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

Lic. Sergio Gonzalez
Departamento de Computación
FCEyN UBA
Buenos Aires, Argentina
Email: segonzalez@dc.uba.ar

Lic. Emiliano Gonzalez
Departamento de Computación
FCEyN UBA
Buenos Aires, Argentina
Email: edgonzalez@dc.uba.ar

Lic. Matias Nitsche
Departamento de Computación
FCEyN UBA
Buenos Aires, Argentina
Email: mnitsche@dc.uba.ar

Dr. Pablo De Cristóforis
Departamento de Computación
FCEyN UBA
Buenos Aires, Argentina
Email: pdecris@dc.uba.ar

***Resumen*—En este trabajo se presenta un sistema de odometría visual de tiempo real para robots móviles basado en el método de Visual EKF-SLAM que utiliza *Smartphones* como unidades de sensado y procesamiento. A partir de la detección de marcas visuales en el ambiente, se construye un mapa que permite estimar la ubicación de la cámara del dispositivo móvil y, por ende, del robot. El sistema desarrollado fue evaluado en distintos escenarios utilizando un *Smartphone* Samsung Galaxy S3 montado en un robot móvil. Los experimentos realizados fueron analizados teniendo en cuenta aspectos de precisión, rendimiento y robustez. Los resultados obtenidos muestran la validez del método propuesto, así como también de la implementación del sistema desarrollado.**

I. Introducción

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*) [1], [2].

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. 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. Este enfoque utiliza un modelo probabilístico para representar las incertezas de las posiciones tanto de los *landmarks* como del robot con distribuciones Gaussianas multidimensional [3].

Los pasos básicos que involucran a EKF-SLAM pueden resumirse en: extracción de marcas naturales del entorno, asociación de esas marcas obtenidas del sensado con el mapa que se tiene hasta el momento del ambiente, localización para estimar la ubicación del sensor (y por ende del robot) y actualización del mapa para agregar, eliminar y actualizar las marcas del ambiente. Dichos pasos se repiten constantemente, lo que hace de EKF-SLAM un método iterativo e incremental.

En caso de utilizar una cámara como sensor, el método se conoce como Visual EKF-SLAM. Para extraer las marcas naturales del entorno (*landmarks*) se detectan en la imágenes capturadas determinadas características (*image features*) que representan la proyecciones de las marcas del entorno en la imagen. Para cada característica se construye un descriptor que permite identificarla univocamente. Los descriptores de las características se utilizan para establecer correspondencias y asociar las características que se extraen en la imagen actual con las almacenadas en el mapa.

En este trabajo se presenta un sistema de odometría visual de tiempo real para robots móviles basado en el método de Visual EKF-SLAM que utiliza *Smartphones* como unidades de sensado y procesamiento.

II. Trabajos relacionados

Considerando que el entorno de ejecución elegido son dispositivos móviles con limitado poder de cómputo, resulta necesario utilizar algoritmos eficientes de Visual EKF-SLAM que permitan alcanzar tiempo real en la ejecución. Para el desarrollo del sistema que se presenta en este trabajo, se toma como base los trabajos de Visual EKF-SLAM propuestos por Davison *et al.* [4], Civera *et al.* [5] y Montiel *et al.* [6]. De estos trabajos relacionados se identifican tres elementos principales que forman parte del núcleo del sistema que se presenta.

El modelo cinemático de Davison es un modelo 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 [4].

1-Point RANSAC es una variante del clásico método de RANSAC (*Random Sample Consensus*) para el filtrado de datos espurios (*outliers*) [5]. A diferencia del algoritmo tradicional de RANSAC, para generar el conjunto de concenso se utiliza una única correspondencia, logrando mejorar el rendimiento del original en varios órdenes de magnitud.

La parametrización inversa de la profundidad (o *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 [7].

III. EKF-SLAM Monocular

En EKF-SLAM monocular se requiere definir: la representación del estado (que debe especificar las variables asociadas al mapa y la ubicación de la cámara), el modelo cinemático (que realiza la predicción del movimiento utilizado para predecir las mediciones), el modelo de sensado (cuya función es predecir las mediciones y extraer datos útiles del sensado) y el modelo de ruidos (que define los ruidos asociados a las variables del modelo).

III-A. Representación del Estado

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

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

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

$$\mathbf{x}_{u,t} = \begin{pmatrix} \mathbf{r}_t & \mathbf{q}_t & \mathbf{v}_t & \omega_t \end{pmatrix}^\top \tag{2}$$

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 inversa de la profundidad (*inverse depth*) hasta convertirlos a la parametrización euclídea XYZ 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 $dim(\mathbf{x}_t) = 13 + 3 \times n_{XYZ} + 6 \times n_{ID}$.

III-B. Modelo Cinemático

El estado se actualiza en dos pasos bien definidos: predicción y actualización. 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 este trabajo se utiliza un modelo de velocidad constante tanto angular como lineal propuesto por Davison [4]. Bajo dicho modelo se asume que los movimientos que realiza la cámara son continuos y de velocidad constante durante toda la ejecución del sistema. Las aceleraciones se modelan como un ruido de distribución Gaussiana con media cero, estos ruidos 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

$$f(\mathbf{x}_u) = \begin{pmatrix} \mathbf{r}_{t|t-1} \\ \mathbf{q}_{t|t-1} \\ \mathbf{v}_{t|t-1} \\ \omega_{t|t-1} \end{pmatrix} = \begin{pmatrix} \mathbf{r}_{t-1} + (\mathbf{v}_{t-1} + V)\Delta t \\ \mathbf{q}_{t-1} \times q((\omega_{t-1} + \Omega)\Delta t) \\ \mathbf{v}_{t-1} + V \\ \omega_{t-1} + \Omega \end{pmatrix} \tag{3}$$

$$\mathbf{n} = \begin{pmatrix} V \\ \Omega \end{pmatrix} = \begin{pmatrix} a\Delta t \\ \alpha\Delta t \end{pmatrix} \tag{4}$$

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 $\mathbf{x}_u$ en el instante t a partir de la información del paso anterior $t-1$.

III-C. Modelo de sensado

El Modelo de sensado para este trabajo está basado en el modelo de cámara Pinhole, agregando además un modelo de distorsión de dos parámetros [8]. Este modelo se utiliza 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 y además para realizar la retroproyección de los puntos detectados, desde la imagen al mapa del sistema.

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

$$\mathbf{y_E} = \begin{pmatrix} X & Y & Z \end{pmatrix}^\top \tag{5}$$

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

$$\mathbf{y_{ID}} = \begin{pmatrix} x & y & z & \theta & \phi & \rho \end{pmatrix}^\top \tag{6}$$

El vector $\mathbf{y_{ID}}$ de la ecuación (6) 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 $\rho = 1/d$.

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

$$f_{ID \to E}(\mathbf{y_{ID}}) = \mathbf{y_E} \tag{7}$$

$$f_{ID \to E}(\mathbf{y_{ID}}) = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \frac{1}{\rho} m(\theta, \phi) \tag{8}$$

$$m(\theta,\phi) = (\ \cos\phi\sin\theta,\ \ -\sin\phi,\ \ \cos\phi\cos\theta\ \ )^\top \quad (9)$$

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

Dado que existen dos formas de representar los puntos del mapa, existen también dos formas de realizar su proyección al plano de la imagen. 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:

$$\mathbf{h} = R(\mathbf{y_E} - \mathbf{r}) = R\left(\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} - \mathbf{r}\right) \quad (10)$$

En el caso de los puntos representados en la parametrización inversa de la profundidad (Inverse Depth), se utiliza la ecuación (7):

$$\mathbf{h} = R(f_{ID\to E}(\mathbf{y_{ID}}) - \mathbf{r}) = R\left(\begin{pmatrix} x \\ y \\ z \end{pmatrix} - \mathbf{r} + \frac{1}{\rho} m(\theta,\phi)\right) \quad (11)$$

donde $\mathbf{h} = (h_x, h_y, h_z)$, $R \in \mathbb{R}^{3\times3}$ es la matriz de rotación del sistema de coordenadas del mundo a la orientación del sistema de coordenadas de la cámara.

La proyección en el plano de la imagen se realiza según el modelo de cámara Pinhole [9], para obtener el pixel en la imagen sin distorsionar $\mathbf{h}_u$, utilizando los parámetros de calibración que deben ser determinados previamente:

$$\mathbf{h}_u = \begin{pmatrix} u_u \\ v_u \end{pmatrix} = \begin{pmatrix} p_x - \frac{f}{d_x}\frac{h_x}{h_z} \\ p_y - \frac{f}{d_y}\frac{h_y}{h_z} \end{pmatrix} \quad (12)$$

donde 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.

Para realizar la retroproyección, es necesario 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.

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

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

Obtener la dirección del rayo de origen en el centro de la cámara y que pasa por el punto $\mathbf{h}_u$, despejando las variables $\frac{h_x}{h_z}$ y $\frac{h_y}{h_z}$ de la ecuación (12):

$$((u_u - p_x)\frac{d_x}{f}, (v_u - p_y)\frac{d_y}{f}, 1)^\top \quad (13)$$

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 como sigue:

$$\tilde{\mathbf{h}} = R_{q_t}\ (u_u,\ v_u,\ 1)^\top \quad (14)$$

donde $\tilde{\mathbf{h}} \in \mathbb{R}^3$ resulta ser un vector direccional no unitario en coordenadas del mundo y $R_{q_t} \in \mathbb{R}^{3\times3}$ es la matriz de rotación del sistema de coordenadas de la cámara a la orientación del sistema de coordenadas del mundo.

Para inicializar un *landmark* con su parametrización inversa de la profundidad (Inverse Depth) y agregarlo al mapa del sistema, 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, de la siguiente manera:

$$P' = J\begin{pmatrix} P_{n\times n} & \mathbf{0}_{1\times2} & 0 \\ \mathbf{0}_{2\times1} & R_i & 0 \\ 0 & 0 & \sigma^2_{\rho_0} \end{pmatrix} J^\top \quad (15)$$

$$R_i = \begin{pmatrix} \sigma_{PixelX} & 0 \\ 0 & \sigma_{PixelY} \end{pmatrix} \quad (16)$$

$$J = \begin{pmatrix} I_{n\times n} & \mathbf{0}_{n\times3} \\ \frac{\partial Init_{ID}}{\partial r_t}, \frac{\partial Init_{ID}}{\partial q_t}, \mathbf{0}_{6\times(n-7)} & \frac{\partial Init_{ID}}{\partial h}, \frac{\partial Init_{ID}}{\partial \rho_0} \end{pmatrix} \quad (17)$$

donde $R_i \in \mathbb{R}^{2\times2}$ 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 según el trabajo de Civera *et al.*, 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.

IV. Implementación del método

El método implementado en este trabajo [1] se puede detallar agrupando conceptos o funcionalidades que interactúan entre sí. Los tres módulos que conforman la implementación (ver Figura 1) son: *Extended Kalman Filter SLAM* encargado de realizar las estimaciones a partir de la información de sensado, *Gestor de features*, responsable de extraer las características de cada imagen para cada iteración, analizarla y establecer las correspondencias con el mapa, y el *Gestor del mapa*, que administra los puntos o marcas naturales del mundo (*landmarks*) del mapa.

Extended Kalman Filter SLAM: es el módulo que realiza la predicción del estado de la cámara (posición, orientación, velocidades) y de los puntos del mapa (*landmarks*). Esta información es filtrada con 1-Point RANSAC, que elimina correspondencias espurias (*outliers*) provenientes de la heurística de búsqueda de correspondencias. Luego se realiza una primer actualización del estado en base al mejor conjunto de consenso calculado mediante 1-Point RANSAC. Como siguiente paso se

[1] https://github.com/lrse/OpenEKFMonoSLAM

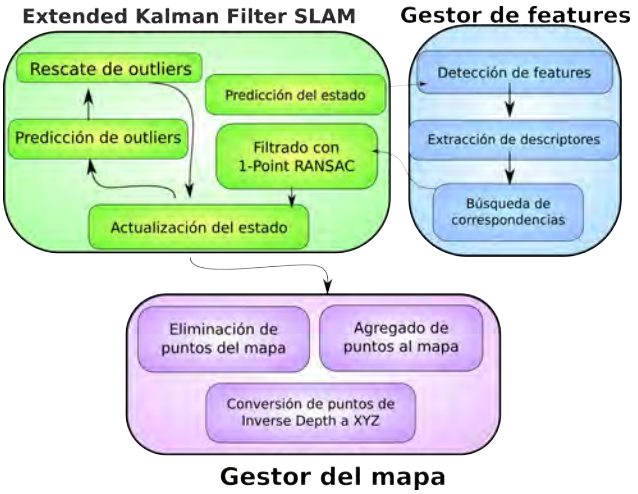

Figura 1. Diagrama de intercomunicación entre las funcionalidades.

realiza una nueva predicción sobre las mediciones, pero sólo teniendo en cuenta aquellas correspondencias que resultaron ser outliers y utilizando la nueva información sobre el estado (proveniente de la primer actualización). De esta forma es posible realizar un rescate de outliers, debido a que en el paso de filtrado con 1-Point RANSAC algunas correspondencias podrían ser desechadas al considerarlas outlier de manera incorrecta.

Gestor de features: la gestión de features involucra la detección, donde se procesa la imagen proveniente de la cámara, utilizando el detector de features STAR [10] de OpenCV. El proceso de extracción de descriptores asociados a cada feature detectado en la imagen se realiza utilizando el método de extracción de descriptores BRIEF. La búsqueda de correspondencias entre los features detectados entre una imagen y los puntos del mapa se realiza utilizando la correspondencia activa (*Active Matching* [11]), que predice la nueva ubicación de cada landmark del mapa antes de obtenerlos de la imagen lo que reduce el espacio de búsqueda de correspondencias de cada feature en la imagen a solamente una elipse de pocos pixeles. Dicha elipse representa el intervalo de confianza (del 99 % de probabilidad) de que dicho landmark se encuentre dentro de esa zona.

Gestor del mapa: En esta etapa se realiza la eliminación de puntos del mapa, se quitan del mapa aquellos puntos que no aporten lo suficiente a la estimación y mantiene el tamaño del mapa en un número reducido de puntos. 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, para esto se mantiene la estadistica

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

para cada feature del mapa. Es aquí donde se realiza la conversión de puntos con parametrización inversa de la profundidad (Inverse Depth) a XYZ cuando su parametrización XYZ supera un cierto límite de linealidad. Por último se realiza un agregado de puntos al mapa, donde se detectan y agregan nuevos puntos al mapa cuando el tamaño del mismo no es suficiente para establecer correspondencias en las siguientes iteraciones.

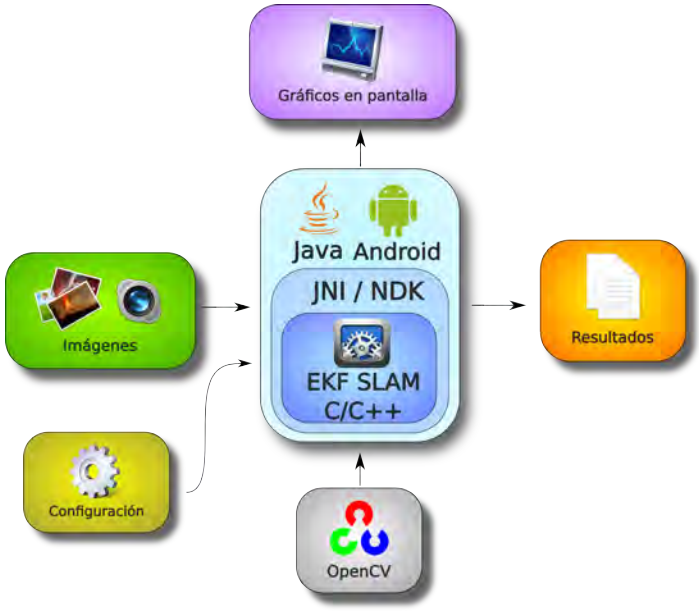

Figura 2. Esquema de alto nivel del sistema para la plataforma Android.

El sistema fue desarrollado en el lenguaje C/C++ y tiene una arquitectura modular. Como única dependencia externa posee la librería de visión por computadora OpenCV [12] que facilita su portabilidad. El sistema puede ser compilado y ejecutado en varias arquitecturas, incluyendo x86 (PC) y ARM (*Smartphones*).

Aplicación Android

Se desarrollo una variante para el sistema operativo Android para *Smartphones* (ver Figura 2). La arquitectura consiste en tres capas. La primera capa contiene el sistema de localización desarrollado en C/C++. La tercera capa (la de más alto nivel) corresponde a la de la aplicación Android, escrita en lenguaje *Java* mediante la librería Android SDK. 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*). 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.

V. Experimentos

El robot utilizado para realizar los experimentos es el Exabot [13], un robot mini móvil multipropósito con un sistema de tracción basado en orugas no-holonómico que cuenta con 2 grados de libertad (rotación y traslación hacia adelante y atrás) en el plano en el que se mueve. Sobre el robot se monta un soporte para el *Smartphone*, y una computadora, que se usa para ejecutar los experimentos en modo offline y contrastar los resultados con los obtenidos en el *Smartphone*.

El *Smartphone* utilizado es un Samsung Galaxy S3, que cuenta con un procesador de arquitectura ARM, modelo Cortex-A9, una frecuencia de reloj de 1.4 GHz y cuatro

Figura 3. Robot Exabot realizado en la Facultsd de Ciencias Exactas y Naturales (FCEyN).

núcleos y una cámara de 8M pixeles y auto foco. El sistema operativo instalado es un Android versión 4.1.2 (*Jelly Bean*). Las imágenes a procesar para los experimentos tienen una resolución de 640×480 pixels (el doble de resolución que el trabajo presentado por Civera *et al.*).

El sistema de localización visual global utilizado para comparar los resultados tiene un error en el orden del milímetro [14]. Es por esta razón que las posiciones obtenidas con este sistema son considerados como los valores reales de posición (*ground truth*) para comparar la estimación del realizado en este trabajo.

Se plantearon diferentes trayectorias en las cuales el robot debería desplazarse y se realizaron a velocidad constante, en línea recta, un recorrido en semi círculo y una trayectoria en zigzag. Todos éstos escenarios se realizaron con dos orientaciones del *Smartphone*: alineado con el frente del robot y ubicado transversalmente con respecto al mismo.

(a) (b)

Figura 4. Trayectoria en línea recta y en zigzag utilizando el robot Exabot.

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.

VI. Resultados

La calidad de la estimación calculada por el sistema de localización es comparada contra el ground truth midiendo la diferencia máxima entre cada uno de los recorridos. De esta forma se acota superiormente el error global. En la figura 5 se puede observar el resultado de las estimaciones entre las diferentes configuraciones cuando el robot se desplaza en linea recta y en zigzag.

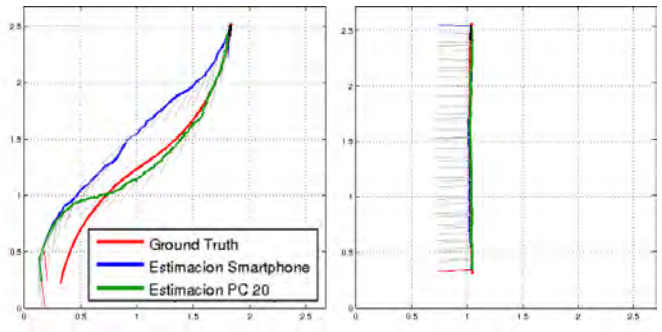

Figura 5. Resultado de la estimación entre el *groundtruth*, la pc con un mínimo de 20 correspondencias y el *Smartphone*. Recorrido en zigzag, con el *Smartphone* en forma frontal y recorrido en linea recta en posición transversal.

El resultado final de las pruebas realizadas se puede observar en la tabla I. En ella se muestra el error máximo de la trayectoria estimada para cada una de las configuraciones.

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 %)	-

Cuadro I. Errores máximos (en cm) de la trayectoria de cada uno de los experimentos junto con el porcentaje sobre la distancia total recorrida.

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. Para esto se divide el sistema en distintas etapas: 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.

En el *Smartphone* los tiempos de procesamiento resultan ser mayores que para el caso de la PC debido a que posee un reducido poder de cómputo en comparación. 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 Tabla II. Analizando los desvíos, éstos resultan ser pequeños, 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 de forma correcta 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.

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

Cuadro II. TIEMPOS PROMEDIO Y DESVÍOS ESTÁNDAR (EN MILISEGUNDOS) DE LAS EJECUCIONES EN EL *Smartphone*.

VII. DISCUSIÓN Y CONCLUSIONES

En este trabajo se presenta el desarrollo de un sistema de localización visual monocular basado en el enfoque conocido como Visual EKF-SLAM. Este sistema es *open source* (código abierto) y tiene dos versiones: una para smartphones y otra para PC. Los experimentos realizados para poner a prueba el sistema incluyeron distintas trayectorias de movimiento entre las que presentamos dos casos: en línea recta 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).

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*).

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.

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 este trabajo 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. De esta forma se alcanzó una implementación de tiempo real de ejecución que posibilita su utilización en *Smartphones* como unidades de sensado y procesamiento de robots móviles para abordar el problema de la odometría visual.

REFERENCIAS

[1] H. Durrant-Whyte and T. Bailey, "Simultaneous localisation and mapping (slam): Part i the essential algorithms," *IEEE ROBOTICS AND AUTOMATION MAGAZINE*, vol. 2, p. 2006, 2006.

[2] T. Bailey and H. Durrant-Whyte, "Simultaneous localisation and mapping (slam): Part ii state of the art," *Robotics & Automation Magazine, IEEE*, vol. 13, no. 3, pp. 108–117, Sep. 2006. [Online]. Available: http://dx.doi.org/10.1109/mra.2006.1678144

[3] L. M. Paz, J. D. Tardos, and J. Neira, "Divide and Conquer: EKF SLAM in O(n)," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1107–1120, OCT 2008.

[4] A. Davison, I. Reid, N. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 6, pp. 1052–1067, 2007.

[5] J. Civera, O. Grasa, A. Davison, and J. Montiel, "1-point ransac for extended kalman filtering: Application to real-time structure from motion and visual odometry," *Journal of Field Robotics*, vol. 27, no. 5, pp. 609–631, 2010.

[6] J. Montiel, J. Civera, and A. J. Davison, "Unified inverse depth parametrization for monocular slam," *analysis*, vol. 9, p. 1, 2006.

[7] J. Civera, A. J. Davison, and J. Montiel, "Inverse depth parametrization for monocular slam," *Robotics, IEEE Transactions on*, vol. 24, no. 5, pp. 932–945, 2008.

[8] E. M. Mikhail, J. S. Bethel, and J. C. McGlone, *Introduction to modern photogrammetry*. Wiley New York, 2001, vol. 31.

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

[10] M. Agrawal, K. Konolige, and M. Blas, "Censure: Center surround extremas for realtime feature detection and matching," *Computer Vision–ECCV 2008*, pp. 102–115, 2008.

[11] M. Chli and A. J. Davison, "Active matching."

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

[13] S. Pedre, M. Nitsche, F. Pessacg, J. Caccavelli, and P. De Cristóforis, "Design of a multi-purpose low-cost mobile robot for research and education," in *15th Towards Autonomous Robotic Systems (TAROS)*, ser. Lecture Notes in Computer Science. Springer, 2014.

[14] T. Krajník, M. Nitsche, J. Faigl, P. Vaněk, M. Saska, L. Přeučil, T. Duckett, and M. Mejail, "A practical multirobot localization system," *Journal of Intelligent & Robotic Systems*, 2014.