SLAM 3D basado en una cámara RGB-D y PCL

Francisco Domínguez, Lucas Liaño, Sebastián Verrastro

Universidad Tecnológica Nacional, Facultad Regional Buenos Aires,
Medrano 951, (C1179AAQ), Ciudad Autónoma de Buenos Aires, Argentina

fdominguez@frba.utn.edu.ar

Recibido el 26 de mayo de 2022, aprobado el 16 de junio de 2022

Resumen

En el presente trabajo se desarrolla un algoritmo capaz de resolver el problema de localización y mapeo simultáneos (SLAM), obteniendo la odometría visual a partir de imágenes en colores con mapa de profundidad (RGB-D) de una cámara Microsoft Kinect bajo el entorno de simulación Gazebo. Para ello, se propone un *pipeline* de registro que tiene como objetivo encontrar la mejor estimación de movimiento de cuerpo rígido para mapear una imagen de profundidad en otra, asumiendo una escena estática tomada por una cámara en movimiento. El *pipelin*e propuesto se basa en nubes de puntos organizadas, esto es, que dichas nubes se presenten como matrices 2D. Aprovechando esto, se emplea una técnica para reducir las muestras extraídas, llamada *normal space sampling*, aumentando la probabilidad de que el registro converja al mínimo global. Los resultados obtenidos se asemejan a la trayectoria real simulada por el robot.

PALABRAS CLAVE: RGB-D – PCL – SLAM3D – SLAM - GAZEBO

Abstract

In the present work an algorithm capable of solving the problem of simultaneous location and mapping (SLAM) is developed, obtaining visual odometry from color images with depth map (RGB-D) of a Microsoft Kinect camera under the environment of Simulation gazebo. For this, a registration *pipeline* is proposed that aims to find the best estimate of rigid body movement to map one depth image to another, assuming a static scene taken by a moving camera. The proposed *pipeline* is based on organized point clouds, that is, these clouds are presented in 2D matrices. Taking advantage of this, a technique is used to reduce the samples drawn, called normal space sampling, increasing the probability that the record will converge to the global minimum. The results obtained are similar to the real trajectory simulated by the robot.

KEYWORDS: RGB-D – PCL - SLAM3D – SLAM - GAZEBO

Introducción

Si bien el uso de robots tiene sus orígenes principalmente en fábricas para el ensamblaje de automóviles, en los últimos años la electrónica moderna ha permitido introducirlos en otras áreas, ya sea desde electrodomésticos hasta viajes espaciales con el fin de explorar planetas desconocidos. Esta expansión se debe a que los mismos permiten reducir la interacción humana no sólo en tareas que presentan un riesgo a la integridad de la persona, sino también aquellas que tienen cierto grado de repetitividad.

En el último tiempo muchas personas han adquirido los llamados robots aspiradoras, los cuales consiguen limpiar la superficie de las casas en un tiempo medianamente razonable, aunque este tiempo no suele ser una preocupación ya que, al ser el robot completamente autónomo, uno puede seguir con sus actividades cotidianas.

No obstante, si se analiza el recorrido que realizan la mayoría de estos robots, se puede apreciar que el mismo suele ser completamente aleatorio, y por ende se tendería a creer que no podrá pasar por toda la superficie y limpiarla en su totalidad. La realidad es que como están mucho tiempo circulando, logran pasar por todos los rincones de la superficie, pero esto genera que circulen más veces por unos lugares que por otros, tal como se observa en la Figura 1.

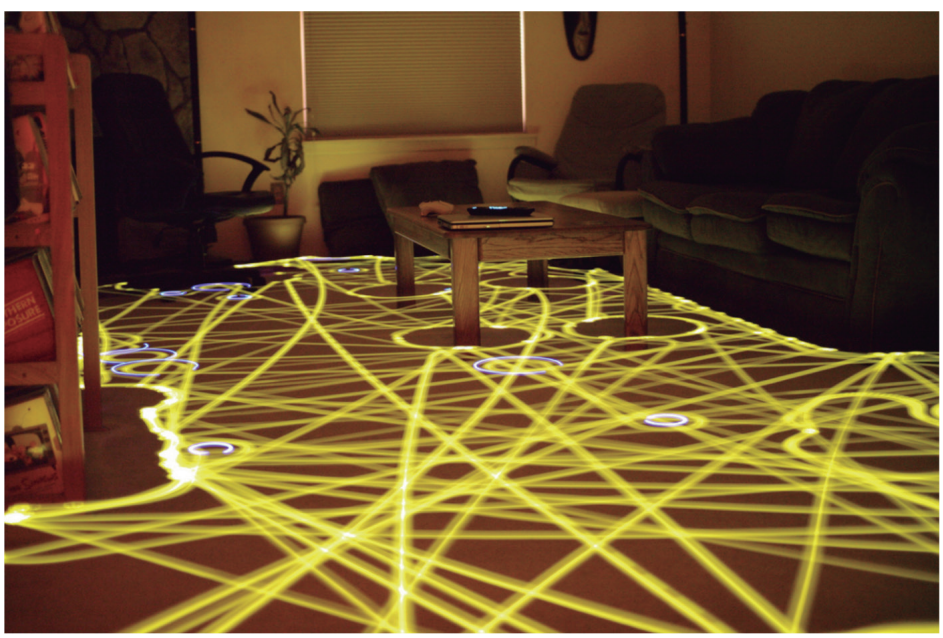

Fig. 1. Ejemplo del recorrido de un robot aspiradora comercial

Si lo que se pretende es optimizar el recorrido del robot, la posición y orientación combinadas del mismo, conocido como su *pose*, entonces, es una parte inherente de aplicaciones como el control del vehículo y el mapeo. Varios sensores se utilizan comúnmente para estimar la pose de un robot. Las unidades de medición inercial (IMU), la cámara y el LIDAR, el cual es un dispositivo que permite determinar la distancia desde un emisor láser a un objeto o superficie utilizando un haz láser pulsado, se encuentran entre los sensores más populares para la localización en interiores (Del Rosario, 2016), mientras que para exteriores suele sumarse el sistema de posicionamiento global (GPS) a esta lista, aunque es necesario aplicar correcciones sobre las medidas tomadas del mismo mediante acción de la IMU (Caron, 2006). A su vez, ciertos robots móviles terrestres suelen presentar el dato proveniente de la odometría de las ruedas tanto para interiores como exteriores, en donde se mide la rotación de las ruedas para estimar cambios de la posición del vehículo a lo largo del tiempo.

En particular, para el caso de la localización y mapeo tridimensional (SLAM 3D), los LIDAR 3D, los cuales suelen contar con una gran cantidad de emisores láser para obtener la representación tridimensional del entorno, han adquirido mucha popularidad en el último tiempo con la llegada de los vehículos autónomos, aunque son muy costosos. Si se pretende entonces una opción más económica, las cámaras estéreo y RGB-D *(D: Depth - profundidad)* son de las más utilizadas para esto, siendo estas últimas una combinación de cámaras monoculares junto a sensores infrarrojos. Los sensores RGB-D como la *Microsoft Kinect* proporcionan directamente mapas de profundidad densos e imágenes en color. En general, los enfoques SLAM que operan en imágenes RGB-D son estructuralmente diferentes de los sistemas estéreo, ya que se combinan los datos de una cámara monocular junto con sensores infrarrojos. A pesar de esto, pueden obtenerse nubes de puntos de profundidad a partir de ambos sensores con el procesamiento adecuado, como los presentados por la librería de visión artificial OpenCV (Open Computer Vision) (Kaehler, 2017).

Una vez obtenidas las nubes de puntos de profundidad, un método para calcular la pose del robot a lo largo del tiempo es mediante la comparación entre dos nubes de puntos contiguas (una con los datos recientes y otra con los datos de la nube de puntos anterior), buscando en definitiva la transformación necesaria para que la nube de puntos actual pueda alinearse (o sea, que coincida lo mejor posible) con la nube de puntos anterior. Este proceso se lo conoce como el *registro de la nube de puntos*, y suele llevar una serie de pasos, lo cual se desarrollará más adelante. Para facilitar el uso de nubes de puntos, hay librerías específicas para trabajar con dicha información, siendo algunas de ellas la *Point Cloud Library* y *Open3D*, las cuales implementan no sólo distintos algoritmos, sino también estructuras para facilitar el manejo de las nubes de puntos.

Si bien existen muchos robots móviles hoy en día en el mercado, gran parte de ellos son costosos, y en particular los que se encuentran equipados para realizar SLAM 3D. Para poder evaluar los algoritmos necesarios sin tener que ponerse en gasto, existen entornos de simulación pensados, entre otras cosas, para robots, a partir de los cuales se consiguen resultados comparables con los reales. Unos de los más conocidos en el ámbito son *V-REP* (Rohmer, 2013) y *Gazebo* (Koenig, 2004), siendo este último el elegido para realizar el presente trabajo.

Contribuciones

En este artículo, se desarrolla un algoritmo de registro de nube de puntos en base a los datos obtenidos de una cámara RGB-D en movimiento dentro de un ambiente estático, bajo el entorno de simulación Gazebo. En base a la transformación obtenida en cada iteración, se la aplica a la nube de puntos entrante, para luego sumarla al mapa 3D generado.

En el presente trabajo se pretende, a su vez, que los datos de posicionamiento y mapeo obtenidos puedan fusionarse fácilmente con los datos provenientes de otros sensores, tales como los proporcionados por una IMU y un LIDAR, para obtener una mejor estimación de la posición y, por consiguiente, obtener un mapa más exacto en caso de ser necesario. Por otra parte, no se incorporarán algoritmos de cierres de ciclo, ya que su implementación será tenida en cuenta para la realización de nuevos trabajos futuros.

Finalmente, se provee de un código compatible con el sistema operativo robótico ROS (Quigley, 2011), el cual es un *framework* pensado para utilizar en robótica, permitiendo portabilidad y versatilidad a la hora de proporcionar los datos de los sensores, ya que el mismo utiliza un sistema de mensajes que permite vincular a dos procesos sin importar el método de adquisición o procesamiento de los datos, mientras los mismos respeten el tipo de mensaje en sí.

Marco teórico

En esta sección se presentan distintos métodos para poder realizar una reconstrucción del entorno en base a las nubes de puntos, además de dar una introducción a la librería de nube de puntos (PCL) y su vínculo con ROS, utilizados para el desarrollo del trabajo.

Reconstrucción del entorno

En base a los datos provistos por los sensores exteroceptivos (LIDAR, cámaras, entre otros), el objetivo principal del trabajo consiste en recolectar dicha información para poder reconstruir el entorno en el cual se encuentra sumergido el robot, además de poder estimar su posición. Sin embargo, estos tipos de sensores proveen un campo de visión limitado, por lo que no es posible describir el mundo real como un todo en base a una sola medición de estos sensores, sino que solo puede mencionarse a una pequeña porción del mismo, denominada *escena.*

A su vez, el tipo de dato que se tome de la escena dependerá del tipo de sensor exteroceptivo que los provea. Por ejemplo, una cámara monocular es capaz de otorgar imágenes 2D, mientras que con una cámara estéreo se consigue una nube de puntos tridimensional.

Point Cloud

Una *nube de puntos*, o de su terminología en inglés, *point cloud*, se utiliza para describir a un conjunto de puntos de datos en un espacio dado. Las nubes de puntos 3D, por ejemplo, son un conjunto de puntos tridimensionales que se caracterizan por tener principalmente coordenadas espaciales XYZ, y opcionalmente pueden dar información de intensidad, color, entre otras.

Como se mencionaba anteriormente, la nube de puntos de la escena adquirida dependerá del principio de medición del sensor utilizado. En concreto, puede categorizarse en (Weinmann, 2016):

-Técnicas pasivas, donde la luz ambiental se encuentra presente, permitiendo utilizar sensores como cámaras estéreo para obtener las imágenes propias del entorno.
-Técnicas activas, donde los sensores emiten radiaciones electromagnéticas, tal como es el caso de los LIDAR 3D o las cámaras infrarrojas.

Dependiendo de la técnica de adquisición involucrada y el dispositivo utilizado, los datos de la nube de puntos adquiridos pueden corromperse con más o menos ruido y, además de la información espacial en forma de coordenadas XYZ, los atributos de los puntos respectivos, como la información de color o intensidad también puede adquirirse, como se mencionaba anteriormente.

Point Cloud Library (PCL)

La *Point Cloud Library* (PCL) (Rusu, 2011) consiste en un proyecto abierto desarrollado en *C++* que ofrece herramientas para procesar imágenes o nubes de puntos tanto 2D como 3D. El *framework* PCL contiene numerosos algoritmos que realizan filtrado, estimación de características, reconstrucción de superficies, registro, ajuste de modelos y segmentación. Con estos métodos, es posible procesar la nube de puntos, extraer *keypoints* para reconocer objetos en el mundo en función de su apariencia geométrica, crear superficies a partir de las nubes de puntos y visualizarlas.

En general, PCL contiene una estructura de datos muy importante, que es *pcl::PointCloud.* Esta estructura de datos está diseñada como una clase de template que toma el tipo de

punto que se utilizará como parámetro. Como resultado de esto, la clase de nube de puntos no es mucho más que un contenedor de puntos que incluye toda la información común requerida por todas las nubes de puntos independientemente de su tipo de punto. Algunos de los tipos de puntos más utilizados son

- *pcl::PointXYZ*, es el más simple que posee la librería. Este almacena la información de la posición en XYZ únicamente.

- *pcl::PointXYZRGB*, almacena además de la posición XYZ, el color en formato RGB de cada punto.

- *pcl::PointNormal*, representa la superficie normal en un punto dado y la medida de su curvatura, además de las coordenadas XYZ.

- *pcl::PointXYZI*, que a las coordenadas XYZ les asocia también información de intensidad del punto.

Debido al gran número de tipos de puntos que existen en la librería, cada algoritmo implementado en la misma refiere a una clase perteneciente a una jerarquía de clases con puntos en común específicos. Gracias a ello, dichos algoritmos pueden ser parametrizados en base a lo que necesite el usuario.

La interfaz PCL para ROS proporciona los medios necesarios para comunicar las estructuras de datos PCL a través del sistema de comunicación basado en mensajes propios por ROS. Para hacer esto, hay varios tipos de mensajes definidos para contener nubes de puntos, así como otros productos de datos de los algoritmos PCL. En combinación con estos tipos de mensajes, también se proporciona un conjunto de funciones para convertir de tipos de datos PCL nativos en mensajes. Uno de los más importantes tipos de mensaje de ROS es el *sensor_msgs::PointCloud2*, el cual contiene una colección de puntos N-dimensionales, que pueden contener información adicional como normales, intensidad, entre otros.

Para relacionar ambos mundos, ROS cuenta con el paquete *pcl_conversions (Open Robotics, pcl_conversions)*.

Point Cloud Registration

Si bien un primer paso es obtener la nube de puntos en base al sensor utilizado, uno de los principales enfoques utilizados en robots móviles es el llamado *registro de nube de puntos* o, de su terminología en inglés, *point cloud registration*. Dicho proceso responde a encontrar una transformación espacial que alinee dos nubes de puntos generalmente contiguas en el tiempo. Los sensores que utilizan normalmente este enfoque son las cámaras estéreo y RGB-D.

En concreto, dada una nube de puntos fuente (también llamada *input o source*) $\boldsymbol{P}$ con puntos $\boldsymbol{p \in P}$, y una nube de puntos objetivo $\boldsymbol{Q}$ (llamada *target)* con puntos $\boldsymbol{q \in Q}$, el problema del registro se basa en encontrar correspondencias entre $\boldsymbol{P}$ y $\boldsymbol{Q}$, y estimar una transformación $\boldsymbol{T}$ que, cuando se aplica a $\boldsymbol{P}$, se alinean todos los pares de puntos correspondientes ($\boldsymbol{p_i \in P,\ q_j \in Q}$). Un problema fundamental del registro es que estas correspondencias generalmente no se conocen y deben ser determinadas por el algoritmo de registro. Dadas las correspondencias correctas, hay diferentes formas de calcular la transformación óptima con respecto a la métrica de error utilizada, como se detalla a continuación.
El registro de dos nubes de puntos puede dividirse en una serie de pasos, los cuales se observan en la Figura 2 y conforman el denominado *registration pipeline*, el cual podría extenderse a un caso más general (García, 2017).

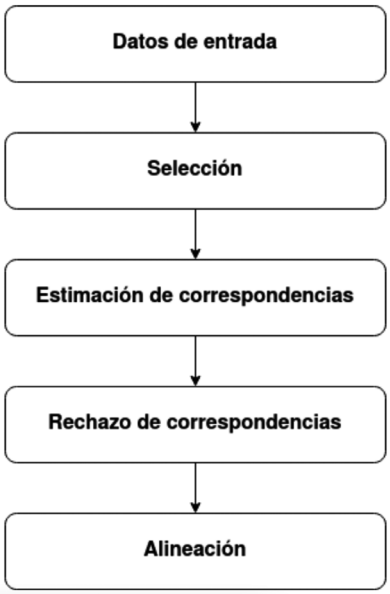

Fig. 2. Registration *pipeline*

En primera instancia, debido a la gran densidad de puntos que presentan las nubes de puntos, además del ruido inherente de los sensores utilizados, es necesario realizar un proceso de *selección* de los datos de interés, no solo para que el *proceso de registro* pueda converger al valor óptimo, sino también para reducir el tiempo que tarda el algoritmo en completar el procesamiento.

Una vez que se tienen los datos seleccionados de cada una de las nubes de puntos a alinear, es necesario encontrar en ellas las correspondencias. Es decir, encontrar un conjunto de puntos (los cuales suelen ser los *keypoints*) en la nube de puntos fuente que se pueden *identificar como los mismos punto*s en la nube de puntos objetivo.

Es habitual que existan correspondencias consideradas como tales que en realidad son desajustes debidos al cambio de punto de vista, entre otros la oclusión (Barazzetti, 2018). Estos desajustes suelen ser suficientes para arruinar los métodos de estimación tradicionales. Por lo tanto, es necesario eliminar o reducir la influencia indebida de los desajustes. Así, luego del pareo de los puntos característicos se deben *rechazar* las correspondencias no válidas para reducir el número de valores atípicos.

Finalmente, con las correspondencias filtradas, se busca la transformación que se ajusta mejor a ambas nubes de puntos mediante un proceso conocido como *alineación.*

A continuación, se desarrollan los pasos nombrados anteriormente.

Selección

Como la información suele ser redundante y lo suficientemente densa como para necesitar una gran capacidad de cómputo a la hora de procesar los datos, es necesario filtrar las nubes de puntos quitando la información irrelevante, reduciendo así el tiempo de ejecución de los algoritmos. Si bien existen en la literatura muchos criterios para decidir cuáles puntos tomar y cuáles no, en principio se pueden distinguir dos métodos de reducción de datos, siendo el primero aquel que extrae automáticamente un pequeño conjunto de *keypoints* únicos y repetibles, mientras que el otro se basa en muestrear los datos originales con respecto a una distribución objetivo deseada. Si bien el primer método está destinado a la alineación inicial basada en características (García Merino, 2016), el segundo se puede utilizar para reducir de manera eficiente la cantidad de da-

tos para los algoritmos de registro iterativos. A continuación, se presentan dos métodos que suelen emplearse en este tipo de tareas, incluidas dentro de PCL

- *Normal space sampling*: Al tratar generalmente con modelos suaves con pequeñas irregularidades (por ejemplo, un plano), el proceso puede resultar en el muestreo de muchos puntos que contienen esencialmente la misma información en términos de vectores normales. Por ello, la estrategia de *normal space sampling* pretende dar uso a esta información (Rusinkiewicz, 2001), y la misma consta de dos pasos, primero el de agrupar puntos en "cubos" de acuerdo con los ángulos entre sus vectores normales (considerados en la esfera unitaria) y los ejes de coordenadas, y luego muestrear uniformemente sobre los cubos resultantes, proporcionando un submuestreo de los puntos con más vectores normales "frecuentes". Dentro de PCL, este algoritmo se encuentra implementado en el método *pcl::NormalSpaceSampling*.

- *Harris 3D*: El operador de Harris (Harris, 1988) refiere a un detector de puntos de interés para imágenes. Es una técnica popular debido a su fuerte invariancia a la rotación, escala, variación de iluminación y ruido de imagen (Schmid, 2000). El detector de Harris se basa en la función de autocorrelación local de una señal que mide los cambios locales de la señal con parches desplazados una pequeña cantidad en diferentes direcciones. Dicho detector se ha utilizado en muchas aplicaciones en procesamiento de imágenes y visión artificial por su sencillez y eficiencia. Sin embargo, el problema con los datos 3D es que la topología es arbitraria y no está claro cómo calcular las derivadas. Para solucionar este problema, se propone transformar a los puntos de la nube de la siguiente manera (Sipiran, 2011):

a. Por cada punto de la nube, se define un punto vecino del mismo, y se calcula el centroide de este último.
b. Todos los puntos de la nube se trasladan para que el centroide coincida con el origen de coordenadas.
c. Luego, se calcula un plano de ajuste a los puntos trasladados.
d. Se aplica el *análisis de componentes principales* (PCA de sus siglas en inglés) (Jollife, 2016) al conjunto de puntos y se elige el *eigenvector* con el eigenvalue asociado más bajo como la normal del plano de ajuste, siendo el *eigenvector* o vector característico de una transformación lineal un vector distinto de cero que cambia como máximo en un factor escalar cuando se le aplica esa transformación lineal, mientras que el *eigenvalue* correspondiente es el factor por el cual se escala el vector propio.
e. Se rotan los puntos hasta que la normal al plano coincide con el eje z.
f. El plano XY resultante se utiliza para calcular las derivadas. Estas derivadas se calculan utilizando una superficie cuadrática de seis términos (paraboloide) ajustada al conjunto de puntos transformados.

Dentro de PCL, en el método *pcl::HarrisKeypoint3D* se encuentra una implementación de dicho algoritmo.

Estimación de correspondencias

La estimación de correspondencias es el proceso de emparejar los puntos p_i desde la nube de puntos fuente *P* con sus vecinos más cercanos q_j en la nube objetivo *Q*.

En PCL, con la función *pcl::registration::DetermineCorrespondences* se obtiene el conjunto de pares de correspondencia encontrados entre las nubes de puntos fuente y objetivo. Cada par consta del índice del punto en la nube de origen y el índice de la coincidencia encontrada en la nube de puntos de destino.

En el caso de datos de entrada provenientes de sensores que cumplen con el modelo de cámara estenopeica (Kaehler, 2017), el procedimiento de estimación de correspondencia puede acelerarse significativamente con la contraparte de perder algo de pre-

cisión. Estos sensores incluyen cámaras RGB-D populares como Microsoft Kinect, y recopilan información del entorno en forma de imágenes de profundidad y color. Para búsquedas de vecinos más cercanos en el espacio 3D, es posible utilizar la naturaleza proyectiva de las imágenes de profundidad para obtener una aproximación razonable. Cada punto de la nube corresponde a un *píxel* en la imagen de profundidad, lo que permite proyecciones desde los puntos de origen hasta el plano de la cámara del marco de destino mediante el uso de los parámetros propios de la cámara (Kaehler, 2017). Este enfoque es rápido, pero impreciso para nubes de puntos con grandes discontinuidades de profundidad o para cuadros que están muy alejados entre sí. Es por eso que se recomienda usar este método solo después de que las dos nubes de puntos se hayan acercado, característica que lo postula como buen algoritmo para alinear nubes de puntos consecutivas en una secuencia grabada a alta velocidad de cuadros.

Rechazo de correspondencias

Dado que las correspondencias no válidas pueden afectar negativamente los resultados del registro, la mayoría de los procesos de registro presentan un paso de rechazo. El mismo consiste en filtrar los pares de puntos emparejados en la etapa anterior para facilitar el algoritmo de estimación de la transformación hacia la convergencia al mínimo global. Este paso puede aprovechar la información auxiliar disponible de las nubes de puntos de entrada, como las normales de superficie locales o las estadísticas sobre las correspondencias. A continuación, se detallan algunos de los algoritmos más utilizados:

- Rechazo de correspondencias basado en la distancia: en base a un umbral, se filtran las correspondencias que estén a una distancia mayor que dicho límite. En PCL, se implementa mediante *pcl::registration::CorrespondenceRejectorDistance.*

- Rechazo en base a la distancia mediana: a diferencia del método anterior, en este caso el umbral se calcula en base a la mediana de todas las distancias punto a punto en las correspondencias estimadas inicialmente. Se utiliza la mediana ya que suele ser más efectiva en reducir la influencia de valores atípicos. El método *pcl::registration::CorrespondenceRejectorMedianDistance* implementa dicha operación.

- Rechazo basado en RANSAC: Este método aplica el RANdom SAmple Consensus (Fischler, 1981) para estimar una transformación para subconjuntos del conjunto dado de correspondencias y elimina las correspondencias atípicas basadas en la distancia euclidiana entre los puntos después de que la transformación calculada se aplica a la nube de puntos fuente. Es muy eficaz para evitar que el algoritmo ICP converja en mínimos locales, ya que siempre produce correspondencias ligeramente diferentes y es bueno para filtrar valores atípicos. Además, proporciona buenos parámetros iniciales para la estimación de la transformación con todas las correspondencias internas que siguen. El mismo cuenta con el método de PCL *pcl::registration::CorrespondenceRejectorSampleConsensus*.

- Rechazo basado en la compatibilidad normal: este filtro usa la información normal sobre los puntos y rechaza aquellos pares que tienen normales inconsistentes, es decir, el ángulo entre sus normales es mayor que un umbral dado. Puede rechazar pares erróneos que parecen correctos cuando se juzgan solo por la distancia entre los puntos. El mismo es implementado por *pcl::registration::CorrespondenceRejectorSurfaceNormal.*

Alineación

A lo largo de los años, ha habido numerosos enfoques matemáticos para resolver la transformación rígida T que minimiza el error de los pares de puntos. T se compone de una rotación R y una traslación t. Tener en cuenta que, a continuación, cuando se hace referencia a una transformada T y un punto p, se utilizarán coordenadas homogéneas.

Hay dos métricas de error principales que se deben minimizar y que se han considerado en la literatura: punto a punto y punto a plano, donde (p_k, q_k) es el *k-ésimo* de las correspondencias de N pares desde la nube de origen a la nube de destino.

- Métrica de error estándar de punto a punto: La métrica de error estándar utilizada en el algoritmo ICP es la métrica de error de punto a punto. Fue mencionado por primera vez por Arun (Arun, 1987); los investigadores propusieron varias formas de minimizarlo, seguidas de la introducción del algoritmo ICP (Besl, 1992). Eggert *(*Eggert, *et al*,1997) evaluó cada uno de estos métodos en términos de estabilidad numérica y precisión, llegando a la conclusión de que tienen un desempeño cercano. PCL ofrece una implementación (*pcl::registration::TransformationEstimationSVD*) utilizando descomposición en valores singulares (SVD) (Horn, 1987).

- Métrica de error de punto a plano: Chen y Medioni (Chen, 1992) introdujeron la métrica de punto a plano y demostraron que es más estable y converge más rápido que el enfoque anterior. Utiliza la distancia entre el punto de origen p_k y el plano descrito por el punto de destino q_k y su normal de superficie local n_{qk}. A diferencia de la métrica punto a punto, no tiene una solución de forma cerrada, por lo que la minimización se realiza con solucionadores no lineales (como Levenberg-Marquadt), o linealizándolo (Low, 2004) (bajo el supuesto de ángulos de rotación pequeños, es decir, sinθ θ y sin≃ 1). Dep≃ndiendo de la superficie subyacente y la distribución de puntos, el uso de la métrica de error de punto a plano puede ser considerablemente más robusto. Dicha funcionalidad se emplea en *pcl::registration::TransformationEstimationPointToPlane*.

- Métrica de error punto-a-plano ponderada: Asignar un peso diferente a cada correspondencia puede mejorar la convergencia. La ponderación de los pares de puntos puede verse como un rechazo de correspondencia suave, ajustando la influencia de los puntos correspondientes ruidosos en el proceso de minimización. La ponderación puede ser una función de la distancia punto a punto o punto a plano entre los puntos, una función del ángulo entre las normales correspondientes a los puntos, o una función del modelo de ruido del sensor que se ha usado.

Parte experimental

Para poder recolectar datos, se optó por realizar las pruebas de los algoritmos en base al entorno de simulación Gazebo, utilizando el modelo de un robot terrestre conocido, siendo este el ROSbot 2.0. El mismo cuenta, entre otras cosas, con una IMU 9DOF, un LIDAR 2D y una cámara RGB-D.

Una vez con la nube de puntos de profundidad, el próximo paso refiere a la generación del mapa y la localización del robot en el entorno, utilizando el denominado *registration pipeline*. A continuación, se desarrollan los pasos realizados a estas nubes de puntos, los cuales pueden verse en la Figura 3.

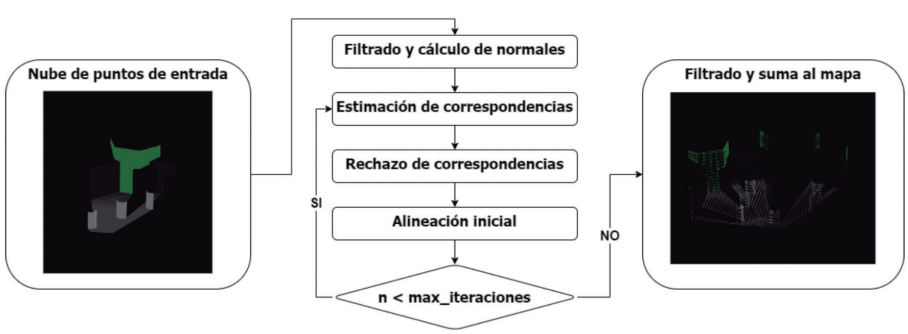

Fig. 3. Diagrama en bloques del pipeline realizado para el registro de nube de puntos tridimensionales y su posterior actualización del mapa 3D

Etapa de selección de datos

Debido a que hay una gran cantidad de puntos a procesar en cada nuevo conjunto de datos, el costo computacional resulta ser muy alto por cada corrida del algoritmo. Para ello, lo primero que se hace es un filtrado de dichos datos. Es importante conservar los bordes ya que describen los límites de los objetos (Lejeune, 2011), por lo que el filtro bilateral es una buena opción. En PCL es implementado mediante *pcl::FastBilateralFilter.*

Como se vio en el Marco Teórico, las normales de los puntos ayudan no solo en la etapa de rechazo de correspondencias, sino también pueden ser útiles en la etapa de filtrado, como es el caso de normal space sampling. Debido a que los datos originales son del tipo *pcl::PointXYZRGB*, es necesario calcular dichas normales. Un método rápido para hacer esto es mediante el uso de imágenes integrales (Holzer, 2012). La ventaja de este método es que solo requiere una etapa de preprocesamiento lineal al tiempo que permite calcular los vectores medios y las normas dentro de un área rectangular de la imagen en tiempo constante. Por lo tanto, en primera instancia mediante la implementación de PCL *pcl::IntegralImageNormalEstimation* se calculan las normales, para luego realizar el normal *space sampling.*

Dado que, el algoritmo de normal *space sampling* requiere de una nube de puntos ordenada, luego de este proceso se procede a la remoción de los datos inválidos de las nubes de puntos, mediante la implementación de PCL *pcl::removeNaNFromPointCloud.*

Estimación de correspondencias

Una vez obtenidos los puntos filtrados y con sus respectivas normales (dando lugar entonces a puntos del tipo *pcl::PointXYZRGBNormal),* se procede al cálculo de las correspondencias entre las nubes de puntos.

Rechazo de correspondencias

Al contar tanto con los datos de la posición como de la normal de cada punto, se emplean dos filtros de correspondencias

- uno para filtrar la distancia mediante el uso de la mediana, dando lugar al *pcl::registration ::CorrespondenceRejectorMedianDistance*, y

- un segundo filtro tomando en cuenta las superficies normales, esto es, *pcl::registration::Corre spondenceRejectorSurfaceNormal*, para rechazar pares de puntos con normales no coincidentes.

Alineación

Luego, con las correspondencias restantes se procede al cálculo de la transformación que responde a las dos nubes de puntos. Para ello, se utiliza el principio de punto a plano ponderado, debido a su rápida convergencia y buena precisión comparado con el método de punto a punto. Por ello, utilizando la función de PCL *pcl::registration::TransformationEstimationPointToPlaneWeighted* se llega a la transformación que describe el movimiento realizado por el robot.

Refinamiento y actualización

Debido a los posibles mínimos locales encontrados en lugar del mínimo global, para poder encontrar la transformación adecuada, se itera en los pasos de estimación de correspondencias, el rechazo de aquellas no válidas y el cálculo de la transformada. Una vez realizada la transformación de la nube de puntos fuente (filtrada y con sus respectivas normales) en base al resultado anterior, se calculan dichos valores nuevamente.

Para finalizar, con la transformación calculada se la aplica a la nube de puntos fuente para reducir su tamaño luego de un nuevo filtrado y agregarla al mapa generado en base a las nubes de puntos previas. En PCL se realiza una suma de dicha nube de puntos transformada con el mapa y se consigue el objetivo.

Resultados

Para realizar las simulaciones se tomó el modelo del robot terrestre Rosbot 2.0 (Figura 4), que cuenta con una Microsoft Kinect y con un LIDAR 2D, IMU y *encoders.*

Fig. 4. Rosbot 2.0

El mismo fabricante proporciona mapas simulados para la realización de pruebas, y con el agregado de los mapas presentes en el robot *Turtlebot3* se consigue una cantidad suficiente de entornos para ensayar el algoritmo descrito. Para el caso, se evaluó el mapa *turtlebot3 world*, que se observa en la Figura 5.

Luego, se ubicó el robot seleccionado en el mapa y se lo movió sin un patrón específico, tratando de que pueda mapearse todo el lugar. Se guardaron los tópicos de interés en un contenedor de ROS (*rosbag*) para finalmente, procesar y evaluar la misma información y ajustar el algoritmo. En base a esto, se fue construyendo el mapa con cada una de las nubes de puntos entrantes, comparando a cada una con la nube de puntos anterior y sumando la nueva nube transformada al mapa generado, mediante el *registration pipeline.* En base a esto, se consiguieron los resultados que se observan en la Figura 6, donde la traza roja representa a la trayectoria real del robot (dato proporcionado por la simulación), mientras que la trayectoria azul corresponde a la trayectoria estimada por el algoritmo realizado.

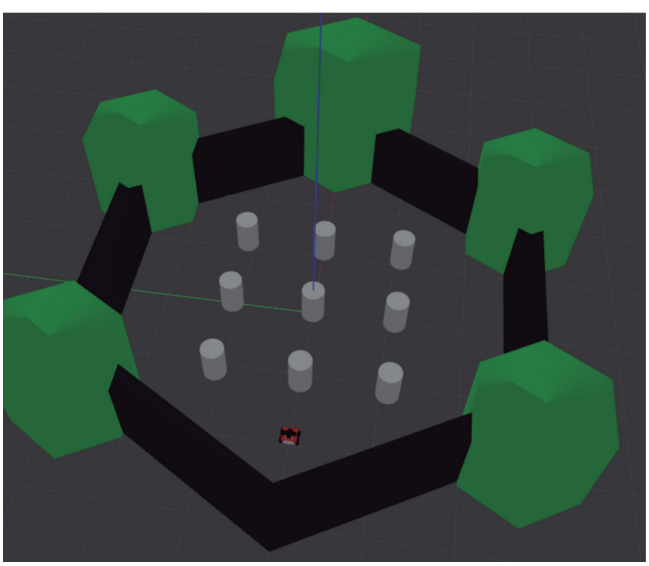

Fig. 5. Entorno simulado turtlebot3 world

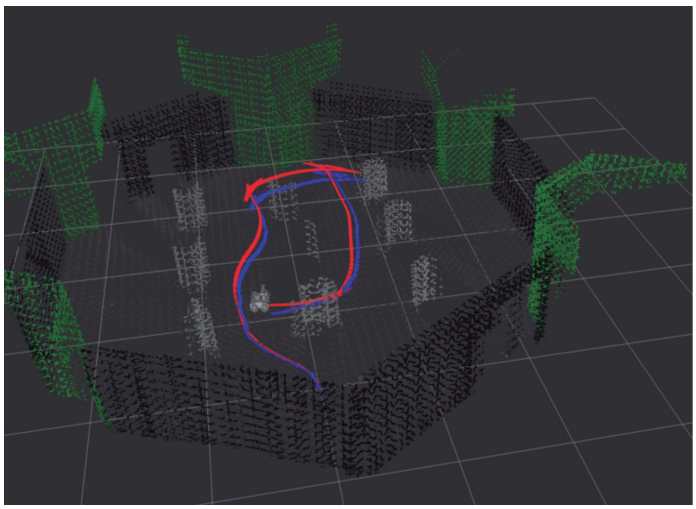

Fig. 6. Mapa reconstruido en base a la cámara RGB-D del entorno simulado, junto a la trayectoria real y estimada por el algoritmo

Como el tiempo de procesamiento en una computadora de 4 núcleos (2,0 GHz) ronda el segundo, se redujo la tasa de ejecución del contenedor de ROS para poder procesar todas las nubes de puntos, ya que se consiguen datos cada medio segundo.

Conclusiones

En el trabajo presentado se desarrolló un algoritmo de SLAM en base a los datos de una cámara RGB-D, con la capacidad de poder reconstruir un mapa, obteniendo las poses relativas de cada iteración. Dicho algoritmo no incluye el cierre de ciclo (Castro, 2016), pero en un futuro se pretende incorporarlo. El hecho de haber desarrollado un algoritmo propio de SLAM permite escalarlo al mapeo colaborativo entre múltiples robots.

Si bien es cierto que se realizaron las mediciones en base a simulaciones, el hecho de haber incorporado ROS permite que el código pueda adaptarse fácilmente no solo a un entorno real, sino también a cualquier robot, permitiendo entonces una gran flexibilidad a la hora de utilizarlo.

Respecto de los tiempos de procesamiento del algoritmo, son suficientes para permitir el movimiento de robots en velocidades moderadas. En cambio, si se pretende utilizar dicho algoritmo a una gran velocidad, es posible que no puedan obtenerse los resultados esperados, debido a que el algoritmo espera que la nube de puntos anterior no esté muy alejada de la nube de puntos actual.

Referencias

ARUN, K.; HUANG, T.; BLOSTEIN, S., (1987) Least- squares fitting of two 3-D point sets, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), vol. 9, n. 5, p. 698–700.
AUSTIN, R., (2010) Unmanned Aircraft Systems. John Wiley & Sons Ltd.
BARAZZETTI, L., (2018) Point cloud occlusion recovery with shallow feedforward neural networks, Advanced Engineering Informatics.
BESL, P.; MCKAY, N., (1992) A method for registration of 3-D shapes, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), vol. 14, n. 2, p. 239–256.
CARON, F.; DUFLOS, E.; POMORSKI, D.; VANHEEGHE, P., (2010) GPS/IMU data fusion using multisensor Kalman filtering: introduction of contextual aspects, Information Fusion.
CASTRO, G., (2016) Detección y Cierre de ciclos en sistemas SLAM basados en visión estéreo, Buenos Aires, Argentina.
CHEN, Y.; MEDIONI, G., (1992) Object modelling by registra- tion of multiple range images, Image Vision Comput., vol. 10, n. 3, p. 145–155.
DEL ROSARIO, M.; LOVELL, N.; REDMOND, S., (2010) Quaternion-based complementary filter for attitude determination ofa smartphone, IEEE Sensors Journal.
EGGERT, D.; LORUSSO, A.; FISCHER, R., (1997) Estimating 3-D rigid body transformations: a comparison of four major algorithms, Mach. Vision Appl., vol. 9, n. 5-6, p. 272–290.
FISCHLER, M.; BOLLES, R., (1981) Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography, Commun. ACM, vol. 35, n. 6, p. 381–395.
GARCIA MERINO, J., (2016) Sistema avanzado de detección de obstáculos y navegación autónoma para vigilancia y protección basado en flota de vehículos aéreos no tripulados, Universidad de Málaga.
GARCIA, F., (2017) Tools for 3D Point Cloud Registration, Doctoral Program in Technology, Girona.
HARRIS, C.; STEPHENS, M., (1988) A combined corner and edge detection, 4th Alvey Vision Conference, p. 147-151.
HOLZER, S.; RUSU, R.; DIXON, M.; GEDIKLI, S.; NAVAB, N., (2012) Real-Time Surface Normal Estimation from Organized Point Cloud Data Using Integral Images, IEEE/RSJInternational Conference on IntelligentRobots and Systems (IROS), Vila Moura, Algarve, Portugal.
HORN, B., (1987) Closed-form solution of absolute ori- entation using unit quaternions, Journal of the Optical Society of America A, vol. 4, n. 4, p. 629–642.
JOLLIFE, I.; CADIMA, J., (2016) Principal component analysis: A review and recent developments, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences.
KAEHLER, A.; BRADSKI, G., (2017) Learning OpenCV3: Computer vision in C++ with the OpenCV library, O'Reilly Media, Inc.
KOENIG, N.; HOWARD, A., (2004) Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator, IEEE/RSJ International Conference on Intelligent Robots and Systems, p. 2149-2154, Sendai, Japan.
LEJEUNE, A.; PIERARD, S.; VAN DROOGENBROECK, M.; VERLY J., (2011) A new jump edge detection method for 3D cameras, International Conference on 3D Imaging (IC3D).
LOW, K., (2004) Linear least-squares optimization for point- to-plane ICP surface registration, Technical Report TR04-004, Department of Computer Science, University of North Carolina at Chapel Hill.
OPEN ROBOTICS, pcl_conversions Disponible en http://wiki.ros.org/pcl_conversions.
QUIGLEY, M.; CONLEY, K.; GERKEY, B. P.; FAUST, J., (2011) ROS: an open-source Robot Operating System, ICRA Workshop on Open Source Software. ROHMER, E.; SINGH,

S.; FREESE, M., (2013) V-REP: A versatile and scalable robot simulation framework, 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems.
RUSINKIEWICZ, S.; LEVOY, M., (2001) Efficient variants of the ICP algorithm, Proceedings Third International Conference on 3-D Digital Imaging and Modeling.
RUSU, R.; COUSINS, S., (2011) 3D is here: Point Cloud Library (PCL), 2011 IEEE International Conference on Robotics and Automation.
SCHMID, C.; MOHR, R.; BAUCKHAGE, C., (2000) valuation of interest point detectors, Int. J. Comput. Vis., vol. 37, n. 2, p. 151–172.
SIPIRIAN, I.; BUSTOS, B., (2011) Harris 3D: A robust extension of the Harris operator for interest point detection on 3D meshes, The Visual Computer.
WEINMANN, M., (2016) Reconstruction and Analysis of 3D Scenes, Springer.