Junio 2013

Tesis de Licenciatura

Smartphones como Unidad de Sensado y Procesamiento para la Localización de Robots Móviles Utilizando Odometría Visual Monocular

González, Emiliano
426/06
edgonzalez@dc.uba.ar

Gonzalez, Sergio
481/06
segonzalez@dc.uba.ar

Directores

Dr. Pablo De Cristóforis
pdecris@dc.uba.ar

Lic. Matias Nitsche
mnitsche@dc.uba.ar

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

RESUMEN

Las aplicaciones relativas a robots autónomos móviles requieren la exploración del entorno, la construcción de mapas y la localización fiable del robot en el mismo. El problema de la localización del robot a partir de un mapa dado del ambiente, o a la inversa, el problema de la construcción de un mapa bajo el supuesto de que la posición y orientación del robot es conocida, han sido abordados y resueltos utilizando diferentes enfoques.

El problema resulta más atractivo cuando no son conocidos ni la trayectoria del robot ni el mapa del ambiente. En este caso, la localización y la construcción del mapa deben ser consideradas simultáneamente. Este problema se conoce como SLAM (Simultaneous Localization And Mapping). La utilización de un Filtro Extendido de Kalman es una de las soluciones más extendidas al problema de SLAM, lo que se conoce como EKF-SLAM.

En este trabajo se presenta un sistema de localización basado en visión (odometría visual) para robots móviles en ambientes desconocidos que parte del método EKF-SLAM y utiliza *smartphones* como unidades de sensado y procesamiento. Este sistema tiene como sensor a la cámara del dispositivo móvil. A partir de la detección de marcas visuales en el ambiente, se construye un mapa que permite estimar la ubicación de la cámara y, por ende, del robot.

El sistema desarrollado fue evaluado en distintos escenarios utilizando un smartphone de última generación montado en el robot móvil ExaBot. Los experimentos realizados fueron analizados teniendo en cuenta aspectos de precisión, rendimiento y robustez. Los resultados obtenidos en esta tesis muestran la validez del método propuesto, así como también de la implementación del sistema desarrollado.

Palabras claves: Simultaneous Localization And Mapping, odometría visual, smartphones, robótica móvil

ABSTRACT

Applications concerning autonomous mobile robots require the environment exploration, map construction and reliable robot localization in it. The robot's localization problem from a given environment map, or instead, the map construction problem under the assumption that the robot's position and orientation is already known, had been addressed and resolved using different approaches.

The problem is more attractive when both environment map and robot's trajectory are unknown. In this case, localization and map construction must be addressed simultaneously. This problem is known as SLAM (Simultaneous Localization And Mapping). Extended Kalman Filter is a widely used solution to this problem, which is known as EKF-SLAM.

This thesis introduces a vision based localization system (visual odometry) for mobile robots in unknown environments that implements EKF-SLAM and uses smartphones as a sensing and processing unit. This system uses the mobile device's camera as a sensor. It builds a natural landmark map from detected features that allows the camera's (and thus robot's) location estimation.

The developed system was evaluated with different scenarios using a last generation smartphone mounted on the mobile robot named ExaBot. The experiments were analyzed in terms of precision, performance and robustness. The obtained results in this thesis validate the proposed method, as well as the developed system implementation.

Keywords: Simultaneous Localization And Mapping, visual odometry, smartphones, mobile robotics

AGRADECIMIENTOS

Al labo de robótica, por brindarnos su espacio y dejar que nos sintiéramos como en nuestra casa.

A nuestros directores Pablo y Matías, por abrirnos las puertas del laboratorio y ayudarnos en todo este trabajo.

A Taihú, por aguantarnos en todas las conversaciones filosóficas dentro del laboratorio, por darnos muchos consejos en todo este tiempo y por su buena onda.

A todos los integrantes del labo, por compartir muchos momentos dentro y fuera del labo.

A todos nuestros compañeros, en especial a Fede, Metegol, Gonza pelo, Gonza rubio, Viki, Marta, Page por compartir muchos días de estudio juntos y llevar la carrera a la par.

A todos nuestros profesores y a la universidad pública, por la oportunidad de estudiar y aprender de los mejores.

Agradecimientos de Emiliano

Ante todo quiero agradecer a mis viejos Julio y Nilda por el constante e incondicional apoyo que me brindan, no sólo en lo que respecta a mis estudios sino también en toda la vida. Son mi admiración y orgullo. Son el ejemplo perfecto de trabajo y sacrificio constante para alcanzar la meta.

A mis hermanos Pablo, Fernando y Hernán, por la paciencia que me tienen y por estar conmigo no sólo en los mejores momentos sino también en los difíciles. Gracias simplemente por ser los excelentes hermanos que son.

A Ninfa que me hizo crecer muchísimo como persona. Mi vida no hubiese sido lo mismo sin ella.

A Ale Rocco por su gran amistad y sus excelentes consejos. Es como un hermano más para mí.

A Sergio (o Checho), mi compañero de tesis, por la paciencia que me tuvo en los momentos de mal humor y en las charlas durante este trabajo. Disfruté mucho de su compañía y me enseñó a ser más abierto a las opiniones de otras personas.

A mis amigos, por los buenos momentos que pasamos y pasaremos.

A todas las personas que de una u otra manera ayudan a mi formación.

Agradecimientos de Sergio

A mi viejo Antonio, por sus infinitos consejos que me ayudaron a mirar un poco mas allá y ver todas las opciones de las cosas que quería hacer, por su aliento en todos los emprendimientos que se me ocurrían y por su enseñanza de tantos años, lo que me ayudó a ser la persona que soy ahora.

A mi vieja Nilda, por ser un ejemplo de constante trabajo, solidaridad hacia los demás y paciencia, por enseñarme a que siempre se puede mejorar.

A mis hermanos y mi sobrinito, Andrea, Athina, Tatiana y Walter, por todos los momentos que pasamos y por estar siempre alegrando con su chispa cada instante.

A todos mis amigos, que a pesar de mis cuelgues o desapariciones (sobre todo en este último tiempo) siempre estuvieron.

A Emi (o Emilio), por haber compartido todo este tiempo con migo, por las innumerables charlas (que desde afuera parecían discusiones) sobre la vida o la misma tesis.

A todos los que de alguna u otra forma me permitieron y ayudaron a llegar hasta acá.

A nuestros padres.

Índice general

1. INTRODUCCIÓN

El objetivo de este capítulo es familiarizar al lector con los conceptos involucrados en este trabajo. Para lograrlo se explica en detalle el problema de la odometría visual en la sección 1.1, se presenta el estado del arte y los avances sobre el tema en las secciones 1.2.4 y 1.3.1, y un resumen sobre la solución propuesta en la sección 1.4.

1.1. Definición del problema

En el campo de la robótica móvil, uno de los problemas más relevantes es alcanzar la completa autonomía del robot, es decir, que el mismo interactúe con el ambiente sin la asistencia de un ser humano. Para abordar este problema es necesario conocer el tipo de robot a utilizar, que puede ser terrestre, volador o acuático. En el caso de esta tesis, el foco está puesto en los robots móviles terrestres. Para que un robot sea autónomo debe disponer de sensores que le permitan capturar información del entorno, unidades de procesamiento (para poder procesar esta información) y actuadores (para que el robot pueda interactuar con su medio). A su vez, las unidades de procesamiento necesitan ejecutar algoritmos, que dotan al robot de la "inteligencia" suficiente para ser considerado autónomo.

El problema de la navegación autónoma se puede dividir en tres preguntas: "¿Dónde estoy?", "¿Hacia dónde voy?", "¿Cómo hago para llegar?". Estas tres preguntas son resueltas mediante la localización, la construcción de mapas del entorno (mapeo) y la planificación del movimiento del robot, respectivamente. La localización consiste en determinar la posición y la orientación del robot dentro del ambiente [1]. El mapeo es la construcción de modelos o representaciones del ambiente donde opera el robot [2], a partir de la información adquirida por los sensores. Por último la planificación del movimiento consiste en generar las acciones (mediante comandos a los actuadores del robot) de forma de alcanzar el objetivo deseado [3]. Las respuestas a las primeras dos preguntas son frecuentemente abordadas de forma simultánea utilizando SLAM (*Simultaneous Localization And Mapping*) [4, 5]. A continuación, se explica en detalle los conceptos nombrados.

1.2. Navegación autónoma

1.2.1. Localización

La localización involucra la estimación de la ubicación del robot, tanto de su posición como de su orientación. Se puede calcular de modo absoluto o relativo. En la localización relativa se intenta determinar la ubicación actual del robot en referencia a otras posiciones y orientaciones estimadas anteriormente. En consecuencia, la precisión del método está ligada a la buena estimación de estas ubicaciones anteriores. En cambio, en la localización absoluta se calcula la ubicación del objeto utilizando un mapa conocido de antemano que representa un marco de referencia, y sensores exteroceptivos que permiten sensar la ubicación del robot en ese mapa, como por ejemplo sistemas de posicionamiento global (GPS), *landmarks* (marcas reconocibles del entorno), entre otros, lo que posibilita la estimación sin la necesidad de conocer las ubicaciones anteriores.

La localización relativa se suele utilizar con una alta frecuencia de muestreo para maximizar la información disponible y minimizar las imprecisiones en la estimación actual. Enfrenta un verdadero obstáculo en el caso extremo en el que el objeto es repentinamente transportado a una posición arbitraria, lo que se conoce como el problema del robot secuestrado o *"Kidnapped robot problem"*. La localización absoluta logra superar este inconveniente pues no requiere de la información de ubicaciones previas. Teniendo esto en cuenta, la frecuencia de muestreo de la misma suele ser baja con respecto a la localización relativa [6].

Localización relativa

También conocido como *Dead Reckoning*, término que originalmente era utilizado por los marineros para estimar la velocidad de movimiento en el mar. Entre los métodos de localización relativa más comunes se encuentra la odometría, que consiste en el uso de sensores propioceptivos para estimar el desplazamiento relativo del robot. Se pueden considerar distintos tipos de odometría, entre ellos la basada en *encoders* (sensores de rotación o encoders) y la odometría visual o *visual odometry* [7]. La odometría con encoders estima la posición del robot a partir de las señales obtenidas de sus actuadores [8]. Existen encoders de tipo absoluto e incremental. El primero devuelve, apenas se lo enciende, la posición actual del eje como un valor absoluto y la conserva mientras se encuentra apagado. En cambio, los encoders incrementales requieren volver a una ubicación de referencia para sensar cambios en su posición. El objetivo de utilizar encoders es convertir la rotación sensada a un valor de distancia recorrida.

Por otro lado, la odometría visual utiliza imágenes obtenidas del ambiente físico para calcular de forma precisa el movimiento del robot en cada instante de tiempo. Esto se logra analizando y comparando imágenes consecutivas del recorrido realizado. Mediante la extracción de marcas visuales del ambiente en las imágenes (*features*) seguido de un proceso para establecer correspondencias (*matching*) entre un par de imágenes consecutivas es posible estimar el movimiento de la cámara (y por ende del robot) relativo al entorno. Para realizar odometría visual se puede utilizar diversos tipos de cámaras, como cámaras monoculares [9], cámaras omnidireccionales [10] y cámaras estéreo [11] (ver Figura 1.1) e incluso es posible integrarla con otro tipo de odometrías para lograr una mejor estimación, por ejemplo con el sensado inercial [12].

(a) (b) (c)

Fig. 1.1: Distintos tipos de cámara que se pueden utilizar para odometría visual: 1.1(a) cámara monocular, 1.1(b) cámara estéreo y 1.1(c) cámara omnidireccional.

La odometría con encoders asume que el movimiento del motor se traduce linealmente a un cierto desplazamiento respecto del suelo. Sin embargo esto depende del rozamiento de las ruedas con el terreno y de las características físicas del robot, lo que frecuentemente acarrea pequeños deslizamientos que provocan la acumulación de error a medida que el mismo se desplaza. Si bien este problema en general es inevitable, existen métodos para reducir sus efectos [13]. En el caso de la odometría visual, este problema no ocurre debido a que en estos casos el desplazamiento percibido a través de las imágenes es el correcto.

Otros métodos de localización relativa se basan en el uso de sensores de movimiento inercial [14] y giroscopios [15]. Para este tipo de odometría se utiliza la gravedad de la tierra y la fuerza centrífuga junto con el efecto Coriolis. Se intenta medir la velocidad del robot de forma indirecta a partir de la integración de las aceleraciones, y por consiguiente la posición mediante el conjunto de velocidades, por lo tanto el error acumulado es crítico.

Localización absoluta

En este caso se asume que el robot cuenta con una representación a priori del entorno (un mapa) que, mediante el sensado del mismo, le permiten conocer su ubicación. Este tipo de localización elimina el error acumulado de la localización relativa, pero en su lugar requiere de información a priori del entorno, lo cual no siempre es factible.

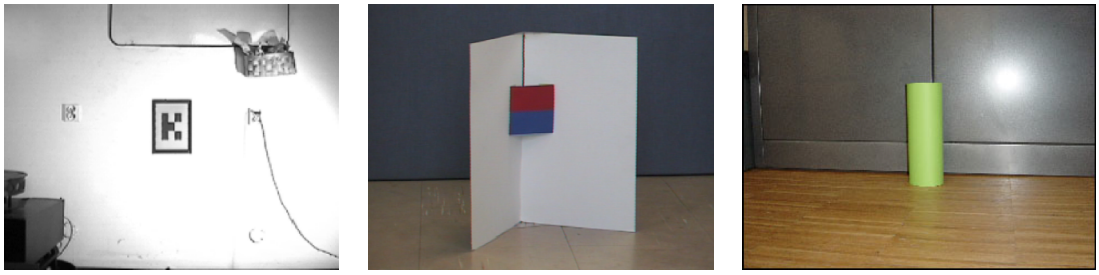
Algunos métodos de localización absoluta buscan la zona del mapa global más parecida a la región local en donde se encuentra el robot [16]. Por otro lado, dadas las ambigüedades de percepción que se pueden presentar, es posible utilizar métodos que generen múltiples hipótesis de ubicación del robot, que luego van siendo descartadas a medida que se adquiere más información del entorno. En general, estos métodos son una implementación particular de localización de Markov [17] y reciben el nombre de *Particle Filter* [44]. Esta información puede ser extraída de landmarks artificiales ubicados previamente en el espacio de navegación, cuya certera detección depende del grado de distinción que el mismo tiene con respecto al ambiente. En el caso de utilizar una cámara como sensor, depende de la iluminación en el ambiente, el color del landmark, su escala, el ruido en la imagen, entre otras cosas. La principal desventaja de los métodos de localización absoluta reside en la necesitad de contar con una representación o mapa del entorno a priori o algún tipo de infraestructura en el entorno, como landmarks artificiales en el caso de métodos basados en visión (ver Figura 1.2) o satélites en el caso de GPS.

1.2.2. Mapeo

En ciertas ocasiones no se conoce el mapa del ambiente por lo que se debe construir uno. Al contrario de la localización absoluta (en la que se conoce el mapa y se intenta calcular la ubicación del robot), en la versión más simple de mapeo se asume que la posición del robot es conocida, y a partir de la misma se desea generar información que describa la disposición del entorno.

Existen distintas formas de representar el entorno que dan lugar a distintos tipos de mapa, entre los que se encuentran los mapas métricos (grilla, geométrico, sensores y landmarks), topológicos e híbridos:

- Mapa de Grilla: el ambiente se divide en celdas de iguales dimensiones. Cada celda representa el área del ambiente que abarca, y tiene un estado asociado. En su forma más simple, el estado determina la probabilidad de que el área representada por dicha



(a) (b) (c)

Fig. 1.2: Distintos tipos de landmarks artificiales que pueden ser utilizados para la localización absoluta: 1.2(a) y 1.2(b) son landmarks basados en patrones [18, 19], 1.2(c) landmarks cilíndricos con distintos colores posibles [20].

celda sea navegable o, por el contrario, se encuentre ocupada por un objeto. Estos mapas reciben el nombre de Grillas de Ocupación (*Occupancy Grid*). Se asume que las variables aleatorias que representan el estado son independientes, y por lo tanto, es posible computar el impacto de las mediciones en la grilla de forma separada [21]. La desventaja de este tipo de mapas reside en el consumo de memoria, dado que el número de celdas resulta exponencial en las dimensiones del mapa.

- Mapa Geométrico: a diferencia de los mapas de grilla, no discretizan el ambiente y son más precisos. Intentan optimizar el consumo de la memoria y modelan solamente regiones importantes del entorno. Para esto utilizan primitivas geométricas, típicamente líneas [22] o polígonos [23] en casos de movimientos en terrenos de no más de 3 grados de libertad, y planos en caso de movimientos con 6 grados de libertad [24].

- Mapa de Sensores: se acumula la información de los sensores durante todo el recorrido sin ser procesada. La idea es analizar los datos en el futuro. Por ejemplo, el *Radish* (Robotics Data Set Repository) o Repositorio de Conjunto de Datos de Robótica, provee una colección de datos en su formato crudo [25]. RAWSEEDS [26] es otro ejemplo de lo mismo.

- Mapa de Landmarks: un landmark es una marca distintiva del ambiente que el robot puede reconocer con sus sensores. Existen landmarks naturales y artificiales. El primer tipo son aquellos objetos o características preexistentes en el ambiente. En cambio los del segundo tipo son objetos o marcas especialmente diseñadas y ubicadas para que el robot pueda reconocerlo fácilmente. Típicamente, los landmark naturales pueden ser detectados mediante el procesamiento de imágenes, aunque también se puede utilizar láser [27], sonares [28] o radares [29]. Es común ver esta técnica aplicada en conjunto con localización, lo que recibe el nombre de SLAM (Simultaneous Localization and Mapping, o Localización y Mapeo Simultáneos). En esta técnica, este tipo de mapas se utilizan para fines de localización, y dado que por lo general se necesita una gran cantidad de muestras, el costo de mantenerlos y procesarlos crece a medida que se incorporan nuevos landmarks al mapa.

- Mapa Topológico: se representa la información utilizando nodos y ejes. Los nodos se corresponden con zonas, y los ejes representan la posibilidad de viajar de un nodo a

otro. Adicionalmente, los vértices y aristas pueden almacenar datos útiles para asistir la navegación. A modo ilustrativo, se puede imaginar este tipo de mapas para el caso del tráfico público, donde cada nodo representa distintos puntos de interés en una ciudad y contienen diferentes características, por ejemplo en el caso de una estación de servicio podrían guardar su cantidad de surtidores, y sus aristas adyacentes, el congestionamiento de tráfico entre este punto de interés y sus vecinos. El problema de este tipo de mapas radica en la dificultad de generarlos a partir de solamente los sensores. Existen numerosas aplicaciones de este tipo de mapas en conjunto con mapas métricos, llamados mapas híbridos [30]. Para navegación en interiores, en el trabajo de Youngblood *et al.* [31] se propone representar habitaciones con grillas ocupacionales asociadas en un grafo global que indica la navegabilidad entre las zonas. Dado que pueden existir habitaciones de gran superficie que generen un alto costo en la planificación del recorrido local, en el trabajo de Nitsche *et al.* [32] se plantea fijar el tamaño de las grillas en un empeño por mejorar este problema. Las mismas ya no representan habitaciones sino áreas menores.

1.2.3. SLAM

El problema de la localización del robot a partir de un mapa dado del ambiente, o a la inversa, el problema de la construcción de un mapa bajo el supuesto de que la posición y orientación del robot es conocida, han sido abordados y resueltos utilizando diferentes enfoques. Sin embargo, el problema resulta más atractivo cuando no son conocidos ni la trayectoria del robot ni el mapa del ambiente. En este caso, la localización y la construcción del mapa deben ser consideradas simultáneamente. Este problema se conoce como SLAM (*Simultaneous Localization And Mapping*) [33, 34]. En SLAM el robot parte de una ubicación desconocida en un ambiente desconocido y a partir de la detección de marcas naturales en el ambiente (landmarks), estima simultáneamente su posición y la ubicación de las marcas. El robot entonces construye un mapa completo de marcas que son utilizadas para la localización del robot. Se han propuesto soluciones eficientes para el problema de SLAM utilizando diferentes enfoques y distintos tipos de sensores. En [35] Thrun *et al.* propone una solución para estimar la posición del robot que escala logarítmicamente en función de la cantidad de marcas en el ambiente, utilizando un láser como sensor. En [36] Wan Kyun Chung *et al.* y Neira *et al.* [37] se propone una forma de abordar el problema de SLAM utilizando un sonar como sensor. En los últimos años la gran mayoría de los trabajos realizados en SLAM tienen una clara tendencia a utilizar cámaras como sensores. Esto recibe el nombre de Visual SLAM. Se han utilizado diversos tipos de cámaras: estéreo [38], monoculares [39, 40, 41], u omni-direccionales [42, 43].

Para construir el mapa del ambiente navegado, los pasos básicos que implican las técnicas de SLAM se pueden resumir de la siguiente manera:

- Extracción de landmarks: el primer paso implica sensar el ambiente y procesar esta información. En caso de utilizar una cámara como sensor, el sensado corresponde a una imagen. Para extraer las marcas naturales del entorno (landmarks) se detectan determinadas características, denominadas *features*, en la imagen (ver Figura 1.3). Por último, para cada feature se construye un descriptor asociado que permite identificar de forma unívoca el landmark en pasados y futuros sensados.

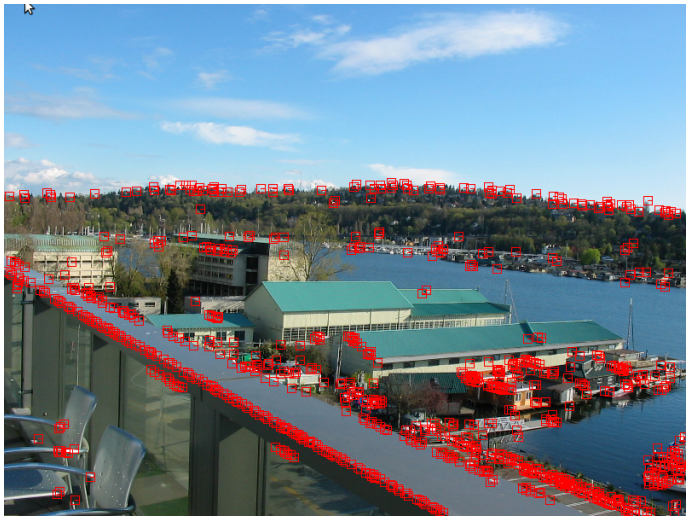


Fig. 1.3: Extracción de features de una imagen.

- Asociación de landmarks: se comparan los datos obtenidos del sensado con el mapa que se tiene hasta el momento para estimar la posición del robot. Existen distintas formas de establecer estas correspondencias, y dependerán del tipo de sensor utilizado, del mapa, y de la manera en que se describen los landmarks. Este proceso recibe el nombre de *matching* (correspondencias) de landmarks. En el caso de utilizar una cámara, las correspondencias se establecen entre los descriptores de los features (feature matching), como se puede ver en la Figura 1.4.

Fig. 1.4: Asociación de features entre imágenes.

- Localización: a partir de la asociación de landmarks se estima la ubicación del robot dentro del mapa. Para esto se utiliza un modelo de movimiento que realiza una predicción sobre la nueva ubicación.

- Actualización del mapa: por último se agregan, eliminan o modifican los landmarks en el mapa según un modelo de actualización que utiliza la información de matching y la estimación de la ubicación del robot.

Estos pasos se repiten constantemente, lo que hace de SLAM un método iterativo e incremental.

1.2.4. Estado del Arte

Entre los métodos más populares de SLAM y Visual SLAM se encuentran aquellos que utilizan marcos probabilísticos para la representación de la localización y del mapa [44]. En particular, el *Kalman Filter* o Filtro de Kalman (KF) [45] es probablemente el más estudiado. Fue propuesto en el año 1950 por Rudolph Emil Kalman como una técnica de filtrado y predicción de sistemas probabilísticos lineales. En el caso de SLAM se requiere justamente una forma de predecir la ubicación del robot y los landmarks después del movimiento, y de actualizar los mismos luego del sensado. Sin embargo el filtro de Kalman no es aplicable directamente ya que el problema de SLAM posee una naturaleza no lineal. Esto se debe a que los sensores del robot presentan limitaciones intrínsecas que agregan ruido a los datos obtenidos del ambiente; y a que los actuadores no se comportan de forma exacta ya que poseen márgenes de error que no se pueden determinar a priori. Por lo tanto, tratar con sensores, actuadores y un ambiente no controlado nos lleva a considerar el problema de SLAM como un problema no lineal. Para solucionar el problema anterior, se utiliza *Extended Kalman Filter* o Filtro de Kalman Extendido (EKF), el cual realiza una linealización previa del modelo. Estos métodos estiman el estado del mapa y localización utilizando variables Gaussianas unimodales, representadas por su media y covarianza. Mientras la media determina la ubicación del robot, la covarianza caracteriza la incertidumbre sobre el estimado. Una inmensa variedad de técnicas de SLAM basadas en EKF fueron investigadas e implementadas, algunas focalizando el rendimiento [46, 47], otros la precisión [48], u otros aspectos [49]. Por otro lado, las variables Gaussianas pueden ser multimodales, dando paso a otro algoritmo llamado *Particle Filter* o Filtro de Partículas [44], también conocido como método de Monte Carlo secuencial. A diferencia del EKF, este mantiene numerosas hipótesis sobre la localización del robot y, a medida que los sensores proveen mayor información del entorno, filtra las partículas espurias y a su vez pondera las mejores.

Dentro de EKF Visual SLAM, uno de los primeros algoritmos que lograron ejecución en tiempo real (30 Hz) con cámara monocular, se encuentra el trabajo de Davison *et al.* [39]. El objetivo del aporte de Davison es navegar en el ambiente y simultáneamente construir un mapa en 3D que sea esparzo (con unos pocos puntos en el espacio) utilizando únicamente landmarks naturales visuales. Uno de los aportes clave de su trabajo es un modelo dinámico general para movimientos continuos de una cámara con 6 grados de libertad de movimiento y velocidad constante (en un espacio 3D). Otro es un método novedoso para establecer correspondencias de manera activa, llamado *Active Matching* [50], que predice la nueva ubicación de cada landmark del mapa antes de obtenerlos de la imagen. Esto reduce el espacio de búsqueda de correspondencias de cada feature en la imagen a solamente una elipse de pocos pixeles. Esta correspondencia se calcula utilizando un área rectangular de pixeles que caracteriza al landmark buscado, llamada "parche".

Más recientemente ha surgido un nuevo método de Visual SLAM llamado *Parallel Tracking And Mapping* (PTAM) o Seguimiento y Mapeo Paralelos [51]. Este método pertenece a la familia de *KeyFrame-SLAM*, en el que la localización del robot se realiza en cada paso, pero solamente se actualiza el mapa utilizando unas pocas imágenes seleccionadas según un cierto criterio (*keyframe*). De esta forma, se logra separar el mapeo y la localización como dos procesos independientes, lo que permite explotar la potencia

tecnológica *multi-thread* de los procesadores actuales. Sin embargo este método se encuentra en desarrollo actualmente y promete ser una solución atractiva al problema de visual SLAM.

1.3. Smartphones como unidad de sensado y procesamiento

Alexander Graham Bell fue el inventor del teléfono. En 1878 realizó su primer comunicación telefónica. Los teléfonos desde entonces han evolucionado constantemente.

(a) (b) (c) (d)

Fig. 1.5: Teléfonos del siglo XIX y XX. 1.5(a) es el teléfono de Alexander Bell hacia 1878; 1.5(b) teléfono suizo hacia el año 1896; 1.5(c) año 1911; 1.5(d) año 1940.

En 1980 Motorola introdujo algunos de los primeros teléfonos celulares al mercado. Eran completamente distintos a los celulares de hoy en día, pues no eran compactos ni baratos y su funcionalidad era muy limitada.

Fig. 1.6: Motorola DynaTAC 8000x (1980).

En 1992 surgen oficialmente los dispositivos llamados PDA (*Personal Digital Assistant*) o computadoras de mano (*handheld*), que originalmente se diseñaron como una agenda personal electrónica, donde era posible configurar un calendario, poseer una lista de contactos, redactar textos, entre otras cosas. La característica principal de un dispositivo PDA es que los mismos cuentan con un sistema operativo, lo que permite la creación de nuevas aplicaciones y ampliar así las funcionalidades del dispositivo. Con el pasar del tiempo se fueron incorporando características, como la reproducción de archivos de audio

y video, contar con acceso a internet, tener la posibilidad de crear documentos, etc. Dichas características se asemejan a las de una computadora. En la Figura 1.7 se muestra un modelo de una PDA.

Fig. 1.7: Personal Digital Assistant marca Dell.

El primer *smartphone* fue desarrollado por IBM y BellSouth, disponible a la venta al público en 1993. Este dispositivo contaba con una pantalla táctil y era capaz de acceder al correo electrónico y enviar fax, entre otras cosas.

Fig. 1.8: Primer smartphone: "Simon"(1993).

Los smartphones son resultado de la evolución y la fusión de los teléfonos móviles y las computadoras de mano (PDAs). Estos celulares se encuentran en su auge tecnológico debido a la expansión del mercado de dispositivos móviles. En el año 2012, en algunos países desarrollados la cantidad de smartphones superó el 30 % de la población, como en España, Australia y Estados Unidos. En Singapur, la cifra llegó a más del 54 % de la población [52].

Fig. 1.9: Smartphones en la actualidad: a la izquierda iPhone 5, al centro Samsung Galaxy S4 y a la derecha Sony Xperia P.

Entre las características más importantes de los smartphones actuales se encuentran:

- Unidad central de procesamiento: los smartphones cuentan con potentes CPUs que les permite ejecutar una gran cantidad de aplicaciones que brindan un abanico de servicios al usuario. Hoy en día se pueden encontrar celulares con procesadores *multicore* (procesadores con varias CPUs) y con la capacidad de ejecutar múltiples procesos al mismo tiempo (*multiprocessing*). Por ejemplo, el Samsung Galaxy S4 posee dos CPUs multicore: uno quad-core de 1,6 GHz Cortex-A15 y otro quad-core de 1,2 GHz Cortex-A7. Además de los CPUs los smartphones actuales cuentan con GPUs (*Graphics Processing Unit*) dedicados al procesamiento de imágenes u otras operaciones multimedia, para aligerar la carga del procesador central en aplicaciones gráficas. Estos recursos de *hardware* son administrados por un sistema operativo.

- Sistema Operativo: los smartphones ejecutan un sistema operativo que brinda una interfaz entre el usuario y el hardware del dispositivo. El sistema operativo coordina la ejecución de las aplicaciones que se encuentran corriendo y administra el acceso a los recursos (memoria, sensores, comunicaciones, etc.). Entre los sistemas operativos más populares de la actualidad se encuentran Android, iOS, Windows 8 y Symbian.

- Aplicaciones: al contar con un sistema operativo, los smartphones tienen la capacidad de poder ejecutar un sinnúmero de aplicaciones (crear y editar documentos, descargar correo electrónico, navegar en internet, sacar y editar fotos, escuchar música, utilizar mapas con GPS, ejecutar video juegos, entre otras). Además, los sistemas operativos de los smartphones permiten programar e instalar nuevas aplicaciones, gracias a que estos dispositivos cuentan con una plataforma de desarrollo, que facilita y promueve la industrialización de *software* para móviles.

- Conectividad: la conexión a Internet se facilita cada vez más con el auge de los smartphones, dando lugar a redes de datos 3G y 4G de alta velocidad, así como también acceso Wi-Fi. En general esto provoca que los usuarios estén conectados las 24 horas del día, recibiendo mensajes y navegando desde prácticamente cualquier punto del globo. Para conectar el smartphone con otro dispositivo también se puede utilizar la tecnología *bluetooth*.

- Sensores: una característica novedosa que incorporan los smartphones son los sensores. En la actualidad cuentan con una amplia gama en los que se pueden mencionar cámaras, acelerómetro, compás, GPS, sensores de proximidad, giróscopo, entre otros. El uso de estos recursos da lugar a una nueva clase de aplicaciones basadas en el procesamiento de la información capturada por dichos sensores.

1.3.1. Estado del arte

Debido a las características antes mencionadas los smartphones resultan plataformas cada vez más atractivas para desarrollar sistemas y aplicaciones que requieran del sensado y el procesamiento integrado en un sólo dispositivo. Además, su portabilidad resulta de particular interés para este trabajo, pues el mismo dispositivo podría utilizarse como unidad de procesamiento y sensado para distintos robots. Existen antecedentes de la utilización de smartphones para ejecutar sistemas de navegación para robots móviles basados en el método de Particle Filter [20, 21]. En este trabajo el foco esta puesto en desarrollar un método de localización basado en los algoritmos *EKF Visual SLAM* [9].

1.4. Objetivos de esta tesis

El objetivo específico de este trabajo es desarrollar un método que posibilite el uso de smartphones como unidades de sensado y procesamiento en robots móviles para la localización y construcción de mapas del entorno. Para ello se propone utilizar la cámara digital del dispositivo como único sensor. Si bien podrían utilizarse los sensores integrados del smartphone el problema resulta inicialmente más atractivo cuando sólo se utiliza una cámara. Posteriormente se podrían incluir los mismos para mejorar las estimaciones.

En particular el foco de esta tesis está puesto en la localización a partir de la estimación de los movimientos de la cámara y, por ende, del robot. Este enfoque recibe el nombre de odometría visual. Se propone utilizar la técnica *Monocular Visual SLAM* basado en el Filtro de Kalman Extendido (EKF) que resulta apto para tareas de navegación que requieren alto rendimiento para lograr respuestas en tiempo real.

El objetivo más general de este trabajo es realizar un aporte en el área de la robótica móvil. Se espera que los resultados obtenidos contribuyan en la resolución del problema bien conocido SLAM (Simultaneos Localization and Mapping) para robots móviles con percepción basada en visión utilizando smartphones como unidades de sensado y procesamiento.

1.5. Núcleo de esta tesis

Considerando que el entorno de ejecución elegido son dispositivos móviles con limitado poder de cómputo, resulta necesario buscar algunas alternativas para alcanzar tiempo real en la ejecución de los algoritmos involucrados. Para el desarrollo de esta tesis se toma como base los trabajos de Davison *et al.* [39], Civera *et al.* [9] y Montiel *et al.* [48] que permiten encontrar un balance entre rendimiento y precisión.

De los trabajos previos se identifican tres elementos principales que en esta tesis serán analizados en detalle en los siguientes capítulos y que hacen al núcleo de la misma.

- Modelo dinámico de Davison: se trata de un modelo dinámico de velocidad constante (tanto angular como lineal) que permite la predicción del movimiento de la cámara

con 6 grados de libertad. La velocidad angular y lineal deben ser configuradas previamente al comienzo de la ejecución del algoritmo. Este modelo de movimiento es el utilizado dentro de un framework de EKF Visual SLAM.

- 1-Point RANSAC: una versión mejorada del clásico método de RANSAC para el filtrado de datos espurios (*outliers*) [53]. Se aprovecha la información provista por EKF sobre la estimación del mapa, el procesamiento de la imagen actual, y la posibilidad de realizar predicciones sobre el modelo de estimación, utilizando un conjunto de datos reducido que lo instancia con solamente una muestra. Se logra, por lo tanto, mejorar el rendimiento del original RANSAC en varios órdenes de magnitud, lo que lo hace ideal para lograr correspondencias óptimas a un bajo costo.

- Inverse Depth: es un método que permite enfrentar los problemas que surgen de observar landmarks con una sola cámara y de los cuales se desconoce inicialmente su profundidad. En estos casos y en los que los mismos se encuentren muy lejanos, este método plantea una representación alternativa de su distancia y posición.

En esta tesis se cambia el detector de features utilizado originalmente por Civera *et al.* en [9] y se reemplazan los parches del método Active Matching por los descriptores asociados a los features, que permiten alternar entre distintas implementaciones aprovechando sus diferentes virtudes [54, 55].

La implementación del sistema se realiza en C/C++ utilizando la librería OpenCV [56] para visión por computadora. Gracias a la portabilidad de ambos, el algoritmo puede ser utilizado en una gran variedad de arquitecturas, como por ejemplo x86, x86-64 y ARM.

1.6. Organización de esta tesis

Esta tesis se organiza de la siguiente manera:

- Capítulo 1: introducción a los conceptos básicos del problema de Localización y Odometría Visual, una breve reseña de su amplio estado del arte y el resumen de la solución propuesta en esta tésis.

- Capítulo 2: descripción de las propiedades teóricas de los métodos que resultan ser la base de la implementación de esta tesis, comenzando por las justificaciones probabilísticas de aquellos que utilicen las propiedades de Markov para realizar estimaciones, las bases del método Kalman Filter y de su consecuente, el Extended Kalman Filter.

- Capítulo 3: análisis en detalle de dos técnicas disjuntas que aportan al sistema resultante. La primera de estas en una representación diferente de los landmarks del mapa llamada Inverse Depth y la segunda refiere a una mejora de la técnica de eliminación de datos espurios RANSAC, denominada 1-Point RANSAC.

- Capítulo 4: definición del Estado y Matriz de Covarianza, Modelo Dinámico y Modelo de Medición necesarios para el algoritmo EKF. Explicación de una propiedad general del algoritmo que refiere a la adimensionalidad de la implementación del mismo en esta tesis. Presentación paso a paso y explicación exhaustiva del sistema desarrollado.

- Capítulo 5: explicación de la implementación del sistema y su arquitectura modular. Detalles de configuración y archivos de resultados disponibles para facilitar el uso por parte de terceros.
- Capítulo 6: exposición del modelo de robot, hardware de la computadora y especificaciones del dispositivo smartphone utilizados para realizar las experiencias. Presentación de experimentos reales y resultados obtenidos con el sistema desarrollado.
- Capítulo 7: conclusiones generales del trabajo realizado, trabajo pendiente y posibles mejoras a futuro.

2. EKF FRAMEWORK

Como se ha mencionado anteriormente, los robots cuentan con sensores que les permiten obtener información de su entorno y que luego procesan para realizar alguna acción. Este proceso describe una interacción entre estas partes, que se pueden separar en dos fases: la fase de sensado y la fase de control. La primera involucra todas las acciones de percepción que permiten obtener información del entorno y del estado del robot, y que sirven para incrementar su conocimiento. La segunda, en cambio, se encuentra asociada a las acciones que modifican el entorno, por ejemplo desplazarse o mover un objeto, aún cuando el robot no realice ninguna acción el entorno podría cambiar, luego por consistencia se asume que el robot siempre ejecuta una acción, incluso cuando decida no mover ninguno de sus actuadores. La ejecución de las acciones implican una pérdida de información dada las imprecisiones intrínsecas en los actuadores del robot y las condiciones cambiantes dentro del ambiente [21]. Las fases de sensado y control se realizan constantemente y aunque pueden ser concurrentes en el tiempo, éstas describen un ciclo (ver Figura 2.1).

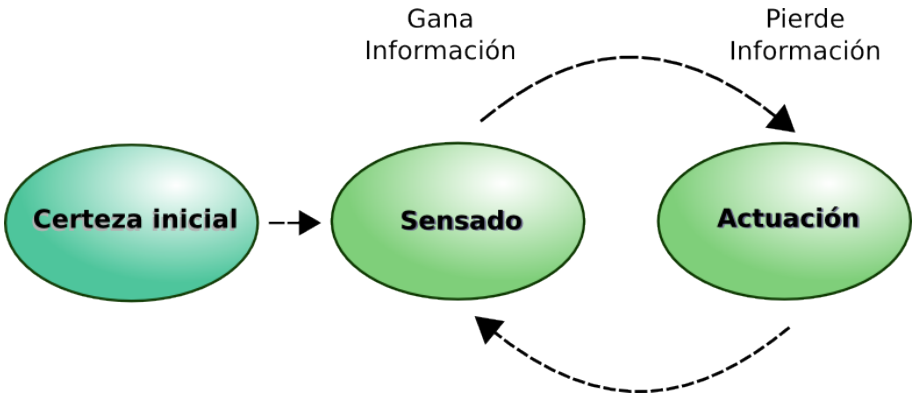


Fig. 2.1: Ciclo de sensado y control.

2.1. El Estado

El ciclo de sensado y control alimenta el estado del robot, que incluye tanto las variables relacionadas a la configuración actual del robot (como su ubicación, velocidad, estado de sus sensores, configuración de sus actuadores, etc.), como también las variables que describen el entorno (como ser la ubicación de objetos de referencia, marcas características, entre otros). Estas variables que conforman el estado del robot pueden cambiar a lo largo del tiempo, ya sea por las acciones que realiza el mismo o por los cambios dentro del entorno.

Un estado x_t (que se interpreta como el estado del robot en el instante t) se define como completo cuando es el que mejor permite predecir el estado futuro. Para esto se deben incluir todos los aspectos del entorno que podrían afectar el futuro y además las variables internas del robot, así como también todos los estados anteriores [21]. Es importante tener en cuenta que esta definición de completitud no requiere que el estado futuro sea una función determinística del estado presente. El futuro puede ser estocástico, pero no puede

depender de otras variables distintas de las que se incluyen dentro de x_t (dado que es completo) que influencien la evolución hacia el estado x_{t+1}. Los procesos que satisfacen esta condición son conocidos como *proceso de decisión de Markov*. En la práctica, resulta imposible definir un estado completo para un problema de la vida real. Por esta razón se intenta buscar un subconjunto pequeño de variables que permitan modelar el entorno de la mejor manera posible. A estos estados se los define como incompletos.

2.2. El estado y su relación probabilística

Las características no deterministas de la función de transición entre estados hacen que el proceso de sensado y control sea estocástico, propiedad que se ve reflejada dentro del estado del robot. De esta forma es posible definir una función de probabilidad entre un estado y el siguiente, donde cada una de las variables que componen el estado tienen a su vez su propia función probabilística de transición. Luego la evolución del estado puede definirse como la siguiente función probabilística [21]:

$$p(x_t|x_{0:t-1}, z_{1:t-1}, u_{1:t})$$

donde las variables $z_{1:t-1}$ definen las mediciones de sensado, $u_{1:t}$ los controles y $x_{0:t-1}$ los estados anteriores. Si se asume que x es completo, entonces x_{t-1} reúne la información de todos los sucesos previos y contiene además, suficiente información estadística sobre todos los controles y mediciones hasta el paso anterior $(t-1)$. Esto permite utilizar la propiedad de probabilidad condicional y de esta forma representar la transición del estado como sigue [21]:

$$p(x_t|x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(x_t|x_{t-1}, u_t) \tag{2.1}$$

Notar que para esta nueva definición, es necesario conocer el control actual (u_t) para poder determinar x_t. De esta forma se indica cómo el estado evoluciona en función de los controles u_t del robot.

Sucede lo mismo para las mediciones, definiendo la probabilidad de transición en el sensado y utilizando la propiedad de probabilidad condicional, asumiendo un estado completo, se puede llegar a la siguiente igualdad [21]:

$$p(z_t|x_{0:t}, z_{1:t-1}, u_{1:t}) = p(z_t, x_t) \tag{2.2}$$

Las ecuaciones (2.1) y (2.2) definen, junto con la probabilidad del estado inicial x_0, un sistema dinámico de tipo estocástico, donde la función de transición de estado indica cómo el mismo cambia a lo largo del tiempo y la función de transición del sensado denota cómo las mediciones se encuentran influenciadas por los estados. Así, por un lado, el estado actual depende estocásticamente sólo del estado anterior y los controles actuales (propiedad de Markov), mientras que el sensado actual depende estocásticamente del estado actual, dado que el sensado resulta de las mediciones con ruido del estado actual (ver Figura 2.2).

En general el estado del robot no es observable de forma directa, sino que sólo lo son algunas de las variables que lo conforman, es decir que el estado es parcialmente observable. Este modelo recibe el nombre de *proceso de decisión de Markov parcialmente observable*. Para estimar el estado en este caso se define la creencia o certeza (*belief*) para dicho estado, utilizando filtros Bayesianos [21].

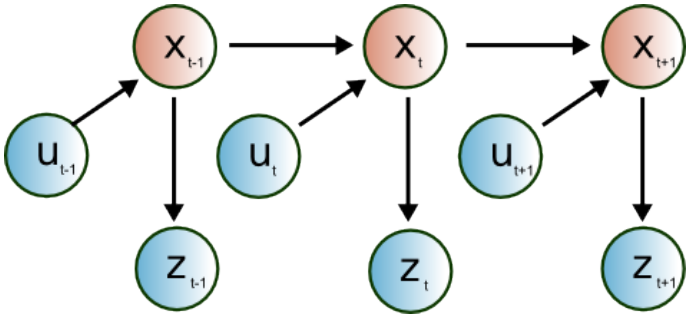


Fig. 2.2: Evolución del estado en función de las mediciones, los controles y el estado anterior.

2.3. Filtros Bayesianos

Los filtros Bayesianos son algoritmos que explotan las características y modelos probabilísticos mencionadas anteriormente, agregando además el concepto de certeza [21]. La certeza refleja el grado de conocimiento del robot acerca del estado. Un ejemplo de esto puede ser determinar la posición del robot dentro de un sistema de coordenadas global. Este valor no se puede obtener directamente, sino que debe ser inferido a partir de los sensados y las acciones que fue ejecutando el robot, pero este trae consigo un factor de incertidumbre.

La forma de representar la incertidumbre del robot es mediante el uso de funciones de probabilidad condicional, donde las mismas asignan una probabilidad a cada una de las variables dentro del estado, denominadas *belief distributions* [21], denotado como $bel(x_t)$. Estas probabilidades son a posteriori, es decir utilizando todos los datos disponibles hasta el momento. Por esta razón surgen dos posibles beliefs sobre el estado x_t. La primera tiene en cuenta tanto las mediciones como los controles hasta el tiempo t, como se puede ver en la ecuación (2.3). Puede haber una variante a la ecuación anterior, donde surge el segundo tipo de belief que tiene en cuenta todos los controles (acciones) realizados hasta el momento t, pero sólo las mediciones hasta el momento $t-1$. Este grado de certeza que se obtiene sobre el estado se denomina predicción, dado que define la probabilidad del estado basada en el estado anterior, al que aún no se le incorporan las mediciones en el tiempo actual, como se puede ver en la ecuación (2.4).

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t}) \tag{2.3}$$

$$\overline{bel(x_t)} = p(x_t|z_{1:t-1}, u_{1:t}) \tag{2.4}$$

A menudo se puede realizar el cálculo de la ecuación (2.3) a partir de la ecuación (2.4). Este método se denomina corrección (*correction*) o actualización a partir de las mediciones (*measurement update*) [21].

El algoritmo de Bayes calcula la incertidumbre (o certeza del estado) a partir de las mediciones y los datos de control (acciones). Este algoritmo es recursivo, dado que se calcula la certeza de un estado ($bel(x_t)$) a partir de la certeza del estado anterior ($bel(x_{t-1})$), junto con el control y las mediciones. En el algoritmo 1 se muestra el pseudocódigo del algoritmo correspondiente a la regla de actualización o *update rule* [21], que tiene como

entrada el cálculo del paso anterior ($bel(x_{t-1})$), los controles (u_t) y las mediciones actuales ($z_[t]$).

Algoritmo 1 Bayes(bel(x_{t-1}), u_t, z_t) : $bel(x_t)$

1: $\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1})\ bel(x_{t-1})dx$
2: $bel(x_t) = \eta\ p(z_t|x_t)\ \overline{bel}(x_t)$

En la línea 1 se calcula la predicción a partir de integrar (o sumar, en el caso discreto) las certezas de los posibles estados anteriores por la probabilidad de llegar al estado actual desde cada uno de los posibles estados anteriores. En la línea 2 se aplica la regla de actualización (update rule) que incorpora las mediciones del instante actual [21]. Se multiplica la predicción calculada en el paso anterior por la probabilidad de que la medición z_t sea observada. El producto resultante generalmente no es una probabilidad, es decir, podría no integrar a 1. Por lo tanto, se normaliza el resultado utilizando la constante η. Para utilizar este algoritmo en casos reales de estimación es necesario limitar el problema a un conjunto finito de estados, para transformar la integral infinita de la línea 1 en una sumatoria.

2.4. Kalman Filter

Dado que el algoritmo de Bayes original puede ser implementado sólo para problemas sencillos, existen algoritmos probabilísticos que definen aproximaciones al mismo para resolver problemas más complejos, focalizando diferentes aspectos o criterios como ser la eficiencia computacional, la facilidad de implementación o comprensión y la precisión entre otros. Dichas aproximaciones definen diferentes restricciones para garantizar cada uno de sus objetivos:

- Eficiencia computacional: a menudo utilizan sistemas Gaussianos de tipo lineal que permiten calcular las incertidumbres en tiempo polinomial en función de las dimensiones del entorno, por ejemplo distribuciones de probabilidad unimodales [21].
- Precisión: algunos métodos que buscan tener mejores resultados en este aspecto intentan estimar una amplia variedad de distribuciones multimodales. Caso contrario son las aproximaciones de tipo Gaussiana dado que se encuentran acotadas a distribuciones unimodales, característica que las hace menos exactas en la mayoría de los problemas [21].
- Facilidad de implementación: estos tipos de enfoque buscan definir aproximaciones reduciendo lo más posible la cantidad de factores de las distribuciones de transición de estado y de medición [21].

El algoritmo de Kalman Filter se encuentra dentro de los métodos que buscan tener un buen desempeño computacional, incluído en los sistemas Gaussianos o filtros Gaussianos, dado que representa la incertidumbre mediante distribuciones normales multivariadas:

$$p(x) = det(2\pi\Sigma)^{-1/2} e[-1/2(x-\mu)^T \Sigma^{-1}(x-\mu)] \tag{2.5}$$

Lo importante a destacar en la ecuación anterior es cómo se encuentra caracterizada la misma, ya que se define por dos conjuntos de parámetros: la media μ y la covarianza Σ.

La media se representa generalmente como un vector que tiene la misma dimensión que el estado definido. La covarianza resulta ser una matriz cuadrada, semi-definida positiva y simétrica, con dimensión igual al cuadrado del estado, lo implica que Σ depende cuadráticamente de la cantidad de variables que estén definidas en x. Estos enfoques son llamados estimadores de momentos o *moments representations* dado que μ y Σ resultan ser el primer y segundo momento de la distribución de probabilidad (2.5) respectivamente [21].

Kalman Filter representa las incertidumbres utilizando los conceptos de momentos, determinando la incertidumbre a través de la media μ y la covarianza Σ para un instante de tiempo t y asegurando que las distribuciones a posteriori sean Gaussianas con ciertas propiedades.

2.4.1. Propiedades

Las siguientes propiedades garantizan que los modelos probabilísticos a posteriori sean Gaussianos (tanto para la ecuación (2.1) como para (2.2)) lo que permite calcularlas en tiempo polinomial, haciendo foco en la eficiencia computacional como se verá más adelante. Las propiedades son las siguientes:

1. La asunción de Markov explicada en la sección 2.3.

2. La distribución de transición de estado (ecuación (2.1)) debe ser lineal en función de sus argumentos, sumándole ruido Gaussiano.

$$\mathbf{x}_t = A_t \mathbf{x}_{t-1} + B_t \mathbf{u}_t + \varepsilon_t \tag{2.6}$$

donde $\mathbf{x}_t$ y $\mathbf{x}_{t-1}$ continúan representando los estados en el instante t y $t-1$ respectivamente y son vectores que tienen la siguiente forma:

$$\mathbf{x}_t = (x_{1,t}, x_{2,t}, ..., x_{n,t})^\top \tag{2.7}$$

y

$$\mathbf{u}_t = (u_{1,t}, u_{2,t}, ..., u_{m,t})^\top \tag{2.8}$$

A_t y B_t son matrices que brindan las características de linealidad para la función de transición de estados. Esto restringe entonces al Kalman Filter a aproximar solamente sistemas dinámicos lineales. A_t es de dimensión $n \times n$ y B_t de $n \times m$, donde n es igual a la dimensión del estado $\mathbf{x}_{t-1}$ y m la dimensión del vector de control $\mathbf{u}_t$. El término ε representa un vector de variables aleatorias de distribución Gaussiana, que modelan la aleatoriedad de la función de transición de estados, con una media centrada en *cero* y con una covarianza R_t [21].

Incorporando toda esta información dentro de la ecuación (2.5) se consigue lo siguiente:

$$p(\mathbf{x}_t|\mathbf{u}_t, \mathbf{x}_{t-1}) = det(2\pi R_t)^{-1/2} e[-1/2(\mathbf{x}_t - A_t\mathbf{x}_{t-1} - B_t\mathbf{u}_t)^T R_t^{-1}(\mathbf{x}_t - A_t\mathbf{x}_{t-1} - B_t\mathbf{u}_t)] \tag{2.9}$$

3. La distribución de las mediciones (ecuación (2.2)) también debe ser lineal en sus argumentos. Por esta razón se define a C_t, una matriz de dimensiones $k \times n$ (donde k es la dimensión de $\mathbf{z}_t$) y a δ_t que es una distribución multivariada con una media en *cero* y varianza Q_t y representa el ruido de las mediciones. De esta manera la linealización queda como sigue:

$$\mathbf{z}_t = C_t \mathbf{x}_t + \delta_t \tag{2.10}$$

Utilizando lo mencionado anteriormente dentro de la ecuación (2.5) se obtiene:

$$p(\mathbf{z}_t|\mathbf{x}_t) = det(2\pi Q_t)^{-1/2} e[-1/2(\mathbf{x}_t - C_t\mathbf{x}_t)^T Q_t^{-1}(\mathbf{x}_t - C_t\mathbf{x}_t)] \tag{2.11}$$

4. La incertidumbre inicial de $\mathbf{x}_0$ también debe ser Gaussiana. Se deben definir entonces las medias y covarianzas iniciales μ_0 y Σ_0. Si se incorporan estas variables en la ecuación (2.5) quedaría lo siguiente:

$$p(\mathbf{x}_0) = det(2\pi\Sigma_0)^{-1/2} e[-1/2(\mathbf{x} - \mu_0)^T \Sigma_0^{-1}(\mathbf{x} - \mu_0)] \tag{2.12}$$

Estas propiedades resultan suficientes para garantizar que las incertidumbres siempre sean Gaussianas para cualquier instante t. Dicho esto, se pasa a definir el algoritmo genérico para Kalman Filter, algoritmo base utilizado en esta tesis.

2.4.2. Algoritmo

El algoritmo de Kalman Filter se nutre de la media μ_{t-1}, la covarianza Σ_{t-1} del instante anterior para calcular junto con los controles $\mathbf{u}_t$ y las mediciones $\mathbf{z}_t$, la incertidumbre del estado para el instante t, como se puede observar:

Algoritmo 2 Kalman_Filter(μ_{t-1}, Σ_{t-1}, $\mathbf{u}_t$, $\mathbf{z}_t$) : μ_t, Σ_t

1: $\overline{\mu}_t = A_t\ \mu_{t-1} + B_t\ \mathbf{u}_t$
2: $\overline{\Sigma}_t = A_t\ \Sigma_{t-1}\ A_t^\top + R_t$
3: $K_t = \overline{\Sigma}_t\ C_t^\top (C_t\ \overline{\Sigma}_t\ C_t^\top + Q_t)^{-1}$
4: $\mu_t = \overline{\mu}_t + K_t(\mathbf{z}_t - C_t\ \overline{\mu}_t)$
5: $\Sigma_t = (I - K_t\ C_t)\ \overline{\Sigma}_t$

Cada una de las partes del algoritmo se corresponden con algunas de las características mencionadas en las secciones anteriores, como se explica a continuación:

- Las líneas 1 y 2 realizan las predicciones tanto para la media como para la covarianza. Este resultado se corresponde con la ecuación (2.4). Es necesario el uso de la ecuación determinística (2.6).
- En las líneas 3 a 5 se agrega la medición en el instante t ($\mathbf{z}_t$) y se realiza el cálculo de la matriz K, llamada matriz de ganancia de Kalman o *Kalman Gain*, que indica en qué medida las mediciones tienen que afectar a la estimación del estado. Con la matriz Kalman Gain se procede a estimar la media μ_t junto con la desviación de las mediciones actuales, y Σ_t agregando proporcionalmente la información de las mediciones.

- Si pensamos el Kalman Gain como un valor flotante entre 0 y 1 (y la matriz identidad como un 1), se puede notar que las línea 4 y 5 intentan ponderar las mediciones dependiendo del Kalman Gain K_t. Si K_t es 0, entonces solamente se tendrá en cuenta la predicción para la media y la covarianza, y si es 1, las mediciones actuales afectarán el resultado junto con la predicción.

Al comienzo de la sección se destacó la eficiencia computacional de los algoritmos Gaussianos, y Kalman Filter no es una excepción. En el pseudocódigo anterior se pueden identificar fácilmente las operaciones más costosas. Las mismas son la inversión de matrices en la línea 3 y las multiplicaciones entre distintas matrices de las demás líneas. Actualmente la complejidad de uno de los mejores métodos para realizar la inversión de matrices se encuentra en el orden de $O(d^{2,37})$ [57], siendo d el tamaño de una matriz de $d \times d$. Teniendo en cuenta la línea 3, se puede definir un límite inferior (aproximado) a la complejidad de la inversión que se realiza en dicho lugar de $O(k^{2,37})$, siendo k la dimensión del vector de mediciones $\mathbf{z}_t$, dado que la matriz resultante antes de ser invertida tiene dimensión $k \times k$. La multiplicación de matrices se encuentra en el orden de complejidad $O(l^2)$ para una matriz de $l \times l$. En la línea 5 también se puede definir un límite inferior de $O(n^2)$, siendo n la dimensión del vector de estado, dado que las dimensiones de $(I - K_t C_t)$ y $\overline{\Sigma}_t$ es n [21]. En general el tamaño de las mediciones resulta ser mucho más pequeño que el tamaño del estado por lo que los factores de este último son los que controlan la complejidad final, que resulta ser de $O(n^2)$.

2.5. Extended Kalman Filter

Hasta el momento, los problemas a resolver mediante el uso de Kalman Filter tenían una restricción importante: deben poder ser modelados con sistemas lineales. Dicha restricción resulta válida, salvo que la mayoría de los problemas de la vida real no tienen esa característica, por lo que resulta difícil determinar una buena aproximación, además de no cumplir con las propiedades anteriormente mencionadas en la sección 2.4.1. Para este tipo de dificultades existe una variante del algoritmo Kalman Filter que puede lidiar con problemas no lineales.

Algoritmo 3 Extended_Kalman_Filter(μ_{t-1}, Σ_{t-1}, $\mathbf{u}_t$, $\mathbf{z}_t$) : μ_t, Σ_t

1: $\overline{\mu}_t = g(\mathbf{u}_t, \mu_{t-1})$
2: $\overline{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^\top + R_t$
3: $K_t = \overline{\Sigma}_t \, H_t^\top (H_t \, \overline{\Sigma}_t \, H_t^\top + Q_t)^{-1}$
4: $\mu_t = \overline{\mu}_t + K_t(\mathbf{z}_t - h(\overline{\mu}_t))$
5: $\Sigma_t = (I - K_t \, H_t) \, \overline{\Sigma}_t$

Esta variante reemplaza las ecuaciones de predicción por nuevas que contienen funciones g y h que linealizan los modelos. Las matrices para los sistemas lineales utilizadas por Kalman Filter son reemplazadas por jacobianos, donde el jacobiano G_t reemplaza a las matrices A_t, B_t y el jacobiano H_t reemplaza a C_t [21]. La complejidad de este nuevo método se mantiene en $O(n^2)$.

Dos de las características más importantes a la hora de buscar un buen desempeño son que las funciones que aproximan el problema sean lo más lineales posibles y que el tamaño de la incertidumbre sea lo más bajo posible. En general si las funciones son aproximadamente lineales, el método funcionará bien. En algunos casos donde los problemas

son no lineales o requieren tener más de una hipótesis (problemas que tengan propiedades multimodales) y dichas funciones se encuentren mal definidas, provocarán que el algoritmo diverja rápidamente. La incertidumbre, por otro lado, puede ser grande o pequeña. Si se tiene mucha seguridad sobre la ubicación robot, entonces el tamaño de la incertidumbre será chica. En cambio, no tener mucha seguridad de la localización genera que la incertidumbre crezca, y esto puede ser un signo de que las funciones de medición y predicción no estén funcionando correctamente.

3. LANDMARKS: MEJORAS EN SU REPRESENTACIÓN Y FILTRADO DE CORRESPONDENCIAS OUTLIERS

En general, en los sistemas de localización visual que utilizan features de las imágenes, se representan los puntos en el mundo (o landmarks) correspondientes a estos features mediante el sistema euclidiano de tres coordenadas en el espacio. Dicha representación tiene algunas desventajas cuando los landmarks son detectados por primera vez y se quieren utilizar en las iteraciones siguientes del algoritmo. Podría ocurrir que su posición en el espacio no se encuentre bien definida para las primeras iteraciones. Inverse Depth [58] surge como una parametrización alternativa a la representación euclidiana de los puntos en el espacio, que permite incorporar inmediatamente los landmarks al mapa contribuyendo de forma positiva a su estimación.

Los sistemas basados en visión que estiman la localización calculan las correspondencias entre features de las diferentes imágenes que se van capturando. Si estas correspondencias son espurias (mal establecidas), resultan nocivas para la estimación. Por esta razón es necesario filtrarlas. La técnica de RANSAC (*Random Sample Consensus*) [59] es un método iterativo que estima los parámetros de un modelo (en el caso de la odometría visual, el modelo de movimiento de la cámara entre dos imágenes) discriminando los datos que podrían ser nocivos para la estimación. A su vez, existe una optimización de esta técnica denominada 1-Point RANSAC [9] que a partir de cierta información previa requiere menos datos para la estimación del modelo.

3.1. Parametrización Inverse Depth

Para estimar la posición en el espacio de cada punto en el mundo con una cámara monocular es necesario observarlo repetidas veces desde distintos puntos de vista (*viewpoints*). Cada vez que se lo visualiza, se puede obtener un rayo que parte del centro óptico de la cámara en dirección hacia el punto observado en el mundo. Este rayo corta el plano de la imagen en el feature correspondiente al punto observado en el mundo, lo cual se denomina proyección del punto en la imagen. El ángulo comprendido entre rayos obtenidos a partir de diferentes puntos de vista se define como *parallax*.

En la Figura 3.1 se observan dos puntos a distinta profundidad con respecto a la cámara, que son observados desde diferentes puntos de vista. Como se puede ver, el punto más cercano presenta un mayor parallax en relación al punto más lejano ($\beta > \alpha$), por lo tanto se deduce que la distancia al primer punto es menor que la distancia al segundo. Es por esto que existe una relación entre parallax y distancia, lo que permite la estimación de la profundidad de los puntos respecto de la cámara.

Al encontrar un nuevo landmark con una cámara monocular no se puede estimar su profundidad, pues en ese instante el mismo no presenta parallax, ya que ha sido observado desde un único punto de vista. Sin embargo, en el trabajo de Davison *et al.* [39] se plantea representar la posición de un punto de forma alternativa. La idea es utilizar la dirección del rayo entre el centro óptico de la cámara y el landmark junto a un intervalo de incertidumbre Gaussiana. En el método de Davison existe una etapa de inicialización en la que se generan múltiples hipótesis Gaussianas sobre la profundidad del punto en el mundo (distribución multimodal, similar al Particle Filter [44]), que van convergiendo a una dis-

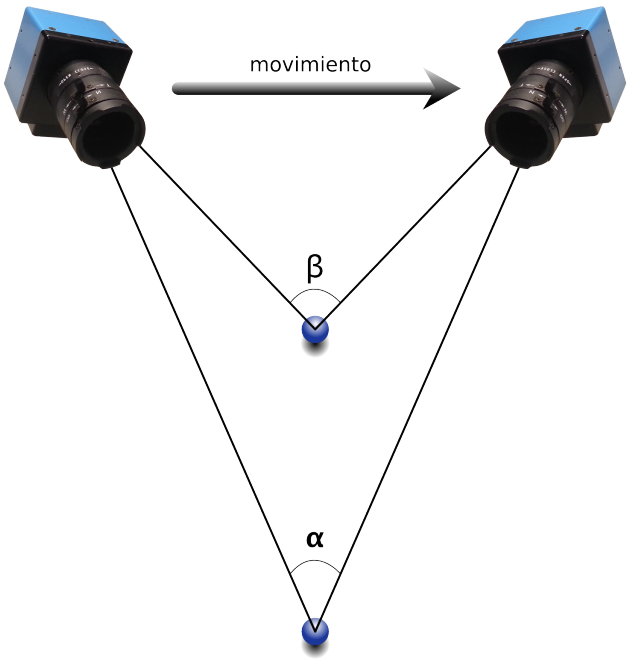


Fig. 3.1: Parallax distinto entre dos puntos del mundo a diferentes distancias respecto de la cámara. Los rayos desde el centro óptico hacia el punto en el mundo forman los ángulos α (punto lejano) y β (punto cercano), con $\alpha < \beta$.

tribución Gaussiana unimodal a medida que pasan las iteraciones del algoritmo, hasta que se lo puede convertir a representación euclidiana XYZ para poder comenzar a utilizarlo en la estimación del movimiento de la cámara (landmark "completamente inicializado"). A esto se lo denomina inicialización retardada.

Una importante limitación de esta estrategia es que la incertidumbre sobre la profundidad, inicializada de esta manera, solamente permite el uso de features correspondientes a puntos en el mundo cercanos a la cámara que presenten alto parallax. En el caso de los features correspondientes a puntos lejanos en el mundo, la distribución Gaussiana resulta ser una mala representación para su profundidad, pues presenta un pico en una distancia determinada y luego decrece lentamente hacia el infinito, lo que imposibilita la convergencia del conjunto de Gaussianas a una única en el infinito. Adicionalmente, el esquema de inicialización retardada no aprovecha los landmarks recientemente detectados hasta varias iteraciones más adelante en el algoritmo.

Son principalmente estos dos problemas los que motivan una distinta representación de la profundidad. Se desea entonces que pueda lidiar con la inicialización de los puntos en el mapa para cualquier profundidad por igual, inclusive a distancia "infinita". También se desea incorporarlos inmediatamente a la estimación para que puedan aportar a la misma desde el primer momento en que fueron detectados. Los landmarks cercanos suelen presentar alto parallax, y son útiles entonces para estimar tanto la traslación como la rotación de la cámara. También son útiles los lejanos, pues por ejemplo, una estrella se verá en la misma posición de la imagen durante kilómetros en el caso de una cámara realizando únicamente movimientos de traslación, por lo que no presentará significante parallax. Sin

embargo, si la cámara realiza movimientos de rotación, la estrella también se verá rotar en la imagen y servirá para actualizar su orientación. No observar parallax de un punto por varias iteraciones en las que la cámara se mueva, sirve para estimar un límite inferior en su profundidad, que dependerá directamente del movimiento estimado.

Para resolver estos problemas se presenta la parametrización Inverse Depth propuesta por Civera *et al.* en [58] y se la comparará con la clásica parametrización euclidiana XYZ.

3.1.1. Parametrización euclidiana XYZ e Inverse Depth

Un punto 3D $\mathbf{y_E}$ en el espacio del mundo se representa con un vector de 3 dimensiones en la parametrización euclidiana XYZ:

$$\mathbf{y_E} = \begin{pmatrix} X & Y & Z \end{pmatrix}^\top \tag{3.1}$$

En cambio, un punto 3D $\mathbf{y_{ID}}$ en el espacio del mundo se representa con un vector de 6 dimensiones en la parametrización Inverse Depth:

$$\mathbf{y_{ID}} = \begin{pmatrix} x & y & z & \theta & \phi & \rho \end{pmatrix}^\top \tag{3.2}$$

El vector $\mathbf{y_{ID}}$ de la ecuación (3.2) define un rayo con origen en el centro óptico de la cámara (x, y, z), con azimuth θ y elevación ϕ. La profundidad (o distancia) d del punto respecto de la cámara sobre ese rayo está representada por su inverso $\rho = 1/d$.

Se utilizará (x, y, z) para representar la posición estimada de la cámara al momento en el que detectó por primera vez el landmark y se inicializa ρ, θ y ϕ como se explica en la sección 3.1.5.

La forma de convertir la parametrización Inverse Depth en XYZ viene dada por la función $f_{ID\to E} : \mathbb{R}^6 \to \mathbb{R}^3$:

$$f_{ID\to E}(\mathbf{y_{ID}}) = \mathbf{y_E} \tag{3.3}$$

$$f_{ID\to E}(\mathbf{y_{ID}}) = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \frac{1}{\rho} m(\theta, \phi) \tag{3.4}$$

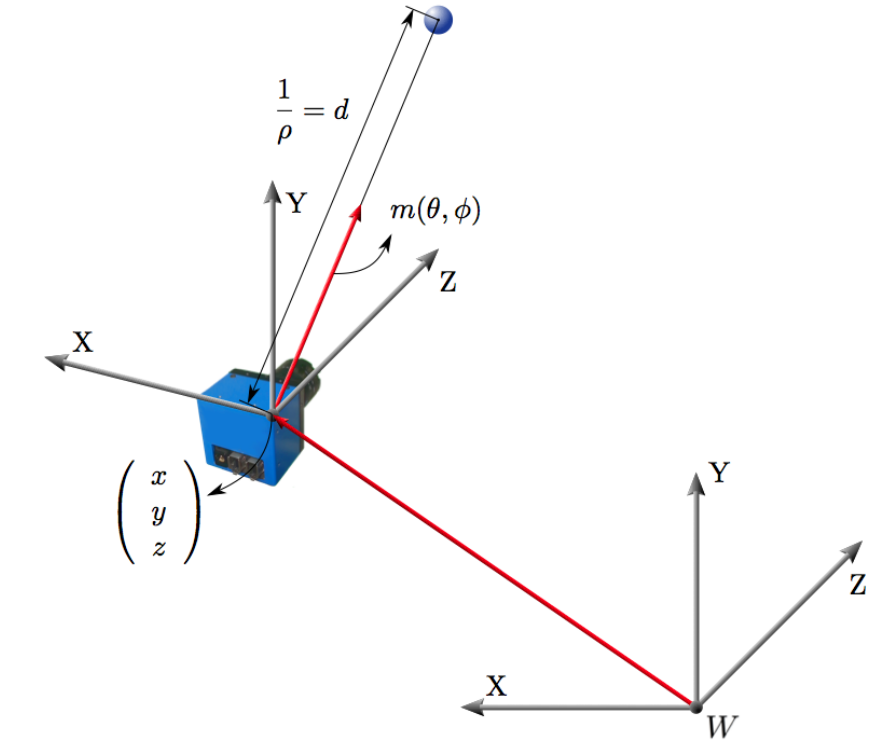
$$m(\theta, \phi) = \begin{pmatrix} \cos\phi \sin\theta, & -\sin\theta, & \cos\phi\cos\theta \end{pmatrix}^\top \tag{3.5}$$

donde el vector $m(\theta, \phi)$ define el vector dirección del rayo.

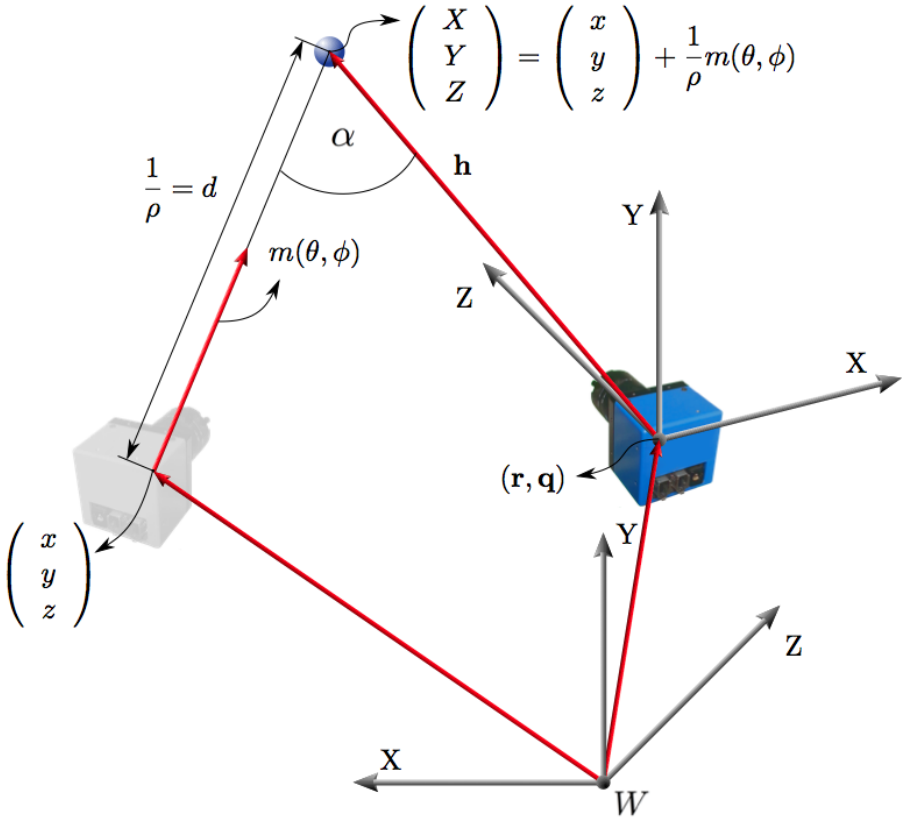
3.1.2. Proyección en la imagen de un punto del mapa

Proyectar un punto del mapa en la imagen es necesario para la predicción de las mediciones, como se detalla luego en el capítulo 4. Dado que existen dos formas de representar los puntos del mapa, existen también dos formas de realizar su proyección al plano de la imagen. A continuación se precisa cómo se realiza dicha tarea.

En el ejemplo de la Figura 3.2, la cámara visualizó por primera vez el landmark $\mathbf{y}$, correspondiente a un punto en el mundo en el instante $t-1$, en el cual la estimación de la posición de la cámara era (x, y, z) y $m(\theta, \phi)$ la dirección desde el centro óptico hacia el punto mencionado (ecuación (3.5)), ambos en relación al sistema de coordenadas del mundo W. En el instante t, la cámara se trasladó a la posición $\mathbf{r} \in \mathbb{R}^3$ con orientación



(a)



(b)

Fig. 3.2: Parametrización inverse depth: (a) la cámara visualiza un punto en el mundo en el instante $t-1$; (b) la cámara visualiza el mismo punto en el mundo en el instante t.

$\mathbf{q} \in \mathbb{R}^4$ representada como un cuaternión, y vuelve a visualizar el mismo punto en el mundo. El ángulo α definido entre ambos rayos corresponde al parallax.

Para realizar un cambio de coordenadas del sistema de coordenadas del mundo al de la cámara en el instante t, se debe calcular $\mathbf{h} \in \mathbb{R}^3$. En el caso de los puntos representados en XYZ:

$$\mathbf{h} = R(\mathbf{y_E} - \mathbf{r}) = R\left(\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} - \mathbf{r}\right) \tag{3.6}$$

En el caso de los puntos representados en Inverse Depth, se utiliza la ecuación (3.3):

$$\mathbf{h} = R(f_{ID\to E}(\mathbf{y_{ID}}) - \mathbf{r}) = R\left(\begin{pmatrix} x \\ y \\ z \end{pmatrix} - \mathbf{r} + \frac{1}{\rho} m(\theta, \phi)\right) \tag{3.7}$$

donde $R \in \mathbb{R}^{3\times 3}$ es la matriz de rotación del sistema de coordenadas del mundo a la orientación del sistema de coordenadas de la cámara (ver Figura 3.2).

Una vez que se obtiene el punto en relación al sistema de coordenadas de la cámara, se lo proyecta en el plano de la imagen según el modelo de cámara Pinhole [53], para obtener el pixel en la imagen sin distorsionar $\mathbf{h}_u$, utilizando los parámetros de calibración que deben ser determinados previamente:

$$\mathbf{h}_u = \begin{pmatrix} u_u \\ v_u \end{pmatrix} = \begin{pmatrix} p_x - \frac{f}{d_x}\frac{h_x}{h_z} \\ p_y - \frac{f}{d_y}\frac{h_y}{h_z} \end{pmatrix} \tag{3.8}$$

donde $\mathbf{h} = (h_x, h_y, h_z)$, f es la distancia focal, (p_x, p_y) el punto principal y dado que los pixels del *CCD* de la cámara no suelen ser cuadrados sino rectangulares, los parámetros d_x y d_y corresponden al ancho y el alto de cada pixel.

Es importante notar que en casos de bajo parallax, ρ tiende a 0. Cuando se realiza la proyección del punto del mundo en la imagen, es posible multiplicar la ecuación (3.7) por ρ para evitar una división por cero, sin afectar los resultados, pues los factores $\frac{h_x}{h_z}$ y $\frac{h_y}{h_z}$ de la ecuación (3.8) correspondiente a la proyección del punto en el plano de la imagen se mantienen inalterados. Entonces, cuando ρ tiende a 0, el comportamiento de esta parametrización es el esperado, pues el valor de la ecuación (3.7) será aproximadamente $\mathbf{h} \approx R(m(\theta, \phi))$ lo que sirve para estimaciones de la orientación.

Por último, debe ser aplicado un modelo de distorsión al punto $\mathbf{h}_u$ proyectado anteriormente, para obtener el pixel en la imagen con distorsión $\mathbf{h}_d$. En esta tesis se utiliza el clásico modelo de dos parámetros de distorsión de fotogrametría [58, 60].

$$\mathbf{h_d} = \begin{pmatrix} u_d \\ v_d \end{pmatrix} = \begin{pmatrix} p_x + (u - p_x)/(1 + \kappa_1 r_d^2 + \kappa_2 r_d^4) \\ p_y + (v - p_y)/(1 + \kappa_1 r_d^2 + \kappa_2 r_d^4) \end{pmatrix} \tag{3.9}$$

$$r_d = \sqrt{(d_x(u - p_x))^2 + (d_y(v - p_y))^2} \tag{3.10}$$

3.1.3. Índice de linealidad

Como es sabido, EKF opera con sistemas no lineales utilizando modelos lineales. Mientras más lineales sean estos modelos, mejor funciona el método. Es por esto que nos interesa

medir la linealidad de la representación de un punto Inverse Depth con respecto a XYZ, pues son parte del modelo de medición. Para esto existe un índice de linealidad [58] para ambos.

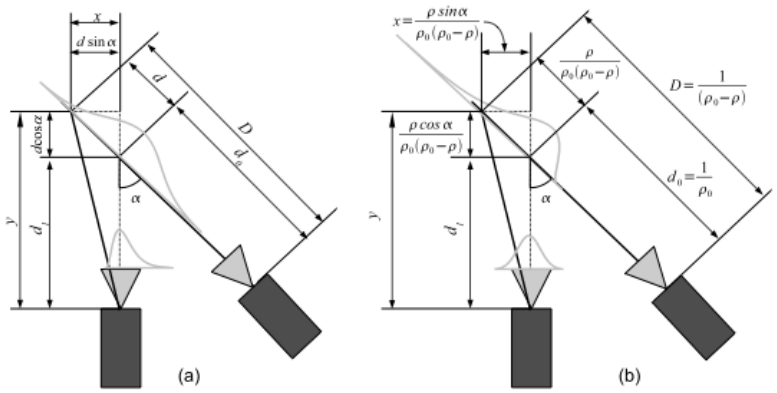


Fig. 3.3: Propagación del error de un punto del mapa: (a) XYZ, (b) Inverse Depth.

Se analiza la linealidad de la representación XYZ con un modelo más simplificado que solamente estima la profundidad del punto en el mundo con respecto a la cámara. En la Figura 3.3 se puede apreciar un punto observado desde dos posiciones distintas, con ángulo de parallax α. En un primer instante se detecta el feature correspondiente a este punto a una distancia estimada d_0, y luego se lo observa a una distancia estimada d_1. El índice de linealidad para ambas representaciones está dado por:

$$L_{XYZ} = \frac{4\sigma_d}{d_1} |\cos(\alpha)| \tag{3.11}$$

$$L_{ID} = \frac{4\sigma_\rho}{\rho_0} \left| 1 - \frac{d_0}{d_1} \cos(\alpha) \right| \tag{3.12}$$

Se sabe que cuando este índice de linealidad $L \approx 0$ esto quiere decir que el modelo es lineal en el intervalo dado [58]. Es importante entonces notar que apenas es inicializado un landmark no suele observarse un parallax significativo, por lo que $d_0 \approx d_1$, y así $\frac{d_0}{d_1} \approx 1$, $cos\alpha \approx 1$, por lo tanto $L_{ID} \approx 0$ y el Inverse Depth es considerado lineal, pero el índice de XYZ es alto. Es por esta superioridad lineal que es conveniente utilizar el Inverse Depth para inicializar los parámetros de los landmarks nuevos detectados.

A medida que el proceso de estimación avanza y se observa mayor parallax para el punto, la estimación mejora gradualmente su precisión, y el índice de la parametrización Inverse Depth continúa teniendo valores bajos, pues el valor de $\left|1 - \frac{d_0}{d_1} \cos(\alpha)\right|$ crece, pero $\frac{4\sigma_\rho}{\rho_0}$ decrece y por lo tanto se compensan. Esto demuestra que la parametrización Inverse Depth es lineal durante todo el proceso de estimación, por lo que es aplicable tanto para landmarks ya existentes como nuevos indiscriminadamente.

3.1.4. Pasaje de Inverse Depth a XYZ

Podría utilizarse únicamente Inverse Depth y el método funcionaría correctamente, pero es por razones de rendimiento que es conveniente convertir los puntos de Inverse

Depth a XYZ, ya que aquellos modelados con la parametrización aquí presentada tienen el doble de tamaño que los XYZ, y por lo tanto el tiempo computacional es mayor al operar con la matriz de covarianza y el estado del método. Además, si bien la parametrización euclidiana XYZ presenta los problemas mencionados anteriormente, si se inician los puntos en Inverse Depth el problema de la inicialización retardada ya no existe, y la profundidad con gran incertidumbre durante los primeros ciclos resulta estar bien representada.

Se propone un umbral para el índice L_{XYZ} que representa el valor a partir del cual la parametrización XYZ es considerada lineal. En el trabajo de Civera *et al.* se realiza una simulación para identificar el umbral $L_{XYZ} < 0{,}1$ para detectar el instante en el que conviene convertir el punto representado en Inverse Depth a XYZ [58]. Cuando el índice de linealidad XYZ de este punto está por debajo de 0,1, se debe utilizar la ecuación (3.3) para realizar dicha conversión.

3.1.5. Inicialización de landmarks con parametrización Inverse Depth

Gracias a la linealidad de la parametrización Inverse Depth para puntos en el mundo que presentan bajo parallax, aquellos landmarks nuevos detectados son agregados al estado inmediatamente. Estos aportarán a la estimación de la ubicación desde un primer momento, principalmente a la orientación de la cámara hasta tanto no presenten un significativo parallax. A medida que la cámara se traslada, mejorará la estimación de la profundidad de los puntos del mapa, y aquellos representados con Inverse Depth comenzarán a contribuir en mayor grado a la estimación de la ubicación de la cámara.

Se debe definir los valores iniciales de la parametrización Inverse Depth (mencionados en la sección 3.1.1) de aquellos puntos nuevos detectados. Estos valores iniciales se pueden expresar en función de:

$$Init_{ID}(\mathbf{r}_t, \mathbf{q}_t, \mathbf{h}_d, \rho_0) = (\hat{x}, \hat{y}, \hat{z}, \hat{\theta}, \hat{\phi}, \hat{\rho})^\top \tag{3.13}$$

donde $\mathbf{r}_t \in \mathbb{R}^3$ es la posición estimada de la cámara en el instante t, $\mathbf{q}_t \in \mathbb{R}^4$ es la orientación representada en cuaterniones, estimada en el instante t, $\mathbf{h}_d = (u_d, v_d)^\top$ es el landmark proyectado en el plano de la imagen (feature con distorsión), y ρ_0 es el valor inicial de la profundidad del punto en el mundo con respecto a la cámara.

La posición desde la que se vio por primera vez el punto es inicializado con la estimación de la posición actual de la cámara, es decir:

$$\begin{pmatrix} \hat{x} & \hat{y} & \hat{z} \end{pmatrix}^\top = \mathbf{r}_t \tag{3.14}$$

La dirección del rayo es calculada a partir de la posición del pixel en la imagen. Para hacerlo primero se debe tener en cuenta su distorsión. Para evitar procesar la imagen completa, se utiliza el modelo de distorsión de dos parámetros ya mencionado previamente en (3.9), para obtener el pixel desdistorsionado $\mathbf{h}_u$ de la siguiente manera:

$$\mathbf{h}_u = \begin{pmatrix} u_u \\ v_u \end{pmatrix} = \begin{pmatrix} p_x + (u_d - p_x) \times (1 + \kappa_1 r_d^2 + \kappa_2 r_d^4) \\ p_y + (v_d - p_y) \times (1 + \kappa_1 r_d^2 + \kappa_2 r_d^4) \end{pmatrix} \tag{3.15}$$

$$r_d = \sqrt{(d_x(u_d - p_x))^2 + (d_y(v_d - p_y))^2} \tag{3.16}$$

y luego se realiza la retroproyección del punto de la imagen desdistorsionado a coordenadas en el mundo

$$\tilde{\mathbf{h}} = R_{q_t} \ (u_u, \ v_u, \ 1)^\top \tag{3.17}$$

donde $\tilde{\mathbf{h}} \in \mathbb{R}^3$ es un vector direccional no unitario en coordenadas del mundo, $R_{q_t} \in \mathbb{R}^{3\times3}$ es la matriz de rotación del sistema de coordenadas de la cámara a la orientación del sistema de coordenadas del mundo, formada a partir del cuaternión $\mathbf{q}_t$ que es la estimación de la orientación de la cámara en el instante t. No es necesario que el vector $\tilde{\mathbf{h}} = (h_x, h_y, h_z)$ sea unitario, ya que finalmente se utilizará para calcular los ángulos de azimuth y elevación θ_i y ϕ_i, como se muestra a continuación:

$$\begin{pmatrix} \hat{\theta} \\ \hat{\phi} \end{pmatrix} = \begin{pmatrix} arctan(h_x, h_z) \\ arctan(-h_y, \sqrt{h_x^2 + h_z^2}) \end{pmatrix} \tag{3.18}$$

$$\hat{\rho} \ = \ \rho_0 \tag{3.19}$$

Finalmente, el valor de ρ_0 y su desvío estándar σ_{ρ_0} son configurados según el trabajo de Civera *et al.* para que tanto distancias cercanas a la cámara como el infinito estén incluidos en el intervalo de confianza del 95 % de la Gaussiana de incertidumbre de la profundidad [58].

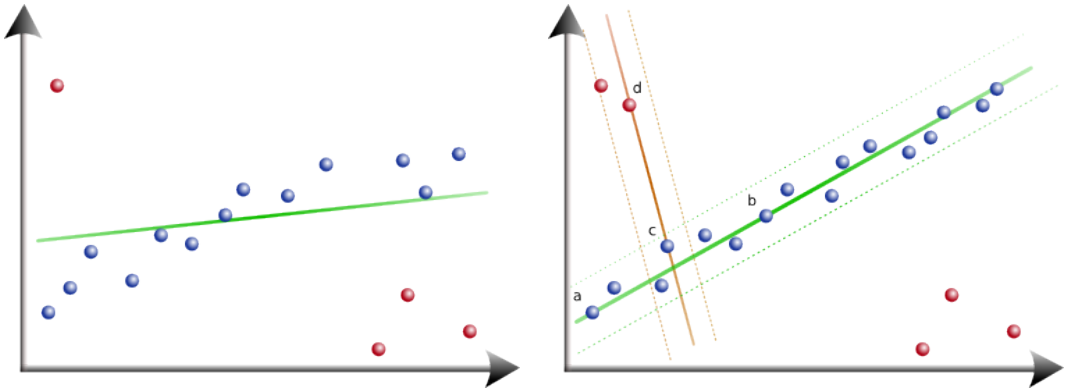
3.2. RANSAC

Dado que las mediciones en todos los sistemas resultan ser inexactas ya que contienen ruido, a menudo incorporarlas a la estimación genera que el método determine de forma incorrecta las acciones realizadas, por lo que es necesario filtrarlos lo más que se pueda para obtener un mejor resultado. En visión por computadora la estimación del movimiento de la cámara se puede realizar mediante las mediciones que detectan puntos dentro de la imagen actual y buscan sus correspondencias (matching) con respecto al frame anterior. Dichas correspondencias en la vida real no resultan ser exáctas, por lo que varias de estas son falsas o distan demasiado de lo que deberían ser. Las que tienen esta característica se denominan *outliers* y los que se consideran buenos para ser incorporados se denominan *inliers*.

El algoritmo RANSAC (RANdom SAmple Consensus) puede lidiar con una gran cantidad de outliers, característica que lo hace muy robusto. Se trata ajustar un modelo utilizando un conjunto de datos y de estos obtener un subconjunto minimal que sea suficiente para determinar el modelo definido. A continuación se mostrará un ejemplo que clarifique algunas de las ideas más importantes del método [53].

En este ejemplo se quiere estimar una línea recta a partir de un conjunto de puntos en un plano tal que minimice la suma de las distancias perpendiculares al cuadrado, con la condición adicional de que los puntos válidos no se desvíen de la recta más que t unidades. Como se puede observar, existen dos problemas que hay que abordar: primero determinar la línea que mejor aproxime al conjunto de datos y el segundo, realizar una clasificación de esta información agrupando los puntos en puntos válidos o inliers (los que se encuentran a distancia menor de t unidades) y puntos inválidos o outliers (aquellos que se encuentran a una distancia mayor que t unidades de la recta), como se puede ver en la Figura 3.4(a). El valor t se denomina umbral y en general se encuentra relacionado al ruido en la medición.

El modelo en este ejemplo es el de una recta de la forma $x' = ax + b$. Por esta razón el subconjunto de puntos minimal suficiente para determinar la recta es de dos puntos. La idea entonces consiste en tomar dos puntos de forma aleatoria, trazar la recta que generan y contar aquellos puntos que se encuentran dentro del umbral t. Este será el conjunto soporte de la recta. Dicho proceso se repetirá una cierta cantidad de veces, y aquel con más soporte se considerará el más robusto. Este proceso entonces toma un subconjunto de datos pequeño al comienzo y luego lo incrementa con datos consistentes siempre y cuando sea posible [53].



(a) (b)

Fig. 3.4: Ejemplo para aproximar una línea. (a): distinción entre puntos inliers (puntos azules) y outliers (puntos rojos). (b): ejemplo de dos conjuntos soporte donde se pueden observar los inliers y outliers de cada uno en función de la línea punteada que indica el umbral.

En la Figura 3.4(a) se muestra un ejemplo de dos líneas tomadas a partir de puntos elegidos al azar: $\langle a, b\rangle$ y $\langle c, d\rangle$, la cantidad de inliers de la línea $\langle a, b\rangle$ es dieciséis y el punto c es outlier, mientras que $\langle c, d\rangle$ tiene un soporte menor de valor 3.

3.2.1. Determinar el valor del umbral

Anteriormente se había mencionado que el valor de t se encontraba relacionado al ruido en la medición, dado que los datos obtenidos provienen de diferentes sensores y por su naturaleza los mismos no son exactos. Se quiere entonces calcular un valor t tal que un dato tenga una probabilidad α de ser inlier. Existen varias formas de calcularlo. En general dicho valor se calcula de forma empírica, pero se puede encaminar una mejor solución asumiendo (siempre que sea posible) que el ruido en la medición es Gaussiano, modelando el ruido como una distribución de ese tipo con media cero y varianza σ [53].

Teniendo en cuenta el ejemplo de la Figura 3.4 y asumiendo una distribución Gaussiana sobre los ruidos de cada una de las mediciones, el cuadrado de la distancia perpendicular a un punto d^2 puede ser calculado como la suma de variables Gaussianas que siguen una distribución χ^2_m, donde m son sus grados de libertad que es igual a la codimensión del modelo. La codimensión del modelo es relativa al espacio en el que se encuentra: si W es un subespacio lineal de un espacio vectorial de dimensión finita V, la codimensión de W en V es $codim(W) = dim(V) - dim(W)$. En este caso, el espacio vectorial V es $\mathbb{R}^2$. Para una recta ($dim(W)$ es 1) la codimensión es 1. Para un modelo que sea un punto ($dim(W)$

es 0), la codimensión es 2.

La probabilidad de que la variable χ^2_m sea menor a un valor k^2 se encuentra dada por la suma de la siguiente distribución chi cuadrado: $F_m(k^2) = \int_0^{k^2} \chi^2_m(\varepsilon)d\varepsilon$. De ésta se puede determinar lo siguiente [53]:

$$\begin{cases} inlier\ d^2 < t^2 \\ outlier\ d^2 >= t^2 \end{cases} con\ t^2 = F_m^{-1}(\alpha)\sigma^2 \tag{3.20}$$

Se puede observar que utilizando la inversa del la función F se puede determinar el valor del umbral buscado según una probabilidad: la probabilidad de que un punto sea inlier, por ejemplo $\alpha = 0{,}95$, indicando que un punto que es inlier va a ser rechazado (tomado como outlier) incorrectamente el 5 % de las veces [53].

3.2.2. Determinar la cantidad de iteraciones

Poder definir la cantidad de iteraciones necesarias para obtener el subconjunto más robusto es muy importante a la hora de utilizar RANSAC. Una opción simple podría ser la de probar todas las combinaciones de los datos eligiendo el que más consenso tenga (aquel con el conjunto de soporte más grande). Dicha opción en muchos casos resulta imposible de realizar, ya que la cantidad de combinaciones es muy grande y se necesitaría demasiado tiempo para probarlas todas, además de que es innecesario porque se puede determinar una cantidad de iteraciones N que garantice con una probabilidad p bastante grande de que dentro de los N subconjuntos de datos tomados exista uno libre de ouliers.

Si la cantidad de datos que tiene cada subconjunto se denota con s y si se define a w como la probabilidad de tomar cualquier dato y sea inlier, entonces se puede deducir lo inverso $\epsilon = 1 - w$ como la probabilidad de que un dato sea outlier. A su vez, asumiendo que los datos se seleccionan independientemente w^n resulta ser la probabilidad de que todos los puntos sean inliers lo que implica que $1 - w^n$ es la probabilidad de que al menos uno de los datos sea outlier.

Como se quiere determinar la cantidad de iteraciones N necesarias para obtener un subconjunto de s datos libre de outliers con cierta probabilidad p, teniendo en cuenta el párrafo anterior se puede definir la siguiente ecuación que indica la probabilidad de que en la N iteraciones el algoritmo nunca seleccionará un subconjunto que contenga datos inliers y que resulta ser el complemento de la probabilidad p definida:

$$(1 - w^s)^N = 1 - p \tag{3.21}$$

Luego resulta simple determinar el valor de N, pues sólo hace falta despejar dicha variable, obteniendo finalmente el valor buscado [53]:

$$N = log(1 - p)/log(1 - (1 - \epsilon)^s) \tag{3.22}$$

3.2.3. Adaptar la cantidad de iteraciones

Existe una forma de acotar la cantidad de iteraciones N cada vez que se selecciona un nuevo subconjunto de datos. Dado que en la mayoría de los problemas la probabilidad de que un dato sea outlier (ϵ) no se conoce, este valor se puede ir ajustando en cada ciclo con cada muestra y viendo la proporción de outliers a partir de la misma. Al comienzo se define una probabilidad de peor caso para ϵ, por ejemplo seteando $\epsilon = 0{,}5$ como probabilidad

inicial y encontrando un subconjunto tal que el 80 % de los datos sean inliers, entonces se puede reducir ϵ a 0,2. Esta reducción en ϵ implica lo mismo en la cantidad de iteraciones total a realizar. Este enfoque tiene una mejora computacional en comparación con el enfoque original.

Algoritmo 4 RANSAC adaptativo

```
N = ínf
iteraciones = 0
Mientras iteraciones < N hacer
    Tomar un subconjunto de datos s y calcular el número de inliers i
    calcular ϵ = (1 − i)/(datos totales)
    adaptar el valor de N función de la probabilidad de inliers deseada
    iteraciones ++
Fin mientras
```

3.3. 1-Point RANSAC

La forma usual para determinar correspondencias entre las diferentes imágenes es comparando los features (puntos característicos) de las diferentes imágenes que se quieran procesar. El método de comparación se realiza utilizando los descriptores de estos features, pero las comparaciones suelen ser erróneas para algunos de los puntos característicos, dado que existen ambigüedades en los descriptores, lo que genera más tarde correspondencias incorrectas.

La forma de disminuir el número de correspondencias erróneas es utilizando el método RANSAC explicado anteriormente. Según el modelo que se quiere ajustar, la cantidad de datos necesarios en cada subconjunto varía, teniendo en cuenta el ejemplo de la Figura 3.4 donde el modelo es una línea, los subconjuntos de puntos serán de tamaño 2. En este enfoque no existe información previa que permita realizar alguna mejora en la búsqueda del subconjunto más robusto, dado que los subconjuntos son generados solamente sobre los datos que se tienen. Pero si se dispone información previa como la distribución de probabilidad para cada uno de los parámetros del modelo, se puede realizar una mejora que consiste en reducir la cantidad de datos en cada uno de los subconjuntos necesaria para instanciar el modelo. Esto impacta directamente en la ecuación (3.22) reduciendo la cantidad de iteraciones totales del algoritmo y por lo tanto su costo computacional.

En la Figura 3.5 se muestra un ejemplo de cómo sería el funcionamiento de 1-Point RANSAC dentro del contexto explicado en la sección 3.2 agregando ahora la suposición de que se tiene un conocimiento previo sobre las variables del modelo $x' = ax + b$. Para instanciar este modelo se necesitarían dos puntos. El conocimiento previo permite fijar uno de estos puntos, lo que permite instanciar la línea sólo con el punto restante.

Esta mejora se aprecia más cambiando de modelo a una estimación visual por ejemplo, donde son necesarios al menos cinco puntos en la imagen para estimar el movimiento del robot teniendo en cuenta los seis grados de libertad para un sistema con una única cámara [61]. En este caso, definiendo los parámetros necesarios en el algoritmo RANSAC original con $p = 0{,}99$ y $w = 0{,}5$ y observando la ecuación (3.22), la cantidad de iteraciones N para encontrar un conjunto libre de outliers con esa probabilidad para una estimación visual sería de 146, mientras que utilizando 1-Point RANSAC se reduce significativamente dicho valor a $N = 7$ [9].

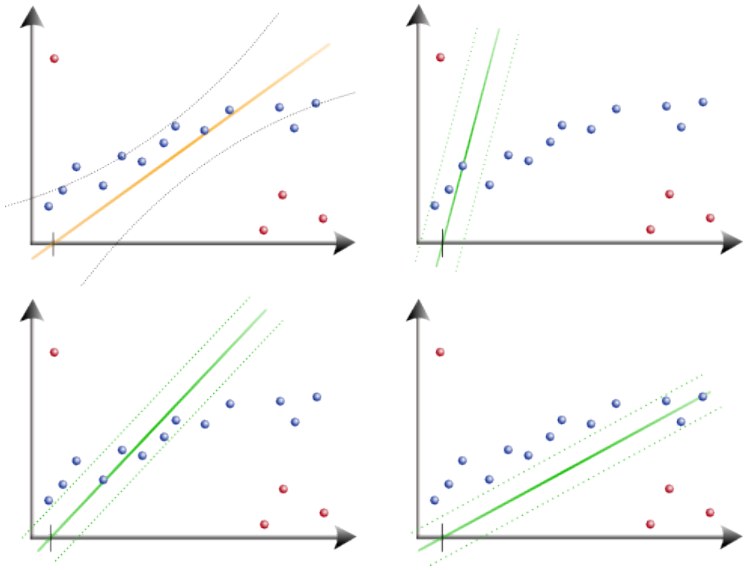


Fig. 3.5: Ejemplo de 1-Point RANSAC al aproximar una recta utilizando información previa. Las imágenes más pequeñas corresponden a distintas iteraciones, en las que se toma un sólo punto para instanciar la recta.

La información previa en un método basado en EKF viene dada por defecto, dado que los parámetros del algoritmo se vienen propagando en el tiempo y siempre se encuentran disponibles, por lo que resulta adecuada la utilización de 1-Point RANSAC en la etapa de eliminación de datos espurios. De esta forma, en el capítulo 4 se explica cómo 1-Point RANSAC puede ser usado en conjunto con EKF, lo que permite realizar predicciones del modelo usando sólo un feature de la imagen.

4. EKF SLAM CON INVERSE DEPTH Y 1-POINT RANSAC

Con los conceptos ya explicados sobre EKF SLAM, Inverse Depth, y 1-Point RANSAC, en este capítulo se los integra en un método de SLAM monocular. Para ello se necesita definir:

1. Representación del Estado: debe especificar todas las variables que se necesita estimar que están asociadas al mapa y la ubicación de la cámara.
2. Modelo Dinámico: realiza la predicción del movimiento que será utilizado para predecir las mediciones.
3. Modelo de Medición: intenta predecir las mediciones y extraer datos útiles del sensado.
4. Modelo de ruidos: los ruidos asociados a las variables del modelo se representan con distribuciones Gaussianas.

4.1. Representación del Estado

El estado del sistema representa una imagen instantánea de variables probabilísticas de la estimación actual de la ubicación de la cámara y de los puntos del mapa junto con su incertidumbre. Es el resultado de la trayectoria de navegación realizada y se corresponde con el ambiente. Persiste información que evoluciona dinámicamente en cada iteración del método, generalmente creciendo cuando se detectan nuevos puntos y reduciéndose cuando se eliminan otros en caso de ser necesario. Dado que su objetivo primario es permitir la localización 3D en tiempo real en lugar de describir completamente el ambiente, se intenta utilizar la cantidad mínima y necesaria de puntos.

El estado en el instante t se representa formalmente con un vector $\mathbf{x}_t$ y una matriz de covarianza P_t

$$\mathbf{x}_t = \begin{pmatrix} \mathbf{x}_{u,t} & \mathbf{y}_{1,t} & \mathbf{y}_{2,t} & \cdots \end{pmatrix}^\top \qquad P_t = \begin{bmatrix} P_{\mathbf{x}_u\mathbf{x}_u,t} & P_{\mathbf{x}_u\mathbf{y}_1,t} & P_{\mathbf{x}_u\mathbf{y}_2,t} & \cdots \\ P_{\mathbf{y}_1\mathbf{x}_u,t} & P_{\mathbf{y}_1\mathbf{y}_1,t} & P_{\mathbf{y}_1\mathbf{y}_2,t} & \cdots \\ P_{\mathbf{y}_2\mathbf{x}_u,t} & P_{\mathbf{y}_2\mathbf{y}_1,t} & P_{\mathbf{y}_2\mathbf{y}_2,t} & \cdots \\ \vdots & \vdots & \vdots & \end{bmatrix} \tag{4.1}$$

$$\mathbf{x}_{u,t} = \begin{pmatrix} \mathbf{r}_t & \mathbf{q}_t & \mathbf{v}_t & \omega_t \end{pmatrix}^\top \tag{4.2}$$

donde $\mathbf{y}_{i,t}$ corresponde a la posición estimada del punto i en el mapa ($1 \leq i \leq k$, siendo k la cantidad de landmarks en el mapa), $\mathbf{x}_{u,t} \in \mathbb{R}^{13}$ donde u refiere a la localización de la cámara, es decir que determina una estimación de los ejes de coordenadas de la cámara en relación los ejes de coordenada del mundo, y define: la posición en el mapa $\mathbf{r}_t \in \mathbb{R}^3$, la orientación representada con un cuaternión $\mathbf{q}_t \in \mathbb{R}^4$, la velocidad lineal $\mathbf{v}_t \in \mathbb{R}^3$ y por último la velocidad angular $\omega_t \in \mathbb{R}^3$. Los $\mathbf{y}_{i,t}$ se representan con la parametrización Inverse Depth hasta convertirlos a la parametrización XYZ según lo explicado en la sección 3.1 y por lo tanto pueden ser vectores $\mathbb{R}^6$ o $\mathbb{R}^3$ respectivamente, por lo tanto la dimensión

final del estado en caso de contener n_{XYZ} puntos con parametrización Euclidiana y n_{ID} puntos representados con parametrización Inverse Depth resulta ser $dim(\mathbf{x}_t) = 13 + 3 \times n_{XYZ} + 6 \times n_{ID}$.

4.2. Modelo Dinámico

El estado se actualiza en dos pasos bien definidos: predicción y actualización, como se muestra en el capítulo 2. El robot se mueve de forma "ciega" entre las capturas de imágenes, y por lo tanto se hace una predicción de su localización en el mapa, y posteriormente se fusiona con el sensado en la actualización. En esta sección se explica la predicción.

Al definir un modelo dinámico es necesario determinar un nivel de detalle, y considerar ruidos que representen las distintas variaciones y discrepancias que puede tener el modelo con respecto al movimiento real. En el caso de robots con ruedas, el modelo debería considerar ciertos factores como pequeños deslizamientos, producto del bajo rozamiento del terreno con las ruedas, entre otros efectos no sistemáticos que generalmente se dan de forma natural. En caso de un robot aéreo que se mueve libremente por el espacio en 3D, se debería tener en cuenta aquellos movimientos desconocidos, por ejemplo por el efecto del viento o de imprecisiones no sistemáticas.

En este trabajo se utiliza un modelo de velocidad constante tanto angular como lineal. Bajo dicho modelo se asume que los movimientos que realiza la cámara son continuos y de velocidad constante durante toda la ejecución del sistema. Además, las aceleraciones (que escapan al modelo de velocidad constante) se modelan como un ruido de distribución Gaussiana. En general se espera que el desplazamiento de la cámara en el espacio sea contínuo, por lo que el método no funcionaría en caso de cambios de velocidad de forma brusca (muy altas aceleraciones positivas o negativas). En resumen, cada iteración del algoritmo asume que se cuenta con ruidos en la velocidad lineal y angular de distribución Gaussiana con media cero, provenientes de impulsos de aceleración de magnitudes desconocidas, y estos errores serán independientes entre sí y de las iteraciones previas.

La predicción de la ubicación de la cámara en el mapa se realiza entonces con el modelo de velocidad constante

$$f(\mathbf{x}_u) = \begin{pmatrix} \mathbf{r}_{t|t-1} \\ \mathbf{q}_{t|t-1} \\ \mathbf{v}_{t|t-1} \\ \omega_{t|t-1} \end{pmatrix} = \begin{pmatrix} \mathbf{r}_{t-1} + (\mathbf{v}_{t-1} + V)\Delta t \\ \mathbf{q}_{t-1} \times q((\omega_{t-1} + \Omega)\Delta t) \\ \mathbf{v}_{t-1} + V \\ \omega_{t-1} + \Omega \end{pmatrix} \tag{4.3}$$

$$\mathbf{n} = \begin{pmatrix} V \\ \Omega \end{pmatrix} = \begin{pmatrix} a\Delta t \\ \alpha\Delta t \end{pmatrix} \tag{4.4}$$

donde a y α representan aceleraciones lineal y angular respectivamente (que se asumen de distribución Gaussiana de media cero) que aplicadas durante un tiempo Δt, conforman el ruido en la velocidad lineal y angular V y Ω respectivamente. La notación $q(\gamma)$ denota el cuaternión equivalente a un vector γ compuesto por ángulos de rotación sobre cada eje en $\mathbb{R}^3$. La notación $\mathbf{r}_{t|t-1}$, $\mathbf{q}_{t|t-1}$, $\mathbf{v}_{t|t-1}$, $\omega_{t|t-1}$ denota la predicción de cada una de las variables de la ubicación de la cámara en el mapa $\mathbf{x}_u$ (explicado en la sección 4.1) en el instante t a partir de la información del paso anterior $t-1$.

La predicción del nuevo estado debe estar acompañada de un incremento en la incertidumbre hasta incorporar una nueva medición a la estimación. Este se lo calcula de la

siguiente manera:

$$E_{\mathbf{n}} = \frac{\partial f}{\partial \mathbf{n}} P_{\mathbf{n}} \frac{\partial f}{\partial \mathbf{n}}^{\top} \tag{4.5}$$

donde $P_{\mathbf{n}}$ es la matriz de covarianza del vector $\mathbf{n}$, que resulta ser diagonal dado que V y Ω, como se dijo, son variables probabilísticas independientes de distribución Gaussiana y media cero. Más adelante en la sección 4.5.1 se detallará el papel que cumple E_v en el método.

Es importante destacar que la tasa de crecimiento del ruido en cada paso depende en gran parte de los valores de ruidos en las aceleraciones lineal y angular, y esto influye directamente en la precisión de la odometría que se desea obtener. Estas magnitudes definen qué tan robusto será el método ante las variaciones bruscas en la velocidad. Contar con valores bajos significa que los movimientos estimados serán suaves, pero expuestos a errores de estimación ante grandes variaciones de velocidad. Por el contrario, con valores altos se obtendrá un crecimiento significante de la incertidumbre en cada paso, por lo que la odometría será menos precisa pero podrá lidiar con aceleraciones bruscas. Altas incertidumbres pueden contrarrestarse realizando mayor cantidad de mediciones, aunque esto implica que cada paso tendrá un mayor costo computacional.

4.3. Modelo de Medición

El Modelo de Medición para este trabajo está basado en el modelo de cámara Pinhole como se explicó en la subsección 3.1.2 agregando además un modelo de distorsión de dos parámetros [60].

Este modelo de medición se utiliza para realizar la retroproyección de los puntos detectados por primera vez, desde la imagen al mapa del sistema y además para tomar los puntos estimados que se tienen en el mapa y proyectarlos en el plano de la imagen, cuando se realiza el paso de predicción.

Para retroproyectar los puntos que se detectan por primera vez y que no se encuentran en el mapa del sistema, es necesario inicializar el punto dentro del mapa, tarea que requiere de los siguientes pasos:

1. Desdistorsionar el punto según los parámetros intrínsecos de la cámara, siendo $\mathbf{h}_d = (u_d, v_d)^{\top}$ las coordenadas del feature en la imagen y $(u_u, v_u)^{\top}$ las coordenadas del feature sin distorsión, utilizando el modelo de dos parámetros κ_1 y κ_2 como se explica en la ecuación (3.15):

$$\mathbf{h}_u = \begin{pmatrix} u_u \\ v_u \end{pmatrix} = \begin{pmatrix} p_x + (u_d - p_x) \times (1 + \kappa_1 r_d^2 + \kappa_2 r_d^4) \\ p_y + (v_d - p_y) \times (1 + \kappa_1 r_d^2 + \kappa_2 r_d^4) \end{pmatrix}$$

$$r_d = \sqrt{(d_x(u_d - p_x))^2 + (d_y(v_d - p_y))^2}$$

2. Obtener la dirección del rayo de origen en el centro de la cámara y pasa por el punto $\mathbf{h}_u$, despejando las variables $\frac{h_x}{h_z}$ y $\frac{h_y}{h_z}$ de la ecuación (3.8):

$$((u_u - p_x)\frac{d_x}{f}, (v_u - p_y)\frac{d_y}{f}, 1)^{\top} \tag{4.6}$$

Como dicho vector se encuentra referenciado desde el eje de coordenadas de la cámara, es necesario describirlo con respecto al eje de coordenadas del mundo según la ecuación (3.17).

3. Inicializar el landmark con su parametrización Inverse Depth y agregarlo al mapa del sistema utilizando lo explicado en la sección 3.1.5. Por otro lado, la matriz de covarianza $P \in \mathbb{R}^{n\times n}$ debe ser modificada al inicializar este nuevo landmark, agregando seis filas y seis columnas, obteniendo una nueva matriz P'. Para esto se utiliza la matriz $J \in \mathbb{R}^{(n+6)\times(n+3)}$ correspondiente al Jacobiano de la función $Init_{ID}(\mathbf{r}_t, \mathbf{q}_t, \mathbf{h}, \rho_0) = (\hat{x}_i, \hat{y}_i, \hat{z}_i, \hat{\theta}_i, \hat{\phi}_i, \hat{\rho}_i)^\top \in \mathbb{R}^6$ instanciada en el nuevo landmark i, mencionada en la sección 3.1.5 de la siguiente manera:

$$P' = J \begin{pmatrix} P_{n\times n} & \mathbf{0}_{1\times 2} & 0 \\ \mathbf{0}_{2\times 1} & R_i & 0 \\ 0 & 0 & \sigma^2_{\rho_0} \end{pmatrix} J^\top \tag{4.7}$$

$$R_i = \begin{pmatrix} \sigma_{PixelX} & 0 \\ 0 & \sigma_{PixelY} \end{pmatrix} \tag{4.8}$$

$$J = \begin{pmatrix} I_{n\times n} & \mathbf{0}_{n\times 3} \\ \frac{\partial Init_{ID}}{\partial r_t}, \frac{\partial Init_{ID}}{\partial q_t}, \mathbf{0}_{6\times(n-7)} & \frac{\partial Init_{ID}}{\partial h}, \frac{\partial Init_{ID}}{\partial \rho_0} \end{pmatrix} \tag{4.9}$$

donde $R_i \in \mathbb{R}^{2\times 2}$ es la matriz de covarianza que modela el error de medición tal que σ_{PixelX} y σ_{PixelY} son los parámetros de *Pixel Error* de la calibración de la cámara, σ_{ρ_0} es el desvío estándar del parámetro ρ_i mencionado en la sección 3.1, y el resultado $P' \in \mathbb{R}^{(n+6)\times(n+6)}$ contiene las 6 columnas y 6 filas adicionales del nuevo punto representado en Inverse Depth con respecto a P.

En el caso inverso, para obtener las coordenadas dentro de la imagen de los puntos que se encuentran en el sistema, se tienen que realizar los siguientes tres pasos:

1. Cambiar el eje de coordenadas de cada uno de los puntos desde el mundo a la cámara, como lo hacen las ecuaciones (3.6) y (3.7) tanto para los puntos representados con la parametrización Inverse Depth como para los representados con parametrización XYZ.

2. Determinar las coordenadas dentro del plano de la imagen como se describe en la ecuación (3.8).

3. Distorsionar los puntos proyectados según la calibración de la cámara con el modelo de dos parámetros κ_1 y κ_2 realizando el despeje de las variables u_d y v_d como se indica en la ecuación (3.9).

De esta forma es posible agregar nuevos landmarks al sistema y además identificar correspondencias entre los puntos del mapa y la información sensada desde la cámara.

4.4. Factor de escala

Los métodos de Localización y Mapeo Simultáneo (SLAM) basados en visión monocular, permiten estimar el movimiento de la cámara. Para reconstruir el entorno observado en escala real del mundo, algunos métodos requieren un parámetro de escala conocido de antemano para inicializar el sistema. En contraposición, el modelo utilizado en este trabajo (ver sección 4.1) tiene como ventaja que no requiere dicha información a priori.

Para lograr esto el presente método divide el estado en dos partes. La primera consiste en un estado sin dimensión conteniendo el mapa y la ubicación del robot a menos de un factor de escala y la segunda parte, la escala propiamente dicha [62].

Este factor que a priori es desconocido se puede calcular a posteriori gracias a la información previa que manejan los estimadores probabilísticos como EKF, por ejemplo una cámara calibrada, un modelo de movimiento conocido y la profundidad del entorno entre otros.

Definir un estado sin dimensión es posible utilizando el Teorema Π de Buckingham [63] sobre el análisis dimensional, demostrando que las leyes físicas son independientes de las unidades, por lo tanto el proceso de estimación también cumple con dicha afirmación.

4.4.1. Teorema Π de Buckingham

Dada una ecuación $\mathcal{F}(X_1, X_2, ..., X_n) = 0$ que involucra n variables con escalas de diferentes tipos, la relación entre las diferentes variables puede ser expresada como otra función $\hat{\mathcal{F}}(\Pi_1, \Pi_2, ..., \Pi_{n-k}) = 0$ donde cada Π_i es un conjunto reducido de $n-k$ grupos de variables que no tienen información de escala, con k indicando el número de cantidades físicas dimensionalmente independientes dentro del problema.

Teniendo en cuenta lo anterior y el modelo de representación del estado explicado en (4.1), el proceso de estimación puede ser expresado como una función que recibe parámetros con información de magnitudes físicas:

$$\begin{pmatrix} \mathbf{x}_{u,t} & \mathbf{y}_{1,t} & \mathbf{y}_{2,t} & \cdots \end{pmatrix}^\top = \mathcal{F}(\sigma_a, \sigma_\alpha, \sigma_{\mathbf{z}}, \mathbf{z}, \Delta t, \rho_0, \sigma_{\rho_0}, \sigma_{v0}, \sigma_{w0}) \tag{4.10}$$

donde σ_a, σ_α, $\sigma_{\mathbf{z}}$ representan ruidos en la aceleración lineal, aceleración angular y medición respectivamente; Δt representa un lapso de tiempo; ρ_0 es el valor de la estimación inicial de la profundidad de los puntos en el mapa; σ_{ρ_0}, σ_{v0} y σ_{w0} definen el desvío estándar de los valores iniciales para la profundidad de los puntos en el mapa, la velocidad lineal y la velocidad angular, respectivamente. El vector $\mathbf{z}$ incluye todos los features obtenidos a lo largo de la secuencia de imágenes y cada una de las variables tiene asociada una unidad que se muestra a continuación en la siguiente Tabla [62]:

variable	$\mathbf{r}$	$\mathbf{q}$	$\mathbf{v}$, σ_{v0}	$\omega, \sigma_{\omega 0}$	$\mathbf{z}, \sigma_{\mathbf{z}}$	σ_a	σ_α	x_i, y_i, z_i	θ_i, ϕ_i	$\rho_i, \rho_0, \sigma_{\rho 0}$
unidad	l	1	lt^{-1}	t^{-1}	1	lt^{-2}	t^{-2}	l	1	l^{-1}

donde l y t corresponden a unidades de distancia y tiempo respectivamente; $\mathbf{r}$, $\mathbf{q}$, $\mathbf{v}$, ω corresponden a las variables de estimación de la ubicación de la cámara en el mapa (ver sección 4.1); σ_{v0}, $\sigma_{\omega 0}$ definen el ruido inicial de la velocidad lineal y la velocidad angular; x_i, y_i, z_i, θ_i, ϕ_i, ρ_i, se relacionan con las variables de un punto del mapa representado con parametrización Inverse Depth (ver sección 3.1).

Con las variables y las unidades asociadas, se pueden determinar los grupos dimensionales independientes. Las unidades l (distancia) y Δt (tiempo) son las involucradas en

todas las variables y a partir de estas se pueden definir los grupos sin dimensión como sigue [62]:

$\Pi_{\mathbf{r}}$	$\Pi_{\mathbf{v}}$	Π_{ω}	$\Pi_{\sigma_{v0}}$	$\Pi_{\sigma_{\omega 0}}$	$\Pi_{\mathbf{z}}$	$\Pi_{\sigma_{\mathbf{z}}}$	$\Pi_{\sigma a}$	$\Pi_{\sigma\alpha}$	Π_{r_i}	Π_{ρ}	$\Pi_{\sigma\rho 0}$
$\mathbf{r}\rho_0$	$\mathbf{v}\rho_0\Delta t$	$\omega\Delta t$	$\sigma_{v0}\rho_0\Delta t$	$\sigma_{\omega 0}\Delta t$	$\mathbf{z}$	$\sigma_{\mathbf{z}}$	$\sigma_a\rho_0\Delta t^2$	$\sigma_\alpha\Delta t^2$	$r_i\rho_0$	ρ_i/ρ_0	$\sigma_{\rho 0}/\rho_0$

Con estos nuevos grupos se redefinen las ecuaciones del modelo explicado en (4.1), reemplazando todas las variables por las nuevas sin dimensión. La información de la cámara y los landmarks quedan como sigue:

$$\mathbf{x}_u = (\Pi_{\mathbf{r}}, \mathbf{q}, \Pi_{\mathbf{v}}, \Pi_{\omega})^\top \quad y \quad \Pi_{\mathbf{y}_i} = (\Pi_{r_i}, \sigma_i, \phi_i, \Pi_{\rho_i})^\top \tag{4.11}$$

El modelo dinámico detallado en 4.2 se puede redefinir para que sea un modelo sin información de escalas ni dimensiones:

$$f(\mathbf{x}_u) = \begin{pmatrix} \Pi_{r_{k+1}} \\ q_{k+1} \\ \Pi_{v_{k+1}} \\ \Pi_{\omega_{k+1}} \end{pmatrix} = \begin{pmatrix} \Pi_{r_k} + \Pi_{v_k} + \Pi_{a_k} \\ q_k \times q(\Pi_{\omega_k} + \Pi_{\alpha_k}) \\ \Pi_{v_k} + \Pi_{\alpha_k} \\ \Pi_{\omega_k} + \Pi_{\alpha_k} \end{pmatrix} \tag{4.12}$$

Las ecuaciones que sirven para convertir un parámetro a Inverse Depth y expresarlo con respecto al eje de coordenadas de la cámara como está descripto en la subsección 3.1.1 también se pueden redefinir utilizando las nuevas variables libres de dimensiones:

$$\Pi'_h = \Pi_{r_i} + \Pi_{\rho_i} m(\theta_i, \phi_i) \quad y \quad \Pi_h = R(\Pi'_h - \Pi_r) \tag{4.13}$$

La proyección a la imagen se redefine como:

$$\mathbf{h}_{img} = \left(\begin{array}{cc} \dfrac{\Pi_h|_x}{\Pi_h|_z} & \dfrac{\Pi_h|_y}{\Pi_h|_z} \end{array} \right)^\top \tag{4.14}$$

donde $\Pi_h|_\gamma$ denota la componente γ de Π_h.

Con todos estos elementos, la escala puede ser recuperada relacionando información de las mediciones con la del mundo real. De esta forma, se pueden tomar las distancias entre dos puntos medidos y los asociados en el mundo, obteniendo el factor de escala de todo el mapa. Si P_1 y P_2 son los puntos que se encuentran en el mundo y la función $D(P_1, P_2)$ calcula la distancia en alguna unidad entre estos, entonces el factor de escala d se puede calcular despejando la siguiente ecuación:

$$D(P_1, P_2) = d\sqrt{(\Pi_{y2}|_x - \Pi_{y1}|_x)^2 + (\Pi_{y2}|_y - \Pi_{y1}|_y)^2 + (\Pi_{y2}|_z - \Pi_{y1}|_z)^2} \tag{4.15}$$

Esto permite redimensionar el mapa a la escala que se necesite y a las unidades que se requieran.

4.5. El método

A continuación, y en las secciones siguientes se describe el método reflejando lo que se encuentra implementado en el código fuente, utilizando pseudocódigos para transmitir los conceptos más generales que luego se desarrollarán en más detalle.

El funcionamiento del método se encuentra guiado por una serie de funciones que engloban todos los conceptos vistos hasta el momento y que permiten el correcto funcionamiento de un sistema probabilístico de estimación basado en EKF. En el pseudocódigo 5 se puede observar la estructura de funcionamiento en el nivel más alto.

Algoritmo 5 EKF

1: $imagen \leftarrow obtenerImagen()$
2: $\mathbf{x}, P \leftarrow EKFInicializar(imagen)$
3: **Mientras** *haya nueva imagen* **hacer**
4: $\quad imagen \leftarrow obtenerImagen()$
5: $\quad \mathbf{x}, P \leftarrow EKFPaso(imagen, \mathbf{x}, P)$
6: **Fin mientras**

Con la primera imagen disponible luego de que se comienza el proceso es necesario inicializar el sistema. Esto consiste en definir los ruidos iniciales del modelo, detectar los primeros landmarks para agregar al sistema, entre otras tareas que se detallarán más adelante. Luego se ingresa en el ciclo de estimación donde se procesarán las siguientes imágenes asociadas a cada uno de los instantes de tiempo en que se realiza el sensado desde la cámara.

La inicialización del sistema es un paso muy importante ya que durante el mismo se definen todos los parámetros necesarios del modelo y que a su vez fijan el comportamiento de este. En el algoritmo 6, que se corresponde con la línea 2 del pseudocódigo 5, se muestran los pasos necesarios para iniciar el método.

Algoritmo 6 EKFInicializar(imagen) : $\mathbf{x}_0$, P_0

1: $\mathbf{x}_{init} \leftarrow (0, 0, 0, 1, 0, 0, 0, 0, 0, 0, \epsilon, \epsilon, \epsilon)$
2: $P_{init} \leftarrow Diag(\epsilon, \epsilon, \epsilon, \epsilon, \epsilon, \epsilon, \sigma_{v0}, \sigma_{v0}, \sigma_{v0}, \sigma_{\omega 0}, \sigma_{\omega 0}, \sigma_{\omega 0})$
3: $\mathbf{s} \leftarrow$ DetectarFeatures($imagen$, $\emptyset$)
4: $\mathbf{x}_0, P_0 \leftarrow$ AgregarFeatures($\mathbf{s}$, $\mathbf{x}_{init}$, P_0)

El estado inicial define los valores relacionados a la cámara: la posición en tres dimensiones con respecto al eje de coordenadas del mundo, que se establece como el origen de coordenadas de dicho espacio. La orientación se define con el cuaternion $(1, 0, 0, 0)$ y las velocidades lineales y angulares en cero, dado que al comienzo se asume que la cámara no se encuentra en movimiento, aunque para evitar problemas numéricos, la velocidad angular se define con un ϵ muy cercano a cero.

La covarianza inicial a su vez contiene las incertidumbres sobre cada una de las variables mencionadas en el estado inicial. La misma es una matriz cuadrada perteneciente a $\mathbb{R}^{13\times 13}$ diagonal, conteniendo ϵ para los lugares asociados a la posición de la cámara y la orientación, mientras que los valores asociados a las velocidades lineal y angular tienen que ver con el ruido inicial definido de antemano (σ_{v0} y $\sigma_{\omega 0}$).

Terminada la etapa de inicialización para los valores de la cámara, es necesario realizar un sensado y a partir de la imagen obtenida, extraer los primeros puntos de interés que servirán para realizar la estimación de los siguientes pasos. Al iniciar el sistema se debe realizar un primer sensado, pues de no hacerlo implicaría que el primer paso del ciclo realizará una predicción que luego no se pueda corregir, ya que no hay forma de asociar el paso anterior con la información sensada actualmente. Esto incorporaría un error al

sistema que nunca podría ser eliminado. La forma de detectar los features en la primera imagen se describe en detalle en la sección 5. Los puntos de interés obtenidos en la imagen se agregan al estado y a la matriz de covarianza en su representación Inverse Depth como se explica en 3.1.5.

Una vez definido el estado con los valores iniciales se ingresa al ciclo de procesamiento indicado entre las líneas 3 y 6 del algoritmo 5. Con cada imagen disponible se realizan los pasos descriptos en el algoritmo 7.

Algoritmo 7 EKFPaso(imagen, $\mathbf{x}_{t-1}$, P_{t-1}) : $\mathbf{x}_t$, P_t

1: $\mathbf{x}_{t|t-1}, P_{t|t-1} \leftarrow$ EKFPrediccionCamara($\mathbf{x}_{t-1}$, P_{t-1})
2: $\mathbf{z}_{t|t-1} \leftarrow$ EKFPrediccionMediciones($\mathbf{x}_{t|t-1}$, $P_{t|t-1}$)
{Se buscan features dentro de las elipses de predicción}
3: $\mathbf{s} \leftarrow$ DetectarFeatures(imagen, $\mathbf{z}_{t|t-1}$)
4: $\mathbf{z}_t \leftarrow$ BuscarCorrespondencias(imagen, $\mathbf{z}_{t|t-1}$, $\mathbf{s}$)
{Se determina el conjunto de máximo soporte utilizando 1-Point RANSAC}
5: **zInliers**, **zOutliers** $\leftarrow$ filtrarOutliers($\mathbf{z}_t$, $\mathbf{x}_{t|t-1}$, $P_{t|t-1}$)
6: $\mathbf{x}_t, P_t \leftarrow$ EKFActualizar(**zInliers**, $\mathbf{x}_{t|t-1}$, $P_{t|t-1}$)
{Se realiza una nueva predicción con los outliers para realizar el rescate}
7: $\mathbf{xOutliers}_t \leftarrow \langle \mathbf{x}_{u,t}, \mathbf{y}_{i,t} \rangle / \mathbf{y}_{i,t}$ es un outlier asociado a algún **zOutliers**
8: **prediccionDeOutliers** $\leftarrow$ EKFPrediccionMediciones($\mathbf{xOutliers}_t$, P_t)
9: **zInliers** $\leftarrow$ rescatarOutliers(**prediccionDeOutliers**)
10: $\mathbf{x}_t, P_t \leftarrow$ EKFActualizar(**zInliers**, $\mathbf{x}_t$, P_t)

En *EKFPaso* se observan las funciones principales encargadas de obtener nueva información desde la imagen, procesarla e incorporarla al sistema de forma tal que mejore la estimación. Las líneas 1 y 2 realizan los pasos de predicción para los valores de la cámara (posición, orientación y velocidades) y para todos los puntos en el mapa que servirán para realizar la Búsqueda Activa (*Active Search*) de los mismos dentro de la imagen como se explica en la sección 4.5.1. Para obtener la información de la imágen, es necesario realizar una detección de features y luego buscar las correspondencias con respecto a los puntos del mapa proyectados en la imágen, como se detalla en las secciones 4.5.2 y 4.5.3.

Posteriormente se deben quitar las correspondencias espurias que incorporen información nociva al sistema utilizando el método 1-Point RANSAC, cuyo procedimiento se precisa en la sección 4.5.4. Luego de esto es necesario actualizar el mapa.

Finalmente en la etapa de actualización se ponderan las mediciones con respecto a las predicciones. La misma es realizada en base a las correspondencias con mayor soporte calculadas con 1-Point RANSAC, realizando una etapa de rescate sobre correspondencias que fueron etiquetadas como outliers y culminando con una actualización final que resulta en la estimación para el instante t. Los detalles de actualización se encuentran explicados en la sección 4.5.5

4.5.1. Predicción del estado

El ciclo del método comienza con una predicción de la localización de la cámara en el último mapa estimado en la iteración anterior. Para realizar esta predicción se utiliza el modelo dinámico explicado en la sección 4.2 que utiliza la última estimación para calcular la ubicación actual del robot partiendo de la suposición que el mismo viaja a velocidades

lineal y angular constantes, considerando errores no sistemáticos en ambas.

Se adopta la notación $\mathbf{x}_{t|t-1}$ y $P_{t|t-1}$ para referenciar la predicción del estado del sistema en el instante t a partir de la estimación en el instante anterior $t-1$

$$\mathbf{x}_{t|t-1} = \begin{pmatrix} \mathbf{x}_{u,t|t-1} & \mathbf{y}_{1,t-1} & \mathbf{y}_{2,t-1} & \cdots \end{pmatrix}^\top \tag{4.16}$$

$$P_{t|t-1} = \begin{bmatrix} P_{\mathbf{x}_u\mathbf{x}_u,t|t-1} & P_{\mathbf{x}_u\mathbf{y}_1,t|t-1} & P_{\mathbf{x}_u\mathbf{y}_2,t|t-1} & \cdots \\ P_{\mathbf{y}_1\mathbf{x}_u,t|t-1} & P_{\mathbf{y}_1\mathbf{y}_1,t-1} & P_{\mathbf{y}_1\mathbf{y}_2,t-1} & \cdots \\ P_{\mathbf{y}_2\mathbf{x}_u,t|t-1} & P_{\mathbf{y}_2\mathbf{y}_1,t-1} & P_{\mathbf{y}_2\mathbf{y}_2,t-1} & \cdots \\ \vdots & \vdots & \vdots & \end{bmatrix} \tag{4.17}$$

donde, si $\mathbf{x}_{t-1}$ y P_{t-1} es la estimación en el instante anterior (definido en la sección 4.1), la predicción $\mathbf{x}_{t|t-1}$ y $P_{t|t-1}$ de la estimación actual se calcula modificando las variables que involucran la ubicación de la cámara. Notar que el estado y la covarianza de los puntos del mapa ($\mathbf{y}_1$, $\mathbf{y}_2$, ... y $P_{\mathbf{y}_1\mathbf{y}_1}$, $P_{\mathbf{y}_2\mathbf{y}_1}$, $P_{\mathbf{y}_1\mathbf{y}_2}$, $P_{\mathbf{y}_2\mathbf{y}_2}$, ...) se mantienen iguales al estado anterior, pues la predicción de estos valores se calcula posteriormente.

Se define $n = dim(x_{t|t-1})$ que refiere a la dimensión del estado, con $P_{t|t-1} \in \mathbb{R}^{n\times n}$. La predicción $\mathbf{x}_{t|t-1}$ y $P_{t|t-1}$ se calcula propagando la Gaussiana $X_{t-1} \sim N(x_{t-1}, P_{t-1})$ al instante t mediante el modelo dinámico f

$$\mathbf{x}_{t|t-1} = f(\mathbf{x}_{t-1}) \tag{4.18}$$

$$P_{t|t-1} = F_t P_{t-1} F_t^\top + G_t Q_t G_t^\top \tag{4.19}$$

$$F_t = \begin{pmatrix} F_t' & 0 \\ 0 & I \end{pmatrix} \qquad G_t = \begin{pmatrix} G_t' \\ 0 \end{pmatrix} \qquad Q_t = \Delta t \begin{pmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_\omega^2 \end{pmatrix}$$

donde $F_t' \in \mathbb{R}^{13\times 13}$ es el Jacobiano de f con respecto a la predicción de la ubicación de la cámara $\mathbf{x}_{u,t|t-1}$, $Q_t \in \mathbb{R}^{6\times 6}$ es la matriz de covarianza que representa el error de las velocidades lineal y angular en el instante t del modelo dinámico utilizado, y $G_t' \in \mathbb{R}^{13\times 6}$ es el Jacobiano de f con respecto al ruido Gaussiano V y Ω en el instante t. Las matrices G_t y Q_t aportan al incremento de la incertidumbre, que fue denotado como E_v en la sección 4.2.

Esta predicción de la localización junto con los puntos guardados en el mapa sirven para facilitar la búsqueda de los mismos en la imagen del instante actual. Con estos datos es posible predecir con cierta incertidumbre Gaussiana el área en la que se podrán detectar los puntos del mapa en la imagen. Esta predicción de las mediciones se corresponde con el conjunto $\mathbf{z}_{t|t-1}$ de la línea 2 del pseudocódigo 7. La búsqueda de los puntos del mapa en áreas determinadas por elipses de incertidumbre Gaussiana se la conoce con el nombre de Active Search o Búsqueda Activa [39, 9]. Se propaga el estado predicho con el modelo de medición $\mathbf{h}_i$ para el instante t y el punto i del mapa

$$\mathbf{h}_{i,t} = h_d(x_{t|t-1}) \tag{4.20}$$

$$S_{i,t} = H_i P_{t|t-1} H_i^\top + R_i \tag{4.21}$$

donde $\mathbf{h}_{i,t} \in \mathbb{R}^{2\times 1}$ denota la proyección en la imagen del punto i del mapa calculando su punto distorsionado h_d según se detalló en la sección 4.3, $H_i \in \mathbb{R}^{2\times n}$ es el Jacobiano de

h_d con respecto al estado $\mathbf{x}_{t|t-1}$ y $R_i \in \mathbb{R}^{2\times2}$ es la matriz de covarianza que representa el error modelado para cada medición. La matriz $S_{i,t} \in \mathbb{R}^{2\times2}$ determina entonces la matriz de covarianza correspondiente a la proyección del punto i del mapa en el instante t. Esto significa que en el instante t el landmark $y_{i,t}$ debe ser buscado dentro de la elipse definida por el intervalo de confianza de 99 % de probabilidad de la Gaussiana $N \sim (h_{i,t}, S_{i,t})$, como se puede ver en la Figura 4.1

Fig. 4.1: Predicción de los puntos del mapa en el instante t. Se pueden ver elipses rojas correspondientes al intervalo de confianza de 99 % de $S_{i,t}$ con un + en su interior correspondiente a su media $h_{i,t}$. El número al costado de las elipses indica que es la predicción correspondiente al landmark $y_{i,t}$. La correspondencia del feature asociado a este landmark debe ser buscada dentro de la elipse calculada.

La Búsqueda Activa de las mediciones conlleva una significativa mejora en cuanto a velocidad de cómputo, pues restringe el área de detección de features en la imagen actual, lo que puede resultar costoso dependiendo del método utilizado para la detección. También limita la cantidad de búsqueda de correspondencias a solamente un subconjunto de features de los que se pueden detectar en la imágen, lo cual implica no sólo una mejora en la rendimiento del sistema sino también en su precisión, ya que logra filtrar gran cantidad de correspondencias espurias que podrían haberse dado de haber contrastado el descriptor del feature con otros que caen fuera del área predicha.

En las siguientes subsecciones se detallará de qué forma se lo busca dentro de esta elipse y cómo se realiza su correspondencia con los features encontrados.

4.5.2. Detección de features

La Búsqueda Activa implica la predicción de la ubicación de las mediciones existentes en el mapa dentro de la imagen según la localización predicha $x_{t|t-1}$ y su incertidumbre $P_{t|t-1}$, y la posterior detección de los features en la misma dentro de las elipses correspondientes. Para esto se utilizan métodos detectores de puntos de interés ampliamente estudiados. El más popular es probablemente el Detector de Bordes Harris [64]. Otros son FAST [65, 66] (*Features from Accelerated Segment Test*), SURF [67] (*Speeded-up robust features*), SIFT [68](*Scale-invariant Feature Transform*), STAR [69] (basado en CenSurE - *Center Surround Extremas*). En general, todos estos analizan la imagen dada y extraen puntos de interés de la misma. Cada uno de ellos tiene diferentes virtudes y defectos, como por ejemplo su tolerancia a las imágenes borrosas (*blur*) y su invarianza ante las transformaciones afines como la rotación o la escala, es decir, cuantos grados de rotación o escala soporta en promedio cada feature. Estas características definen la precisión y la estabilidad de cada detector y pueden ser comparados con respecto a dicha invarianza [54, 55]. Es importante notar que dentro de la elipse de predicción, durante la Búsqueda Activa, podrían detectarse múltiples features. De todos estos se debe seleccionar aquel que corresponde al que se está buscando, para lo cual es necesario utilizar métodos de correspondencia como se explica en la sección 4.5.3.

En este trabajo específicamente se emplea STAR como detector de features, aunque también es posible utilizar otros detectores (SURF, SIFT, FAST, etc) mediante la previa configuración del sistema, como se explica en la sección 5.2.

Hay casos en los que el método reconoce que necesita mayor cantidad de features para mejorar su estimación. Esto lo hace mediante una heurística simple durante la etapa de gestión del mapa que será explicada luego en la sección 4.5.6. En este proceso, el algoritmo utiliza las elipses de la predicción de las mediciones para realizar una búsqueda inteligente de nuevos features.

Algoritmo 8 DetectarNuevosFeatures(imagen, $\mathbf{h}_{t,i}$, $S_{t,i}$) : features nuevos

```
features nuevos ← ∅
candidatos ← detectar features fuera de las elipses definidas por el intervalo de confianza de 99 % de probabilidad de la N ∼ (h_{t,i}, S_{t,i})
Separar la imagen en K celdas rectangulares de igual tamaño
Mientras #features nuevos < N y #candidatos > 0 hacer
    celda candidato ← obtener la celda de menor cantidad de features (candidatos y existentes) y que no haya sido descartada
    Si existen features candidatos que están dentro de la celda candidato entonces
        Seleccionar un feature candidato al azar dentro de la celda candidato
        Descartar candidatos que caigan dentro de un radio R alrededor del candidato seleccionado
        Agregar feature candidato a features nuevos
    Si no
        Descartar celda candidato
    Fin si
Fin mientras
```

Las variables N, K y R corresponden a la cantidad de features nuevos deseados, la cantidad de celdas en las que se divide la imagen y el radio alrededor del candidato selec-

cionado que filtrará las siguientes búsquedas, respectivamente. La heurística mencionada tiene por objetivo lograr la distribución homogénea de los features en la imagen y garantizar una buena dispersión de los puntos del mapa con respecto al eje Z. Se pueden presentar casos en los que la mayor parte de los features se encuentran concentrados en un objeto que esta siendo visualizado por la cámara. Este es un caso particular que podría ser perjudicial para la estimación y por lo tanto se intenta minimizar su efecto y consecuentemente eludir la aglomeración de mediciones en el mapa. Por ende, enfrentar este problema es importante para la buena estimación de las velocidades angular y lineal. Adicionalmente, la distribución homogénea de los features en la imagen garantiza que el método, en aquellos casos en los que por un error no sistemático un objeto se mueve en la escena y afecta la estimación (pues se asume que el ambiente es estático), no será críticamente afectado como lo hubiese sido si la mayoría de los features se concentrasen en ese objeto.

El agregado de puntos iniciales al mapa también se realiza mediante este algoritmo, como se puede ver en el pseudocódigo 6, y resulta ser un caso particular (sin elipses) del recientemente explicado.

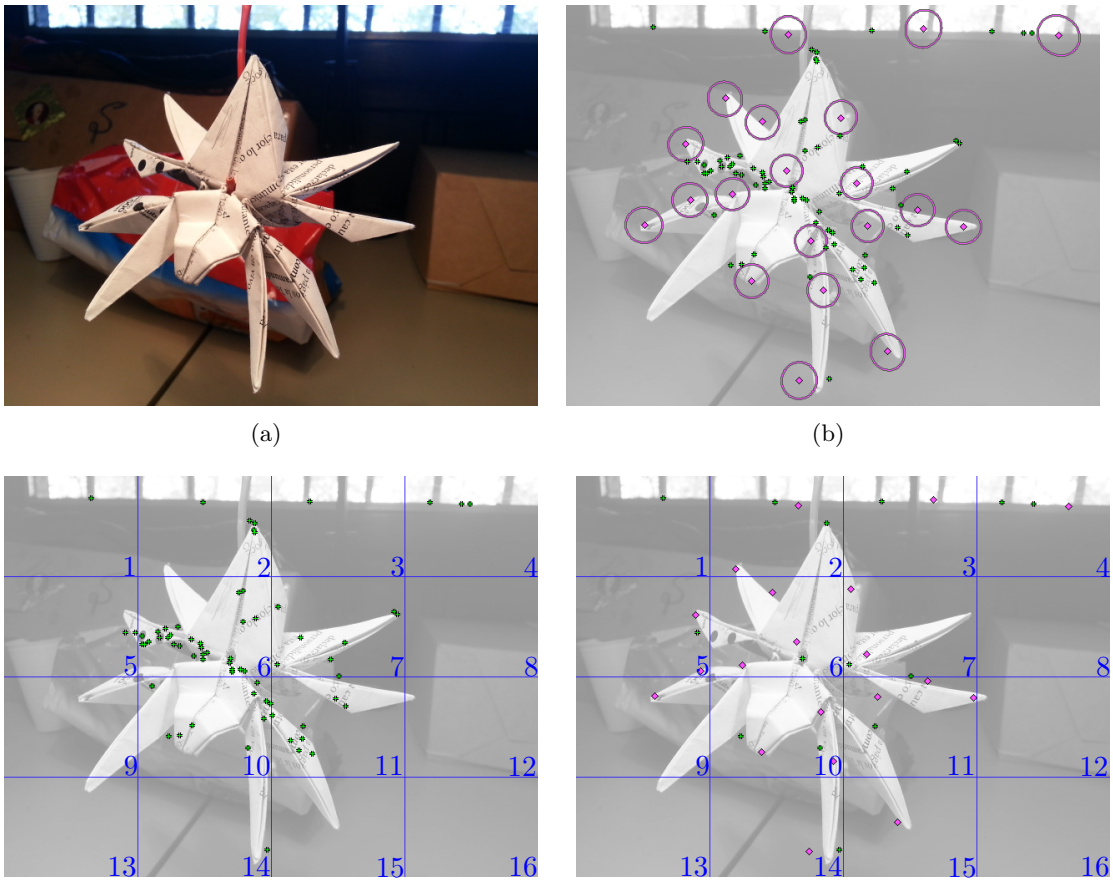


Fig. 4.2: Ejemplo de ejecución de heurística de detección de features: (a) imagen original, (b) detección de features fuera de las elipses, (c) separación de la imagen en celdas rectangulares y agrupación de features por celda, (d) selección inteligente de los features.

A modo de ejemplo, en la Figura 4.2 se puede apreciar el resultado de una ejecución de la heurística. Los valores $\mathbf{h}_{t,i}$ y $S_{t,i}$ se muestran en color rosa, y los features detectados con cruces de color verde. Es importante notar que la heurística logra balancear la cantidad de features resultantes en cada celda de la imagen siempre que sea posible (cuando haya una cantidad suficiente de features detectados). Cuando esto no ocurre, como en las celdas 1, 3 y 14, en las que solamente había un feature detectado fuera de las elipses, la cantidad de features en las mismas resulta menor que en las demás. En el caso particular de las celdas 4 y 5, la heurística agregó los features y luego filtró aquellos que estaban demasiado cercanos a otros ya seleccionados, por lo que no se pudieron extraer nuevas mediciones en esa zona. En las zonas 8, 9, 12, 13, 15 y 16 no hubo candidatos por lo que no pudo agregarse ninguno nuevo. En el resto de las celdas, la cantidad de features finales resultó ser homogénea.

4.5.3. Búsqueda de correspondencias

Cada feature tiene asociado un descriptor que permite ser comparado con otros y así establecer correspondencias que surgen de observar un mismo elemento en dos imágenes distintas. Un descriptor es un conjunto de datos que idealmente representa de forma unívoca el feature, y facilita cualquier tipo de análisis visual que se desee realizar. Este conjunto de datos pueden ser de tipo número flotante o de tipo entero, entre otros. Varios de los detectores como SURF y SIFT (que utilizan descriptores de valores de tipo flotante) cuentan con algoritmos extractores de descriptores de features. Por el contrario, otros detectores como FAST no cuentan con un extractor de descriptores asociado, pero se suelen utilizar en conjunto con otro extractor de descriptor. Por ejemplo, FAST podría utilizarse en conjunto con el extractor de descriptores de SURF o SIFT.

Los descriptores son ampliamente utilizados en el contexto de Visión por computadora para reconstrucción de ambientes, *trackeo* o seguimiento de objetos, y por supuesto, también para SLAM. Como método extractor de descriptores en esta tesis se emplea BRIEF (cuyos descriptores son de tipo binario y se almacenan en secuencias de 16, 32 ó 64 bytes) para el establecimiento de correspondencias de features. Junto a STAR resultan ser ideales para las tareas de Odometría Visual y SLAM Visual que se realizan en este trabajo [70].

Establecer correspondencias entre los features de la imagen y las predicciones correspondientes a los puntos del mapa requiere detectar los features en la imagen actual, cuyo resultado es un conjunto de features s expresados en coordenadas (x, y) en pixeles, y posteriormente realizar los siguientes pasos:

Algoritmo 9 BuscarCorrespondencias($imagen$, $\mathbf{z}_{t|t-1}$, s) : $\mathbf{z}_t$

1: *grupos de candidatos* ← Agrupar los features s según su correspondiente predicción $\mathbf{z}_{t|t-1}$
2: **Para todo** $grupo_i \in$ *grupos de candidatos* **hacer**
3: *descriptores* ← Extraer descriptores del conjunto de features $grupo_i$ de *imagen*
4: $descriptor_i$ ← Obtener descriptor del feature predicho asociado al $grupo_i$
5: Computar las distancias de $descriptor_i$ a cada descriptor en *descriptores*
6: Agregar a $\mathbf{z}_t$ el feature (y su correspondiente descriptor) de menor distancia a $descriptor_i$
7: **Fin para**

Es importante recordar que cada predicción $\mathbf{z}^i_{t|t-1} \in \mathbf{z}_{t|t-1}$ de un punto en el mapa

denota la región de búsqueda en la imagen del landmark $\mathbf{y}_{i,t}$ para el instante t descripta por la elipse de corte del intervalo de confianza de 99 % de probabilidad de la Gaussiana $N \sim (\mathbf{h}_{t,i}, S_{t,i})$, resultado de la Búsqueda Activa, como se explicó en la sección 4.5.1. Por lo tanto, para agrupar los features según su correspondiente predicción (línea 1 del pseudocódigo) se deben obtener los features detectados s_j que caigan dentro de la elipse de la predicción $\mathbf{z}^i_{t|t-1}$. Adicionalmente, la predicción $\mathbf{z}^i_{t|t-1}$ contiene el descriptor asociado al feature $\mathbf{y}_{i,t}$, y se aprovecha esto en la línea 4 del pseudocódigo. Por último la línea 5 computa la distancia entre descriptores para posteriormente elegir la mejor correspondencia y agregarla a $\mathbf{z}_t$ en la línea 6. La distancia entre dos descriptores $desc_i$ y $desc_j$ se calcula con la norma L^2 (norma euclidiana) en caso de que estos descriptores sean de valores flotantes, o utilizando la distancia de Hamming, para el caso de descriptores de valores binarios.

En la Figura 4.3 se ejemplifica el proceso del establecimiento de correspondencia con un caso particular similar al de la sección 4.5.2

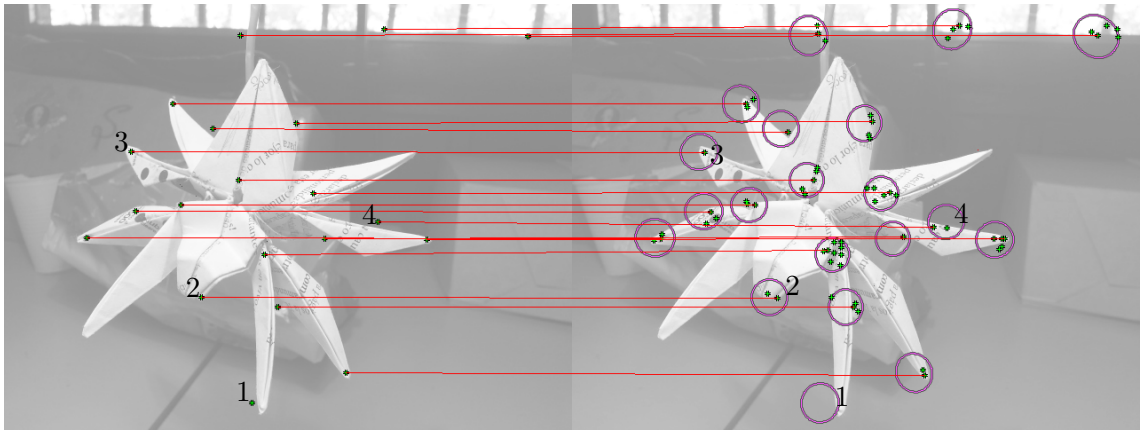


Fig. 4.3: Ejemplo de búsqueda de correspondencias. Las líneas rojas representan la correspondencia feature a feature. Las elipses de color violeta representan las predicciones de las mediciones, y los features dentro de las elipses son resultado del detector (sección 4.5.2). A la izquierda se muestra la imagen que se corresponde al instante $t-1$, y a la derecha se muestra la correspondiente al instante actual t, aunque en el sistema desarrollado en este trabajo no se realizan correspondencias de imagen a imagen sino del mapa a la imagen actual.

Se pueden distinguir distintas situaciones durante el establecimiento de correspondencias. Es posible que no se encuentre ninguna correspondencia debido a la falta de features en la zona de la elipse de predicción, como es el caso del feature 1 expuesto en la Figura 4.3. Otra situación consiste en encontrar múltiples candidatos en la elipse de predicción y posteriormente elegir aquel de menor distancia entre los descriptores de los features involucrados (como se explicó en el pseudocódigo 9), por ejemplo el feature 2 de la Figura, en donde se elige correctamente la correspondencia del feature. Otra situación puede darse si la elipse contiene un único feature detectado. En este caso se asume que la correspondencia es ese feature, como en el caso 3 de la Figura. Sin embargo, el conjunto de correspondencias en general puede contener falsos positivos, posiblemente consecuentes de haber encontrado un único candidato similar al caso 3 y que el mismo sea incorrecto. O encontrando múltiples candidatos y eligiendo uno tal que el de menor distancia entre descriptores no corresponde al feature buscado, como el caso 4. Por lo tanto se necesita

eliminar dichos falsos positivos para evitar perjudicar la estimación. Es por eso que se requiere una estrategia de filtrado como 1-Point RANSAC que se presentará en la siguiente sección.

4.5.4. Filtrado con 1-Point RANSAC

A partir de las correspondencias entre los puntos del mapa y la imagen se desea seleccionar aquellos que mejor estimen la relación entre los puntos almacenados en el mapa y los features percibidos en la imagen actual. Para esto será necesario detectar aquellas correspondencias espurias y eliminarlas. En este trabajo se utiliza el método RANSAC, más precisamente la variante 1-Point RANSAC. A continuación se presenta el algoritmo utilizado para determinar el subconjunto de correspondencias con mayor consenso, que luego será utilizado para actualizar el estado. Este algoritmo se corresponde con la línea 5 del pseudocódigo 7.

Algoritmo 10 filtrarOutliers($matches$, $\mathbf{x}_{t|t-1}$, $P_{t|t-1}$) : $Mejor\ Consenso$

1: $N \leftarrow \infty$
2: $Mejor\ Consenso \leftarrow \varnothing$
3: **Mientras** la cantidad de iteraciones realizadas son menores que N **hacer**
4: $match \leftarrow$ Seleccionar un match aleatoriamente del conjunto $matches$
5: $\mathbf{x}'_{t|t-1} \leftarrow$ EKFActualizarSoloEstado($match$, $\mathbf{x}'_{t|t-1}$, $P_{t|t-1}$)
6: Realizar una predicción de las mediciones a partir de $\mathbf{x}'_{t|t-1}$
7: $Consenso\ Actual \leftarrow$ seleccionar los matches tales que su predicción correspondiente a partir de $\mathbf{x}'_{t|t-1}$ se encuentra a una distancia menor que un umbral th
8: **Si** $Consenso\ Actual$ tiene mayor soporte que $Mejor\ Consenso$ **entonces**
9: $Mejor\ Consenso \leftarrow Consenso\ Actual$
10: Actualizar N adaptativamente
11: **Fin si**
12: **Fin mientras**

A diferencia del algoritmo RANSAC tradicional presentado en el algoritmo 4, para generar el conjunto de consenso se utiliza una única correspondencia dentro del conjunto de entrada *matches*. Para esto, se actualiza el estado (como en la sección 4.5.5 pero sin actualizar la covarianza) con dicha correspondencia. Los nuevos datos provenientes del estado $\mathbf{x}'_{t|t-1}$ sirven para realizar una predicción de las mediciones (línea 6 del algoritmo 10). Finalmente el conjunto de consenso se determina a partir de la utilización de la información de la predicción para encontrar las correspondencias que mejor se ajusten a dicha predicción. Tal como en el caso del algoritmo RANSAC original, el mejor conjunto de consenso se determina como el que tenga mayor soporte.

El umbral th es definido en base al ruido de sensado asociado al uso de la cámara, y que se obtiene mediante la calibración de la misma. Definiendo σ_{pixels} como el error en la medición de la cámara, el umbral utilizado para seleccionar aquellas correspondencias que forman parte del conjunto soporte es de $th = 2 * \sigma_{pixels}$, que se corresponde con el 99 % de que la selección corresponda con un *match inlier*.

4.5.5. Actualización del estado

A diferencia del algoritmo original de Kalman Filter, donde la fase de actualización se realiza una vez por cada paso, en la versión realizada con 1-Point RANSAC existen dos fases para actualizar de forma definitiva el estado y la covarianza, que se corresponden con las líneas 6, 7, 8, 9 y 10 del algoritmo 7.

Durante la primera fase se realiza una actualización completa del sistema (estado y covarianza) como se muestra en la formula (4.22), en base al mejor conjunto de consenso calculado mediante 1-Point RANSAC:

$$\mathbf{x}_t = \mathbf{x}_{t|t-1} + K_t(zInliers_t - h(\mathbf{x}_t^{\mathbf{zInliers}})) \tag{4.22}$$

$$P_t = (I - K_t H_t) P_{t|t-1} \tag{4.23}$$

$$K_t = P_{t|t-1} H_t^\top (H_t P_{t|t-1} H_t^t - R_t)^{-1} \tag{4.24}$$

donde $h(x_t^{zInliers})$ es la función que proyecta al plano de la imagen los features seleccionados como inliers, $H_t = (H_1, H_2, ..., H_n)$ es el vector de jacobianos de dicha función para cada uno de los n features seleccionados y R_t es la covarianza del ruido de la cámara.

Como siguiente paso, en la línea 8 del algoritmo 7, se realiza una predicción sobre las mediciones, pero teniendo en cuenta sólo aquellas correspondencias que resultaron ser outliers y el nuevo estado. Con estas predicciones se realiza el proceso denominado "rescate de outliers" [9] que tiene como objetivo incrementar la cantidad de correspondencias (usadas para actualizar el estado) que fueron consideradas como outliers antes de tener más información sobre el estado actual (que se obtiene a partir de las predicciones).

Una consecuencia favorable de aplicar este proceso puede ser la siguiente. Dado que los puntos lejanos contribuyen a realizar una buena estimación sobre la rotación, mientras que los cercanos ayudan a calcular mejor la traslación, en el caso en el que algunos de estos fueron inicialmente considerados outliers, "rescatar" los mismos utilizando la predicción de las mediciones permitiría mejorar la estimación considerablemente [9].

El proceso de rescate es posible debido a que, luego de la actualización realizada en base a las correspondencias inliers, la covarianza del estado se reduce y esto posibilita realizar una nueva predicción sobre las que resultaron ser outliers y así intentar rescatarlos. En este caso no es necesario buscar nuevamente un consenso para rescatarlos, ya que la predicción propaga la incertidumbre del sistema (que se vio reducida) y es posible utilizar esta información para definir condiciones más simples para rescatar las correspondencias [9].

En la línea 9 del algoritmo 7 se realiza el rescate propiamente dicho. Cada correspondencia es rescatada si se encuentra dentro de la elipse de 99 % de incertidumbre predicha para la misma. Luego del proceso de rescate se realiza una última actualización del estado como se indica en la línea 10 del algoritmo 7.

Si bien existe un paso de actualización adicional con respecto al algoritmo de Kalman Filter original, la complejidad del mismo no se ve afectada dado que las actualizaciones se realizan sobre las mediciones, cuyo número es significativamente mayor al número de landmarks en el mapa [9].

4.5.6. Gestión del mapa

Las gestión del mapa consiste en el mantenimiento de estadísticas sobre la información del estado, con el objetivo de mantener buenos puntos dentro del mapa, además de garan-

tizar una cantidad mínima de los mismos de forma que el método funcione correctamente, entre otras.

Algoritmo 11 Gestión del Mapa

1: Actualizar las estadísticas del mapa
{Eliminar landmarks con un porcentaje de bajo de matches}
2: **Para** cada landmark $\mathbf{y}_t$ del estado $\mathbf{x}_t$ **hacer**
3: **Si** $\frac{\#correspondencias(y_t)}{\#predicciones(y_t)} < U$ **entonces**
4: Eliminar $\mathbf{y}_t$ del estado $\mathbf{x}_t$ y P_t
5: **Fin si**
6: **Fin para**
{Determinar que hacer con los puntos que no se ven por la cámara}
7: Eliminar landmarks fuera del rango de visión
8: Convertir landmarks de parametrización Inverse Depth a parametrización XYZ en caso de ser posible.
9: **Si** #puntos inliers $< L$ **entonces**
10: $s \leftarrow$ DetectarNuevosFeatures(imagen, $\mathbf{h}_t$, S_t)
11: $\mathbf{x}_t, P_t \leftarrow$ AgregarFeatures(s, $\mathbf{x}_t$, P_t)
12: **Fin si**

En cada ciclo del método se calculan dos valores para cada uno de los puntos dentro del mapa. El primero lleva el conteo de la cantidad de veces que el punto fue proyectado en la imagen por medio del paso de predicción, y el segundo calcula el número de veces que fue correspondido en la etapa de búsqueda de correspondencias, es decir que fue inlier o rescatado.

Los valores anteriores sirven para establecer una estadística sobre cada punto del mapa y determinar si realmente ayudan o no a la estimación del método. Un landmark bueno dentro del sistema es aquel que por cada vez que se proyecta en la imagen se corresponde con un punto en la misma y además se incluye dentro de las etapas de actualización. Dicho punto tendría 100 % de efectividad en la relación

$$\frac{\#correspondencias(y_t)}{\#predicciones(y_t)}$$

y sería el caso ideal. En la práctica obtener una efectividad del 100 % para todos los puntos no es posible, por esta razón es necesario tener en cuenta todos los que tengan un porcentaje alto de aporte, que también se consideran buenos, ya que la mayoría de las veces ayudan a la estimación. El umbral U (correspondiente a la línea 3 del algoritmo 11) es configurable dependiendo del nivel de rigurosidad que se busque. Aquellos puntos que se encuentren por debajo de dicho umbral se eliminan del estado.

En esta implementación existe la posibilidad de definir una condición (en la línea 7) para eliminar aquellos puntos del mapa que se encuentren fuera del rango de visión de la cámara. Dado que uno de los criterios posibles consiste directamente en no eliminar nunca los puntos fuera del rango de visión de la cámara, es posible pasar de un caso de una implementación de Visual Odometry (donde no se almacena un mapa a largo plazo) a una de SLAM propiamente dicho. A continuación se muestran las condiciones que pueden formar dicho criterio:

- No definir condiciones: esto permite construir un mapa de todo el recorrido del robot, sin eliminar información del mismo (SLAM).
- Cantidad de puntos del mapa o tamaño del estado: si dicha cantidad de puntos o cantidad de variables del estado se supera, todos los puntos del mapa que quedan fuera del rango de visión se eliminan (Visual Odometry). Debido a que la parametrización XYZ implica el uso de tres variables (con respecto a las seis de la parametrización Inverse Depth) se pueden distinguir la cantidad de puntos en el mapa de la cantidad de variables del estado (que será menor para el caso de muchos puntos en parametrización XYZ).

El proceso de eliminación es utilizado en general para establecer una cota en el tiempo de procesamiento requerido para cada iteración del método. Esto resulta muy útil cuando se ejecuta el mismo dentro de dispositivos con limitado poder de procesamiento o cuando no resulta necesario realizar un mapeo completo del entorno.

El paso siguiente del algoritmo consiste en el pasaje de parametrización Inverse Depth a parametrización XYZ (ver sección 3.1.1). Para aquellos puntos representados con parametrización Inverse Depth de los cuales se posea mayor información sobre su profundidad, se irá reduciendo además su índice de linealidad (ver sección 3.1.3), hasta llegar a un nivel donde sea conveniente convertirlos a parametrización XYZ (ver 3.1.4). Este pasaje no sólo afecta al estado $\mathbf{x}_t$ del método sino también a su correspondiente matriz de covarianza P_t. Para convertir el punto del mapa $\mathbf{y}_{i,ID}$ a parametrización XYZ se debe utilizar la ecuación (3.3). Por otro lado, la matriz de covarianza se actualiza utilizando la matriz $J \in \mathbb{R}^{(n-3)\times n}$ correspondiente al Jacobiano de la ecuación (3.3):

$$\mathbf{x}' = \begin{pmatrix} \mathbf{x}_u & \mathbf{y}_1 & \mathbf{y}_2 & \cdots & \mathbf{y}_{i,E} & \cdots \end{pmatrix}^\top \tag{4.25}$$

$$\mathbf{y}_{i,E} = f_{ID\to E}(\mathbf{y}_{i,ID}) = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} + \frac{1}{\rho_i} m(\theta_i, \phi_i) \tag{4.26}$$

$$P' = JPJ^\top, \quad J = diag\left(I, \frac{\partial f_{ID\to E}}{\partial \mathbf{y}_{i,ID}}, I\right) \tag{4.27}$$

Debido a que la parametrización XYZ implica mantener tres variables menos con respecto a la parametrización Inverse Depth, con esta reducción se logra una considerable mejora en los tiempos de procesamiento del sistema.

Luego del proceso completo de eliminación (líneas 4 y 7), puede pasar que quede una cantidad baja de puntos en el mapa. Si no se mantiene una cantidad mínima de datos dentro del mapa, se corre el riesgo de que no sea posible encontrar correspondencias de forma de lograr una buena estimación en iteraciones siguientes. Por esta razón resulta conveniente agregar nuevos puntos al mapa en el caso en el que la cantidad de inliers (incluyendo rescatados) no sea suficiente. En las líneas 9, 10, 11 y 12 se describe el procedimiento para agregar nuevos puntos cuando sea necesario, mediante la función *DetectarNuevosFeatures* cuyos detalles se encuentran la sección 4.5.2 y luego agregando los puntos al mapa en parametrización Inverse Depth (ver sección 3.1.5).

5. IMPLEMENTACIÓN

La organización del sistema, sus principales módulos y funcionalidades asociadas, las variantes de implementación desarrolladas, así como decisiones de diseño e interfaz con el usuario son explicadas en detalle en este capítulo. En las siguientes secciones se describe cómo se agrupan las distintas funcionalidades y métodos presentados en los capítulos anteriores y cómo se relacionan entre sí para obtener el sistema completo desarrollado en esta tesis.

5.0.7. Descripción general de la implementación del método

El método implementado en esta tesis se puede detallar agrupando conceptos o funcionalidades que interactúan entre sí. Los tres módulos que conforman la implementación (ver Figura 5.1) son: *Extended Kalman Filter SLAM* (encargado de realizar las estimaciones a partir de la información de sensado), *Gestor de features*, (responsable de extraer información de cada imagen para cada iteración, analizarla y establecer su relación con el mapa), y el *Gestor del mapa*, (administra los puntos del mapa).

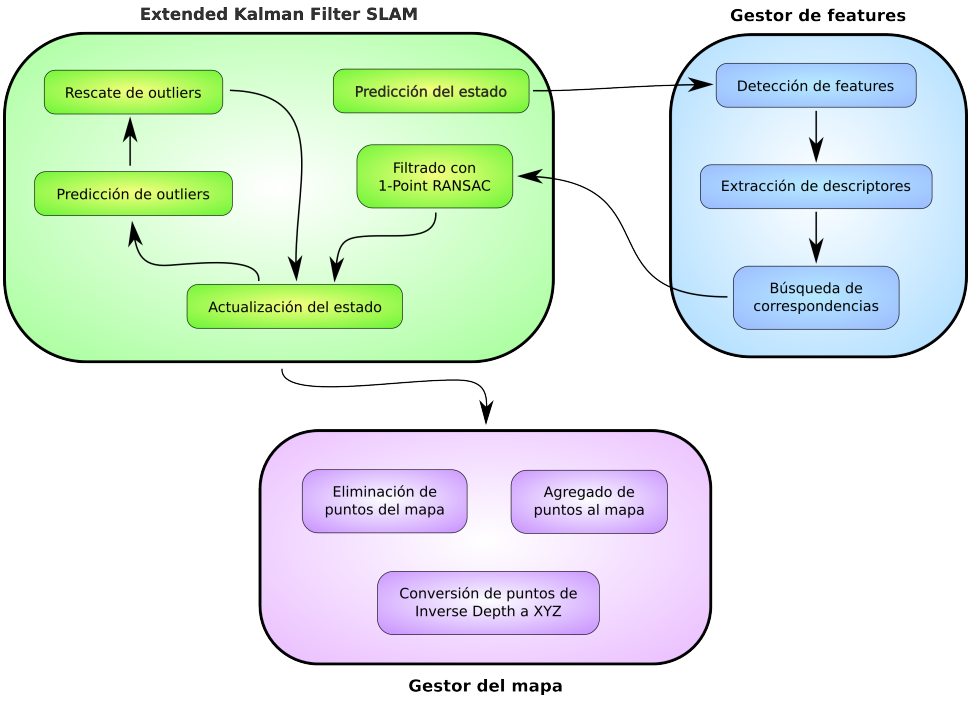


Fig. 5.1: Diagrama de intercomunicación entre las funcionalidades del método.

A continuación se detalla cada módulo sus distintas funcionalidades:

- Extended Kalman Filter SLAM:
  - Predicción del estado (ver sección 4.5.1): realiza una primera estimación

"a ciegas" (sin sensado) de la ubicación de la cámara y de la posición en la imagen de los puntos existentes en el mapa.

- Filtrado con 1-Point RANSAC (ver sección 4.5.4): elimina correspondencias espurias (outliers) provenientes de la heurística de búsqueda de correspondencias.
- Actualización del estado (ver sección 4.5.5): realiza un segundo paso de estimación utilizando los primeros datos provenientes del sensado y una predicción previa del estado.
- Predicción de outliers (ver sección 4.5.5): realiza una predicción más precisa sobre la posición en la imagen de los outliers, teniendo información adicional sobre el estado proveniente de la actualización del mismo.
- Rescate de outliers (ver sección 4.5.5): debido a que en el paso de filtrado con 1-Point RANSAC algunas correspondencias serán desechadas al considerarlas outliers, el rescate de outliers mejora la estimación reconsiderando las mismas teniendo nueva información a partir de la predicción de outliers.

- Gestor de features:

  - Detección de features (ver sección 4.5.2): procesa la imagen proveniente de alguna fuente (cámara, archivos de imágenes o video) utilizando la predicción del estado para detectar los features. En nuestro caso utilizamos el detector STAR [69] de OpenCV.
  - Extracción de descriptores (ver sección 4.5.3): obtiene los descriptores asociados a cada feature detectado en la imagen.
  - Búsqueda de correspondencias (ver sección 4.5.3): establece correspondencias entre los features detectados en una imagen y los puntos del mapa, utilizando sus descriptores asociados.

- Gestor del mapa:

  - Eliminación de puntos del mapa (ver sección 4.5.6): quita del mapa aquellos puntos que no aporten lo suficiente a la estimación y mantiene el tamaño del mapa en un número reducido de puntos.
  - Conversión de puntos de Inverse Depth a XYZ (ver sección 4.5.6): cambia la parametrización de los puntos del mapa cuando su parametrización XYZ supera un cierto límite de linealidad (ver sección 3.1.3). Debido al tamaño reducido de la parametrización XYZ con respecto a la parametrización Inverse Depth esta funcionalidad logra mejorar el rendimiento del método significativamente.
  - Agregado de puntos al mapa (ver sección 4.5.6): detecta y agrega nuevos puntos al mapa cuando el tamaño del mismo no es suficiente para establecer correspondencias en las siguientes iteraciones.

5.1. Características del sistema

Este sistema fue desarrollado en el lenguaje C/C++ y tiene una arquitectura modular, cuyas funciones principales fueron descriptas en la sección anterior. Como única dependencia externa posee la librería de visión por computadora OpenCV [56] que facilita

su portabilidad. La facilidad de configuración del sistema es una de las cualidades más salientes así como también la cantidad de información que puede ser consultada, tanto visualmente como numéricamente. Dichas cualidades tienen foco en el usuario, que le ayudan a configurar mejor el sistema y poder analizar los resultados más rápidamente. El sistema puede ser ejecutado en varias arquitecturas, incluyendo x86 (PC) y ARM (smartphones). Si bien cada variante de implementación (o versión) comparte los módulos principales, las mismas disponen de diferentes características como se muestran en las siguientes secciones.

5.1.1. Versión x86 (PC)

Esta variante de implementación (ver Figura 5.2) permite elegir el origen de las imágenes a procesar, dado que se puede configurar el sistema para que las obtenga de distintas fuentes. Los parámetros de calibración de la cámara, del modelo, del detector y del descriptor de features a utilizar, entre otros, son configurados mediante un archivo de tipo *XML* o *YAML* con el formato explicado en la sección 5.2.1.

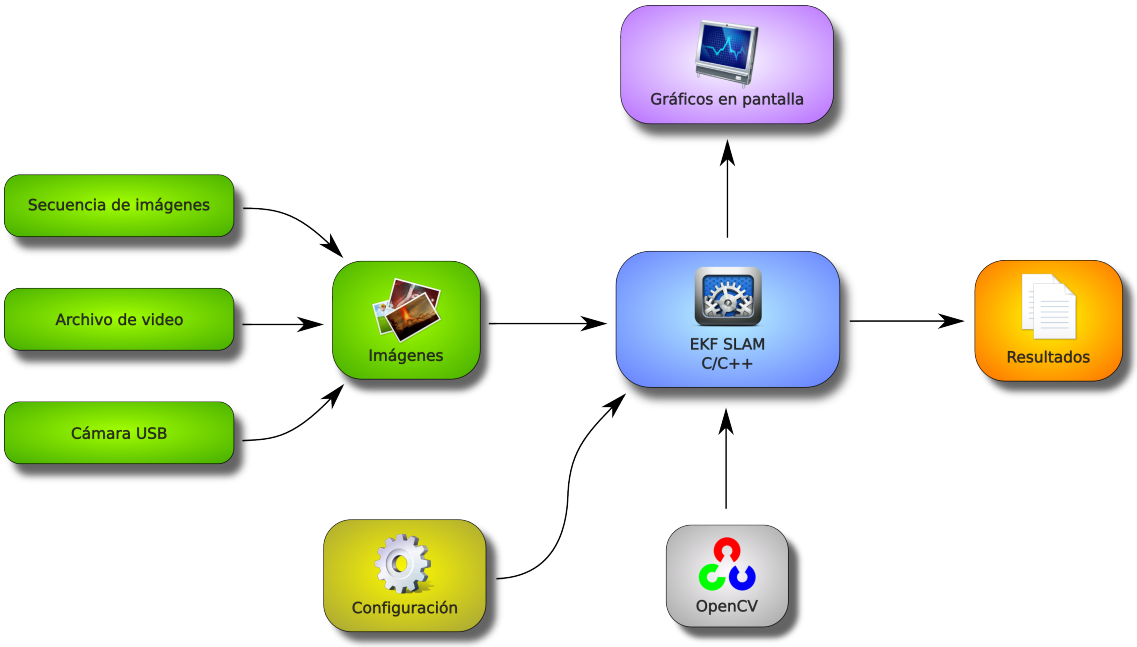


Fig. 5.2: Esquema de alto nivel de la aplicación versión PC.

Las distintas posibles fuentes de imágenes son las siguientes:

- Cámara de video: las imágenes se procesan bajo demanda.
- Un archivo de video: el mismo puede ser procesado cuadro a cuadro o simulando un *frame rate* (perdiendo algunos frames) menor al del *stream* de imágenes del video de entrada. Esta funcionalidad tiene como objetivo contemplar la diferencia existente entre procesar las imágenes en tiempo real y procesarlas offline.
- Una carpeta conteniendo una secuencia de imágenes: como en el caso anterior, las imágenes pueden ser procesadas con un frame rate simulado.

Una vez que el sistema se encuentra corriendo se da una respuesta al usuario mostrando información de cada paso en una o más ventanas. Por consola se muestran todos los datos relativos a la cámara (su posición, orientación, velocidad lineal y velocidad angular), como también la posición de cada uno de los puntos en el mapa (en la representación que se encuentren, ya sea XYZ o Inverse Depth). Un ejemplo de la información brindada por consola se puede ver en la Figura 5.3.

```
~~~~~~~~~~~~ STEP 162 ~~~~~~~~~~~~

cantidad de matches: 36
cantidad de inliers: 36
cantidad de ouliers: 0
Cantidad de matches rescatados: 0
Se eliminaron del mapa los features que no se ven en el frame actual.
Se convirtió a DEPTH el feature de índice 10
Cantidad de features detectados al agregar nuevos: 0
Se agregaron 0 nuevos features al mapa.
Posicion de la camara: 0.796067, 0.00470655, 0.0119048
Orientacion(cuaternions): 0.999835, 0.00186379, -0.0180743, 3.54535e-05
Orientacion en angulos eulerianos: 0.00372812, -0.0361506, 3.52436e-06
Velocidad lineal (con respecto al mundo): 0.00489127, 0.000117717, 0.000189266
Velocidad angular (con respecto a la camara): 9.28429e-05, -0.000215946, 7.56582e-06
Cantidad de features en el mapa: 36
```

(a)

```
Map Features (36):
0: 0.335011, -0.407694, 1.22456 (163/163)
1: 0.510133, -0.407702, 1.22811 (163/163)
2: 0.159926, -0.269457, 1.21996 (163/163)
3: 0.335107, -0.269357, 1.2209 (163/163)
4: 0.335184, -0.130708, 1.21805 (163/163)
5: 0.685891, -0.130884, 1.2302 (163/163)
6: 0.159891, 0.0081085, 1.21818 (163/163)
7: 0.335224, 0.00810073, 1.21722 (163/163)
8: 0.510148, 0.00810305, 1.22221 (163/163)
9: 0.685999, 0.00810439, 1.23018 (163/163)
10: 0.159902, 0.147078, 1.21927 (163/163)
11: 0, 0, 0, 0.130215, -0.227899, 0.790882 (163/163)
12: 0, 0, 0, 0.267654, -0.221612, 0.769758 (163/163)
13: 0, 0, 0, 0.394263, -0.211808, 0.736087 (163/163)
14: 0, 0, 0, 0.50828, -0.199835, 0.695784 (163/163)
```

(b)

Fig. 5.3: Ejemplo de información mostrada por consola: en (a) se observa la cantidad de matches inliers, outliers y rescatados, los datos asociados a la cámara como la posición, la orientación y velocidades más información de la cantidad de landmarks que fueron agregados, eliminados y cuáles de los que se encuentran en el mapa en parametrización Inverse Depth fueron convertidos a representación XYZ. En (b) se observan las coordenadas de todos los puntos del mapa, donde existen algunos con tres coordenadas y otros con seis junto a la cantidad a la cantidad de veces que sirvieron a la estimación con respecto a la cantidad de veces que fueron predichos en la imagen, entre paréntesis.

Es posible configurar el sistema para que se muestren las imágenes con las elipses de predicción de las mediciones. La secuencia de imágenes permite analizar el progreso del sistema a medida que avanza y da una idea sobre la calidad de la estimación. Por ejemplo, observando un tamaño de elipses muy grande, se puede inferir que el sistema no esta logrando una buena estimación. En la Figura 5.4 se observa la ventana con la información mencionada, donde las elipses se presentan con dos colores: en verde se identifican aquellos puntos del mapa en parametrización XYZ y en rojo, los que todavía se encuentran

descriptos en parametrización Inverse Depth.

Las correspondencias con las que realizaron los pasos de predicción se pueden ver en la Figura 5.5 y tiene como objetivo observar rápidamente la cantidad de puntos con las cuales se actualizó el sistema, una baja cantidad de puntos podría dar un indicio sobre una mala configuración del detector de features, descriptor, del método de matching o del algoritmo de RANSAC.

Para verificar el correcto funcionamiento del sistema se implementó una funcionalidad que permite visualizar una ventana que muestra para cada una de las iteraciones los resultados de 1-Point RANSAC, resaltando el punto que se toma para calcular el conjunto soporte y los que resultan ser inliers y outliers para ese conjunto. Agregado a lo anterior se muestran las distancias entre los puntos correspondientes a los matches y las predicciones realizadas a partir del feature seleccionado. Todo en conjunto permite verificar el buen funcionamiento de RANSAC si es necesario.

La fase de rescate puede ser verificada observando los puntos marcados como outliers y cuáles de estos fueron rescatados.

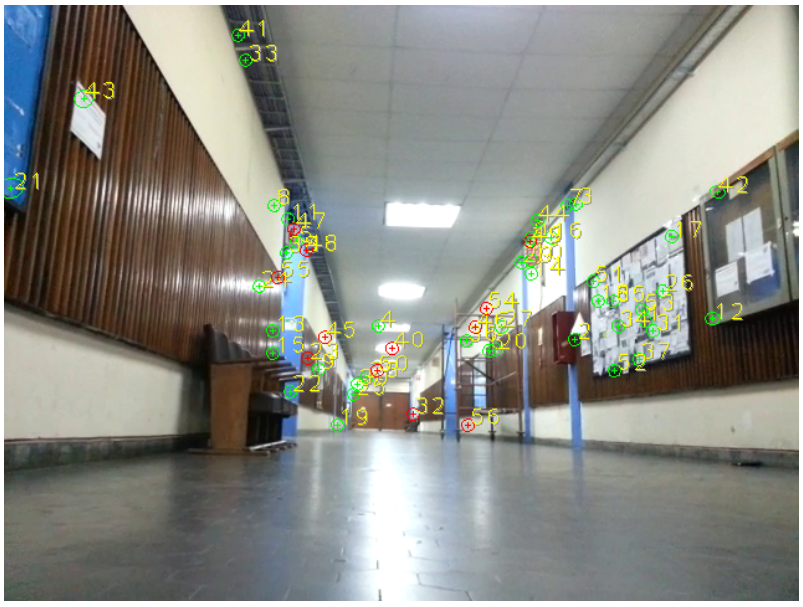


Fig. 5.4: Predicciones y las elipses de incertidumbre. Las elipses de color verde se corresponden con puntos del mapa representados con parametrización XYZ, las elipses de color rojo, con puntos representados con parametrización Inverse Depth.

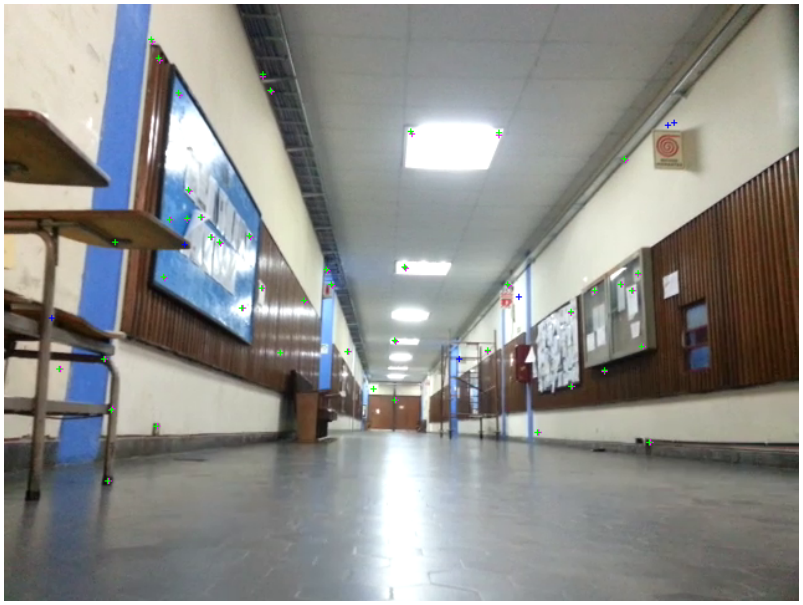
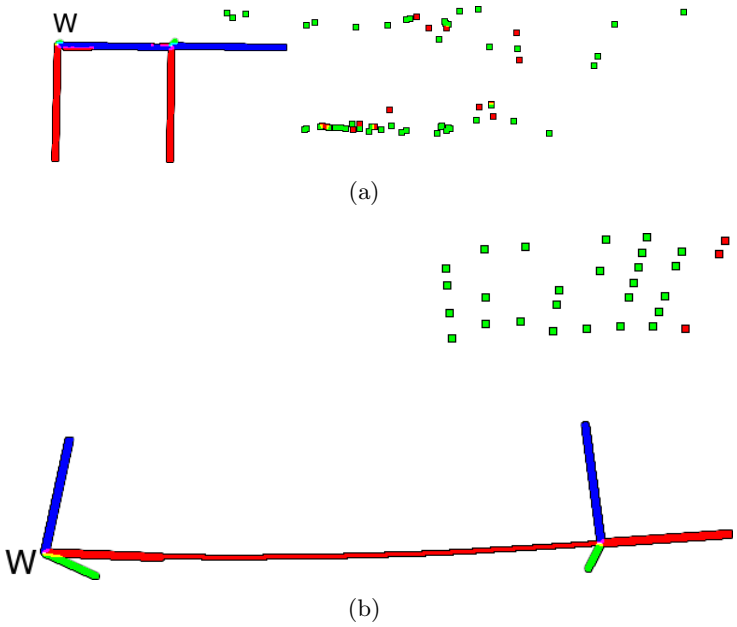


Fig. 5.5: Conjunto final de todos los puntos que fueron utilizados para las actualizaciones del estado para el paso actual. Se muestran aquellos puntos inliers seleccionados por RANSAC y sumados los outliers que fueron rescatados.



(a)

(b)

Fig. 5.6: Ejemplo del recorrido que realiza el robot dentro de un pasillo (Figura (a)). Se puede observar la posición de inicio marcada por el eje de coordenadas inferior, la posición estimada actual en el eje de coordenadas superior y la posición de los puntos del mapa. La Figura (b) muestra un recorrido paralelo a una pared donde se aprecia mejor la trayectoria del robot.

El sistema también permite visualizar la estimación de la ubicación de la cámara y

de los puntos en el mapa. Por último, una ventana que permite ver globalmente como resulta la estimación del sistema, la librería Point Cloud Library [71]. En la Figura 5.6 se muestra el ejemplo de un recorrido de la cámara sobre un pasillo visto desde arriba, donde se aprecia que los puntos del entorno corresponden a puntos sobre las paredes del mismo y que forman un plano.

5.1.2. Aplicación Android

Además de la variante de implementación para PC se desarrollo una variante para el sistema operativo Android para smartphones (ver Figura 5.7). La arquitectura de esta variante consiste en tres capas. La primera capa contiene el sistema de localización desarrollado en C/C++, que corresponde al núcleo de la variante para PC. La tercera capa (la de más alto nivel) corresponde a la de la aplicación Android, escrita en lenguaje *Java* mediante la librería Android SDK [72]. En esta capa se capturan las imágenes desde la cámara del dispositivo y se brinda una interfaz visual. Entre la primera y tercera capa se encuentra otra que cumple el rol de interfaz desarrollado mediante el framework *Java Native Interface* (*JNI*) [73]. Java Native Interface (JNI) es un framework de programación que permite que un programa escrito en Java pueda interactuar con programas escritos en C/C++. La aplicación para Android fue generada mediante la herramienta de desarrollo Android NDK [74].

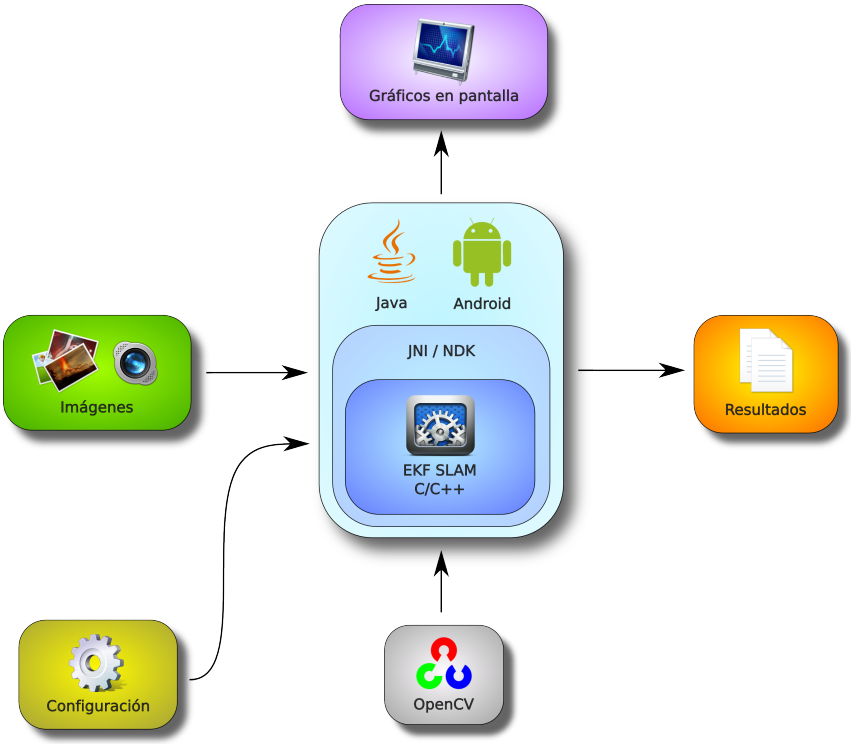
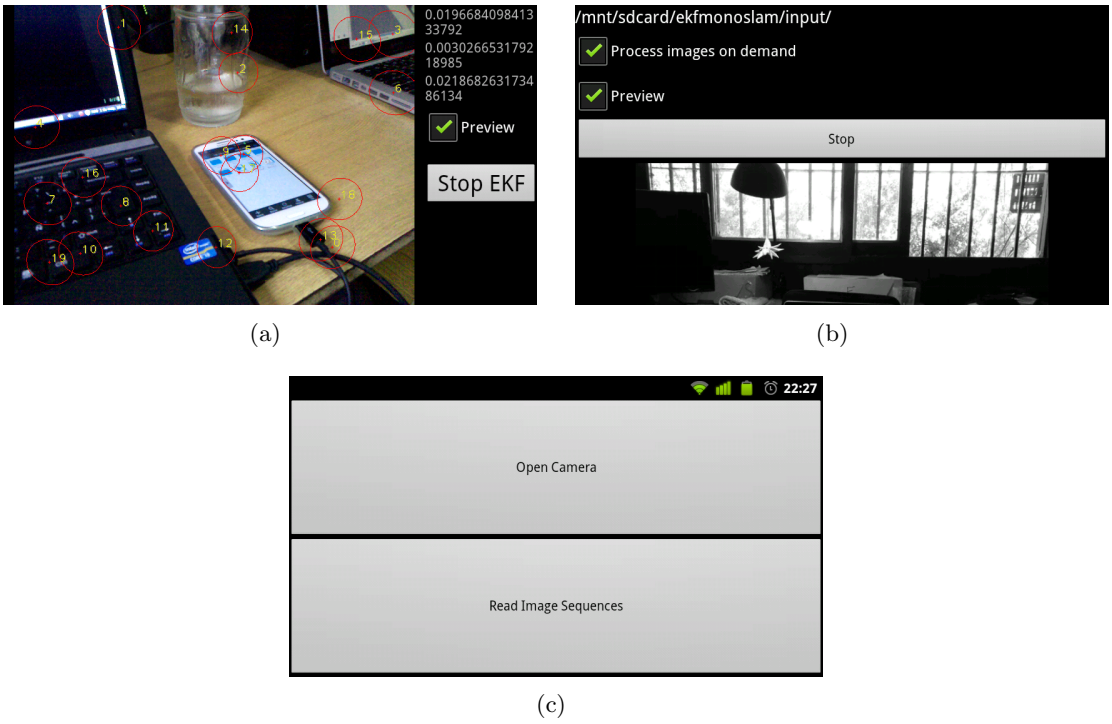


Fig. 5.7: Esquema de alto nivel del sistema para Android. Las imágenes se obtienen utilizando el framework Android SDK y se pasan al núcleo desarrollado en C/C++ utilizando las interfaces JNI y NDK.

Interfaz gráfica

Como se mencionó previamente, el objetivo de este trabajo es desarrollar un método por el cual se haga uso del smartphone como unidad de sensado y procesamiento en un robot móvil. En particular la cámara digital del dispositivo corresponde al único sensor utilizado. Por otro lado, para evaluar el correcto funcionamiento del método, es útil permitir ejecutar el sistema utilizando un video o una secuencia de imágenes en forma offline.

Teniendo en cuenta estas consideraciones, la variante de implementación presenta un menú principal con dos opciones que permiten ejecutar el sistema de localización como se observa en la Figura 5.8. La primer opción permite obtener imágenes de la cámara del dispositivo, y previsualizar de forma ininterrumpida una vista previa. Por otro lado, la segunda opción en el menú permite procesar secuencias de imágenes o un video grabado anteriormente.



(a) (b)

(c)

Fig. 5.8: Aplicación Android: en (a) se muestra como se procesan de forma *online* los cuadros provenientes desde la cámara del dispositivo, donde se muestra además como se predicen los puntos del mapa en la imagen actual. En (b) la versión *offline* donde se procesa secuencias de imágenes generadas a partir de un video. En (c) se muestra el menú principal de la aplicación, donde se pueden seleccionar lo modos anteriormente descriptos.

Tras presionar el botón “Start EKF” de la vista se comienza a procesar el cuadro actual inicializando el sistema como se explica en la sección 4.5. Como respuesta para el usuario, luego de procesar una imagen se actualizará la vista del dispositivo con las coordenadas estimadas del robot en el sistema XYZ (en la parte superior del botón) y se mostrará la imagen procesada (opcional) junto con la información del paso de predicción, que consiste en posición en la imagen estimada para cada feature correspondiente a un landmark y su elipse de incertidumbre.

Dado que el sistema podría no terminar de procesar un frame antes de que la cámara capture el siguiente, es posible que se pierdan frames, lo que se encuentra asociado a la complejidad del algoritmo y a las características del dispositivo (poder de procesamiento, velocidad de la memoria ram, etc.). Por esta razón para el caso del procesamiento de una secuencia de imágenes o un video y con el objetivo de simular dicha perdida de cuadros, existe una opción con la leyenda "Process images on demand". Si no se selecciona dicha opción, el sistema realizará su estimación en base a todos los cuadros disponibles, sin saltear ninguno.

5.1.3. Datos generados por el sistema

La información generada por el sistema se guarda en diferentes archivos que permiten luego analizar los resultados y que también permiten reproducir exactamente la misma ejecución con el objetivo de modificar optimizar parámetros y evaluar el correcto funcionamiento del sistema. Los archivos a crear son los siguientes:

- Un archivo en formato YAML conteniendo, para cada iteración del sistema de localización, el tiempo requerido por las secciones más importantes del algoritmo, todos los datos asociados al cámara y su matriz de covarianza.
- Un archivo de texto plano con toda la información que se muestra por consola.
- La secuencia de imágenes procesada con su información de predicción.
- Un video con todas las imágenes procesadas junto con su información de predicción (sólo para la versión PC).

```
%YAML:1.0
Frame 1:
   #
   # Running time (microseconds)
   #
   Prediction: 1815.
   Matching: 24038.
   Ransac: 418.
   UpdateLI: 12755.
   RescueOutliers: 11.
   UpdateHI: 1.
   MapManagement: 24340.
   #
   # State and Covariance Estimation
   #
   StateEstimation: !!opencv-matrix
      rows: 1
      cols: 13
      dt: d
      data: [ 3.8424652212252974e-03, 5.6675633463358803e-06,
          1.5496472694982342e-04, 9.9999999980238785e-01,
          3.1191640901960278e-07, 1.9862802593658001e-05,
          7.7184797501602939e-07, 3.8424652212253061e-03,
          5.6675633463378319e-06, 1.5496472694982429e-04,
          6.2383281816247066e-07, 3.9725605195166330e-05,
          1.5436959503371253e-06 ]
   MapFeaturesInvDepthCount: 30
   MapFeaturesDepthCount: 0
   StateCovarianceMatrixEstimation: !!opencv-matrix
      rows: 13
      cols: 13
      dt: d
      data: [ 5.9707220408609268e-07, 8.8159300863010075e-09,
          2.0712255124494457e-07, 2.7918385989682420e-12,
          3.8381051538137809e-10, -1.4073047328848069e-07,
```

Fig. 5.9: Ejemplo de una porción del archivo de tipo YAML con los resultados.

El archivo de salida con formato YAML resulta útil para hacer un seguimiento de la trayectoria de la cámara durante todo el proceso, dado que el mismo contiene la información de su ubicación a lo largo de las iteraciones. De ser necesario se puede disponer de los datos de la matriz de covarianza de dicho vector, lo que permite analizar la exactitud con la que el sistema estima cada una de las variables de posición, de velocidad y orientación.

Asimismo, se encuentra disponible la cantidad de puntos en el mapa en cada una de sus representaciones para cada paso del sistema. Los tiempos de ejecución de cada paso intermedio del sistema también son registrados, ya que son datos que resultan útiles para realizar análisis de rendimiento (predicción de features en la imagen, el método de matching, el método RANSAC, la actualización del sistema en base a los inliers calculados por RANSAC, el rescate de outliers, la actualización del sistema utilizando la información rescatada y la gestión del mapa). En la Figura 5.9 se muestra un ejemplo de como se vería el archivo de salida con parte de su información.

5.2. Configuración del sistema

En el caso actual del sistema desarrollado en esta tesis, la capacidad de poder modificar cada uno de los parámetros de configuración del método resulta primordial, dados los cambios en el entorno, la utilización de diferentes robots y el tipo smartphone que realiza el procesamiento, entre otros. La variabilidad a la que se puede someter el sistema requiere de una herramienta que permita realizar estos cambios de forma simple y sin necesidad de recompilar todo el código.

5.2.1. Archivo de configuración

Los parámetros que definen el comportamiento del sistema se encuentran dentro de un archivo con formato YML. El mismo tiene un diseño que permite fijar perfiles con distintos tipos de configuraciones, característica que evita tener muchos archivos distintos, y además facilita el acceso a las diferentes configuraciones en una única vista simplificando su búsqueda, modificación y comparación.

En el sistema hay cuatro módulos configurables. Cada uno de estos tiene asociado una serie de parámetros que pueden ser definidos. Además, en algunos casos, debido a las variantes de implementación de cada módulo, es posible elegir distintos perfiles de configuración. Cada uno de estos perfiles tiene asociado su propio conjunto de parámetros. A continuación se describe brevemente cada uno de estos módulos:

- *EKF SLAM*: para este módulo es posible configurar los distintos parámetros para la ejecución del sistema.
- *Detector de features*: eligiendo un perfil determinado, es posible elegir entre las distintas variantes de implementación de este módulo (por ejemplo: SURF, FAST, STAR, etc.).
- *Extractor de descriptores*: existen distintos perfiles asociados a cada una de las implementaciones de extractores de descriptores (por ejemplo: SURF, SIFT, BRIEF, etc.) que pueden ser elegidos independientemente del detector de features utilizado.
- *Calibración de cámara*: dado que una cámara tiene sus propios parámetros de calibración, se pueden utilizar los distintos perfiles según la cámara que se esté utilizando (por ejemplo: cámara del smartphones, *webcam*, etc.).

La configuración del módulo *EKF SLAM* incluye parámetros que definen los ruidos iniciales del sistema, los ruidos luego de la primera iteración y el comportamiento de las funciones que realizan la gestión del mapa, los parámetros utilizados por RANSAC entre otros. El listado de parámetros correspondientes se puede ver en el apéndice A.1.1.

La función de estos parámetros dentro del módulo *EKF SLAM* es la de permitir la búsqueda de la configuración óptima del sistema de acuerdo a las condiciones de movimiento y del entorno así también como la de realizar restricciones en algunos aspectos del algoritmo con el objetivo de mejorar el tiempo de respuesta del sistema dependiendo del poder de procesamiento de la plataforma donde se este ejecutando.

El módulo *Calibración de cámara* tiene como objetivo cargar la calibración de las cámaras de los distintos dispositivos smartphones o cámaras externas que puedan ser utilizadas para ejecutar el sistema, tales como: los parámetros intrínsecos, el tamaño de las imágenes a procesar y el ángulo de visión, entre otros. El listado correspondiente se encuentra en el apéndice A.1.2.

6. EXPERIMENTOS

En este capitulo se muestran los experimentos realizados con el método de localización desarrollado en esta tesis. Se realizaron diferentes pruebas con el objetivo de evaluar el comportamiento del sistema ante diferentes escenarios. Cada uno de estos experimentos conlleva el ajuste de los valores en los parámetros de configuración, teniendo en cuenta los movimientos, recorridos y características físicas del entorno.

6.1. Materiales y Métodos

Para realizar los experimentos se utilizaron: un smartphone con cámara de 8 mega pixeles, un sistema de localización global permite analizar la trayectoria real del robot y contrastarla con la estimación, un robot móvil (Exabot) portador del smartphone, y una computadora, para correr los experimentos offline y contrastar los resultados con los obtenidos en el smartphone. A continuación se detallará cada uno de los dispositivos involucrados.

6.1.1. El robot ExaBot

El robot utilizado es el ExaBot [75], un robot móvil multipropósito que puede ser usado para una gran variedad de tareas. El mismo soporta la configuración de varios sensores y unidades de procesamiento deferentes. Su sistema de tracción está basado en orugas, lo que le permite transitar tanto en superficies de interior como de exterior y superar pequeños obstáculos. Por otro lado, presenta una restricción en cuanto a su desplazamiento, pues concretamente, no puede desplazarse hacia los costados. Es por esto que el Exabot es un robot no holonómico, pues cuenta con 2 grados de libertad (rotación y traslación hacia adelante y atrás) en el plano en el que se mueve. Sin embargo, el modelo dinámico del método de localización propuesto soporta movimientos en los 6 grados de libertad (ver capitulo 4) que representa una mejora, ya que puede lidiar con vibraciones en la cámara y terrenos que tengan pendientes o depresiones.

El Exabot dispone de comunicación WIFI. Se aprovechó esta característica para realizar una aplicación Android que facilitó el control remoto del robot mediante una interfaz simple y que resultó ser de gran utilidad en los experimentos realizados.

Se instaló un soporte en el ExaBot que permite colocar el dispositivo smartphone como se muestra en la Figura 6.1. Este dispositivo es el encargado de capturar las imágenes y ejecutar el sistema de localización.

Fig. 6.1: Robot Exabot realizado en la Facultad de Ciencias Exactas y Naturales (FCEyN).

6.1.2. Smartphone y PC

La plataforma utilizada por defecto para realizar los experimentos del sistema es un smartphone. Como segunda plataforma se utilizó una laptop con el objetivo de tener un punto de comparación a la hora de realizar un análisis de los resultados obtenidos, como los tiempos de procesamiento y calidad de las estimaciones, considerando el poder de cómputo que cada uno posee.

El smartphone utilizado es un Samsung Galaxy S3 (Figura 6.2), que cuenta con un procesador de arquitectura ARM, modelo Cortex-A9, una frecuencia de reloj de 1.4 GHz y cuatro núcleos. El sistema operativo instalado es un Android versión 4.1.2 (*Jelly Bean*). La cámara utilizada para realizar todos los experimentos es la correspondiente a este dispositivo móvil, que cuenta con 8 MegaPixeles y auto foco. Si bien dicho sensor soporta una resolución alta de vídeo (FullHD 1920×1080 pixels), todos los videos utilizados para los experimentos fueron grabados con una resolución de 640×480 pixels. El tamaño de imágenes se escogió en esos valores para obtener un balance entre calidad y rendimiento, ya que una resolución menor impactaría en la cantidad y calidad de los features detectados, lo que se vería reflejado en los resultados del sistema. Si en cambio, las imágenes fueran de resoluciones muy altas, el poder de cómputo requerido para procesarlas también sería alto. Considerando las limitaciones del ambiente de ejecución, esto podría afectar la cantidad de cuadros por segundo que se logran procesar, afectando la estimación de la localización.

La PC utilizada para procesar los experimentos es una laptop, con un procesador Intel(R) Core(TM) i3-2310M de una frecuencia de 2.10GHz y una memoria de 3 GB DDR3.

El sistema de localización desarrollado en esta tesis, requiere conocer los parámetros

Parámetro	Valor
PixelsX	640
PixelsY	480
FX	525,06
FY	524,24
K1	$-7{,}61e^{-3}$
K2	$9{,}38e^{-4}$
CX	308,64
CY	236,53
DX	0,007021
DY	0,007027
PixelErrorX	0,25
PixelErrorY	0,25
AngularVisionX	62,72
AngularVisionY	49,16

Tab. 6.1: Parámetros de calibración intrínsecos obtenidos para la cámara utilizada.

intrinsecos de la cámara utilizada (ver sección A.1.2). Se utilizaron dos herramientas para obtener estos parámetros intrínsecos: *Camera Calibration Toolbox for Matlab* [76], de código abierto disponible para *Matlab* y *Photomodeler* [77], un software propietario. Los valores asociados a los parámetros mencionados se encuentran en la Tabla 6.1.

Fig. 6.2: Samsung Galaxy S3 utilizado para ejecutar los experimentos.

6.1.3. Sistema de localización global

Los resultados obtenidos por medio del sistema de localización realizado se contrastaron con un sistema de localización global, cuyo error se encuentra en el orden de los pocos milímetros. Es por esta razón que las posiciones obtenidas con este sistema son

considerados como valores reales de posición (*ground truth*) para comparar la estimación del sistema realizado en esta tesis.

El sistema de localización global necesita que cada experimento realizado sea grabado por una cámara estacionaria externa, que registra toda el área donde se realiza el experimento. Se deben conocer los parámetros de calibración de dicha cámara para que pueda determinar las posiciones con el mínimo error posible. La cámara externa debe visualizar un patrón de forma circular de dimensiones conocidas. El eje de coordenadas del plano en el que se mueve este patrón es fijado utilizando tres patrones circulares dispuestos de tal forma que describan los ejes del plano XY (ver Figura 6.3). Una vez fijados estos ejes, se los remueve en las siguientes imágenes, para dejar un único patrón que se moverá en todo el área visible. El sistema de localización global realiza una búsqueda de dicho patrón para determinar su posición con respecto a los ejes definidos.

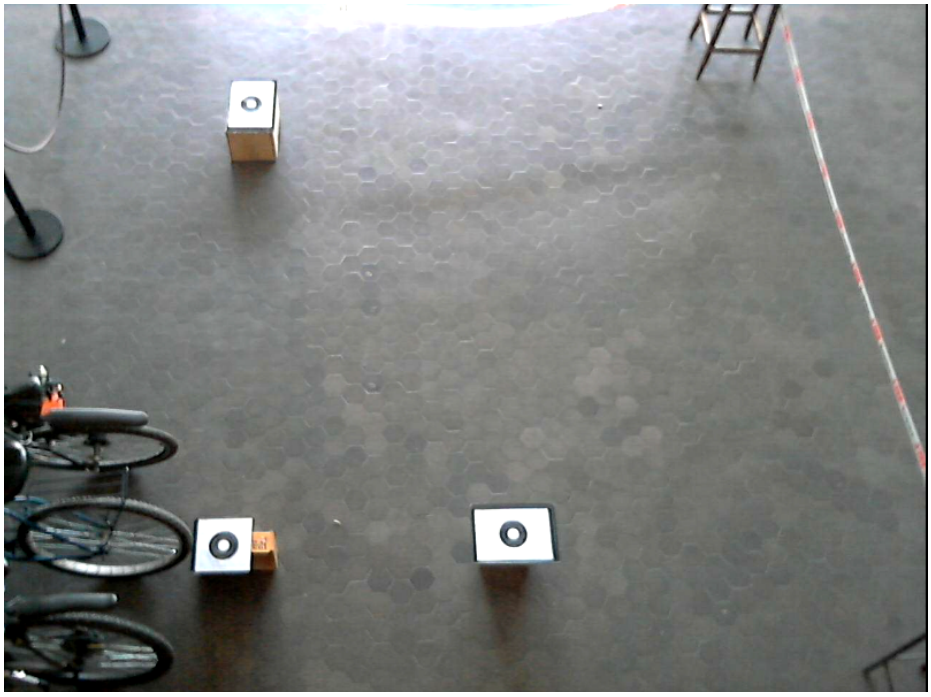


Fig. 6.3: Patrones dispuestos en forma de eje cartesiano. En base a este plano quedan definidas las posiciones del robot.

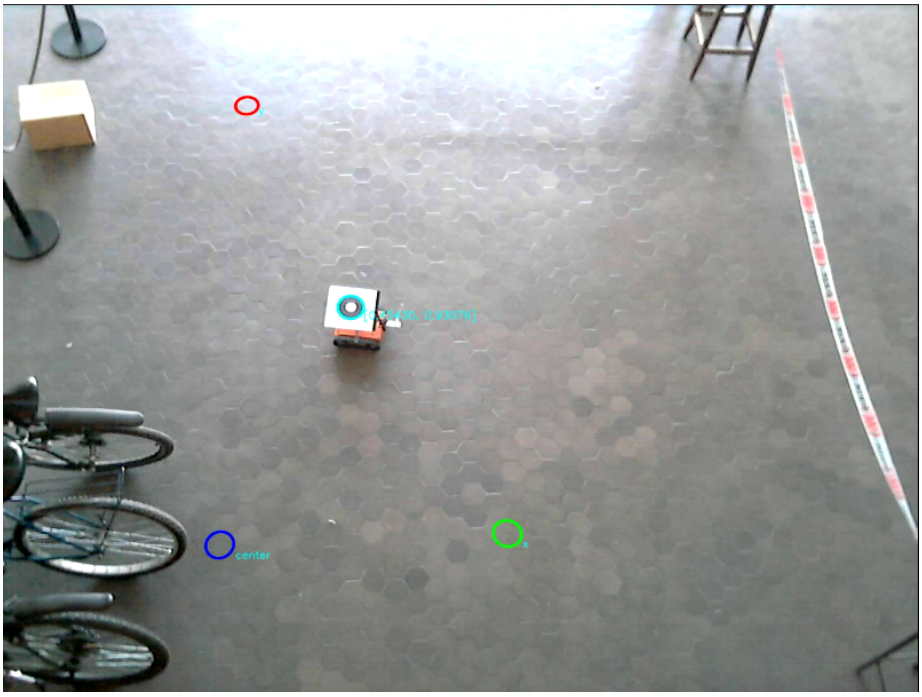


Fig. 6.4: Seguimiento del robot luego de definir el eje de coordendas. Se debe fijar el patrón circular al robot y el mismo debe ser visto por la cámara estacionaria durante todo el experimento.

Una vez definido el plano de coordenadas, se puede utilizar este sistema para un patrón fijado sobre el robot, y de este modo determinar su posición con alta precisión. En la Figura 6.4 se muestra un ejemplo extraído de los experimentos realizados, donde se puede observar el eje de coordenadas establecido y el seguimiento del patrón que determina su posición relativa.

6.1.4. Configuración del sistema

Para realizar los experimentos se evaluaron diferentes trayectorias con el robot ExaBot dentro de un ambiente interior de un tamaño aproximado de 2,5 m^2. En la Figura 6.5 se puede observar el plano de coordenadas definido para el sistema de localización global.

Cada una de las trayectorias realizadas fueron procesadas en la PC y en el smartphone y los desplazamientos se realizaron a velocidad constante para cumplir con los requerimientos del modelo dinámico. Las ejecuciones realizadas dentro de la PC tienen como característica, el procesamiento de todos los cuadros del video, mientras que el smartphone procesa los frames bajo demanda, es decir que dentro del flujo de imágenes, el dispositivo móvil procesa sólo los que puede dependiendo de su poder de cómputo, mientras los otros se pierden.

Las estimaciones realizadas por la PC tienen dos variantes, las cuales se diferencian en la cantidad mínima (20 ó 60) de correspondencias dentro de cada imagen. Los experimentos realizados en el smartphone usan todos 20 correspondencias como mínimo para que la cantidad de cuadros por segundo que se puedan procesar no sea muy baja.

Se probaron dos orientaciones distintas para el smartphone: alineado con el frente del robot y ubicado transversalmente con respecto al mismo. La primera corresponde a la ubicación más común para cualquier sistema de localización o SLAM, mientras que la

segunda se incluye para verificar cuál es el efecto al observar puntos con otro tipo de desplazamiento dentro de la imagen.

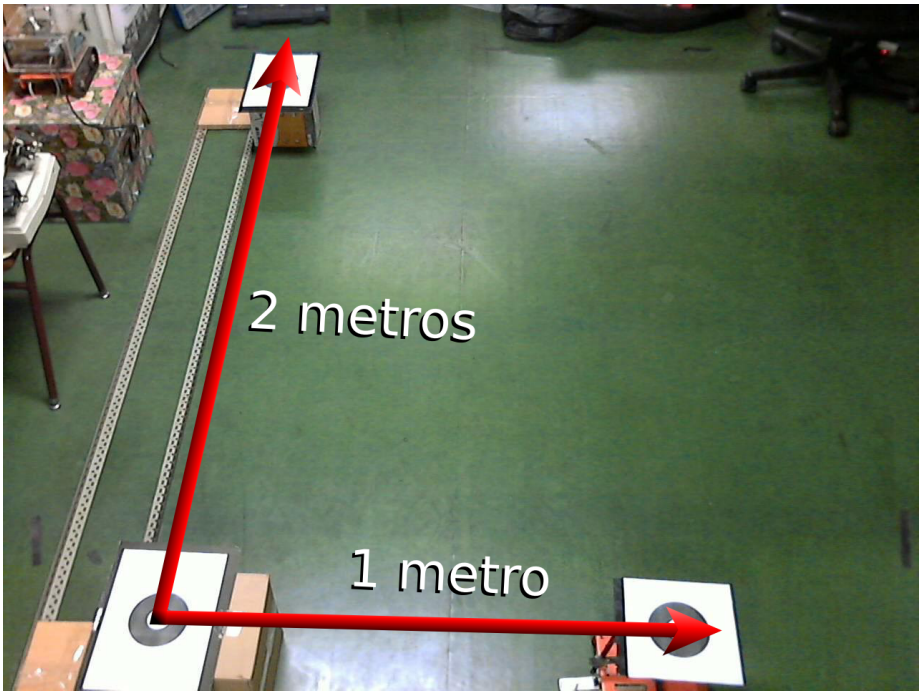


Fig. 6.5: Dimensiones utilizadas para definir el plano relativo al que se calculan las coordenadas.

En la Tabla 6.2 se muestran los parámetros de configuración del sistema de localización realizado que tienen en común en todos los experimentos. Aquellos que se encuentran fuera de este conjunto se mostrarán debidamente en las secciones siguientes.

Parámetro	PC 20	PC 60	smartphone
`ReserveFeaturesDepth`	1024	1024	1024
`ReserveFeaturesInvDepth`	1024	1024	1024
`InverseDepthRhoSD`	1,0	1,0	1,0
`MaxMapSize`	240	240	240
`AlwaysRemoveUnseenMapFeatures`	true	true	true
`MapManagementFrequency`	1	1	1
`DetectNewFeaturesImageAreasDivideTimes`	2	2	2
`MatchingCompCoefSecondBestVSFirst`	1,0	1,0	1,0
`GoodFeatureMatchingPercent`	0,5	0,5	0,5
`RansacThresholdPredictDistance`	1,0	1,0	1,0
`RansacAllInliersProbability`	0,99	0,99	0,99
`RansacChi2Threshold`	5,9915	5,9915	5,9915
`InverseDepthLinearityIndexThreshold`	0,1	0,1	0,1

Tab. 6.2: Parámetros de configuración en común para todos los experimentos.

6.1.5. Trayectorias de prueba

A continuación se detallan las trayectorias realizadas dentro del entorno junto con los valores de los parámetros para cada caso, tanto para la PC como para el smartphone. Estos parámetros fueron obtenidos de forma empírica y teniendo en cuenta los movimientos en cada uno de los experimentos.

Trayectoria recta

Este experimento consiste en realizar un desplazamiento en línea recta a velocidad constante. El objetivo es analizar el funcionamiento del sistema en un caso en donde los landmarks observados, se encuentran a una distancia relativamente grande al comienzo, situación que disminuye la información para determinar la traslación. En la Figura 6.6 se puede observar la trayectoria del robot para este experimento.

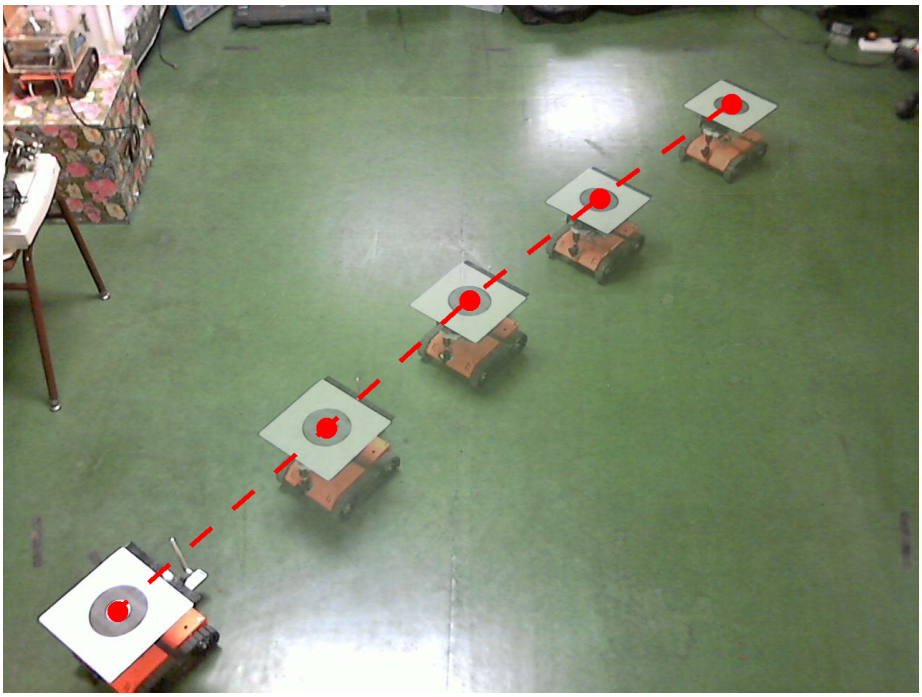


Fig. 6.6: Desplazamiento del robot realizando una trayectoria recta.

Los parámetros de configuración del sistema (descriptos en 5.2.1) utilizados para este experimento se muestran en la Tabla 6.3.

Como se puede observar, los valores de desvío estándar para la velocidad angular, tanto para cuando se inicia un feature como para las siguientes imágenes en los que aparece (`InitAngularAccelSD` y `AngularAccelSD`), son levemente más grandes. Esto se debe a que la cantidad de cuadros que puede procesar el smartphone es menor que la PC, por lo que existe más diferencia entre una imagen y otra, que sumado a las vibraciones en el movimiento del robot, generan un desplazamiento horizontal mayor de los features en la imagen. La desviación estándar fijada genera elipses de tamaño más grande, por esta razón, se aumenta valor del parámetro `DetectNewFeaturesImageMaskEllipseSize` para evitar que se agreguen al mapa features cercanos entre si dentro de la imagen.

Parámetro	PC 20	PC 60	smartphone
`InitInvDepthRho`	1,0	1,0	1,0
`InitLinearAccelSD`	0,001	0,001	0,001
`InitAngularAccelSD`	0,004	0,004	0,005
`LinearAccelSD`	0,0007	0,0007	0,0007
`AngularAccelSD`	0,002	0,002	0,003
`DetectNewFeaturesImageMaskEllipseSize`	10	10	50
`MinMatchesPerImage`	20	60	20

Tab. 6.3: Parámetros de configuración del sistema de localización realizado para el experimento línea recta con la cámara al frente.

Parámetro	PC 20	PC 60	smartphone
`InitInvDepthRho`	1,0	1,0	1,0
`InitLinearAccelSD`	0,0022	0,0022	0,006
`InitAngularAccelSD`	0,0022	0,0022	0,006
`LinearAccelSD`	0,0011	0,0011	0,003
`AngularAccelSD`	0,0011	0,0011	0,003
`DetectNewFeaturesImageMaskEllipseSize`	20	20	20
`MinMatchesPerImage`	20	60	20

Tab. 6.4: Parámetros de configuración del sistema de localización realizado para el experimento línea recta con la cámara ubicada transversalmente.

En el caso de la trayectoria con la cámara ubicada de forma transversal, el desplazamiento de los features en los cuadros del video es horizontal. En este caso el robot se encuentra a una distancia cercana de diversos objetos dentro del entorno durante todo el recorrido.

Los parámetros de configuración del sistema para esta variante se observan en la tabla 6.4.

Para ejecutar el sistema en el smartphone se realizaron algunas modificaciones en los desvíos estándar de las velocidades angulares (`InitAngularAccelSD` y `AngularAccelSD`), dadas las características mencionadas en la sección anterior. Si bien este aumento influye en el tamaño de las elipses, el valor asociado a la distancia de los nuevos features con respecto a los que ya existen (determinado por el parámetro `DetectNewFeaturesImageMaskEllipseSize`), no es necesario incrementarlo, dado que influye en los resultados obtenidos.

Trayectoria semi circular

En este experimento, el robot describe una trayectoria semi circular. Para una buena estimación de la velocidad angular se requieren landmarks lejanos que ayudan a estimar bien la rotación y para una buena estimación de la velocidad lineal, landmarks cercanos que estiman mejor la traslación. El desplazamiento del robot para este escenario se muestra en la Figura 6.7.

Parámetro	PC 20	PC 60	smartphone
`InitInvDepthRho`	0,5	0,5	0,5
`InitLinearAccelSD`	0,001	0,001	0,001
`InitAngularAccelSD`	0,005	0,005	0,005
`LinearAccelSD`	0,001	0,001	0,001
`AngularAccelSD`	0,005	0,005	0,0055
`DetectNewFeaturesImageMaskEllipseSize`	20	20	20
`MinMatchesPerImage`	20	60	20

Tab. 6.5: Valores de los parámetros de configuración utilizados para el experimento de semi círculo con la cámara al frente.

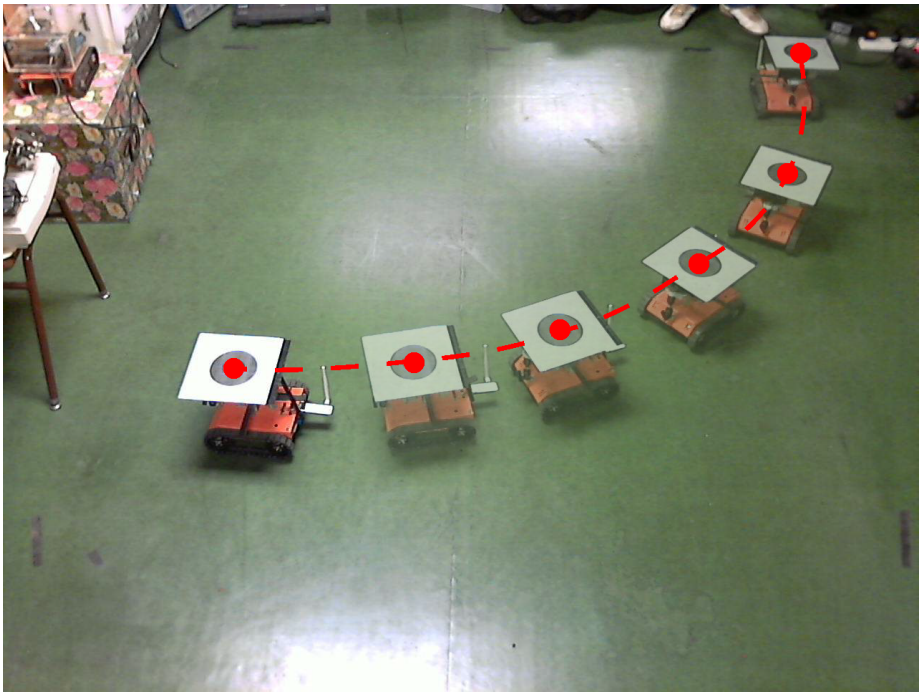


Fig. 6.7: Desplazamiento del robot realizando una trayectoria semi circular.

Los parámetros de configuración del sistema para esta variante se observan en la tabla 6.5.

En este caso se obtuvo un mejor comportamiento iniciando los nuevos features a una distancia más lejana, fijando el parámetros `InitInvDepthRho` a 0,5. Dicho valor está relacionado a la distancia inicial a la que se encuentra el robot con respecto de los objetos de donde se extraen los features, que es lejana.

La configuración utilizada para el dispositivo móvil varia, pero no demasiado con respecto a la configuración utilizada para la ejecución en la PC. Sólo el valor asociado al parámetro `AngularAccelSD` es aumentado para obtener un buen resultado y esto se debe a que con dicho valor, el tamaño de las elipses de predicción es suficientemente grande como para que sistema siempre encuentre sus correspondencias.

Para el caso en que el smartphone se encuentra ubicado de forma transversal, se observa

Parámetro	PC 20	PC 60	smartphone
InitInvDepthRho	1,0	1,0	1,0
InitLinearAccelSD	0,004	0,004	0,008
InitAngularAccelSD	0,004	0,004	0,008
LinearAccelSD	0,002	0,002	0,004
AngularAccelSD	0,002	0,002	0,004
DetectNewFeaturesImageMaskEllipseSize	30	30	150
MinMatchesPerImage	20	60	20

Tab. 6.6: Parámetros de configuración asignados al sistema de localización realizado para el experimento de trayectoria semi círculo con la cámara de forma transversal.

un escenario donde la mayoría de los features se desplazan de forma horizontal dentro de la imágenes, lo que aumenta el parallax de los mismos. Además, la distancia entre el robot y los objetos de los cuales se extraen los features es menor que en el escenario anterior, algo que permite obtener más información al sistema sobre los puntos utilizados.

La configuración de los parámetros utilizados para este caso se ven en la Tabla 6.6.

Nuevamente los parámetros utilizados para el smartphone fueron modificados aumentando los desvíos estándar de ambas velocidades (`InitLinearAccelSD`, `InitAngularAccelSD`, `LinearAccelSD` y `AngularAccelSD`) para ajustar las elipses de predicción a causa del gran desplazamiento de imagen a imagen. Además se aumentó la distancia mínima a la cual deben estar los nuevos features de los ya existentes en la imagen (parámetro `DetectNewFeaturesImageMaskEllipseSize`), teniendo en cuenta el nuevo tamaño de las elipses de predicción.

Trayectoria en zig-zag

En este experimento, el robot realiza un desplazamiento en forma zig-zag formado por dos partes semi circulares. El experimento de zig-zag contiene una combinación de movimientos que en los demás no se encuentran. En la Figura 6.8 se muestra el recorrido realizado por el Exabot.

Este escenario tiene como cualidad que los puntos detectados, se encuentran a una distancia relativamente lejana con respecto a la ubicación del robot y las mismas se van acercando a medida que el mismo termina su desplazamiento. Esto plantea una dificultad adicional para el sistema ya que al principio, el mismo tendrá menos información para calcular el desplazamiento dado que dispondrá de menor información de parallax.

La configuración de los parámetros utilizados para este caso se ven en la Tabla 6.7.

La configuración del smartphone sólo varía en los desvíos estándar de las velocidades (`InitLinearAccelSD`, `InitAngularAccelSD`, `LinearAccelSD` y `AngularAccelSD`), que aumentan su valor e impactan en el tamaño de las elipses de predicción. Para compensar esto, se aumenta también el valor del parámetro `DetectNewFeaturesImageMaskEllipseSize`, para evitar que los features que se detectan, no se encuentren muy cercanos entre si y de esta forma dar pie a errores en el proceso de matching.

La configuración de los parámetros utilizados para cuando el smartphone se encuentra ubicado de forma transversal se ven en la Tabla 6.8.

Como el movimiento de los features es diferente dada la disposición de la cámara con respecto al movimiento del robot, los valores de ruido para las aceleraciones son mayores

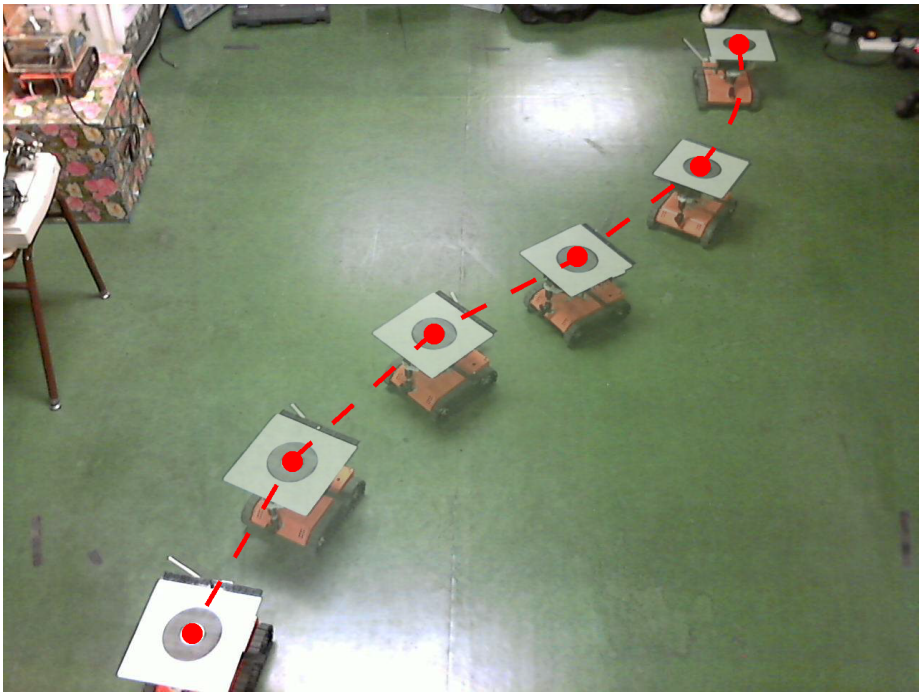


Fig. 6.8: Desplazamiento que consiste en dos semi círculos unidos por una trayectoria recta muy corta para evitar cambio de orientación repentino.

Parámetro	PC 20	PC 60	smartphone
`InitInvDepthRho`	0,5	0,5	0,5
`InitLinearAccelSD`	0,01	0,01	0,01
`InitAngularAccelSD`	0,01	0,01	0,01
`LinearAccelSD`	0,003	0,003	0,0045
`AngularAccelSD`	0,003	0,003	0,0045
`DetectNewFeaturesImageMaskEllipseSize`	10	10	50
`MinMatchesPerImage`	20	60	20

Tab. 6.7: Valores asignados a los parámetros de configuración del sistema de localización realizado para el experimento de zig-zag.

Parámetro	PC 20	PC 60	smartphone
`InitInvDepthRho`	2,0	2,0	2,0
`InitLinearAccelSD`	0,003	0,003	0,008
`InitAngularAccelSD`	0,003	0,003	0,008
`LinearAccelSD`	0,001	0,001	0,004
`AngularAccelSD`	0,001	0,001	0,004
`DetectNewFeaturesImageMaskEllipseSize`	50	50	150
`MinMatchesPerImage`	20	60	20

Tab. 6.8: Valores asignados a los parámetros de configuración del sistema de localización realizado para el experimento zig-zag con la cámara en posición transversal.

en el caso del smartphone que en el caso de la PC.

Comparando las configuraciones de este experimento entre las dos disposiciones de la cámara, se puede observar que existe una diferencia en la distancia inversa a la cual se inician todos los nuevos features al comienzo (indicado por el parámetro `InitInvDepthRho`), teniendo en cuenta las distancias entre los objetos de donde se extraen los puntos y el robot. En el caso donde la cámara se encuentra alineada con el frente, los landmarks se encuentran más lejos en la mayor parte del desplazamiento. Por ello el valor de la variable `InitInvDepthRho` es fijado en 0,5, distancia mayor que 2,0, valor utilizado para cuando el smartphone se encuentra de forma transversal al robot, donde la distancia a los objetos es menor.

6.2. Resultados

En las secciones a continuación se analizan los resultados obtenidos en tres andariveles: el primero se enfoca en el estudio de las trayectorias estimadas por el sistema, tanto en el smartphone como en la PC, comparándolas con respecto al ground truth. El segundo, consiste en el análisis de la efectividad de las correspondencias. La última parte dentro de esta sección se enfoca en el rendimiento del sistema. Es necesario tener en cuenta que los resultados obtenidos pueden estar afectados por un error causado por la vibración del robot y por la disposición de la cámara (que no se encuentra exactamente perpendicular al piso, ni perfectamente alineada al frente del robot o transversalmente).

6.2.1. Trayectoria estimada versus ground truth

La calidad de la estimación calculada por el sistema de localización es comparada contra el ground truth midiendo la diferencia más grande entre cada uno de los recorridos. De esta forma se acota superiormente el error global.

Antes de poder comparar los datos obtenidos por el sistema de localización global y los obtenidos por método de estimación, es necesario realizar dos pasos: determinar un factor de escala y hallar la orientación correcta.

El primer paso de ajuste de escalas se debe a que la información de estimación brindada por el sistema de localización desarrollado en esta tesis contiene tanto la posición en tres dimensiones de la cámara (y por ende del robot) en valores sin escala, mientras que la información obtenida desde el sistema de localización global se encuentra en una escala conocida. Por esta razón se debe determinar el factor de escala descripto en la sección 4.4.1 y de esta forma es posible obtener las posiciones de ambos sistemas en la misma escala. Para encontrar dicho factor se calcula la distancia entre landmarks bien estimados (aquellos que se encuentran representados mediante la parametrización XYZ) y la distancia en el mundo real de estos mismos landmarks.

El segundo paso consiste en alinear las posiciones de la estimación a los del ground truth, dado que el sistema de localización global no informa la orientación del robot. Para alinear los datos, se implementó una heurística consiste en el cálculo de un segmento entre la primera posición del robot y otra unos instantes después, obtenida por el sistema de localización global. Este segmento define la orientación inicial del robot. Con esta información, se realiza una traslación de las posiciones estimadas y una rotación con respecto al segmento, que tiene como resultado la alineación de las trayectorias. Los resultados que se muestran en esta sección utilizan los métodos mencionados anteriormente.

Cabe destacar que sólo se toman las posiciones estimadas en los ejes que corresponden al plano de desplazamiento utilizado por el sistema de localización global, descartando la estimación en el eje de coordenadas normal al piso.

Trayectoria recta

El desempeño del sistema para esta trayectoria resultó ser muy bueno ya que el error obtenido en la estimación del sistema realizado es pequeño respecto de los valores obtenidos por el sistema de localización global.

En la Figura 6.9 se muestra la trayectoria recta calculada por el sistema de localización global (de 2,15 metros) junto con las estimaciones realizadas con el smartphone y la PC con un mínimo de 20 correspondencias en la imagen. Como se puede observar, ambas estimaciones son muy parecidas con respecto al desplazamiento real del robot. Teniendo en cuenta que el dispositivo móvil tiene un poder de procesamiento más acotado que la PC y que el mismo puede procesar menos imágenes, se puede deducir de estos resultados que el procesar una menor cantidad de cuadros por segundo no impacta demasiado en la estimación final para este caso. Esto puede deberse a que el sistema obtiene niveles de parallax altos para los puntos del mapa dado el movimiento lateral de la cámara. El error máximo que se obtiene en cada una de las estimaciones con respecto al recorrido real se muestra en la Tabla 6.9, donde se pueden observar que dichos valores son pequeños, menores a 4 cm.

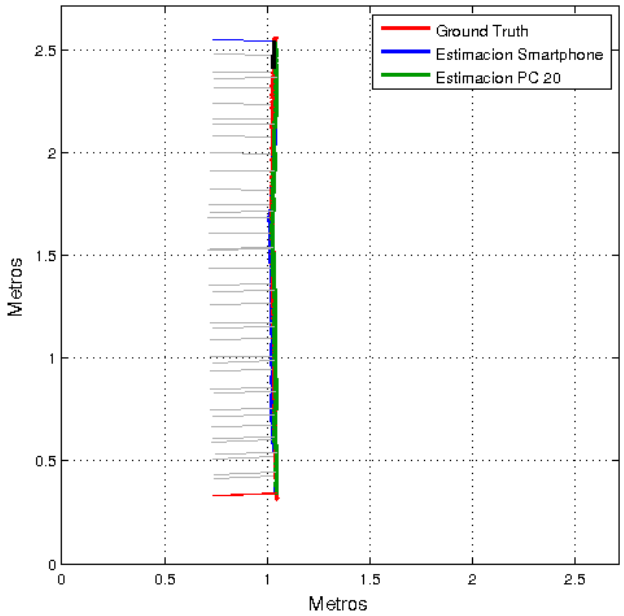


Fig. 6.9: Comparación entre la trayectoria realizada con el robot comparada con las estimaciones del sistema cuando la cámara se encuentra en posición transversal. Las líneas grises muestran la estimación de la orientación de la cámara durante el recorrido.

La trayectoria realizada cuando la cámara se encuentra mirando al frente es de 2,42 metros. El desempeño del sistema corriendo en el smartphone para este experimento es

muy similar con respecto al procesamiento en la PC y se mantiene en los mismos órdenes que en la trayectoria recta con la cámara transversal (cuyos valores se encuentran en la Tabla 6.9). En el caso del smartphone, el error aumenta al final del trayecto y resulta ser de mayor magnitud que en el caso de la posición transversal. En la Figura 6.10 se muestra el error en la estimación, que es considerablemente mayor comparando la realizada por la PC y la misma cantidad de matches mínimos dentro de la imagen. Si se observa la orientación además, se puede observar a simple vista que se encuentra levemente desviada con respecto a la orientación que debería ser, dando otra pauta de que el método de estimación en la zona final era de menor calidad.

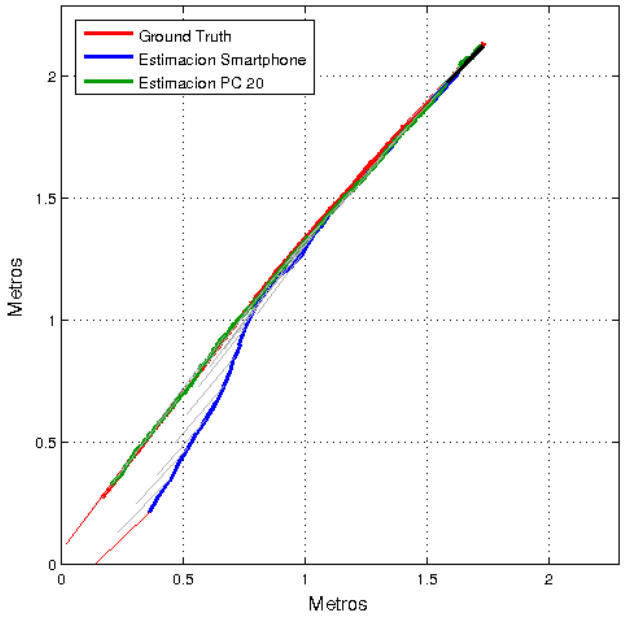


Fig. 6.10: Trayectorias en línea recta cuando el smartphone se encuentra ubicado al frente del robot. Las líneas grises representan la estimación en la orientación de la cámara. Es al final de la estimación de la trayectoria realizada por el dispositivo móvil donde se observa un desvío, responsable del error máximo obtenido para dicha plataforma dentro de este escenario.

Desplazamiento semi circular

En este caso, la velocidad angular no es despreciable y por lo tanto es necesario disponer tanto de landmarks cercanos como lejanos para obtener una buena estimación para la traslación como para la rotación del robot durante su desplazamiento. Esta condición influye en mayor medida sobre las estimaciones realizadas cuando la cámara se encuentra al frente del robot.

Como el movimiento del robot combina un cambio de orientación con un desplazamiento, la estimación resulta más difícil de realizar, algo que es más notorio ejecutando el sistema dentro del smartphone, donde el tipo de movimiento y la baja cantidad de cuadros por segundo que se llegan a procesar parecen ser los responsables del error que se observa en la Figura 6.11 con la cámara en posición transversal. La estimación dentro

de la PC cuando se tiene un mínimo de matches igual a 20, es visiblemente mejor, por lo que se puede suponer que el tipo de movimiento no influye demasiado en la calidad de la estimación de la trayectoria como el frame rate. La magnitud de los errores son mostrados en la Tabla 6.9 para un recorrido de 1,95 metros.

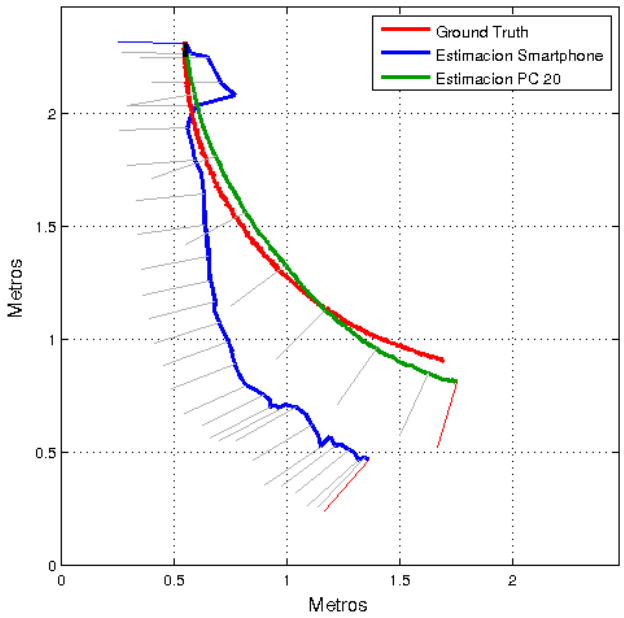


Fig. 6.11: Trayectoria real del robot, estimada por la PC con 20 correspondencias como mínimo y el smartphone cuando la cámara se encuentra de forma transversal.

Considerando las ejecuciones realizadas cuando la cámara se encuentra al frente del robot (de un recorrido real de 2,46 metros), se puede observar a simple vista en la Figura 6.12 que las estimaciones contienen un error mayor que con la disposición anterior, mostrado en la Tabla 6.9. Esta diferencia entre los errores máximos obtenidos puede deberse además del tipo de movimiento, a la distancia que existe entre los objetos de los cuales el sistema obtiene los features. Como se mencionó previamente, la distancia de estos puntos influye en la estimación. Por otro lado, los errores mayores obtenidos cuando la cámara se encuentra al frente del robot, pueden estar relacionados a una cantidad insuficiente de landmarks cercanos, que permitirían realizar una buena estimación sobre la traslación del robot. En la Figura 6.12 se puede ver que la traslación es imperfecta, ya que los recorridos estimados se encuentran desfasados en el eje vertical, algo que no se debe a una mala estimación en la orientación sino a la traslación y que se evidencia más en el procesamiento realizado por el smartphone. Se observa entonces que el movimiento en semi círculo resulta más difícil de estimar en el caso en que existe una baja cantidad de puntos cercanos.

En la Figura 6.13 se muestra como se ve el entorno desde la cámara al comienzo de cada experimento. Como se puede observar, la cantidad de landmarks relativamente cercanos al robot cuando el smartphone se encuentra alineado con la dirección de desplazamiento parece ser menor en comparación a la cantidad que existe al fondo de la escena en la Figura 6.13(a), ubicados a una distancia mucho mayor. Esta cantidad de landmarks cer-

canos ayudan a determinar el desplazamiento con respecto al real en el comienzo de la estimación, tanto para la ejecución en PC como en el smartphone. Luego a medida que el robot continúa la trayectoria, los puntos cercanos que se encuentran a la izquierda de la escena de la Figura 6.13(a) quedan fuera del ángulo de visión, degradando la estimación sobre la traslación del robot (ya que en su mayoría los puntos son lejanos), y es donde se comienzan a ver los primeros errores en la trayectoria.

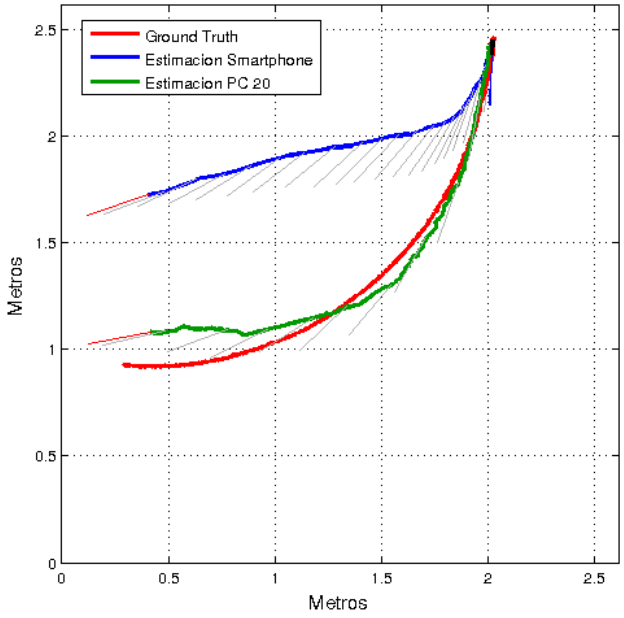


Fig. 6.12: Trayectoria del robot calculada por el sistema de localización global y por el sistema de localización realizado en esta tesis utilizando el smartphone y la PC con 20 correspondencias mínimas por imagen. La cámara se encuentra frente al robot.

El error en la estimación de la trayectoria podría disminuirse si el escenario contuviese más objetos cercanos de los cuales extraer features. Sino, la estimación podría mejorarse aumentando la cantidad mínima de matches (por ende de puntos dentro del mapa), lo que en consecuencia obtendría más cantidad de landmarks cercanos. Para el experimento realizado con la cámara al frente del robot, se realizó una estimación de la trayectoria elevando a 60 la cantidad de puntos mínimos que deben ser correspondidos en cada imagen para la versión PC. Esta nueva estimación se compara con la que utiliza 20 matches como mínimo y se puede observar en la Figura 6.14.

(a) (b)

Fig. 6.13: Entorno visto por el robot al comienzo de cada experimento al realizar la trayectoria semi circular. La distancia de los landmarks es bastante más lejana cuando la cámara se encuentra frente al robot (Figura 6.13(a)) que cuando el smartphone se encuentra trasversalmente (Figura 6.13(b)).

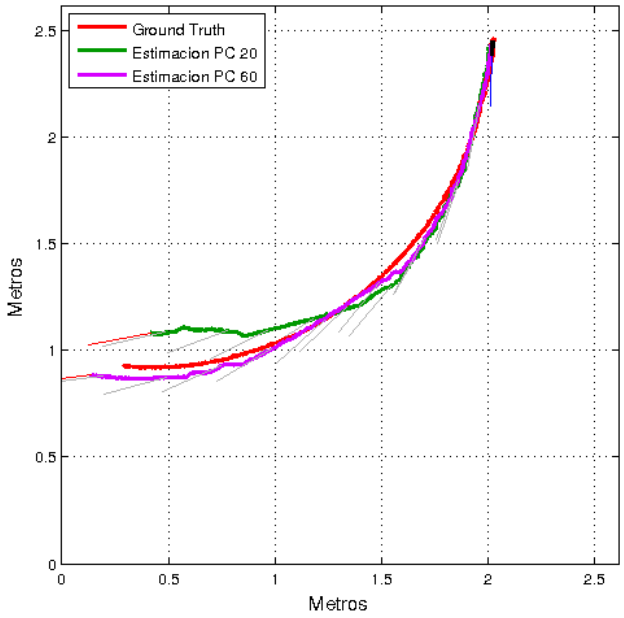


Fig. 6.14: Trayectoria semi circular estimada con una cantidad de 20 y 60 correspondencias mínimas por imagen.

Se puede observar que la trayectoria estimada por el sistema realizado se asemeja al sistema de localización global. En la Tabla 6.9 se observa que el error máximo es menor cuando se procesan más puntos en cada imagen, lo que verifica la hipótesis sobre la cantidad de features.

Desplazamiento en zig-zag

Observando la Figura 6.15 que corresponde a la comparación de las estimaciones cuando el smartphone se encuentra ubicado de forma transversal al robot, se puede notar que el error producido por el cambio de sentido después de la mitad del recorrido afecta en menor medida a la estimación realizada imagen a imagen por la PC con 20 matches como mínimo. Esto sucede tanto en el error máximo, como en la forma general de la estimación, cuyo parecido es mucho mayor al recorrido real (de un total de 2,74 metros). Esta aparente mejora puede deberse al tipo de movimiento que permite obtener mayor parallax de los puntos del mapa entre imágenes y también a la distancia de los mismos con respecto al robot, ya que este se encuentra más cerca de los objetos del entorno al momento de efectuarse el cambio de dirección, lo que reduciría su impacto en la estimación.

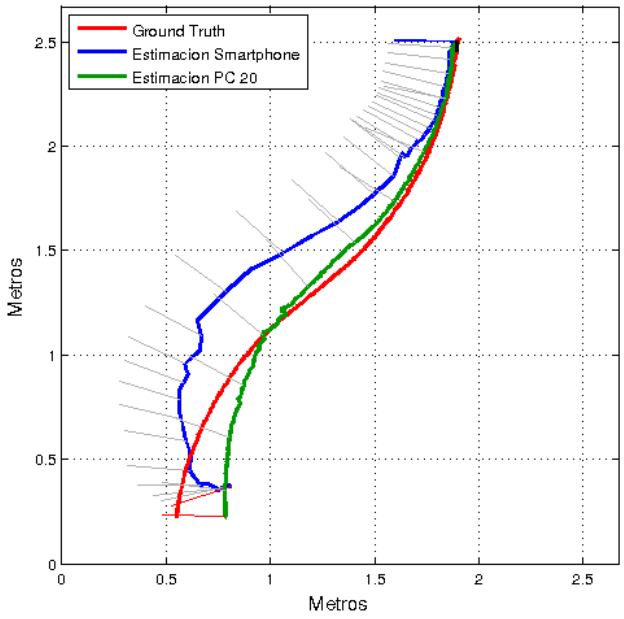


Fig. 6.15: Estimación de las trayectorias en zig-zag procesadas con el dispositivo móvil ubicado en forma transversal. Las mismas son estimadas por el smartphone y la PC con un mínimo de 20 correspondencias.

Algo diferente ocurre cuando no es posible procesar todas las imágenes del video. Este caso correspondiente a la estimación realizada por el dispositivo móvil (que también tiene un mínimo de 20 matches en la imagen), muestra que el movimiento que permite obtener mayor parallax de los landmarks se vuelve contraproducente si la taza de procesamiento de cuadros por segundo es baja, ya que la trayectoria realiza un desplazamiento semi circular más cerrado que el real, como se puede ver en la Figura 6.15. Este hecho, sumado a la distancia cercana al final del recorrido, genera que los desplazamientos de los features entre imágenes que pueden procesarse sea muy grande y sus posiciones escapen por fuera de las elipses de predicción, que permite obtener una estimación más ajustada al recorrido real. Estos desplazamientos que exceden los límites de la elipses producen una menor cantidad de correspondencias dentro del sistema, lo que disminuye la posibilidad de realizar

una buena estimación del movimiento a partir de las imágenes asociadas a esa parte del recorrido.

En la Tabla 6.9 se observan errores mayores que disponiendo la cámara de frente y pueden deberse a la cercanía de los puntos dentro del entorno al final del recorrido y al cambio de trayectoria, como se menciona en párrafos anteriores.

El experimento realizado con el smartphone posicionado al frente del robot muestra un comportamiento similar al del desplazamiento semi circular. Sin embargo en este caso la trayectoria no difiere tan rápidamente cuando la estimación es realizada por el dispositivo móvil, sino que parece desviarse al comienzo y luego mantener un error similar hasta el final de la estimación, como se muestra en la Figura 6.16. Este error más estable puede deberse a una velocidad angular menor para este experimento, donde los desplazamientos de los features en las imágenes son menores (haciéndolos más predecibles), algo que favorecería al dispositivo móvil teniendo en cuenta su poder de procesamiento.

El cambio de sentido parece afectar más a la estimación realizada por la PC, que procesa todas las imágenes del video. Esto se puede inferir observando también la Figura 6.16 ya que la estimación antes de la mitad del recorrido, resulta ser más exacta y se corresponde en gran medida con el recorrido real (de un total de 2,65 metros), y después de la mitad, que es cuando se invierte el sentido de giro del robot, la estimación comienza a diferir de a poco con respecto al ground truth realizando un movimiento de semi círculo más cerrado de lo que es realmente.

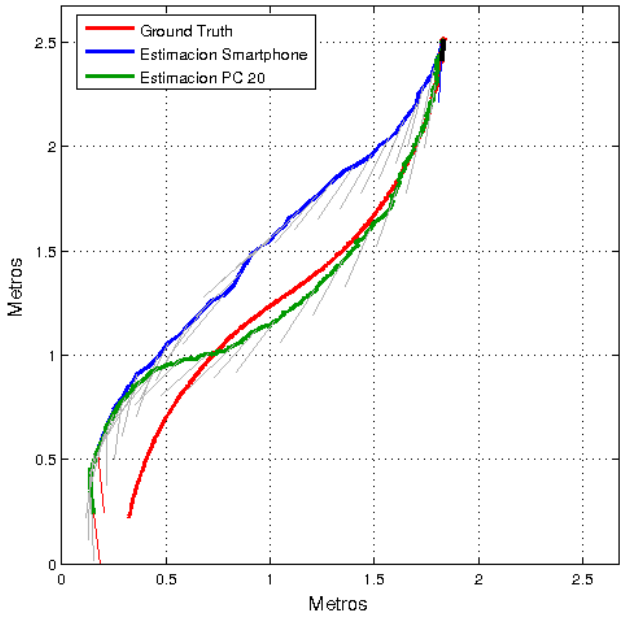


Fig. 6.16: Trayectoria en zig-zag con la cámara dispuesta al frente del robot.

Aunque la trayectoria estimada por la PC difiere considerablemente al final del recorrido, el error máximo obtenido comparando la estimación realizada por el smartphone, resulta ser menor. Aunque el cálculo del error promedio no se realiza, a simple vista también se puede ver que el error en general es mucho menor cuando se procesan todas las

imágenes disponibles en la PC con 20 matches como mínimo, haciendo visible que es requerido un frame rate más alto para obtener mejores resultados en el dispositivo móvil. Como se observa en la Tabla 6.9, el error máximo resulta ser aproximadamente la mitad del error en el dispositivo móvil.

Como sucede en el recorrido semi circular de la sección anterior, es posible que la estimación del desplazamiento se encuentre afectada por una cantidad baja de landmarks cercanos, dado que se comienza a una distancia relativamente lejana de los elementos de donde se extraen los mismos y además por la calidad de los puntos cercanos al final del recorrido. Por este motivo se realizó una nueva estimación donde se procesan todas las imágenes del video y se aumenta a 60 la cantidad mínima de correspondencias dentro de la imagen, con el objetivo de disminuir el error de traslación en el caso de estar lejos de los objetos y disponer de mayor cantidad de puntos cuando el robot se encuentra cerca. El resultado de esta ejecución arrojó resultados muy buenos donde la estimación difiere muy poco con respecto al recorrido real, obteniendo un error de sólo 5,3 cm, como se muestra en la Tabla 6.9. La estimación de dicho recorrido se muestra en la Figura 6.17 donde se observa una mejoría notable en la calidad de la estimación. En este caso, la estimación de la trayectoria al final del recorrido tiene una similitud muy grande con respecto al *ground truth*.

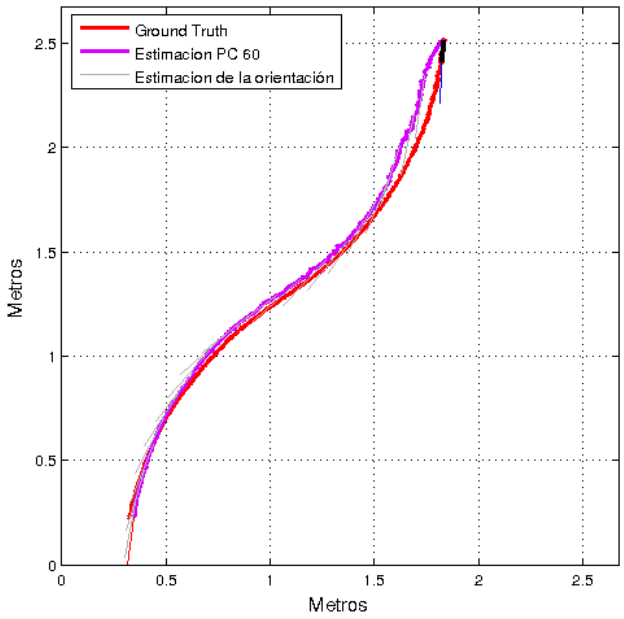


Fig. 6.17: Estimación de las trayectorias en zig-zag procesando todas las imágenes del video con 60 correspondencias mínimos en la imagen.

Experimento	smartphone [cm]	PC 20 [cm]	PC 60 [cm]
Línea recta frontal	20,1 (8,3 %)	6,4 (2,6 %)	-
Línea recta transversal	2,9 (1,3 %)	3,2 (1,4 %)	-
Semi círculo frontal	80,5 (32,7 %)	19,9 (8,0 %)	-
Semi círculo transversal	54,5 (27,9 %)	11,4 (5,8 %)	-
Zig-zag frontal	35,3 (12,8 %)	17,0 (6,2 %)	5,3 (1,9 %)
Zig-zag transversal	47,34 (17,8 %)	23,4 (8,8 %)	-

Tab. 6.9: Errores máximos en cm de la trayectoria de cada uno de los experimentos junto con el porcentaje sobre la distancia total recorrida, para las ejecuciones en el smartphone y la PC con 20 y 60 correspondencias mínimas por imagen.

6.2.2. Desempeño del sistema en función de las correspondencias detectadas

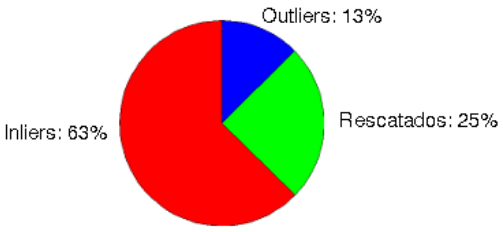
Otro posible análisis a realizar sobre las estimaciones obtenidas corresponde a observar las correspondencias efectuadas y la proporción de éstas que fueron utilizadas para mejorar la estimación.

En cada uno de los experimentos se analiza la cantidad de matches determinada por el sistema, la proporción de éstos que fueron inliers (aquellos pertenecientes al mejor conjunto de consenso determinado por RANSAC), utilizados para realizar la primer actualización, y la proporción de correspondencias rescatadas, para luego para actualizar de forma definitiva la estimación realizada en cada iteración. Estas proporciones ayudan a entender mejor los resultados obtenidos en la estimación de la trayectoria de cada uno de los experimentos.

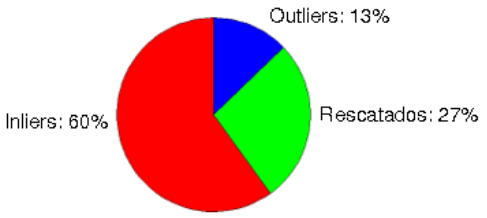
Trayectoria recta

Como se muestra en la sección donde se analizan los errores en las estimaciones del sistema realizado con respecto al sistema de localización global, el error obtenido en todas las ejecuciones de este experimento es bajo. Por medio del análisis del uso de las correspondencias en las ejecuciones realizadas dentro de la PC, se puede ver en la Figura 6.18 un alto porcentaje alto de correspondencias inlier. Esto indica no sólo la efectividad del algoritmo RANSAC para determinar un conjunto con buen soporte, si no también la certeza del sistema de localización realizado para predecir el movimiento del robot en cada iteración.

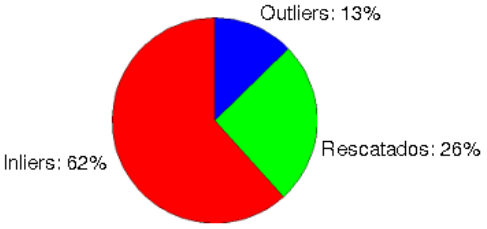
Se puede ver que la etapa de rescate resulta ser muy efectiva dado que se utiliza un porcentaje alto de aquellas correspondencias descartadas por el algoritmo RANSAC, dejando sólo el 13 % de matches sin usar. Sumando los porcentajes correspondientes a la primer actualización y la segunda, se obtiene un 87 % de correspondencias, que resulta ser muy bueno. Estos porcentajes son similares para todas las ejecuciones realizadas por la PC, con 20 y 60 correspondencias como mínimo y con ambas posiciones de cámara, que también pueden apreciarse en la Figura 6.18, donde se destaca la similitud entre éstas variantes, que es un resultado acorde a los comportamientos observados en este experimento.



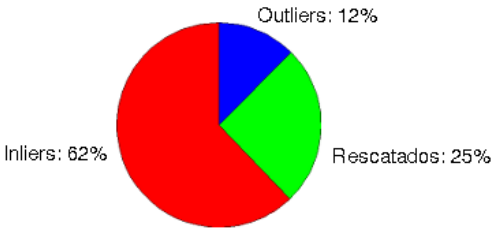
(a) Cámara al frente, 20 matches como mínimo.



(b) Cámara al frente, 60 matches como mínimo.



(c) Cámara transversal, 20 matches como mínimo.

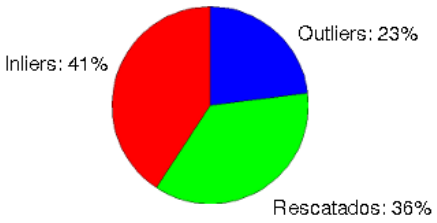


(d) Cámara transversal, 60 matches como mínimo.

Fig. 6.18: Porcentaje de correspondencias inlier, rescatados y outliers, para todas las ejecuciones dentro de la PC, con 20 y 60 matches como mínimo y con ambas disposiciones de la cámara.

La relación entre los distintos tipos de correspondencias obtenidas mediante las ejecuciones en el smartphone, se encuentran más distribuidas que en los casos anteriores, aunque la suma de los porcentajes de inliers y rescatados se mantiene relativamente alto, por encima del 65 % del total. Estas proporciones parecen ser suficientes para realizar una estimación aceptable dentro este experimento. En la Figura 6.19 se muestran las proporciones correspondientes para el caso del smartphone. Se puede ver que existe una mínima diferencia en el porcentaje de inliers utilizados, que resulta ser mayor cuando la cámara se encuentra en posición transversal. Tal como se pudo ver al analizar las estimaciones correspondientes a las dos disposiciones de la cámara, para el caso de la disposición transversal se puede ver en la Figura que la cantidad de correspondencias finales (inliers más rescatados) es mayor en este caso, hecho que suele estar asociado con buenas estimaciones.

(a) Cámara al frente.

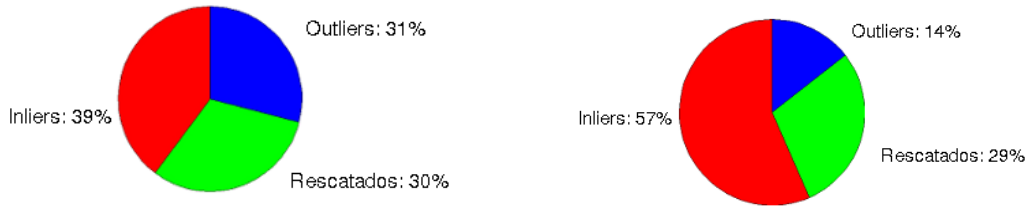


(b) Cámara transversal.

Fig. 6.19: Porcentaje de correspondencias obtenidas por el sistema cuando es ejecutado en el smartphone, con una cantidad de 20 matches como mínimo.

Desplazamiento semi circular

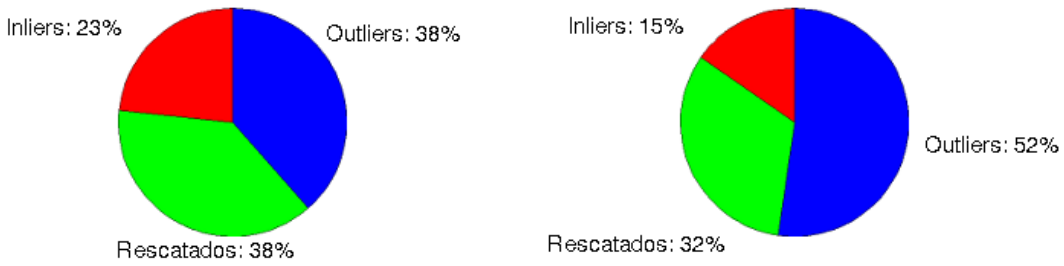
Las diferencias obtenidas entre las estimaciones del sistema de localización realizado y el recorrido real dentro de este experimento pueden analizarse mejor observando las proporciones de correspondencias obtenidas. En la Figura 6.20(a), correspondiente al caso cuando la cámara esta al frente, se puede ver que la cantidad de inliers es considerablemente menor que en el caso de la cámara transversal, que se presenta en la Figura 6.20(b). Esto va de la mano con la diferencia entre las estimaciones en estos dos casos. Un comportamiento similar se puede ver en el caso de 20 correspondencias como mínimo.



(a) Cámara al frente, 60 matches mínimo. (b) Cámara transversal, 60 matches mínimo.

Fig. 6.20: Proporción de correspondencias utilizadas por el sistema en las ejecuciones en la PC con 60 matches como mínimo.

Por otro lado, en el caso del smartphone (presentado en la Figura 6.21), en donde la mejor estimación también corresponde al caso de la cámara transversal se observan resultados opuestos al analizar el porcentaje de inliers. Esto parece indicar que por más que haya una menor cantidad de correspondencias finales, para el caso de la cámara transversal, las mismas probablemente arrojen más información sobre el desplazamiento de la cámara, debido al alto parallax de los puntos asociados.



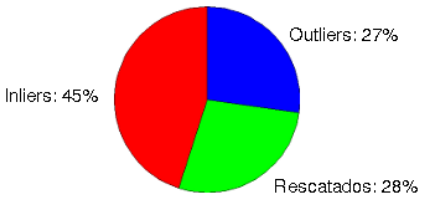
(a) Cámara al frente. (b) Cámara transversal.

Fig. 6.21: Correspondencias utilizadas para actualizar el sistema realizado en ejecuciones con el smartphone.

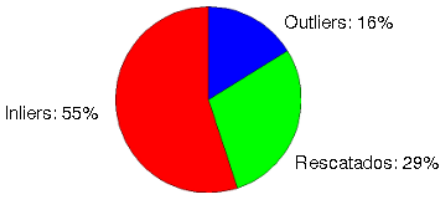
En general, se puede deducir, que en algunas ocasiones el entorno puede compensar la falta de información para estimar la trayectoria del robot y aún así obtener resultados aceptables.

Desplazamiento en zig-zag

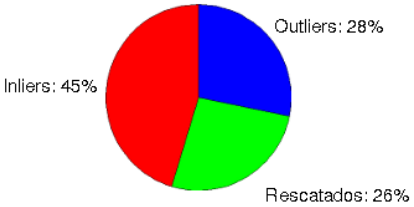
En este experimento, se pudo observar una diferencia considerable entre usar 20 correspondencias como mínimo y 60, sin importar la plataforma. Analizando para el caso particular de la PC la proporción de correspondencias (Figura 6.2.2) puede verse que en realidad en este caso no se observa una diferencia considerable al modificar el parámetro mencionado. A partir de esto, se puede concluir que para algunos existe una cantidad mínima de correspondencias necesarias para obtener una buena estimación más allá de la proporción de los distintos tipos de éstas.



(a) Cámara al frente, 20 matches como mínimo.



(b) Cámara transversal, 20 matches como mínimo.



(c) Cámara al frente, 60 matches como mínimo.

(d) Cámara transversal, 60 matches como mínimo.

Fig. 6.22: Porcentaje de correspondencias utilizadas durante las ejecuciones en PC.

Por otro lado, analizando los casos en donde la estimación no fue del todo satisfactoria (20 correspondencias como mínimo) se puede ver que, como se menciona antes el momento en el que el robot cambia el sentido del giro parece corresponder a la fuente del error en la estimación en general. De esta forma, resulta de interés analizar la cantidad de correspondencias de cada tipo en cada uno de los pasos de estimación. Dicho análisis se presenta en la Figura 6.23 para el caso particular del smartphone.

Como se puede observar en la Figura mencionada, en los primeros pasos de estimación existe una cantidad elevada de correspondencias inlier y rescatados, y es donde existe un menor error en la trayectoria estimada. Sin embargo, a partir del paso 25 aproximadamente, la cantidad de correspondencias finales resulta ser menor que el mínimo establecido (20). Por esta razón se observa en este paso un gran incremento en la cantidad de matches candidatos. Aunque la cantidad de correspondencias candidatas aumenta, no se observa el mismo incremento en la cantidad de inliers, sino que disminuye. Esto parecería indicar que las nuevas correspondencias candidatas, no se condicen con el modelo estimado o, en otras palabras, no aportan información útil.

Cabe destacar además que la cantidad de matches totales en algunos pasos puede encontrarse por debajo del límite de 20. Esto no es algo que pueda ser controlado, dado

que depende de poder extraer suficientes features dentro de las elipses y de establecer correspondencias mediante la comparación de sus descriptores.

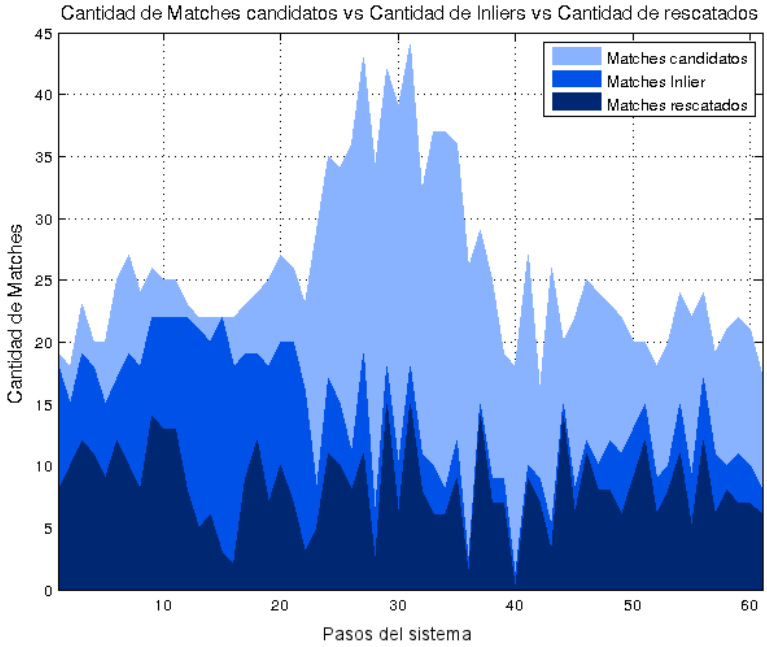


Fig. 6.23: Cantidad de correspondencias encontradas de cada tipo, para el caso de la cámara transversal, la ejecución en el smartphone y con 20 matches como mínimo.

6.2.3. Tiempos de ejecución

Un aspecto que resulta interesante examinar es el rendimiento del sistema cuando es ejecutado en las distintas plataformas. Este análisis permite identificar las partes de la implementación más costosas computacionalmente y de esta forma determinar cuáles pueden ser el foco de optimizaciones.

En la Figura 6.24 se muestran los tiempos de ejecución promedio que requiere cada parte del sistema en los distintos experimentos y plataformas. Las etapas del sistema presentadas son las siguientes: predicción, DEM (Detección, Extracción y *Matching*), RANSAC, primer y segunda actualización, rescate de outliers y gestión del mapa. La etapa DEM incluye la tarea de detección de features, la extracción de descriptores para cada uno dentro de las elipses de predicción y el proceso de matching.

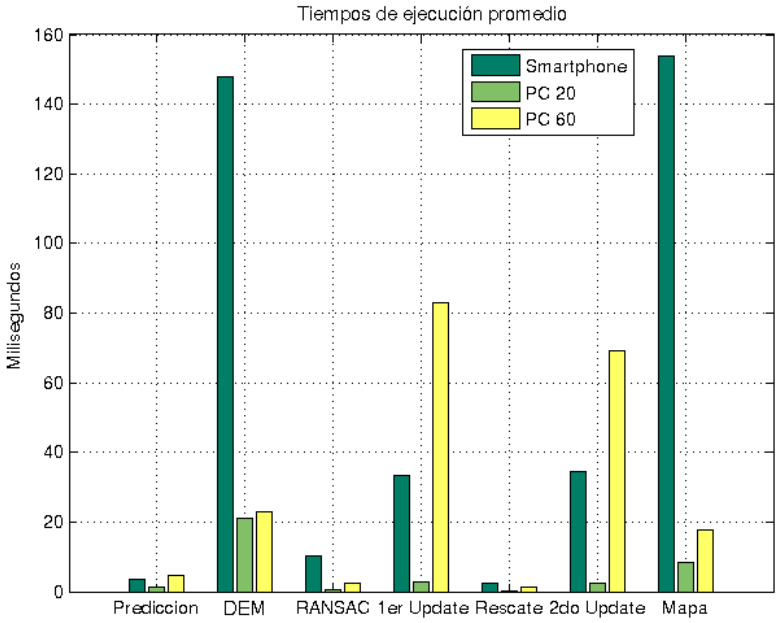


Fig. 6.24: Tiempo de procesamiento promedio para todos los escenarios y posiciones de cámara, ejecutados en el smartphone, PC con 20 correspondencias mínimo y PC con 60 correspondencias mínimo.

En particular, en la Tabla 6.10 se muestra el porcentaje promedio del tiempo utilizado por cada una de las tareas dentro de la etapa DEM. Como se puede observar, el proceso de detección de features resulta ser el que más tiempo consume. Tomando como ejemplo el caso de ejecución en la PC con 20 correspondencias como mínimo, se observó un total de 21,7 milisegundos para la etapa DEM. De este total, la detección de features requiere 18,7 milisegundos, la extracción de descriptores toma 2,6 milisegundos y la etapa de matching requiere menos de un milisegundo (0,39 milisegundos). Este último tiempo considerablemente bajo, es producto de utilizar descriptores tipo BRIEF cuya comparación se realiza computando la distancia de Hamming, la cual es una operación de costo mínimo.

Detección de features	Extracción de descriptores	Matching
86,2 %	12 %	1,8 %

Tab. 6.10: Porcentaje promedio de tiempo consumido para cada una de las tareas dentro de la etapa DEM.

Los valores más bajos de tiempo de procesamiento para todas las etapas, corresponden a las ejecuciones realizadas en la PC con la menor cantidad de correspondencias mínimas por imagen (20). En este caso se destaca la etapa DEM como la que consume mayor tiempo de procesamiento con un valor promedio de 21 milisegundos. Sumando los tiempos de todas etapas del sistema se obtiene en promedio, un tiempo de 36,6 milisegundos para procesar cada imagen, lo que determina una tasa de procesamiento de 27 cuadros por segundo. Teniendo en cuenta los requerimientos de tiempos de respuesta para el caso particular del

robot ExaBot, esto corresponde a procesamiento en tiempo real.

En la Figura 6.25 se muestran los tiempos promedio de cada etapa del sistema para el caso de ejecución con PC y 20 correspondencias como mínimo, junto con sus desvíos estándar (los valores correspondientes se encuentran en la Tabla 6.11). Algo llamativo resulta ser la gran variación de los tiempos de la etapa de gestión. Esto se debe a que el agregado y la eliminación de puntos del mapa no se realiza en todas las iteraciones, sino que ocurre según ciertas condiciones descritas en secciones previas de esta tesis. En la Figura 6.26 se presentan los tiempos consumidos por la etapa de gestión del mapa para cada uno de los pasos del sistema, donde se evidencia esta gran variación.

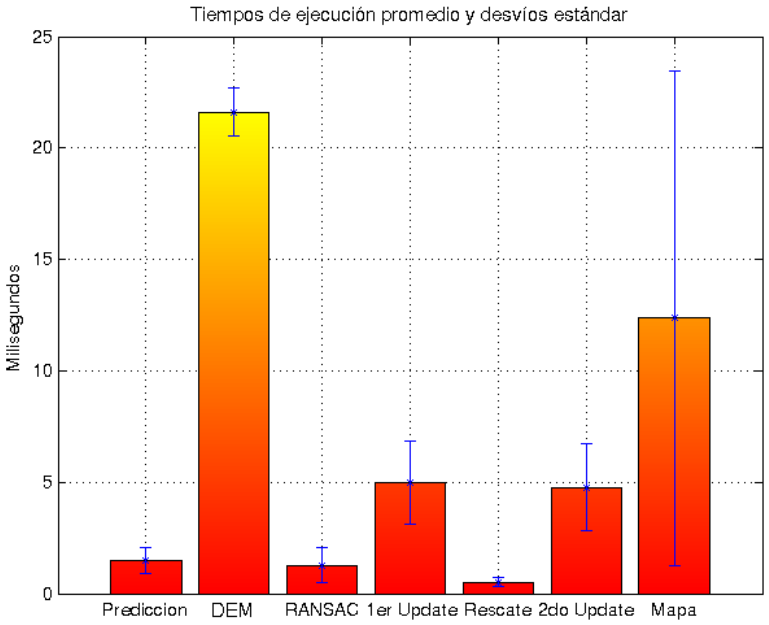


Fig. 6.25: Detalle de los tiempos de procesamiento promedio que requiere cada parte del sistema en la plataforma PC con 20 matches mínimos junto con su desvío estándar.

	Predicción	DEM	RANSAC	1er Update	Rescate	2do Update	Mapa
Tiempo	1.5	21.6	1.3	4.9	0.5	4.8	12.4
Desvío	0.6	1.1	0.8	1.9	0.2	1.9	11.0

Tab. 6.11: Tiempos promedio (en milisegundos) y desvíos estándar de las ejecuciones en PC con 20 matches como mínimo, asociados a la Figura 6.25.

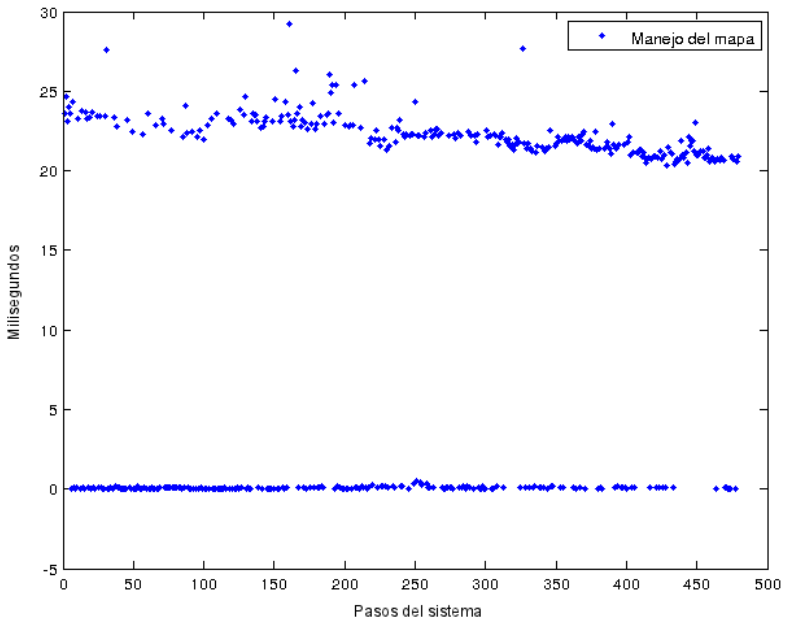


Fig. 6.26: Tiempo en milisegundos consumido por la etapa de gestión para cada uno de los pasos de estimación.

En cuanto al smartphone, como es esperado, los tiempos de procesamiento resultan ser mayores que para el caso de la PC debido al reducido poder de cómputo del que dispone. Se puede observar que la etapa DEM junto con la de gestión del mapa son las que dominan el tiempo total de procesamiento requerido para cada paso, que asciende a 385,9 milisegundos, lo que equivale a una tasa de 2,5 cuadros por segundo. Por otra parte, las etapas de actualización del modelo Extended Kalman Filter no requieren un tiempo de ejecución considerable respecto del total. Esto parece deberse a la cantidad acotada de puntos que suele haber en el mapa. El detalle de los tiempos requeridos junto con sus desvíos se encuentran en la Figura 6.27 y en la Tabla 6.12, respectivamente. Analizando los desvíos, éstos resultan ser más pequeños en comparación con los de la PC, sobre todo en la etapa de gestión del mapa. Este desvío menor se debe a que en el smartphone, dado su reducida tasa de procesamiento por segundo, los puntos no son tan frecuentemente correspondidos correctamente lo que resulta en eliminaciones frecuentes de los mismo en el mapa. Asimismo, para compensar esta falta de puntos en el mapa, el agregado de los mismos también se realiza frecuentemente. En otras palabras, esta etapa requiere un promedio de ejecución mayor y en forma sostenida que en el caso de la PC.

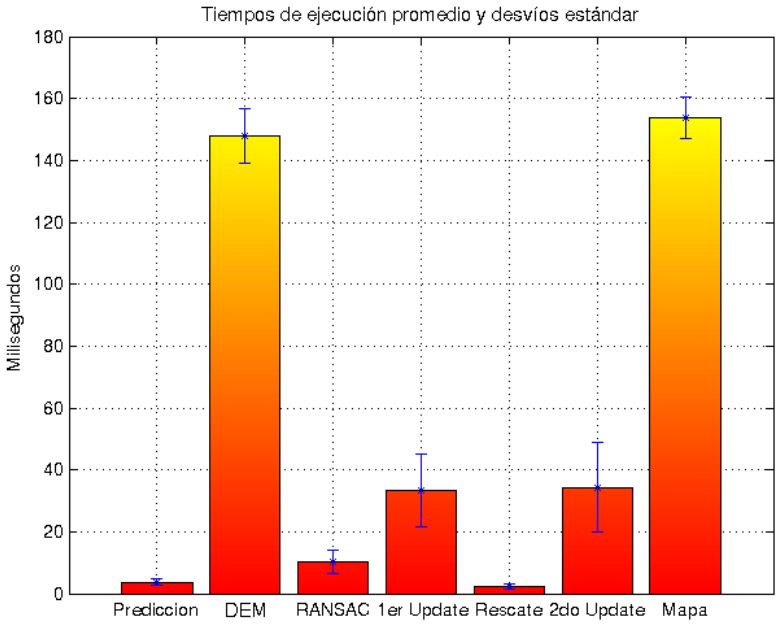


Fig. 6.27: Tiempos de procesamiento promedio en el smartphone junto con su desvío estándar.

	Predicción	DEM	RANSAC	1er Update	Rescate	2do Update	Mapa
Tiempo	3.7	147.9	10.3	33.4	2.3	34.4	153.9
Desvío	1.0	8.7	3.9	11.6	0.8	14.6	6.8

Tab. 6.12: Tiempos promedio (en milisegundos) y desvíos estándar de las ejecuciones en el smartphone correspondientes a la Figura 6.27.

Volviendo al caso de la PC, al utilizar un mínimo de correspondencias de 60, se hace evidente que los tiempos que gobiernan las ejecuciones de cada paso corresponden a las etapas de actualización de la estimación. Estos tiempos, que superan los 60 milisegundos, llevan el tiempo total requerido para procesar cada imagen a 201,2 milisegundos en promedio, lo que supone una tasa de 5 cuadros por segundo. Sin embargo, los tiempos de respuesta asociados siguen siendo suficientemente bajos para lograr ejecuciones en tiempo real en el caso del ExaBot.

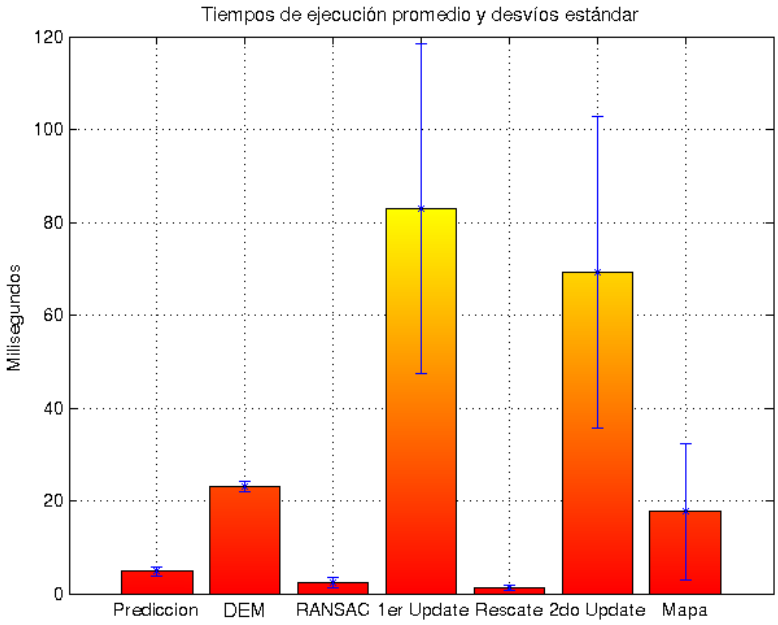


Fig. 6.28: Promedio de tiempos requeridos para procesar cada imagen para la plataforma PC con 60 matches mínimos junto con su desvío estándar.

	Predicción	DEM	RANSAC	1er Update	Rescate	2do Update	Mapa
Tiempo	4.7607	23.062	2.3946	82.878	1.2851	69.166	17.708
Desvío	1.1005	1.0152	1.1715	35.499	0.46091	33.61	14.673

Tab. 6.13: Tiempos promedio en milisegundos y desvíos estándar de cada uno de las partes del sistema en las ejecuciones dentro de la PC con un mínimo de 60 correspondencias en cada imagen.

Para este último caso, la elevada cantidad de puntos en el mapa, además de aumentar los tiempos promedio de procesamiento para las etapas de actualización, genera grandes desvíos en las mismas, como se puede ver en la Figura 6.28. Esto se debe a que para los casos en los que exista una gran cantidad de puntos en el mapa, ligeras variaciones en la cantidad de los mismos se traducen en grandes variaciones en los tiempos correspondientes, debido a la complejidad no lineal de los algoritmos de actualización del estado. En la tabla 6.13 se muestran los valores de tiempos promedio y desvío estándar para cada etapa del sistema.

7. CONCLUSIONES

En esta tesis se presenta el desarrollo de un sistema de localización visual monocular basado en el enfoque conocido como EKF-SLAM. Este sistema es *open source* (código abierto) y tiene dos versiones: una para smartphones y otra para PC. La evaluación del sistema se llevó a cabo utilizando el robot móvil ExaBot y el smartphone Samsung Galaxy S3. El smartphone se montó sobre el robot y se utilizó la cámara del dispositivo como sensor. Los resultados obtenidos fueron satisfactorios al ser comparados con un sistema de localización global.

Estos experimentos incluyeron distintas trayectorias de movimiento: en línea recta, semicírculo y zig-zag. Se contemplaron dos casos para la ubicación de la cámara: de frente y transversal a la orientación del robot. Cada una de estas distintas trayectorias y diferentes ubicaciones de la cámara fueron evaluadas tanto para la versión PC como para la versión smartphone. En la versión PC, los errores se ubicaron entre 1,4 % y 8,8 % del total de la trayectoria recorrida. En la versión smartphone, los errores máximos fueron entre 1,3 % y 32,7 % del total de la trayectoria recorrida. Estos resultados confirman que la versión PC resulta ser más precisa ya que procesa más imágenes que la versión para smartphones. Además, los resultados de los experimentos muestran que en algunos casos no es requerida una cantidad de correspondencias alta para obtener un resultado aceptable (por ejemplo, en el caso de trayectoria de línea recta con cámara transversal, con 20 correspondencias es suficiente).

En algunos experimentos se exponen resultados con un límite inferior de 60 correspondencias para mostrar que su trayectoria puede ser bien estimada con una mayor cantidad de correspondencias. Este límite es inferior al que se usa en otros trabajos (generalmente más de 100 correspondencias como mínimo), y aún así en esta tesis se lograron buenos resultados. Esto se debe posiblemente al uso de nuevos detectores de características en las imágenes (features) como STAR y nuevos descriptores de features como BRIEF, que incrementan el rendimiento y la robustez al establecer correspondencias.

Otro aspecto que parece incidir en la calidad de la estimación de la localización es la disposición de la cámara. Observando los resultados se puede concluir que la ubicación de modo transversal en general obtiene mejores estimaciones de la trayectoria realizada. Esta diferencia con respecto a la ubicación frontal de la cámara puede estar ligada a un mayor ángulo de parallax de los puntos del mapa (landmarks).

Por otro lado, se comparó el rendimiento de la versión PC contra la versión de smartphone: la primera logra procesar 27 cuadros por segundo, mientras que la segunda alcanza 2,5 cuadros por segundo. Este salto se debe básicamente a diferencias en el poder de cómputo entre ambas plataformas de hardware. Además, de cada parte del método (la predicción, la detección, extracción y matching de features, RANSAC, la primera actualización, el rescate, la segunda actualización y la gestión del mapa) se evaluó el tiempo de ejecución. Se puede apreciar que tanto la predicción de puntos del mapa en la imagen como filtrado con RANSAC y rescate de outliers, tienen tiempos de ejecución en general reducidos con respecto a las demás partes. Sin embargo, la detección, extracción y matching de features, la actualización y la gestión del mapa dependen del experimento y del parámetro de mínima cantidad de correspondencias. El costo de la detección, extracción y el matching de features se encuentra limitado mayormente por la detección de features

(método STAR), y predomina en los experimentos de pocas correspondencias (20 mínimo), ya sea para la versión PC como para la versión smartphone. Además, en el caso de smartphones, el costo de la detección de features y extracción de descriptores es solamente equiparado por las tareas de gestión del mapa. Esto se debe a la baja tasa de procesamiento de cuadros por segundo alcanzada para smartphones que afecta directamente las mediciones, provocando una baja cantidad de inliers y rescatados.

En los experimentos de mayor cantidad de correspondencias (60 mínimo), el costo del algoritmo resulta condicionado por la actualización del estado (tanto por la primera como por la segunda actualización). Aquí es donde es notoria la limitación propia del enfoque EKF-SLAM en lo que respecta a la complejidad del cálculo del Kalman Gain, que incluye el cómputo de la inversa de una matriz de grandes dimensiones. Es esta una de las causas principales que impulsan la investigación y el desarrollo de nuevas técnicas SLAM como PTAM, que separa las funcionalidades de modo que el mapeo (el más costoso) no retrase la localización y de esta manera aprovechar la potencia de los procesadores actuales multicore.

Algunos de los resultados de los experimentos ejecutados en el smartphone demuestran que no es necesario una alta tasa de cuadros por segundo para obtener buenas estimaciones (por ejemplo, en el caso de trayectoria en linea recta con la cámara transversal). Sin embargo, en otros experimentos, procesar las imágenes con una baja frecuencia afecta la estimación, lo que es especialmente notorio en el caso de desplazamiento semicircular con la cámara dispuesta al frente. En el mismo, la estimación parece haber sido buena en cuanto a la orientación de la cámara, aunque no fue así con respecto a la traslación. El error en la trayectoria estimada podría disminuirse si el escenario contuviera más objetos cercanos de los cuales extraer features. Entre las alternativas para obtener mejores resultados en las estimaciones realizadas por el smartphone se pueden considerar: utilizar un dispositivo móvil con mayor poder de cómputo (no disponibles al momento de desarrollo de esta tesis) y/o realizar optimizaciones al sistema de localización propuesto en esta tesis.

El único factor limitante para obtener una buena estimación de la localización de la cámara utilizando el enfoque EKF-VisualSLAM con dispositivos smartphone es su capacidad de cómputo y, aunque esto no parece ser un gran problema considerando la última tendencia, se puede observar que con suficiente cantidad de cuadros por segundo el desempeño del método propuesto es satisfactorio. Además, esto último es reforzado por los resultados de los experimentos ejecutados en PC, que en general logran un muy buen rendimiento y una estimación eficiente, y que indica que el sistema que se entrega como parte de esta tesis fue correctamente implementado.

En el trabajo de Civera *et al.* se reporta que con un procesador Intel(R) Core(TM) i7 2.67GHz se logró una ejecución en tiempo real (30 cuadros por segundo) con imágenes con una resolución de 320×240 pixeles. En comparación con esto, el sistema desarrollado en esta tesis logra una ejecución de rendimiento similar (27 cuadros por segundo) con un procesador Intel(R) Core(TM) i3-2310M 2.10GHz y con imágenes del doble de tamaño (640×480 pixeles). Esta mejora puede deberse en gran parte a la modificación realizada en lo que refiere a la utilización del detector STAR y los descriptores de BRIEF en contraposición a FAST y los parches como descriptores del trabajo de Civera.

Por último, es importante notar que el valor de los parámetros del método resulta ser de gran importancia para su desempeño. Varios de estos parámetros se mantuvieron invariantes durante los experimentos (su valor fue fijado a partir del trabajo de Civera et. al.). Otros en general varían, como los que involucran la velocidad angular y lineal y

el que fija la cantidad mínima de correspondencias. Además, a partir de los resultados realizados, se pudo observar que el método es muy sensible a los valores de los parámetros. Por lo tanto el ajuste de estos valores resultan de suma importancia para lograr una buena estimación de la localización de la cámara durante toda la trayectoria del robot.

7.1. Trabajo a futuro

El trabajo realizado en esta tesis ofrece un amplio horizonte de posibilidades de interesantes extensiones. Una alternativa posible es incorporar los comandos del robot a la estimación, ya sea obteniendo la orden de velocidades provistas a los motores del robot o utilizando sensores adicionales como encoders. De este modo se puede fusionar este método con otro sistema de odometría y así lograr una estimación más precisa. Para esto es necesario extender el modelo dinámico del método propuesto en esta tesis.

También se puede integrar al sistema otros sensores adicionales con los que cuenta la gran mayoría de smartphones (GPS, compás, acelerómetro). El compás y el acelerómetro serían útiles para estimar mejor la orientación de la cámara y su velocidad. El GPS permite el uso de un método de localización global, que junto con el sistema desarrollado en esta tesis se lograría una estimación más precisa para trayectorias de largo recorrido en ambientes exteriores. Para la incorporación de estos nuevos sensores también sería necesario extender el modelo de sensado del método que se presenta en este trabajo.

Otro posible trabajo futuro consiste en incorporar el mapeo del ambiente para utilizar el sistema en problemas de SLAM. Para esto se podría implementar las extensiones de *Submapping* [78] y *Loop Closure* [79]. El primero trata de acotar la cantidad de puntos del mapa involucrados en la actualización, organizándolo en una jerarquía de mapas de menor magnitud. De esta manera se logra mejorar su rendimiento, ya que la parte del EKF que involucra el cálculo de la inversa de la matriz que contiene la incertidumbre del mapa se mantiene constante. El segundo trata de aprovechar la información presente en el mapa cuando se cierra un ciclo en la trayectoria realizada por la cámara, ajustando la posición de todos los puntos del mapa involucrados.

Como ya se dijo anteriormente, el método es muy sensible al valor de los parámetros. Una pequeña variación en el valor de alguno puede provocar grandes cambios en la estimación. Considerando además la cantidad de parámetros involucrados, sería de gran utilidad evaluar una forma de prescindir de algunos (que sean calculados automáticamente) y/o hacer más estable el método ante cambios en sus valores. Esto facilitaría la ejecución online del método (capturar imágenes con la cámara y estimar simultáneamente).

El modelo dinámico del método presentado en esta tesis permite 6 grados de libertad de movimiento. El robot terrestre utiliza solamente 2 grados de libertad. Sería interesante realizar experimentos con robots aéreos de tipo cuadracópteros que permitan movimientos de 6 grados de libertad y de este modo aprovechar al máximo las capacidades del sistema.

Por último, sería interesante comparar la eficiencia y rendimiento de este método con otros actuales, como por ejemplo PTAM, para contrastar las limitaciones y ventajas de los mismos en contraste con EKF SLAM Monocular e investigar la mejor opción para realizar odometría visual utilizando un dispositivo smartphone.

Apéndice

A. CONFIGURACIÓN DEL SISTEMA

En este apéndice se numeran los parámetros disponibles para configurar la ejecución del método desarrollado en esta tesis. Estos parámetros se encuentran dentro de un archivo con formato *YML* que deberá ser provisto para ejecutar el sistema.

A.1. Parámetros de configuración

A.1.1. EKF SLAM (`ExtendedKalmanFilter`)

- `ReserveFeaturesDepth` y `ReserveFeaturesInvDepth`: define un límite inicial para la cantidad de información con la que instanciarán las estructuras internas del estado.
- `InitLinearAccelSD`, `InitAngularAccelSD`, LinearAccelSD y AngularAccelSD: determinan los desvíos estándar de la aceleración lineal y angular, que se ajustan para evitar que el estimador no pierda información de acuerdo a la rapidez con que se mueve el robot al inicio. Una vez en movimiento, luego del primer *frame* procesado, el modelo de movimiento requiere los desvíos estándar para las mismas aceleraciones que se utilizan para modelar el ruido en el modelo dinámico del sistema.
- `InitInvDepthRho` y `InverseDepthRhoSD`: fijan la distancia inversa con la cual los puntos del mapa de tipo Inverse Depth se inicializan, tanto para el primer paso del sistema como para los que siguen.
- `MaxMapSize`: define el tamaño máximo de la matriz de covarianza dentro del sistema, lo que impone un límite dentro a la cantidad de *features* del mapa. El valor indicado se asigna teniendo en cuenta la cantidad de *features* Inverse Depth, que son los que requieren más coordenadas y por lo tanto requieren más espacio dentro de la matriz de covarianza.
- `AlwaysRemoveUnseenMapFeatures`: directiva para indicar al método de gestión del mapa que elimine siempre o no aquellos puntos que quedan fuera del ángulo de visión de la cámara.
- `MapManagementFrequency`: indica la frecuencia con la cual los métodos de gestión del mapa son ejecutados, determinando cada cuantos pasos se quiere ejecutar dichas políticas.
- `DetectNewFeaturesImageAreasDivideTimes` y `DetectNewFeaturesImageMaskEllipseSize`: definen los valores asociados a la cantidad de celdas en que se divide la imagen y el tamaño de las elipses alrededor de los *features* ya existentes a la hora de agregar nueva información al mapa.
- `MatchingCompCoefSecondBestVSFirst`, `MinMatchesPerImage` y `GoodFeatureMatchingPercent`: fijan valores asociados a las correspondencias. Al momento de determinar el mejor punto en la imagen que se corresponde con el *feature* a corresponder se obtienen los dos mejores puntos cuyas distancias al descriptor son las menores, luego aquel con menor distancia debe ser a su vez menor

que el segundo mejor multiplicado por un factor, dicho factor es el definido por MatchingCompCoefSecondBestVSFirst. Cuando se hablo de la posibilidad de llevar una estadística sobre los *features* usados en los procesos de actualización y los predichos en la imagen se definió un porcentaje por el cual, aquellos *features* que se encontraran por debajo, serían eliminados, este porcentaje puede ser ajustado mediante GoodFeatureMatchingPercent, así como la cantidad mínima de *matches* en cada imagen por MinMatchesPerImage.

- `RansacThresholdPredictDistance`, `RansacAllInliersProbability` y `RansacChi2Threshold`: el comportamiento del método de RANSAC es definido por estos parámetros. La probabilidad requerida para encontrar un consenso con todos sus puntos *inliers* viene dado por RansacAllInliersProbability, la distancia máxima entre el *match* y la predicción con la que se considera que un punto en la imagen es o no un *inlier* se define en RansacThresholdPredictDistance y el valor necesario de la distribución *chi* cuadrado que ayuda a calcular la cantidad máxima de iteraciones del método (el valor N definido en secciones anteriores), fijado por RansacChi2Threshold.

- `InverseDepthLinearityIndexThreshold`: define el valor utilizado para detectar cuando resulta conveniente convertir un *feature* a su representación XYZ, como se explica en la sección 3.1.3.

A.1.2. Calibración de cámara (`CameraCalibration`)

- `PixelsX` y `PixelsY`: definen el tamaño de las imágenes a procesar en pixels.

- `FX` y `FY`: determinan la distancia focal en cada uno de los ejes en pixels.

- `K1` y `K2`: se indican los coeficientes que definen la curva de distorsión radial del lente de la cámara utilizada.

- `CX` y `CY`: representa el punto principal dentro de la imagen obtenido luego de la calibración y se encuentra representado en pixels.

- `DX` y `DY`: tamaño de cada pixel dentro de la pastilla *CCD* de la cámara en milímetros (de centro a centro) en las dos direcciones X e Y.

- `PixelErrorX` y `PixelErrorY`: representan el desvío estándar del error en la reproyección en pixels obtenida luego de la calibración.

- `AngularVisionX` y `AngularVisionY`: los ángulos de visión del lente de la cámara son requeridos y es en estos parámetros donde los mismos se establecen.

Bibliografía

[1] I.R. Nourbakhsh R. Siegwart. *Introduction to autonomous mobile robots*. The MIT Press, 2004.

[2] S. Thrun et al. Robotic mapping: A survey. *Exploring artificial intelligence in the new millennium*, pages 1–35, 2002.

[3] H. Choset, K.M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L.E. Kavraki, and S. Thrun. *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005.

[4] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping (SLAM): Part I. *Robotics & Automation Magazine, IEEE*, 13(2):99–110, 2006.

[5] T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping: Part ii. *Robotics & Automation Magazine, IEEE*, 13(3):108–117, 2006.

[6] J. Borenstein, HR Everett, L. Feng, and D. Wehe. Mobile robot positioning-sensors and techniques. Technical report, DTIC Document, 1997.

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

[8] A. Martinelli, N. Tomatis, and R. Siegwart. Simultaneous localization and odometry self calibration for mobile robot. *Autonomous Robots*, 22(1):75–85, 2007.

[9] J. Civera, O.G. Grasa, A.J. Davison, and JMM Montiel. 1-point ransac for extended kalman filtering: Application to real-time structure from motion and visual odometry. *Journal of Field Robotics*, 27(5):609–631, 2010.

[10] D. Scaramuzza, F. Fraundorfer, and R. Siegwart. Real-time monocular visual odometry for on-road vehicles with 1-point ransac. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 4293–4299. IEEE, 2009.

[11] A.I. Comport, E. Malis, and P. Rives. Accurate quadrifocal tracking for robust 3d visual odometry. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 40–45. IEEE, 2007.

[12] K. Konolige, M. Agrawal, and J. Sola. Large-scale visual odometry for rough terrain. *Robotics Research*, pages 201–212, 2011.

[13] J. Borenstein and L. Feng. Measurement and correction of systematic odometry errors in mobile robots. *Robotics and Automation, IEEE Transactions on*, 12(6):869–880, 1996.

[14] C.W. Tan and S. Park. Design of accelerometer-based inertial navigation systems. *Instrumentation and Measurement, IEEE Transactions on*, 54(6):2520–2530, 2005.

[15] H. Chung, L. Ojeda, and J. Borenstein. Accurate mobile robot dead-reckoning with a precision-calibrated fiber-optic gyroscope. *Robotics and Automation, IEEE Transactions on*, 17(1):80–84, 2001.

[16] C.F. Olson. Probabilistic self-localization for mobile robots. *Robotics and Automation, IEEE Transactions on*, 16(1):55–66, 2000.

[17] Dieter Fox. *Markov localization: A probabilistic framework for mobile robot localization and navigation*. PhD thesis, PhD thesis, Dept. of Computer Science, University of Bonn, Germany, 1998.

[18] Robert Baczyk, Andrzej Kasinski, and Piotr Skrzypczynski. Vision-based mobile robot localization with simple artificial landmarks. 2003.

[19] Gijeong Jang, Sungho Lee, and Inso Kweon. Color landmark based self-localization for indoor mobile robots. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 1, pages 1037–1042. IEEE, 2002.

[20] André Guilherme Nogueira Coelho dos Santos. *Autonomous Mobile Robot Navigation using Smartphones*. PhD thesis, Master thesis, Dept. of Computer Science, Universidade Técnica de Lisboa, Portugal, 2008.

[21] S. Thrun, W. Burgard, D. Fox, et al. *Probabilistic robotics*, volume 1. MIT press Cambridge, MA, 2005.

[22] A. Kosaka and A.C. Kak. Fast vision-guided mobile robot navigation using model-based reasoning and prediction of uncertainties. *CVGIP: Image understanding*, 56(3):271–329, 1992.

[23] L.J. Latecki, R. Lakaemper, X. Sun, and D. Wolter. Building polygonal maps from laser range data. In *INTERNATIONAL COGNITIVE ROBOTICS WORKSHOP (COGROB)*, volume 4, pages 1–7. Citeseer, 2004.

[24] A. Nuchter, H. Surmann, and J. Hertzberg. Automatic model refinement for 3d reconstruction with mobile robots. In *3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings. Fourth International Conference on*, pages 394–401. IEEE, 2003.

[25] A. Howard and N. Roy. The robotics data set repository (radish), 2003.

[26] Simone Ceriani, Giulio Fontana, Alessandro Giusti, Daniele Marzorati, Matteo Matteucci, Davide Migliore, Davide Rizzi, Domenico G. Sorrenti, and Pierluigi Taddei. Rawseeds ground truth collection systems for indoor self-localization and mapping. *Auton. Robots*, 27(4):353–371, November 2009.

[27] J. Guivant, E. Nebot, and S. Baiker. Autonomous navigation and map building using laser range sensors in outdoor applications. *Journal of robotic systems*, 17(10):565–583, 2000.

[28] S. Fazli and L. Kleeman. Simultaneous landmark classification, localization and map building for an advanced sonar ring. *Robotica*, 25(3):283–296, 2007.

[29] T. Deissler and J. Thielecke. Feature based indoor mapping using a bat-type uwb radar. In *Ultra-Wideband, 2009. ICUWB 2009. IEEE International Conference on*, pages 475–479. IEEE, 2009.

[30] P. Buschka and A. Saffiotti. Some notes on the use of hybrid maps for mobile robots. In *Proc. of the 8th Int. Conf. on Intelligent Autonomous Systems*, pages 547–556, 2004.

[31] G. Michael Youngblood, Lawrence B. Holder, and Diane J. Cook. A framework for autonomous mobile robot exploration and map learning through the use of place-centric occupancy grids. In *ICML Workshop on Machine Learning of Spatial Knowledge*, 2000.

[32] M. Nitsche, P. de Cristoforis, M. Kulich, and K. Kosnar. Hybrid mapping for autonomous mobile robot exploration. In *Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), 2011 IEEE 6th International Conference on*, volume 1, pages 299–304. IEEE, 2011.

[33] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localisation and mapping (slam): Part i the essential algorithms. *IEEE ROBOTICS AND AUTOMATION MAGAZINE*, 2:2006, 2006.

[34] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localisation and mapping (slam): Part ii state of the art. *Robotics & Automation Magazine, IEEE*, 13(3):108–117, September 2006.

[35] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *In Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 593–598. AAAI, 2002.

[36] Jinwoo Choi, SungHwan Ahn, and Wan Kyun Chung. Robust sonar feature detection for the slam of mobile robot. In *IROS*, pages 3415–3420, 2005.

[37] Juan D. Tardós, José Neira, Paul M. Newman, and John J. Leonard. Robust mapping and localization in indoor environments using sonar data. *Int. J. Robotics Research*, 21:311–330, 2002.

[38] L.M. Paz, P. Piniés, J.D. Tardós, and J. Neira. Large-scale 6-dof slam with stereo-in-hand. *Robotics, IEEE Transactions on*, 24(5):946–957, 2008.

[39] A.J. Davison, I.D. Reid, N.D. Molton, and O. Stasse. Monoslam: Real-time single camera slam. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052–1067, 2007.

[40] Pedro Piniés and Juan D Tardós. Large-scale slam building conditionally independent local maps: Application to monocular vision. *Robotics, IEEE Transactions on*, 24(5):1094–1106, 2008.

[41] Hauke Strasdat, JMM Montiel, and Andrew J Davison. Scale drift-aware large scale monocular slam. In *Proceedings of Robotics: Science and Systems (RSS)*, volume 2, page 5, 2010.

[42] Sunhyo Kim and Se-Young Oh. Slam in indoor environments using omni-directional vertical and horizontal line features. *Journal of Intelligent & Robotic Systems*, 51(1):31–43, 2008.

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

[44] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT Press, 2008.

[45] Greg Welch and Gary Bishop. An introduction to the kalman filter, 1995.

[46] J.E. Guivant and E.M. Nebot. Optimization of the simultaneous localization and map-building algorithm for real-time implementation. *Robotics and Automation, IEEE Transactions on*, 17(3):242–257, 2001.

[47] Lina M. Paz, Juan D. Tardos, and Jose Neira. Divide and Conquer: EKF SLAM in O(n). *IEEE Transactions on Robotics*, 24(5):1107–1120, OCT 2008.

[48] JMM Montiel, Javier Civera, and Andrew J Davison. Unified inverse depth parametrization for monocular slam. *analysis*, 9:1, 2006.

[49] J. Nieto, J. Guivant, and E. Nebot. Denseslam: Simultaneous localization and dense mapping. *The International Journal of Robotics Research*, 25(8):711–744, 2006.

[50] Margarita Chli and Andrew J. Davison. Active matching.

[51] G. Klein and D. Murray. Parallel tracking and mapping on a camera phone. In *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*, pages 83–86. IEEE, 2009.

[52] Anson Alexander. Smartphone usage statistics 2012, 2013.

[53] Zisserman A. Hartley R. *Multiple view geometry in computer vision*. Cambridge University Press, 2008.

[54] Óscar Mozos, Arturo Gil, Monica Ballesta, and Oscar Reinoso. Interest point detectors for visual slam. *Current Topics in Artificial Intelligence*, pages 170–179, 2007.

[55] Arturo Gil, Oscar Martinez Mozos, Monica Ballesta, and Oscar Reinoso. A comparative evaluation of interest point detectors and local descriptors for visual slam. *Machine Vision and Applications*, 21(6):905–920, 2010.

[56] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[57] V Vassilevska Williams. Breaking the coppersmith-winograd barrier. *Unpublished manuscript, Nov*, 2011.

[58] Javier Civera, Andrew J Davison, and J Montiel. Inverse depth parametrization for monocular slam. *Robotics, IEEE Transactions on*, 24(5):932–945, 2008.

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

[60] Edward M Mikhail, James S Bethel, and J Chris McGlone. *Introduction to modern photogrammetry*, volume 31. Wiley New York, 2001.

[61] Davide Scaramuzza, Friedrich Fraundorfer, and Roland Siegwart. Real-time monocular visual odometry for on-road vehicles with 1-point ransac. In *Proceedings of the 2009 IEEE international conference on Robotics and Automation*, ICRA'09, pages 488–494, Piscataway, NJ, USA, 2009. IEEE Press.

[62] Javier Civera, Andrew J. Davison, and J. M. M. Montiel. Dimensionless monocular slam. In Joan Martí, José-Miguel Benedí, Ana Maria Mendonça, and Joan Serrat, editors, *Pattern Recognition and Image Analysis, Third Iberian Conference, IbPRIA 2007, Girona, Spain, June 6-8, 2007, Proceedings, Part II*, volume 4478 of *Lecture Notes in Computer Science*, pages 412–419. Springer, 2007.

[63] E. Buckingham. On physically similar systems, illustrations of the use of dimensional equations. *PHYSICAL REVIEW*, 4(4):345–376, 1914.

[64] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.

[65] Edward Rosten and Tom Drummond. Fusing points and lines for high performance tracking. In *IEEE International Conference on Computer Vision*, volume 2, pages 1508–1511, October 2005.

[66] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision (to appear)*, May 2006.

[67] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (SURF). *Computer vision and image understanding*, 110(3):346–359, 2008.

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

[69] Motilal Agrawal, Kurt Konolige, and Morten Blas. Censure: Center surround extremas for realtime feature detection and matching. *Computer Vision–ECCV 2008*, pages 102–115, 2008.

[70] Pablo De Cristóforis. *Sistema de navegación monocular para robots móviles en ambientes interiores/exteriores*. PhD thesis, Departamento de Robótica y Sistemas Embebidos, Universidad de Buenos Aires, Argentina, 2013.

[71] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *International Conference on Robotics and Automation*, Shanghai, China, 2011 2011.

[72] Google. Android software development kit (sdk).

[73] Sun Microsystems. Java native interface (jni).

[74] Google. Android native development kit (ndk).

[75] S. Pedre, P. De Cristóforis, J. Caccavelli, and A. Stoliar. A mobile mini robot architecture for research, education and popularization of science. *Journal of Applied Computer Science Methods, Guest Editors: Zurada, J., Estevez*, page 2, 2010.

[76] Jean-Yves Bouguet. Camera calibration toolbox for matlab.

[77] Eos Systems Inc. Photomodeler.

[78] Carlos Estrada, José Neira, and Juan D Tardós. Hierarchical slam: real-time accurate mapping of large environments. *Robotics, IEEE Transactions on*, 21(4):588–596, 2005.

[79] L.A. Clemente, A.J. Davison, I. Reid, J. Neira, and J.D. Tardós. Mapping large loops with a single hand-held camera. In *Robotics: Science and Systems*, 2007.