Proyecto de Investigación de Excelencia de la Junta de Andalucía P10-TEP-6101-R

"Navegación autónoma de un robot móvil 4x4 en entornos naturales mediante GPS diferencial y telémetro láser tridimensional"

Obtención de barridos 3D nivelados mientras Andábata se desplaza

Manuel Zafra Granados
Jorge L. Martínez Rodríguez
Jesús Morales Rodríguez

Departamento de Ingeniería de Sistemas y Automática
Universidad de Málaga

Septiembre de 2015

Índice

1. Objetivo

Andábata es un pequeño robot móvil terrestre diseñado para desplazarse por entornos naturales. Para poder navegar se hace imprescindible conocer con exactitud el entorno que rodea al robot. Para ello, el robot tiene incorporado un telémetro láser 3D con el que capturar una nube de puntos para procesarla.

El objetivo principal de este informe es describir la adquisición de barridos 3D nivelados mientras el robot se mueve por terreno irregular. Las coordenadas Cartesianas a las que están referenciados los puntos de la nube 3D tiene el origen del sistema de referencias en el centro geométrico del rectángulo definido por los puntos de contacto de las ruedas con el suelo, el eje Z apuntando en sentido opuesto al de la gravedad y el eje X en la proyección del eje X del vehículo sobre el plano perpendicular al eje Z de los puntos, según se miestra en la figura 1.

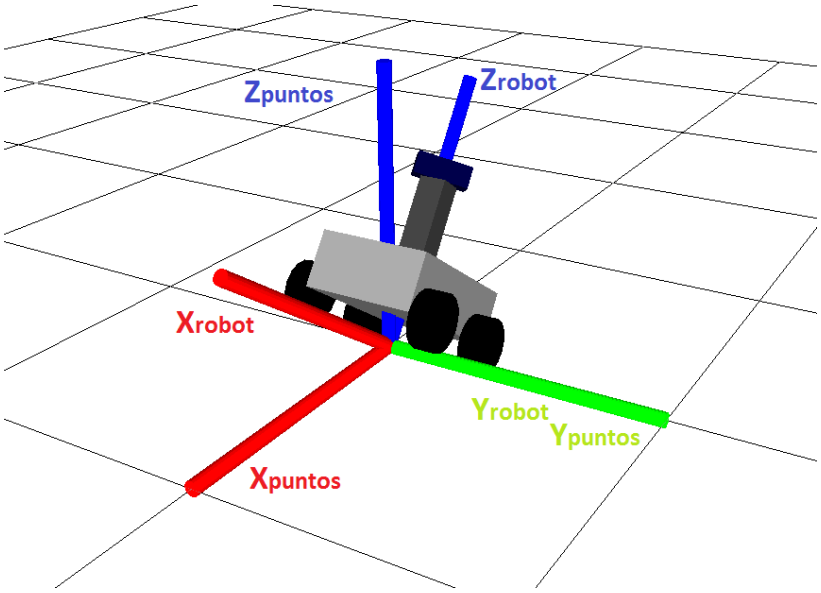

Figura 1: Ejes de coordenadas a los que están referidos los puntos de la nube 3D.

2. Punto de partida

Andábata es capaz de adquirir barridos láser 2D e ir añadiéndolos uno a uno a un mapa 3D mediante *octrees*, de forma que dependiendo del ángulo del sensor 2D se actualiza una parte u otra del mapa. Todo ello se describe en el proyecto final de carrera [1].

La comunicación con el sensor láser se realiza mediante dos nodos de ROS, uno que envía las órdenes de velocidad de giro a la base y otro que lee el ángulo en el que se toma cada barrido 2D.

El robot también tiene un teléfono móvil, del que se podrían utilizar sus sensores (IMU, magnetómetro, barómetro, GPS...), pero que todavía no se encuentran disponibles para la computadora del robot.

También tiene una tableta que se utiliza para la teleoperación, y que, entre otras, usa una aplicación para android llamada *android_sensors_driver* [2] que envía las medidas de los sensores de la tableta directamente al entorno de ROS en forma de publicaciones en *topic*s. Esta aplicación publica la información de cada sensor en *topics* independientes.

El robot es capaz de estimar su posición y orientación en un entorno interior y en dos dimensiones [1]. El cómputo de esta odometría se realiza gracias a un modelo cinemático aproximado 2D del robot que utiliza las medidas de los sensores Hall que tiene incorporados cada uno de los motores de las ruedas.

3. Modelo cinemático aproximado 2D

Para integrar las medidas de la odometría del robot en el posicionamiento 3D se hace necesario adaptar el modelo cinemático al exterior, ya que, los parámetros dependen del tipo de suelo por el que se desplace. El modelo utilizado y el método de obtención de los parámetros son los presentados en [3]. En concreto se ha utilizado el modelo simétrico derecha/izquierda y con $y_{CIR} = 0$, realizandose experimentos en linea recta para obtener μ y girando sobre si mismo para obtener x_{CIR}.

Los datos experimentales de los que se disponía en interior, obtenidos sobre una superficie nivelada plana [1], se presentan en las Tablas 1 y 2 para el parámetro angular y lineal respectivamente.

Velocidad (PWM)		**Pulsos**		**Distancia lineal (mm)**		$\boldsymbol{\phi}$		$\boldsymbol{x_{CIR}}$
Der	**Izq**	**Der**	**Izq**	$\boldsymbol{\int V_d dt}$	$\boldsymbol{\int V_i dt}$	**º**	**Rad**	**(mm)**
250	-250	2002	-2054	2495,83	-2560,65	356	6,2134	406,9
-250	250	-2066	2118	-2575,61	2640,44	-365	-6,3705	409,4
125	-125	1982	-2028	2470,90	-2528,24	359	6,2657	398,9
-125	125	-2014	2039	-2510,79	2541,95	-360	-6,2832	402,1
								404,3

Cuadro 1: Resultados del experimento realizado para determinar x_{CIR} para interiores.

Velocidad (PWM)		**Pulsos**		**Distancia lineal (mm)**		**d**	$\boldsymbol{\mu}$
Der	**Izq**	**Der**	**Izq**	$\boldsymbol{\int V_d dt}$	$\boldsymbol{\int V_i dt}$	**(mm)**	
250	250	3170	3243	3951,93	4042,94	3900	0,9756
-250	-250	-3124	-3193	-3894,59	-3980,61	-3770	0,9574
375	375	3572	3642	4453,09	4540,36	4310	0,9585
-375	-375	-3592	-3681	-4478,03	-4588,98	-4330	0,9551
							0,9617

Cuadro 2: Resultados del experimento realizado para determinar μ para interiores.

Los datos obtenidos tras repetir los experimentos sobre tierra se pueden ver en las Tablas 3 y 4 donde, comparándolas con las anteriores se puede observar que en exteriores al robot le cuesta más desplazarse (μ menor) y girar (x_{CIR} mayor).

Velocidad (PWM)		**Pulsos**		**Distancia lineal (mm)**		$\boldsymbol{\phi}$		$\boldsymbol{x_{CIR}}$
Der	**Izq**	**Der**	**Izq**	$\int \boldsymbol{V_d dt}$	$\int \boldsymbol{V_i dt}$	º	**Rad**	**(mm)**
250	-250	2061	-2089	2569,38	-2604,29	345	6,0237	429,4
-250	250	-2150	2164	-2680,33	2697,786667	-362	-6,3132	425,9
125	-125	2108	-2007	2627,97	-2502,06	346	6,0332	425,1
-125	125	-2156	2089	-2687,81	2604,29	-367	-6,4054	413,1
								423,4

Cuadro 3: Resultados del experimento realizado para determinar x_{CIR} para exteriores.

Velocidad (PWM)		**Pulsos**		**Distancia lineal (mm)**		**d**	$\boldsymbol{\mu}$
Der	**Izq**	**Der**	**Izq**	$\int \boldsymbol{V_d dt}$	$\int \boldsymbol{V_i dt}$	**(mm)**	
250	250	3080	3160	3839,73	3939,47	3630	0,9332
-250	-250	-3081	-3158	-3840,98	-3936,97	-3630	0,9334
375	375	3614	3646	4505,45	4545,35	4200	0,9281
-375	-375	-3697	-3730	-4608,93	-4650,07	-4310	0,9310
							0,9314

Cuadro 4: Resultados del experimento realizado para determinar μ para exteriores.

De esta forma, la velocidad lineal y angular del robot se calculan como:

$$V = \frac{\mu\,(v_d + v_i)}{2} \qquad \omega = \frac{\mu\,(v_d - v_i)}{2\,x_{CIR}} \tag{1}$$

siendo v_d y v_i las velocidades del lado derecho e izquierdo de Andábata, respectivamente.

4. Implementación en ROS

Para alcanzar el objetivo planteado hay que calcular la estimación del movimiento del robot en 3D y agrupar barridos 2D de forma coherente con esta información.

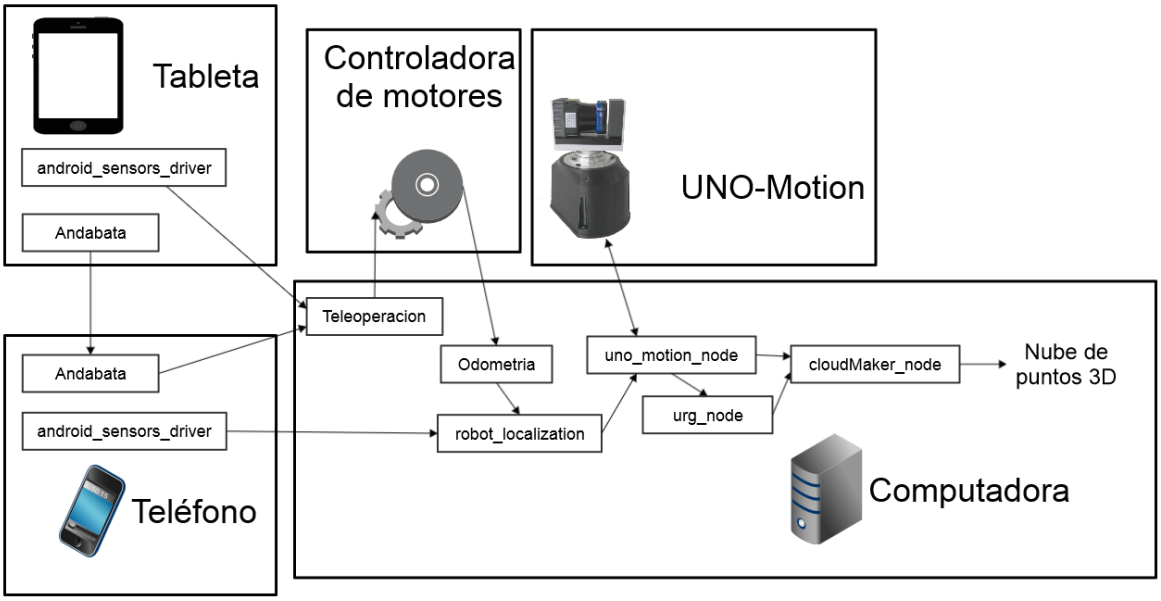

Figura 2: Esquema general de bloques.

El esquema de bloques implementado se muestra en la Figura 2, donde se puede ver que la estimación de la posición del robot proporcionada por la odometría se mejora al fusionar las medidas de los sensores del teléfono publicadas en ROS por una versión modificada de *android_sensors_driver*. También se muestra la actualización de los *driver* del telémetro láser 3D que pasan de ser dos nodos a ser solo uno llamado *uno_motion_node*.

4.1. Posicionamiento 3D

Para realizar la estimación del movimiento del robot y conseguir que sea tridimensional, el primer paso a realizar era adaptar el modelo cinemático sobre tierra, tal y como se ha descrito en la sección 3. El siguiente paso es hacer accesibles para la computadora del robot los datos de los sensores del teléfono móvil y por último, una vez el robot tiene todos los datos disponibles, fusionarlos en una sola medida de posición y orientación.

Se ha decidido integrar los datos de la IMU del teléfono móvil, en particular giroscopos e inclinómetros, y de odometría del robot para obtener los desplazamientos y giros relativos a la posición de arranque del robot, que se tomará como origen del sistema de referencia global.

4.1.1. android_sensor_driver

Se necesita que la computadora del robot pueda acceder a las medidas de los sensores del teléfono móvil. Se ha utilizado la misma aplicación que utiliza la tableta como *joystick* [1]. En el caso del teléfono se transmite la información de la IMU, magnetómetro, GPS, barómetro y fotosensor.

El hecho de que ambos dispositivos utilicen la misma aplicación hace imposible distinguir de cual de los dos proviene la medida recibida y, puesto que en el entorno de ROS no pueden existir dos nodos con el mismo nombre, las aplicaciones entran en conflicto, funcionando exclusivamente la última que se lance. La solución para estos problemas ha sido recompilar la aplicación renombrando, de forma diferente para la tableta y para el móvil, los *topics* en los que publican y dando diferentes nombres a los nodos en los diferentes dispositivos.

Para poder compilar la aplicación ha sido necesario instalar el entorno de programación de aplicaciones Android para ROS mediante los siguientes pasos:

1. Instalar Oracle java `http://www.ubuntu-guia.com/2012/04/instalar-oracle\-java-7-en-ubuntu-1204.html`

   ```
   sudo add-apt-repository ppa:webupd8team/java
   sudo apt-get install oracle-java8-installer
   ```

   Para cambiar entre las alternativas:

   ```
   sudo update-alternatives --config java
   ```

2. Descargar android-studio `https://developer.android.com/sdk/index.html`, instalarlo en el equipo, ir a "SDK Manager" y descargar:

   - Android SDK Build-tools 21.1.2
   - API 10
   - API 15

3. Descargar y compilar *android_core*:

```
git clone https://github.com/rosjava/android_core.git
cd android_core
gedit local.properties
sdk.dir=<path to android sdk>
cmake .
make
```

4. Compilar *android_sensors_driver*:

   Clonar *android_sensors_driver* en la misma carpeta que esta *rosjava*.

   ```
   git clone https://github.com/chadrockey/android_sensors_driver.git
   ```

   Modificar el archivo *settings.gradle* de la carpeta *android_core*, añadir *android_sensors_driver* al final de la lista de *#include*. Una vez hecho esto se puede compilar:

   ```
   cmake .
   make
   ```

   Una vez se tiene el entorno instalado y se puede compilar la aplicación, se ha procedido a cambiar el nombre de los *topics* y de los nodos y se ha instalado en el teléfono móvil, repitiendo el proceso con diferentes nombres para la tableta.

5. Instalar la *apk* mediante android-studio conectando el dispositivo con el modo *debug* activado.

4.1.2. robot_localization

Para la fusión de las medidas de los sensores y la estimación de la posición se ha utilizado el paquete de ROS *robot_localization* [4] que contiene diversos nodos que, mediante una implementación no lineal del filtro de Kalman, estiman el estado del robot en su movimiento 3D. El paquete posee dos implementaciones diferentes, *ekf_localization_node* que integra un *extended Kalman filter* y *ukf_localization_node* que integra un *unscented Kalman filter* y esta última es la que se ha utilizado. El paquete contiene otro nodo, *navsat_transform_node*, que ayuda en la integración de los datos del GPS del teléfono.

Las características a destacar de este paquete son:

- Fusión de un número ilimitado de sensores. Los nodos no limitan el número de entradas de datos.

- Soporte para múltiples tipos de mensajes de ROS: *nav_msgs/Odometry, sensor_msgs/Imu, geometry_msgs/PoseWithCovarianceStamped, o geometry_msgs/TwistWithCovarianceStamped.*
- Selección de los datos de entrada. Si un sensor dado proporciona información que no se quiere integrar en la estimación, los nodos de este paquete permiten excluirla.
- Estimación continua. Los nodos de *robot_localization* comienzan a estimar el estado del vehículo en cuanto reciben una primera medida. En el caso de que los sensores dejen de proveer información durante un tiempo *robot_localization* continuará estimando el estado del robot mediante un modelo cinemático interno.
- Estimación 15-dimensional. Los nodos contenidos en este paquete pueden estimar el estado del vehículo en 15 dimensiones (x, y, z, *roll*, *pitch*, *yaw*, x', y', z', *roll'*, *pitch'*, *yaw'*, x'', y'', z'').

Para poder utilizar el paquete es aconsejable seguir los siguientes pasos:

1. Instalar el paquete:

```
source /opt/ros/indigo/setup.bash
cd catkin_ws/src
git clone --recursive https://github.com/cra-ros-pkg/robot_localization
      --branch indigo-devel
cd ..
catkin_make -DCMAKE_BUILD_TYPE=Release
```

2. Modificar uno de los ejemplos de *launch* que el paquete trae y adaptarlo a Andábata. Tener cuidado al integrar las aceleraciones lineales, pueden hacer inestable la estimación de la posición.

   En concreto se han fusionado las medidas de velocidad lineal en el eje X(para indicar explicitamente que es siempre 0) e Y, y ángulos y la velocidad de giro en *yaw* de la odometría del robot y de la IMU los giros en *roll* y *pitch* y la velocidad de giro en *yaw*. Se descartan las medidas de los acelerometros. Estas medidas no se van a incorporar de forma diferencial, ya que lo que el paquete hace en este modo sería diferenciar las medidas consecutivas e integrar los resultados. Las medidas de la odometría se fusionaran de forma relativa a la posición inicial, pero las de la IMU no. Esto hará que al arrancar el robot siempre se encontrará en el origen de coordenadas, pero con inclinación real.

3. Definir ejes de referencia siguiendo lo especificado en `http://www.ros.org/reps/rep-0105.html`

   Se tendrán uno o dos ejes coordenados globales ("*map*", fijo al mundo, y "*odom*"), uno asociado a la base del robot y uno asociado a cada sensor. El árbol de transformadas debería ser "$map \rightarrow odom \rightarrow base\_link \rightarrow sensors\_links$" (ver Figura 3). Si no se va a usar ningún tipo de posicionamiento global se pueden eliminar los ejes de referencia "*map*" y establecer "*odom*" como los ejes fijos.

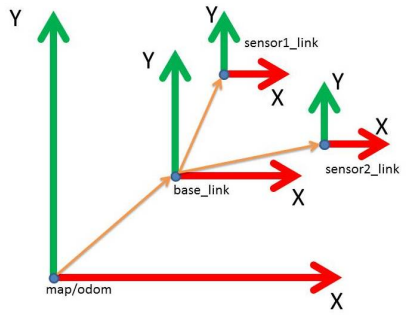

Figura 3: Ejes de referencia estandar.

   En Andábata se han seguido estas directrices. Se ha tomado "*odom*" como los ejes de referencia fijos, se ha fijado a la base del robot los ejes "*base_link*" y para el teléfono móvil "*imu_link*".

4. Publicar las medidas de los sensores teniendo en cuenta las siguientes consideraciones:

   - Verificar que las medidas de los sensores son publicadas con el formato de mensaje de ROS correcto.
   - Rellenar la covarianza de las medidas con valores adecuados, diferentes de cero. Esto es, cuanto más fiables sean las medidas mas pequeña será la covarianza y siempre trabajando con el orden de magnitud de la medida.
   - Asegurarse que los sensores siguen el convenio del paquete sobre donde se encuentra el cero de la medida. Por ejemplo, las medidas de las IMUs deben de estar referenciadas al sistema ENU(por sus nombre en inglés *East*, *North*, *Up*, referiendose a las direcciones y sentidos de los ejes X, Y y Z respectivamente). Revisar la documentación de la página: `http://wiki.ros.org/robot_localization/Tutorials/Preparing%20Your%20Sensor%20Data`

- Utilizar el nodo *static_transform_publisher* del paquete *tf2_ros* para publicar los ejes de referencia de los sensores, relativos a los ejes coordenados "*base_link*".
- Usar la nomenclatura de *tf2* para declarar los ejes coordenados, es decir, no utilizar barras oblicuas al principio de los nombres. Esto debe ser aplicado en los mensajes que publican los sensores ("frame_id" y "child_frame_id"), en los parámetros del *launch* y en los nodos *static_transform_publisher*.

La aplicación *android_sensor_driver* publica las medidas en los tipos de mensaje correctos, pero se ha tenido que rellenar la covarianza de las medidas con unos valores aproximados y adaptar la nomenclatura de los ejes de referencia a lo especificado por el paquete *tf2_ros*. En el caso de la IMU, no tiene el cero de la medida donde el paquete especifica, por lo que la transformación "$base\_link -> imu\_link$" contiene, aparte de una translación, una rotación.

5. No publicar la transformada "$odom -> base\_link$", lo hará *robot_localization*.

4.2. Agrupación de barridos 2D

Se ha actualizado la interfaz con la base del sensor 3D creando un nodo, llamado *uno_motion_node*, que se encarga del envío de órdenes y de la recepción y publicación de la posición del sensor 2D de forma más eficiente.

Para agrupar los barridos se ha creado un nodo *cloudMaker_node*, que captura los barridos 2D, los posiciona de forma tridimensional respecto al eje de referencia del robot en el inicio del barrido y los acumula, hasta que se ha completado un barrido 3D. Este nodo publica una nube de puntos 3D correspondiente a un barrido completo del telémetro láser 3D (una vuelta completa del sensor 2D) referenciada al comienzo de la toma del barrido.

Se ha creado un nuevo paquete de ROS, llamado *uno_motion_driver*, que contiene ambos nodos y la librería *uno_motion_driver* de la que dependen.

4.2.1. uno_motion_driver

Se ha creado una librería *uno_motion_driver* donde se definen dos clases. *UNO-Motion* representa la base del telémetro láser 3D, y contiene los métodos necesarios para interactuar con ella. La otra clase definida, *CloudMaker*, está basada en el nodo *laserScan_to_cloud2* ya descrito en [1].

La clase *UNOMotion* representa un telémetro láser 3D *UNO-Motion* dados sus parámetros de calibración intrínsecos y extrínsecos [5]. Los métodos públicos de esta clase permiten recibir ordenes de un *topic* y enviarlas a la base, leer la posición del sensor 2D y publicar su posición, tanto en forma de eje coordenado como en forma de mensaje.

En la clase *CloudMaker*, para poder acumular los barridos 2D, primero se ha tenido que modificar los ejes asignados a los puntos, para que al transformar los puntos del formato *LaserScan* a *PointCloud2*, los ejes de referencia finales sean los propios del robot y no los del sensor 2D, como eran en la anterior versión.

Se han extendido las funcionalidades para que, además de publicar los mensajes *LaserScan*, procedentes del paquete *urg_node* [6], transformados en *PointCloud2*, acumule dichos mensajes, de forma que, cada vuelta del sensor 2D, publique una nube de puntos 3D con la información de un barrido 3D completo.

4.2.2. uno_motion_node

El nodo *uno_motion_node* hace uso de la librería para crear una instanciación de la clase *UNOMotion* con los parámetros de calibración propios del sensor que tiene instalado Andábata y establece la comunicación entre el entorno de ROS y la base del telémetro láser 3D.

4.2.3. cloudMaker_node

El nodo *cloudMaker_node* instancia un objeto de la clase *CloudMaker* y queda a la espera de la llegada de los mensajes.

5. Resultados experimentales

Se ha hecho circular el robot por un tramo de carretera en el que, en un determinado momento, comienza una cuesta hacia abajo (ver Figura 4). Mientras se desplaza el robot publica un barrido 3D completo cada media vuelta de la cabeza del telémetro láser 3D.

Figura 4: Entorno en el que se ha realizado el experimento.

Con este experimento se quiere comprobar que los barridos se encuentran nivelados incluso cuando el robot no lo está y que la composición del mismo es coherente.

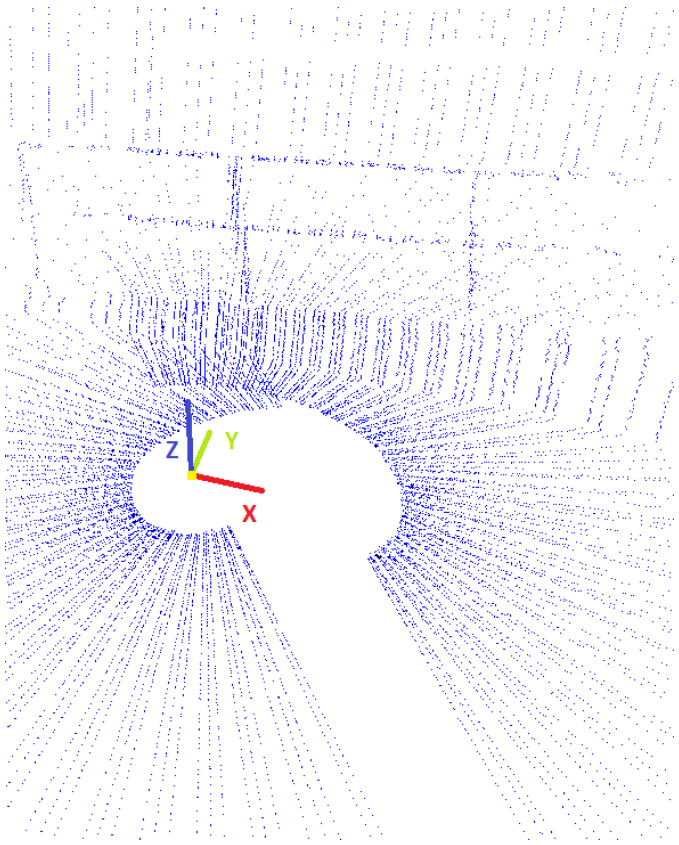

Figura 5: Barrido 3D tomado con media vuelta del sensor 2D mientras el robot está en movimiento (ejes del vehiculo).

Como se puede ver en la Figura 5, debido al desplazamiento del robot y que el barrido se realiza con media vuelta del sensor 2D, el barrido 3D no contiene puntos de todo el terreno que rodea al robot, pudiendo ser un problema a la hora de construir mapas. En la figura también se puede observar la zona ciega del sensor, centrada en el robot. No obstante, el barrido es coherente, bien formado y nivelado.

Se ha modificado el experimento para realizarlo de forma que cada barrido 3D contenga los datos de una vuelta completa del sensor 2D, a la vez, se ha aumentado la velocidad de giro para que el tiempo de captura no aumentase tanto.

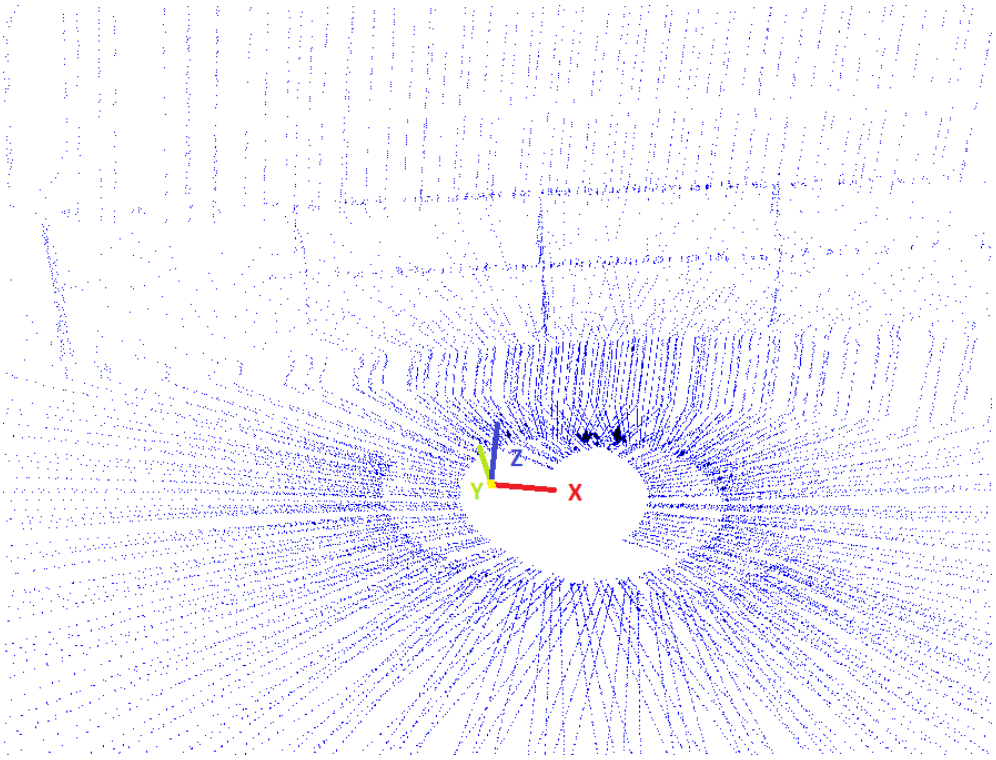

Figura 6: Barrido 3D tomado con una vuelta del sensor 2D mientras el robot está en movimiento (ejes del vehiculo).

En la Figura 6 se puede ver que, en este caso, el terreno circundante está completamente escaneado y con ello se evita el problema anteriormente descrito. Además, se puede comprobar que estos cambios no afectan al nivelado ni a la correcta formación del barrido 3D.

Además, en el video que acompaña a este informe, se puede observar que los sucesivos barridos 2D, a lo largo de la trayectoria de ida y vuelta del robot, se superponen sin apenas errores gracias al posicionamiento 3D.

6. Referencias

[1] M. Zafra, "Construcción de mapas de exteriores mediante octrees para un robot móvil equipado con un telémetro láser 3D," Proyecto Final de Carrera (E.T.S.I. Industrial), Universidad de Málaga, 2015.

[2] C. Rockey y A. Furlan, "ROS Android Sensors Driver." `https://github.com/chadrockey/android_sensors_driver`.

[3] A. Mandow, J. L. Martínez, J. Morales, J. L. Blanco, A. García-Cerezo, y J. González, "Experimental Kinematics for Wheeled Skid-Steer Mobile Robots," en *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, (San Diego, USA), pag. 1222 – 1227, 2007.

[4] T. Moore y D. Stouch, "A Generalized Extended Kalman Filter Implementation for the Robot Operating System," en *Proc. 13th International Conference on Intelligent Autonomous Systems*, (Padova, Italia), 2014.

[5] J. L. Martínez, J. Morales, A. J. Reina, A. Mandow, A. Pequeño-Boter, y A. García-Cerezo, "Construction and Calibration of a Low-cost 3D Laser Scanner with Full Field-of-view for Mobile Robots," en *Proc. IEEE International Conference on Industrial Technology*, (Sevilla, España), pag. 149–154, 2015.

[6] C. Rockey y A. Furlan, "urg_node." `https://github.com/ros-drivers/urg_node`.