cámara, se tuvo que implementar la gestión de estos sensores. Es ahora posible utilizar cualquier cámara de la línea RealSense que posea un par de cámaras estéreo y una IMU. Como trabajo futuro, sería interesante comparar el tracking interno de una T265 con los distintos sistemas externos integrados en este trabajo.

La cámara D455 tiene un par de sensores con líneas de visión paralelas y por ende las dos imágenes comparten la mayoría de sus respectivos campos de visión. Las cámaras tienen una separación de unos 10 cm entre ellas. Presentan imágenes estéreo *prerectificadas*, es decir que las imágenes llegan al sistema de SLAM virtualmente sin ningún tipo de distorsión, esto evita que se necesiten modelos de cámaras particulares implementados en los sistemas externos. Poseen un *obturador global* y el sensor en sí es de *buena calidad* siendo capaz de producir imágenes con buena cantidad de información incluso en situaciones con poca luz y con valores de exposición y ganancia bajos. Cuenta con la capacidad de *ajustar automáticamente* la ganancia y exposición, aunque no pareciera que el algoritmo en uso beneficie a los sistemas de SLAM y por ende se decidió desactivar esta característica para en su lugar configurar estos valores manualmente desde la interfaz gráfica de Monado.

Intel provee un SDK en C/C++ de código abierto y con una licencia permisiva Apache 2.0 [28]. El SDK facilita la interacción entre Monado y las cámaras; se encarga de la gestión de colas y provee estructuras para manejo automático de la memoria de los cuadros recibidos. Con el SDK es posible configurar múltiples formas de ejecución de los sensores. El modelo D455 soporta frecuencias de hasta 90 fps y resoluciones de hasta 1280x720 píxeles en cada cámara, mientras que los sensores que conforman la IMU se encuentra explicitamente divididos en acelerómetro que puede ejecutarse a 60hz y 250hz y giroscopio que logra alcanzar frecuencias de 200hz y 400hz.

Los *relojes de la IMU y cámaras* están sincronizados por hardware, pero es necesario aplicar un parche al kernel de la versión de Linux instalada lo cual dificulta el uso. Sin este parche, se intenta utilizar los tiempos de arribo al host de las muestras, pero es usual que estos causen que los sistemas divergan por lo poco preciso que son. Una vez que se tiene este parche, el SDK estima automáticamente la sincronización entre los *relojes del host y de la cámara* y por ende es capaz de traducir timestamps de un dominio temporal al otro y viceversa.

5.3 WINDOWS MIXED REALITY

La integración realizada en la Contribución 13 con el casco Samsung Odyssey+ de la plataforma Windows Mixed Reality fue, en compara-

17 https://www.collabora.com/assets/images/blog/ProjectNorthStar.jpg

ción a la cámara RealSense D455, desafiante. Este casco no soporta Linux por defecto y el controlador WMR es desarrollado y mantenido por miembros de la comunidad de Monado. El soporte de características básicas de estos dispositivos necesita un análisis cuidadoso de paquetes USB obtenidos mediante capturas realizadas en sesiones de uso en Windows, el único sistema operativo soportado oficialmente. Con este proceso de ingeniería inversa se logró entender cómo activar los sensores y cómo leer sus muestras así como también la posibilidad de configurar parámetros como la ganancia y exposición de las cámaras.

El dispositivo cuenta internamente con un bloque de configuración que se puede decodificar a un archivo JSON. Este presenta los parámetros de calibración de los sensores. En particular, presenta parámetros para un modelo de calibración de las cámaras que no es muy usual que los sistemas de SLAM soporten, este es el modelo radial-tangencial [43] de 8 parámetros. Inicialmente se recalibró el dispositivo a un modelo más usual (Kannala-Brandt [44] de 4 parámetros) pero eventualmente se decidió extender y añadir a Basalt el modelo de 8 parámetros en la Contribución 21. Otras peculiaridades de estas cámaras es que no son paralelas como en el caso de la D455, sino que tienen un gran ángulo de separación haciendo que solo la mitad de ambas imágenes posean una zona de visión compartida. Se describen estos problemas y posibles soluciones en más detalle en la Discusión 19.

Otra particularidad es que las muestras de IMU no están precalibradas y por ende deben calibrarse en tiempo de ejecución. Por suerte esta característica es soportada por Basalt, pero no por los otros dos sistemas. Un punto en el que el casco fue más conveniente que la cámara RealSense es que se puede acceder a las timestamps sincronizadas de la IMU y las cámaras sin necesidad de aplicar ningún parche en el kernel del usuario. Para poder traducir timestamps desde el reloj del dispositivo al reloj del host, se utilizó un filtro de suavizado exponencial sencillo para estimar el offset entre los dos relojes que parece funcionar suficientemente bien.

Las cámaras son monocrómaticas con obturador global y presentan cuadros con una resolución fija de 640x480 píxeles a 30 fps. La IMU reporta 250 mensajes por segundo y cada uno de estos contiene 4 muestras del sensor; 1000 Hz. En el controlador se decidió promediar estas 4 muestras para tener así una frecuencia efectiva de 250 Hz. El sensor utilizado parecer ser de menor calidad que la D455, ya que presenta imágenes notablemente más ruidosas y requiere valores más altos de exposición y ganancia para lograr imágenes con brillo suficiente. La comunicación con el hardware ocurre mediante comandos USB directos construidos con la ayuda de `libusb`[18] y basados en la ingeniería inversa aplicada sobre el controlador oficial.

18 https://libusb.info

Parte IV

CONCLUSIONES

Para cerrar presentaremos algunos resultados que intentan describir el rendimiento y la precisión logrados en la implementación actual. Concluiremos con una revisión general de los temas tratados y posibles líneas de trabajo a considerar para el futuro.

6

RESULTADOS

El proceso de evaluar y obtener métricas en sistemas de SLAM/VIO requiere de ciertas consideraciones. A grandes rasgos, podemos dividir las métricas de interés usuales en medidas de *precisión* y de *eficiencia* que describen, respectivamente, la exactitud de la trayectoria estimada y el uso de recursos por parte de los sistemas. Para la evaluación de sistemas se desarrollaron funcionalidades y herramientas dedicadas a la evaluación de sistemas de SLAM en Monado; ver Contribuciones 5, 7, 12, 16 y 17. Para más información sobre evaluación que la que presentaremos en esta sección, referimos al lector al trabajo de Kümmerle y col. [45] que detalla en mayor profundidad el proceso de evaluación y a la suite de herramientas SLAMBench[1] [46] que intenta generalizarlo para una gran variedad de sistemas.

Para la evaluación se utilizan *conjuntos de datos* o *datasets* pregrabados con distintos dispositivos. Utilizaremos dos datasets populares en el área: *EuRoC* [26] que es grabado con un *vehículo micro aéreo* (*MAV* o *drone*) y *TUM-VI* [31] con muestras provenientes de un dispositivo que es sostenido con la mano (*handheld*) lo cual lo hace particularmente bueno para evaluar tracking en aplicaciones de XR. Además, estos conjuntos de datos fueron tomados con sistemas de captura de movimiento (*MoCap*) externos de gran precisión pero también de gran costo. Esto les permite presentar una trayectoria muy precisa que se utiliza como punto de referencia y se la conoce como *ground truth*.

Además de estos datasets, se presentan datos tomados especialmente para este trabajo[2] con los dispositivos introducidos en el Capítulo 5: la cámara RealSense D455 y el casco Odyssey+. Se tienen también datos monoculares del celular móvil Poco X3 Pro capturados con el trabajo de Huai y col. [47] pero no llegaron a utilizarse en estos resultados; es decir, todos los datos evaluados son con cámaras estéreo. Estos conjuntos capturados con la D455 y el Odyssey+ no presentan ground truth, pero ofrecen grabaciones especialmente pensadas para XR y utilizan dispositivos que Monado ya soporta. En la figura Figura 6.1 se pueden observar algunas imágenes de los distintos datasets utilizados en al evaluación.

Como es usual en este tipo de estudios comparativos, utilizaremos acrónimos para referirnos a los distintos conjuntos de datos con las siguientes características:

1 https://apt.cs.manchester.ac.uk/projects/PAMELA/tools/SLAMBench

2 https://drive.google.com/drive/folders/163KuF88viW_wPcVNZJ2Onxe7zHf2Qo7L?usp=sharing

Figura 6.1: Imágenes para visualizar el tipo de entorno en el que los datasets de evaluación fueron capturados.

- C*: Datasets específicos para este trabajo (*custom*). Cada uno presenta dos modalidades, la primera EASY tiene movimientos tranquilos similares a los que se verían al inspeccionar una habitación. Por otro lado la modalidad HARD contiene una sucesión de movimientos agitados e intenta simular momentos de acción en un juego. Los sensores utilizados son:
  - C6*: RealSense D455 en 640x480 a 30 fps e IMU a 250 Hz.
  - C8*: RealSense D455 en 848x480 a 60 fps e IMU a 250 Hz.
  - CO*: Odyssey+ en 640x480 a 30 fps e IMU a 250 Hz. Para Basalt en particular tenemos dos posibles modelos de cámara para utilizar con los lentes de este casco. Por defecto se utilizará el modelo radial-tangencial de 8 parámetros dado de fábrica e implementado en la Contribución 21, pero también se comparará con el modelo Kannala-Brandt de 4 parámetros (KB4) recalibrado y nativo en Basalt.
- E*: Los datasets EuRoC en dos habitaciones (V1 y V2) y una sala de máquinas (MH) en 752x480 a 20 fps e IMU a 200 Hz.
- T*: TUM-VI en una habitación (R) en 512x512 a 20 fps e IMU a 200 Hz.

También utilizaremos acrónimos para los distintos sistemas evaluados, que son variantes de Basalt, Kimera y ORB-SLAM3 dados actualizaciones significativas que le ocurrieron a Basalt[3] y ORB-SLAM3[4] durante el desarrollo de este trabajo.

- K: Kimera-VIO
- OO: ORB-SLAM3 original antes de la versión 1.0.
- ON: ORB-SLAM3 nueva versión 1.0.

3 https://gitlab.com/VladyslavUsenko/basalt/-/commit/24325f2a
4 https://github.com/UZ-SLAMLab/ORB_SLAM3/releases/tag/v1.0-release

- BO: Basalt original.
- BNF: Basalt nuevo luego de la actualización presentada en Demmel y col. [21] la cual incluye, entre otras cosas, la posibilidad de especificar la precisión de punto flotante utilizada. Esta versión utiliza `float`.
- BND: Idéntico al punto anterior pero con precisión `double`.

Todas las corridas fueron realizadas sobre un procesador Intel i7-1065G7 con consumo de hasta 15 W y memoria suficiente; ninguno de los sistemas hace uso de GPU.

6.1 COMPLETITUD

Nuestro primer análisis se encuentra en la Tabla 6.1 y se centra en la capacidad de los sistemas integrados de no presentar fallas fatales durante las corridas de los conjuntos de datos probados. Esta métrica muestra la tolerancia a fallas de las distintas implementaciones. La tabla presenta a Basalt como el más estable mientras que ORB-SLAM3 presenta algunas fallas ocasionales. Kimera por el otro lado presenta severas fallas incluso en conjuntos estándares como TUM-VI. Vale la pena aclarar que en la práctica, si bien Basalt resulta más estable que los otros sistemas, existen situaciones que pueden hacerlo fallar, particularmente ante la presencia de muestras de baja calidad durante tiempos prolongados. Es notable que la introducción de los datasets C* es particularmente desafiante para la estabilidad de los sistemas en comparación a los conjuntos estándares EuRoC y TUM-VI para los cuales las implementaciones suelen estar bien probados y configurados por defecto.

Tabla 6.1: Completitud de ejecución

DATASET	BND	BNF	BO	K	ON	OO
C6EASY	✓	✓	✓	✓	✓	✓
C6HARD	✓	✓	✓	35.89 %	✓	✓
C8EASY	✓	✓	✓	✓	✓	✓
C8HARD	✓	✓	✓	52.25 %	56.61 %	55.86 %
COEASY	✓	✓	✓	✓	✓	✓
(KB4)	✓	✓	✓			
COHARD	✓	✓	✓	✓	✓	95.71 %
(KB4)	✓	✓	✓			
EMH01	✓	✓	✓	✓	✓	✓
EMH02	✓	✓	✓	✓	✓	✓
EMH03	✓	✓	✓	✓	✓	✓
EMH04	✓	✓	✓	✓	✓	✓
EMH05	✓	✓	✓	✓	✓	96.48 %
EV101	✓	✓	✓	✓	✓	✓
EV102	✓	✓	✓	✓	✓	✓
EV103	✓	✓	✓	✓	✓	✓
EV201	✓	✓	✓	✓	✓	✓
EV202	✓	✓	✓	✓	✓	✓
TR1	✓	✓	✓	40.25 %	✓	✓
TR2	✓	✓	✓	38.32 %	✓	✓
TR3	✓	✓	✓	✓	✓	✓
TR4	✓	✓	✓	63.58 %	✓	✓
TR5	✓	✓	✓	52.67 %	74.81 %	✓
TR6	✓	✓	✓	52.37 %	✓	✓
Media	**100 %**	**100 %**	**100 %**	**83.42 %**	**96.88 %**	**97.64 %**

6.2 TIEMPOS

En este trabajo nos limitaremos a presentar el tiempo promedio que le toma a cada sistema devolver la estimación de la pose. Se plantea como trabajo a futuro la posibilidad de automatizar la medición del consumo de otros recursos como memoria, energía y capacidad de cómputo. Recordar que los conjuntos de datos probados presentan cuadros a 20, 30 y 60 fps respectivamente y para lograr tracking a tiempo real necesitaríamos tiempos de estimación menores a 50, 33 y 16 ms respectivamente.

Viendo la Tabla 6.2 vemos que Basalt (BNF) sale ganador con tiempos bien por debajo de los 16 ms. La versión BND se presenta como cu-

riosidad y muestra lo ineficiente que es el pipeline cuando se utilizan números `double`; como veremos en las siguientes tablas el tracking de BNF y BND dan los mismos resultados de precisión gracias a la técnica introducida en la actualización [21]. ORB-SLAM3 y Kimera tienen tiempos de ejecución significativamente mayores indicando que convendría utilizar sensores a menor frecuencia para estos sistemas si se quieren evitar congestiones. Es necesario aclarar que los tiempos de ORB-SLAM3 podrían ser mejorados si se evitara utilizar las configuraciones que vienen por defecto. De la misma manera, la construcción del mapa en tiempo real de ORB-SLAM3 es pesada y en la nueva versión (ON) soporta la capacidad construirlo de antemano para luego sólo ejecutar los módulos de localización y no de mapeo, reduciendo así considerablemente los tiempos de cómputo. Probar estas configuraciones con ORB-SLAM3 se deja como trabajo a futuro.

Tabla 6.2: Tiempos de ejecución medio por cuadro [ms]

DATASET	BND	BNF	BO	K	ON	OO
C6EASY	826.60 ± 441.30	5.84 ± 1.38	9.05 ± 2.40	46.00 ± 6.10	36.11 ± 7.67	35.09 ± 11.81
C6HARD	668.75 ± 555.59	5.58 ± 1.38	9.83 ± 3.01	47.45 ± 7.94	30.66 ± 9.64	32.92 ± 12.41
C8EASY	940.54 ± 485.31	6.93 ± 2.10	12.89 ± 13.63	49.23 ± 7.37	33.69 ± 10.50	33.24 ± 10.22
C8HARD	716.58 ± 579.89	6.20 ± 2.46	12.60 ± 8.38	46.28 ± 7.89	35.83 ± 11.75	37.43 ± 12.04
COEASY	873.74 ± 404.36	6.17 ± 1.12	10.96 ± 2.94	37.25 ± 4.94	35.40 ± 9.35	29.41 ± 9.69
(KB4)	734.47 ± 357.71 K	6.24 ± 1.02 K	10.92 ± 2.98 K			
COHARD	617.47 ± 419.17	5.71 ± 1.16	12.90 ± 3.83	37.31 ± 5.20	21.52 ± 7.61	23.69 ± 7.98
(KB4)	592.62 ± 410.55 K	5.81 ± 1.02 K	12.81 ± 3.81 K			
EMH01	2123.22 ± 1118.05	10.63 ± 3.22	14.17 ± 3.15	53.15 ± 7.00	30.29 ± 6.63	36.73 ± 12.68
EMH02	2280.38 ± 1118.44	11.16 ± 4.28	15.33 ± 5.76	53.93 ± 6.05	29.29 ± 5.37	35.32 ± 10.40
EMH03	2184.08 ± 940.11	11.02 ± 2.81	15.17 ± 3.65	53.83 ± 6.05	32.09 ± 5.71	37.08 ± 12.91
EMH04	2117.07 ± 916.61	11.82 ± 3.73	15.83 ± 3.41	53.12 ± 7.19	29.77 ± 6.94	32.67 ± 11.86
EMH05	2187.28 ± 902.72	11.17 ± 2.15	15.47 ± 3.63	53.40 ± 7.07	29.04 ± 6.17	34.06 ± 15.61
EV101	1687.89 ± 524.66	10.23 ± 1.76	13.62 ± 2.21	54.37 ± 6.18	30.26 ± 5.95	35.43 ± 13.93
EV102	1322.72 ± 624.59	10.18 ± 2.09	15.35 ± 3.63	55.48 ± 5.79	29.74 ± 6.06	32.71 ± 13.47
EV103	844.55 ± 609.03	11.65 ± 2.56	17.31 ± 4.37	56.54 ± 6.47	34.74 ± 11.43	31.13 ± 10.70
EV201	1628.73 ± 718.45	10.08 ± 1.89	15.53 ± 3.04	55.00 ± 5.66	36.63 ± 11.87	32.51 ± 10.04
EV202	1296.74 ± 667.75	10.65 ± 3.49	17.57 ± 4.14	55.37 ± 5.24	37.77 ± 10.90	34.78 ± 11.45
TR1	800.61 ± 327.45	6.37 ± 1.02	12.54 ± 2.70	21.34 ± 3.51	46.72 ± 11.81	44.95 ± 12.33
TR2	767.92 ± 287.46	6.08 ± 0.93	11.47 ± 2.37	21.42 ± 2.53	44.78 ± 12.06	43.94 ± 13.21
TR3	697.36 ± 285.12	5.96 ± 0.93	11.93 ± 2.47	23.31 ± 3.69	38.33 ± 9.31	41.27 ± 11.67
TR4	857.84 ± 330.19	6.57 ± 1.19	11.74 ± 2.38	22.62 ± 6.31	39.54 ± 10.50	42.37 ± 11.85
TR5	694.90 ± 308.46	6.09 ± 1.01	12.53 ± 2.98	20.44 ± 3.01	32.79 ± 5.37	42.42 ± 11.87
TR6	1007.87 ± 269.40	7.00 ± 1.05	10.72 ± 1.80	22.33 ± 5.05	33.19 ± 6.98	44.83 ± 12.68
Media	**1186.25 ± 566.77**	**8.13 ± 1.91**	**13.26 ± 3.86**	**42.69 ± 5.74**	**34.01 ± 8.62**	**36.09 ± 11.86**

6.3 PRECISIÓN DE LA TRAYECTORIA

Para medir la precisión de la trayectoria completa estimada por el sistema se la compara con las trayectorias ground truth provistas por los datasets. La métrica usual para representar esta precisión es la *raíz del error cuadrático medio* ($RMSE$) del *error de trayectoria absoluto* (ATE) que se define de la siguiente manera [48].

Una trayectoria será una secuencia de poses $P_i \in SE(3)$ con i siendo el tiempo o timestamp en el que la pose ocurrió. Tenemos dos trayectorias que considerar; la estimada que denotaremos con P_i^{est} y la de referencia o groud truth que denotaremos con P_i^{ref} con $i = N$ la última timestamp.

El ATE_i entre cada pose P_i^{est} y P_i^{ref} al momento i es la distancia entre las posiciones de las poses es decir:

$$ATE_i = ||pos(P_i^{est}) - pos(P_i^{ref})|| \tag{6.1}$$

Con $pos(P) \in \mathbb{R}^3$ el componente de posición o traslación de la pose P. Notar que no se considera el componente de rotación de las poses. Finalmente utilizamos el RMSE de los ATE al cuadrado:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} ATE_i^2} \tag{6.2}$$

Esta es la métrica que se utiliza en esta sección para medir la precisión en las trayectorias estimadas. Cabe aclarar además que las poses dadas por la ground truth y las dadas por las estimaciones, en general, no van a coincidir en sus timestamps y se debe utilizar alguna forma de relacionarlas. Este trabajo utiliza la biblioteca EVO [49] y por defecto esta simplemente utiliza la trayectoria con menos poses y las relaciona a las poses con timestamps más cercanas de la otra trayectoria. Otro detalle en el proceso que es estándar en la práctica es alinear previamente las dos trayectorias minimizando con cuadrados mínimos el error de las mismas con el trabajo de Umeyama [50].

Ahora sí, podemos ver los resultados en la Tabla 6.3. En general Basalt es también superior en estas métricas, pero aquí hay una aclaración importante que hacer. ORB-SLAM3 es usualmente considerado el estado del arte porque las métricas que reporta ocurren luego de que los datasets hayan terminado de procesarse. Esto hace que el mapa utilizado sea el final ya construido, lo cual afecta retroactivamente a las poses computadas anteriormente. En este caso como estamos corriendo ORB-SLAM3 en tiempo real y utilizamos las poses que reporta inmediatamente con el mapa en construcción, estas son mucho peores. Se deja como trabajo próximo la evaluación de ORB-SLAM3 con el mapa pre construido, creemos que en este caso deberían verse valores más parecidos a los reportados por la publicación original del sistema y probablemente sobrepasen a Basalt. También hay que notar que todos los

datasets tienen duraciones menores a 5 minutos, esto hace que el desvío que se acumula en sistemas como Basalt que no tienen noción de mapa global no se note tanto. Con esto queremos decir que es usual en Basalt notar luego de sesiones de uso más largas que el entorno simulado empieza a cambiar de lugar lentamente a causa de la deriva (*drift*) usuales en sistemas de VIO y no de SLAM. Se plantea también como trabajo a futuro mejorar este aspecto de Basalt (ver Discusión 18).

Tabla 6.3: Error absoluto de la trayectoria (ATE) [m]

DATASET	BND	BNF	BO	K	ON	OO
EMH01	0.061 ± 0.023	0.061 ± 0.023	0.087 ± 0.026	0.290 ± 0.568	0.173 ± 0.230	0.216 ± 0.306
EMH02	0.043 ± 0.022	0.043 ± 0.022	0.049 ± 0.023	0.127 ± 0.051	0.151 ± 0.133	0.627 ± 0.811
EMH03	0.059 ± 0.019	0.059 ± 0.019	0.075 ± 0.039	0.192 ± 0.056	1.797 ± 1.175	2.513 ± 1.797
EMH04	0.107 ± 0.038	0.107 ± 0.038	0.099 ± 0.040	0.188 ± 0.081	0.815 ± 0.517	2.065 ± 1.132
EMH05	0.139 ± 0.041	0.139 ± 0.041	0.120 ± 0.041	0.206 ± 0.071	1.797 ± 0.785	3.537 ± 1.868
EV101	0.040 ± 0.017	0.040 ± 0.017	0.040 ± 0.016	0.071 ± 0.027	9.842 ± 10.408	0.179 ± 0.168
EV102	0.043 ± 0.013	0.043 ± 0.013	0.053 ± 0.019	0.093 ± 0.039	0.600 ± 0.359	0.951 ± 0.393
EV103	0.049 ± 0.020	0.049 ± 0.020	0.067 ± 0.026	0.182 ± 0.050	13.274 ± 9.972	0.127 ± 0.105
EV201	0.036 ± 0.015	0.036 ± 0.015	0.031 ± 0.017	0.046 ± 0.024	0.141 ± 0.130	0.098 ± 0.096
EV202	0.045 ± 0.021	0.045 ± 0.021	0.060 ± 0.022	0.120 ± 0.041	0.323 ± 0.351	0.471 ± 0.248
TR1	0.096 ± 0.048	0.096 ± 0.048	0.093 ± 0.042	4264.6 ± 2534.0	0.081 ± 0.028	0.546 ± 0.567
TR2	0.067 ± 0.040	0.067 ± 0.040	0.062 ± 0.030	4447.9 ± 2728.6	0.087 ± 0.075	0.061 ± 0.082
TR3	0.110 ± 0.057	0.110 ± 0.057	0.123 ± 0.063	6916.5 ± 4071.1	0.076 ± 0.032	0.123 ± 0.127
TR4	0.050 ± 0.029	0.050 ± 0.029	0.049 ± 0.022	4918.2 ± 2749.4	0.105 ± 0.059	0.211 ± 0.175
TR5	0.160 ± 0.067	0.160 ± 0.067	0.121 ± 0.051	5417.1 ± 2905.8	0.159 ± 0.122	0.112 ± 0.086
TR6	0.018 ± 0.011	0.018 ± 0.011	0.018 ± 0.009	5003.9 ± 2511.9	0.105 ± 0.059	0.122 ± 0.168
Media	**0.070 ± 0.030**	**0.070 ± 0.030**	**0.072 ± 0.030**	**1935.6 ± 1093.8**	**1.845 ± 1.527**	**0.747 ± 0.508**

6.4 PRECISIÓN DE LOS MOVIMIENTOS

Finalmente presentamos en la Tabla 6.4 el RMSE del *error relativo de la trayectoria* (*RTE*). Este error es capaz de representar las imprecisiones en tramos cortos de la trayectoria. Una forma de pensar esto en el contexto de XR es qué tan mal se sentirán los movimientos individuales realizados por un usuario.

Para definir este error primero dividimos la trajectoria P_k en vectores definidos entre pares de timestamps i y j:

$$\delta_{ij} = pos(P_j) - pos(P_i) \in \mathbb{R}^3 \tag{6.3}$$

Luego, definimos el error de traslación entre las timestamps i y j como:

$$RTE_{ij} = ||\delta_{ij}^{ref} - \delta_{ij}^{est}|| \tag{6.4}$$

Y finalmente el RMSE RTE queda definido como:

$$RMSE = \sqrt{\frac{1}{N} \sum_{\forall i,j} RTE_{ij}^2} \tag{6.5}$$

Notar que aquí la elección de que tan largo son los vectores δ_{ij} puede variar. En nuestro caso, usando EVO, utilizamos vectores con timestamps separadas por el equivalente tiempo a 6 cuadros de cada dataset, es decir unos 0.3, 0.2, y 0.1 segundos para 20, 30 y 60 fps.

Tabla 6.4: Error relativo de la trayectoria (RTE, intervalos de 6 cuadros) [m]

DATASET	BND	BNF	BO	K	ON	OO
EMH01	0.004 ± 0.003	0.004 ± 0.003	0.004 ± 0.003	0.069 ± 0.283	0.138 ± 0.113	0.137 ± 0.110
EMH02	0.004 ± 0.002	0.004 ± 0.002	0.004 ± 0.003	0.019 ± 0.019	0.140 ± 0.094	0.147 ± 0.167
EMH03	0.009 ± 0.008	0.009 ± 0.008	0.010 ± 0.008	0.038 ± 0.030	0.368 ± 0.398	0.385 ± 0.460
EMH04	0.010 ± 0.008	0.010 ± 0.008	0.011 ± 0.009	0.043 ± 0.031	0.335 ± 0.281	0.341 ± 0.392
EMH05	0.009 ± 0.006	0.009 ± 0.006	0.010 ± 0.007	0.041 ± 0.030	0.307 ± 0.308	0.365 ± 0.660
EV101	0.011 ± 0.006	0.011 ± 0.006	0.011 ± 0.006	0.044 ± 0.024	0.222 ± 1.958	0.136 ± 0.080
EV102	0.011 ± 0.005	0.011 ± 0.005	0.011 ± 0.005	0.040 ± 0.022	0.277 ± 0.183	0.276 ± 0.188
EV103	0.011 ± 0.007	0.011 ± 0.007	0.014 ± 0.009	0.039 ± 0.025	0.358 ± 2.249	0.246 ± 0.173
EV201	0.003 ± 0.002	0.003 ± 0.002	0.003 ± 0.002	0.015 ± 0.012	0.092 ± 0.064	0.097 ± 0.081
EV202	0.007 ± 0.006	0.007 ± 0.006	0.012 ± 0.025	0.025 ± 0.018	0.219 ± 0.148	0.221 ± 0.160
TR1	0.007 ± 0.005	0.007 ± 0.005	0.008 ± 0.006	384.484 ± 305.665	0.505 ± 0.288	0.524 ± 0.294
TR2	0.006 ± 0.005	0.006 ± 0.005	0.007 ± 0.006	468.756 ± 475.490	0.492 ± 0.421	0.503 ± 0.421
TR3	0.005 ± 0.004	0.005 ± 0.004	0.006 ± 0.005	262.503 ± 201.940	0.618 ± 0.488	0.624 ± 0.486
TR4	0.005 ± 0.005	0.005 ± 0.005	0.005 ± 0.005	342.893 ± 179.226	0.295 ± 0.161	0.300 ± 0.164
TR5	0.009 ± 0.007	0.009 ± 0.007	0.010 ± 0.008	341.326 ± 155.828	0.477 ± 0.284	0.483 ± 0.285
TR6	0.003 ± 0.002	0.003 ± 0.002	0.003 ± 0.002	355.299 ± 219.485	0.268 ± 0.214	0.275 ± 0.227
Media	**0.007 ± 0.005**	**0.007 ± 0.005**	**0.008 ± 0.007**	**134.727 ± 96.133**	**0.319 ± 0.478**	**0.316 ± 0.272**

6.5 RESULTADOS ESPECÍFICOS

Presentamos a continuación algunos gráficos para dar una idea cualitativa del funcionamiento de los sistemas. Las figuras fueron obtenidas utilizando BNF, ON y K sobre el dataset EV201 que es correctamente procesado por las tres implementaciones. La Figura 6.2 presenta una comparación de tiempos de procesamiento. La Figura 6.3 muestra la forma general de las trayectorias estimadas mientras que la Figura 6.4 presenta algunos detalles sobre ORB-SLAM3. Por último se pueden ver los valores de ATE_i en los distintos tiempos de la trayectoria de Basalt (BNF) en la Figura 6.5.

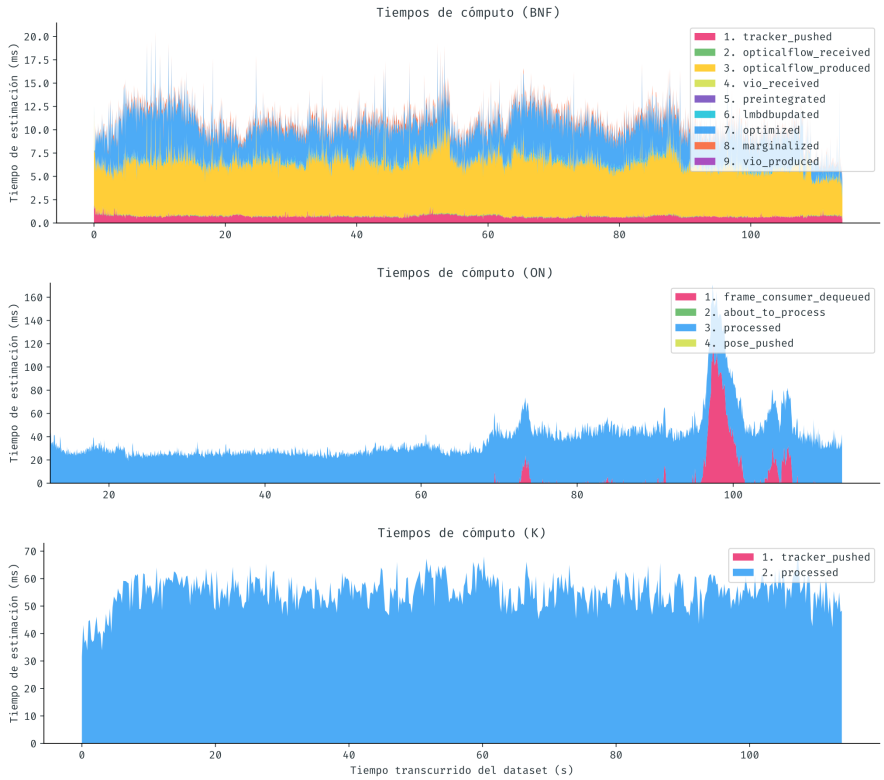


Figura 6.2: Tiempos de cómputo de BNF, ON y K sobre el dataset EV201. Notar que Basalt tiene más información sobre las distintas etapas de su pipeline interna simplemente porque fue el más estudiado. Notar que ORB-SLAM3 presenta problemas de congestión y que los tiempos de Kimera son bastante estables en comparación a Basalt a pesar de ser mayores. Por no usar OpenCV, Basalt requiere (por ahora) copiar los cuadros de entrada y ese tiempo puede visualizarse en rosa en el gráfico.

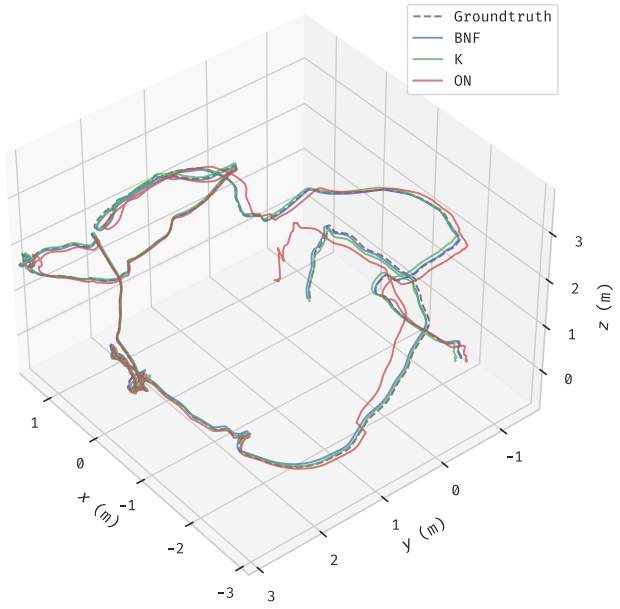


Figura 6.3: Trayectorias de los tres sitemas comparados a la groundtruth. Considerar que estas trayectorias están alineadas con Umeyama para minimizar sus diferencias y por esta razón no coinciden sus inicios ni finales.

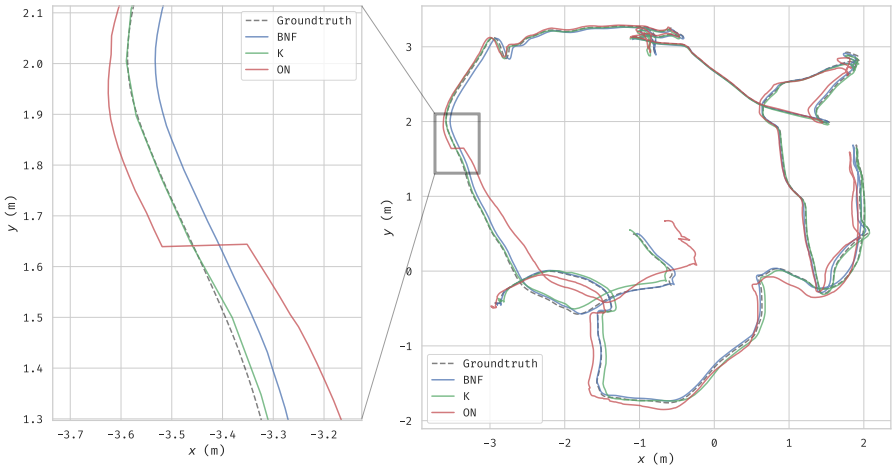
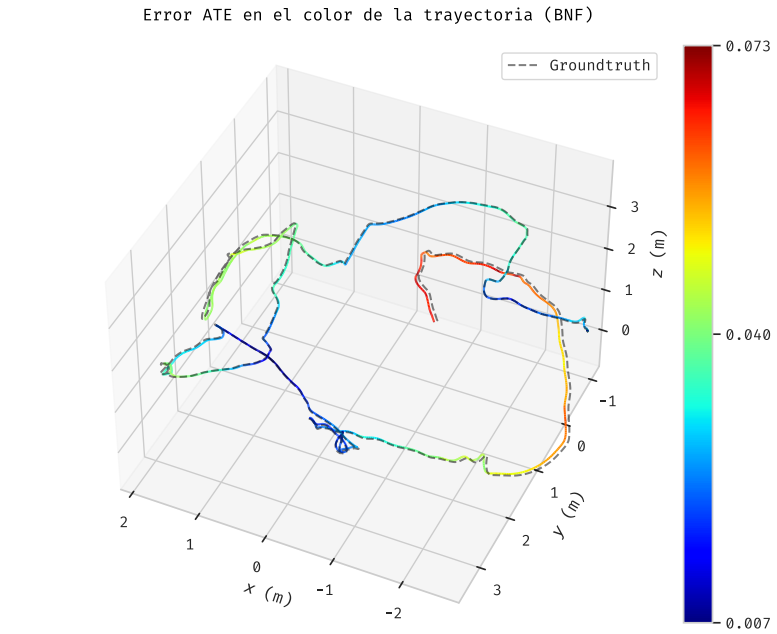


Figura 6.4: Vista del plano XY de la trayectoria a derecha. A izquierda se hace zoom a una porción de la trayectoria para mostrar cómo un momento de loop closure que en ORB-SLAM3 cuando el mapa fue actualizado durante la corrida genera una corrección brusca que es indeseable en XR.



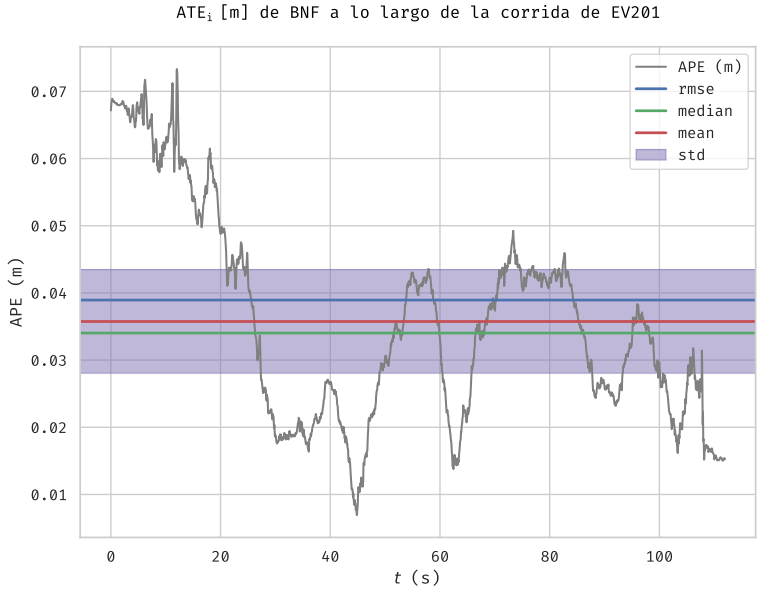


Figura 6.5: Abajo se muestra el error ATE_i para cada timestamp i de Basalt. Además este error se mapea en el color de la trayectoria en la figura de arriba.

7

CONCLUSIONES Y TRABAJO FUTURO

En este trabajo se estudiaron distintos sistemas de SLAM/VIO en el contexto de localización en tiempo real para XR. Vimos algunos de los conceptos fundamentales que estos utilizan como el algoritmo de Gauss Newton para resolver problemas de optimización no lineal, de los cuales el área de visión por computadora está plagado, y SLAM no es la excepción. También vimos las distintas formas de representar transformaciones y rotaciones en dos y tres dimensiones: ángulos euler, cuaterniones, ángulo axial, matrices de rotación y una mirada práctica sobre los grupos de Lie $SO(n)$ y $SE(n)$ junto a sus álgebras de Lie $\mathfrak{so}(n)$ y $\mathfrak{se}(n)$.

Posteriormente nos adentramos en la implementación de la capa de odometría visual-inercial de Basalt. Esto permitió ver de primera mano los distintos tipos de algoritmos que se reúnen en este tipo de sistemas. Se integró Kimera-VIO, ORB-SLAM3 y Basalt a Monado, el runtime OpenXR de código libre. Para esto hizo falta diseñar una interfaz eficiente que generaliza de forma razonable estos sistemas. Se analizaron los problemas de implementación particulares a considerar para XR como la predicción y filtrado de poses, o como lidiar con la imperfección de los sensores de cámara e IMU. Se contribuyeron a Monado todas estas mejoras, incluyendo la extensión de dos controladores de dispositivos que ahora son capaces de aprovechar este tipo de tracking. Uno de ellos es una plataforma VR de producción comercial que ahora puede ser utilizada por usuarios entusiastas que deseen utilizar este tipo de hardware en GNU/Linux con un stack de software completamente libre.

Este proyecto plantea las bases de infraestructura en Monado para este tipo de sistemas, pero aún hay mucho por hacer y por mejorar para lograr tracking con calidades similares a las que se encuentran en productos comerciales. Se plantea como trabajo futuro:

- Mejorar la experiencia de usuario para al utilizar el SLAM tracker en Monado. El trabajo realizado actualmente puede resultar un poco complejo de instalar y configurar para un usuario inexperto.
- Permitir el uso de múltiples implementaciones de SLAM/VIO de forma dinámica. Es decir, poder tener distintas implementaciones corriendo en simultáneo y localizando a distintos dispositivos.
- Hay espacio de mejora en el rendimiento de las implementaciones. En general, en este trabajo nos limitamos a hacer lo justo y

necesario para que el tracking funcione a tiempos razonables y no retrase el pipeline de Monado. Más aún, parece existir poca cantidad de trabajos que apliquen unidades de cómputo masivamente paralelas, como lo son las GPU, al problema de localización visual-inercial. Creemos que existen posibles ganancias de eficiencia en esta línea de trabajo.

- Sería bueno extender Basalt para soportar algún tipo de mapeo global en tiempo real que permita tener trayectorias consistentes que no tiendan a moverse lentamente con el tiempo. Más información al respecto en la Discusión 18.

- Sería bueno mejorar las formas de testeo y evaluación de sistemas SLAM en Monado, poder automatizarlas e integrarlas en los procesos de integración continua del proyecto. Esto permitiría el impacto que nuevos cambios traen al rendimiento y la precisión del sistema.

- Existen pocos conjuntos de datos para SLAM aptos para XR (p. ej. TUM-VI [31]), y ante la dificultad de producirlos, sería ideal aprovecharse de las herramientas fotorrealistas que son fácilmente accesibles en la actualidad para la generación de datos sintéticos.

- Existen métodos de predicción más eficientes que podrían adaptarse a Monado en lugar del método ad hoc desarrollado en este trabajo. En particular el mismo trabajo de preintegración de muestras de IMU utilizado en Basalt [32] puede ser un muy buen punto de partida para un método de predicción más preciso.

- Finalmente, muchos de los módulos que forman parte de estos sistemas son útiles de manera individual. La integración de estos en Monado podría beneficiar a distintos controladores que quieran hacer uso de algoritmia de visión por computadora específica en otros contextos.

Parte V

APÉNDICE

A

LISTADO DE CONTRIBUCIONES

Durante el desarrollo de este trabajo se produjeron distintas contribuciones, discusiones y repositorios complementarios. Por la metodología de trabajo usual del software libre, todo esto queda plasmado públicamente. En esta sección se compilan los enlaces más significativos de estos registros en la web y se los describe brevemente.

A.0.1 *Repositorios*

1. Monado, el proyecto en el que se realizaron la mayor parte de las contribuciones en forma de *merge requests*:
   `gitlab.freedesktop.org/monado/monado`
2. Adaptación de Kimera-VIO para usar en Monado:
   `gitlab.freedesktop.org/mateosss/Kimera-VIO`
3. Adaptación de ORB-SLAM3 para usar en Monado:
   `gitlab.freedesktop.org/mateosss/ORB_SLAM3`
4. Adaptación de Basalt para usar en Monado:
   `gitlab.freedesktop.org/mateosss/basalt`
5. Herramientas para análisis de métricas producidas por Monado:
   `gitlab.freedesktop.org/mateosss/xrtslam-metrics`

A.0.2 *Monado*

6. Controlador `qwerty` para emular dispositivos XR con teclado y ratón:
   `gitlab.freedesktop.org/monado/monado/-/merge_requests/714`
7. Controlador `euroc` para la reproducción de datasets visual-inerciales:
   `gitlab.freedesktop.org/monado/monado/-/merge_requests/880`
8. Integración inicial de sistemas externos de VIO/SLAM en Monado:
   `gitlab.freedesktop.org/monado/monado/-/merge_requests/889`
9. Extensión del controlador `realsense` para tracking por VIO/SLAM:
   `gitlab.freedesktop.org/monado/monado/-/merge_requests/907`
10. Implementación de Basalt como posible sistema externo de SLAM:
    `gitlab.freedesktop.org/monado/monado/-/merge_requests/941`
11. Actualización de `slam_tracker` con características dinámicas:
    `gitlab.freedesktop.org/monado/monado/-/merge_requests/1016`
12. Grabador de datasets EuRoC:
    `gitlab.freedesktop.org/monado/monado/-/merge_requests/1017`
13. Extensión del controlador `wmr` para tracking por SLAM/VIO:
    `gitlab.freedesktop.org/monado/monado/-/merge_requests/1035`

14. Predicción sencilla para el SLAM tracker:
    gitlab.freedesktop.org/monado/monado/-/merge_requests/1060
15. Implementación de filtrado y extensión del algoritmo de predicción:
    gitlab.freedesktop.org/monado/monado/-/merge_requests/1067
16. Generación de datos para evaluación de rendimiento y precisión:
    gitlab.freedesktop.org/monado/monado/-/merge_requests/1152
17. Herramienta para evaluar datasets en lote:
    gitlab.freedesktop.org/monado/monado/-/merge_requests/1172

A.0.3 *Basalt*

18. Discusión e ideas para generar un mapa consistente en tiempo real:
    gitlab.com/VladyslavUsenko/basalt/-/issues/69
19. Discusión sobre uso de cámaras similares a las de dispositivos WMR:
    gitlab.com/VladyslavUsenko/basalt/-/issues/62
20. Discusión sobre calibración de la IMU:
    gitlab.com/VladyslavUsenko/basalt-headers/-/issues/8
21. Modelo de cámara radial-tangencial de 8 parámetros de WMR:
    gitlab.com/VladyslavUsenko/basalt-headers/-/merge_requests/21

[1] Steven Michael LaValle. *Virtual Reality*. Cambridge ; New York: Cambridge University Press. URL: http://lavalle.pl/vr/.

[2] G. Welch y E. Foxlin. «Motion Tracking: No Silver Bullet, but a Respectable Arsenal». En: *IEEE Computer Graphics and Applications* 22.6 (nov. de 2002). DOI: 10.1109/MCG.2002.1046626.

[3] Vector Nav. *Inertial Navigation Primer*. URL: https://www.vectornav.com/resources/inertial-navigation-primer.

[4] Myriam Servières, Valérie Renaudin, Alexis Dupuis y Nicolas Antigny. «Visual and Visual-Inertial SLAM: State of the Art, Classification, and Experimental Benchmarking». En: *Journal of Sensors* 2021 (feb. de 2021). DOI: 10.1155/2021/2054828.

[5] Takafumi Taketomi, Hideaki Uchiyama y Sei Ikeda. «Visual SLAM Algorithms: A Survey from 2010 to 2016». En: *IPSJ Transactions on Computer Vision and Applications* 9.1 (jun. de 2017). DOI: 10.1186/s41074-017-0027-2.

[6] The Khronos Group Inc. *The OpenXR Specification*. URL: https://www.khronos.org/registry/OpenXR/specs/1.0-khr/pdf/xrspec.pdf.

[7] Antoni Rosinol, Marcus Abate, Yun Chang y Luca Carlone. «Kimera: An Open-Source Library for Real-Time Metric-Semantic Localization and Mapping». En: *arXiv:1910.02490 [cs]* (mar. de 2020). arXiv: 1910.02490 [cs].

[8] Carlos Campos, Richard Elvira, Juan J. Gómez Rodríguez, José M. M. Montiel y Juan D. Tardós. «ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM». En: *IEEE Transactions on Robotics* 37.6 (dic. de 2021). DOI: 10.1109/TRO.2021.3075644.

[9] Vladyslav Usenko, Nikolaus Demmel, David Schubert, Jörg Stückler y Daniel Cremers. «Visual-Inertial Mapping with Non-Linear Factor Recovery». En: *IEEE Robotics and Automation Letters* 5.2 (abr. de 2020). DOI: 10.1109/LRA.2019.2961227.

[10] Ken Shoemake. «Animating Rotation with Quaternion Curves». En: *ACM SIGGRAPH Computer Graphics* 19.3 (jul. de 1985). DOI: 10.1145/325165.325242.

[11] F. Sebastin Grassia. «Practical Parameterization of Rotations Using the Exponential Map». En: *Journal of Graphics Tools* 3.3 (mar. de 1998), págs. 29-48. ISSN: 1086-7651. DOI: 10.1080/10867651.1998.10487493.

[12] Ethan Eade. *Derivative of the Exponential Map*. Inf. téc. URL: https://ethaneade.com/exp_diff.pdf.

[13] Timothy D. Barfoot. *State Estimation for Robotics*. Cambridge: Cambridge University Press, 2017. DOI: 10.1017/9781316671528.

[14] Joan Solà, Jeremie Deray y Dinesh Atchuthan. «A Micro Lie Theory for State Estimation in Robotics». En: *arXiv:1812.01537 [cs]* (dic. de 2021). arXiv: 1812.01537 [cs].

[15] Thomas S. Shores. *Applied Linear Algebra and Matrix Analysis*. 2007th edition. New York: Springer, ago. de 2007. ISBN: 978-0-387-33195-9.

[16] Jorge Nocedal y Stephen J. Wright. *Numerical Optimization*. 2nd ed. Springer Series in Operations Research. New York: Springer, 2006. ISBN: 978-0-387-30303-1.

[17] Stephen Boyd. *Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares*. 1st edition. Cambridge, UK ; New York, NY: Cambridge University Press, ago. de 2018. ISBN: 978-1-316-51896-0.

[18] *GNU General Public License Version 3*. URL: https://opensource.org/licenses/gpl-3.0.html.

[19] *The 2-Clause BSD License*. URL: https://opensource.org/licenses/BSD-2-Clause.

[20] *The 3-Clause BSD License*. URL: https://opensource.org/licenses/BSD-3-Clause.

[21] Nikolaus Demmel, David Schubert, Christiane Sommer, Daniel Cremers y Vladyslav Usenko. «Square Root Marginalization for Sliding-Window Bundle Adjustment». En: *arXiv:2109.02182 [cs]* (sep. de 2021). arXiv: 2109.02182 [cs].

[22] Richard Hartley y Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Second. Cambridge: Cambridge University Press, 2004. DOI: 10.1017/CBO9780511811685.

[23] Raul Mur-Artal y Juan D. Tardos. «Visual-Inertial Monocular SLAM with Map Reuse». En: *IEEE Robotics and Automation Letters* 2.2 (abr. de 2017). DOI: 10.1109/LRA.2017.2653359.

[24] Frank Dellaert y Michael Kaess. «Factor Graphs for Robot Perception». En: *Foundations and Trends in Robotics* 6.1-2 (2017). DOI: 10.1561/2300000043.

[25] Lance Williams. «Pyramidal Parametrics». En: *SIGGRAPH* (1983). DOI: 10.1145/800059.801126.

[26] M. Burri, J. Nikolic, Pascal Gohl, T. Schneider, J. Rehder, Sammy Omari, Markus Achtelik y R. Siegwart. «The EuRoC Micro Aerial Vehicle Datasets». En: *Int. J. Robotics Res.* (2016). DOI: 10.1177/0278364915620033.

[27] Edward Rosten, Reid Porter y Tom Drummond. «Faster and Better: A Machine Learning Approach to Corner Detection». En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.1 (ene. de 2010). DOI: 10.1109/TPAMI.2008.275.

[28] *Apache License, Version 2.0*. URL: https://opensource.org/licenses/Apache-2.0.

[29] Bruce D. Lucas y Takeo Kanade. «An Iterative Image Registration Technique with an Application to Stereo Vision». En: *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI'81. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., ago. de 1981, págs. 674-679.

[30] Jean yves Bouguet. «Pyramidal implementation of the Lucas Kanade feature tracker». En: *Intel Corporation, Microprocessor Research Labs* (2000).

[31] David Schubert, Thore Goll, Nikolaus Demmel, Vladyslav Usenko, Jörg Stückler y Daniel Cremers. «The TUM VI Benchmark for Evaluating Visual-Inertial Odometry». En: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (oct. de 2018). DOI: 10.1109/IROS.2018.8593419.

[32] Christian Forster, Luca Carlone, Frank Dellaert y Davide Scaramuzza. «On-Manifold Preintegration for Real-Time Visual-Inertial Odometry». En: *IEEE Transactions on Robotics* 33.1 (feb. de 2017). DOI: 10.1109/TRO.2016.2597321.

[33] *Boost Software License 1.0 (BSL-1.0)*. URL: https://opensource.org/licenses/BSL-1.0.

[34] John Lakos. *Large-Scale C++ Software Design*. 1st edition. Reading, Mass: Addison-Wesley, jul. de 1996. ISBN: 978-0-201-63362-7.

[35] Muhammad Huzaifa y col. *Exploring Extended Reality with ILLIXR: A new Playground for Architecture Research*. 2021. arXiv: 2004.04643 [cs.DC].

[36] Zichao Zhang y Davide Scaramuzza. «A Tutorial on Quantitative Trajectory Evaluation for Visual(-Inertial) Odometry». En: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Madrid: IEEE, oct. de 2018. DOI: 10.1109/IROS.2018.8593941.

[37] Greg Welch y Gary Bishop. «SCAAT: Incremental Tracking with Incomplete Information». En: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '97*. Not Known: ACM Press, 1997. DOI: 10.1145/258734.258876.

[38] Géry Casiez, Nicolas Roussel y Daniel Vogel. «1 € Filter: A Simple Speed-Based Low-Pass Filter for Noisy Input in Interactive Systems». En: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Austin Texas USA: ACM, mayo de 2012. DOI: 10.1145/2207676.2208639.

[39] Claus Gramkow. «On Averaging Rotations». En: *Journal of Mathematical Imaging and Vision* 15 (jul. de 2001). DOI: 10.1023/A:1011217513455.

[40] Jan-Philipp Stauffert, Kristof Korwisi, Florian Niebling y Marc Erich Latoschik. «Ka-Boom!!! Visually Exploring Latency Measurements for XR». En: *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*. 20. New York, NY, USA: Association for Computing Machinery, mayo de 2021. DOI: 10.1145/3411763.3450379.

[41] Patrick Geneva, Kevin Eckenhoff, Woosik Lee, Yulin Yang y Guoquan Huang. «OpenVINS: A Research Platform for Visual-Inertial Estimation». En: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. Paris, France: IEEE, mayo de 2020. DOI: 10.1109/ICRA40945.2020.9196524.

[42] Zichao Zhang, Christian Forster y Davide Scaramuzza. «Active Exposure Control for Robust Visual Odometry in HDR Environments». En: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. Singapore, Singapore: IEEE, mayo de 2017. DOI: 10.1109/ICRA.2017.7989449.

[43] Dean Brown. «Decentering Distortion of Lenses». En: *Photogrammetric Engineering* 32 (1966).

[44] J. Kannala y S.S. Brandt. «A Generic Camera Model and Calibration Method for Conventional, Wide-Angle, and Fish-Eye Lenses». En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28.8 (ago. de 2006). DOI: 10.1109/TPAMI.2006.153.

[45] Rainer Kümmerle, Bastian Steder, Christian Dornhege, Michael Ruhnke, Giorgio Grisetti, Cyrill Stachniss y Alexander Kleiner. «On Measuring the Accuracy of SLAM Algorithms». En: *Autonomous Robots* 27.4 (nov. de 2009). DOI: 10.1007/s10514-009-9155-6.

[46] Luigi Nardi y col. «Introducing SLAMBench, a Performance and Accuracy Benchmarking Methodology for SLAM». En: *2015 IEEE International Conference on Robotics and Automation (ICRA)* (mayo de 2015). DOI: 10.1109/ICRA.2015.7140009.

[47] Jianzhu Huai, Yujia Zhang y Alper Yilmaz. «The Mobile AR Sensor Logger for Android and iOS Devices». En: *arXiv:2001.00470 [cs, eess]* (dic. de 2019). arXiv: 2001.00470 [cs, eess].

[48] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard y Daniel Cremers. «A Benchmark for the Evaluation of RGB-D SLAM Systems». En: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. de 2012. DOI: 10.1109/IROS.2012.6385773.

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

[50] S. Umeyama. «Least-Squares Estimation of Transformation Parameters between Two Point Patterns». En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13.4 (abr. de 1991). DOI: 10.1109/34.88573.