eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA EN TECNOLOGÍA INDUSTRIAL

MANUAL DE USUARIO

PUESTA EN MARCHA Y ACTUALIZACIÓN DE PLATAFORMA ROBÓTICA MÓVIL BASADA EN ROS KINETIC

Alumno/Alumna: <Blázquez Muguerza, Eneko>

Directo/Directora (1): <Irigoyen Gordo, Eloy>

Director/Directora (2): <Larrea Sukia, Mikel>

Curso: <2018-2019>

Fecha: <XXXX, día, mes, año>

Índice

Ilustraciones:

1. DATOS BÁSICOS DEL PROYECTO

- Alumno: Eneko Blázquez Muguerza.
- Director (1): Eloy Irigoyen Gordo.
- Director (2): Mikel Larrea Sukia.
- Ámbito: Control inteligente.
- Título: Puesta en marcha y actualización de plataforma robótica móvil basada en ROS Kinetic.
- Resumen: Se ha realizado una puesta en marcha y actualización de una plataforma móvil terrestre modelo Summit, de Robotnik, en ROS Kinetic. Para la conducción manual del modelo simulado, se le ha añadido un sistema de sensorización, asi como otros parámetros para la generación de mapas del entorno.
- Palabras clave o etiquetas: ROS (Robot Operating System), Summit, robot, kinect, control remoto, navegación, control inteligente.

2. ESTRUCTURA DEL DOCUMENTO

Este documento que se va a desarrollar como añadido del trabajo principal está dirigido a todo tipo de usuarios. Serían suficientes unos pequeños conocimientos del sistema operativo Ubuntu, para utilizar las diferentes aplicaciones que se usan en este proyecto, ya que se pretende explicar de forma sencilla la utilización de las diferentes aplicaciones desarrolladas en él. Las instrucciones están orientadas a un usuario que carezca de conocimientos sobre la temática del proyecto, es decir, sobre robótica, automática y programación.

El documento está dividido en diferentes apartados, los cuales corresponden a las diferentes fases del proyecto. Estas fases se explican con detalle en la memoria del proyecto.

En cada fase se han desarrollado diferentes aplicaciones y, a pesar de estar relacionadas entre sí en muchos aspectos, se ha creído apropiado exponer las instrucciones de uso de las aplicaciones por separado, ya que probablemente haya interés en utilizar solamente una aplicación concreta.

En caso de tener dudas acerca de cualquiera de los aspectos del proyecto en este manual, se recomienda consultar el resto de la documentación, ya que ofrece información amplia y detallada sobre todo él.

Para terminar, si hubiera alguna duda con respecto al uso del sistema operativo Ubuntu o algún otro elemento no relacionado directamente con el proyecto, se recomienda realizar una búsqueda en la web para aclarar dichas dudas. Incluir instrucciones sobre el sistema operativo en el documento lo extendería demasiado, y la idea del documento es un manual breve y conciso.

3. INTRODUCCIÓN

El presente proyecto a consistido en el desarrollo de diferentes aplicaciones para el control de la simulación de la plataforma terrestre móvil modelo Summit, de Robotnik. Un robot con forma de vehículo todoterreno diseñado específicamente para trabajos de investigación.

El robot Summit lleva incorporado un ordenador, en el cual esta instalada la plataforma Ubuntu además del software de programación de robots ROS. Aunque directamente no se vaya a utilizar este robot a lo largo del proyecto, es necesario describirlo. El trabajo que se hará será todo en el ámbito de la simulación, pero no obstante, se basaran todas las aplicaciones a este modelo de Summit.

El proyecto se divide en dos desarrollos principales diferentes, uno de ellos dedicado al control remoto del robot y sistemas de sensorización implementados, y el otro a la generación del mapa del entorno simulado a lo cual nos referiremos como “mapeo”. En los próximos puntos se explican los pasos a seguir para utilizar esta aplicación.

Para terminar, cabe mencionar que las aplicaciones se utilizan mediante la plataforma Ubuntu. No se explica en este documento cada comando en profundidad, únicamente se mencionan los comandos necesarios para el uso de las aplicaciones.

4. PAUTAS PARA LA CORRECTA INSTALACIÓN DE LOS PAQUETES

A continuación en este capitulo se redactaran los pasos a seguir para la correcta instalación de todos los paquetes necesarios para el desarrollo del proyecto.

A su vez, este capitulo servirá de guía instructiva para futuros proyectos que se quieran desarrollar en este campo. Los comandos deberán escribirse en el terminal de esta manera:

```
git clone https://github.com/Txumeko/summit_sim.git
git clone https://github.com/Txumeko/ackermann-drive-teleop.git
git clone https://github.com/Txumeko/depthimage_to_laserscan.git
git clone https://github.com/Txumeko/summit_mapping.git
git clone https://github.com/Txumeko/summit_localizer.git
git clone https://github.com/Txumeko/robotnik_sensors.git
git clone https://github.com/Txumeko/robotnik_purepursuit_planner.git
sudo apt-get install ros-kinetic-ackermann-msgs
sudo apt-get install ros-kinetic-robotnik-msgs
sudo apt-get install
sudo apt-get install ros-kinetic-gazebo-ros-pkgs
sudo apt-get install ros-kinetic-ros-control
cd
cd catkin_ws
catkin_make
```

Una vez terminada toda la instalación se realizará el comando “catkin_make” que actualizará todos los ficheros y confirmará si los paquetes se instalaron debidamente.

5. INSTRUCCIONES PARA EL CONTROL REMOTO

Se ha desarrollado un único paquete para el control remoto del robot. Este tipo de control se hará vía el teclado del ordenador remoto, en el cual trabajaremos únicamente. Gracias a este paquete es posible controlar la trayectoria del robot en el entorno de simulación que se cargara.

Para manejar el modelo del Summit en simulación, hay que trabajar en el ordenador remoto. Abrimos el terminal de comando de Ubuntu, y ejecutamos la siguiente serie de comandos (El "Ctrl+Shift+T" abre una nueva pestaña del terminal. Si se desea abrir una nueva ventana, el comando seria "Ctrl+Shift+N"):

```
roslaunch summit_gazebo summit_office.launch
Ctrl+Shift+T
rosrun rviz rviz
Ctrl+Shift+N
rosrun ackermann_drive_teleop keyop.py
```

De esta forma cargamos inicialmente el entorno de simulación con el modelo simulado del robot Summit en la oficina de Robotnik en Valencia (puesto que este es el mapa que hemos elegido). A continuación se abre la ventana RViz, una aplicación que nos permite visualizar lo que el robot percibe. En esta aplicación se tendrá que decidir que nodos queremos ver, ya sea el modelo del robot simulado, el laser que proyecta el sensor de dicho robot, etc. Por ultimo, se carga en una nueva ventana de terminal la aplicación para el control remoto de nuestro robot.

Como se ha escrito en la memoria del trabajo, el archivo "keyop.py" necesita de unas modificaciones en su estructura. Este archivo es el encargado de enviar la información necesaria a las ruedas de nuestro robot para que este se mueva a nuestro gusto. Para ello es imprescindible modificar el "topic" encargado de enviar dicha información. Inicialmente el "topic" esta casado con un nodo inexistente llamado /cmd_vel. Lo que debe hacerse es descubrir en nuestro robot cual es el nodo encargado de recibir la

información de movimiento. Principalmente debería encontrarse en la carpeta dedicada al control del robot. Una vez tengamos este código, en nuestro caso /summit_robot_control/command, únicamente habría que reemplazarlo por el que se encuentra en el archivo “keyop.py”.

En la próxima imagen se pueden ver todas las ventas que se abrirían, junto con la ventana del terminal dedicada al movimiento del robot.

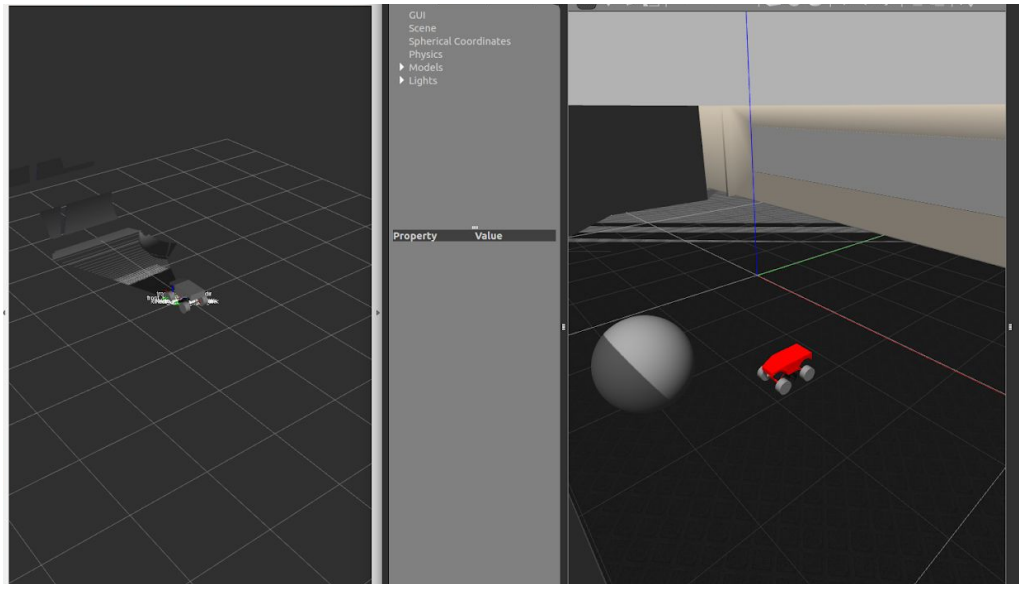

Ilustración 1: a la izquierda el programa RViz, a la derecha Gazebo

6. INSTRUCCIONES PARA LA IMPLEMENTACION DEL SISTEMA DE SENSORES

Como ya se ha expuesto en el documento general de la memoria del trabajo, inicialmente se decidió utilizar únicamente la cámara kinect como sistema de sensorización. No obstante se acabo implementando un segundo sensor para mejorar algunos aspectos del trabajo.

En este apartado veremos como cargar la simulación del modelo del robot implementando la cámara kinect, posteriormente el sensor hokuyo, y finalmente ambos a la vez.

Instalación de la cámara kinect

Para visualizar la cámara kinect en nuestro robot se modificaron los archivos “summit.urdf.xacro” y “summit.gazebo”. Este primer archivo se encarga de cargar todos los "topics" y nodos correspondientes al modelo del Summit, por eso hay que añadir todo el código relativo a la cámara kinect. Además, para poder visualizar la cámara en la pantalla de simulación, hay que modificar el segundo archivo añadiendo también el código relacionado con la cámara. Inicialmente la cámara aparecerá colocada debajo de nuestro robot, en lo que seria la coordenada (0,0,0), al querer mantener una relación lo mas posible con la realidad, se decidió mover la cámara simulada de manera que estuviera sobre nuestro robot y adelantada un poco para que la visión de la cámara fuera mas precisa. Además es necesario modificar en la carpeta referente al control del robot, que se ha añadido un sistema de sensorización. Para ello habría que modificar el archivo “summit_robot_control.launch” que esta incluido en el “.launch” principal, y añadir el “topic” correspondiente a nuestra cámara kinect, como por ejemplo <topicName>kinect_data</topicName>.

Para cargar la simulación del robot se deberían seguir estos comandos:

```
roslaunch summit_gazebo summit_kinect.launch
Ctrl+Shift+T
rosrun rviz rviz
Ctrl+Shift+T
```

Estos comandos cargarían la aplicación de simulación gazebo y RViz. En la primera veríamos el modelo del robot con la cámara integrada en la posición deseada. En la segunda, veríamos lo proporcionado por la cámara, es decir, la visión, la nube de puntos, y una vez integrado el paquete para la conversión al laser kinect, una franja roja que seria la simulación del laser de la cámara.

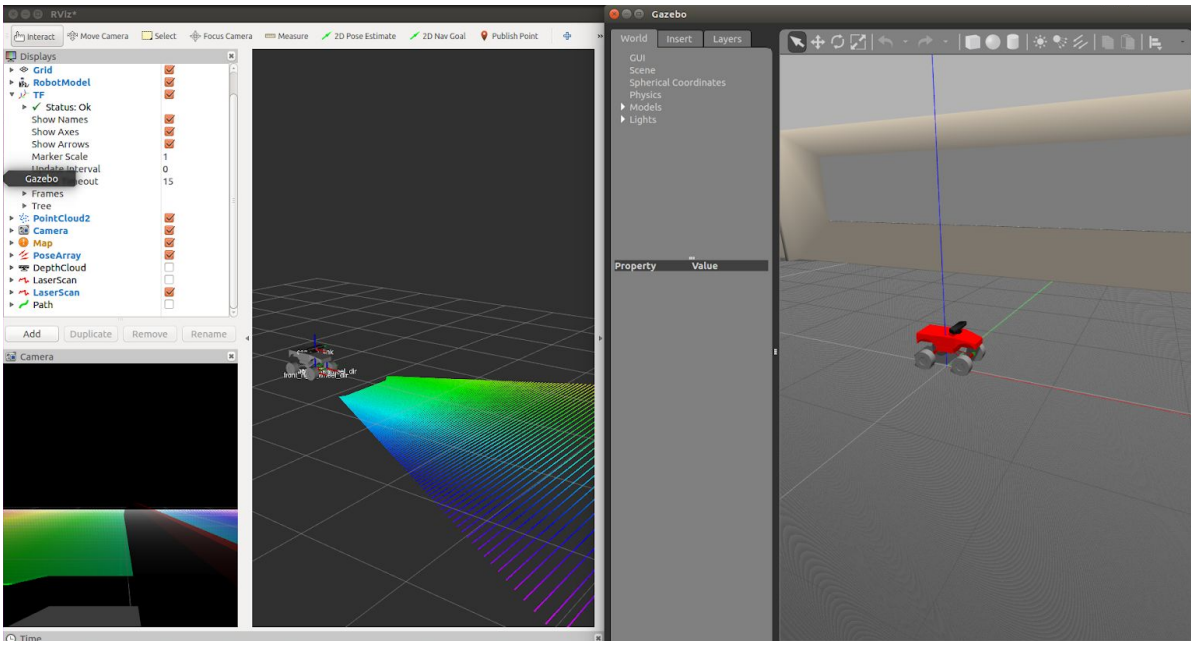

Ilustración 2: simulación del robot con la cámara kinect.

Implementación del paquete para la optimización del láser

Como se ha expuesto en el documento general de la memoria del trabajo de fin de grado, es necesaria la modificación de los parámetros obtenidos por parte de la cámara kinect para ser transformados en un laser óptico. Para ello se ha creado un paquete, el cual se presentara en este apartado, que realice dichas transformaciones.

Para empezar creamos un paquete llamado /depthimage_to_laserscan. Al cual se tendrá libre acceso desde la pagina de GitHub y el cual ha sido ofrecido en el capitulo anterior. En este paquete se creara una carpeta que servirá de almacén para el archivo ".launch" encargado de ejecutar los comandos requeridos.

A continuación de adjunta el código del archivo "di2ls.launch" creado para la conversión de los parámetro obtenido de la cámara kinect en un laser óptico.

```
<launch>

 <!-- DepthImage To LaserScan openner -->
 <node pkg="depthimage_to_laserscan" type="depthimage_to_laserscan"
name="depthimage_to_laserscan" args="load
depthimage_to_laserscan/DepthImageToLaserScanNodelet">
   <remap from="camera_info" to="/kinect/depth/camera_info" />
   <remap from="image" to="/kinect/depth/image_raw" />
   <remap from="scan" to="/depth_scan" />
   <param name="scan_height" value="10" />
   <param name="range_max" value="30" />
   <param name="output_frame_id" value="camera_depth_frame" />
 </node>

</launch>
```

Como se puede observar, se cogieron los “topics” de la información dada por la cámara, asi como la imagen, para crear uno nuevo llamado “/depth_scan”. Además se amplio el parámetro “range_max” de 3 a 30 para que su alcance fuera mayor.

Con todo esto el esquema de todos los “topics” seria el siguiente, en el que se puede apreciar como afecta este paquete en el proyecto.

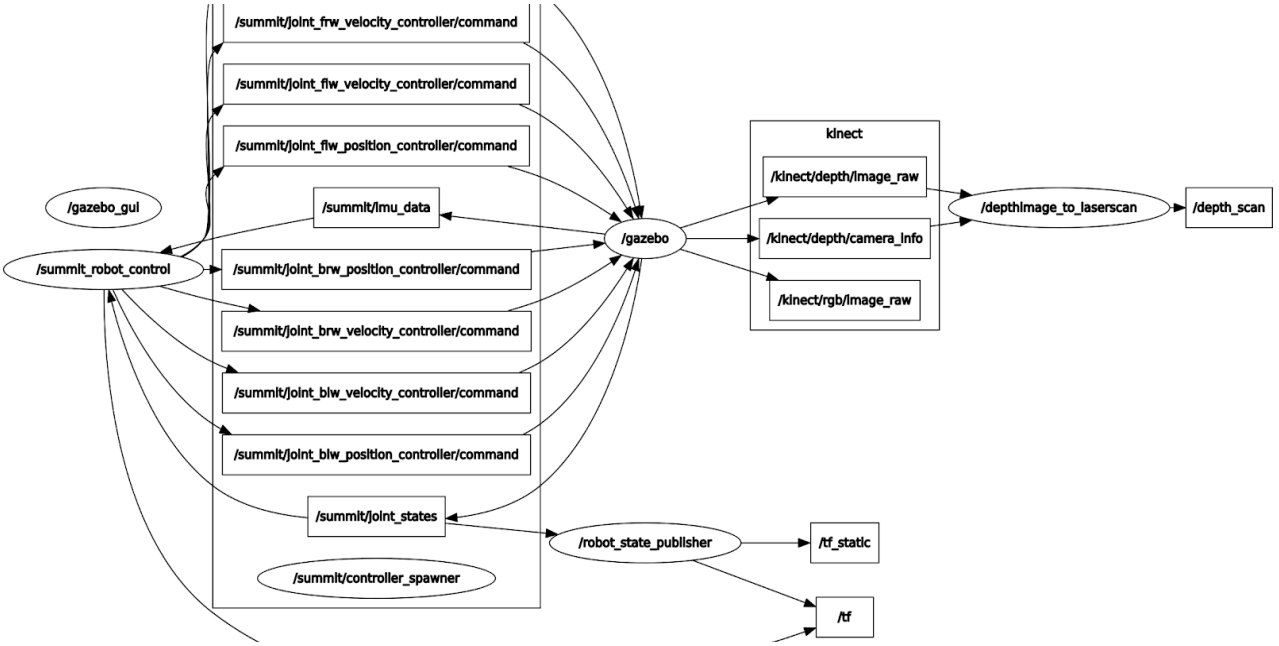

Ilustración 3: esquema de los “topics”

Instalación del sensor hokuyo

Exactamente de igual manera que hemos hecho con la cámara kinect, debería hacerse ahora con el sensor hokuyo. Primero añadir el código relativo en el archivo encargado de generar los nodos, “summit.urdf.xacro”; segundo añadir el código para cargar la simulación del sensor en el robot y colocarlo en un lugar preciso, “summit.gazebo”; y por ultimo añadir el “topic” correspondiente en “summit_robot_control.launch” como por ejemplo <topicName>hokuyo_data</topicName>.

Una vez tengamos todo modificado, los comandos a seguir serian los mismos que en el anterior. Estos comandos cargarían ahora ambos sensores en la simulación gazebo, y en la aplicación RViz, podrían verse todo lo relativo a los dos sensores.

```
roslaunch summit_gazebo summit_hokuyo.launch
Ctrl+Shift+T
rosrun rviz rviz
Ctrl+Shift+T
```

7. INSTRUCCIONES PARA LA GENERACION DEL MAPA DEL ENTORNO SIMULADO

Con motivo de finalizar este documento, se proporcionaran los comandos necesarios para la generación del mapa en el entorno simulado, o como antes hemos mencionado, “mapeo”.

Primero de todo se han creado dos paquetes en nuestro espacio de trabajo. La primera de ellas llamada summit_mapping, encargada de generar un mapa vacío que llenar con información para posteriormente guardarlo. La segunda carpeta se llamara summit_localizer, este paquete se encargara de conseguir mandarle al robot la directriz por si acaso este estuviera perdido en el mapa previamente guardado.

Para ello, en el primer paquete se creara una carpeta para el almacenaje de todos los mapas, y una carpeta con el archivo del lanzador. Como inicialmente se utilizo la cámara kinect para el mapeo, se crearon dos archivos, uno con el que mapear únicamente con el laser que proporciona la cámara kinect, y otro archivo con similares características para utilizar el sensor hokuyo. Como finalmente se prefiere utilizar este segundo sensor para dicha tarea, solo se mostraran los comandos requeridos para ello.

Además, cabe mencionar que se ha modificado el principal archivo lanzador de summit_office.launch a summit_double.launch. La razón de esto es únicamente que al realizarse distintas pruebas con ambos sensores, se crearon lanzadores específicos para cada sensor, por lo tanto, se creo a su vez un lanzador que simulara ambos sensores a la vez y se decidió darle ese nombre. En resumen, tendremos estos archivos: summit_office.launch, que solo generara la simulación del robot; summit_kinect.launch, que simulara el robot con la cámara kinect incorporada; summit_hokuyo.launch, que simulara el robot con el sensor hokuyo incorporado; y summit_double.launch, que cargara en el entorno de simulación el robot, la cámara kinect y el sensor hokuyo.

Como se ha dicho previamente en el documento de la memoria, gracias al aprendizaje realizado, se creo un archivo llamado “mapping.launch” el cual recibía la información

del laser de la cámara kinect. Como finalmente se utilizara el laser del sensor hokuyo, se creo otro archivo llamado “hokuyo.launch”. Para no interferir el uno con el otro, se ha acabado dejando ambos archivos por si en un futuro se quisiera implementar el “mapeo” con la cámara kinect.

Como ambos archivos son prácticamente idénticos, solo describiremos el que vamos a utilizar como sistema de generación de mapas en nuestro proyecto, el archivo “mapping_hokuyo.launch”:

```
<launch>
  <arg name="base_frame"  default="base_link"/>
  <arg name="odom_frame"  default="odom" />
```

```
<node pkg="gmapping" type="slam_gmapping"
name="slam_gmapping" output="screen">
   <param name="base_frame" value="$(arg base_frame)"/>
   <param name="odom_frame" value="$(arg odom_frame)"/>
   <param name="map_update_interval" value="5.0"/>
   <param name="maxUrange" value="6.0"/>
   <param name="maxRange" value="8.0"/>
   <param name="sigma" value="0.05"/>
   <param name="kernelSize" value="1"/>
   <param name="lstep" value="0.05"/>
   <param name="astep" value="0.05"/>
   <param name="iterations" value="5"/>
   <param name="lsigma" value="0.075"/>
   <param name="ogain" value="3.0"/>
   <param name="lskip" value="0"/>
   <param name="minimumScore" value="200"/>
   <param name="srr" value="0.01"/>
   <param name="srt" value="0.02"/>
   <param name="str" value="0.01"/>
   <param name="stt" value="0.02"/>
   <param name="linearUpdate" value="0.5"/>
   <param name="angularUpdate" value="0.436"/>
   <param name="temporalUpdate" value="-1.0"/>
   <param name="resampleThreshold" value="0.5"/>
   <param name="particles" value="80"/>

   <param name="xmin" value="-1.0"/>
   <param name="ymin" value="-1.0"/>
   <param name="xmax" value="1.0"/>
   <param name="ymax" value="1.0"/>

   <param name="delta" value="0.05"/>
   <param name="llsamplerange" value="0.01"/>
   <param name="llsamplestep" value="0.01"/>
   <param name="lasamplerange" value="0.005"/>
   <param name="lasamplestep" value="0.005"/>
   <remap from="scan" to="/hokuyo_scan"/>
  </node>
</launch>
```

Como se puede apreciar en el código, al inicio se configuran los parámetros que se van a recibir para la generación del mapa. Además en la ultima línea, también se aprecia que la información del escáner la saca del sensor hokuyo remapeando el "topic".

A continuación se ofrecerá la serie de comandos que han de seguirse para la utilización de este paquete:

```
roslaunch summit_gazebo summit_double.launch
Ctrl+Shift+T
rosrun rviz rviz
Ctrl+Shift+T
roslaunch summit_mapping mapping_hokuyo.launch
```

Al teclear estos comandos se abrirán la aplicación gazebo, en la que se podrá ver la simulación del robot con ambos sensores, y la aplicación RViz. Con el ultimo comando veremos únicamente en RViz como se ira generando una "mancha" blanca alrededor de nuestro robot. Esta "mancha" es lo que nuestro robot esta percibiendo gracias al sensor hokuyo y que una vez finalizada la tarea de mapeo habrá que guardarse.

Para pasar a guardar el mapa, sin cerrar ninguna de las pestañas de las terminales anteriores, deberá escribirse este comando en otra pestaña:

```
Ctrl+Shift+T
roscd summit_mapping/config
rosrun map_server map_saver map_1
```

Con estos comandos lo que haremos será guardar el mapa generado en la carpeta config que se sitúa dentro de la carpeta summit_mapping.

El paquete relacionado con la localización del robot se han seguido similares pasos. Se ha creado una carpeta dentro de summit_localizer encargada de almacenar los archivos ".launch".

En este caso como en el anterior también se generaron dos archivos, uno que cogiese la información del laser de la cámara kinect, y otro archivo que será el que expondremos, llamado "localizer_hokuyo.launch", el cual obtenía la información del sensor hokuyo. Este es el código que se ha generado para dicha tarea:

```
<launch>
 <arg name="use_map_topic"   default="false"/>
 <arg name="scan_topic"     default="/hokuyo_scan"/>
 <arg name="initial_pose_x"  default="0.0"/>
 <arg name="initial_pose_y"  default="0.0"/>
 <arg name="initial_pose_a"  default="0.0"/>
 <arg name="odom_frame_id"   default="odom"/>
 <arg name="base_frame_id"   default="base_link"/>
 <arg name="global_frame_id" default="map"/>
 <node pkg="amcl" type="amcl" name="amcl">
  <param name="use_map_topic"          value="$(arg use_map_topic)"/>
  <!-- Publish scans from best pose at a max of 10 Hz -->
  <param name="odom_model_type"        value="diff"/>
  <param name="odom_alpha5"            value="0.1"/>
  <param name="gui_publish_rate"       value="10.0"/>
  <param name="laser_max_beams"          value="60"/>
  <param name="laser_max_range"        value="12.0"/>
  <param name="min_particles"          value="500"/>
  <param name="max_particles"          value="2000"/>
  <param name="kld_err"              value="0.05"/>
  <param name="kld_z"                value="0.99"/>
  <param name="odom_alpha1"            value="0.2"/>
  <param name="odom_alpha2"            value="0.2"/>
```

```
  <!-- translation std dev, m -->
  <param name="odom_alpha3"          value="0.2"/>
  <param name="odom_alpha4"          value="0.2"/>
  <param name="laser_z_hit"          value="0.5"/>
  <param name="laser_z_short"        value="0.05"/>
  <param name="laser_z_max"          value="0.05"/>
  <param name="laser_z_rand"         value="0.5"/>
  <param name="laser_sigma_hit"      value="0.2"/>
  <param name="laser_lambda_short"   value="0.1"/>
  <param name="laser_model_type"     value="likelihood_field"/>
  <!-- <param name="laser_model_type" value="beam"/> -->
  <param name="laser_likelihood_max_dist" value="2.0"/>
  <param name="update_min_d"         value="0.25"/>
  <param name="update_min_a"         value="0.2"/>
  <param name="odom_frame_id"        value="$(arg odom_frame_id)"/>
  <param name="base_frame_id"        value="$(arg base_frame_id)"/>
  <param name="global_frame_id"      value="$(arg global_frame_id)"/>
  <param name="resample_interval"    value="1"/>
  <!-- Increase tolerance because the computer can get quite busy -->
  <param name="transform_tolerance"  value="1.0"/>
  <param name="recovery_alpha_slow"  value="0.0"/>
  <param name="recovery_alpha_fast"  value="0.0"/>
  <param name="initial_pose_x"       value="$(arg initial_pose_x)"/>
  <param name="initial_pose_y"       value="$(arg initial_pose_y)"/>
  <param name="initial_pose_a"       value="$(arg initial_pose_a)"/>
  <remap from="scan"                 to="$(arg scan_topic)"/>
 </node>
</launch>
```

Como se puede observar, se utiliza el “topic” del sensor hokuyo, de la odometria, la base del robot y se crea un “topic” llamado “/map” que será el encargado de hacer que se vea el mapa en el programa RViz. Se adjuntara a continuación el es que de todos los “topics” del sistema.

Los comandos a seguir serian los siguientes:

```
roslaunch summit_gazebo summit_double.launch
Ctrl+Shift+T
rosrun rviz rviz
Ctrl+Shift+T
roscd summit_mapping/config
rosrun map_server map_server map_1.yaml
Ctrl+Shift+T
roslaunch summit_localizer localizer_hokuyo.launch
```

Al igual que los anteriores comandos, este solo se vera reflejado en la aplicación RViz. En ella se vera a nuestro modelo de robot sobre un punto cualquiera del mapa generado anteriormente. Alrededor de nuestro robot se podrá apreciar una nube de puntos, los cuales se refieren a la posible localización del robot, pues este no sabe con gran exactitud donde se sitúa en toda esa nube. Aplicando el comando de movimiento, se podrá apreciar como el robot se ira recolocando a lo que el crea que esta. No obstante, siempre se puede utilizar el sistema manual ubicado en la barra de herramientas superior en la aplicación RViz llamado “2D Pose Estimate” para aconsejarle al robot en que posición y con que orientación se encuentra.