

Estado del arte

# Investigación en autolocalización visual

Marcos Pieras Sagardoy

Móstoles, 27 de Junio de 2016

# Índice

<b>1. Introducción</b>	<b>3</b>
1.1. Robótica . . . . .	4
1.1.1. Localización . . . . .	5
1.2. Visión artificial . . . . .	6
1.2.1. Autolocalización visual . . . . .	6
1.3. Realidad aumentada . . . . .	6
<b>2. Métodos de autolocalización visual</b>	<b>7</b>
2.1. Balizas artificiales . . . . .	7
2.1.1. <i>Perspective-n-Points</i> . . . . .	8
2.2. Métodos geométricos . . . . .	9
2.3. Métodos probabilísticos . . . . .	11
2.3.1. Métodos evolutivos . . . . .	13
2.4. VisualSlam . . . . .	13
<b>3. Algoritmos de visualSlam</b>	<b>15</b>
3.1. MonoSlam . . . . .	15
3.1.1. Funcionamiento . . . . .	15
3.1.2. Resultados . . . . .	18
3.2. PTAM . . . . .	19
3.2.1. Funcionamiento . . . . .	20
3.2.2. Resultados . . . . .	20
3.3. Basados en métodos directos . . . . .	21
3.4. SVO . . . . .	23
3.4.1. Funcionamiento . . . . .	23
3.4.2. Resultados . . . . .	26
3.5. LSD-Slam . . . . .	28
3.5.1. Funcionamiento . . . . .	29
3.5.2. Resultados . . . . .	31
3.6. ORB-SLAM . . . . .	32
3.6.1. Funcionamiento . . . . .	32
3.6.2. Resultados . . . . .	34
<b>4. Benchmarks</b>	<b>35</b>
<b>5. Conclusiones</b>	<b>36</b>
5.1. Trabajo futuro . . . . .	36

# 1. Introducción

La autolocalización visual, consiste en la determinación de la posición y orientación de la cámara utilizando para ello únicamente las imágenes recibidas de ésta, se ha convertido en un tema muy popular en la investigación. Primordialmente por dos aplicaciones, su uso en robótica, especialmente en sistemas *unmanned aerial vehicles* y en aplicaciones de realidad aumentada/virtual. Sistemas que se han introducido en el mercado para el público general.

En robótica, la tecnología está suficientemente madura para vender productos como el *Dyson 360 eye*, que podemos observar en la figura 1, un robot aspiradora capaz de autolocalizarse visualmente y llevar a cabo tareas de limpieza.



Figura 1: Robot aspiradora *Dyson 360 eye*

En el campo de la realidad virtual/aumentada, se han desarrollado sistemas de autolocalización visual para poder proyectar hologramas en la escena e interactuar con las personas. Un ejemplo es el sistema *HoloLens* de *Microsoft*, se puede observar su funcionamiento en la figura 2.



Figura 2: Ejemplo de funcionamiento del sistema *HoloLens*

La optimización de los algoritmos de autolocalización ha llegado al punto de poder utilizarse en dispositivos móviles. Un ejemplo es el proyecto Tango de *Google*, un sistema para poder mapear el entorno y ubicarse en él. En la figura 3 se puede ver una captura de su funcionamiento.

Finalmente, por nivel de maduración y grado de disruptividad la aplicación estrella son los coches autónomos, que son legales en algunos estados de EEUU. El uso de coches

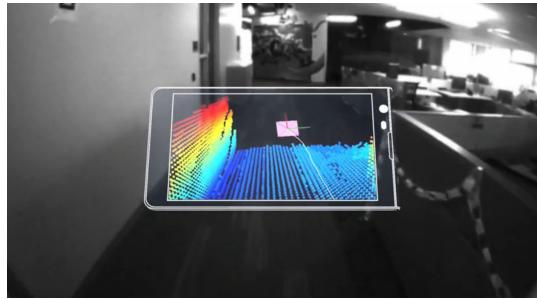


Figura 3: Ejemplo de funcionamiento del proyecto Tango.

autónomos ( figura 4 ) permitirá reducir los accidentes de tráfico, la congestión en la carretera y el consumo de combustible. Aunque el sensor principal es un LIDAR, estos sistemas usan cámaras para localizarse.



Figura 4: Coche autónomo de *Google*.

Este trabajo se estructura de la siguiente manera. Despues de haber definido el problema se contextualizará dentro de los campos donde se aplica. En la sección 2 se explicarán los métodos generales para resolver el problema de la autolocalización visual, seguidamente, en la sección 3 se describirán los algoritmos más importantes de este campo. En la sección 4 se explicarán los diferentes *dataset* con que se comparan los algoritmos. El trabajo finaliza con unas conclusiones

## 1.1. Robótica

La robótica es la rama de las ingenierías mecánica, electrónica y de las ciencias de la computación que se encarga del diseño, construcción, operación y programación de robots. Un robot se puede definir como un sistema informático que puede percibir su entorno mediante sensores, interactuar con él mediante actuadores todo ello dirigido por un sistema informático. Los robots se utilizan para desempeñar labores de riesgo, que requieren velocidad o precisión que está fuera del alcance humano.

Los robots perciben su entorno mediante sensores, los sensores son instrumentos que traducen una magnitud física en una magnitud eléctrica o digital. Detrás de los sensores debe haber un procesamiento para extraer la información útil.

### 1.1.1. Localización

Un robot para realizar sus tareas con éxito necesita poder navegar por el entorno. Para llevar a cabo una correcta navegación, el robot debe llevar a cabo cuatro tareas [29]:

- **Percepción**, el robot debe interpretar sus sensores para extraer información útil de los datos.
- **Localización**, el robot debe determinar su posición en el entorno.
- **Cognición**, el robot decide como debe actuar para conseguir sus objetivos.
- **Control del movimiento**, el robot debe modular la respuesta de sus actuadores para conseguir la trayectoria deseada.

Centrándonos en la localización, esta se puede dividir en tres categorías de problemas [34]:

- **Seguimiento de la posición**, este problema asume que la posición inicial es conocida y que el ruido en los procesos es pequeño. Se puede aproximar la posición mediante una distribución unimodal gaussiana.
- **Localización global**. Aquí la posición inicial del robot es desconocida y no se puede delimitar la incertidumbre en ella. Así utilizar distribuciones unimodales fallan.
- **Problema ante secuestro**. En este problema, durante el funcionamiento, el robot puede ser raptado y teletransportado a otra localización. La habilidad de recuperación con fallos es esencial para obtener un comportamiento autónomo.

Hay sensores específicos para resolver el problema de la localización, estos sensores proporcionan información sobre los movimientos del robot o sobre la posición concreta respecto a un sistema de referencia. Algunos de ellos son:

- **Inertial measurement unit**, [IMU] estos sensores combinaciones de acelerómetros y giroscopios. Estos sensores permiten obtener las aceleraciones y los giros a los que se somete al robot. Integrando dos veces la aceleración se obtiene la posición. La desventaja de estos sensores es que con el paso del tiempo acumulan mucho error.
- **Navegación por satélite**, la información devuelta por estos dispositivos es una posición global del robot en el mundo. El sistema consiste conociendo la posición de varios satélites triangula la posición del robot en la tierra, además hace una corrección con estaciones en la tierra. La desventaja de estos sistemas es que a veces la señal no está disponible, como en zonas montañosas o en entornos interiores, además proporcionan una resolución muy grande el orden de magnitud de los movimientos del robot.
- **Escáneres**, estos sensores se utilizan para medir distancias entre el robot y los obstáculos. Estas distancias se obtienen midiendo el tiempo que tarda una onda en ser emitida, rebotada en el obstáculo y recibida por el robot. En el caso de ser usado para localización, este método se llama *scan matching*, consiste en dada unas medidas y el mapa del entorno, el algoritmo descarta las posiciones que no se pueden explicar con esas medida.

Dentro de los sensores en robótica, el sensor que destaca es la cámara, esto es debido a que es un sensor que proporciona mucha información, es barata y compacta. La ciencia

que estudia el uso de cámaras se llama visión por ordenador.

## 1.2. Visión artificial

La visión por ordenador es el área que se dedica a extraer información de las imágenes con el fin de comprender lo que en ellas está sucediendo. Originalmente se consideraba un área de la inteligencia artificial aunque ahora ya es una disciplina por sí misma. Tiene múltiples campos de aplicación.

### 1.2.1. Autolocalización visual

Uno de los campos destacados dentro de la visión por ordenador es el de la autolocalización visual, que consiste en la determinación de la posición y la orientación de la cámara utilizando para ello las imágenes recibidas de ésta, puede utilizar información adicional, llamados a *priori*. Esta información adicional puede venir de otros sensores o de cierto conocimiento de la escena. En la figura 5, podemos observar distintos grados de aprioris sobre el conocimiento de la escena. Según la cantidad de información y de los requerimientos usaremos unos métodos u otros.

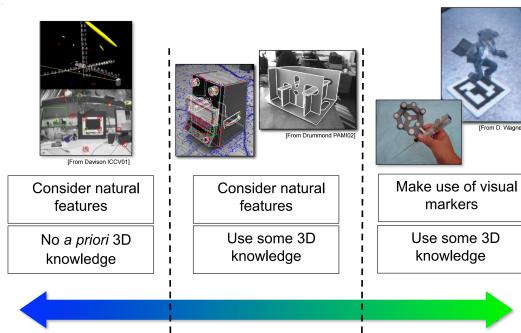


Figura 5: Distintos *priors*.

El problema de la autolocalización visual ha sido abordado por dos comunidades científicas a la vez. Por un lado, la de visión por ordenador denominó el problema como *Structure from Motion* (SfM), que consiste dado una serie de imágenes sin orden, reconstruir la escena sin necesidad de tiempo real. Por otra parte, la comunidad robótica denominó el problema como *Simultaneous localization and mapping* (SLAM) y su extensión con cámaras VisualSLAM, en este caso el problema consiste, en dado unas imágenes que llegan cada cierto intervalo de tiempo obtener una localización y una descripción de la escena en tiempo real. En la figura 6 se pueden ver los resultados obtenidos para los dos sistemas.

## 1.3. Realidad aumentada

Una de las aplicaciones fundamentales de la autolocalización visual es servir de base para sistemas con realidad aumentada. Es necesario conocer la posición de la cámara y la estructura de la escena para poder proyectar en ella objetos virtuales que se observen de manera realista.

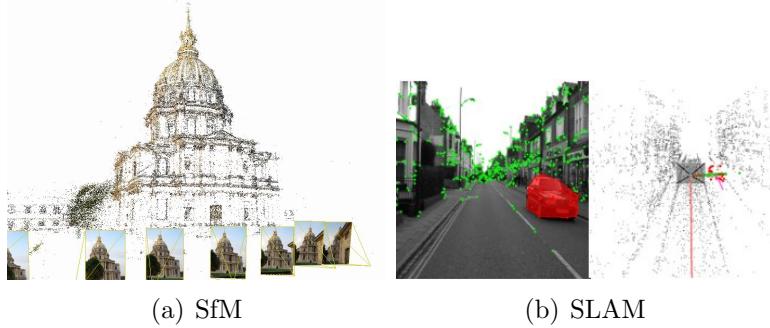


Figura 6: Sistemas autolocalización.

Hay que diferenciar entre realidad virtual y realidad aumentada. La primera consiste en una simulación artificial del entorno donde se sumerge al usuario en ella. Por otra parte, la realidad aumentada consiste en superponer objetos virtuales en entorno reales. Un algoritmo que se ideó para poder es el algoritmo de PTAM. En la figura 7 se puede observar el ejemplo.



Figura 7: Ejemplo de realidad aumentada.

## 2. Métodos de autolocalización visual

En esta sección explicaremos los diversos métodos de localización visual.

### 2.1. Balizas artificiales

La localización basadas en balizas artificiales consiste en distribuir por el entorno balizas por donde se moverá el robot, él mediante los sensores capturará la relación entre él y la baliza y se localizará. Hay dos tipos de balizas:

- Activas: Este tipo de balizas emiten alguno tipo de energía a la escena para poder ser detectadas. El uso de balizas activas disminuye la carga computacional, pero

estas deben ser alimentadas. En el campo de balizas visuales activas se hacen servir sensores basados en infrarrojos.

- Pasivas: Estas balizas no emiten ningún tipo de energía a la escena, por lo tanto no necesitan alimentación, en cambio el coste computacional de localizarse con ellas es muy alto. Las características de las balizas pasivas visuales es que posean un alto contraste con la escena para facilitar la segmentación y que se distinguen entre sí fácilmente. Hay diversos modelos como las *AprilTags*, *QR* y *Aruco*.

Dentro de las pasivas se encuentran las balizas visuales. Estas balizas no emiten energía, son más baratos y fáciles de instalar, por otro lado el procesamiento es más complejo. Las principales características de las balizas es que se puedan distinguir de la escena y poder discernir entre ellas. Unos ejemplos serían las balizas *AprilTags*, *QR* y *Aruco* ampliamente utilizadas en realidad virtual/aumentada.

Un ejemplo sería el de la figura 8 donde se han distribuido una serie de balizas en el entorno. Se sabe la relación entre las balizas y el sistema de coordenadas mundo. Un robot navegando por el entorno al observar una baliza puede establecer la relación de transformación entre el sistema de referencia de la cámara y el de la baliza resolviendo el problema de *Perspective n points*. Finalmente solo haría falta concatenar la relación Cámara-Baliza con Baliza-Mundo para localizar el robot.

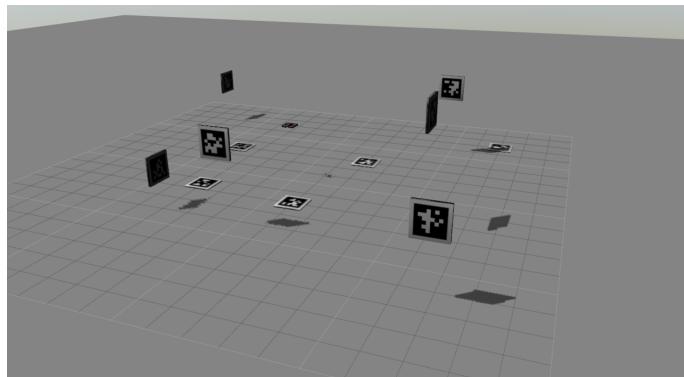


Figura 8: Escenario con balizas artificiales.

La principal ventaja de estos sistemas es su baja complejidad algorítmica y su precisión. La desventaja es que el escenario del robot está acotado hasta donde hay balizas y también la incertidumbre que se crea cuando el robot no observa una marca. Estos métodos se usan en entornos controlados y como evaluación para algoritmos de autolocalización sin marcas.

### 2.1.1. *Perspective-n-Points*

El problema *Perspective-n-Points* [PnP] o también conocido como *camera resection* consiste en dado  $n$  correspondencias de puntos 3D y sus proyecciones en el plano imagen y la matriz de intrínsecos de la cámara, encontrar la matriz de extrínsecos. O lo que es lo mismo, la relación de transformación del sistema de coordenadas de los puntos 3D con el sistema de coordenadas del centro óptico de la cámara. En la figura 9 se puede observar un esquema del planteamiento del problema.

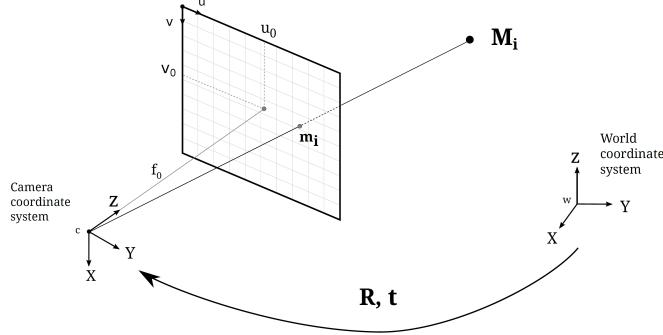


Figura 9: Esquema problema PnP.

La matriz de extrínsecos es una matriz de 6 grados de libertad, considerando que cada correspondencia de puntos ofrece 2 ecuaciones, con 3 puntos se encuentra la solución mínima.

Según [16], este problema es similar a otros problemas en visión por ordenador, dado unos datos y un modelo, estimar los parámetros del modelo. La clasificación de los métodos para resolverlo sería la siguiente

- Métodos algebraicos, en estos métodos se minimiza el error algebraico, son soluciones de forma cerrada. Unos ejemplos serían el uso de la *Direct Linear transformation*, P3P [13], representa resolución con la cantidad mínima de correspondencias y EPNP [21], método algebraico seguido de uno iterativo.
- Métodos iterativos, se define una medida de error (normalmente el error de reproyección) y se minimiza utilizando técnicas iterativas. Las funciones de coste suelen ser no lineales por lo tanto se necesitarán técnicas de optimización no lineal como descenso del gradiente, Gauss-Newton o Levenberg–Marquardt.
- Estimadores robustos. Con la presencia de ruido las técnicas anteriores darían estimaciones erróneas. Minimizando el error de reproyección los puntos que tienen otra distribución contribuyen exponencialmente a la función de coste y causando estimaciones erróneas. Para ello se utilizan unos estimadores que son robustos a la presencia de *outliers* uno de ellos es *RANSAC* y otro sería los *M-estimators*.

## 2.2. Métodos geométricos

A partir de la información proporcionada por la cámara y mediante un mapa se aplican restricciones geométricas para localizar el robot. Estas restricciones limitan las posibles posiciones del robot, las técnicas que se usan son:

- Triangulación, Cálculo de la posición utilizando el ángulo en el que se encuentran varios puntos.
- Trilateración, Cálculo de la posición utilizando la distancia que existe entre los puntos.

Un ejemplo de este método es el desarrollado por [26] para localizarse en la competición *RoboCup*. La competición *RoboCup* es un proyecto a nivel internacional para promover la inteligencia artificial, la robótica y otros campos relacionados. Una de las competiciones es

la de *RoboCupSoccer*, que consiste en programar unos robots autónomos para que jueguen un partido de fútbol. En la figura 10 se puede ver una imagen del terreno de juego.

La solución que proponen los autores es a partir de la detección de las porterías y conociendo sus dimensiones ir delimitando las posibles localizaciones del robot a partir de restricciones geométricas.

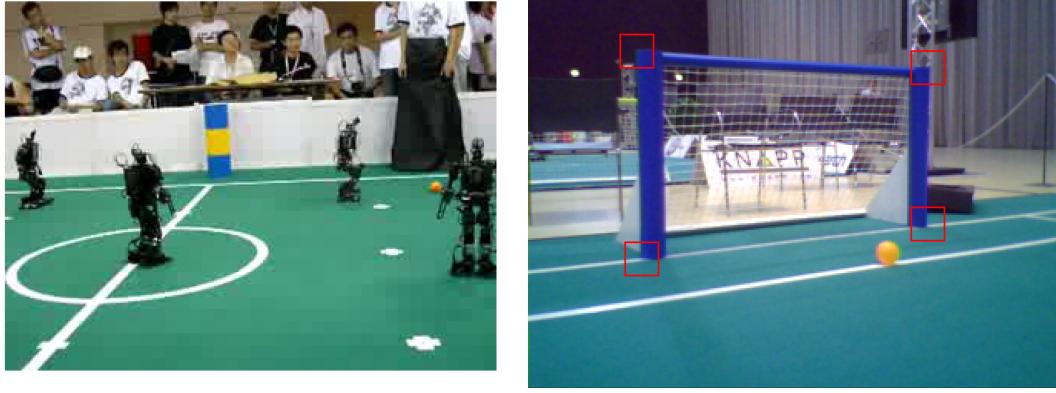


Figura 10: Imágenes *Robocup*.

La técnica que utilizan se basa en el ángulo con el que se ve los postes de la portería en la imagen. En el momento en el que vemos un poste en la imagen cuyos extremos forman un ángulo determinado, solo podemos encontrarnos en ciertos lugares del campo para ver a ese poste con ese ángulo. Si tenemos en cuenta solamente el plano que forma la cámara con los dos extremos del poste, el lugar geométrico en 2D en el que nos encontramos vendrá dado por el arco capaz y formará un arco de circunferencia que pasará por los dos extremos del poste. En la figura 11 podemos observar el arco.

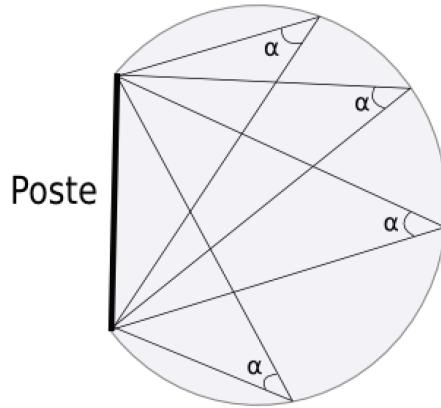
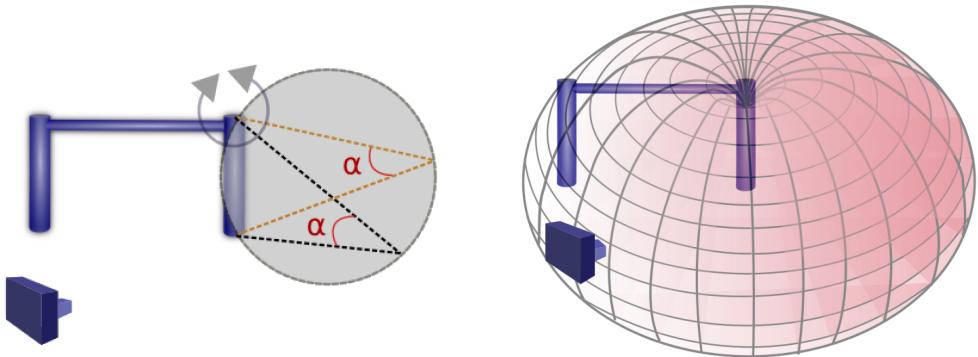


Figura 11: Distribución *M-estimator*.

Si revolucionamos la circunferencia completa respecto al eje del poste, obtendremos una figura geométrica que se conoce con el nombre de toro de revolución, que se puede observar en la figura 12, esta figura geométrica indicará todas las posibles posiciones en las que se encuentra la cámara para ver el poste con ese ángulo. Este resultado lo podemos observar en 12.



(a) Revolución de la circunferencia.

(b) Toro de revolución.

Figura 12: Relaciones geométricas.

Si hacemos lo mismo para el otro poste obtendremos otro toro de revolución y además conociendo la altura restringimos las posiciones posibles que estén en el plano paralelo al suelo y a la altura del robot obtenemos la posición del robot. En la figura 13 se puede observar el esquema de la posición.

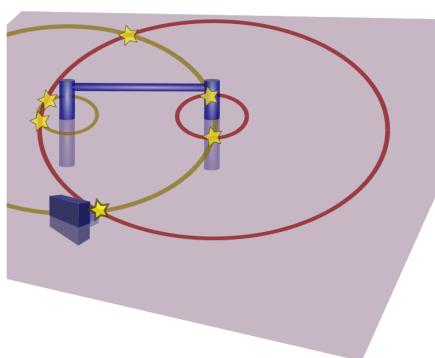


Figura 13: Esquema

### 2.3. Métodos probabilísticos

Esta técnica calcula las posiciones más probables en base a las acciones efectuadas por el robot y las lecturas sensoriales obtenidas a lo largo del tiempo. Permiten reflejar ambigüedades, similaridades y la incertidumbre en la posición.

Para calcular la distribución de probabilidad sobre la posición se sigue el siguiente esquema:

- **Predicción**, a partir de la posición anterior y de la señal de control se predice donde se encuentra el robot.
- **Actualización**, con un modelo de observación que infiere la posición del robot a partir de los datos del sensor, el robot incrementa la probabilidad de las posiciones que se explican con esas medidas.

Las funciones de probabilidad de cada parte del ciclo se fusionan durante el tiempo mediante el teorema de Bayes. En la secuencia 14 se observa la secuencia explicada.

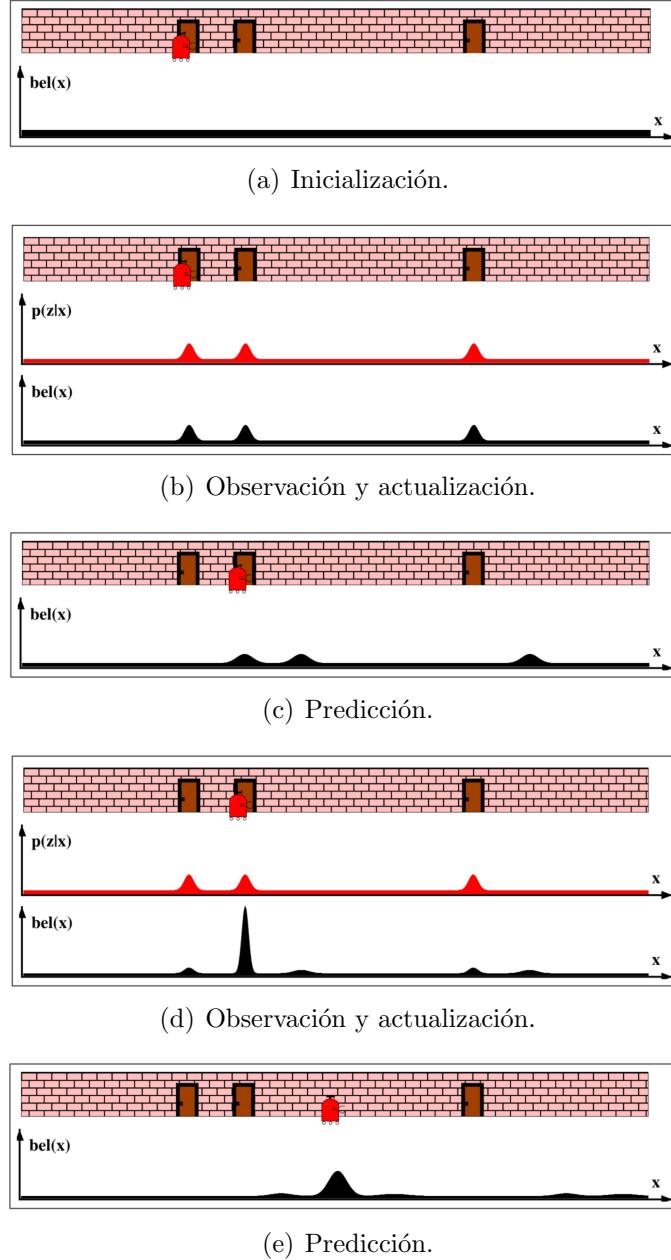


Figura 14: Ciclo probabilístico.

Los métodos probabilísticos se pueden dividir en:

- **Analíticos**, mantiene los valores de las distribuciones de probabilidad de todos los espacios de dimensiones. Computacionalmente son muy costosos, ya tienen que actualizar todas las funciones de probabilidad de manera analítica, además aumenta su complejidad cuando utilizan distribuciones de probabilidad que no siguen distribuciones gaussianas unimodales.
- **Discretizados**, dividen el espacio de posibles soluciones en zonas discretas con un valor de probabilidad propio que se va actualizando conforme con el ciclo predicción-actualización. La solución se encontrará en la posición con mayor valor de

probabilidad. Por su modo de funcionamiento, este tipo de técnicas suelen ser costosas en cómputo y memoria, además presentan problemas de escalabilidad, ya que el número de zonas crece de forma exponencial a medida que se aumenta el área del problema.

- **Muestreados** consideran a la vez parte del espacio de soluciones, muestreando posiciones de interés y actualizando la localización de estas muestras en función de la probabilidad. Entre estas técnicas destacan las técnicas de Monte Carlo, concretamente el algoritmo de filtro de partículas. En la figura 15 se puede observar una secuencia del uso de esta técnica.

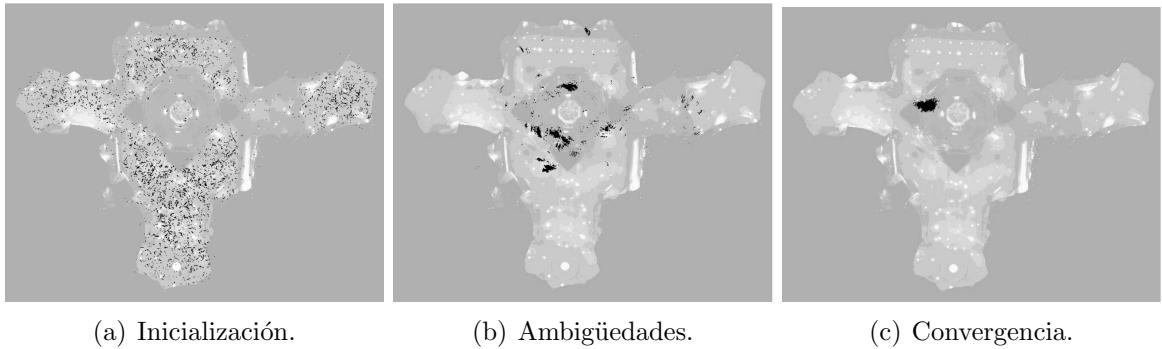


Figura 15: Filtro de partículas.

### 2.3.1. Métodos evolutivos

Los algoritmos evolutivos son un subconjunto de los denominados algoritmos metaheurísticos. Este tipo de algoritmos se utilizan cuando no existe un algoritmo de optimización exacta y específica para el problema o cuando no se puede implementar por restricciones técnicas o del propio problema.

Los algoritmo evolutivos son algoritmos de optimización que se inspiran en mecanismos de la evolución biológica como pueden ser reproducción, mutación, recombinación genética y la selección natural, con el fin de encontrar la mejor solución al problema.

El espacio de soluciones del problema se representa como un conjunto de individuos con una serie de propiedades que los distinguen entre ellos. Este conjunto de individuos o población parte de un estado inicial y con el paso del tiempo va evolucionando, es decir, modificando sus propiedades y mediante la creación de individuos más saludables y la extinción de los menos saludables. Conforme se produce esta evolución, los individuos se acercan paulatinamente a un estado donde alcanzarían la salud máxima que representa una solución válida dentro del problema.

## 2.4. VisualSlam

En los métodos anteriores nos hemos apoyado en que conocíamos cierta información sobre el mundo, como posición de ciertos elementos o el mapa entero, que nos permitían obtener la posición del robot. Así con una cámara y un mapa obteníamos información de posición.

Para dotar de un comportamiento autónomo real al robot este no debe tener información del mundo. Los métodos de *Simultaneous localization and mapping* conocido por sus siglas SLAM, consiste en colocar al robot en una posición desconocida en un entorno y que el robot incrementalmente construya un mapa del entorno mientras simultáneamente determine su localización en el mapa [7].

La comunidad robótica ha solucionado estos problemas con varios sensores y técnicas pero las solucionadas con cámaras se llaman visual-Slam.

Así el objetivo de SLAM es obtener una estimación global y consistente del camino del robot (*tracking*). Esto implica crear un mapa (*mapping*) y hacer un seguimiento de él, porque el robot necesita darse cuenta que revisita una zona (*loop closure*). Cuando se detecta un *Loop closure*, esta información se usa para reducir la deriva en el mapa y en la trayectoria.

Hay que diferenciar visualSlam de odometría visual, esta consiste en estimar la trayectoria de la cámara pose a pose, estos sistemas a veces construyen un mapa, pero solo para apoyar la localización no para obtener una representación del entorno como hacen los sistemas de visualSlam [28].

La odometría visual puede verse como un bloque de visualSlam, el encargado de hacer el *tracking* de la cámara. Este *Tracking* consiste en estimar la transformada de sólido rígido entre frames y concatenar estas transformadas para obtener el camino global. Para ello hace uso de las características dispersas de las imágenes o de la imagen entera.

El *mapping* consiste en obtener nuevos puntos del mapa y refinar los anteriores, para ello se hace uso de la geometría epipolar. Algunas trabajos añaden un marco de trabajo probabilístico para conseguir robustez.

El robot al obtener la pose incrementalmente va acumulando errores. Para reducir este error se utilizan técnicas para optimizar la pose. Algunas de ellas son:

- **Bundle Adjustment**, consiste en optimizar la posición de la cámara y de los puntos 3D.
- **Pose-graph**, para añadir restricciones, la transformaciones se igualan con las transformaciones no adyacentes.

Para obtener eficiencia se suele utilizar solo los últimos  $m$  frames y se añade el prefijo *local* o *windowed* a las técnicas comentadas anteriormente. En la figura 16 se puede observar un gráfico de estas técnicas.

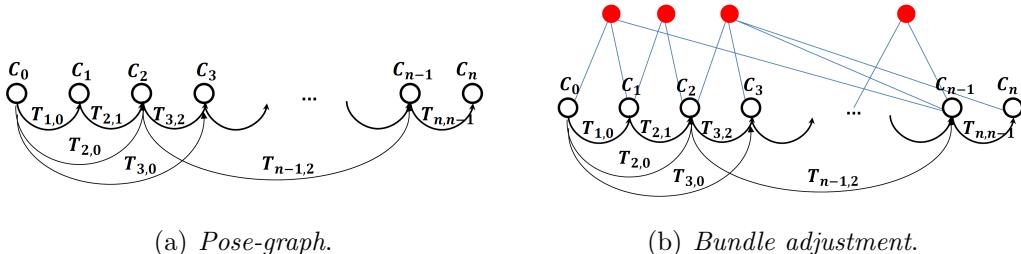


Figura 16: Métodos de *Loop closure*.

Para añadir más restricciones y aumentar la precisión se añade un mecanismo de *Loop closure*, que consiste en reobservar una zona de la escena que no se ha visto durante mucho tiempo. Para detectar estas zonas visitadas se han empleado diversos mecanismos como, descriptores de imágenes o sistemas de más alto nivel como *Bag of words*. Una vez que se ha detectado un *loop* se añade otra restricción al grafo.

### 3. Algoritmos de visualSlam

En esta sección se explicarán los algoritmos de viuslSlam más relevantes.

#### 3.1. MonoSlam

Los autores presentan en [6] un algoritmo de SLAM, que permite recuperar la trayectoria 3D y crear mapas a partir de una sola cámara. Este trabajo tiene relevancia al ser el primer algoritmo de SLAM puramente monocular. Consiguiendo una solución en tiempo real y robusta. Hasta el momento la comunidad robótica se había centrado en resolver los sistemas SLAM con otros sensores como láser [14], cámaras estéreo [5] o odometría visual [15].

El algoritmo propuesto, a partir de 4 puntos conocidos en el espacio estima la posición y orientación de la cámara y además va estimando una representación del entorno. Para ello hacen uso del filtro extendido de Kalman. Este algoritmo se centra más en la localización que en el mapeo de la escena, así se construye un mapa disperso que apoye la localización.

##### 3.1.1. Funcionamiento

A continuación se explicará el funcionamiento del algoritmo. El algoritmo tiene una representación del sistema con un vector de estado  $\hat{x}$  y una matriz de covarianza  $P$ , este vector concatena la información de la cámara y de las posiciones de los puntos 3D con una representación probabilística de una gaussiana multivariante en el espacio de dimensión igual al tamaño del vector de estado y cuya desviación se representa con  $P$ .

$$\hat{x} = (\hat{x}_v \ y_1 \dots y_n)^T$$

Donde  $\hat{x}_v$  y  $y_n$  representan la información de la cámara y de los puntos 3D respectivamente. Por parte de la información de la cámara el vector encapsula la posición ( $r^W$ ), orientación ( $q^{WR}$ ), la velocidad lineal ( $v^W$ ) y angular ( $\omega^R$ ).

$$\hat{x}_v = (r^W \ q^{WR} \ v^W \ \omega^R)^T$$

Finalmente el vector  $y_n$  representa las coordenadas 3D de un punto. El papel del mapa es principalmente para permitir la localización en tiempo real más que para obtener una descripción completa de la escena. Para mantener la ejecución en tiempo real el vector de estado no puede tener más de 100 puntos 3D.

Como hemos dicho anteriormente, el algoritmo necesita 4 puntos conocidos en el espacio para inicializarse, estos puntos se pueden observar en la figura 17.



Figura 17: Patrón inicialización conocido.

A medida que la cámara se mueve y toma medidas visuales ( estas medidas consisten en puntos obtenidos por el detector FAST ) el vector de estado y la matriz de covarianzas se actualizan con las ecuaciones de EKF. El sistema al tener componentes no lineales, se linealizan con expansiones Taylor, estas quedan justificadas debido al alto tiempo de ejecución. Estas ecuaciones constan de dos partes: el paso de predicción y el paso de actualización.

En el paso de predicción, el robot predice la posición a partir de la posición anterior y un modelo de movimiento. Este modelo de movimiento se basa en un modelo de movimiento de velocidad lineal y angular constante, con aceleraciones indeterminadas de tipo gaussiano. Se puede observar en las ecuaciones siguientes:

$$f_v = \begin{pmatrix} r_{nueva}^W \\ q_{WR}^{nueva} \\ v_{nueva}^W \\ \omega_{nueva}^R \end{pmatrix} = \begin{pmatrix} r^W + (v^W + V^W)\Delta t \\ q^W \times q((\omega^R + \Omega^R)\Delta t) \\ v^W + V^W \\ \omega^R + \Omega^R \end{pmatrix}$$

Además el proceso de predicción va acompañado de un incremento en la incertidumbre del estado, esta incertidumbre se modela con  $Q_v$  y se rige por:

$$Q_v = \frac{\partial F_v}{\partial n} P_n \frac{\partial F_v}{\partial n}^T$$

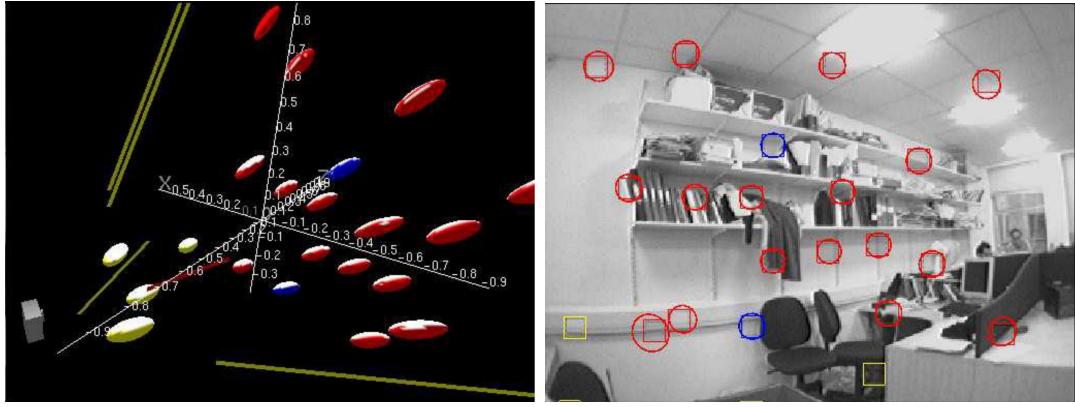
En el paso de actualización, basados en la predicción de la nueva posición de la cámara, predecimos que puntos serán visibles y donde serán visibles en la imagen. Para ello usamos un modelo de observación  $h$ , que a partir de la posición del punto  $y_i$  y la predicción de la posición de la cámara  $\hat{x}_t$  nos dice donde se encontrará el punto en la imagen  $z_i$ , este modelo también nos dice que incertidumbre tenemos sobre esta estimación, esta información está codificada en la matriz inovación  $\Sigma_{IN}$ .

$$z_i = h_t(\hat{x}_t, y_i) \quad \Sigma_{IN} = H \hat{P}_t + R$$

Así usamos esta incertidumbre con forma de elipsoide y la proyectamos en la imagen, proyectará con forma de elipse y será nuestra región de búsqueda. Este paso se observa en la figura 18.

Una vez que hemos predecido las observaciones, hacemos las observaciones y actualizamos el estado según las ecuaciones de EKF:

$$x_t = \hat{x}_t + K_t(z_{0:n-1} - h_{0:n-1}(\hat{x}_t, y_{0:n-1}))$$



(a) Representación 3D del espacio.

(b) Proyecciones en el plano imagen.

Figura 18: Puntos en monoSlam.

$$P_t = \hat{P}_t - K_t \Sigma_{IN} K_t^T$$

Donde  $K_t$  es la ganancia de Kalman  $K_t = \hat{P}_t H \Sigma_{IN}^{-1}$ .

A continuación se explicará como se inicializan nuevos puntos. Con un sistema monocular no podemos invertir la matriz de proyección y a partir de los puntos en el plano imagen obtener la posición 3D del punto desde que la profundidad es desconocida. Para estimar la profundidad necesitaremos mover la cámara y hacer varias medidas desde diferentes puntos de vista. La parametrización que hacen los autores es la siguiente:

$$y_{pi} = \begin{pmatrix} r_i^W \\ h_i^W \end{pmatrix}$$

Donde  $r_i^W$  es la posición del punto final y  $h_i^W$  es un vector unitario describiendo su dirección.

El método que proponen los autores consiste en cada vez que aparezca un punto nuevo, se reparten unas partículas uniformemente distribuidas sobre el rayo de retroproyección, estas hipótesis de posiciones se proyectan en las vistas sucesivas y las que coincidan incrementan su densidad. Cuando la densidad es inferior a un cierto umbral, el punto se añade al vector de estado. La secuencia de este proceso lo podemos observar en la figura 19.

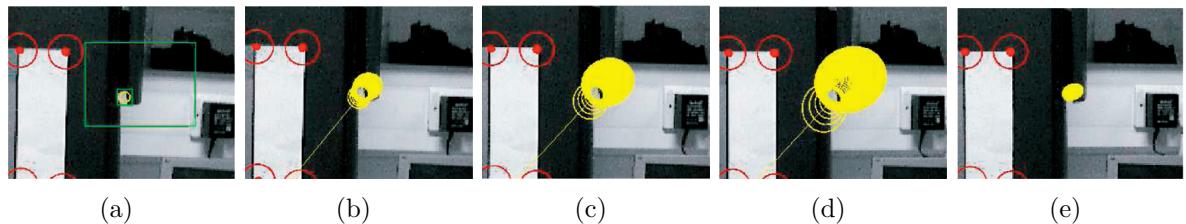


Figura 19: Secuencia estimación profundidad.

En una modificación [2] posterior, los autores proponen parametrizar los puntos observados de otra manera. El modo anterior solo podía representar puntos entre 0.5 y 5 metros que es suficiente para entornos interiores, pero en exteriores donde algunos puntos se

inicializan en el infinito esta parametrización falla. Por eso los autores proponen lo que llaman la parametrización inversa, así un punto se representaría de la siguiente forma:

$$y_i = (x_i \ y_i \ z_i \ \theta_i \ \phi_i \ \rho_i)^T$$

Con las coordenadas  $x_i \ y_i \ z_i$  representan la posición de la cámara donde se ha visto el punto por primera vez,  $\rho_i$  la profundidad del punto,  $\phi_i$  y  $\theta_i$  los ángulos que forman las proyecciones con el centro óptico de la cámara. Con esta parametrización se permite representar puntos en el infinito y además se introduce desde el principio en el vector de estado, aunque su profundidad tenga mucha incertidumbre, no contribuye a disminuir la incertidumbre de la posición pero ayuda a disminuir la de la orientación.

### 3.1.2. Resultados

Para comprobar la precisión del sistema se compara con las medidas de una IMU. Se mueve la cámara a los cuatro vértices de un rectángulo imaginario. En la tabla 1 se observan los resultados proporcionados por la IMU y el sistema, podemos observar que la precisión es muy alta, solo hay un error de 1-2 cm.

GT			Estimado		
x	y	z	x	y	z
0.00	0.00	-0.62	0.00±0.01	0.01±0.01	0.64±0.01
-1.00	0.00	-0.62	-0.93±0.03	0.06±0.02	0.63±0.02
-1.00	0.50	-0.62	-0.98±0.03	0.46±0.02	0.66±0.02
0.00	0.50	-0.62	0.01±0.01	0.47±0.02	0.64±0.02

Cuadro 1: Evaluación del algoritmo.

Otra prueba consiste en incorporar el sistema a un robot humanoide, el HRP-2. Se programa al robot para que realice una circunferencia, en la figura 20 se observa que la estimación del sistema consigue un *Loop closure*.

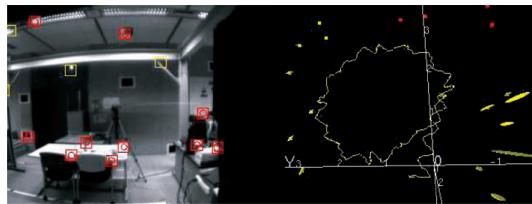


Figura 20: Comparación con la IMU.

También se usa el algoritmo como aplicación de realidad virtual, aunque la representación de la escena no es muy densa, se consigue proyectar y mantener correctamente las figuras durante el movimiento de la cámara. En la figura 21 se puede observar una imagen con el patrón proyectado.

En la tabla 2 se puede observar los tiempos desglosados del algoritmo, se observa que se consigue el tiempo real.

Las innovaciones de este algoritmo fueron la implementación de un sistema de SLAM monocular en tiempo real. Las desventajas de monoSlam es que ha medida que incorpora



Figura 21: Proyección de una estructura en la imagen.

Carga y administración de la imagen	2 ms
Búsqueda en la imagen	3 ms
Actualización filtro kalman	5 ms
Inicialización de puntos	4 ms
Rendering de gráficos	5 ms
Total	19 ms

Cuadro 2: Desglose tiempos monoslam.

puntos al vector de estado la complejidad se incrementa cuadráticamente, con un límite de procesamiento de 100 puntos. Con tan poca cantidad de puntos, solo puede usarse en interiores no muy grandes. Además debido a usar técnicas *filtering*, el sistema no permite la relocalización.

### 3.2. PTAM

PTAM son las siglas de *Parallel tracking and mapping* [20], es un algoritmo de slam monocular desarrollado para aplicaciones de realidad aumentada, aun así se convirtió en el estándar de los algoritmos de visualSlam monoculares.

El algoritmo anterior MonoSlam, debido a que usa técnicas *filtering* tenía el problema que su tiempo de ejecución aumentaba a medida que se incorporaban nuevos puntos al vector de estado. Esto es debido a que en cada iteración del algoritmo se actualiza tanto la posición de cada punto del mapa como la posición actual de la cámara.

PTAM parte de la idea de que solo es necesario funcionar en tiempo real en la parte de *tracking*, mientras que el *mapping* no tiene porqué realizarse en cada iteración. Así, el *tracking* y el *mapping* funcionan en los hilos separados de forma asíncrona.

El algoritmo de PTAM hace uso de *keyframes*, fotogramas clave que se utilizan para explicar la posición en el mapa como para crear el mapa de puntos.

### 3.2.1. Funcionamiento

En esta sección se explicará el funcionamiento del algoritmo. Primero se explicará como se realiza el *tracking* y luego el *mapping*.

El *tracking* recibe imágenes de la cámara y mantiene una estimación en tiempo real de la posición relativa al mapa construido. En cada frame se realizan las siguientes operaciones.

Se calcula la posición a *priori* a partir de un modelo de movimiento, que consiste en un modelo de velocidad decayente, seguidamente los puntos del mapa se proyectan en la imagen según esa posición. Para realizar la asociación entre punto en el frame actual y anterior se hace una búsqueda con enfoque piramidal. Dada un conjunto de asociaciones entre puntos (alrededor de 50 puntos) se calcula la minimizando el error de retroproyección.

Para mejorar la estimación de la posición de la cámara se realiza una nueva estimación de la posición pero con un conjunto de 1000 puntos en el nivel más bajo de la pirámide.

Finalmente, debido a oclusiones y al fenómeno *motion blur* el *tracking* puede fallar por este motivo el sistema estima una medida de calidad del de este. Esta medida consiste en una fracción respecto al total de punto que se han seguido correctamente. Y este valor es menor a un cierto umbral, se sigue con el funcionamiento normal pero no se envían *keyframes* al mapa. Si el valor es incluso menor durante varios frames se considera que se ha perdido y se tratará de hacer una relocalización iniciando la secuencia de *tracking* otra vez.

A continuación se explicará el *mapping*. Para inicializar el mapa se requiere la cooperación del usuario, este coloca la cámara en la escena que se va a mapear y pulsa un botón. En este punto se selecciona el primer *keyframe*, el usuario realiza un movimiento de traslación suave y vuelve a pulsar un botón. Con esta segunda pulsación se extrae el segundo *keyframe*, mediante las correspondencia de puntos se encuentra la matriz esencial y se triangulan los puntos.

Así como la cámara se aleja de la posición inicial se van añadiendo nuevos *keyframes*. A partir de la información de la pose proporcionada por el *tracking*, se refina la posición de los puntos que se ven mediante la reproyección de estos. Los puntos 3D nuevos se van añadiendo mediante triangulación con *keyframe* anterior.

Finalmente se realiza un *Bundle adjustment*, con esta técnica se refina la posición de los puntos 3D y la posición de la cámara.

### 3.2.2. Resultados

A continuación se discutirá varios aspectos del algoritmo. En la tabla 3 se puede observar el desglose del tiempos del *tracking*, para unos 4000 puntos. Se puede observar que se obtiene una ejecución en tiempo real.

En la figura 22 se compara con MonoSlam, la desviación sobre la *ground truth* de MonoSlam es de 135mm y de PTAM de 6mm.

PTAM se ideó para aplicaciones de realidad aumentada, los autores muestran su uso dentro de un juego de realidad aumentada. En la figura 23 se puede ver una captura de este vídeo, donde se ha obtenido el mayor plano y se ha proyectado los gráficos.

Preparación del keyframe	2.2ms
Proyección de características	3.5ms
Búsqueda de bloques	9.8ms
Actualización de pose iterativa	3.7ms
Total	19.2ms

Cuadro 3: Tabla de tiempos.

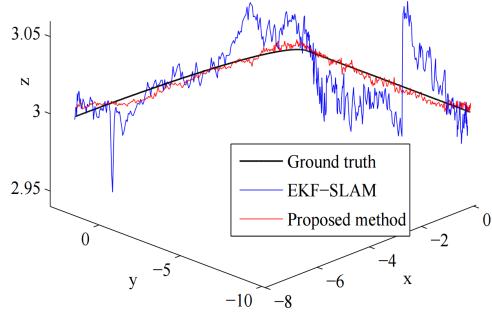


Figura 22: Comparación trayectoria con MonoSlam.



Figura 23: Ejemplo de realidad aumentada.

### 3.3. Basados en métodos directos

Los algoritmos descritos anteriormente, abstraen las imágenes en un conjunto de características dispersas. El flujo de funcionamiento de estos algoritmos consiste en detectar y extraer características, hacer el matching en los siguientes frames, estimar el movimiento y la reconstrucción a partir de la correspondencia entre características. En contra de estos métodos, están los métodos directos, que usan todos los píxeles de la imagen para estimar el movimiento de la cámara y la reconstrucción. En la figura 24 se puede observar una esquema de la diferencia entre métodos.

Estos métodos se basan en la ecuación de conservación de la intensidad [17].

$$I_1(x) = I_2(\pi(g_\xi(z * x)))$$

Esta ecuación dice que el valor de intensidad de un punto será el mismo para la imagen transformada. Conociendo la profundidad  $z$  se puede calcular la transformación de mo-

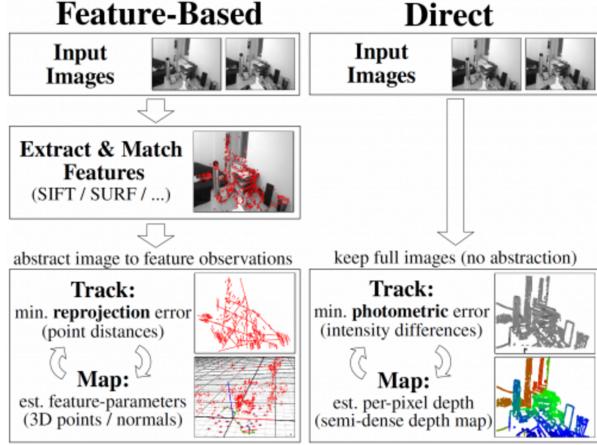


Figura 24: Esquema diferencia de métodos.

vimiento  $g_\xi$  o conociendo el movimiento inferir la profundidad. En la figura 25 podemos observar una representación gráfica.

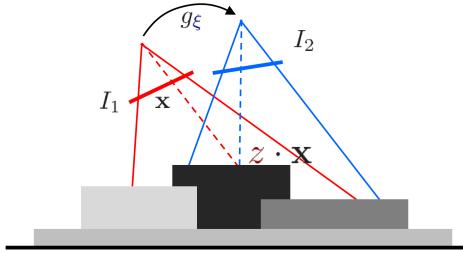


Figura 25: Esquema conservación de la intensidad [32].

Las ventajas frente a los métodos basados en características son las siguientes [4]:

- Son más robustos. Estos métodos utilizan todos los píxeles de la imagen como restricción para las ecuaciones, en cambio los métodos basados en características solo un conjunto disperso de puntos (monoSlam 100 puntos y PTAM 1000 puntos). Además son robustos en escenas con poca textura.
- Reconstrucciones más densas. Estos métodos reconstruyen la escena densamente, no solo un conjunto disperso de puntos.
- Rápidos. Estos métodos son típicamente más rápidos al eliminar los pasos de extracción de características y matching.

Estos métodos se han podido llevar a cabo por el incremento computacional de los últimos años.

Ha habido mucho interés recientemente en estos métodos, algunos de los trabajos son:

- En [24] los autores proponen un algoritmo que calcula la geometría de la escena y el movimiento de la cámara mediante métodos directos. Es un algoritmo de SLAM aunque está limitado a entornos pequeños ya que no lleva a cabo un *Loop closure*

para escalar a escenas más grandes. La resolución de este planteamiento es posible gracias al uso de GPU.

- En [12], los autores proponen un algoritmo basado en métodos directos ( aunque también usan características para elaborar el mapa ). Este método es de odometría visual no de visualSlam, ya que el mapa que construyen no es muy consistente. Este algoritmo se explicará en este trabajo.
- Finalmente, otro de los algoritmos basados en métodos directos es [10], los autores presentan un algoritmo de SLAM basado en métodos semidensos, ya que solo extraen la geometría de zonas donde tienen gradiente de la imagen. Además puede aplicarse a entornos grandes ya que realiza un *loop closure*. Este algoritmo se explicará en este trabajo.

A continuación se explicará el algoritmo de SVO

### 3.4. SVO

Los autores proponen en [12] un algoritmo de odometría visual. Las siglas del nombre vienen de *Semi-direct visual odometry*, es un algoritmo que usa métodos directos para estimar el movimiento de la cámara y correspondencias de características para inicializar la profundidad de los puntos. Separan en dos hilos de ejecución el *tracking* y el *mapping*, y realizan un *bundle adjustment* entre *keyframes*. El algoritmo llega a ejecutarse a 300 frames por segundo en un ordenador portátil y 55 frames por segundo en un sistema embebido.

Las contribuciones de este algoritmo son utilizar una aproximación semi directa, eliminando la fase de extracción y *matching* y establecer un marco probabilístico para estimar la profundidad de los puntos.

A continuación se explicará el funcionamiento del algoritmo.

#### 3.4.1. Funcionamiento

Antes de describir el algoritmo se explicará la notación. La imagen adquirida en el tiempo  $k$  se denota como:  $\Omega \subset \mathbb{R}^2$ , donde  $\Omega$  es el dominio de la imagen. Los puntos 3D  $p = (x, y, z)^T \in S$  en la escena visible  $S \subset \mathbb{R}^3$  mapea a las coordenadas de imagen  $u = (u, v)^T$  a través del modelo de proyección de la cámara  $\pi : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ .

La posición y orientación de la cámara se modela como una transformación de sólido rígido  $T_k$  en el álgebra de Lie.

En la figura 26, se puede observar un diagrama de funcionamiento del algoritmo. Como hemos dicho, separa *tracking* y *mapping* en dos hilos de ejecución, para el *tracking* se necesita conseguir tiempo real mientras que para el *mapping* las condiciones de tiempo real se relajan.

A continuación se explicará el *tracking*. El algoritmo calcula una suposición inicial sobre el movimiento relativo de la cámara por la correspondencia de métodos directos y finaliza con un refinamiento con métodos basados en características.

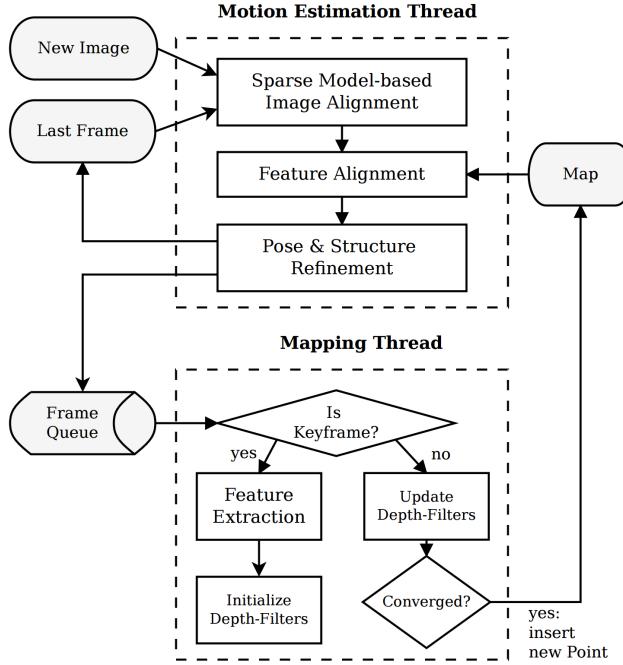


Figura 26: Diagrama funcionamiento SVO.

El residuo de la intensidad  $\partial I$  se define por la diferencia fotométrica entre píxeles observando el mismo punto 3D. Se puede calcular por retroproyectando un punto  $u$  2D del frame anterior  $I_{k,k-1}$  y seguidamente proyectándolo en el frame actual:

$$\partial I(T, u) = I_k(\pi(T \cdot \pi^{-1}(u, d_u))) - I_{k-1}(u) \quad \forall u \in \mathbb{R}$$

En contraste con otros algoritmos basados en métodos directos solo sabemos la profundidad de un conjunto disperso de puntos  $u_i$ , así que denotamos un parches de 4 x 4 píxeles donde asumimos que comparten la misma profundidad. Así buscamos la transformación de movimiento que minimiza el error fotométrico entre todos los parches:

$$T_{k,k-1} = \operatorname{argmin}_{T_{k,k-1}} \frac{1}{2} \sum \|\partial I(T_{k,k-1}, u_i)\|^2$$

Dado que la ecuación no es lineal, se soluciona mediante el algoritmo de Gauss-Newton. No se deforman los parches ya que el movimiento es muy pequeño. Este proceso se puede observar en la figura 27.

El paso anterior ha alineado el frame actual con el frame anterior, a partir de la retroproyección, la posición relativa encontrada implícitamente define una posición de las características 3D en el plano imagen. Debido a las imprecisiones, la estimación se puede mejorar. Así todos los puntos 3D del mapa se proyectan en el frame y se minimiza el error de proyección.

$$u'_i = \operatorname{argmin}_{u'_i} \frac{1}{2} \|I_k(u'_i) - A_i \cdot I_r(u_i)\|^2$$

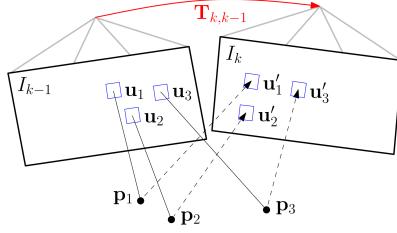


Figura 27: Alineamiento disperso de parches.

En este paso aplicamos una deformación  $A_i$  ya que la distancia es mayor. En la figura 28 se puede observar este paso.

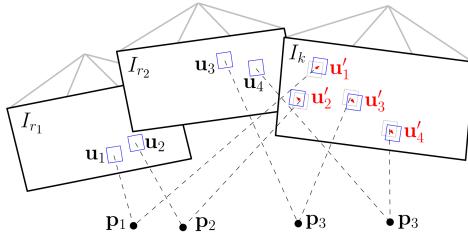


Figura 28: Minimización del error de retroproyección.

En los pasos anteriores hemos establecido correspondencia de características con una precisión de subpíxeles, pero hemos generado un error de reproyección  $\|\partial u_i\|$  de 0.3 píxeles de media, se puede observar en la figura 29. Así que como paso final de esta sección optimizamos la posición de la cámara para minimizar estos residuos.

$$T_{k,w} = \operatorname{argmin}_{T_{k,k-1}} \frac{1}{2} \sum \|u_i - \pi(T_{k,w}, p_i)\|^2$$

Esta optimización es conocida como *motion only* de *Bundle adjustment*

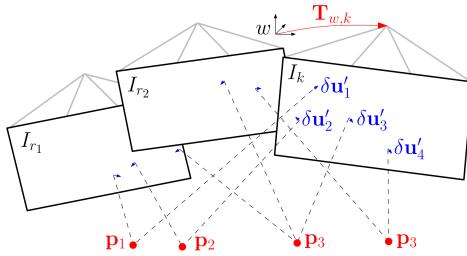


Figura 29: Minimización del error de retroproyección.

El primero y último paso pueden ser redundantes ya que los dos optimizan la posición de la cámara. Lo que típicamente se hace es hacer el *tracking* mediante el algoritmo de Lukas Kanade, pero este enfoque es más lento ya que se tiene que llevar a cabo una implementación piramidal y además no es muy preciso.

A continuación se explicará el *mapping*. Dada una imagen y su pose  $I_k, T_{k,w}$ , el hilo de *mapping* estima la profundidad de las características 2D. La profundidad de cada

característica se modela como una distribución de probabilidad, con cada observación se busca en su línea epipolar y por triangulación se va actualizando la distribución de manera recursiva mediante el teorema de Bayes. Este paso se puede observar en la figura 30. La profundidad se parametriza inversamente para poder representar el infinito.

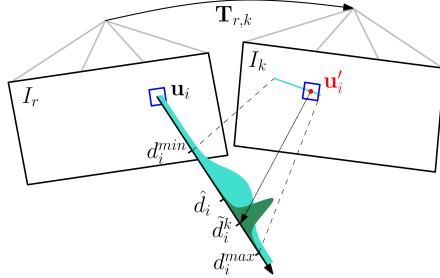


Figura 30: Estimación probabilística de la profundidad.

En la secuencia 31, se puede observar convergencia de la distribución de probabilidad.

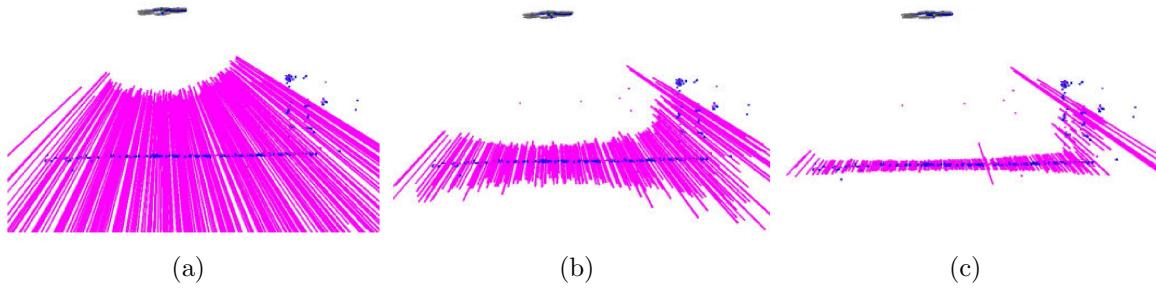


Figura 31: Secuencia estimación profundidad.

### 3.4.2. Resultados

El algoritmo tiene dos reglajes de funcionamiento, *fast* donde se prioriza la rapidez y *accuracy*, donde se prioriza la precisión. Las diferencias se pueden observar en la tabla 4.

	<b>fast</b>	<b>accurate</b>
<i>Cantidad máxima de características por imágenes</i>	120	200
<i>Cantidad máxima de keyframes</i>	10	50
<i>Local Bundle adjustment</i>	no	yes

Cuadro 4: Diferencia de reglajes SVO.

El algoritmo se compara con PTAM [20] que hasta la fecha de publicación del artículo es el mejor algoritmo para sistemas monoculares (concretamente [35], la versión modificada para escenarios grandes).

Los dos algoritmos se comparan con una secuencia obtenido con sistemas de *motion capture*. En la figura 32 se puede observar que SVO es más preciso al estimar la trayectoria y en la tabla 5 se puede observar el error de posición y rotación. SVO es más preciso obteniendo la posición y similar a PTAM en la rotación.

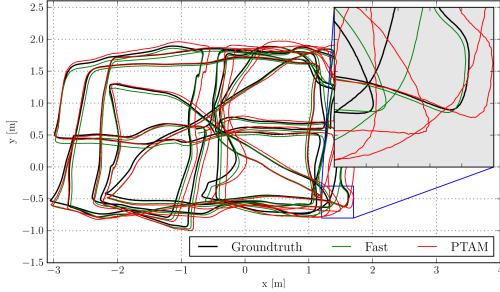


Figura 32: Comparación recorrido.

	<b>Pos-RMSE</b> [m/s]	<b>Pos-Median</b> [m/s]	<b>Rot-RMSE</b> [deg/s]	<b>Rot-Median</b> [deg/s]
<i>svo_fast</i>	0.0059	0.0047	0.4295	0.3686
<i>svo_accurate</i>	0.0051	0.0038	0.4519	0.3858
<i>PTAM</i>	0.0164	0.0142	0.4585	0.3808

Cuadro 5: Tabla errores de la trayectoria 32.

Otro aspecto de la evaluación es el tiempo de ejecución, en la tabla 6 se puede observar los máximos *frames per second* que pueden llegar los algoritmos para distintas plataformas. SVO es claramente superior a PTAM. Esta diferencia se debe a que SVO no extrae características de las imágenes durante el *tracking*, de media tarda 3,04ms (el desglose de tiempos se puede observar en 33) mientras que PTAM tarda 7ms. Además durante la etapa de *mapping*, SVO extrae menos características que PTAM, ya que el uso de filtros de profundidad hace sean más fiables.

	<b>Laptop</b>	<b>Embedded</b>
<i>SVO</i>	300 fps	55 fps
<i>PTAM</i>	91 fps	27 fps

Cuadro 6: Rango de FPS de los algoritmos.

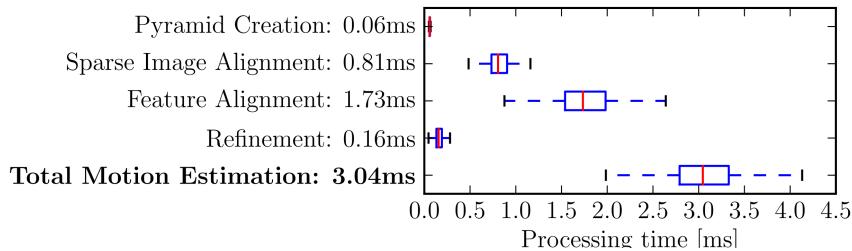


Figura 33: Desglose de tiempos.

La rapidez y precisión de SVO es debido al uso de filtros de profundidad, que producen un mínimo de *outliers*, una comparación con PTAM se puede observar en la figura 34. Además es robusto en escenas con poca textura repetida como las de la figura 35, donde los métodos basados en características fallarían.

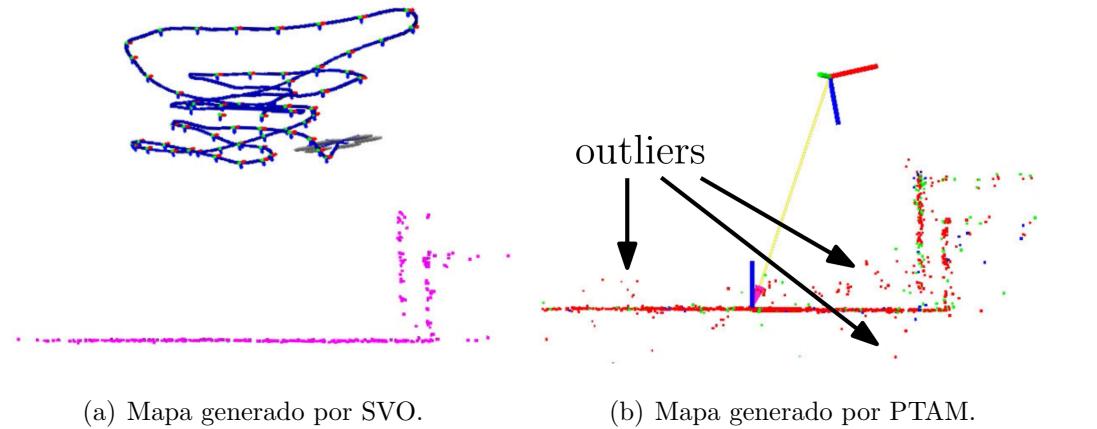


Figura 34: Comparación de puntos.

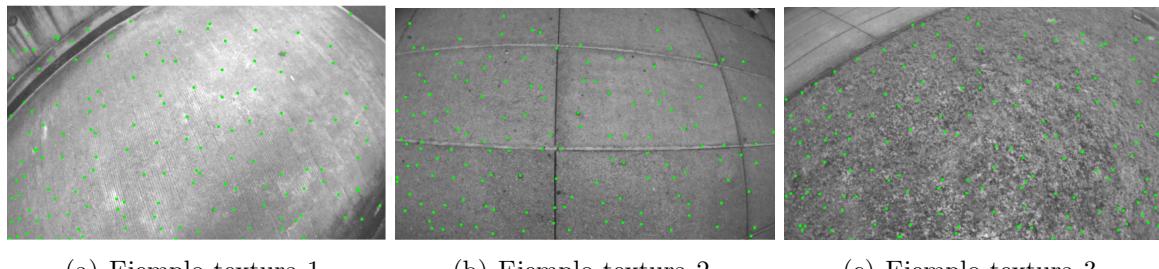


Figura 35: Diferentes texturas.

### 3.5. LSD-Slam

Como hemos descrito anteriormente, los autores proponen en [10] un algoritmo de visualSlam monocular basado en métodos directos. Además implementa un *Loop closure* lo que le hace robusto a grandes desplazamientos. Las siglas vienen de *Large scale direct SLAM*. El sistema resultante se ejecuta en una CPU consiguiendo tiempo real.

Las contribuciones de este algoritmo son las siguientes:

- El algoritmo estima un mapa de profundidad semi denso, al que asocia una profundidad inversa. Es semi-denso ya que solo reconstruye los píxeles con suficiente gradiente.
- No solo construye un mapa de profundidad, además le asocia una incertidumbre, que se propaga y actualiza.
- En los sistemas monoculares la reconstrucción y el movimiento están en función de la escala, en este método se parametriza la escala para poder extraerla.
- Consigue una consistencia global gracias a la implementación de un *Loop closure*.

A continuación se explicará en detalle el algoritmo.

### 3.5.1. Funcionamiento

La visión en conjunto del algoritmo se puede observar en la figura 36, los autores dividen el algoritmo en:

- *Tracking*, este componente hace el seguimiento de los nuevos frames, estima la transformación de sólido rígido entre *keyframes*.
- *Depth map estimation*, este componente usa el seguimiento del componente anterior para refinar o remplazar el *keyframes*. El mapa de profundidad se refina mediante comparación estéreo entre *keyframes*.
- *Map optimization*, cada vez que se actualiza el *keyframes* actual, el anterior se incorpora al conjunto de *keyframes* pasados, en este conjunto se aplica un *loop closure*.

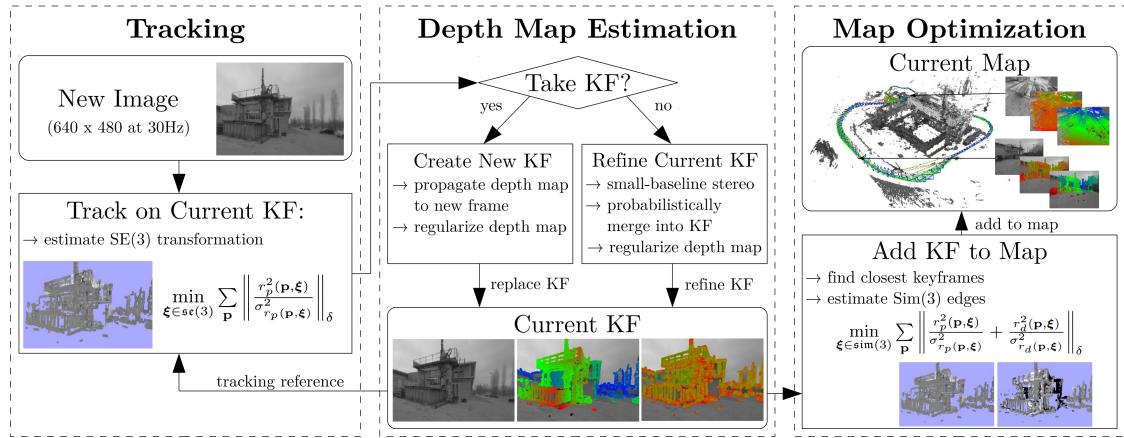


Figura 36: Esquema de la estructura del algoritmo.

Antes de continuar con la explicación se introducirán unas breves notas sobre la notación. Las imágenes como  $I: \Omega \rightarrow \mathbb{R}$ , la profundidad inversa de cada píxel como  $D, \Omega \rightarrow \mathbb{R}^+$  y la varianza de la profundidad inversa  $V, \Omega \rightarrow \mathbb{R}^+$ , donde  $\Omega \subset \mathbb{R}^2$  es el conjunto de coordenadas normalizadas de los píxeles.

La transformación de sólido rígido con la notación del álgebra de Lie queda como  $\xi \in SE(3)$ , finalmente definimos una función proyectiva de deformación  $\omega$ , que proyecta un punto de la imagen  $p$  y su profundidad inversa  $d$  por una transformación entre frames  $\xi$  como:  $\omega(p, d, \xi)$ .

Como primer paso, el primer keyframe se inicializa con una profundidad aleatoria y una varianza grande. Se le da ciertos movimientos de translación y después de varios *keyframes* la profundidad converge.

Para llevar a cabo el *tracking* entre keyframes  $K_i = (I_i, D_i, V_i)$ , la transformación relativa  $\xi$  se calcula por la minimización de la varianza normalizada por el coste de Hubber del error fotométrico:

$$E_p(\xi_{ij}) \sum \frac{r_p^2(p, \xi_{ij})}{\sigma_{r_p(p, \xi_{ij})}^2}$$

donde

$$r_p(p, \xi_{ji}) = I_i(p) - I_j(\omega(p, D_i(p)), \xi_{ji})$$

y la varianza

$$\sigma_{r_p(p, \xi_{ji})}^2 = 2\sigma_I^2 + \left( \frac{\partial r_p(p, \xi_{ji})}{\partial D_i(p)} \right)^2$$

está minimización se lleva a cabo mediante el uso del algoritmo de Gauss-newton ponderado. Posteriormente se concatenan las estimaciones de movimiento.

$$\xi^{n+1} = \partial \xi^n \circ \xi^n$$

En la figura 37 se pueden observar los errores fotométricos  $r_p(p, \xi_{ji})$  para dos frames consecutivos.

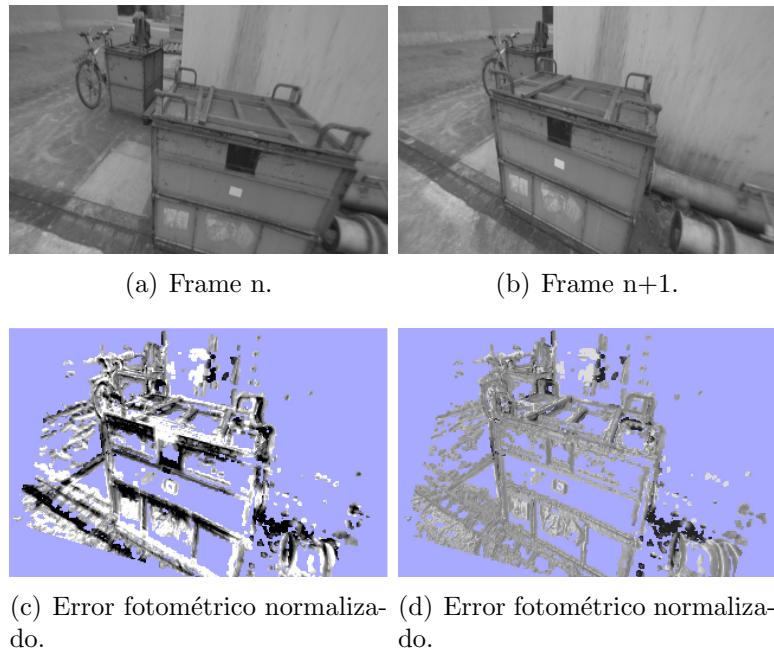


Figura 37: Errores fotométricos entre frames.

A continuación se explicarán los pasos en *Depth map estimation*. Cuando la cámara se mueve muy lejos del *keyframe* actual, se crea un nuevo *keyframe*. Para inicializar su mapa de profundidad se proyectan los puntos de *keyframes* anteriores. Si en los frames que se hace seguimiento no son elegidos como *keyframes* se utilizan para refinar el mapa de profundidad basándose en una comparación estéreo. En la figura 38 se puede observar un resultado.

A continuación se explicará el hilo de *Map optimization*. Después de que un nuevo *keyframe*  $K_i$  se añada al mapa, se usan los últimos 10 *keyframes* y una detección basada en la apariencia para realizar una *loop closure*.

La función de coste queda definida como:

$$E(\xi_{W1} \cdots \xi_{Wn}) = \sum (\xi_{ji} \circ \xi_{Wi}^{-1} \circ \xi_{Wj})^T \Sigma_{ji}^{-1} (\xi_{ji} \circ \xi_{Wi}^{-1} \circ \xi_{Wj})$$

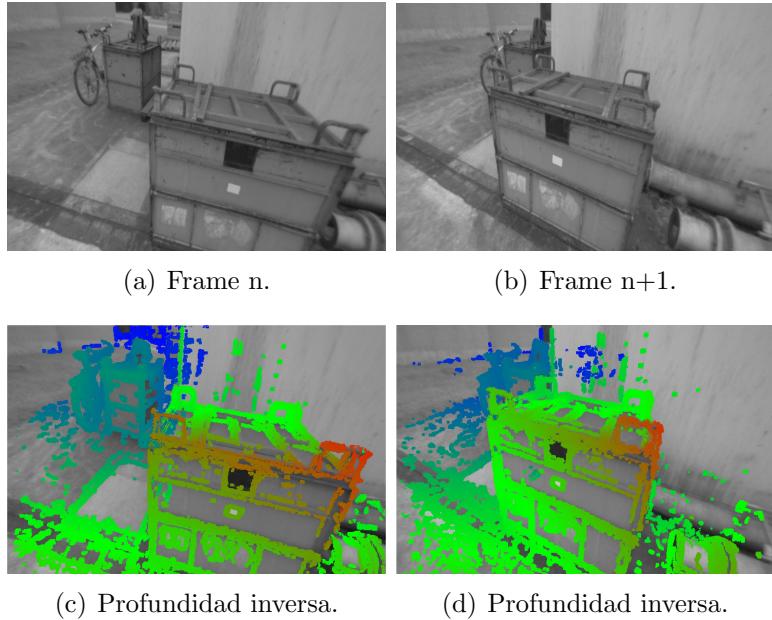


Figura 38: Profundidad entre frames.

Es una función de coste de mínimos cuadrados no lineales que se resuelve mediante Levenberg–Marquardt. En la figura 39 se puede observar la trayectoria del algoritmo y el reconocimiento de una escena ya visitada.

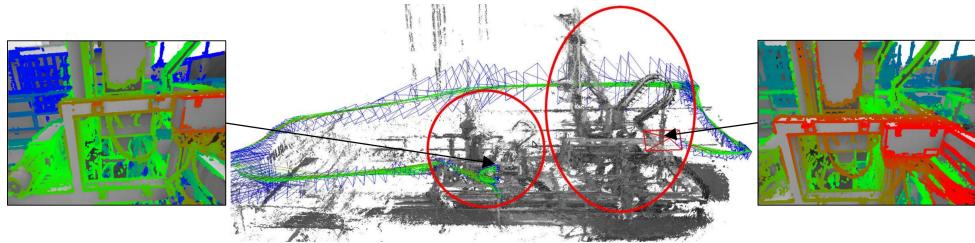


Figura 39: Ejemplo de *Loop closure*.

### 3.5.2. Resultados

El algoritmo se ha evaluado con el *dataset* de [33], los algoritmos de comparación son:

- Un algoritmo de odometría visual basado en métodos directos [9].
- El algoritmo de PTAM [20].
- Un algoritmo de SLAM monocular basado en métodos directos [19].
- Un algoritmo de SLAM con cámaras RGBD basado en características [8].

En la tabla 7 se puede observar el error en la trayectoria comparado con otros algoritmos.

Se puede observar que supera a algoritmo de odometría visual y al de PTAM, por otro lado, obtiene resultados similares al algoritmo de slam con cámaras RGBD.

	LSD	[9]	[20]	[19]	[8]
fr2/desk	4.52	13.5	x	1.77	9.5
fr2/xyz	1.47	3.79	24.28	1.18	2.6
sim/desk	0.04	1.53	-	0.27	-
sim/slowmo	0.35	2.21	-	0.13	-

Cuadro 7: Comparación del error en la trayectoria.

### 3.6. ORB-SLAM

En [22] presentan un algoritmo de SLAM basado en características, que opera en tiempo real, en pequeños y grandes escenarios. Comparándolo con los algoritmos de PTAM y LSD-SLAM en los dataset más populares, obtiene resultados superiores.

Los autores optan por el uso de técnicas de *Bundle adjustment* basadas en *keyframe*, además añaden un módulo de reconocimiento de escenas para realizar un *loop closure* y una inicialización del sistema sin intervención humana ( módulos que no incorpora PTAM ). A continuación se explicará el funcionamiento del algoritmo.

#### 3.6.1. Funcionamiento

El algoritmo ejecuta los módulos de *tracking*, *mapping* y el *loop closure* en tres hilos en paralelo. En la figura 40 se puede observar una descripción del algoritmo.

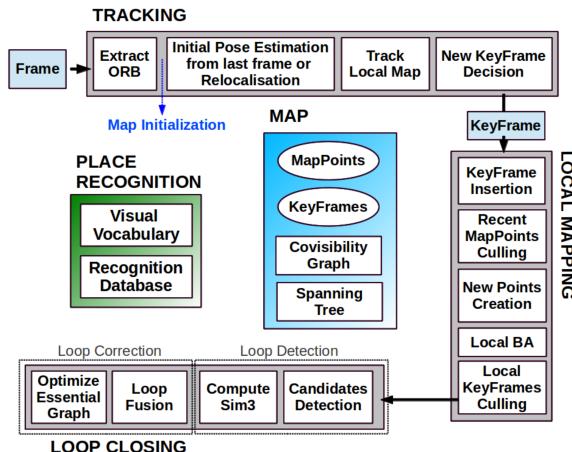


Figura 40: Diagrama del algoritmo.

El algoritmo tiene 2 grafos que representan el mapa, que se actualizan en cada iteración, aunque parezca redundante, ayuda a añadir robustez al algoritmo. El grafo de covisibilidad cada nodo es *keyframe* y se vinculan si estos comparten un 60 % de los puntos. El grafo esencial mantiene la el grafo de la trayectoria, donde cada nodo representa un *keyframe* y la conexión la transformación de sólido riágido entre ellos.

A continuación se explicarán los tres módulos principales. El objetivo del hilo del *tracking* es el de localizar la cámara en cada frame y decidir cuando insertar un nuevo *keyframe* en el mapa.

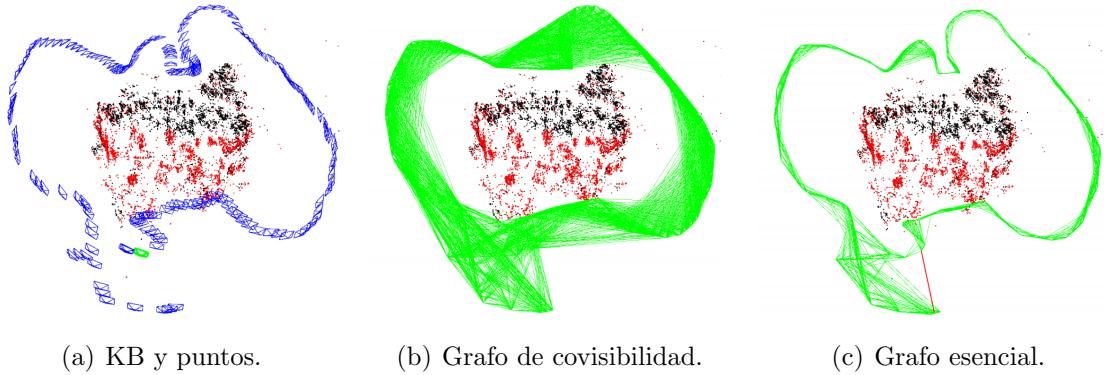


Figura 41: Representaciones de ORB-SLAM.

Lo primero que hace es extraer puntos ORB mediante un enfoque piramidal e intentando que los puntos estén repartidos homogéneamente por la imagen. Si el *tracking* del frame anterior ha resultado satisfactorio se utiliza un modelo de movimiento de velocidad constante para predecir la posición de la cámara y hacer una búsqueda guiada de los puntos del frame anterior, esta búsqueda consiste en comparar los descriptores de ORB obtenidos en frames sucesivos con el actual. Una vez hecho el *matching* se optimiza la pose del frame.

Puede darse el caso que se haya perdido el *tracking*, en este caso se convierte el frame en una instancia de *Bag of words* y se busca entre las instancias de *Bag of words* de los *keyframes* anteriores.

A partir de los puntos del mapa visibles por el *keyframe* correspondiente y con las detecciones del frame actual se relaciona con el algoritmo de *perspective n points*.

Una vez que tenemos la estimación de la pose de la cámara y un conjunto de puntos relacionados, optimizamos la pose de la cámara mediante la minimización del error de retroproyección de los puntos.

Finalmente, el último paso de este módulo consiste en decidir si se inserta un *keyframe* en el mapa. Las reglas que se deben cumplir para insertar un *keyframe* son:

- Deben haber pasado más de 20 frames desde la última inserción.
- El hilo de *mapping* debe estar ocioso.
- El frame actual debe tener más de 50 puntos detectados.

Estas reglas se han encontrado para asegurarse de una buena relocalización y un *tracking*.

A continuación se explicará que operaciones lleva a cabo el hilo de *mapping* con cada *keyframe* insertado. Lo primero que se hace es actualizar el grafo de covisibilidad y luego se calcula su instancia de *Bag of words*.

Seguidamente, se triangulan los puntos característicos mediante el uso de la geometría epipolar entre *keyframes*. Después se lleva a cabo una optimización mediante *local Bundle adjustment* de los puntos en el mapa y de las poses de los *keyframes* conectados.

Finalmente, para obtener una reconstrucción compacta, el hilo de *mapping* intenta eliminar *keyframes* redundantes. Así se eliminan esos *keyframes* cuyos 90 % puntos se han

visto por otros *keyframes*.

El hilo de *Loop closure*, coge el *keyframe* procesado por hilo de *mapping* e intenta detectar y cerrar el *loop*. Lo primer que hace es calcular la similiaridad de la instancia de *bag of words* del *keyframe* actual con los *keyframes* vecinos en el grafo de covisibilidad. El que obtenga mayor similaridad será el candidato, a partir de sus puntos 3D calculamos la relación de transformación con el *keyframe* actual. Esta transformación será la deriva de la trayectoria acumulada, esta se propaga por los vecinos y se alinea la trayectoria. Finalmente se actualiza y optimiza el grafo esencial.

### 3.6.2. Resultados

En este apartado se explicarán los resultados del algoritmo. El primer análisis que se hace es de rendimiento, en las tablas 8 y 9 se pueden observar los tiempos del *tracking* y *mapping* para una secuencia con una cantidad típica de puntos. Se puede observar que el *tracking* se obtiene una ejecución de tiempo real, en cambio *mapping* no, ya que no se necesita. Este tiempo no se incrementa ya que el módulo de *Bundle adjustment* solo se hace con  $m$  *keyframes* no con todos los *keyframes*.

extracción ORB	11.42 ms
estimación pose inicial	3.38 ms
seguimiento mapa local	14.84 ms
<b>Total</b>	<b>30.57</b>

Cuadro 8: Tiempos hilo *tracking*.

inserción de KB	10.29 ms
sacrificio de puntos del mapa	0.10 ms
creación de puntos del mapa	66.79 ms
BA local	298.08 ms
sacrificio de KB	8.07 ms
<b>Total</b>	<b>383.59 ms</b>

Cuadro 9: Tiempos hilo *mapping*.

Se compara el algoritmo con los algoritmos de PTAM, LSD-SLAM y RGBD-SLAM [8] (conjunto de algoritmos con que se compara en LSD-SLAM) en el *benchmark* [33]. En la tabla 10 se puede observar un subconjunto de esta comparación, la mediada, es la diferencia de la trayectoria obtenida con la *ground truth*.

	ORB-SLAM	PTAM	LSD-SLAM	RGBD-SLAM
fr2/desk	0.88	-	4.57	3.94
fr2/xyz	0.30	0.20	2.15	2.61
fr1/desk	1.69	-	10.65	2.52
fr2/desk_person	0.63	-	31.73	2.00

Cuadro 10: Comparación de algoritmos.

En términos de precisión ORB-SLAM y PTAM son similares, mientras que ORB-SLAM es más preciso en escenas grandes. Estos dos algoritmos son superiores a LSD-SLAM y RGBD-SLAM en todos los escenarios, salvo en aquellos en que no hay textura.

En la figura 42 se puede observar la trayectoria de ORB-SLAM comparándola con la *ground truth*, también es interesante ver el efecto de la aplicación del *Bundle adjustment*.

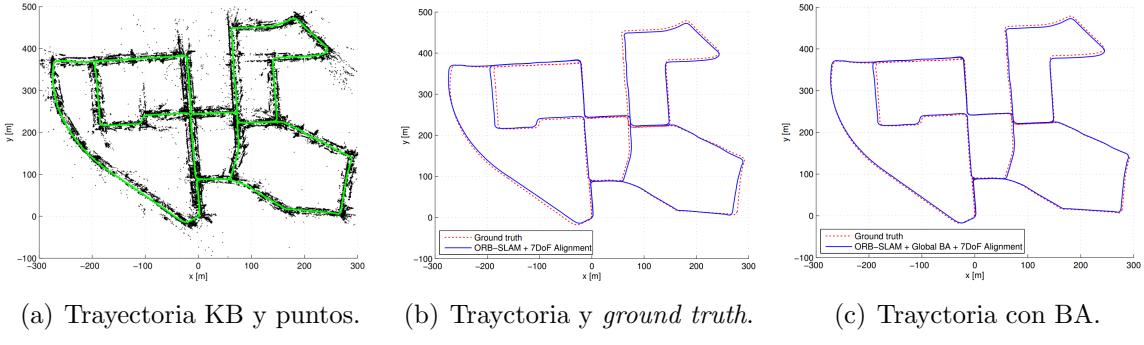


Figura 42: Secuencia de ORB-SLAM.

## 4. Benchmarks

Con los *datasets* públicos se permite una evaluación y comparación objetiva de los algoritmos. Se ha demostrado que el uso de los *datasets* públicos ayudan a impulsar las investigaciones.

La mayoría de los *datasets* contienen una *ground truth* con la que comparar el algoritmo, esta se obtiene con sensores independientes. La *ground truth* se compara con el resultado del algoritmo mediante dos métricas, el error de pose relativa, que es la diferencia entre la estimación obtenida y la verdadera en cada frame y con el error de trayectoria absoluta, esta primero alinea las dos trayectorias y calcula la diferencia. Hay que añadir que la trayectoria también se compara visualmente.

Algunos de los *benchmarks* específicos para autolocalización visual son:

- [30], este *dataset* contiene medidas de escáneres, IMU y cámaras. En cuanto a cámaras incluye: cámaras onmidireccionales, pares estéreo y una sola cámara. Muy usado en la comunidad de robótica móvil.
- [33], este es un *dataset* creado por el grupo de visión por ordenador de la Universidad Técnica de Munich [TUM]. Consiste en unas secuencias de imágenes RGBD junto a una *ground truth* obtenida mediante un sistema de captura de movimiento. Contempla diversos escenarios, como escenarios interiores, exteriores, y escenas con poca textura. Muy establecido en la comunidad de autolocalización visual.

## 5. Conclusiones

En este trabajo se ha planteado el problema de la localización visual con sistemas monoculares, se han presentado los métodos principales para resolver el problema y los artículos más significativos.

El algoritmo de MonoSlam fue el primero en resolver el problema de SLAM con un sistema monocular en tiempo real, ha sido el máximo exponente de las técnicas *Filtering*. Luego vino PTAM, que separa el *tracking* y el *mapping* en dos hilos y además usa técnicas de *Bundle adjustment* para optimizar la trayectoria. En el artículo [31] analizan las dos técnicas y obtienen como resultado que las técnicas basadas en *Bundle adjustment*, son más precisas que las *filtering* para el mismo coste computacional.

Así PTAM se ha convertido en el estándar con el que se han comparado el resto de algoritmos. El algoritmo estima la trayectoria de la cámara con precisión y rápida, además la representación que hace del mapa es bastante densa y es fácilmente escalable para entornos grandes.

En los últimos años han surgido los llamados métodos directos que utilizan la información de toda la imagen, no solo una cantidad reducida de puntos de ellas. Uno de los algoritmos es SVO, este algoritmo, no se considera de SLAM sino de odometría visual ya que la representación del mapa no es muy densa. También está LSD-SLAM, que utiliza métodos semidirectos para localizar la cámara y mapear el entorno.

Finalmente, el algoritmo de ORB-SLAM [23], un algoritmo de SLAM basado en características, sigue la estructura de PTAM y añade módulos de reconocimiento de escenas para llevar a cabo un *loop closure* y corregir la deriva. Consiguiendo unos resultados superiores al resto de algoritmos. Hay que añadir que este algoritmo falla en las imágenes donde hay *motion blur*, baja textura y textura repetible.

### 5.1. Trabajo futuro

En la conferencia *International Conference of Computer vision* (ICCV) hubo un taller titulado *The future of real-time SLAM* [3], donde se analizó hasta donde se ha llegado con los algoritmo de SLAM y hacia donde se encaminaría el trabajo futuro.

El único gran cambio en los algoritmos de SLAM ha sido la introducción de las cámaras RGBD de bajo coste, ya que se usan las mismas técnicas de hace 10 años.

Por otro lado dentro del trabajo futuro podríamos decir:

- Integrar información semántica en los sistemas SLAM. La unión de ambos técnicas ayudaría a obtener representaciones de la escena más sustanciales. Algunos algoritmos que hacen uso de estas técnicas [27]. No obstante también podríamos considerar el uso de redes de convolución para resolver el problema de la localización como en [18] sin hacer uso de SLAM.
- El uso de información multiagente, consiste en estudiar como se combinan la información procedente de distintos robots para extraer información global más consistente y precisa. Un ejemplo es [11].

- Utilizar *hardware* más eficiente. Los sistemas actuales se ejecutan en GPU o CPU donde el consumo es del orden de 100 vatios. Para dotar de autonomía energética a los robots hay que reducir el consumo.
- Robustez en escenas dinámicas. La mayoría de algoritmos presentados asumen una escena estática, pero el mundo real no es así, es dinámico. Algunos algoritmos que solucionan este problema [25].
- Escalabilidad, consiste en poder autolocalizarse en escenas más grandes. Un ejemplo es [1].

## Referencias

- [1] Winston Churchill and Paul Newman. Experience-based navigation for long-term localisation. *The International Journal of Robotics Research*, 32(14):1645–1661, 2013.
- [2] Javier Civera, Andrew J Davison, and JM Martinez Montiel. Inverse depth parametrization for monocular slam. *Robotics, IEEE Transactions on*, 24(5):932–945, 2008.
- [3] International conference on computer vision. The future of real-time slam. 2015.
- [4] Dr. Daniel Cremers. Multiple view geometry. <http://vision.in.tum.de/teaching/ss2016/mvg2016>, 2013.
- [5] Andrew J Davison and David W Murray. Simultaneous localization and map-building using active vision. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):865–880, 2002.
- [6] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052–1067, 2007.
- [7] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110, June 2006.
- [8] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard. An evaluation of the rgb-d slam system. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1691–1696, May 2012.
- [9] J. Engel, J. Sturm, and D. Cremers. Semi-dense visual odometry for a monocular camera. In *2013 IEEE International Conference on Computer Vision*, pages 1449–1456, Dec 2013.
- [10] Jakob Engel, Thomas Schöps, and Daniel Cremers. *Computer Vision – ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part II*, chapter LSD-SLAM: Large-Scale Direct Monocular SLAM, pages 834–849. Springer International Publishing, Cham, 2014.
- [11] Christian Forster, Simon Lynen, Laurent Kneip, and Davide Scaramuzza. Collaborative monocular slam with multiple micro aerial vehicles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

- [12] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 15–22. IEEE, 2014.
- [13] Xiao-Shan Gao, Xiao-Rong Hou, Jianliang Tang, and Hang-Fei Cheng. Complete solution classification for the perspective-three-point problem. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(8):930–943, 2003.
- [14] D. Hahnel, W. Burgard, D. Fox, and S. Thrun. An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements. In *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 1, pages 206–211 vol.1, 2003.
- [15] C G Harris and J M Pike. 3d positional integration from image sequences. In *In Proc. Alvey Vision Conference, Cambridge.England*, 1987.
- [16] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.
- [17] M. Irani and P. Anandan. All about direct methods, 1999.
- [18] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Convolutional networks for real-time 6-dof camera relocalization. *CoRR*, abs/1505.07427, 2015.
- [19] C. Kerl, J. Sturm, and D. Cremers. Dense visual slam for rgbd cameras. In *Proc. of the Int. Conf. on Intelligent Robot Systems (IROS)*, 2013.
- [20] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007.
- [21] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o (n) solution to the pnp problem. *International journal of computer vision*, 81(2):155–166, 2009.
- [22] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, Oct 2015.
- [23] Raúl Mur Artal, José María Martínez Montiel, and Juan Domingo Tardos Solano. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. 2015.
- [24] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. Dtam: Dense tracking and mapping in real-time. In *2011 International Conference on Computer Vision*, pages 2320–2327, Nov 2011.
- [25] Richard A Newcombe, Dieter Fox, and Steven M Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 343–352, 2015.
- [26] Eduardo Perdices García and José María Cañas Plaza. Autolocalización visual en la robocup basada en detección de porterías 3d. In *Proyecto final de carrera*. URJC, 2009.
- [27] Renato F. Salas-Moreno, Richard A. Newcombe, Hauke Strasdat, Paul H.J. Kelly, and Andrew J. Davison. Slam++: Simultaneous localisation and mapping at the level

- of objects. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.
- [28] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry [tutorial]. *Robotics & Automation Magazine, IEEE*, 18(4):80–92, 2011.
  - [29] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. Bradford Company, Scituate, MA, USA, 2004.
  - [30] M. Smith, I. Baldwin, W. Churchill, R. Paul, and P. Newman. The new college vision and laser data set. *The International Journal of Robotics Research*, 28(5):595–599, May 2009.
  - [31] Hauke Strasdat, J. M. M. Montiel, and Andrew J. Davison. Real-time monocular slam: Why filter?, 2012.
  - [32] Jürgen Sturm. Direct methods for visual slam. *Autonomous Navigation for flying robots*.
  - [33] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 573–580. IEEE, 2012.
  - [34] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.
  - [35] Stephan Weiss, Markus W Achtelik, Simon Lynen, Michael C Achtelik, Laurent Kneip, Margarita Chli, and Roland Siegwart. Monocular vision for long-term micro aerial vehicle state estimation: A compendium. *Journal of Field Robotics*, 30(5):803–831, 2013.