AJEA - Actas de Jornadas y Eventos Académicos de UTN
VI Jornadas de Intercambio y Difusión de los Resultados de Investigaciones de los Doctorandos en Ingeniería
DOI: https://doi.org/10.33414/ajea.1088.2022

Mapeo consistente entre poses mediante ICP en GPU embebida

Consistent interpose mapping through ICP on embedded GPU

Presentación: 4 y 5 de Octubre de 2022

Doctorando/a:
Martin Nievas
Universidad Tecnológica Nacional, Facultad Regional Córdoba - Argentina
martin.nievas.ar@gmail.com

Director/a:
Gastón R. Araguás

Codirector/a:
Claudio J. Paz

Resumen

La habilidad de un robot para generar un mapa del ambiente que lo rodea y localizarse en el mismo es un problema ampliamente explorado en la comunidad robótica. En este trabajo se plantea un algoritmo de SLAM orientado a plataformas con GPU embebida, capaz de generar un mapa 2D del ambiente utilizando mediciones de odometría y LIDAR 2D. Para determinar las correspondencias entre las mediciones laser se utiliza un algoritmo ICP implementado en CUDA, el cual utiliza la GPU embebida presente en el robot. Se presentan resultados y comparaciones experimentales.

Palabras clave: SLAM, GTSAM, 2D-LIDAR.

Abstract

The ability of a robot to generate a map of the environment is a problem widely explored in the robotic community. We present a SLAM algorithm oriented to platforms with embedded GPU. It generates a 2D map of the environment based on odometry and 2D Lidar measurements. An ICP algorithm developed in CUDA is used to compute the correspondence between the laser measurements. Experimental results and comparisons are presented against algorithms widely used in robotics.

Keywords: SLAM, GTSAM, 2D-LIDAR

Introducción

Para que un robot autónomo pueda operar en un ambiente real, una de las habilidades más importantes es poder aprender una representación del ambiente que lo rodea y localizarse en el mismo. Dependiendo del método de representación para el ambiente podemos encontrar mapas topológicos (Akdeniz & Bozma, 2015; Blochliger et al., 2018), mapas métricos 2D (Hess et al., 2016; Mobarhani et al., 2011), como también mapas métricos 3D (Hornung et al., 2013; Selin et al., 2019). Una vez que el robot construye una representación del ambiente, tiene que estimar su posición dentro de la misma, en base a las mediciones de sus sensores. Este proceso conocido como localización, puede ser resulto simultaneamente al proceso de construcción del mapa, también conocido como SLAM (por las siglas en ingles de Simultaneous Localization and Mapping).

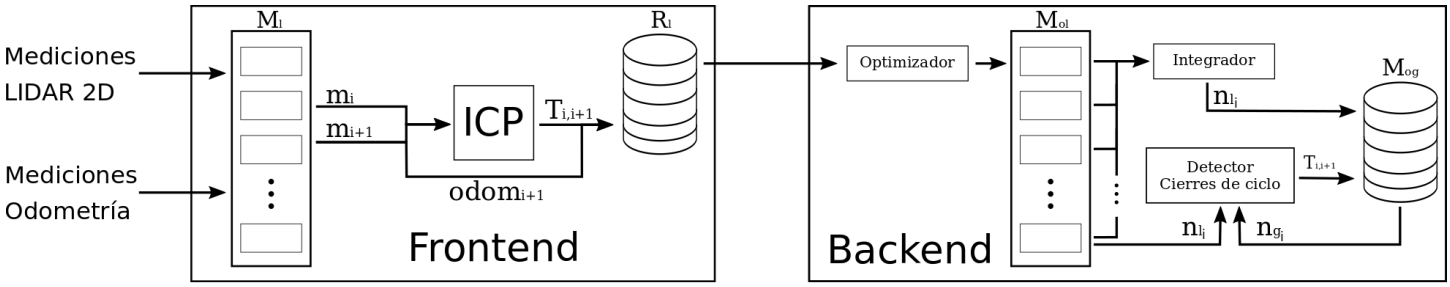

Fig. 1: Diagrama general del esquema SLAM implementado. A la izquierda el frontend encargado de fusionar las mediciones de odometría y laser 2D. A la derecha el backend encargado de la optimización y cierres de ciclos

Comúnmente los algoritmos de SLAM están compuestos por dos partes principales, un frontend y un backend. El primero está enfocado a estimar la posición del robot en tiempo real, mientras que el segundo se encarga de optimizar las poses del robot en base a las restricciones generadas por el frontend. La implementación de estas partes varía dependiendo si se utiliza un enfoque basado en un filtro del Kalman extendido (Bailey et al., 2006), filtro de partículas (Montemerlo et al., 2003) o filtro Rao-Blackwellized (Murphy, 1999), por mencionar algunos. La implementación también estará definida por el tipo de sensor que se utilice, siendo los sensores LIDAR ampliamente utilizados tanto en esquemas 2D (Grisetti et al., 2007), (Murphy, 1999), (Hess et al., 2016), como en sistemas 3D (Kohlbrecher et al., 2011). Para una revisión más en detalle de los diferentes esquema SLAM, incluyendo V-SLAM (SLAM visual), VI-SLAM (SLAM visual e inercial), entre otros, se recomienda al lector mirar (Cadena et al., 2016).

Otras implementaciones de SLAM introducen el concepto de grafos (Franchi et al., 2009; Palacios et al., 2017, 2017), en el cual el ambiente es representado por nodos conectados mediante aristas. Cada nodo representa una posición válida en el ambiente, mientras que los vértices corresponden a restricciones entre dos nodos sucesivos. Estas restricciones pueden provenir de las mediciones de un sensor, o de un evento externo. En (Palacios et al., 2017) se presenta un algoritmo de exploración mediante grafos que maximizan la cobertura de mapa en el proceso. El método está basado en el principio de SRT (por las siglas en ingles de Sensor baser Random Tree) para construir un grafo del ambiente a explorar, considerando puntos aleatorios. Una de sus ventajas es la acumulación de conocimiento a través del concepto de control de bordes, que almacena información acerca de las áreas que el robot deja atrás durante el proceso de exploración, y necesitan ser re visitadas. Esta característica, sumada con la generación del mapa en forma de grafo, permite un regreso óptimo hacia las áreas no exploradas. A diferencia de los algoritmos anteriores, en (Palacios & López, 2019) se presenta un método de exploración basado grafos, que define una forma sistemática de analizar la siguiente posición a explorar, eliminando la aleatoriedad en el proceso de decisión. Con esto se consigue minimizar los movimientos que el robot tiene que realizar para alcanzar la posición y por consiguiente el tiempo que demora en llegar a ese lugar.

En este trabajo se presentan los resultados preliminares de un sistema SLAM 2D utilizando un sensor LIDAR 2D y mediciones de odometría. El sistema propuesto utiliza un algoritmo de ICP (determinación del punto más cercano por sus siglas en ingles) implementado en el lenguaje CUDA, el cual hace utilización de plataformas con memoria unificada y GPU (Unidad de procesamiento gráfico por las siglas en inglés) embebida como las Jetson de NVIDIA™. Para la etapa de optimización, se utiliza la biblioteca GTSAM, la cual fusiona las mediciones del laser con las de odometría.

Desarrollo

El sistema Frontend está compuesto por un módulo ICP que calcula la transformación entre dos nubes de puntos, para obtener el desplazamiento del robot entre las dos mediciones. Estas nube de puntos junto a las mediciones de odometría, componen los datos de entrada para el sistema Frontend. Para el algoritmo ICP se utilizó un esquema similar a (Bedkowski et al., 2013), utilizando el lenguaje de programación CUDA (Compute Unified Device Architecture en inglés) el cual permite programar una GPU embebida. Estas se encuentras disponibles en plataformas como las Jetson de NVIDIA™, en las cuales se puede aprovechar el uso de memoria unificada para acelerar el procesamiento de las mediciones (Nievas et al., 2020).

El sistema de backend está compuesto por dos partes como se puede observar en la Fig. 1. Por un lado el Frontend recibe las mediciones de odometría y LIDAR 2D, mediante las cuales se calcula el desplazamiento del robot y se contruye una lista de restricciones que luego serán enviadas al backend mediante mensajes del ROS, ampliamente utilizado en robótica.

Estas restricciones son luego fusionadas y optimizadas localmente en el Backend, para generar un submapa. En esta etapa se realiza la detección de ciclos, los cuales se determinan mediante una métrica de distancia utilizando las nubes de puntos de los submapas locales. Tanto para la fusión como la optimización, se utilizó la biblioteca GTSAM(Dellaert et al., 2022). Esta es una biblioteca C++, que permite la fusión de sensores utilizados en robótica, como también de aplicaciones en visión por computadora, incluida: SLAM (localización simultánea y mapeo), VO (Odometría visual) y SFM (estructura del movimiento). Esta biblioteca utiliza un esquema de representación basado en grafo de factores y redes Bayesianas, en lugar de matrices dispersas para optimizar la configuración más probable.

Frontend

Las lecturas provenientes del sensor LIDAR 2D son almacenadas en un buffer de mediciones locales M_l, cuyo tamaño puede ser configurado dependiendo la geometría del ambiente. En los experimentos realizados en ambientes interiores, valores entre 3 y 6 mediciones generaron buenos resultados. Este valor puede ser ajustado dependiendo la complejidad del ambiente y el alcance del sensor LIDAR 2D, para incrementar o reducir el tamaño de los mapas locales.

Junto a las mediciones laser $laser_i$, se almacena la posición actual estimada del robot $odom_i$, determinada por la odometría. Dado que las mediciones de odometría ocurren a intervalos cortos de distancia, se puede considerar que son lo suficientemente precisas para ser insertadas diréctamente en el mapa local. En el mapa global, estas mediciones acumulan error, por lo que es necesario determinar el error de la posición mediante las mediciones laser. Todas las mediciones de odometría se consideran que occuren en el marco de referencia *odom.*

Las mediciones de odometría son agregadas al buffer de mediciones locales M_l cuando el robot se desplaza una distancia *update_distance* o realiza una rotación con un ángulo *update_rotation*, definidos en un archivo de configuración. En este archivo también se define la cantidad máxima de iteraciones *max_iter* para el algoritmo ICP, y la cantidad de mediciones *active_submaps* que conforman un mapa local. Estos valores deben ser ajustados dependiendo del entorno a reconstruir.

Una vez que el buffer de mediciones locales M_l es completado, se recorre el mismo calculando la transformación entre cada par de mediciones $m_i, m_{i+1} \in M_l$ mediante el algoritmo ICP. El mismo está implementado en CUDA siguiendo un esquema similar a (Bedkowski et al., 2013). Considerando dos nubes de puntos $pc_i \in m_i$, $pc_{i+1} \in m_{i+1}$, el algoritmo inicialmente calcula una correspondencia utilizando el algoritmo de vecinos más cercanos implementado mediante radix-sort en la GPU. Luego, se calcula la distancia para cada una de las correspondencias de puntos, y se calcula la transformación $T_{i,i+1} \in m_{i+1}$ para las nubes de puntos pc_i y pc_{i+1} mediante reducción de mínimos cuadrados de forma iterativa. Como métrica para finalizar el algoritmo, se estableció la cantidad máxima de iteraciones. Finalmente se reporta la distancia

acumulada $d_{acc}(pc_i, pc_{i+1})$ entre las correspondencias calculadas. Las transformaciones $T_{i,i+1}$ junto a las mediciones de odometría $odom_i$ son almacenadas en un buffer de restricciones R_l que será enviado al Backend para la generación de los mapas locales y global. Estas restricciones son enviadas mediante mensajes implementados en ROS.

Backend

En esta etapa se reciben las restricciones R_i provenientes del Frontend, que contienen las mediciones de odometría y las transformaciones calculadas mediante el algoritmo ICP. Estas restricciones son utilizadas por el algoritmo de optimización GSTAM para definir los nodos ($odom_i$) y aristas ($T_{i,i+1}$) de un grafo que representa las mediciones. El algoritmo determina la configuración más probable para las mediciones y transformaciones, teniendo en cuenta el modelo de ruido de cada una. Si bien el sistema puede manejar mediciones de odometría con diferentes valores de incertidumbre, en el trabajo actual se considera que las mediciones tienen la misma incertidumbre ya que se realizan a intervalos cortos de distancia en el mapa local. De la misma forma para las transformaciones $T_{i,i+1}$ calculadas por el algoritmo de ICP, se considera que las mismas tienen una incertidumbre constante. Si bien el algoritmo de ICP permite estimar el error de la transformación calculada, no es tenido en cuenta en este trabajo y se incluirá en un trabajo futuro.

Tanto para el algoritmo de Frontend como para el de optimización, se considera que las mediciones son realizadas en el marco de referencia *odom,* por lo que el mapa optimizado será en este mismo marco de referencia. Esto es importante destacar, ya que para un caso futuro, donde la construcción del mapa se realice con múltiples robots, todos tendrían que compartir el mismo marco de referencia para el mapa global fusionado.

Una vez que las restricciones R_i son optimizadas mediante el algoritmo GTSAM, se obtiene una nueva lista de mediciones corregidas M_{ol} . Estas mediciones contienen las poses corregidas del robot $pose_{ol_i}$ para construir el mapa local LM_i utilizando las mediciones de laser $laser_i$ correspondiente. Con estas mediciones de laser $laser_i$ se genera una grilla de probabilidad de ocupación, tomando como origen la pose corregida del robot $pose_{ol_i}$. Para generar esta grilla, se utiliza el algoritmo de Breshenman, el cual realiza una operación de trazado de rayos por cada punto de la nube pc_i . Esto genera una grilla de ocupación por cada una de las mediciones corregidas M_{ol} . Para la fusión de estas grillas, se establece como origen de coordenadas la primer medición $m_{l_1} \in M_{ol}$ y luego se insertan los valores de las siguientes grillas. Los valores de cada celda son actualizados mediante promedio simple. Otros esquemas de fusión serán analizado un un trabajo futuro. Una vez calculada la grilla de ocupación local, las mediciones locales optimizadas M_{ol} son insertadas en una lista de mediciones globales M_{og} , las cuales serán utilizadas para determinar el mapa global construido por el robot.

Una parte importante en los algoritmos de SLAM es la detección y cómputo de cierres de ciclos. Los mismos implica determinar si el robot visitó anteriormente esta zona, y agregar una nueva restricción al grafo global. Los cierres de ciclos son importantes, ya que permiten reducir el error acumulado en el mapa global, pero también pueden llevar a incrementar el error si se establece una restricción errónea. Para el trabajo actual, los puntos candidatos para cierres de ciclos son calculados a partir de las últimas mediciones corregidas M_{ol} y la lista de mediciones globales M_{og} . Para determinar si un nodo $n_{l_i} \in m_{li}$ corresponde a un cierre de ciclo con otro punto $n_{g_i} \in m_{gi}$, el mismo tiene que estar dentro de un radio r_{loop} . Esta condición es necesaria, pero no suficiente ya los puntos podrían estar separados por una pared y dentro del radio de análisis. Una estrategia válida sería hacer un proceso de raycasting desde el punto n_{l_i} a n_{g_i} , y determinar si en el camino no se detectó ningún obstáculo. Dado que esta operación implica realizar mediciones en dos grillas de ocupación que no necesariamente están alineadas, se optó por una métrica de distancia a partir del algoritmo ICP, dejando el proceso de trazado de rayos para un trabajo futuro. La métrica utilizada consiste en estimar la transformación de las nubes de punto $p_{l_i} \in m_{li}$ y $p_{g_i} \in m_{gi}$, y obtener la distancia acumulada d_{acc} entre todas las correspondencias. Si d_{acc} es superior a un umbral previamente definido, se considera que los nodos no corresponden a un cierre de ciclo, y se descarta. Caso contrario, la transformación $T_{n_{l_i}, i+n_{g_i}}$ es agregada al grafo global como un cierre de ciclo, y se realiza un proceso de re optimización en el algoritmo GTSAM.

Resultados

Las pruebas fueron realizadas utilizando un robot diferencial equipado con un LIDAR 2D de 360° en el simulador ROS/Gazebo. En la Fig. 2 se puede observar el ambiente de oficina simulado sobre el cual se realizaron las pruebas de construcción del mapa. El mismo corresponde a una sala de 7x10m compuesta por tres pasillos separados por dos paredes.

Para realizar una comparación cualitativa del mapa generado, se evalúa el mapa generado por el algoritmo propuesto contra el mapa generado por el paquete *hector_slam* ampliamente utilizado en la comunidad robótica. Cabe destacar que el algoritmo *hector_slam* no implementa cierres de ciclo, por lo que para la comparación, los cierres de ciclo no son tenidos en cuenta para el algoritmo propuesto.

Como se puede observar en la Fig. 2, el algoritmo propuesto genera un mapa del ambiente muy similar al generado por *hector_slam*. En el mismo podemos ver que llegando al final del pasillo central, el robot no puede estimar una transformación correcta y el mapa presenta inconsistencias en la parte superior. Esto es debido a que el robot, al atravesar un portal o salir de un pasillo, encuentra nueva información que no puede ser comparada con lo que venía observando hasta el momento. Esto puede ser solucionado implementando cierres de ciclo con mediciones previas, utilizando las mediciones de odometría.

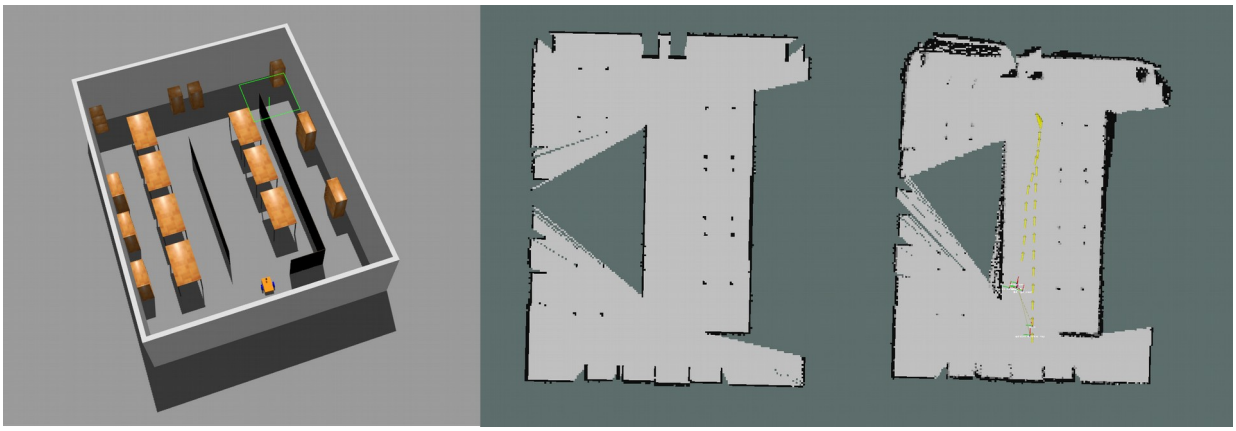

Fig. 2: Comparación del mapa generado. A la izquierda el ambiente utilizado para las simulaciones. Al centro el mapa generado por el paquete hector_slam. A la derecha el mapa generado por el algoritmo presentado. En amarillo se puede observar las diferentes posiciones del robot durante el proceso de mapeo.

Conclusiones

Se presentó un algoritmo que es capaz de generar un mapa métrico 2D de un ambiente inicialmente desconocido, a partir de mediciones LIDAR 2D y odometría. El esquema utilizado de frontend y backend resultó de gran utilidad para realizar un procesamiento simultaneo de las mediciones y cierres de ciclo. El mapa generado durante las pruebas muestra que al atravesar puertas o salir de pasillos, el robot adquiere nueva información que antes no estaba disponible, por lo que no puede establecer una correspondencia correcta y acumula error en ese nodo. Esto puede ser solucionado mediante estrategias de re visitar lugares con incertidubre y cerrar ciclos. Se plantea como trabajo futuro mejorar la métrica para identificar cierres de ciclos, incorporando información de los submapas locales generados. También se planea expandir el sistema para incorporar mediciones de cámaras RGB-D y generar mapas 3D del ambiente.

Referencias

Akdeniz, B. C., & Bozma, H. I. (2015). Exploration and topological map building in unknown environments. 2015 IEEE International Conference on Robotics and Automation (ICRA), 1079-1084.

Bailey, T., Nieto, J., Guivant, J., Stevens, M., & Nebot, E. (2006). Consistency of the EKF-SLAM algorithm. 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, 3562-3568.

Bedkowski, J., Majek, K., & Nüchter, A. (2013). General purpose computing on graphics processing units for robotic applications. Journal of Software Engineering for Robotics, 4(1), 23-33.

Blochliger, F., Fehr, M., Dymczyk, M., Schneider, T., & Siegwart, R. (2018). Topomap: Topological mapping and navigation based on visual slam maps. 2018 IEEE International Conference on Robotics and Automation (ICRA), 1-9.

Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., & Leonard, J. J. (2016). Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. IEEE Transactions on robotics, 32(6), 1309-1332.

Dellaert, F., Roberts, R., Agrawal, V., Cunningham, A., Beall, C., Ta, D.-N., Jiang, F., lucacarlone, nikai, Blanco-Claraco, J. L., Williams, S., ydjian, Lambert, J., Melim, A., Lv, Z., Krishnan, A., Dong, J., Chen, G., Chande, K., … roderick-koehle. (2022). Borglab/gtsam (4.2a7). Zenodo. https://doi.org/10.5281/zenodo.5794541

Franchi, A., Freda, L., Oriolo, G., & Vendittelli, M. (2009). The sensor-based random graph method for cooperative robot exploration. IEEE/ASME Transactions on Mechatronics, 14(2), 163-175.

Grisetti, G., Stachniss, C., & Burgard, W. (2007). Improved techniques for grid mapping with rao-blackwellized particle filters. IEEE transactions on Robotics, 23(1), 34-46.

Hess, W., Kohler, D., Rapp, H., & Andor, D. (2016). Real-time loop closure in 2D LIDAR SLAM. 2016 IEEE international conference on robotics and automation (ICRA), 1271-1278.

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., & Burgard, W. (2013). OctoMap: An efficient probabilistic 3D mapping framework based on octrees. Autonomous Robots, 34(3), 189-206.

Kohlbrecher, S., Von Stryk, O., Meyer, J., & Klingauf, U. (2011). A flexible and scalable SLAM system with full 3D motion estimation. 2011 IEEE international symposium on safety, security, and rescue robotics, 155-160.

Mobarhani, A., Nazari, S., Tamjidi, A. H., & Taghirad, H. D. (2011). Histogram based frontier exploration. Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on, 1128-1133.

Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B., & others. (2003). FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. IJCAI, 3, 1151-1156.

Murphy, K. P. (1999). Bayesian map learning in dynamic environments. Advances in neural information processing systems, 12.

Nievas, M., Paz, C. J., & Araguás, G. R. (2020). Paralelización del submuestreo de nube de puntos para plataformas con memoria unificada. 2020 IEEE Congreso Bienal de Argentina (ARGENCON), 1-7.

Palacios, A. T., & López, A. S. (2019). Extending the Limits of the Random Exploration Graph for Efficient Autonomous Exploration in Unknown Environments. En Path Planning for Autonomous Vehicles. IntechOpen.

Palacios, A. T., Sánchez L, A., & Bedolla Cordero, J. M. E. (2017). The random exploration graph for optimal exploration of unknown environments. International Journal of Advanced Robotic Systems, 14(1), 1729881416687110.

Selin, M., Tiger, M., Duberg, D., Heintz, F., & Jensfelt, P. (2019). Efficient autonomous exploration planning of large-scale 3-d environments. IEEE Robotics and Automation Letters, 4(2), 1699-1706.