XXXVIII Jornadas de Automática
https://doi.org/10.17979/spudc.9788497497749.0218



# EXPLORACIÓN Y RECONSTRUCCIÓN 3D DE FONDOS MARINOS MEDIANTE AUVs Y SENSORES ACÚSTICOS

Oscar L. Manrique, Mario Garzón y Antonio Barrientos
Centro de Automática y Robótica (UPM-CSIC), Universidad Politécnica de Madrid
c/ José Gutiérrez Abascal, nº2, 28006 Madrid
ol.manrique@alumnos.upm.es, ma.garzon@upm.es, antonio.barrientos@upm.es

## Resumen

*Este trabajo presenta un sistema para la reconstrucción 3D de fondos acuáticos poco profundos utilizando medidas de un array de sensores de distancia tipo sonar acoplado a un robot submarino tipo torpedo. Se utiliza el simulador UWSim, específicamente desarrollado para misiones subacuáticas así como la arquitectura de control y navegación COLA2 adaptada al modelo de robot utilizado. Se ha desarrollado también una aplicación que controla la trayectoria del robot, de modo que se garantice la cobertura completa del área teniendo en cuenta los solapamientos necesarios para crear el mapa. A medida que el robot se desplaza por el escenario, las medidas del multibeam se integran para crear nubes de puntos parciales, que luego se agrupan para obtener el mapa 3D del fondo completo. Esta tarea de agrupación requiere de un proceso de corrección errores de localización relativa, para lo que se utiliza el algoritmo Iterative Closest Point. Este algoritmo requiere un complejo ajuste de filtros y configuraciones que para afinar el resultado final y conseguir una reconstrucción de buena calidad. Finalmente, el sistema se probó sobre varios escenarios y trayectorias diferentes, diseñados con el fin de evaluar la robustez del sistema y la calidad del mapa generado.*

**Palabras clave:** Reconstrucción 3D, PointCloud, Sensor Multibeam, Reconstrucción de fondos marinos, UWSim, COLA2.

## 1. INTRODUCCIÓN

El reconocimiento del fondo marino siempre ha supuesto un reto para la navegación subacuática, ya que las condiciones en este entorno son muy variables y normalmente desconocidas, además no es posible el uso de tecnologías de localización o de comunicaciones, en las cuales se basan hoy en día la mayoría de los sistemas de navegación terrestre. Por tanto, el estudio de los fondos marinos y su conocimiento a la hora de llevar a cabo las misiones, supone una gran ventaja cuando hablamos de navegación, y más aún si se trata de sistemas autónomos.

Este trabajo tiene como objetivo principal el desarrollo de un sistema completo para la generación de mapas 3D utilizando robots submarinos. Se pretende por tanto generar una misión que incluye generación de trayectorias, navegación autónoma, recolección de datos y procesamiento on-line y offline de los datos para finalmente obtener una reconstrucción correcta del fondo. Las posibles aplicaciones de este trabajo son muy variadas, empezando por la navegación autónoma de AUVs, así como tareas de búsqueda y rescate además de otras aplicaciones de la robótica de alto nivel, pudiéndose integrar además en flotas heterogéneas de múltiples robots.

Para alcanzar el objetivo de este trabajo ha sido necesaria una combinación de desarrollos propios así como de integración y configuración de diferentes tecnologías o desarrollos previos, empezando por la simulación de entornos acuáticos, para la cual se utiliza en software *UWSim* [1]. Esta es una herramienta, de código abierto, específica para la simulación de misiones marinas. Es fácilmente configurable mediante un documento de *XML*, donde es posible incluir tantos objetos como se desee, así como parametrizar el entorno, añadiendo oleaje, presión, corrientes, claridad del agua entre otros, de manera que el escenario se asemeje a condiciones reales de operación. Este simulador ofrece también una gran variedad de sensores que se pueden adaptar fácilmente tanto al escenario como al vehículo que se introduzca en la escena.

El siguiente componente a integrar es el control y navegación del robot, para el que se utiliza la arquitectura *COLA2* [2], la cual se encuentra integrada en *UWSim* y ROS. Esta arquitectura, brinda las herramientas necesarias controlar los movimientos del robot y realizar una navegación hacia un punto de paso. Fue por tanto necesario desarrollar una aplicación, objetivo es ejercer como capitán del vehículo, es decir que se encargue de definir la trayectoria del vehículo y enviar a *COLA2* las sucesivas posiciones objetivo que se desea que el robot alcance para que recorra la trayectoria predefinida.

A medida que el robot se desplaza por el escenario, el sensor multibeam va publicando sus mediciones.

Es necesario por tanto, recoger estos datos y a partir de ellos crear nubes de puntos, que representan las medidas suministradas por el multibeam durante un periodo de tiempo. Una vez tenemos las nubes de puntos almacenadas, es posible generar un mapa 3D completo. Esta tarea podría realizarse de manera muy sencilla simplemente concatenando las diferentes nubes de puntos parciales. Pero debido a errores en las mediciones y en la localización del robot, cuyo error va incrementando continuamente, dos nubes de puntos pueden no coincidir perfectamente. Para solventar estos errores, se aplica el algoritmo *Iterative Closest Point (ICP)* [3], implementado en la librería *libpointmatcher* [4], este algoritmo se encarga de encontrar el movimiento relativo de una nube de puntos respecto otra, siempre que las dos compartan un área en común, es decir, teniendo dos nubes de puntos de áreas que se solapen entre sí, el algoritmo intentará rotar y/o trasladar una de las nubes de puntos de tal manera que ambas coincidan lo máximo posible. Esto solventa, en cierta medida, los errores de localización del robot que se han ido arrastrando desde el inicio de la misión.

Para finalizar, se han diseñado distintos escenarios, con objetos de diferentes tipos (conos, esferas, prismas). Así mismo, se han generado trayectorias diferentes para la cobertura del área, que requerirán tiempos de ejecución distintos y producirán nubes de puntos que se solapen más o menos. Estas trayectorias y escenarios se combinan y evalúan con el objetivo de realizar una valoración global del sistema desarrollado.

## 2. METODOLOGÍA Y DESARROLLO

Para la desarrollo del trabajo, ha sido necesario pasar por varias fases, cumpliendo objetivos parciales en cada una de ellas. Los más relevantes se detallan a continuación.

### 2.1. Integración del sensor con UWSim y ROS

*UWSim* está pensado de tal forma que su comunicación con el Sistema Operativo para Robots (ROS) sea especialmente sencilla. Incluso en su configuración por defecto se encuentran varios servicios y tópicos listos para ser utilizados, adicionalmente, en el archivo de configuración del escenario a simular, se incluye un bloque para definir la interfaz de comunicación con ROS, en la que se ofrecen 22 interfaces distintas. Así, por ejemplo, en este trabajo se utilizará un sensor *multibeam*, por lo que basta con añadir la interfaz *multibeamSensorToLaserScan* para disponer de un canal donde se publiquen las medidas realizadas por el sensor correspondiente.

El sensor *multibeam* simula un conjunto de sensores de distancia cuya orientación se configura previamente, es decir se trata de una colección de sensores de alcance colocados en un mismo punto apuntando a distintos ángulos y contenidos en un mismo plano. Tanto el la resolución, o incremento angular, como los ángulos máximo y mínimo de apertura se pueden configurar fácilmente. Este sensor se ubicó en la parte ventral del robot, de manera que realice un barrido transversal a la dirección de navegación del robot. Esta localización se debe reflejar también en ROS mediante la publicación de una transformación entre la base del robot y la posición del sensor, en la Figura 1 se muestra una imagen de los datos generados por este sensor.

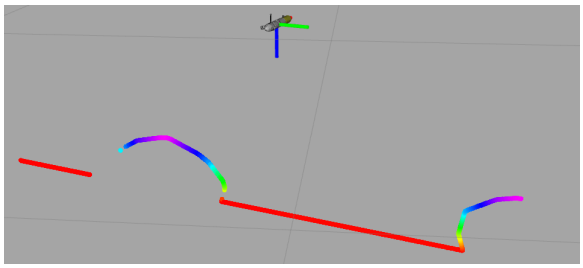


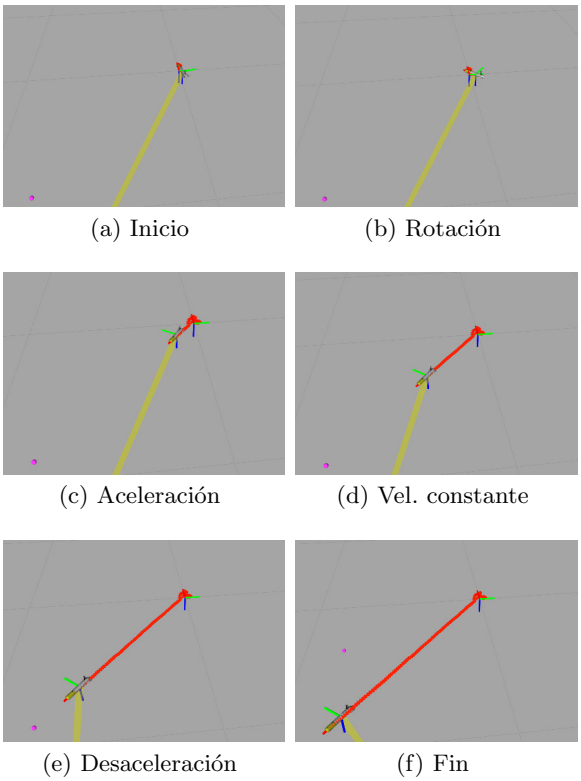
Figura 1: **Sensor multibeam**. Visualización de las lecturas recogidas por el sensor multibeam.

### 2.2. Guiado, Navegación y Control del AUV

La siguiente tarea a desarrollar es la navegación autónoma del robot, tarea necesaria para conseguir que el robot recorra un camino predefinido. Para esto se ha utilizado la librería *Component Oriented Layer-based Architecture for Autonomy (COLA2)*[2]. Esta arquitectura implementa un control basado en tres capas, reactiva, ejecución y misión y permite realizar misiones de manera autónoma, Además incluye los modelos cinemáticos y dinámicos de 2 robots extensamente utilizados en la investigación de robótica submarina (Sparus_II y Girona_500). Se ha utilizado una configuración propia de *COLA2* para que se adecue a los escenarios y a la misión.

Para el generación de la trayectoria de cobertura de área, se ha desarrollado un programa que envía al robot diferentes posiciones que debe alcanzar, de manera que al final el robot haya recorrido la trayectoria deseada y por tanto realizado un barrido completo de la superficie de interés, el algoritmo no enviará una nueva posición objetivo mientras que el vehículo no haya alcanzado la meta parcial. Para enviar la posición deseada del

robot, se utiliza un servicio que ofrece *COLA2*, con nombre **/enable_goto**. Este servicio define de manera automática los canales y señales de control que se deben utilizar, y requiere una serie de parámetros, en los que se define, entre otras cosas, la posición y velocidad deseada, el ángulo de rotación y la profundidad del punto final. La configuración que le pasemos afectará a la calidad de los resultados. Por poner un ejemplo, cuanto más rápido se ejecute el barrido, menos lecturas del fondo se realizará por metro recorrido.



(a) Inicio (b) Rotación

(c) Aceleración (d) Vel. constante

(e) Desaceleración (f) Fin

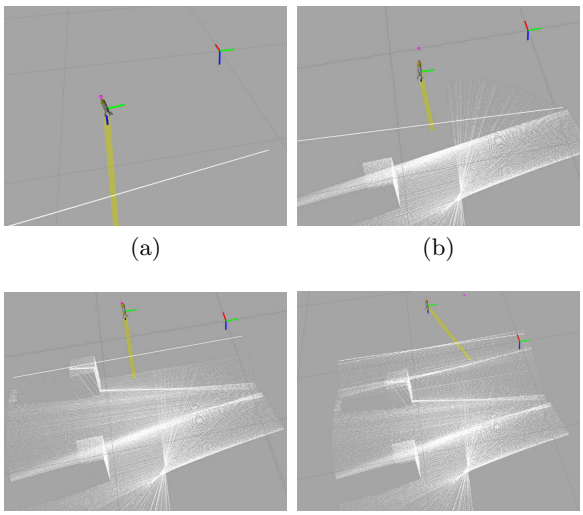
Figura 2: Secuencia de movimiento hacia una posición objetivo.

Como se aprecia en la Figura 2, el algoritmo que utiliza *COLA2* para alcanzar la posición objetivo inicia con una rotación pura hasta orientarse correctamente con la posición objetivo para después avanzar hacia ella. La secuencia avance inicia con una aceleración, continua con un movimiento a velocidad constante y termina con una desaceleración hasta alcanzar la posición objetivo. En la figura 2f el vehículo ya ha alcanzado la posición objetivo, es ese momento se le envía la siguiente posición objetivo, hasta completar la trayectoria deseada.

### 2.3. Creación de nubes de puntos parciales

Los datos generados por el sensor multibeam se publican en forma de mensajes de tipo *LaserScan*, que es utilizado para representar los datos de un escáner láser 2D. Gracias a ello, podemos trabajar de manera cómoda con los datos obtenidos. Para convertir estos datos en una nube de puntos y visualizarlas en 3D, es necesario agrupar una serie de estos mensajes y transformarlos a un mensaje tipo nube de puntos (*Pointcloud2*). Esta conversión se realiza utilizando el software *laser_assembler*[1] que recoge todas las medidas publicadas por el multibeam y las transforma en una nube de puntos que contiene todos los datos.

El proceso de creación de una nube de puntos se puede observar en la Figura 3, en la cual se inicia con una sola medida del multibeam y se van agrupando progresivamente hasta completar una imagen más significativa.



(a) (b)

(c) (d)

Figura 3: Secuencia temporal de lectura de medidas del multibeam.

El software de ensamblado de datos permite definir el intervalo de recolección de datos, definiendo dos instantes temporales correspondientes al inicio y final de la recolección. Para controlar este proceso se ha creado una aplicación que se encarga de enviar las solicitudes marcando los correspondientes en intervalos de tiempo. Esta aplicación debe crear las nubes de puntos garantizando que se solapen ciertas medidas entre una nube de puntos y la siguiente, ya que de no ser así no podría realizarse una correcta alineación en el siguiente paso del proceso, al intentar generar un mapa con nubes de puntos que no se solapan, el algoritmo

[1] http://wiki.ros.org/laser_assembler

las agrupará de forma que varias secciones del mapa se superpongan en una sola, esto se visualiza claramente en la Figura 4.

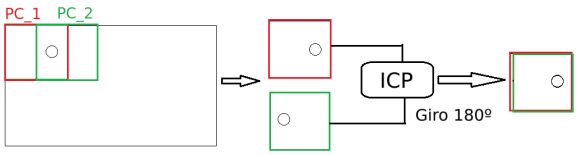



Figura 4: **Solapado erróneo.** El algoritmo ICP superpone las dos nubes de puntos para reducir al mínimo la distancia entre ambas.

## 3. Reconstrucción 3D del mapa

El siguiente paso del proceso, es la reconstrucción del mapa 3D juntando las diferentes nubes de puntos en una sola, de forma que se represente el escenario completo. Como se ha mencionado anteriormente, esta unión se realiza utilizando el programa *libpointmatcher* [4] y el algoritmo *ICP*. De modo que se alineen dos o más nubes de puntos que se superponen, de modo que se puedan corregir posibles errores y generar un mapa que se ajuste mejor a la realidad. Para poder hacer uso del libpointmatcher, es necesario convertir los Pointclouds a un formato valido para el programa, por se ha desarrollado una aplicación que transforma las nubes de puntos de mensajes *Pointcloud2* de ros a ficheros en formato *Visualization ToolKit (.vtk)*.

El algoritmo *ICP* realiza una medición inicial de las distancias entre cada punto de una primera imagen o nube de puntos, con el punto más cercano en la segunda imagen. Una vez realizadas las medidas, se estima una matriz de transformación que aporte un movimiento de traslación y otro de rotación, aplica esta matriz a todos los puntos y vuelve a medir distancias con sus nuevos vecinos más cercanos. Este proceso se repite iterativamente hasta que se ha alcanzado una distancia mínima aceptable entre ambas imágenes.

*Libpointmatcher* aporta además otras funcionalidades, como la capacidad de aplicar varios filtros a las imágenes, antes y después del procesamiento. Esto permite eliminar posibles ruidos, o reducir la densidad de puntos cuando es excesivamente alta.

Otra funcionalidad que ofrece libpointmatcher y, como veremos más adelante, ha sido clave para el sistema, es la verificación de transformaciones (*TransformationChecker*). Cuando ICP estima una transformación, este componente se encarga de comprobar si los resultados del algoritmo cumple límites preestablecidos. Es posible utilizar tres tipos de configuraciones, que conjuntamente afectan tanto el tiempo de ejecución como la calidad del mapa obtenido, estas configuraciones se listan a continuación.

- Número de iteraciones: Acota superiormente el numero de iteraciones que el algoritmo puede realizar.
- Límite de error: Establece un valor mínimo de diferencia entre las transformaciones estimadas por debajo del cual consideramos que el resultado del algoritmo es aceptable.
- Acotación superior de rotación y traslación: Limita los movimientos estimados de traslación y de rotación. Si el algoritmo estima unas transformaciones superiores, entonces el proceso finaliza.

Es posible encontrarse en una situación en que las nubes de puntos generadas no tengan un solapamiento suficiente, o cuando la superficie es muy homogénea. Para explicar esta situación, suponemos un entorno en el que solo existen unos pocos objetos y ejecutamos la misión para obtener los Pointclouds.

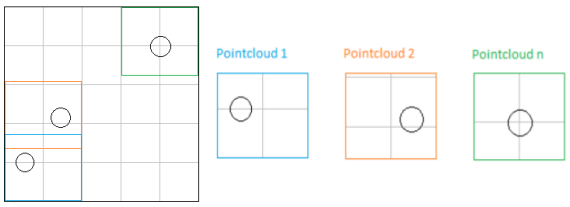



Figura 5: **Ejemplo de escenario y nubes de puntos.** Se presenta un esquema de un escenario sencillo con 3 objetos y se señalizan tres nubes de puntos tomadas durante la ejecución de la misión.

Como se puede ver en la Figura 5, el resultado serían tres nubes de puntos distintas a las que aplicaremos el *ICP*. Este algoritmo busca la transformación que consiga un menor error entre los puntos. En este caso, el resultado óptimo es juntar las nubes de puntos, dando lugar a una representación de la superficie irreal, como se muestra en la Figura 6.

Para solucionar esta situación ha sido necesario realizar dos adaptaciones al sistema completo, la primera de ellas consiste en buscar una mayor superposición de las nubes de puntos, para lo cual se modificó el método de obtención de las nubes de puntos, aislándolo completamente de la misión y generando una aplicación externa que genere las nubes de puntos con referencias de tiempo superpuestas, es decir, cada nueva captura inicia antes de terminar la anterior.

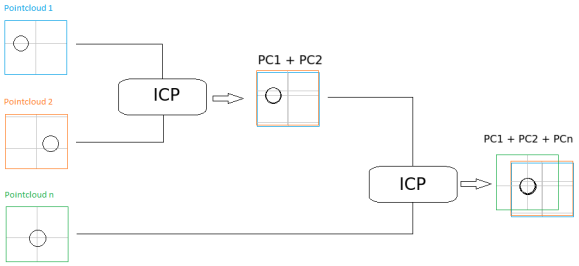



Figura 6: **ICP sin límites establecidos.** Se puede observar que el algoritmo ICP agrupa las nubes de puntos de manera erronea

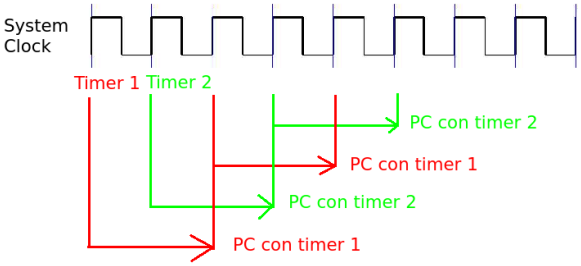



Figura 7: **Esquema de la toma de referencias temporales.**

Como se aprecia en la Figura 7, se utiliza un temporizador que genera las referencias de tiempo que se alternan entre sí, por tanto resulta en nubes de puntos que comparten aproximadamente el 50 % de su área total.

La segunda adaptación para solventar el problema del solapamiento fue el empleo de límites para el algoritmo *ICP*, como se ha explicado anteriormente, *libpointmatcher* permite establecer ciertos límites en el proceso de unión de imágenes. Se decide por ello imponer límites superiores en la matriz de transformación estimada por el *ICP*, tanto para la rotación como para la traslación. En los casos en que estos límites sean superados, se concatenarán las dos nubes de puntos sin corrección, es decir tomando solamente la posición del robot como referencia.

De esta manera, retomando el ejemplo presentado en la Figura 5, es posible conseguir que el *ICP* no ejecute la corrección de las nubes de puntos y que simplemente las concatene, una generando una única imagen que represente el escenario de manera más fehaciente, tal como se observa en la Figura 8.

## 4. Experimentos y resultados

Esta sección presenta una serie de experimentos que buscan realizar una valoración del sistema al completo y de los aspectos que pueden afectar a la calidad de los mapas obtenidos. Se evaluará eñ efecto del tipo de trayectoria implementado así como de los diferentes objetos que se puedan encontrar en el fondo.

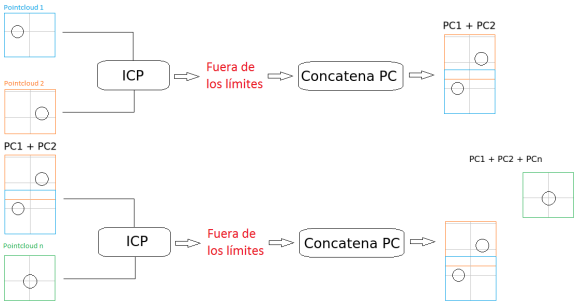



Figura 8: **ICP con límites de rotación y traslación.**

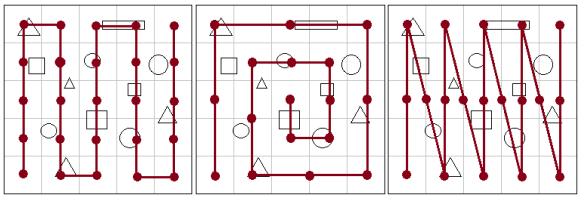
Es importante destacar la relevancia de la trayectoria elegida, pues como se ha explicado, la superposición de las nubes de puntos es un factor muy importante a la hora de crear el mapa global. Al generar las nubes de puntos a una frecuencia fija, el número de las mismas, el nivel de solapamiento y las áreas visitadas en repetidas ocasiones dependen de la trayectoria. Además, como es evidente, el tiempo que tardará la misión en realizarse depende directamente de la trayectoria realizada.

### 4.1. Tipos de trayectorias y escenarios

Para los experimentos se han generado tres tipos de trayectorias diferentes, descritas a continuación.

- **Rectangular** (Figura 9a): Trayectoria de 25 waypoints y 8 puntos de giro de 90 grados. El área de interés es recorrido de forma homogénea por lo que obtendremos un mapa con una densidad de puntos bastante similar.
- **Espiral** (Figura 9b): Trayectoria de 17 waypoints con 8 puntos de giro de 90 grados. Esta trayectoria inicia con un bajo grado de superposición que va aumentando a medida que se realiza el recorrido.
- **Triangular** (Figura 9c): Trayectoria de 19 waypoints con 8 puntos de giro de más de 90 grados. El objetivo de realizar esta trayectoria es la de estudiar casos en los que el robot tenga que realizar rotaciones más largas, de casi 180 grados. Se consigue además un alto grado de superposición entre todas las nubes de puntos.

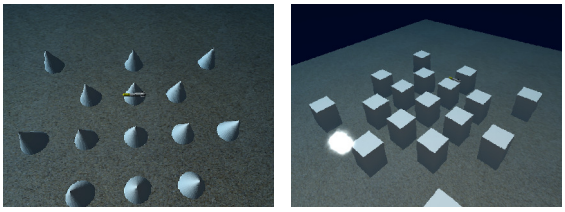
De igual manera, se han creado escenarios con distintos tipos de superficies: conos, cilindros, prismas, esferas y uno con varios elementos diferentes.



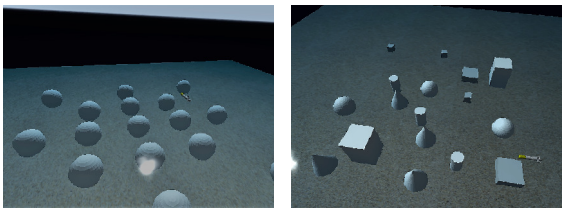
(a) Rectangular (b) Espiral (c) Triangular

Figura 9: **Tipos de trayectoria utilizados**

El objetivo es evaluar si el tipo de objectos que se encuentran en el fondo influyen en la calidad del mapa generado, estos escenarios se muestran en la Figura 10.



(a) Conos (b) Prismas



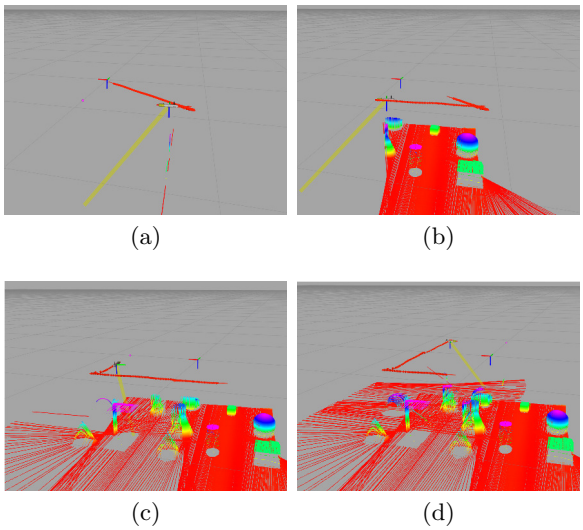
(c) Esferas (d) Variado

Figura 10: **Escenarios utilizados**

Una vez definido el escenario y el tipo de trayectoria se realiza el proceso de captura de datos. Este proceso inicia con el robot en la posición inicial de la ruta, a continuación el robot se mueve, siguiendo la trayectoria que se le ha definido, y a medida que avanza va capturando datos con el sensor multibeam. Estos datos se van agrupando en nubes de puntos parciales, que pueden visualizarse inicialmente una tras otra, es decir sin el proceso de corrección basado en *ICP*, los datos se almacenan en disco durante el experimento, y posteriormente se realiza la alineación utilizando *ICP*. La Figura 11 muestra una secuencia de captura de datos, así como las nubes de puntos parciales que se crean en línea.

### 4.2. Resultados

En esta sección presentamos algunos de los resultados obtenidos. Se debe tener en cuenta que para que esta valoración sea consistente, todas las misiones se han ejecutado con la misma configuración, tanto del entorno acuático como de los parámetros del vehículo. La Figura 12 muestra el resultado del proceso en un escenario con elementos variados, en los cuales se puede apreciar la mejora en el detalle de los mismos, por ejemplo en los vértices de los conos o en la forma de los prismas.



(a) (b)

(c) (d)

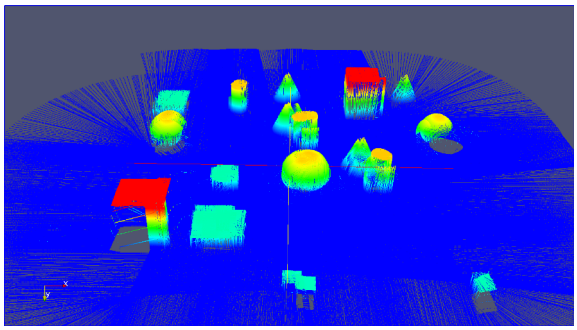
Figura 11: **Proceso de captura de datos** La línea roja en la parte superior muestra la ruta del robot. Los datos recogidos por el sensor multibeam se muestran en colores que varían según la altura relativa al fondo

La Figura 13 muestra el resultado antes y después de la alineación de las nubes de puntos para el escenario de esferas y la ruta rectangular, se puede observar que en este caso la mejore no es tan notoria, posiblemente debido tanto a la ruta del robot como a la configuración del algoritmo *ICP*.

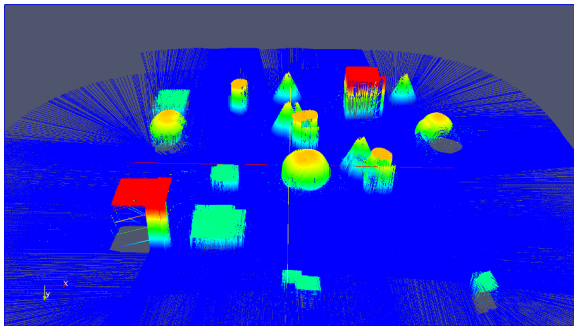
### 4.3. Análisis de resultados

Una vez obtenidos los resultados, es posible realizar algunas observaciones del comportamiento del sistema desarrollado:

- **Giros durante la misión**: Las nubes de puntos adquiridas mientras el robot realiza una rotación contienen bastantes errores, por tanto, para obtener un mapa de mayor calidad se podría realizar una selección previa con el fin de eliminarlas.
- **Filtrado de datos**: *libpointmatcher* ofrece una gran variedad filtros para las nubes de puntos, que permiten obtener procesos de ejecución más o menos rápidos y unos resultados de mayor o menor calidad. Estos filtros deben ser calibrados detalladamente para conseguir mapas de la mejor calidad posible.

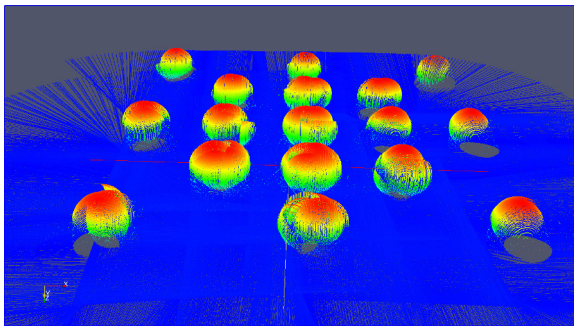


(a) Antes

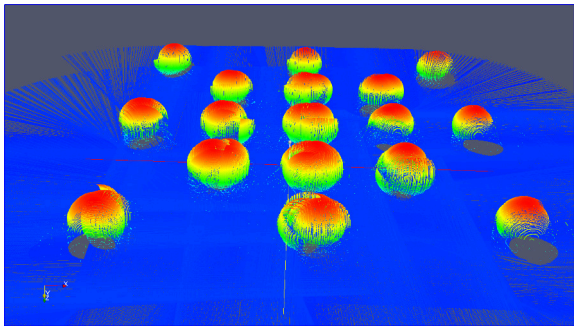


(b) Después

Figura 12: **Escenario variado** Se observan las nubes de puntos del escenario variado, antes y después del proceso de ajuste del mapa 3D utilizando ICP.



(a) Antes



(b) Después

Figura 13: **Escenario esferas** Se observan las nubes de puntos del escenario con esferas, antes y después del proceso de ajuste del mapa 3D utilizando ICP.

- **Influencia de los objetos**: No se ha observado que el tipo de superficie de los objetos influya directamente en la calidad de los mapas obtenidos. En todos los tipos de figuras se observan errores aparentemente similares. Sin embargo, si se ha detectado que en los objetos de menor tamaño se pueden apreciar más distorsiones o fallos en la reconstrucción. Otro aspecto a destacar, es que los errores destacan de manera más notoria en elementos que contengan vértices más cerrados, por ejemplo, en los conos.
- **Influencia de las trayectorias**: En este caso sí se aprecian claras diferencias entre unas trayectorias y otras, la mejor trayectoria para generar los mapas es la circular ya que se aprecia una mejor definición de los objetos así como una distribución de errores no homogéneos. Esto se debe a que se realizan más barridos por la zona central que por el exterior. En cuanto a los otros tipos de trayectorias, la rectangular y la triangular, no se han observado resultados concluyentes, sin embargo, los giros más amplios en la trayectoria triangular parecen afectar un poco más el resultado. El tipo de trayectoria realizada es el factor que más afecta la duración de la misión, ya que varía considerablemente la longitud del recorrido. También tiene una influencia directa sobre el numero de nubes de puntos obtenidas, pues la generación de estos se realiza a una frecuencia fija, por lo que en el caso de las trayectorias triangulares se obtienen muchas más nubes de puntos y tarda también un mayor tiempo en completar la misión.
- **Velocidad del movimiento**: Las misiones deben plantearse de tal manera que la velocidad sea la adecuada con respecto a la frecuencia del multibeam. De esta manera podremos conseguir una mejor definición del fondo con un solo barrido del vehículo.

## 5. Conclusiones

El trabajo presentado brinda una solución para la generación de mapas de fondos acuáticos, se han desarrollado e integrado diferentes herramientas que permiten generar los mapas abarcando el proceso completo, es decir, desde la planificación de una misión y los componentes necesarios para obtener los datos, hasta un tratamiento de los mimos y la generación del mapa final.

La integración de distintas herramientas para conseguir crear un mapa 3D paso a paso se ha basado

en el uso del framework ROS como punto de conexión entre los diferentes algoritmos y herramientas de software o librerías utilizadas (*UWSim*, *COLA2*, *libpointmatcher*).

La integración del algoritmo *ICP* como etapa final del proceso, ha permitido afinar el resultado, sin embargo, se ha comprobado que su uso correcto requiere de una ajuste muy fino de los parámetros y del pre-procesado de los datos.

El sistema desarrollado se ha probado en diferentes escenarios y utilizando rutas diversas. Se ha observado que la calidad del mapa resultante no varía considerablemente según los objetos presentes en el escenario. Sin embargo, se ha verificado que la trayectoria realizada si afecta el resultado. Las pruebas han mostrado que una trayectoria circular permite una mejor reconstrucción, debido a que esta trayectoria realiza más barridos de la zona central y además permite visitar en varias ocasiones zonas cercanas, de modo que se facilita el trabajo del algoritmo *ICP*.

**Agradecimientos**

Este trabajo ha sido parcialmente financiado por los proyectos RoboCity2030-III-CM (Robótica aplicada a la mejora de la calidad de vida de los ciudadanos. fase III; S2013/MIT-2748), financiado por los Programas de Actividades I+D en la Comunidad de Madrid y cofinanciado por los Fondos Estructurales de la Unión Europea y PRIC (Protección robotizada de infraestructuras críticas, DPI2014-56985-R), financiado por el Ministerio de Economía y Competitividad del Gobierno de España. Los autores quisieran además agradecer al grupo IRSLab en la Universidad Jaume-I de Castellón por su desarrollo del simulador UWSim así como al grupo ViCOROB de la Universidad de Girona por su desarrollo y colaboración con la arquitectura COLA2.

## Referencias

[1] M. Prats, J. Pérez, J. J. Fernández, and P. J. Sanz. An open source tool for simulation and supervision of underwater intervention missions. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2577–2582, Oct 2012.

[2] Andres El-Fakdi y Marc Carreras Narcis Palomeras. Cola2: A control architecture for auvs. pages 695 – 716, Aug 2012.

[3] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, Feb 1992.

[4] François Pomerleau, Francis Colas, Roland Siegwart, and Stéphane Magnenat. Comparing ICP Variants on Real-World Data Sets. *Autonomous Robots*, 34(3):133–148, February 2013.