**ROS PACKAGES para Occupancy Grid based Graph SLAM and Exploration**

Usamos una serie de librerías en paquetes de ROS:

* **TORO**: es la librería de optimización de grafos mediante SGD (graph-SLAM) que se puede encontrar en openslam.org. Está sin modificar, no he cambiado nada, tan solo encapsulada en un paquete de ROS para que sea más fácil de linkar.
* **Matlib**: esta librería incluye la bastantes de la clases base de mi simulador MRXT. Como por ejemplo, calculo matricial con matrices redimensionables Ematrix (ideales para aumentar las matrices de estado en un EKF), gridmaps binarios para usar como máscaras de regiones (ESZ), e implementación de occupancy gridmaps con funciones para extraer fronteras, ESZ... Hay una implementación de esta clase que permite cargar un mensaje de ROS nav\_msgs::OccupancyGrid como estructura de datos. (Esta librería requiere tener instalado opencv y loki)
* **Gridmap\_description\_tools**: Incluye una serie de funciones para convertir nav\_msgs::OccupancyGrid a opencv y funciones para extraer características SIFT o SURF sobre el gridmap o sobre la transformada distancia del gridmap. También incluye asociación de datos entre características de dos gridmaps, y alineamiento de dichas características usando RANSAC o mínimos cuadrados.

Además tenemos una serie de paquetes de ROS que incluyen nodos ejecutables. Esto incluye nodos de control de movimientos de alto y bajo nivel, SLAM, scan matching....

* **Behaviour\_based\_local\_planner**: Este nodo realiza el control reactivo del robot. Incluye un mapping local con ventana deslizante usando el láser y un control basado en comportamientos usando dicho mapa. En realidad se generan 2 mapas, uno para navegación, y otro para SLAM. En el de SLAM tan solo se van proyectando las medidas que se obtienen del láser. Cuando la ventana se desplaza las nuevas celdas que aparecen en el mapa se consideran desconocidas. El mapa de navegación funciona de forma similar pero permite inicializar las celdas que entran a formar parte de la ventana al desplazarse desde un mapa global. El mapa de SLAM tan solo se publica, el de navegación es el que se usa en los comportamientos reactivos. Este nodo no hace nada de localización. Por lo tanto hay que especificarle que sistema de coordenadas está asociado al robot y cual se va a utilizar como sistema de referencia para la navegación. Aquí lo ideal es usar el sistema de referencia de la odometría o el del scan matcher si se quiere que el mapa local se genere con mucha más precisión.
* **Explore\_tree\_planner**: Este nodo implementa el planificador de alto nivel de mi arquitectura híbrida. Requiere de un mapa global. Según su planificación especificará al behaviour\_based\_local\_planner como debe funcionar.
* **scan\_tools/laser\_scan\_matcher**: Es un scan matcher para obtener una odometría basada en láser que es bastante precisa. Puede presentar problemas en pasillos donde no hay suficientes puntos de referencia. Este nodo no lo he programado yo. Funciona mediante ICP: *A. Censi, "An ICP variant using a point-to-line metric" Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2008*. Pero tenía un bug de sincronización que lo hacía fallar notablemente en cuanto la máquina está cargada. Utilizar esta versión modificada en lugar de la original.
* **graph\_optimizer\_slam**: Este nodo lee los mapas locales generados por el behaviour\_based\_local\_planner, utiliza el Gridmap\_description\_tools para describirlos mediante características, obtener transformadas entre mapas y optimizarlos mediante TORO. Finalmente fusiona todos los mapas y publica el occupancy grid global. Hay implementadas dos formas de fusión. La primera es desde el nodo más antiguo hasta el más moderno superponiendo todas aquellas celdas que sean ocuapdas o libres. La segunda implementación asigna a cada celda su valor correspondiente en el mapa del nodo más cercano. Actualmente solo funciona monorobot. TORO da un error si en el grafo hay partes inconexas. Habría que ir optimizando dos grafos por separado y juntarlos cuando hubiese suficiente información.
* **map\_fusion**: Este nodo sirve para fusionar 2 mapas de ocupación. Se suscribe a ambos y publica un tercero con los 2 mapas fusionados. Es lo que usé en el paper de las jornadas de automática 2013. En ese caso eran 2 gmapping independientes los que crean los mapas de 2 robots, y map\_fusion los fusiona. El procedimiento de alineamiento y fusión es similar al del graph\_optimizer\_slam pero con 2 mapas de mayores dimensiones.

Paquetes para pruebas:

* **carmen\_publisher**: sirve para publicar datos de láser y odometría procedentes de logs en formato CARMEN. No utiliza los timestamps del log sino que va publicándolos con un pequeño retardo. Hay algunos datos importantes que no aparecen en el log como por ejemplo el rango máximo y número de medidas del láser que están introducidos manualmente en el carmen\_publisher. (TODO: pasarlos como parámetros). Hay un log en formato carmen llamado odometria.log que tiene datos del pioneer moviéndose por el Quorum V.
* **explore\_test**: incluye los launch que use para hacer las pruebas para el paper de las jornadas. En este caso está la navegación funcionando con 2 robots simulados en Stage. Se usan 2 gmapping independientes para hacer el SLAM particular de cada robot y luego se alinean con map\_fusion. Funciona muy lenta con 4 núcleos. También hay una versión mono-robot para probar el algoritmo de exploración. La carpeta world incluye distintos mapas que se pueden cargar en Stage para como escenario virtual.
* **graph\_optimizer\_slam**: También hay una serie de launch files en este paquete para probar el graph-slam con occupancy grids. En este caso son monorobot. Hay un launch para probarlo con toda la exploración en marcha en Stage, otro para probar solo el slam con un rosbag de ros y otro para probar solo el slam con un log de carmen. Hay bags y los logs están en la carpeta dataset. Además hay un script para gnuplot para visualizar el grafo de forma continua.

>> gnuplot

>> load “plot.gnu”

* **arvc\_pioneer\_teleop**: permite teleoperar el robot mediante el teclado. Hay un launch en explore\_test para usarlo para poder grabar un rosbag con el robot moviéndose de forma teleoperada en Stage. El rosbag que hay en la carpeta dataset de graph\_optimizer\_slam está grabado así. También se podría usar el robot real para hacer un rosbag con datos reales.

Uso de ROS:

En la web de ros.org se puede encontrar como instalar ros. Los nodos están probados para la versión ROS groovy. Una vez instalado:

* Hay que iniciar *roscore* en la línea de comandos
* Se usa *rosrun nombre\_paquete ejecutable* para ejecutar un nodo
* Se usa *make* desde el directorio o *rosmake* *nombre\_paquete* para compilar
* Un roslaunch ejecuta varios nodos a la vez pudiendo pasar parámetros fácilmente:

*roslaunch nombre\_paquete launch\_file*

* Usar *rosrun rviz rviz* para visualizar. Este visualizador utiliza la tarjeta gráfica para mostrar gráficos en 3D y suele dar bastantes problemas de compatibilidad. En mi ordenador se cuelga bastante, incluso nada más iniciarlo. Es muy sensible a los cambios de tamaño de la ventana.