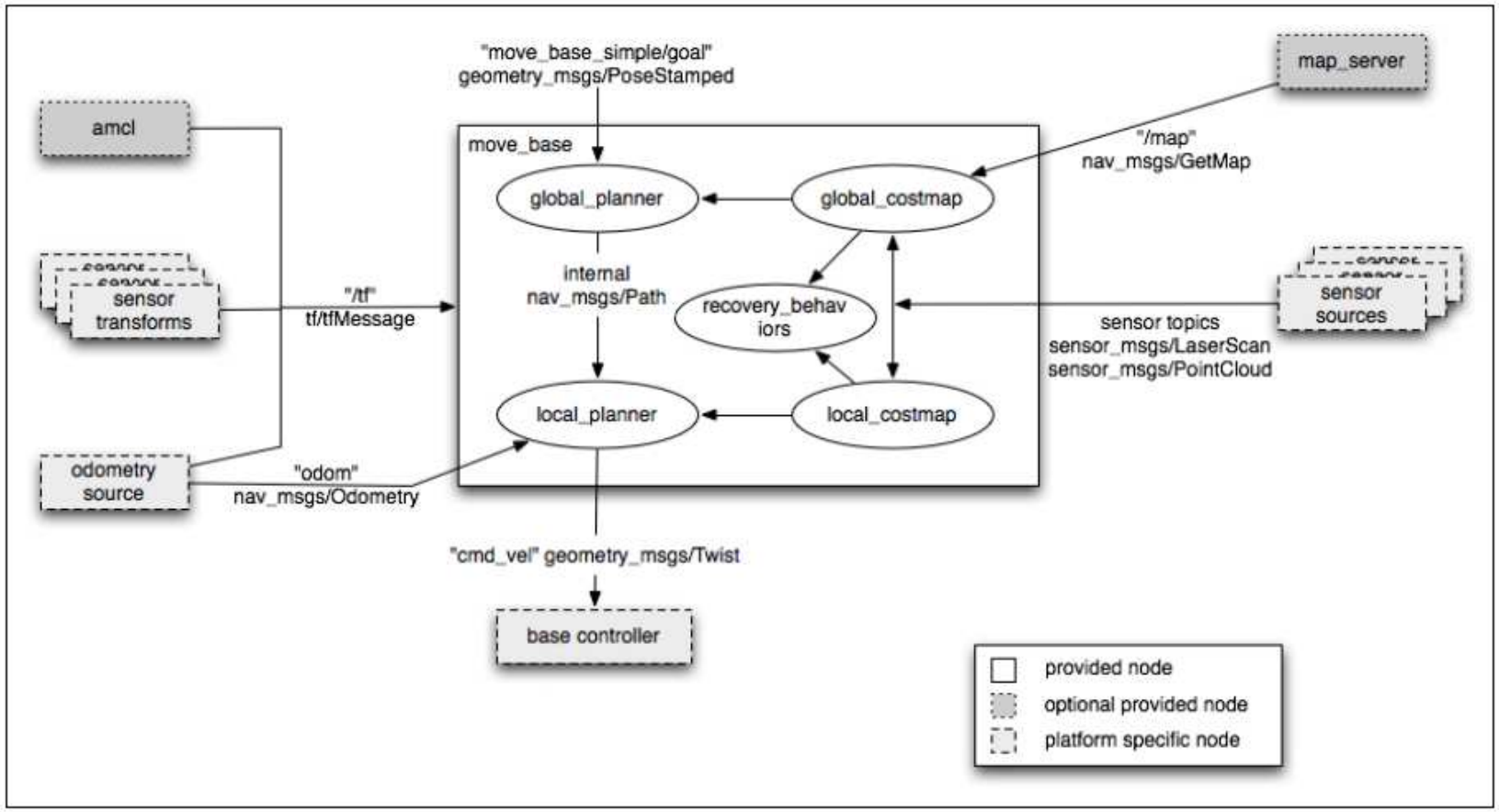
Navegación con Navigation Stack y Move_Base

- http://wiki.ros.org/navigation
  - Un stack de paquetes ROS que
    - a partir de información sobre odometría, sensores y una pose objetivo,
    - devuelve comandos de velocidad enviados a una base móvil de robot
  - Diseñada para mover cualquier robot móvil sin que se quede perdido ni choque.
  - Incorpora soluciones para la Navegación Global y Navegación Local con mapa.
  - **Instalar Navigation Stack:**
    - **sudo apt-get install ros-<distro>-navigation**

Navigation Stack



Navigation Stack Requirements

- Tres requisitos fundamentales:
  - *Navigation stack* solo maneja robots con ruedas con conducción diferencial y holonómicos.
    - Puede hacer algo más con robots bípedos, como localización, pro siempre que el robot no se mueva de lado
  - La base tiene que tener un laser montado para poder crear mapas y localizarse
    - O bien otro sensores equivalentes a los scans de un laser(como sonars o Kinect por ejemplo)
  - Funciona mejor con robots con forma aproximada cuadrada o circular.

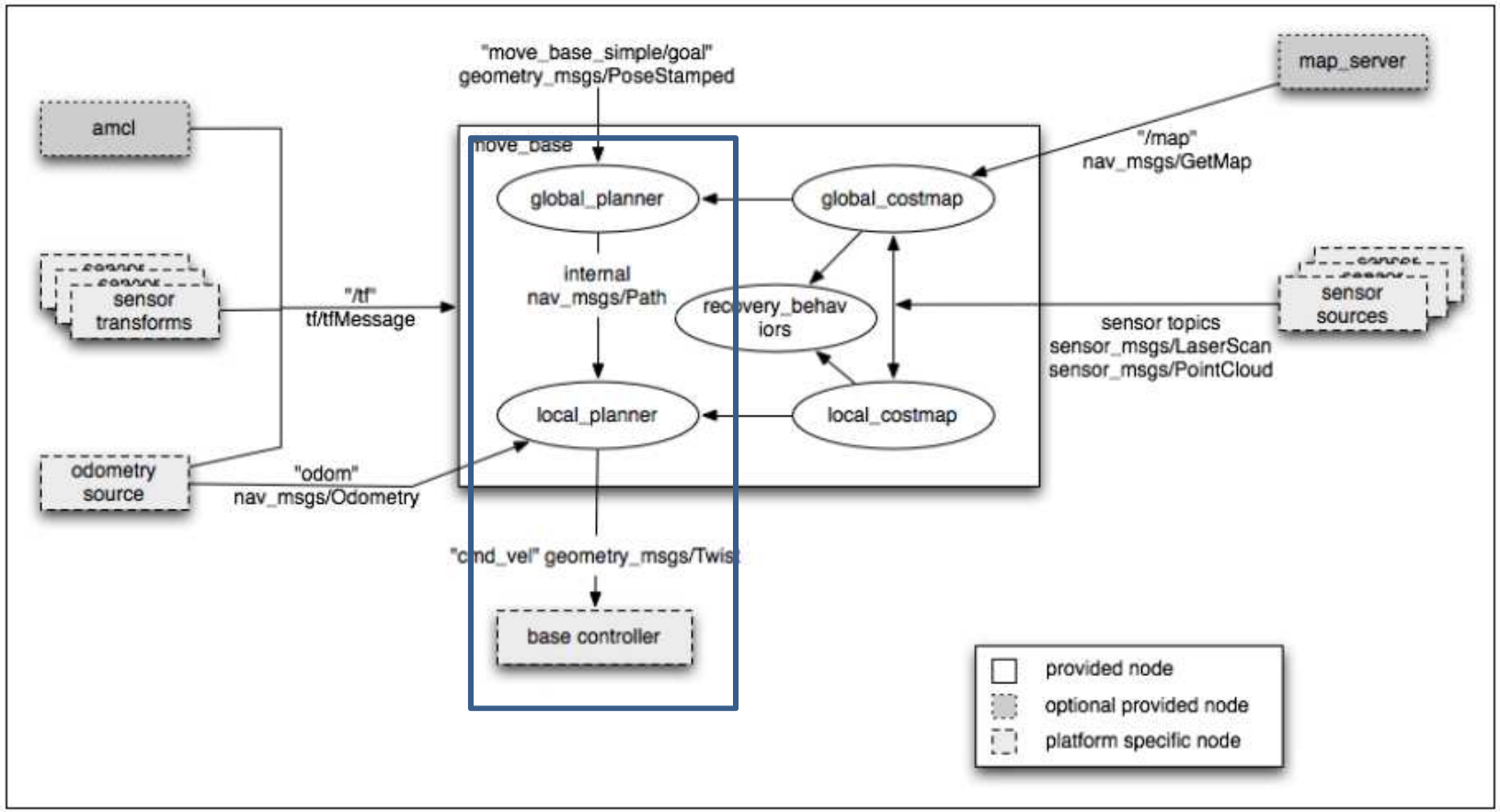
PAQUETE MOVE_BASE

move_base

- Este paquete permite mover el robot a una posición deseada usando ***navigation_stack***
- El *nodo move_base* aúna un *global_planner* y un *local_planner* para desempeñar la tarea de ***navegación global.***
- El *nodo move_base* puede realizar opcionalmente *recovery behaviours* cuando el robot percibe que está atascado.
- Basta con ejecutar un fichero *launch* porque el paquete ***move_base*** viene con la instalación de ROS.

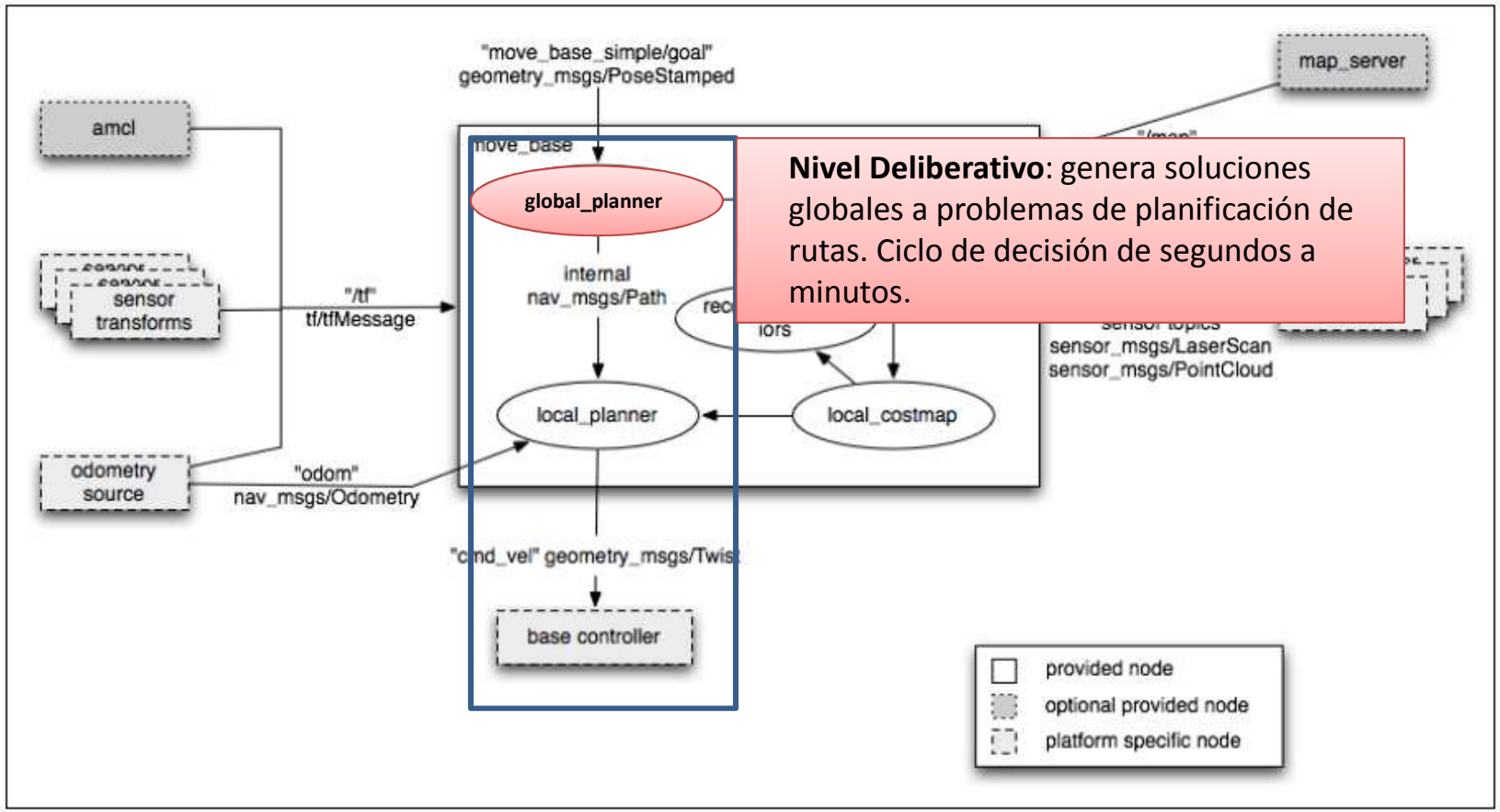
move_base package

- Move_base implementa una arquitectura híbrida de 3 niveles (recordar sesión anterior).



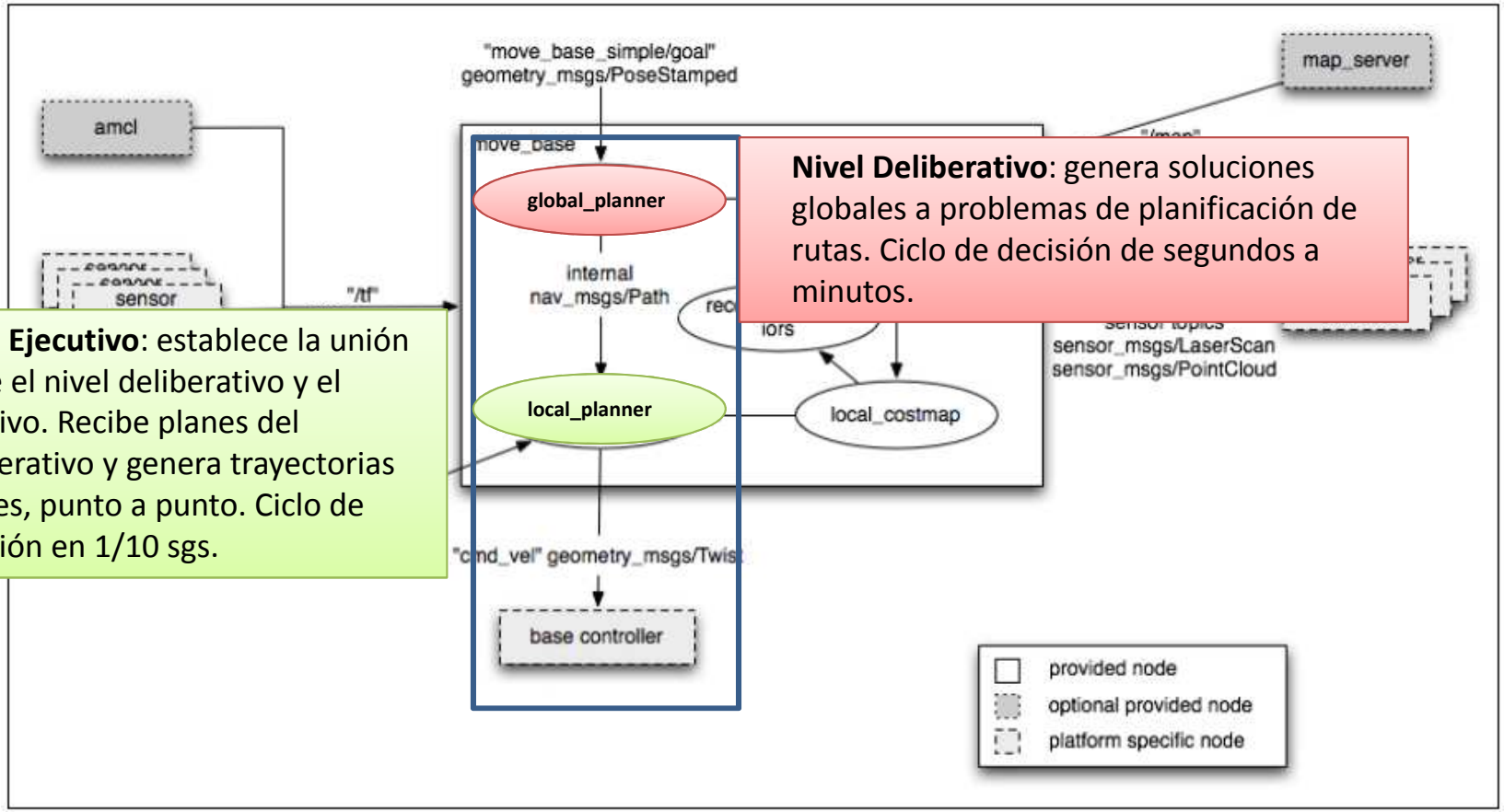
move_base package

- Move_base implementa una arquitectura híbrida de 3 niveles (recordar sesión anterior).



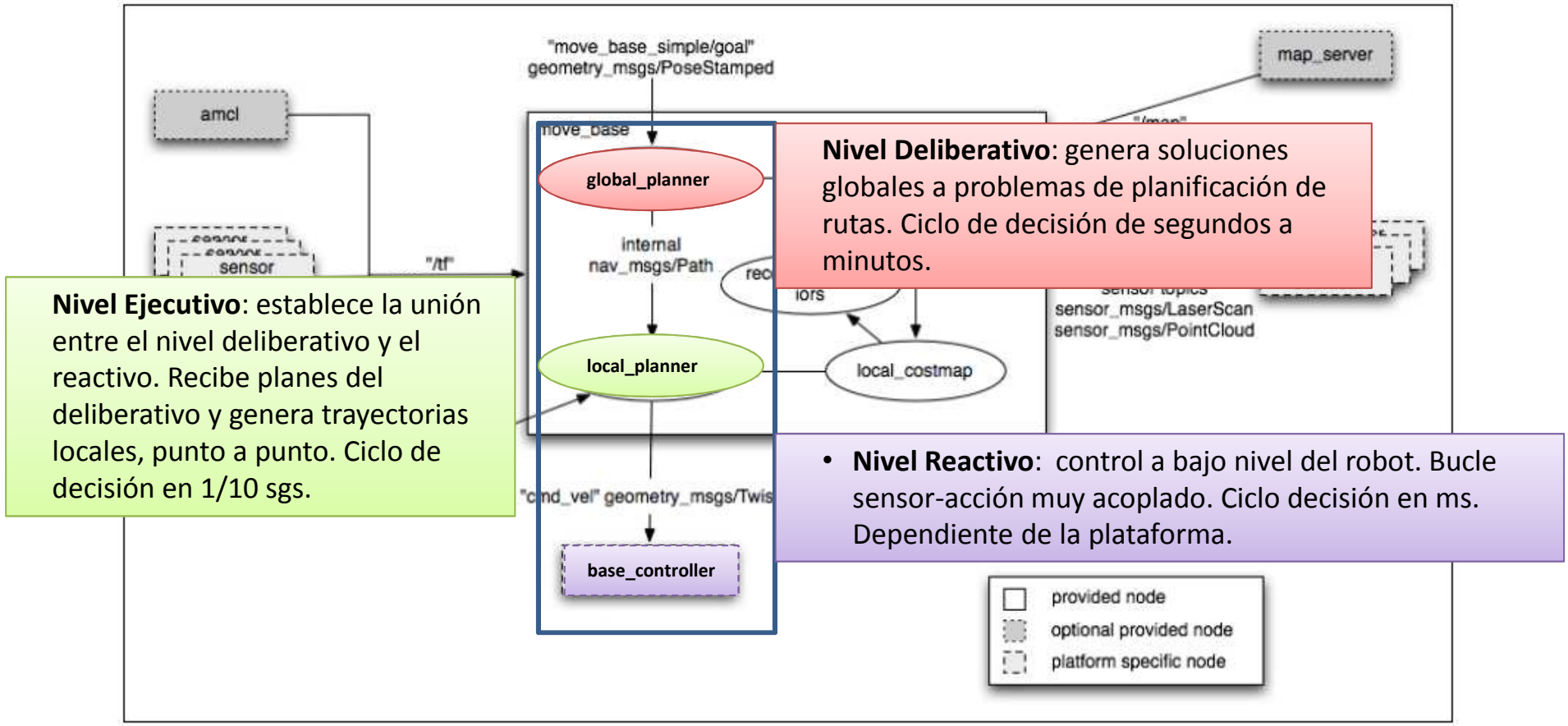
move_base package

- Move_base implementa una arquitectura híbrida de 3 niveles (recordar sesión anterior).



move_base package

- Move_base implementa una arquitectura híbrida de 3 niveles (recordar sesión anterior).



Ejecutar ROS navigation_stack con *Stage*

- Descargar los tutoriales de navegación en git (o directamente desde PRADO)
  - https://github.com/ros-planning/navigation_tutorials

```
$mkdir -p catkin_ws_navegacion/src
$cd catkin_ws_navegacion/src
$catkin_init_workspace
$cd ..
$catkin_make
$source devel/setup.bash
$cd src
$descargar aquí el fichero navigation_tutorials-hydro-devel.zip
$descomprimir aquí
$comprobar que hay un directorio en src "navigation_tutorials-hydro-devel"
$ cd ~/catkin_ws_navegacion
$catkin_make
```

- Crea un paquete ***navigation_stage*** *en el workspace .*
  - Contiene ejemplos de ficheros *launch* para ejecutar ***navigation_stack*** con distintas configuraciones.
- No olvidar source devel/setup.sh

Ejemplos de ficheros launch

Launch File	Description
launch/move_base_amcl_5cm	Example launch file for running the navigation stack with **amcl** at a map resolution of 5cm
launch/move_base_fake_localization_10cm.launch	Example launch file for running the navigation stack with **fake_localization** at a map resolution of 10cm
launch/move_base_multi_robot.launch	Example launch file for running the navigation stack with multiple robots in stage.
launch/move_base_gmapping_5cm.launch	Example launch file for running the navigation stack with **gmapping** at a map resolution of 5cm

Contenido: move_base_amcl_5cm.launch

```
<launch>
 <master auto="start"/>
 <param name="/use_sim_time" value="true"/>
 <include file="$(find navigation_stage)/move_base_config/move_base.xml"/>
 <node name="map_server" pkg="map_server" type="map_server" args="$(find
navigation_stage)/stage_config/maps/willow-full-0.05.pgm 0.05" respawn="false" />
 <node pkg="stage_ros" type="stageros" name="stageros" args="$(find
navigation_stage)/stage_config/worlds/willow-pr2-5cm.world" respawn="false" >
   <param name="base_watchdog_timeout" value="0.2"/>
 </node>
 <include file="$(find navigation_stage)/move_base_config/amcl_node.xml"/>
 <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
navigation_stage)/single_robot.rviz" />
</launch>
```

- Ejecutar:

```
$ roslaunch navigation_stage move_base_amcl_5cm.launch
```

move_base_amcl_5cm.launch

```
<launch>
  <master auto="start"/>
  <param name="/use_sim_ti
  <include file="$(find navigation_stage)/move_base_config/move_base.xml"/>
  <node name="map_server" pkg="map_server" type="map_server" args="$(find
navigation_stage)/stage_config/maps/willow-full-0.05.pgm 0.05" respawn="false"
/>
  <node pkg="stage_ros" type="stageros" name="stageros" args="$(find
navigation_stage)/stage_config/worlds/willow-pr2-5cm.world" respawn="false" >
    <param name="base_watchdog_timeout" value="0.2"/>
  </node>
  <include file="$(find navigation_stage)/move_base_config/amcl_node.xml"/>
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
navigation_stage)/single_robot.rviz" />
</launch>
```

Lanza el Fichero de configuración de move_base (lo vemos más adelante). Observar: todos los ficheros de configuración están en el paquete **navigation_stage**

move_base_amcl_5cm.launch

```
<launch>
  <master auto="start"/>
  <param name="/use_sim_time" value="true"/>
  <include file="$(find navigation_stage)/move_base_config/move_base.xml"/>
  <node name="map_server" pkg="map_server" type="map_server" args="$(find navigation_stage)/stage_config/maps/willow-full-0.05.pgm 0.05" respawn="false" />
  <node pkg="stage_ros" type="stageros" name="stageros" args="$(find navigation_stage)/stage_config/worlds/willow-pr2-5cm.world" respawn="false" >
    <param name="base_watchdog_timeout" value="0.2"/>
  </node>
  <include file="$(find navigation_stage)/move_base_config/amcl_node.xml"/>
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(find navigation_stage)/single_robot.rviz" />
</launch>
```

Lanza map_server para servir un mapa a resolución 0.05 m/pixel

move_base_amcl_5cm.launch

```
<launch>
  <master auto="start"/>
  <param name="/use_sim_time" value="true"/>
  <include file="$(find navigation_stage)/move_base_config/move_base.xml"/>
  <node name="map_server" pkg="map_server" type="map_server" args="$(find
navigation_stage)/stage_config/maps/willow-full-0.05.pgm 0.05" respawn="false"
/>
  <node pkg="stage_ros" type="stageros" name="stageros" args="$(find
navigation_stage)/stage_config/worlds/willow-pr2-5cm.world" respawn="false" >
    <param name="base_watchdog_timeout" value="0.2"/>
  </node>
  <include file="$(find navigation_stage)/move_base_config/amcl_node.xml"/>
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
navigation_stage)/single_robot.rviz" />
</launch>
```

Lanza Stage con un fichero de configuración.

move_base_amcl_5cm.launch

```
<launch>
  <master auto="start"/>
  <param name="/use_sim_time" value="true"/>
  <include file="$(find navigation_stage)/move_base_config/move_base.xml"/>
  <node name="map_server" pkg="map_server" type="map_server" args="$(find
navigation_stage)/stage_config/maps/willow-full-0.05.pgm 0.05" respawn="false"
/>
  <node pkg="stage_ros" type="stageros" name="stageros" args="$(find
navigation_stage)/stage_config/worlds/willow-pr2-5cm.world" respawn="false" >
    <param name="base_watchdog_timeout" value="0.2"/>
  </node>
  <include file="$(find navigation_stage)/move_base_config/amcl_node.xml"/>
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
navigation_stage)/single_robot.rviz" />
</launch>
```

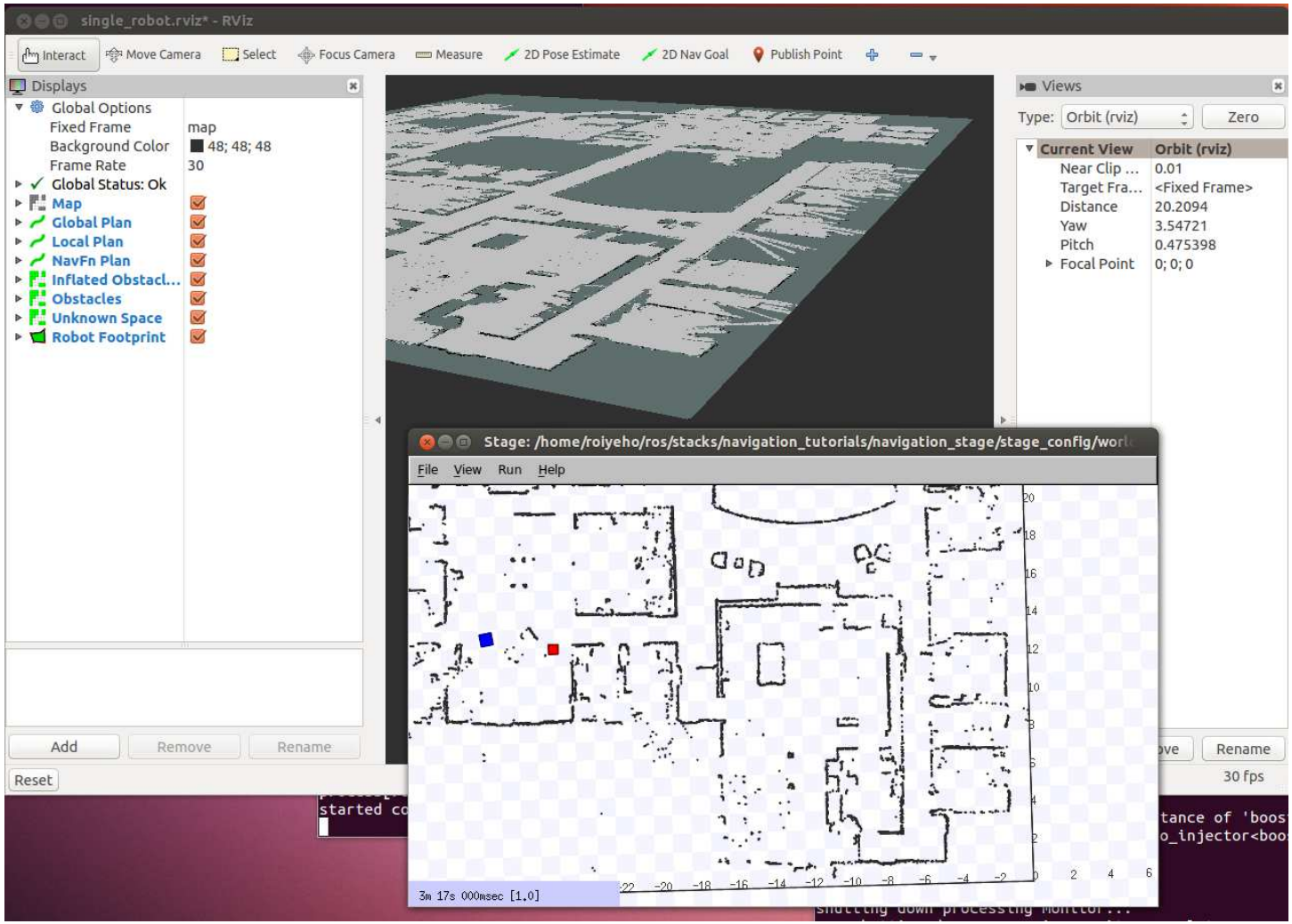
Lanza el nodo de localización monte carlo (lo veremos más adelante)

move_base_amcl_5cm.launch

```
<launch>
  <master auto="start"/>
  <param name="/use_sim_time" value="true"/>
  <include file="$(find navigation_stage)/move_base_config/move_base.xml"/>
  <node name="map_server" pkg="map_server" type="map_server" args="$(find
navigation_stage)/stage_config/maps/willow-full-0.05.pgm 0.05" respawn="false"
/>
  <node pkg="stage_ros" type="stageros" name="stageros" args="$(find
navigation_stage)/stage_config/worlds/willow-pr2-5cm.world" respawn="false" >
    <param name="base_watchdog_timeout" value="0.2"/>
  </node>
  <include file="$(find navigation_stage)/move_base_config/amcl_node.xml"/>
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
navigation_stage)/single_robot.rviz" />
</launch>
```

Lanza rviz con un fichero preconfigurado para navegación.

Ejecutando fichero launch

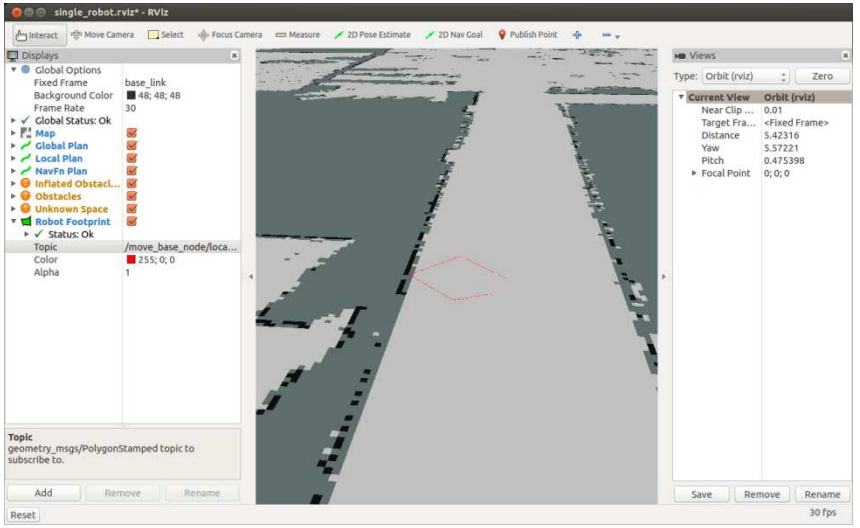


Rviz con *Navigation Stack*

- Las funciones principales de Rviz para navegación:
  1. Visualizar la “footprint” del robot.
  2. Asignar la pose del robot para el sistema de localización.
  3. Visualizar toda la información que suministra ***navigation stack***.
  4. Enviar goals desde rviz.
  5. Observar el movimiento del robot para alcanzar el objetivo propuesto.

Display “Robot Footprint”

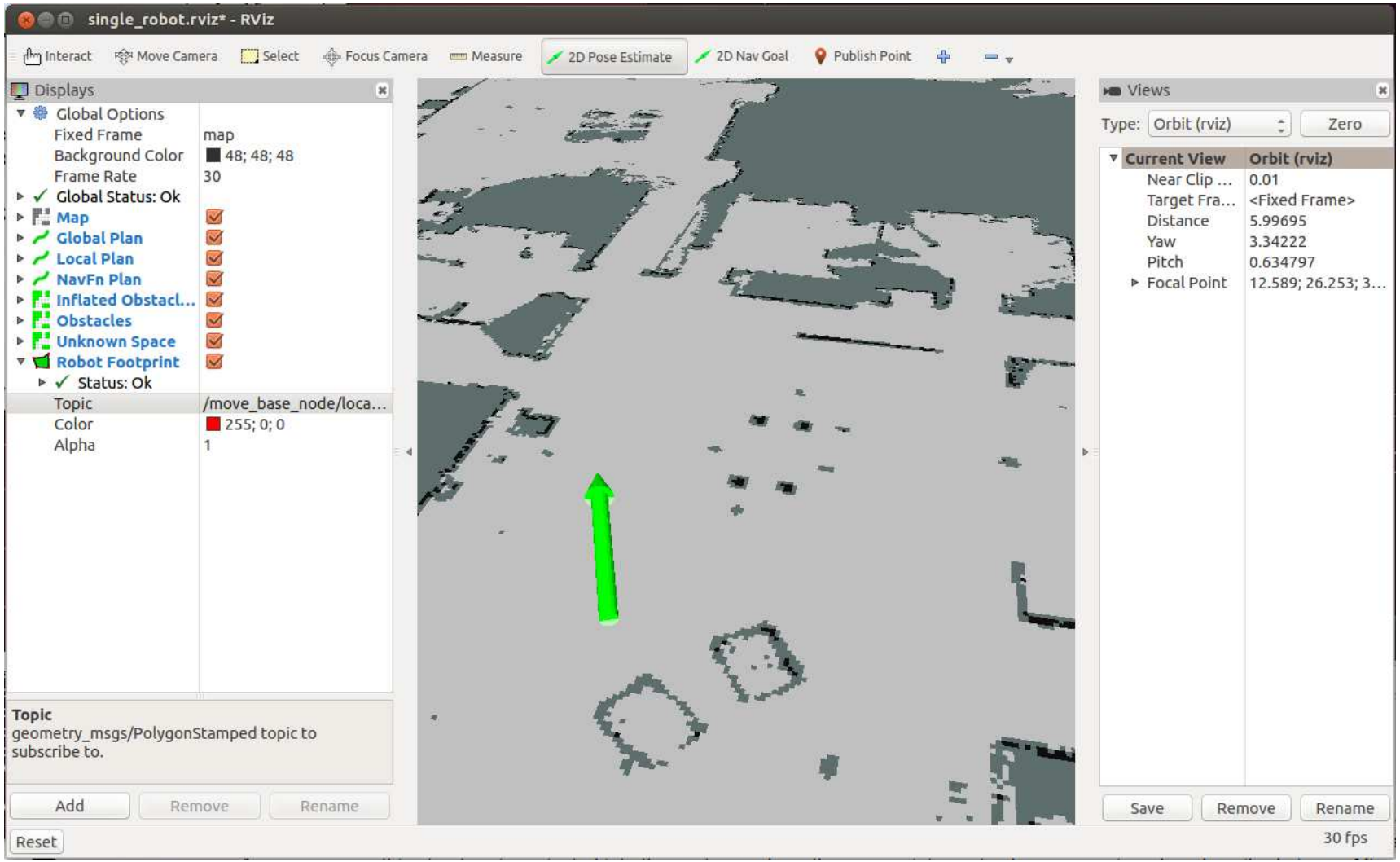
- Muestra la “huella” **(footprint)** del robot
  - En nuestro caso un pentágono
  - Parámetro configurado en el fichero *costmap_common_params.*
- Topic:
  - move_base_node/local_costmap/footprint
- Type:
  - [geometry_msgs/PolygonStamped]



Botón “2D Pose Estimate”

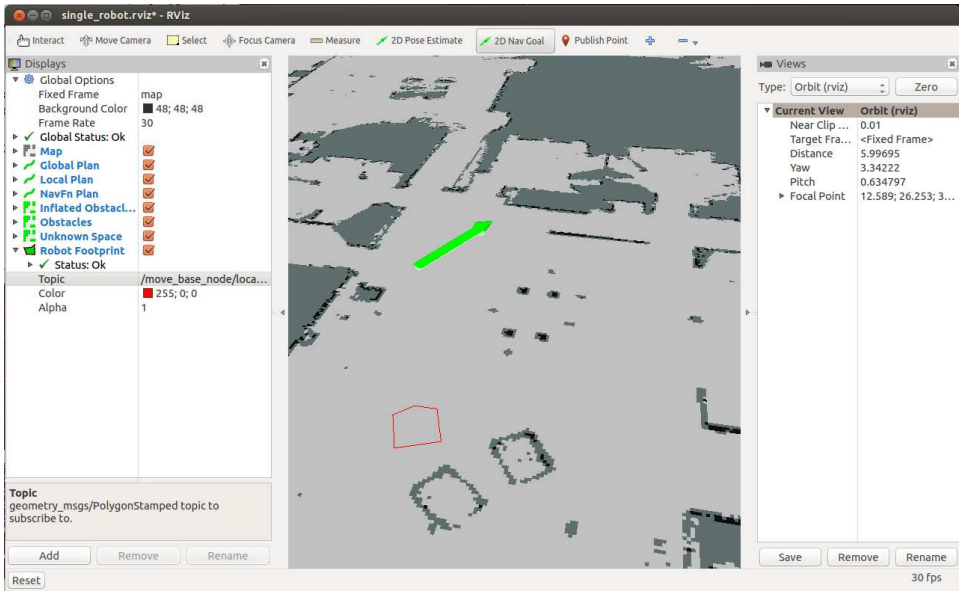
- La estimación de la pose en 2D (P shortcut) nos permite inicializar el sistema de localización usado por *navigation* stack mediante la asignación de la *pose* del robot.
- *Navigation* stack **espera** a que se le asigne esta nueva pose en un *topic* llamado **initialpose**.
- Para asignar la *pose*
  - Click on the **2D Pose Estimate** button
  - Then click on the map **to indicate the initial position** of your robot.
  - If you don't do this at the beginning, the robot will start the auto-localization process and try to set an initial pose.
- ***Note: For the "2d Nav Goal" and "2D Pose Estimate" buttons to work, the Fixed Frame must be set to "map".***

2D Pose Estimate



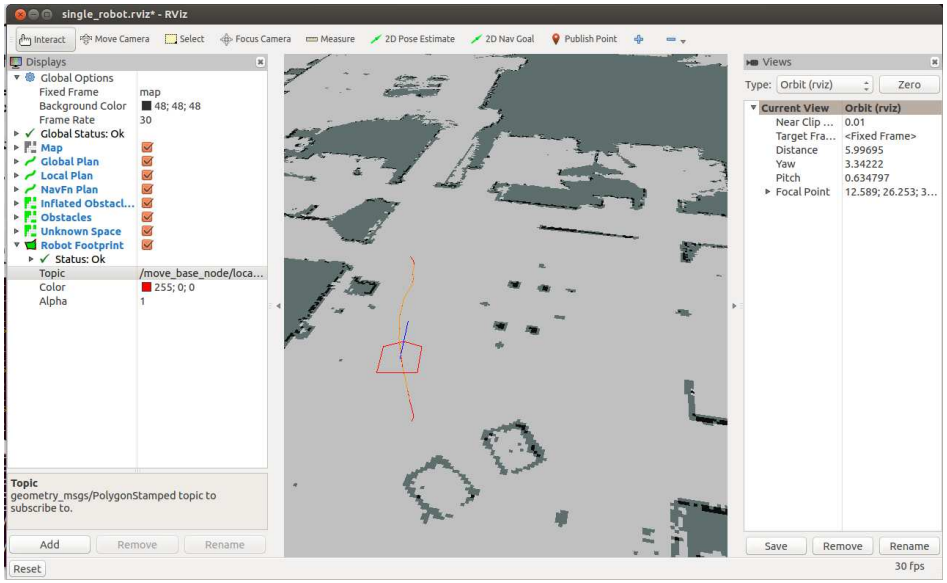
2D Nav Goal

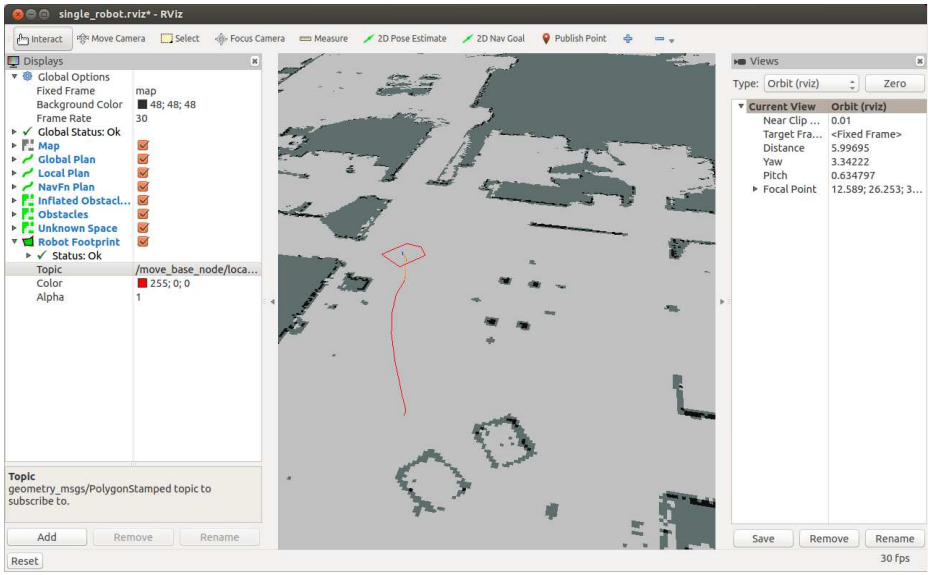
- 2D nav goal (G shortcut) nos permite enviar un goal a *navigation stack.*
- Click on the **2D Nav Goal** button and select the map and the goal for your robot.
- Podemos seleccionar en pantalla la posición (x,y) y la orientación (theta) del robot deseados.



El robot se mueve al destino

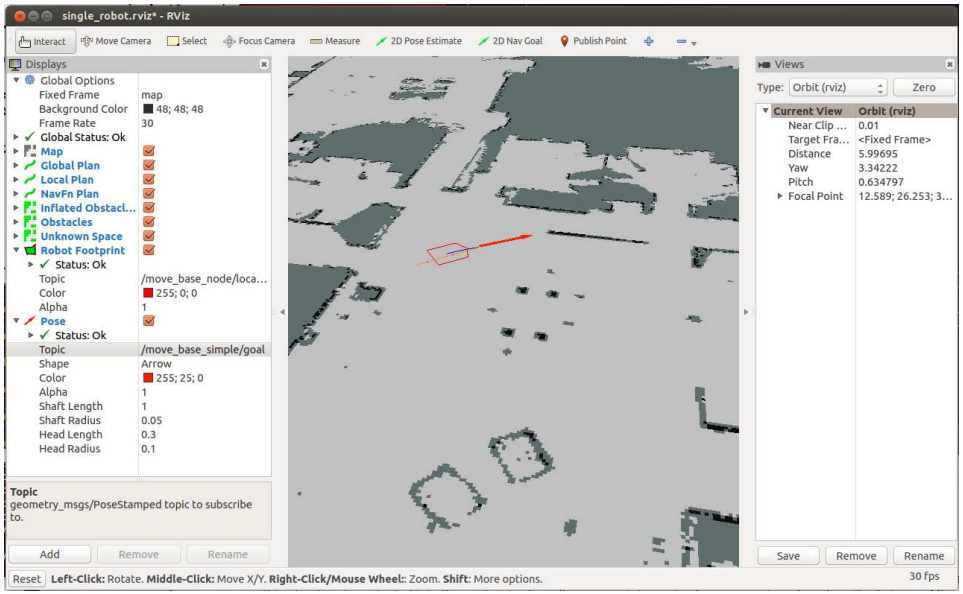
- Observar que el robot se desplaza y se visualiza en Rviz, lo que se ve en Stage es una “simulación del mundo real”.
- Recordar:
  - Rviz es una herramienta de **visualización**
  - Stage es un **simulador**
- En Rviz se visualizan distintas trayectorias de las que hablaremos más adelante.





Current Goal

- Para mostrar el goal actual que *navigation stack* trata de alcanzar añadir un ***Pose Display.***
- Poner su topic a **/move_base_simple/goal**
- El goal se visualiza como una flecha roja.
- Puede usarse para conocer la posición final del robot.



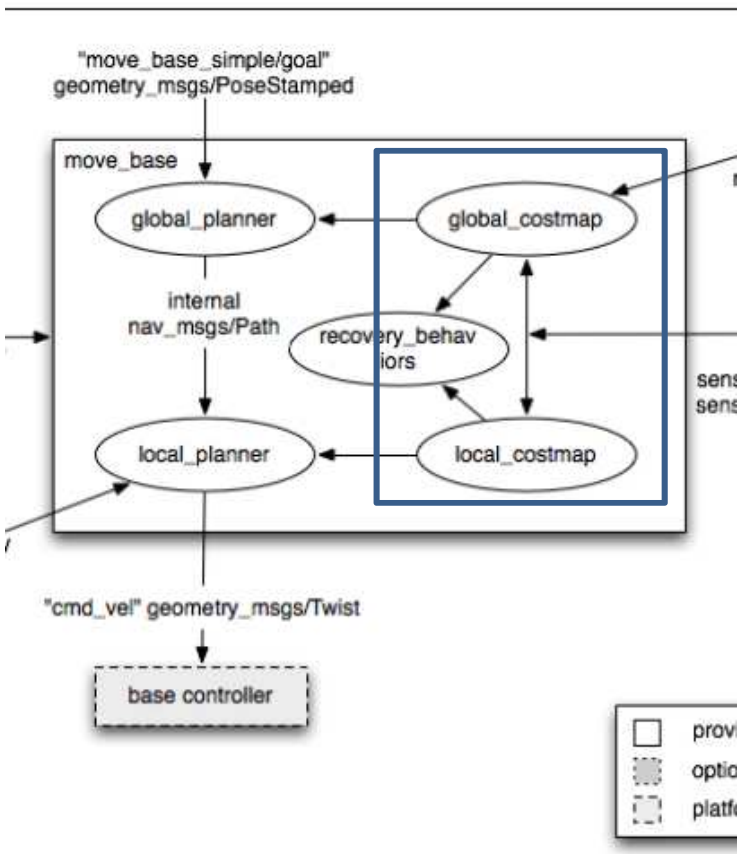
COSTMAPS

Costmaps

- Un ***costmap*** es una estructura de datos que representa lugares seguros en los que el robot puede estar. En ROS es un vector lineal de celdas, resultado de un proceso de discretización.
- Los valores en cada celda del ***costmap*** representan información sobre cómo de alejado se encuentra la celda de un obstáculo, dentro de un rango amplio de valores.
- Distinguir del mapa creado por ***gmapping:*** *en este sólo se representan los valores ocupado/libre/desconocido.*
- Usados en **técnicas de navegación con mapa**
- En ROS la gestión de cotsmaps se hace con **nodos de tipo costmap,** se encargan de actualizar automáticamente la información conforme el robot se mueve.

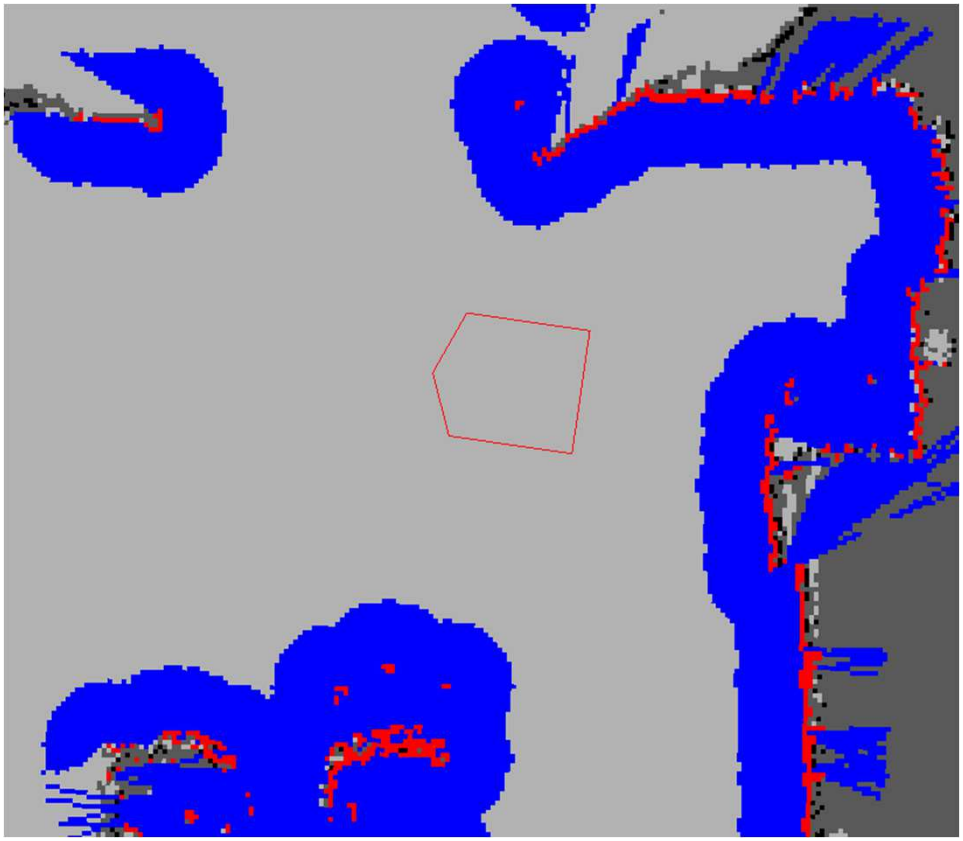
Navigation Stack: Global y Local Costmap

- El robot navega apoyado en una arquitectura que contiene un planificador global y un planificador local
- **Navegación global**: planificar rutas para un *goal* en el mapa o una distancia más lejana.
  - Usa un ***global_costmap: un mapa global del entorno donde actúa el robot para planificar rutas.***
- **Navegación local:** planificar trayectorias en distancias cercanas y *evitar obstáculos*
  - Usa ***local_costmap: un mapa del entorno local del robot para generar trayectorias seguras y no colisionar***



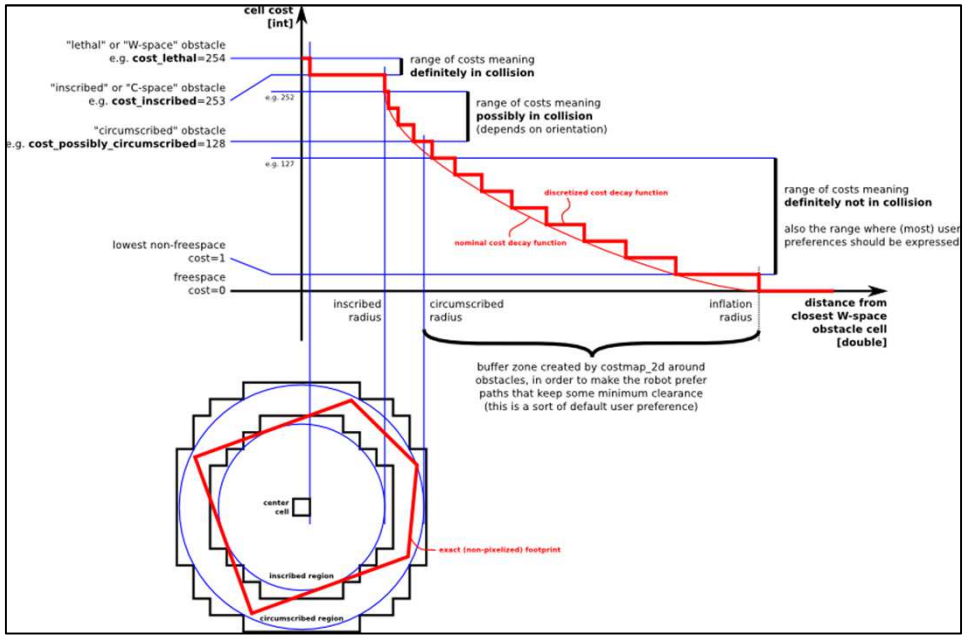
Paquete costmap_2d

- http://wiki.ros.org/costmap_2d
- **costmap_2d** package:
  - usa datos de sensores e información del mapa estático (construido con gmapping, p.ej.)
  - construye una malla de ocupación en 2D con costos basados en la información de ocupación e información del usuario.



API de costmap_2d. Valores en un Costmap

- Each cell in the costmap has a value in the range [0, 255] (integers).
- There are some special values frequently used in this range. (defined in include/costmap_2d/cost_values.h)
  - costmap_2d::**NO_INFORMATION** (255) - Reserved for cells where not enough information is sufficiently known.
  - costmap_2d::**LETHAL_OBSTACLE** (254) - Indicates a collision causing obstacle was sensed in this cell.
  - costmap_2d::**INSCRIBED_INFLATED_OBSTACLE** (253) - Indicates no obstacle, but moving the center of the robot to this location will result in a collision.
  - costmap_2d::**FREE_SPACE** (0) - Cells where there are no obstacles and the moving the center of the robot to this position will not result in a collision.



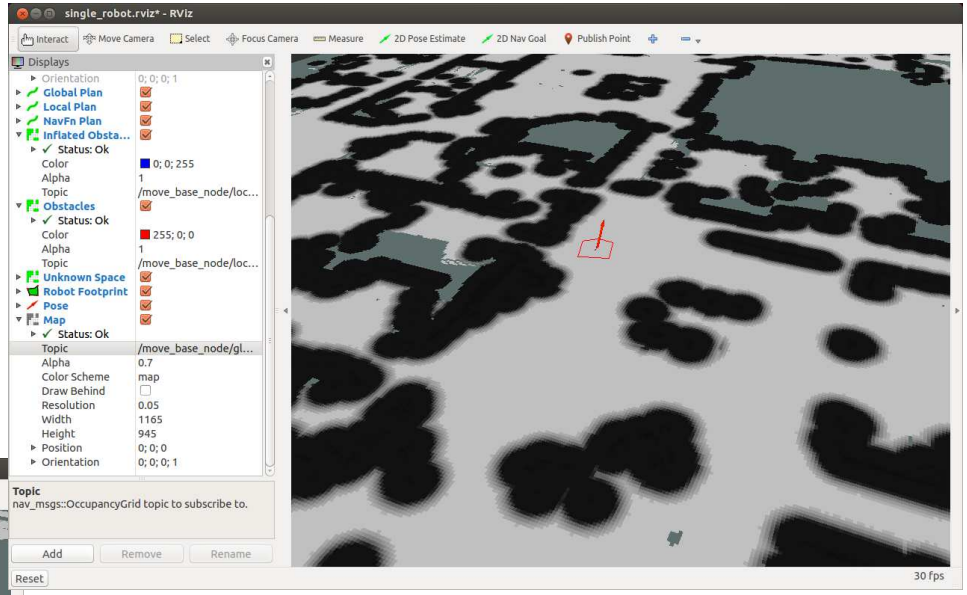
- API de costmap: http://docs.ros.org/indigo/api/costmap_2d/html/

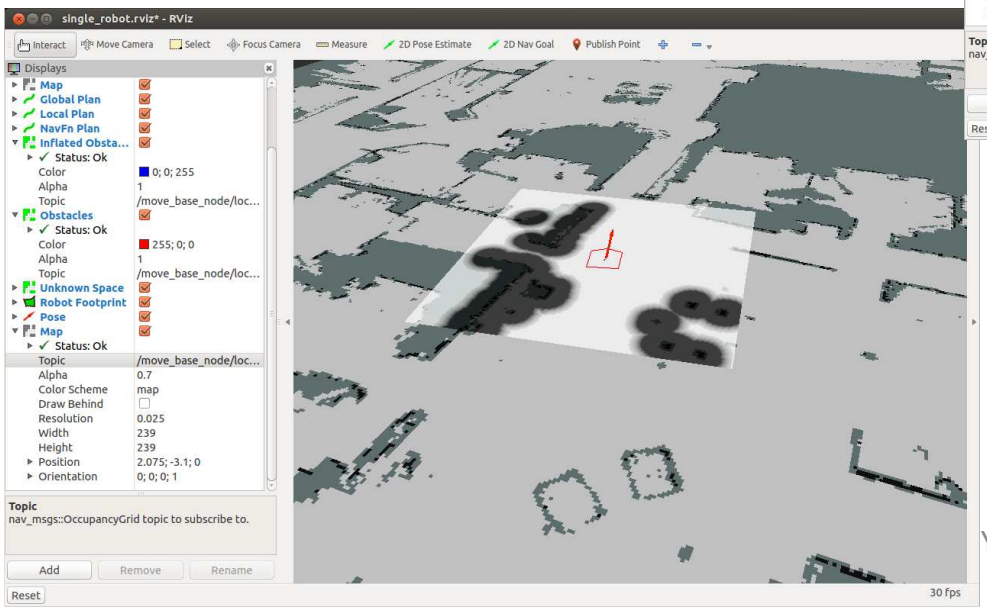
Visualización de costmaps

- Añadir un ***Map Display.***
- Local costmap topic:

  /move_base_node/local_costmap/costmap
- Global costmap topic:

  /move_base_node/global_costmap/costmap





Yehoshua

Ejemplo configuración de Costmaps

- Descargar my_astar_planner desde PRADO
  - Es un paquete que incluye un planificador global basado en búsqueda en Anchura.
  - Incluye un fichero move_base.xml donde se configuran costmaps, global planner y local planner.
- Compilar con catkin_make
- Observar que el directorio launch contiente varios ficheros launch.

Configuración de costmaps

- Observar el fichero move_base_config/move_base.xml que está incluido en el launch anterior.
- Contiene la configuración del costmap global y local haciendo referencia a varios ficheros .yam.

```
<launch>
<!--
  Example move_base configuration. Descriptions of parameters, as well as a full list of all amcl parameters, can be found
at http://www.ros.org/wiki/move_base.
-->
  <node pkg="move_base" type="move_base" respawn="false" name="move_base_node" output="screen">
    <param name="footprint_padding" value="0.01" />
    <param name="controller_frequency" value="10.0" />
    <param name="controller_patience" value="3.0" />

    <param name="oscillation_timeout" value="30.0" />
    <param name="oscillation_distance" value="0.5" />

    <param name ="GlobalPlanner/visualize_potential" value = "true" />
    <param name ="GlobalPlanner/use_dijkstra" value = "false" />

    <!--
    <param name="base_local_planner" value="dwa_local_planner/DWAPlannerROS" />
    -->

    <rosparam file="$(find my_astar_planner)/move_base_config/costmap_common_params.yaml" command="load"
ns="global_costmap" />
    <rosparam file="$(find my_astar_planner)/move_base_config/costmap_common_params.yaml" command="load" ns="local_costmap"
/>
    <rosparam file="$(find my_astar_planner)/move_base_config/local_costmap_params.yaml" command="load" />
    <rosparam file="$(find my_astar_planner)/move_base_config/global_costmap_params.yaml" command="load" />
    <rosparam file="$(find my_astar_planner)/move_base_config/base_local_planner_params.yaml" command="load" />
    <!--
    <rosparam file="$(find my_astar_planner)/move_base_config/dwa_local_planner_params.yaml" command="load" />
    -->
  </node>
```

Configuración de costmaps

- **Ver documento GuiaCostmaps2018.pdf en PRADO para más información.**
- **Costmap_common_params.yaml**: se configuran parámetros comunes a ambos costmaps como, por ejemplo,
  - qué **topics** relativos a fuentes sensoriales (observation_sources) se van a usar (especificando el topic y tipo de mensaje entre otras cosas),
  - qué **valor de coste del costmap usamos como umbral** para, a partir de él, considerar que la celda que contiene ese valor es un obstáculo,
  - qué **radio consideramos para hacer la inflación** (en este ejemplo 55 cm, es decir, los valores de las casillas a menos de 55 cm de un obstáculo tendrán un valor distinto al de FREE_SPACE),
  - qué **dimensiones tiene el footprint del robot** (especificado en este caso como los puntos de un pentágono "con pico"), etc.

Configuración de costmaps

- **Local_costmap_params.yaml**: parámetros exclusivos del costmap local:
  - a qué frecuencia se modifica el costmap (**update frequency**). Este valor es **importante** porque hay que asignarlo considerando la frecuencia a la que se actualiza el escaneo láser y la frecuencia a la que se envían órdenes al robot.
  - **Frecuencia de publicación**: este valor es **fundamental** porque por defecto está a 0 (significa no publicar el costmap) puede dar dolores de cabeza si no se pone a un valor adecuado, por ejemplo 2 o 5 Hz.
  - Configuración del **rolling window** (ventana que avanza junto al robot):
    - rolling_window:**tiene que estar a true**,
    - las **dimensiones de la ventana** (width y height, en metros),
    - el **origen de coordenadas** (dejarlo a origin_x = origin_y = 0)
    - **la resolución** (resolution en metros/celda) indica "**cómo de grandes son las celdas del costmap"**, a mayor resolución, las celdas serán más pequeñas, pero esto afectará al tamaño del costmap y por tanto a la eficiencia de cualquier algoritmo que lo recorra.
- **Global_costmap_params.yaml**: son los mismos parámetros que los del local costmap pero con valores distintos, especialmente
  - **rolling_window tiene que ser false** (el global costmap no se desplza con el robot), por tanto es estático (**static_map = true**) y
  - es importante especificar en **qué topic se publica el mapa** del que se nutre el costmap local que en nuestro caso es "/map".

Global and Local Planner

Global planner: *navfn*

http://wiki.ros.org/navfn

- este paquete proporciona funciones para calcular planes globales (búsqueda en grafos)

- El plan global se calcula antes de que el robot se mueva a la próxima posición.
- Asume un robot circular y opera en un ***costmap global*** para encontrar un camino de coste mínimo en el mapa.

Local planner: *base_local_planner*

http://wiki.ros.org/base_local_planner

- este paquete proporciona funciones para calcular trayectorias como planes locales.

- Monitoriza datos de sensores y selecciona las velocidades angular y lineal apropiadas para que el robot atraviese un segmento del plan global.
- Combina datos de odometría con los mapas de coste global y local para seleccionar el camino que debe seguir el robot.
- Puede recalcular el camino sobre la marcha para mantener al robot lejos de zonas de colisión, garantizando alcanzar el destino.

Global and Local Planner

Global planner: *navfn*

- Hay ficheros de configuración para definir el comportamiento del global planner por defecto (Dijkstra).
- Ver: http://wiki.ros.org/global_planner?distro=indigo
- Es posible implementar global planners diferentes, siguiendo las recomendaciones e interfaces provistas por ROS.

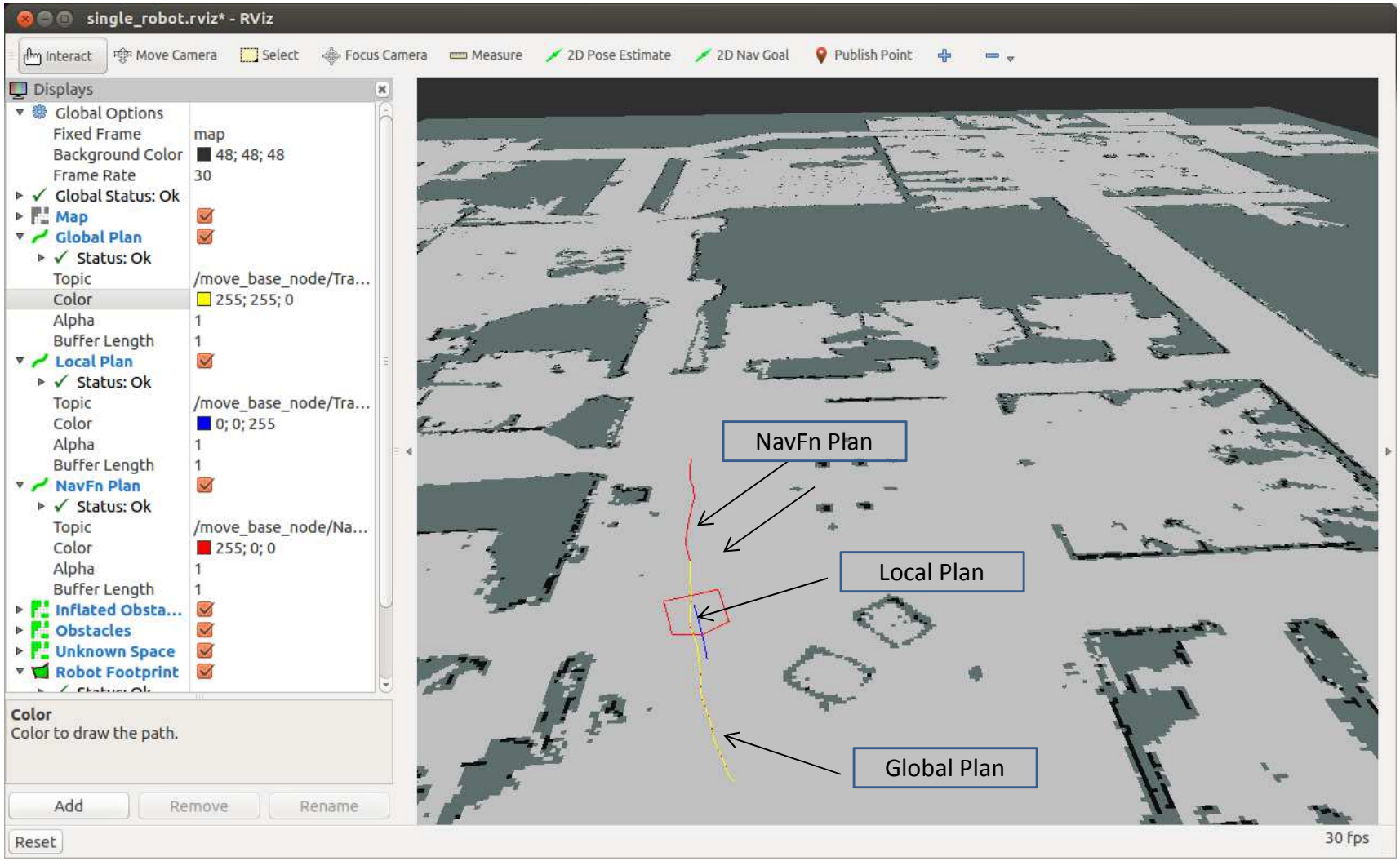
Ver documentación en: http://wiki.ros.org/navigation/Tutorials/Writing%20A%20Global%20Path%20Planner%20As%20Plugin%20in%20ROS

Local Planner: *base_local_planner*

- Implementa dos tipos distintos de técnicas de navegación local:
  - **Trajectory Rollout.** Brian P. Gerkey and Kurt Konolige. "Planning and Control in Unstructured Terrain". Discussion of the Trajectory Rollout algorithm in use on the LAGR robot
  - **Dynamic Window.** D. Fox, W. Burgard, and S. Thrun. "The dynamic window approach to collision avoidance". The Dynamic Window Approach to local control.
- Puede reimplementarse también. Ver base_local_planner::TrajectoryPlannerROS.

- **NavFn Plan**
  - Displays the full plan for the robot computed by the global planner
  - Topic: /move_base_node/NavfnROS/plan
- **Global Plan**
  - It shows the portion of the global plan that the local planner is currently pursuing.
  - Topic: /move_base_node/TrajectoryPlannerROS/global_plan
- **Local Plan**
  - It shows the trajectory associated with the velocity commands currently being commanded to the base by the local planner
  - Topic: /move_base_node/TrajectoryPlannerROS/local_plan

Visualización de planes en rviz



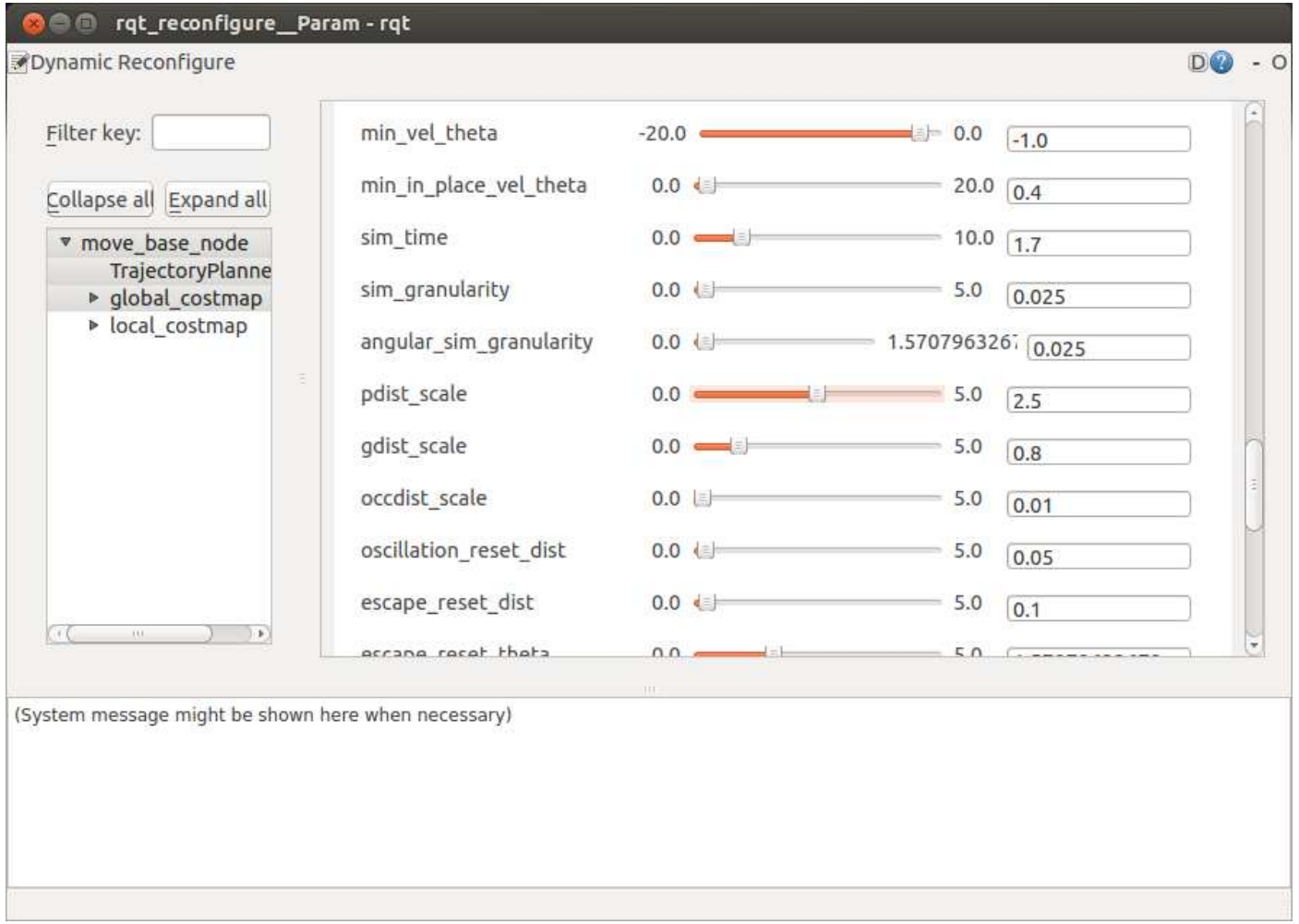
Cambiar configuraciones por defecto dinámicamente

- Run rqt_reconfigure

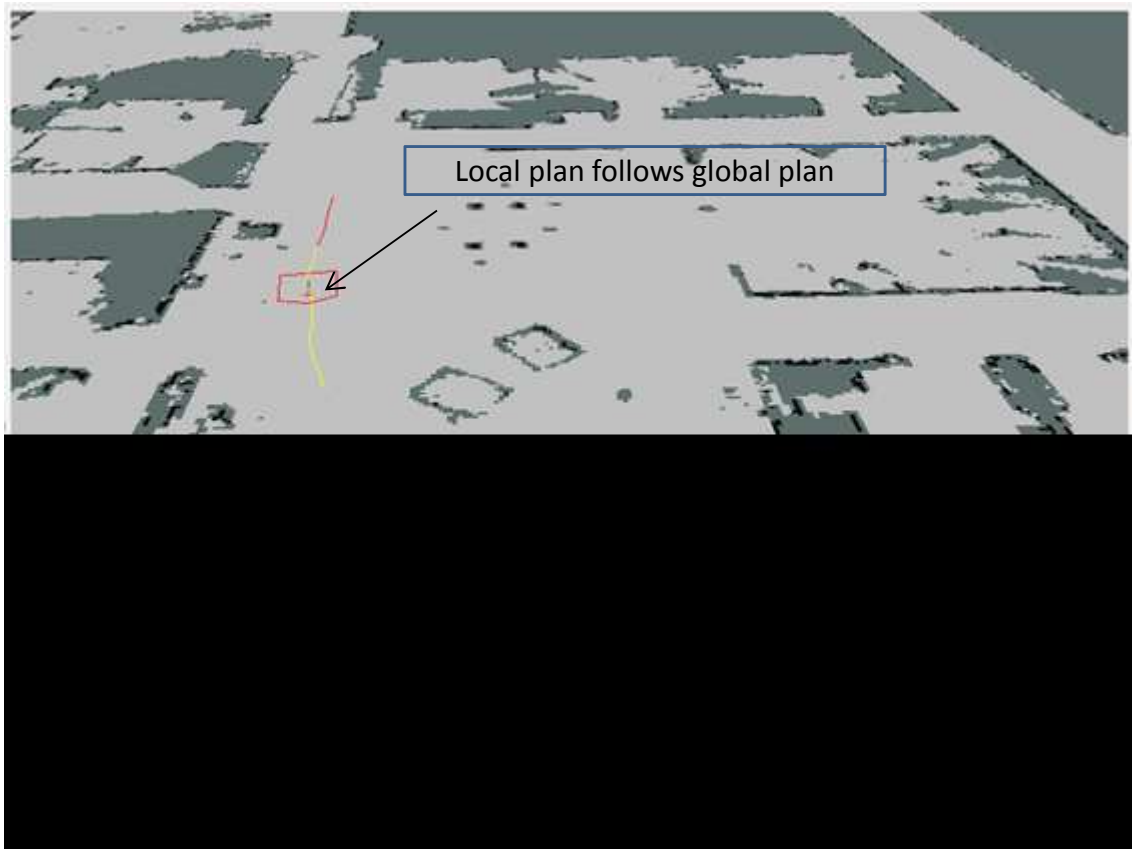
```
$ rosrun rqt_reconfigure rqt_reconfigure
```

  - Permite cambiar valores de configuración dinámicamente, en tiempo real
- Open the move_base group
- Select the TrajectoryPlannerROS node
- Then set the pdist_scale parameter to something high like 2.5
- After that, you should see that the local path (blue) now more closely follows the global path (yellow).

Changing Trajectory Scoring



Changing Trajectory Scoring



Idea de la Práctica 1

- Demostración de una implementación de A* propia.
  - Visualización de Abiertos y Cerrados en rviz.
  - En my_astar_planner de PRADO hay implementada una búsqueda en anchura.
  - Descargar my_astar_planner, compilarlo y ejecutar
  - roslaunch my_astar_planner move_base_amcl_10cm+myAstar.launch
- Se pide implementar un A* mejorando la implementación de anchura.
  - Se pide también mejorar el algoritmo A* para tratar de reducir el tiempo en que tarda en encontrar una solución.
  - Este es un requisito importante porque la implementación está integrada en una arquitectura para la navegación de un robot en tiempo real.
  - Para intentar reducir el tiempo de búsqueda puede usarse cualquier técnica explicada en teoría o alguna de las técnicas descritas en un blog muy referenciado sobre A* para juegos (que apunta técnicas totalmente aplicables a nuestro problema en robótica) .
    http://theory.stanford.edu/~amitp/GameProgramming/

¿Qué tengo que saber?

- Move_Base funciona con un planificador global por defecto pero puede cambiarse por otro planificador global implementado específicamente, por ejemplo A*.
- Para desarrollar un planificador global hay que saber que en ROS se pueden desarrollar plugins para adaptar funcionalidades a nuestras necesidades.
- ¿Cómo configurar move_base para que ejecute mi plugin?
  - Siguiente transparencia

move_base_amcl_10cm+myAstar.launch

```
<launch>
  <master auto="start"/>
  <param name="/use_sim_time" value="true"/>


  <!-- Lanzamos move_base para navegacion, con la configuración de un planificador global específico -->

  <include file="$(find my_astar_planner)/move_base_config/move_base+global_planner.xml"/>

 <!-- Lanzamos map_server con un mapa, a una resolución de 10cm/celda -->

  <node  name="map_server"  pkg="map_server"  type="map_server"  args="$(find  my_astar_planner)/stage_config/maps/willow-
full.pgm 0.1" respawn="false" />

<!-- Lanzamos stage con el mundo correspondiente al mapa -->

  <node  pkg="stage_ros"  type="stageros"  name="stageros"  args="$(find  my_astar_planner)/stage_config/worlds/willow-
pr2.world" respawn="false">
    <param name="base_watchdog_timeout" value="0.2"/>
  </node>

<!-- Lanzamos el nodo amcl -->

  <include file="$(find my_astar_planner)/move_base_config/amcl_node.xml">
     <arg name="initial_pose_x"  value="47.120"/>
     <arg name="initial_pose_y"  value="21.670"/>
     <arg name="initial_pose_a"  value="0.0"/>

  </include>

<!-- Lanzamos rviz -->

 <node name="rviz" pkg="rviz" type="rviz" args="-d $(find my_astar_planner)/rviz_config/single_robot_OpenClosed.rviz" />
</launch>
<launch>
```

- Un plugin se desarrolla como un paquete normal, pero en el directorio raíz del paquete tiene que haber un **fichero xml** que declare que lo que se está desarrollando es un plugin.
- El resultado de la compilación no es un nodo ejecutable, sino una librería, y se almacena en el directorio lib del workspace.
- Vamos a ver primero
  - como se gestiona la ejecución de plugins en move_base,
  - luego vemos como se implementa.

move_base+global_planner.xml

```
<launch>
<!--
  Example move_base configuration. Descriptions of parameters, as well as a full list of all amcl parameters, can be found
at http://www.ros.org/wiki/move_base.
  Ejemplo de configuración de move_base para un planificador global específico.
  Los parámetros que se han añadido al move_base.xml estándar están marcados
-->
  <node pkg="move_base" type="move_base" respawn="false" name="move_base_node" o
    <param name="footprint_padding" value="0.01" />
    <param name="controller_frequency" value="10.0" />
    <param name="controller_patience" value="10.0" />
    <param name="oscillation_timeout" value="30.0" />
    <param name="oscillation_distance" value="0.5" />

    <param name="planner_patience" value="120.0" /> <!-- Define el timeout para en
    <param name="base_global_planner" value="my_astar_planner/MyAstarPlanner"/> <!--Declara el tipo de global planner que
se va a usar -->

    <!--
    <param name="base_local_planner" value="dwa_local_planner/DWAPlannerROS" />
    -->

    <rosparam file="$(find my_astar_planner)/move_base_config/costmap_common_params.yaml" command="load"
ns="global_costmap" />
    <rosparam file="$(find my_astar_planner)/move_base_config/costmap_common_params.yaml" command="load" ns="local_costmap"
/>
    <rosparam file="$(find my_astar_planner)/move_base_config/local_costmap_pa
    <rosparam file="$(find my_astar_planner)/move_base_config/global_costmap_p
    <rosparam file="$(find my_astar_planner)/move_base_config/base_local_plann
    <!--
    <rosparam file="$(find my_astar_planner)/move_base_config/dwa_local_planne
    -->
  </node>
</launch>
```

Configuración planificador local
http://wiki.ros.org/base_local_planner

Configuración planificador global
http://wiki.ros.org/move_base

Configuración costmaps

http://wiki.ros.org/costmap_2d

Declarar un plugin

- ¿Dónde especificar que "**myastar_planner/MyastarPlanner"** es un plugin posible para move_base?
  - En el fichero xml del plugin que he desarrollado previamente.
- Observar que la carpeta del paquete "my_astar_planner" contiene un fichero xml adicional ***astar_planner_plugin.xml*** con el siguiente contenido

```
<library path="lib/libmy_astar_planner">
  <class name="my_astar_planner/MyAstarPlanner" type="myastar_planner::MyastarPlanner"
         base_class_type="nav_core::BaseGlobalPlanner">
    <description>
      El planner que me invento yo.
    </description>
  </class>
</library>
```

  - Observar que el valor del parámetro de move_base.xml que declara el uso del plugin es el argumento "name" del tag "class".
  - Más información en
  - http://wiki.ros.org/navigation/Tutorials/Writing%20A%20Global%20Path%20Planner%20As%20Plugin%20in%20ROS

Código fuente A*

- El directorio include/my_astar_planner contiene el fichero donde se define la clase del planificador global.
  - Es una clase que implementa el A*.
- El directorio src/ contiene el fichero .cpp donde se implementa el A*.
- Para compilar:

```
$cd <workspace>
$catkin_make (debería compilar sin error)
```

- para comprobar que el plugin está exportado con éxito hacer

```
$source devel/setup.sh (comprobar después con rospack find my_astar_planner)
$rospack plugins --attrib=plugin nav_core
(debería salir una lista de ficheros xml y entre ellos)
/home/<tuhome>/<workspace>/src/my_astar_planner/astar_planner_plugin.xml
```

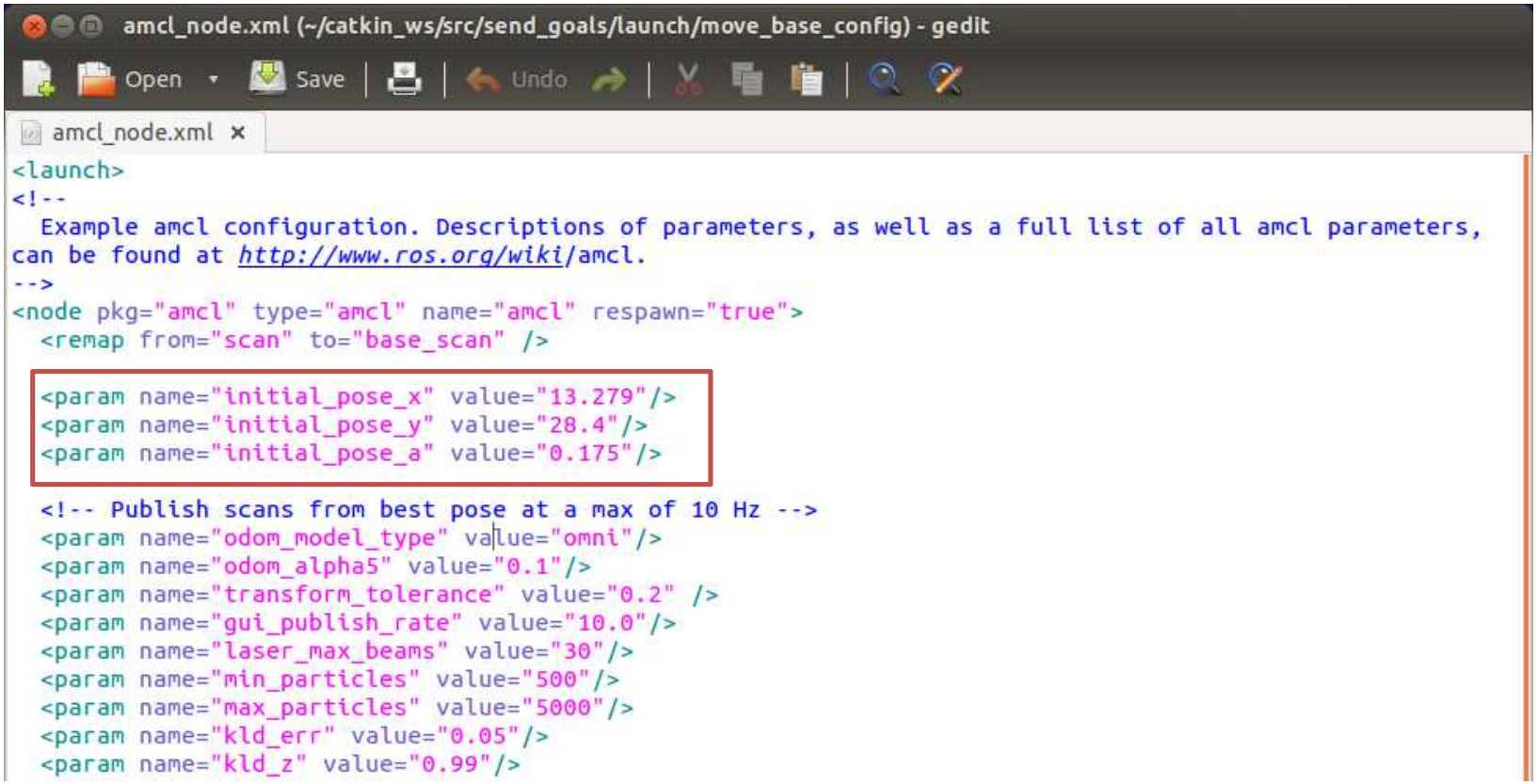
launchs

- En el directorio launch del paquete my_astar_planner hay varios ficheros para lanzar distintas configuraciones:
- move_base_**cm.launch
  - lanza el planificador global por defecto de ROS con mapas configurados a una resolución de 5 y 10 cm.
- move_base_**cm+myAstar.launch
  - lanza el planificador global construido por nosotros.


DECSAI
ejecutar el launch

Configurar amcl.xml para recibir una pose inicial por defecto

- Para evitar estar todo el rato indicándole la pose inicial, podemos reconfigurar amcl para que reciba una pose inicial por defecto.



move_base_amcl_10cm+myAstar.launch

- Para evitar estar todo el rato indicándole la pose inicial, podemos reconfigurar amcl para que reciba una pose inicial por defecto.

```
 <!-- Lanzamos map_server con un mapa, a una resolución de 10cm/celda -->

  <node  name="map_server"  pkg="map_server"  type="map_server"  args="$(find  my_astar_planner)/stage_config/maps/willow-
full.pgm 0.1" respawn="false" />

<!-- Lanzamos stage con el mundo correspondiente al mapa -->

  <node   pkg="stage_ros"   type="stageros"   name="stageros"   args="$(find   my_astar_planner)/stage_config/worlds/willow-
pr2.world" respawn="false">
     <param name="base_watchdog_timeout" value="0.2"/>
  </node>

<!--  Lanzamos el nodo amcl  -->

  <include file="$(find my_astar_planner)/move_base_config/amcl_node.xml">
     <arg name="initial_pose_x"  value="47.120"/>
     <arg name="initial_pose_y"  value="21.670"/>
     <arg name="initial_pose_a"  value="0.0"/>

  </include>

<!-- Lanzamos rviz -->

 <node name="rviz" pkg="rviz" type="rviz" args="-d $(find my_astar_planner)/rviz_config/single_robot_OpenClosed.rviz" />
</launch>
<launch>
```