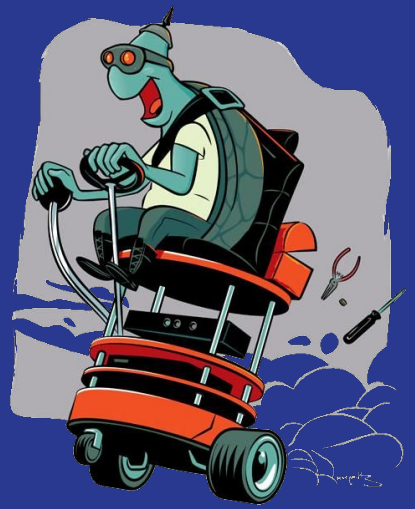
Proyecto: Navegación

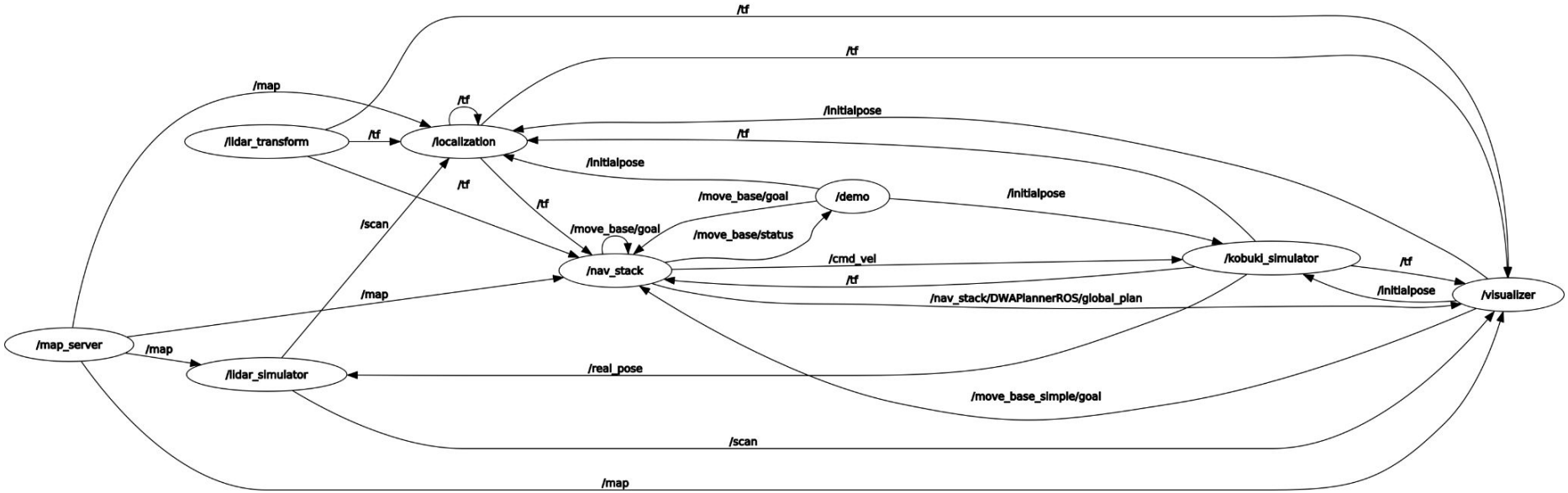
IIC2685 - Robótica Móvil

Equipo:

- Benjamín Farías
- Rafael Fernández
- Lukas Fuenzalida



Diseño del Software

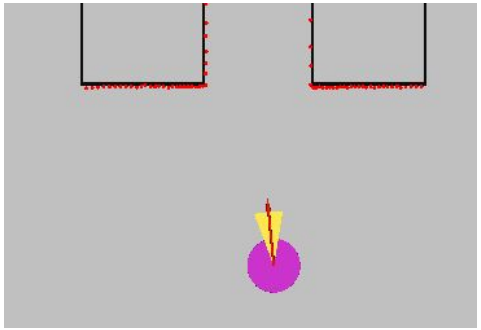


Stack de Navegación - Componentes

Simulador del Turtlebot: Las dinámicas del robot son modeladas utilizando el *Kobuki Simulator*, logrando una simulación que considera error en el movimiento.

Simulador de LIDAR: Las mediciones del sensor LIDAR son simuladas mediante el *LIDAR Simulator*, que considera errores de medición en los láser.

Servidor de Mapa: El nodo *map_server* carga el mapa y lo expone a los demás componentes del stack.



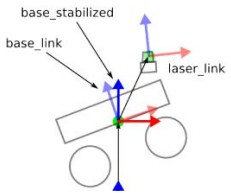
Stack de Navegación - Componentes

Transformación de Coordenadas: El nodo *static_transform_publisher* del paquete *tf* se encarga de trasladar las coordenadas desde el sensor hacia el centro del robot. En este **caso particular**, el **vector de transformación** es:

```
(x, y, z, yaw, pitch, roll) = (0, 0, 0, 0, 0, 0)
```

Esto se debe a que el sensor se encuentra **justo en el centro del robot**.

Visualizador: El nodo *rviz* contiene una interfaz que permite visualizar el mapa, robot, partículas y mediciones del sensor leyendo la información desde los tópicos respectivos.



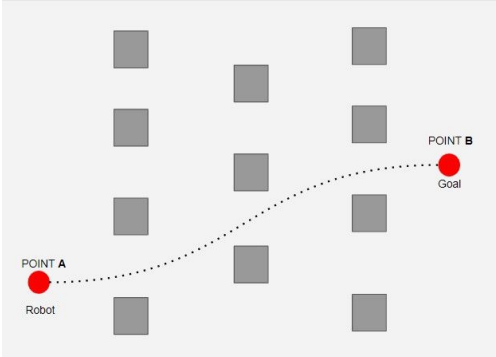
Stack de Navegación - Componentes

Localizador: El algoritmo corresponde al *Adaptive Monte Carlo Localization* (*AMCL*), que **adapta las muestras de partículas** para **mejorar la eficiencia y converger más rápido**. Se implementa a través del nodo *amcl* y tiene los siguientes **parámetros:**

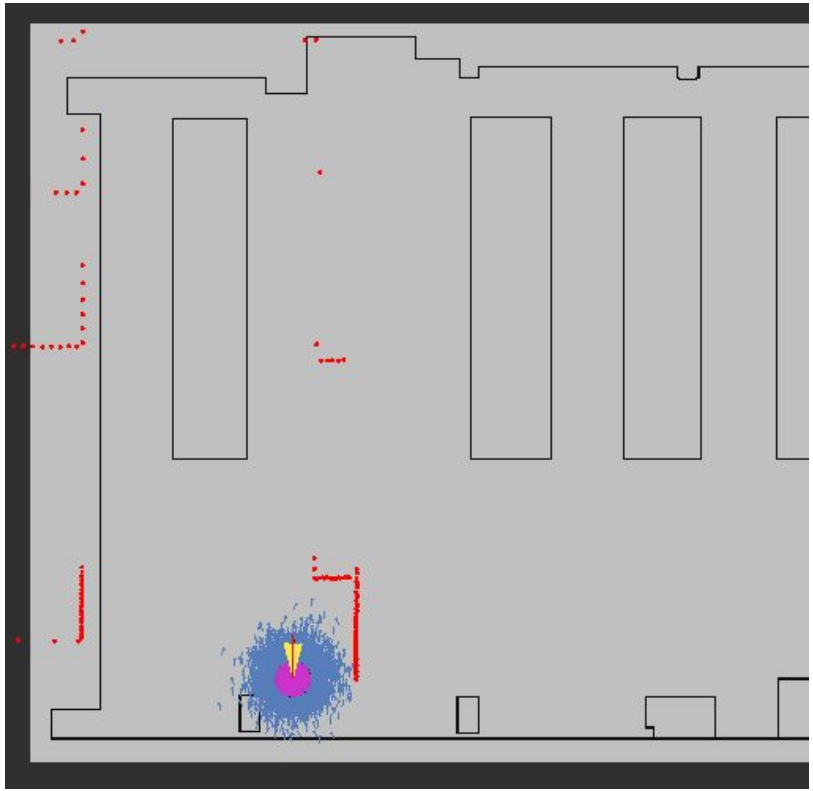
- **odom_model_type:** El modelo de movimiento. Utilizamos ***"diff"***, que considera el error tal como aparece en el libro *Probabilistic Robotics*.
- **use_map_topic:** Se coloca en **true** para que el modelo obtenga el mapa desde el servidor.
- **initial_pose_x/y/a:** Permiten inicializar el filtro con una distribución **Gaussiana** que asume una **estimación de la pose inicial** del robot. Usamos **(5.325, 0.814, 1.57)**.
- **laser_max_beams:** Cantidad de **haces** considerados para la medición del sensor. Usamos **181**.
- **min_particles: Mínimo de partículas** por iteración. Usamos **250**.

Stack de Navegación - Componentes

Navegación: La **planificación** de rutas y **seguimiento reactivo** del mejor camino vienen implementadas en el nodo *move_base*. Adicionalmente, se configuran **parámetros** de este nodo para ajustar el funcionamiento del **algoritmo de planificación y sus funciones de costo**. Estos parámetros se especifican en **tags** de tipo *<rosparam>*, ya que así es posible cargarlos desde archivos *yaml* que los **agrupan según su caso de uso**.

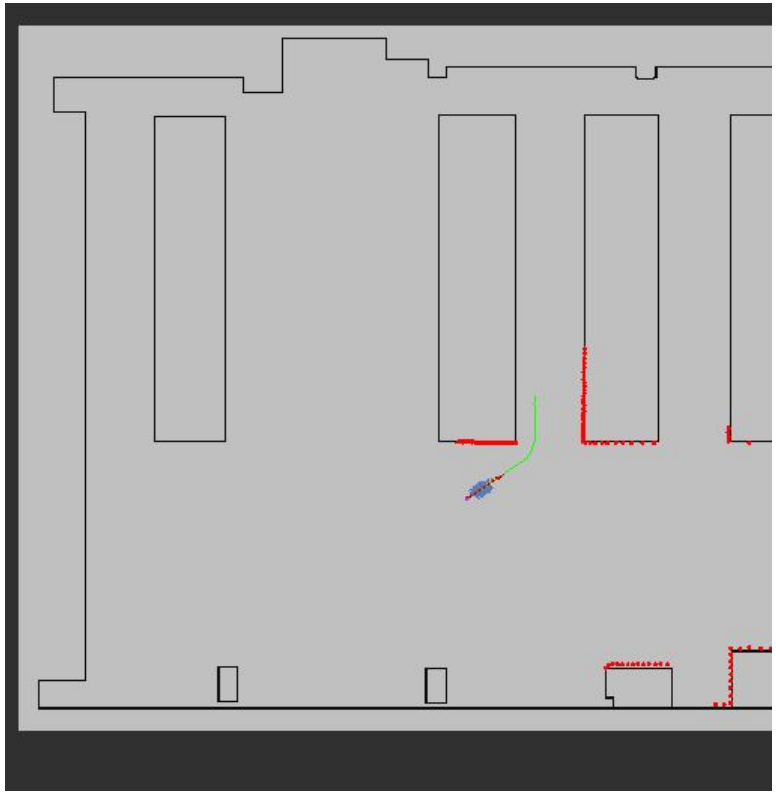


Visualizador - Estado Inicial



- Se observa la **pose inicial** del localizador
- Las partículas se ubican en una **nube** que rodea a la pose inicial
- Las lecturas del LIDAR son **incoherentes**, debido a que las poses iniciales del **Simulador** *Kobuki* y del **Localizador NO** están **sincronizadas**

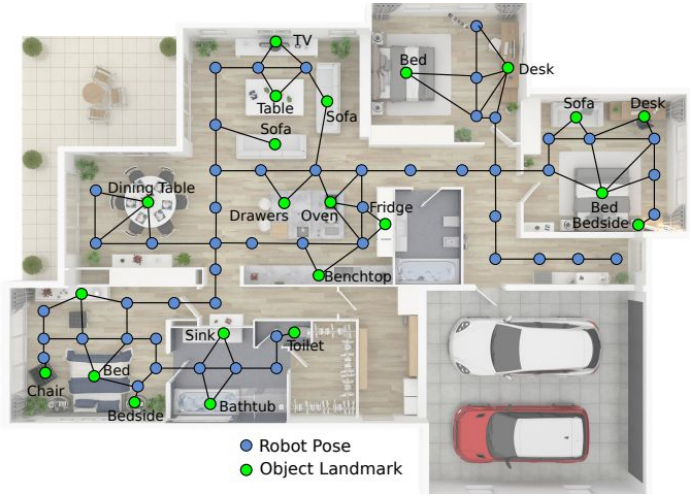
Visualizador - Navegación



- Al marcar un **destino**, el robot navega hacia él de forma **bastante fluida**, a excepción de las pausas menores que ocurren debido al **tiempo de procesamiento** del algoritmo
- El camino seguido es **óptimo** (o muy bueno), y es representado por la **línea verde** que se extiende desde el robot
- Partículas **convergen rápidamente**

Rutina de Navegación Programada

Se programó una **rutina de navegación** en la que dada una **pose inicial estimada del robot**, éste navega hacia **3 poses distintas en el mapa** (de forma **secuencial**). Esto se realiza aprovechando la **API** *actionlib* que viene implementada en **ROS**.



Conclusiones

- El stack de navegación de ROS logra **localizar** al robot constantemente dentro del mapa, y de manera muy **eficiente y rápida**.
- El stack de navegación de ROS logra **guiar** al robot desde una pose inicial a una de destino mediante el **mejor camino**.
- El stack de navegación de ROS permite que el robot se mueva y **reaccione ante obstáculos** en su camino.
- La **rutina de navegación programada** demuestra la gran utilidad del stack de navegación de ROS para lograr la realización de **tareas automáticas**.