VOLUMEN 16
XXVII Verano De la Ciencia
ISSN 2395-9797
www. jóvenesenlaciencia.ugto.mx

Reconstrucción tridimensional de interiores para navegación autónoma de robots móviles

Cesar de Jesús Vázquez Flores[1], Agustín Ramirez Manriquez[2], Eduardo Arath Zúñiga Bermúdez[3], Felipe de Jesus Trujillo Romero[4]

[1,2,3]Ingeniería en Mecatrónica - DICIS
cdj.vazquezflores@ugto.mx[1]
a.ramirezmanriquez@ugto.mx[2]
ea.zunigabermudez@ugto.mx[3]
[4]Depto. Ingeniería Electrónica - DICIS
fdj.trujillo@ugto.mx[4]

Resumen

En diversas tareas de robótica móvil se requiere tener un mapa del espacio de trabajo lo más detallado posible con la finalidad de que el robot realice la tarea asignada de manera eficiente. De hecho, una de esas tareas es que el robot pueda navegar de manera autónoma y generé el mapa del sitio explorado utilizando sensores de profundidad. Por esa razón, en este proyecto se realizó la implementación de un algoritmo de reconstrucción tridimensional para escenarios de interior donde un robot móvil pueda navegar usando la información que recibe del entorno. De esta manera se implementaron los algoritmos de Iterative Closest Points para la fusión de dos nubes de puntos y el algoritmo de Ball pivoting para la triangulación de la nube de puntos resultante de la fusión de todas las vistas de la escena de interés. La validación se realizó en un el entorno de simulación CoppeliaSim mediante un robot virtual y la implementación de los algoritmos se hizo en la librería Open3D usando Python como lenguaje de programación.

Palabras clave: Fusión de vistas 3D; Nubes de puntos 3D; Reconstrucción 3D; Robot móvil.

Introducción

Con el auge de la tecnología cada vez contamos con más y mejores sensores 3D. Estos sensores son cada vez más sensibles y con una resolución mayor que aquella que se puede obtener al usar un par estéreo de cámaras de video. Además, permiten obtener la modelización de objetos que no poseen textura. Esto es de gran utilidad, pero por otra parte surgen algunos inconvenientes. Dentro de los inconvenientes podemos mencionar que al tener una mejor resolución también se tiene una gran cantidad de puntos tridimensionales obtenidos. Y esto hace que los métodos para fusión de nubes de puntos 3D y de triangulación de mallas sean muy lentos. Esta restricción no permite que se utilicen dichos métodos de reconstrucción 3D en tiempo de ejecución. Ya que esto consumiría mucho tiempo máquina y en una aplicación en tiempo real, obviamente, el tiempo es vital para el buen procesamiento de la información recibida.

De forma general el proceso de modelización para la obtención de un modelo tridimensional se divide en tres partes principales:

1. Adquisición de datos tridimensional
2. Fusión de puntos de vista
3. Construcción de la malla

En la figura 1 se puede observar el diagrama de flujo de las diferentes etapas que se necesitan para poder obtener el modelo de una escena 3D. Cabe mencionar que en la lista precedente solo se mencionan tres ya que no todas las etapas mostradas en la figura 1 se llevan a cabo. Esto depende del tipo de sensor utilizado y de los criterios en los que se desee realizar el modelado 3D.

Por ejemplo, cuando se tiene una gran cantidad de puntos es muy difícil realizar la triangulación de la malla debido al tiempo que se va a requerir para ello. Sin embargo, existen técnicas de reducción de polígonos

como la decimación, que eliminan puntos reduciendo con ello el tiempo necesario para la obtención de la triangulación.

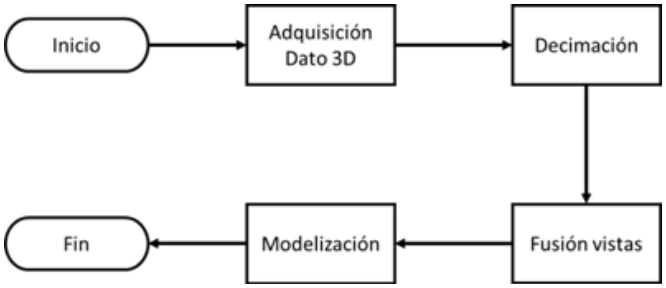


Figura 1. Etapas de la modelización.

Por otro lado, para realizar la reconstrucción de un objeto 3D, existen varios métodos. Entre ellos los más usados son: *Marching Cubes* (Lorensen & Cline, 1987), *Ball Pivoting* (Bernardini et al., 1999) y superficies deformables (Xu & Prince, 1997). Si bien es cierto que estos algoritmos presentan un buen desempeño también tienen algunos inconvenientes. Por ejemplo, *Marching Cubes* tiende a fusionar partes del objeto que están muy juntas y las unifica en una sola. Mientras que *Ball Pivoting* es capaz de resolver el problema anterior tiene el inconveniente de que es más lento. En el caso del algoritmo de superficies deformables sino se configuran bien sus parámetros es difícil obtener un resultado óptimo.

Métodos y materiales

En las siguientes secciones se presentan de manera breve los diferentes métodos y las herramientas usadas para la realización de este proyecto. Comenzaremos explicando el algoritmo de empleado para la fusión de diferentes vistas el cual se llama *Iterative Closest Points* (ICP). Posteriormente se hará mención del algoritmo usado para la triangulación de las nubes de puntos que se eligió para generar el modelo 3D y que fue *Ball Pivoting* (BP). También se mencionarán tanto la librería que nos permitió implementar los algoritmos de ICP y BP la cual fue la librería de Open3D; así como el entorno de simulación robótica usado para visualizar el comportamiento en la tarea de navegación dentro de un entorno virtual.

Fusión de nubes de puntos 3D

Uno de los algoritmos para fusionar nubes más utilizados es el presentado en los trabajos de Chen y Medioni (1992) y de Besl y McKay (1992) quienes desarrollaron el algoritmo de ICP. Debido a que este algoritmo es el principal y sobre él se realizan multitud de variantes y modificaciones, dentro de la literatura de modelización es conocido también como ICP Clásico. Este algoritmo se basa en el apareamiento de puntos entre dos nubes de puntos 3D en función de la minimización de la distancia euclidiana entre ellos. La función principal de este algoritmo es la alineación local entre dos nubes de puntos. Es decir, dados dos conjuntos de puntos 3D P y M la tarea es encontrar la transformación euclidiana que lleve a P a alinearse lo mejor posible con M. El algoritmo de ICP consiste principalmente en los siguientes pasos:

1. Buscar los pares para cada punto en P al punto más cercano en M.
2. Calcular la transformación que minimice el error cuadrático medio (MSE).
3. Aplicar la transformación encontrada a P y actualizar el MSE.
4. Iterar este proceso hasta que se converja en términos de MSE.

Con el fin de que este algoritmo pueda converger a un mínimo, casi cero, es necesario darle ciertos parámetros de entrada tal es el caso de la transformación inicial. Esta transformación inicial es una matriz en la cual se expresan la rotación y la translación existentes entre las dos nubes de puntos. Normalmente, esta matriz de trasformación inicial es una aproximación de los valores reales puesto que lo que se desea que realice el algoritmo de ICP es la obtención de estos parámetros.

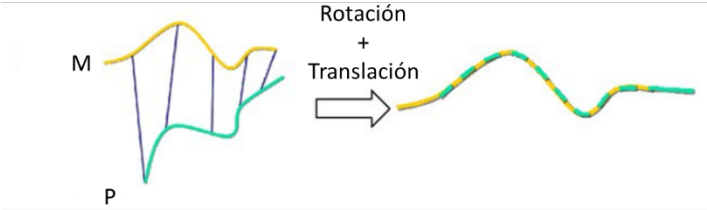


Figura 2. El algoritmo ICP alinea dos nubes de puntos correspondientes (M y P) al encontrar y aplicar una Rotación y Traslación específicas..

Este algoritmo presenta características tales que hacen su utilización, en la fusión de nubes de puntos, eficiente pero que no siempre converge. El algoritmo de ICP presenta problemas cuando el ángulo entre las dos nubes es muy grande o cuando las nubes están muy alejadas la una de la otra, es decir se necesita de una transformación inicial lo más precisa posible a fin de que el algoritmo pueda converger. Además, que no existe una manera de discriminar las diferentes partes de un objeto simétrico, p.e. un cilindro o una esfera, al momento de realizar la fusión lo cual genera que muchas partes queden traslapadas unas con otras. Sin embargo, y pese a ello al final se obtiene una fusión de los puntos que es aceptable y sobre todo útil para poder realizar otros procesos sobre la nube resultante. Como en el caso de la triangulación de la nube de puntos que se abordará en la sección siguiente.

Triangulación de nubes de puntos

Cuando se tiene una nube de puntos la cual tiene una gran cantidad de puntos es muy difícil realizar la triangulación de la malla. Esto debido a la cantidad de tiempo que va a requerir para poder triangular toda la nube. Existen técnicas de reducción de polígonos como la decimación descrita por Schroeder et al. (1992) Los algoritmos para reducción de puntos son funcionales, pero se corre el riesgo de perder información que pudiera servir en la reconstrucción. Lo ideal es reconstruir el modelo con todos los puntos que posee la nube. Por esa razón en este proyecto se utilizaron las nubes de puntos sin decimar.

Ahora bien, existen varios métodos para reconstrucción de un objeto 3D. Entre ellos el más famoso es el de Marching Cubes (Lorensen & Cline, 1987). Este es un algoritmo rápido, pero tiene el inconveniente de que al existir dos partes del objeto, que se está reconstruyendo, muy juntas las unifica en una sola. Existen mejoras y variantes de este algoritmo que si bien generan resultados aceptables siguen con algunos inconvenientes.

Otro algoritmo utilizado en la reconstrucción de objetos 3D es el de superficies deformables. Estos algoritmos utilizan una superficie geométrica que se debe deformar de tal manera que siga el contorno de la nube de puntos. La ventaja de modelizar con superficies deformables es que al adaptarse a la nube de puntos la modelización que se obtiene es buena. Pero tiene la desventaja de que si no se configuran bien sus parámetros es difícil de obtener un resultado óptimo.

Finalmente mencionemos, el algoritmo de Ball Pivoting (Bernardini et al., 1999) cuya finalidad es la reconstrucción de la superficie de un objeto de tres dimensiones usando una nube de puntos que conforman al objeto de interés. Este método se basa en un concepto simple. Suponiendo que *M* es la superficie de un objeto tridimensional y *S* una muestra puntual de *M*. Supongamos que *S* es tan denso como para que una bola o pelota de radio *p* no pueda atravesar la superficie sin tocar puntos de muestra. Se comienza colocando la bola *p* en contacto con tres puntos de muestra cualquiera de la nube de puntos. Cuidando que la bola *p* este en contacto con dos de los tres puntos donde está ubicada actualmente dicha bola. A partir de esa posición esta “girará” hasta tocar otro punto lo cual se repetirá hasta abarcar cada borde de los límites de la malla, en Figura 3 se muestran varios casos que son comunes. Cada triplete de puntos con las que la bola hará contacto formara triángulos, estos triángulos formados constituyen a la superficie de la malla.

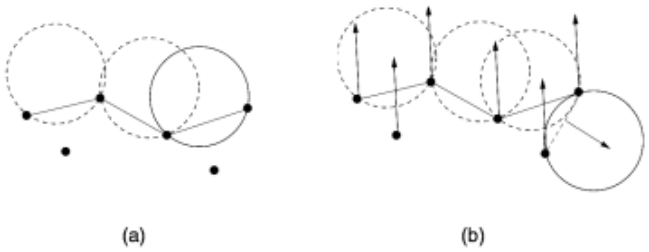


Figura 3. (a) Los puntos de muestra que se encuentran por debajo de la superficie no son tocadas por la bola p, por lo que no se tomaran en cuenta. (b) Debido a la falta de datos, la bola girara alrededor de un borde hasta que toque un punto que pertenezca a una parte diferente de la superficie.

Así que usando este algoritmo se procederá a reconstruir escenarios en tridimensionales con la finalidad que puedan ser usados por un robot para desarrollar la tarea de navegación.

Entorno de simulación

Para tener una visualización que nos permitiera apreciar el comportamiento del robot al momento de realizar el seguimiento de la trayectoria obtenida, fue necesario usara un simulador. Por esa razón se usó el entorno llamado CoppeliaSim (Rohmer et al., 2013), este entorno está desarrollado por la empresa Coppelia Robotics y fue utilizado para poder observar tanto el proceso de captura de una escena mediante un sensor RGB-D como para ver el robot durante su movimiento realizando la navegación en el entorno modelado.

Las ventajas que este entorno nos ofrece son: contar con un entorno tridimensional, un robot diferencial, colocar las nubes de puntos e importar el modelo 3D generado a partir de la triangulación de la nube de puntos adquirida.

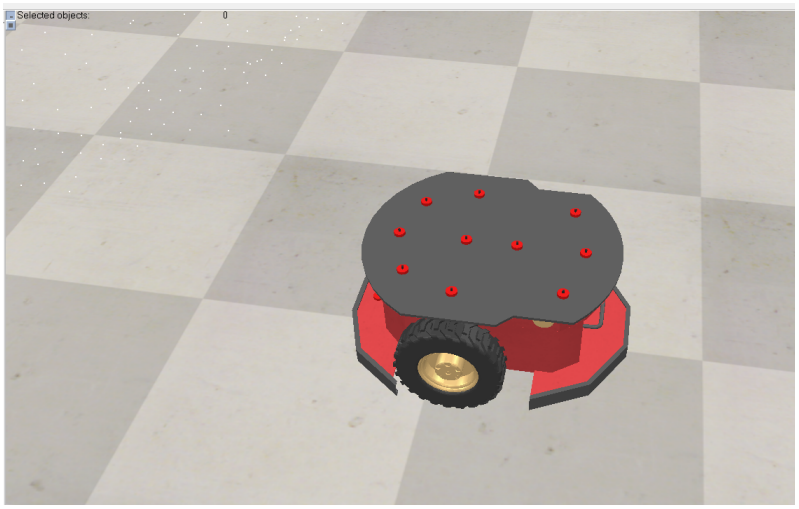


Figura 4. Entorno de Simulación CoppeliaSim con el robot PioneerP3DX.

Al poder agregar las diferentes mallas tridimensionales nos permitió observar cómo se comportaba el robot en un entorno tridimensional. Además, el robot utilizado, el Pioneer P3DX, posee varios sensores ultrasónicos que se utilizaron para este trabajo y que fueron de mucha utilidad para ver el comportamiento reactivo del robot.

Open3D

Open3D (Zhou et al., 2018) es una biblioteca de código abierto que admite el rápido desarrollo de software que trata con datos 3D. La interfaz de Open3D expone un conjunto de estructuras de datos y algoritmos cuidadosamente seleccionados tanto en C++ como en Python. Además de que su arquitectura interna está altamente optimizada y está configurada para poder utilizar paralelización de los algoritmos.

Las características principales de Open3D incluyen, el uso de diferentes estructuras de datos 3D, algoritmos de procesamiento de datos 3d, reconstrucción de escena, alineación de superficies, visualización de datos 3d, soporte de aprendizaje automático 3D con PyTorch y TensorFlow, aceleración de GPU para operaciones 3D básicas y se puede programar tanto en C++ como en Python.

Por las características antes mencionadas fueron las razones para utilizar esta librería de Open3D. Además de que al permitirnos programar en Python facilita la integración tanto con el entorno de simulación elegido como con otras librerías o códigos desarrollados en dicho lenguaje de programación.

Implementación

En los siguientes párrafos se muestra la implementación de los algoritmos mencionados en la sección anterior para realizar la triangulación de una nube de puntos 3D. En la parte de la implementación haremos uso de los sensores que proporciona el entorno de simulación para verificar y validar los diferentes algoritmos implementados.

Para comenzar con esta parte de implementación se tiene la adquisición de la nube de puntos. En este caso hacemos uso de un escenario creado en el entorno de simulación, dicho escenario se muestra en la figura 5. En esta figura 5, se puede observar una escena simple que contiene solamente una mesa, una silla, un anaquel y una planta; además, se tienen cuatro paredes y una puerta.

También se muestra el robot PioneerP3DX y arriba de este se colocó un sensor Velodyne que será el que nos permite en la vida real la adquisición de los datos tridimensionales de una escena ya sea de interior o de exterior. En este caso, se recuperara la información de la escena virtual.

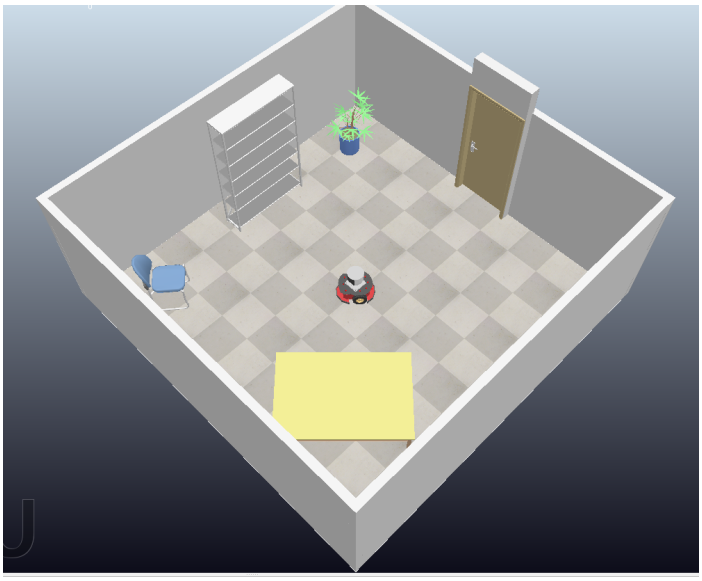


Figura 5. Entorno de Simulación CoppeliaSim con el robot PioneerP3DX.

A partir de la escena antes mencionada se comienza a generar los datos 3D para poder obtener las nubes de puntos y poder posteriormente tener el modelo correspondiente del escenario de interés.

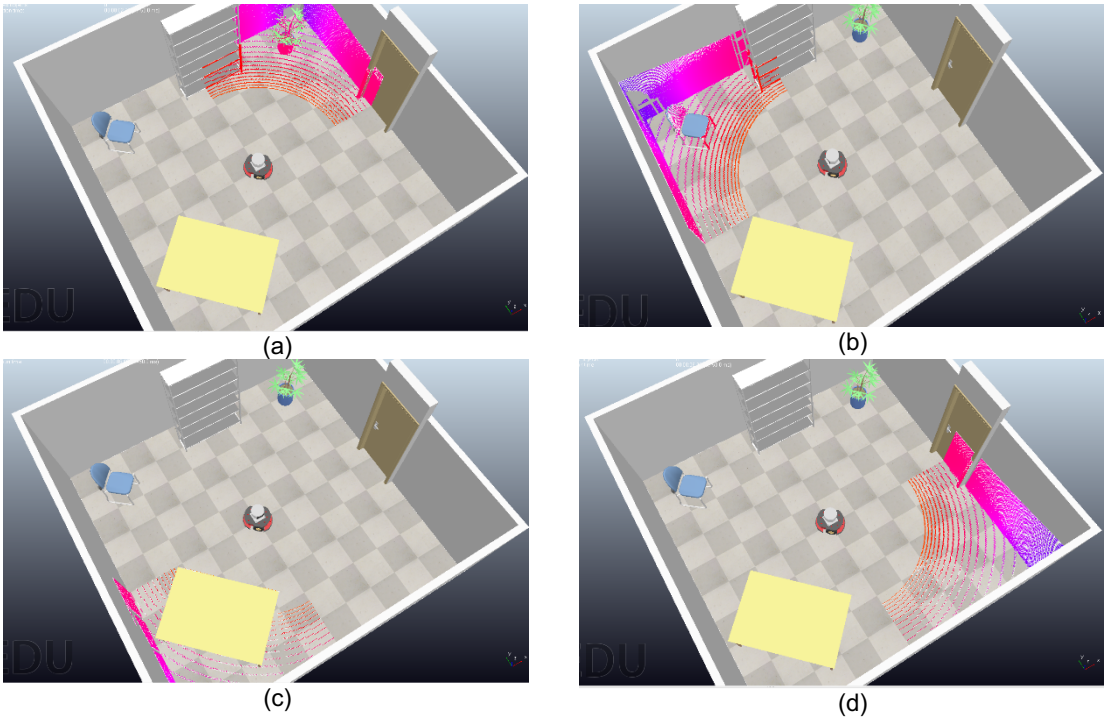


Figura 6. Entorno de Simulación CoppeliaSim con el robot PioneerP3DX.

Una vez iniciando la simulación el sensor empieza a escanear la habitación para recuperar la información tridimensional. En la figura 6, se muestran 4 diferentes vistas que se obtiene a partir del sensor Velodyne que se encuentra arriba del robot. Cabe mencionar que el robot está estático y lo que se mueve es el interior del sensor permitiendo proyectar una serie de puntos generados mediante láseres mediante los cuales se puede medir las distancias y por consecuencia obtener una representación mediante puntos de la escena u objeto que se encuentra en el lugar en donde se utilice el sensor.

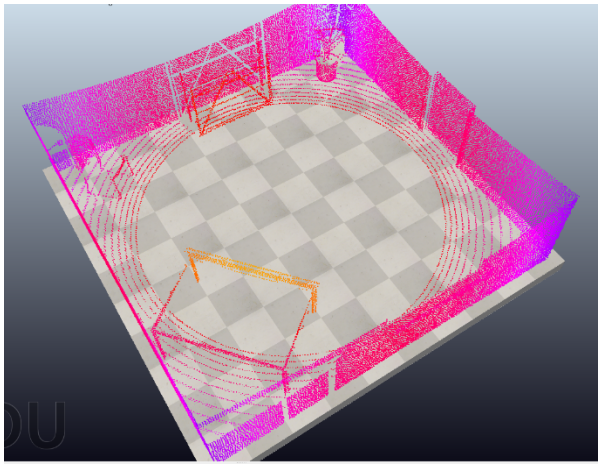


Figura 7. Entorno de Simulación CoppeliaSim con el robot PioneerP3DX.

A partir de la adquisición de datos mostrado en la figura 6 y utilizando el algoritmo de ICP se genera la nube de puntos correspondiente a la escena de la Figura 5. Esta nube de puntos es la que se triangulará mediante el algoritmo de BP para generar un modelo 3D correspondiente a la escena escaneada por el sensor de profundidad en este caso el Velodyne.

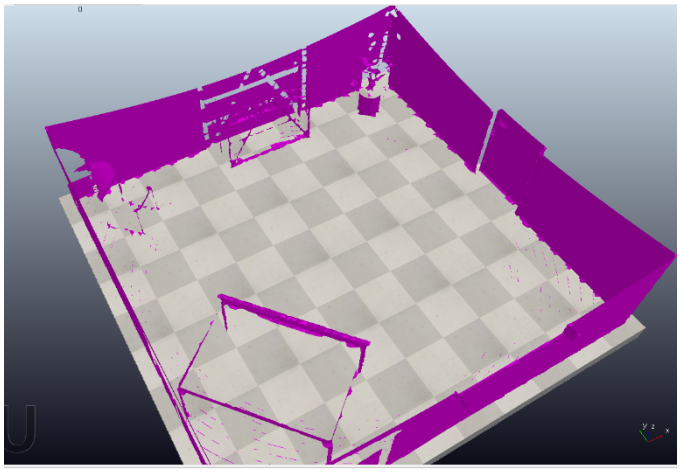


Figura 8. Entorno de Simulación CoppeliaSim con el robot PioneerP3DX.

Finalmente, y como lo comentamos en el párrafo anterior se utiliza el algoritmo de BP para generar el modelo 3D del escenario que se muestra en la Figura 5. El resultado de dicha triangulación se puede observar en la figura 8. Ambos algoritmos tanto el de ICP como el de BP se implementaron a partir de la librería de Open3D.

Resultados

Una vez mostrado que los algoritmos implementados funcionan procedemos a usarlo en datos reales. En este trabajo se usarán nubes de puntos tridimensionales recuperadas del web debido principalmente a que, no se cuenta con un sensor Velodyne. Sin embargo, los datos recuperados nos van a permitir probar el algoritmo en información en cierta manera como si nosotros la hubiéramos obtenido. Además, que se tiene la facilidad de contar con diferentes bases de datos para trabajos futuros.

Comencemos entonces por mostrar la escena de referencia que se utilizó para obtener los resultados que se presentan en esta sección. La figura 9, presenta una parte de la habitación que corresponde al edificio de Bimaristan Nur al-Din que se encuentra en Siria. Estos datos pertenecen al proyecto de openheritage3D (CyArk, 2019).

Figura 9. Escena utilizada para validar los algoritmos de ICP y BP en datos reales. (CyArk, 2019).

La información que nos proporciona la base de datos de openHeritage3D ya es una nube de puntos, sin embargo tenemos acceso a diferentes secciones en las cuales podemos validar el algoritmo de ICP para la fusión de nubes de puntos como se puede apreciar en la figura 10.

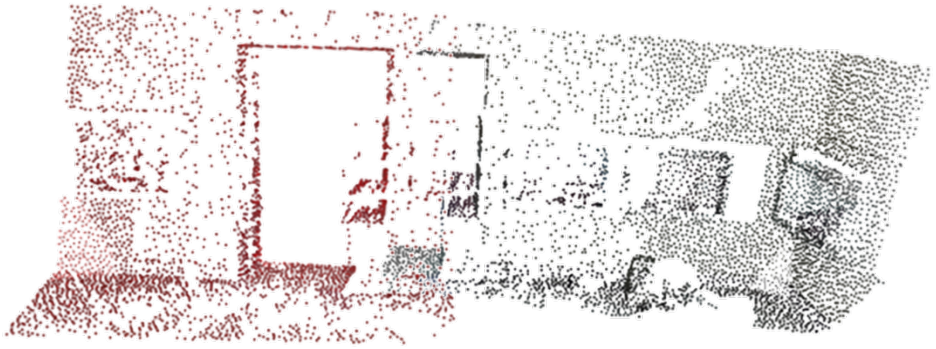


Figura 10. Dos nubes de puntos correspondientes a pate de la escena mostrada en la Figura 9.

En esta figura 10, se puede observar dos nubes diferenciadas por el color ya que mientras la nube de la izquierda es de color rojo la de la derecha los puntos son negros. También se puede apreciar que hay zonas en ambas nubes que son similares y son dichas similitudes que utiliza ICP para fusionar ambas nubes en una sola.

Así que después de aplicarlo en los datos anteriores se obtiene la nube de puntos resultantes que se muestra en la figura 11. Aquí, ya se han fusionado ambas nubes de puntos, se sigue conservando el color de los puntos de ca una de ellas para que se pueda apreciar la intersección entre ellas. Cabe mencionar que le algoritmo de fusión, en este caso ICP, elimina los puntos repetidos para evitar que existan redundancias que afecten a los procesamientos posteriores como lo es la triangulación.

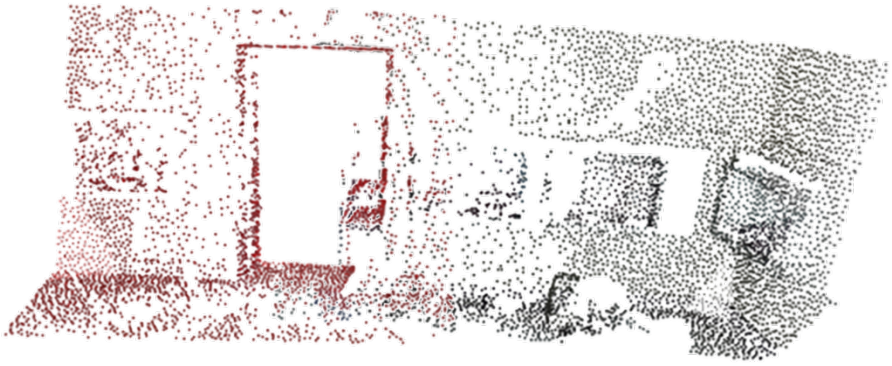


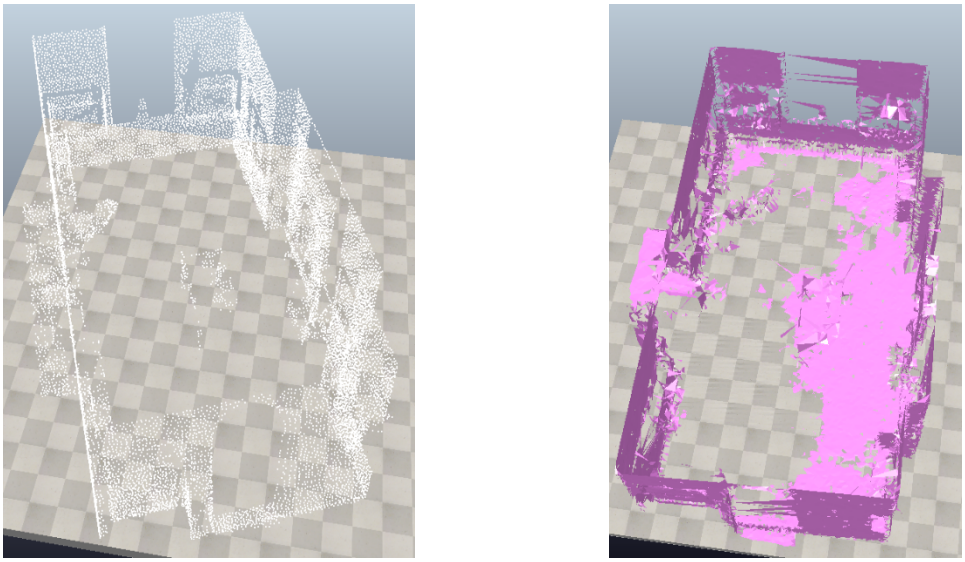
Figura 11. Fusión de las nubes de puntos que se presentan en la Figura 10.

Validado la fusión de nubes este proceso se debe de aplicar a todas y a cada una de las diferentes vistas que se tengan de la escena que nos interese reconstruir. Para el caso que estamos analizando se obtiene la nube de puntos 3D que se observa en la Figura 12(a).

Y de igual manera a como se hizo en los datos presentado en la sección de implementación se aplico el algoritmo de Ball Pivoting para obtener el modelo tridimensional o la malla correspondiente de la nube de puntos de la Figura 12(a). Esta malla resultante se presenta en la Figura 12(b).

VOLUMEN 16
XXVII Verano De la Ciencia
ISSN 2395-9797
www. jóvenesenlaciencia.ugto.mx

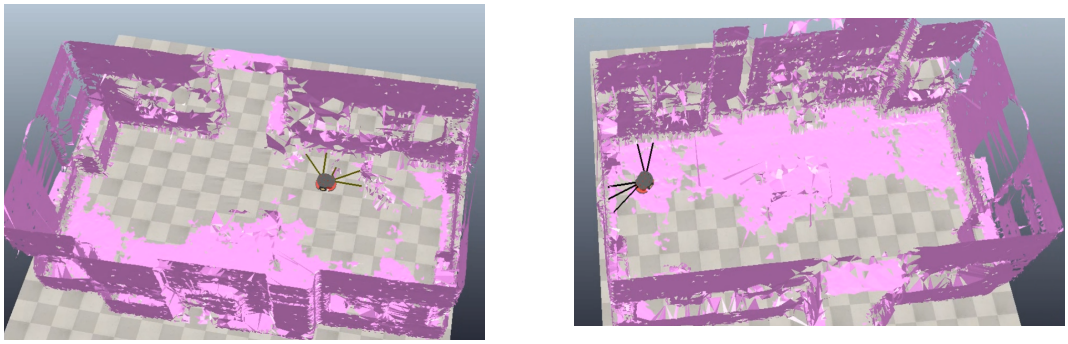
Tanto la nube de puntos como la malla resultante se muestran en el entorno de simulación. Lo anterior debido a que lo que se persigue es que un robot móvil sea capaz de navegar en el escenario creado a partir de la triangulación de una nube de puntos tridimensionales.



(a) (b)

Figura 12. (a) nube de puntos completa y (b) triangulación de (a) correspondiente a la escena de la Figura 9.

Finalmente, en la figura 13 se muestran dos imágenes en las que se puede observar que el robot virtual PioneerP3DX está navegando por el escenario creado. Se pueden apreciar que del robot salen una serie de líneas de color negro hacia diferentes zonas del modelo 3D. Estas líneas representan la interacción de los sensores de ultrasonido que posee el robot con el entorno. Dicha interacción nos indica, entre otras cosas, que los sensores del robot son capaces de medir la distancia entre el robot y los objetos presentes y por consecuencia que este pueda esquivarlos y continuar con la tarea asignada, la cual en este caso el la de navegar por el entorno.



(a) (b)

Figura 13. Dos diferentes capturas del robot navegando en el escenario creado.

VOLUMEN 16
XXVII Verano De la Ciencia
ISSN 2395-9797
www. jóvenesenlaciencia.ugto.mx

Conclusiones

Como conclusión se puede mencionar que los algoritmos implementados fueron eficientes para realizar la tarea tanto de fusión de vistas parciales en el caso de ICP y de la triangulación de puntos 3D para generar una modelo de la escena de interés utilizando el algoritmo de BP.

Si bien es cierto y como se puede apreciar en algunas figuras la triangulación no es muy precisa ya que quedan huecos en donde debiera de haber una sección completa. Sin embargo, para la tarea de navegación de un robot móvil esta falta de información en el modelo no es algo crítico, ya que en las pruebas realizadas el robot se desplaza sin problema alguno. Empero si sería mejor tener el modelo del entorno con un buen nivel de detalle óptimo para que el robot pueda realizar tareas donde requiera tener una mayor precisión en esta.

Así que en futuros trabajos se abordaran estos puntos para mejorar el desempeño de los algoritmos aquí presentados o en su caso evaluar otros que nos proporcionen una calidad óptima, principalmente en la creación de la malla 3D.

Referencias

Lorensen, W. E., & Cline, H. E. (1987). Marching cubes: A high resolution 3D surface construction algorithm. ACM SIGGRAPH Computer Graphics, 21(4), 163–169. https://doi.org/10.1145/37402.37422

Bernardini, F., Mittleman, J., Rushmeier, H., Silva, C., & Taubin, G. (1999). The ball-pivoting algorithm for surface reconstruction. IEEE Transactions on Visualization and Computer Graphics, 5(4), 349–359. https://doi.org/10.1109/2945.817351

Xu, C., & Prince, J. L. (1997, June 1). Gradient vector flow: a new external force for snakes. IEEE Xplore. https://doi.org/10.1109/CVPR.1997.609299

Chen, Y., & Medioni, G. (1992). Object modelling by registration of multiple range images. Image and Vision Computing, 10(3), 145–155. https://doi.org/10.1016/0262-8856(92)90066-c

Besl, P. J., & McKay, N. D. (1992). A method for registration of 3-D shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence, 14(2), 239–256. https://doi.org/10.1109/34.121791

Schroeder, W. J., Zarge, J. A., & Lorensen, W. E. (1992). Decimation of triangle meshes. ACM SIGGRAPH Computer Graphics, 26(2), 65–70. https://doi.org/10.1145/142920.134010

Rohmer, E., Singh, S. P. N., & Freese, M. (2013). V-REP: A versatile and scalable robot simulation framework. 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2013. https://doi.org/10.1109/iros.2013.6696520

Zhou, Q-Y., Park, J., & Koltun, K. (2018). Open3D: A Modern Library for 3D Data Processing, arXiv:1801.09847. doi:10.48550/arXiv.1801.09847

CyArk. (2019). Barristan Nur al-Din - LiDAR - Terrestrial, Photogrammetry - Terrestrial. Collected by Directorate General of Antiquities and Museums. Distributed by Open Heritage 3D. doi: 10.26301/nkv2-bn91.