
# Técnicas de los Sistemas Inteligentes

## Práctica1: Robótica.
## Sesion4. Planificación **de caminos y Navegación Global**

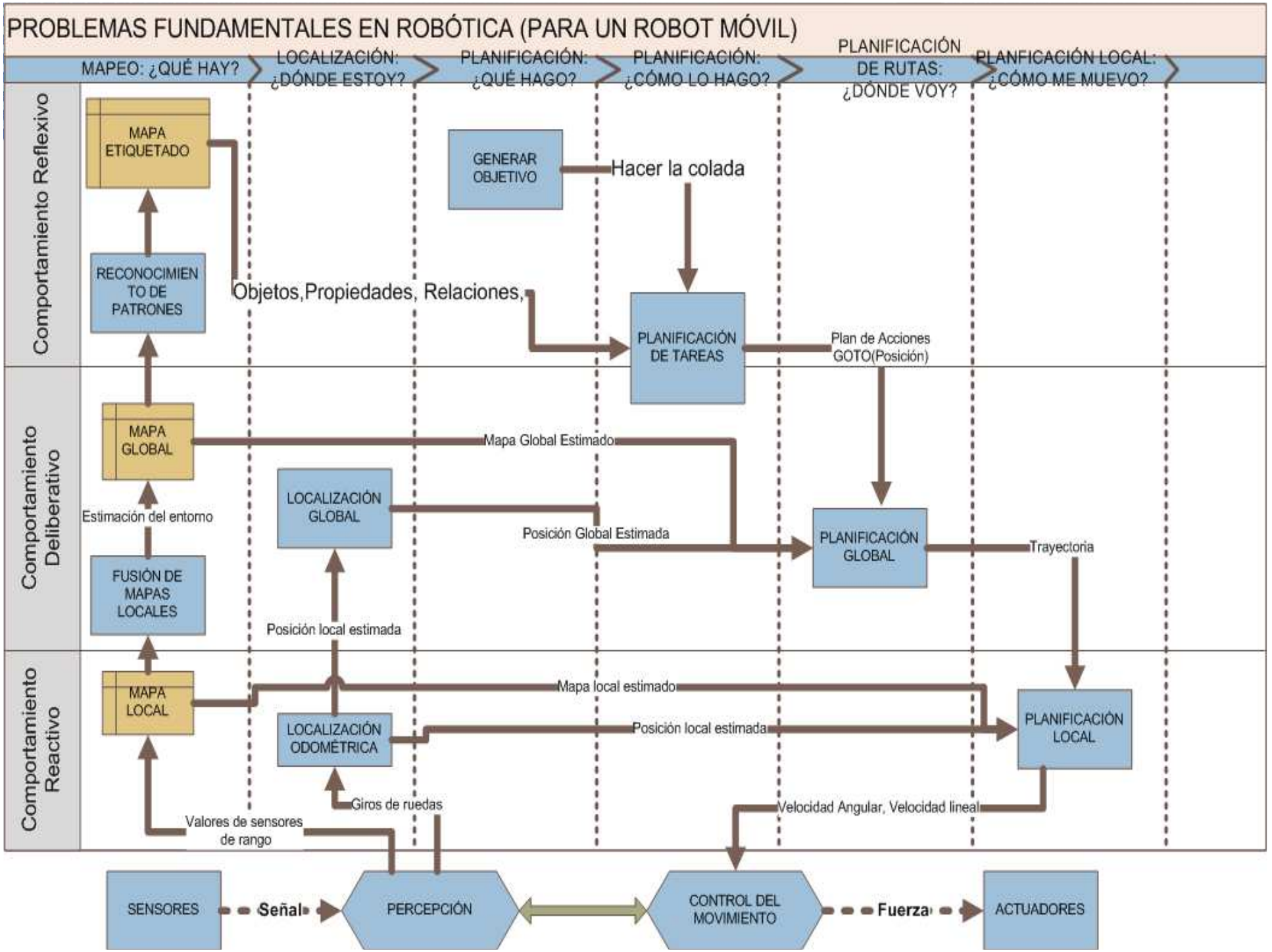
# Agenda

- Conceptos básicos de Navegación Global.
- Tipos de Mapas y Generación de mapas de ocupación
  - Paquetes gmapping y map_server
- Costmaps
- RVIZ: herramienta de visualización de ROS.
- ROS Navigation Stack
  - Un conjunto de paquetes que facilitan la navegación (Global y Local).
- Paquete move_base:
  - Planificador global y local.
  - Interacción con RVIZ para navegar.
- Cómo construir nuestro planificador global y ejecutar planes gobales de navegación.

# Introducción

- Navegación local:
  - Hemos visto comportamientos reactivos (basados en campos de potencial) para planificación punto a punto, o planificación local

- Navegación global:
  - En muchas situaciones la navegación requiere deliberación
    - Por ejemplo, atasco en mínimos locales
    - Recorrer las habitaciones de un edificio
    - Hacer tareas complejas (mover paquetes) + patrullar.


PROBLEMAS FUNDAMENTALES EN ROBÓTICA (PARA UN ROBOT MÓVIL)
MAPEO: ¿QUÉ HAY?
LOCALIZACIÓN: ¿DÓNDE ESTOY?
PLANIFICACIÓN: ¿QUÉ HAGO?
PLANIFICACIÓN: ¿CÓMO LO HAGO?
PLANIFICACIÓN DE RUTAS: ¿DÓNDE VOY?
PLANFICACIÓN LOCAL: ¿CÓMO ME MUEVO?
Comportamiento Reflexivo
Comportamiento Deliberativo
Comportamiento Reactivo
MAPA ETIQUETADO
RECONOCIMIENTO DE PATRONES
GENERAR OBJETIVO
Hacer la colada
Objetos,Propiedades, Relaciones,
PLANIFICACIÓN DE TAREAS
Plan de Acciones GOTO(Posición)
MAPA GLOBAL
Mapa Global Estimado
LOCALIZACIÓN GLOBAL
Estimación del entorno
Posición Global Estimada
PLANIFICACIÓN GLOBAL
Trayectoria
FUSIÓN DE MAPAS LOCALES
Posición local estimada
MAPA LOCAL
Mapa local estimado
LOCALIZACIÓN ODOMÉTRICA
Posición local estimada
PLANIFICACIÓN LOCAL
Giros de ruedas
Valores de sensores de rango
Velocidad Angular, Velocidad lineal
SENSORES
Señal
PERCEPCIÓN
CONTROL DEL MOVIMIENTO
Fuerza
ACTUADORES


# Construyendo un mapa

- Antes de empezar a usar la navegación en ROS con ***navigation stack*** , necesitamos suministrar al robot un mapa del mundo
- Hay varias opciones para crear un mapa inicial:
  - Obtenerlo de una fuente externa
    - Por ejemplo a partir de la planta de un edificio
  - Hacer navegación manual mediante teleoperación
  - Usar un algoritmo de deambulación aleatoria.
  - Algoritmos más sofisticados
    - por ejemplo., *Frontier-Based Exploration*, *Online Coverage*

# Construyendo un mapa: SLAM

- **Simultaneous localization and mapping (SLAM)**
  - una técnica usada por robots para construir un mapa en un entorno desconocido, a la vez que tener un registro (ir descubriendo) su posición.
- SLAM puede verse como el problema "del huevo y la gallina":
  - necesito un mapa correcto para poder calcular mi posición, mientras que para construir un mapa correcto necesito información precisa de la pose.
- Lo veremos con algo más de detalle en la siguiente sesión (si nos da tiempo).

# Construyendo un mapa: gmapping

- http://wiki.ros.org/gmapping
- El paquete *gmapping* package implementa un nodo ROS que provee SLAM basado en láser llamado **slam_gmapping**.
- Entradas: laser scans y odometría
- Salidas: Un mapa de ocupación basado en cuadrículas (**occupancy grid map** OGM)
- Actualiza el mapa a medida que el robot se mueve
  - o cuando (después de algún movimiento) tiene una buena estimación de la localización del robot y cómo es el mapa

# Construyendo un mapa: gmapping

- El mapa se publica en un *topic* llamado **/map**
- Tipo del mensaje : nav_msgs/OccupancyGrid
- La (probabilidad de) ocupación se representa como un entero en el rango [0,100]:

  0 es completamente libre

  100 completamente ocupado

  -1 completamente desconocido

# gmapping Algorithm

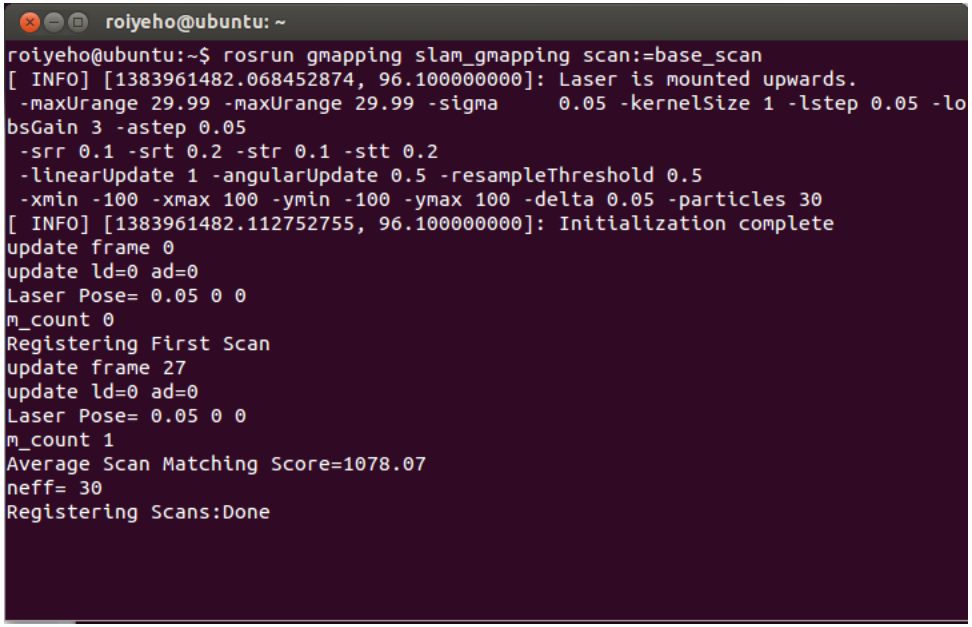
- gmapping implementa FastSLAM 2.0
- Es un algoritmo basado en filtrado de partículas (siguiente sesión).
- Detalles en este artículo:

  http://robots.stanford.edu/papers/Montemerlo03a.pdf

# gmapping

- Arrancar mapping

```
$ roscore
$ rosrun stage_ros stageros `rospack find stage_ros`/world/willow-erratic.world
$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py
$ rosrun gmapping slam_gmapping scan:=base_scan
```



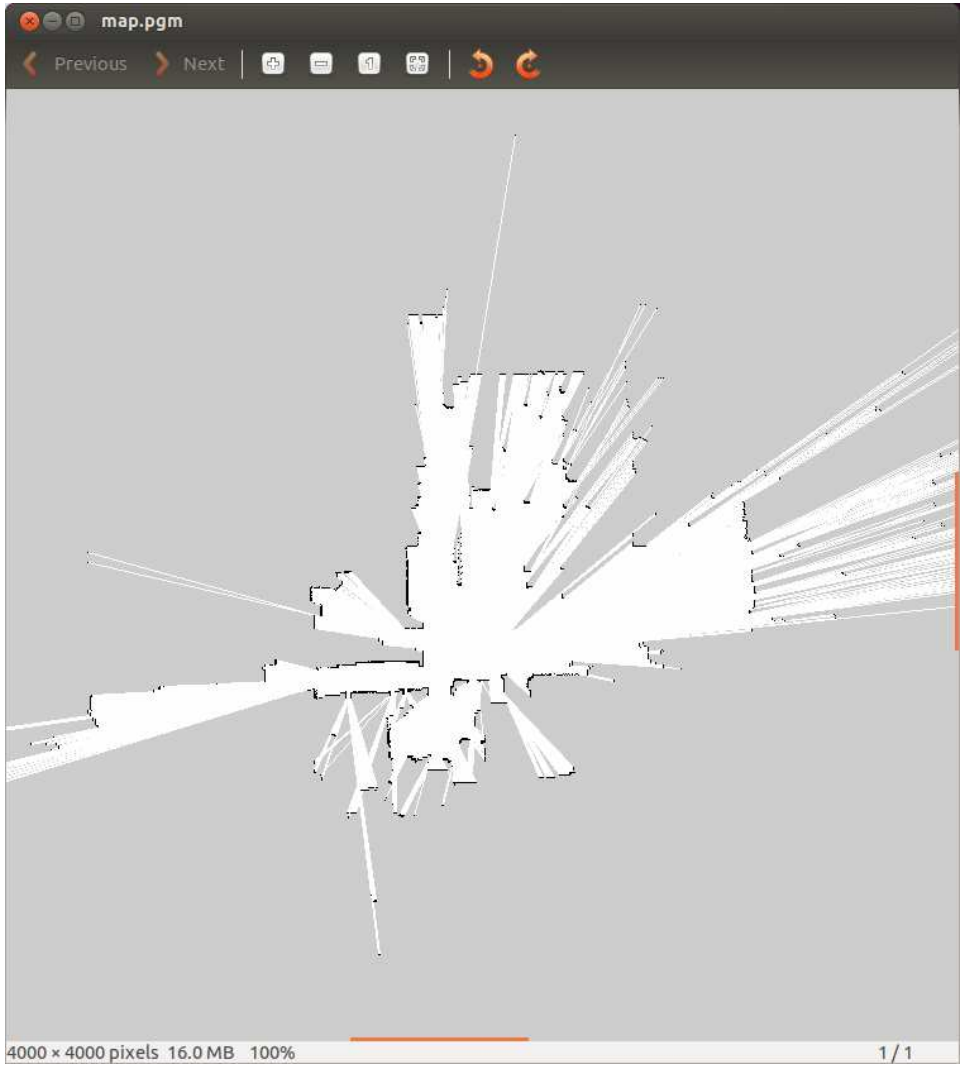
# Guardar el mapa con *map_server*

- ***ROS map_server node*** permite guardar mapas generados en un fichero.
- Ejecutar en una nueva terminal:

```
$ rosrun map_server map_saver [-f mapname]
```

- map_saver recupera los datos del mapa y los guarda en **map.pgm** y **map.yaml** in el directorio actual
  - La opción **-f** sirve para poner una base de nombre diferente para los ficheros de salida.
  - Para ver el mapa usar un visor de imágenes de Ubuntu (por ejemplo, *eog).*

# Guardar el mapa con *map_server*




```
roiyeho@ubuntu: ~
roiyeho@ubuntu:~$ rosrun map_server map_saver
[ INFO] [1383963049.781783222]: Waiting for the map
[ INFO] [1383963050.139135863, 83.100000000]: Received a 4000 X 4000 map @ 0.050
 m/pix
[ INFO] [1383963050.142401554, 83.100000000]: Writing map occupancy data to map.
pgm
[ INFO] [1383963051.553055634, 84.500000000]: Writing map occupancy data to map.
yaml
[ INFO] [1383963051.555821175, 84.500000000]: Done

roiyeho@ubuntu:~$
```

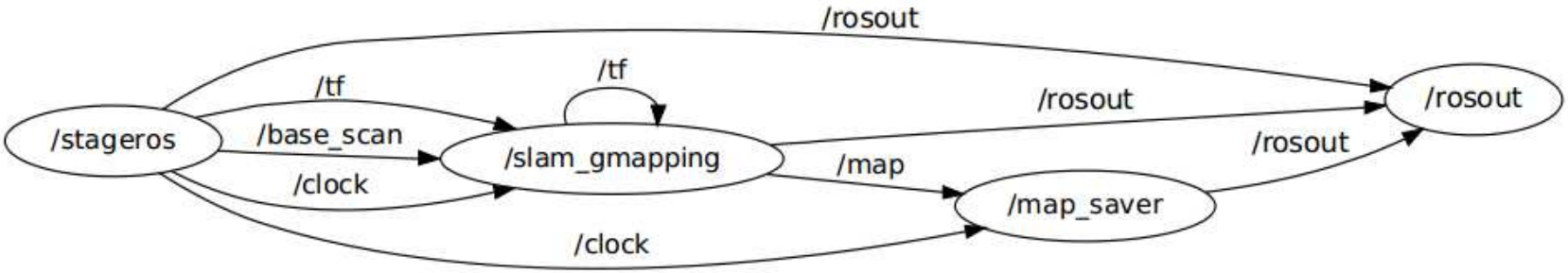
# Formato de la imagen

- La imagen describe el estado de ocupación de cada celda en el mundo en el color del pixel correspondiente.
- Pixels más claros representan espacio libre, más oscuros espacio ocupado, entre ambos colores representan desconocido.
- Los umbrales para dividir las categorías están definidos en un fichero YAML.


```
image: map.pgm
resolution: 0.050000
origin: [-100.000000, -100.000000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

- Important fields:
  - **resolution**: Resolution of the map, meters / pixel
  - **origin**: The 2-D pose of the lower-left pixel in the map as (x, y, yaw)
  - **occupied_thresh**: Pixels with occupancy probability greater than this threshold are considered completely occupied.
  - **free_thresh**: Pixels with occupancy probability less than this threshold are considered completely free.

# Grafo de nodos



# Map_server

- La funcionalidad principal de map_server es la de publicar un mapa como una matriz de ocupación, a partir de un fichero con una imagen.
- Descargar y descomprimir mistage1516
- Abrimos terminal en mistage1516/launch
- Comprobar contenido launch_mapserver.launch
  - Se lanza el servidor de mapas con un nuevo mapa.
  - Se lanza la herramienta rviz, para visualización de entornos con robot.
- Lanzamos launch_mapserver.launch
- Observar que se muestra el nuevo mapa en rviz
  - Es así porque rviz se subscribe al topic /map, que es el topic donde map_server publica el mapa
- Pero no hay robot …

# Map_server y Stage

- Ahora vamos a lanzar stage de manera que en el simulador:
  - Su mundo simulado se haga a partir de la imagen actualmente tratada por map_server y mostrada por rviz.
- Lanzar el fichero launch_mapserver_stage.launch
  - Ahora sí hay robot …. Pero no se ve aun en rviz.
- Observar el fichero launch_mapserver_stage.launch
- Contiene como argumento el fichero mi-simplerooms.world que está en /mistage156/configuracion/mundos
- Es el fichero de configuración del mundo de Stage.
- Ver el fichero .world. En la documentación hay un tutorial sobre cómo hacer ficheros world en Stage.

# Map_server y Stage

- Basta con configurar el objeto “floorplan”
  - Name nombre del mundo
  - Bitmap: localización del fichero
  - Size: tamaño [x y z] en metros
  - Pose del mapa respecto a la ventana (objeto “window”)
- El objeto robot  pr2( pose [x y z theta] name “string” color “blue”)
- El objeto window
  - Size: tamaño de la ventana de Stage
  - scale: ¿cuántos metros corresponden a cada pixel?
    - Valor óptimo=round(W/M)
    - W:tamaño ventana
    - M:tamaño mapa
  - Descripción detallada en:
- http://playerstage.sourceforge.net/doc/Stage-3.2.1/group__worldgui.html

# Map_server y Stage

```
define block model
(
  #definimos un block como un cubo de 50x50x75 cm cubicos.
  size [0.500 0.500 0.750]
  gui_nose 0
)

define topurg ranger
(
  #caracteristicas              del              modelo              sensor
      http://rtv.github.io/Stage/group__model__ranger.html
  sensor(
    range_max 30.0
    fov 270.25
    samples 1081
  )
  # generic model properties
  color "black"
  size [ 0.050 0.050 0.100 ]
)

define pr2 position
(
  size [0.650 0.650 0.250]
  origin [-0.050 0.000 0.000 0.000]
  gui_nose 1
  drive "omni"
  topurg(pose [ 0.275 0.000 0.000 0.000 ])
)

define floorplan model
(
  # sombre, sensible, artistic
  color "gray30"

  # most maps will need a bounding box
  boundary 1

  gui_nose 0
  gui_grid 0

  gui_outline 0
  gripper_return 0
  fiducial_return 0
  ranger_return 1.000
)
```

```
# set the resolution of the underlying raytrace model
in meters
resolution 0.02

interval_sim 100  # simulation timestep in milliseconds


window
(
  size [ 608 300 ]

  rotate [ 0.000 0.000 ]
  scale 12.100
)

# load an environment bitmap
floorplan
(
  name "simple_rooms"
  bitmap "../maps/simple_rooms.png"
  size [40.000 30.000 1.000]
  pose [ 0.000 0.000 0.000 0.000 ]
)

# throw in a robot
pr2( pose [ 8.557 6.313 0.000 -127.719 ] name "pr2"
color "blue")
block( pose [ 16.649 5.740 0.000 0.000 ] color "red")
block( pose [ -25.062 12.909 0.000 180.000 ] color
"red")
block( pose [ -25.062 12.909 0.000 180.000 ] color
"red")
```

# Experimentación con mapas

- Tener en cuenta esta información para generar distintos mapas con los que experimentar.

# costmaps

- Vamos a conocer y visualizar un nuevo tipo de objeto fundamental para tareas de la entrega 2 y entrega 3.
- En PRADO tenéis una guía para manejar costmaps y ejemplos de cómo implementar costmaps para poder implementar un proceso de búsqueda local en el planificador de trayectorias, mejorando los campos de potencial.

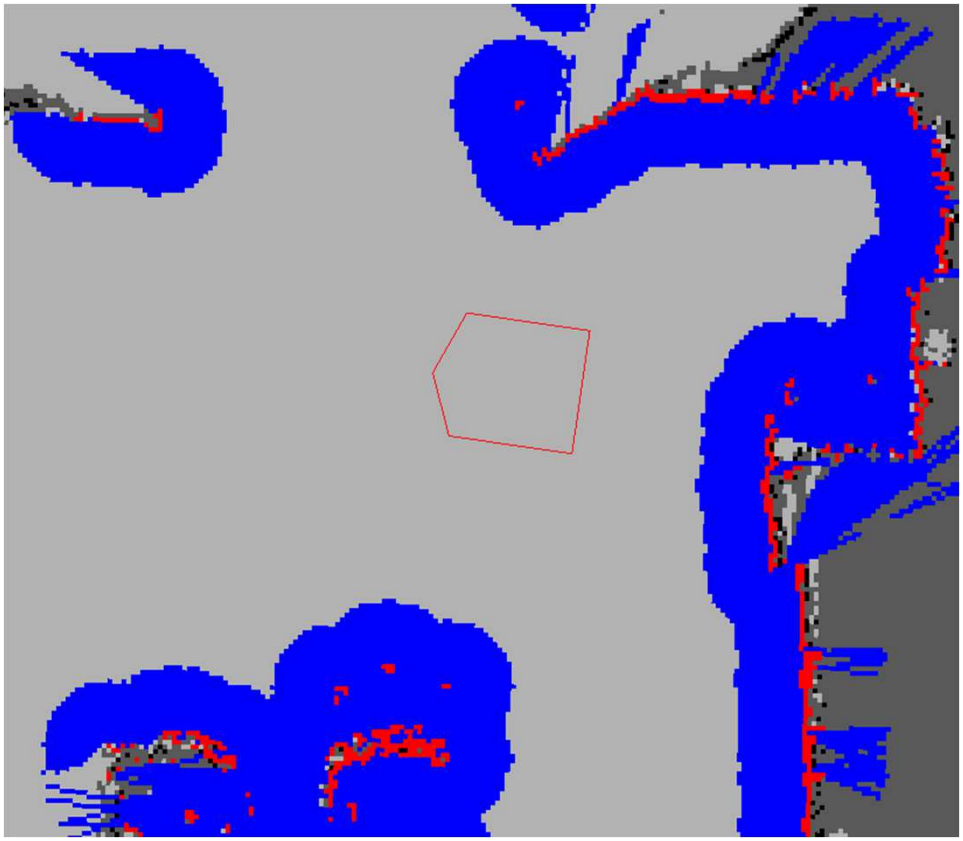
# Costmap

- Un ***costmap*** es una estructura de datos que representa lugares seguros en los que el robot puede estar, en una cuadrícula de celdas.
- Los valores en el ***costmap*** representan o espacio libre o lugares donde puede colisionar, dentro de un rango amplio de valores.
- No confundir con el mapa creado por ***gmapping:*** *en este sólo se representa ocupado/libre/desconocido.*
- Usados en **técnicas de navegación con mapa** (campos de potencial es sin 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.

- El robot se mueve usando navegación global o local.
- **Navegación global**: crear rutas para un *goal* en el mapa o una distancia más lejana:
  - Uso de ***global_costmap***
- **Navegación local:** crear rutas en distancias cercanas y *evitar obstáculos*
  - Uso de ***local_costmap.***

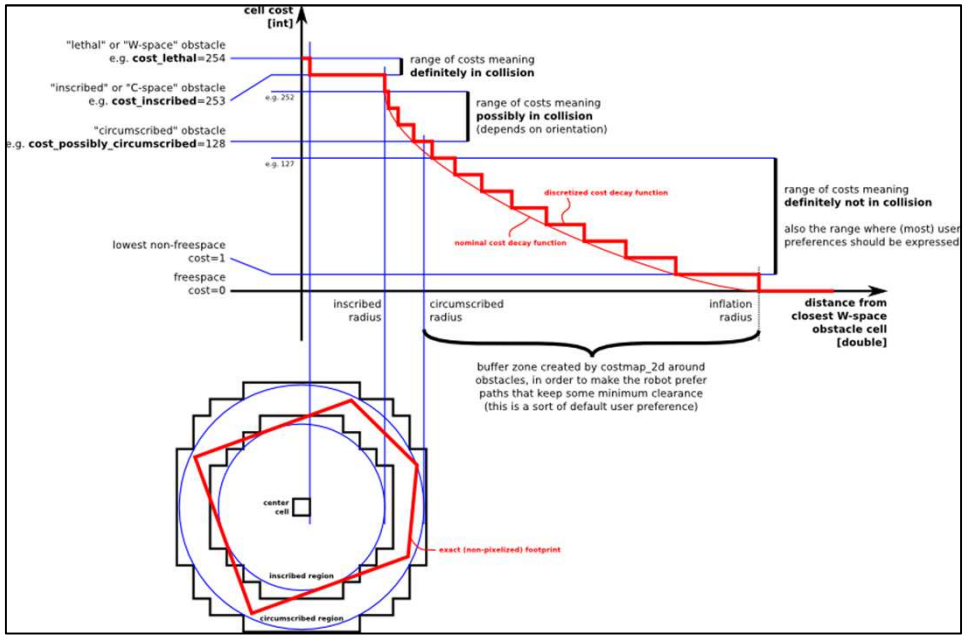
# costmap_2d Package

- 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.



# Costmap Values

- 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/

# costmaps

- Vamos a crear dos costmaps, uno local y otro global y visualizarlos.
- Descargar el fichero miscostmaps.zip en vuestro espacio de trabajo.
- Catkin_make

- El fichero /src/miscostmaps_node.cc p
- Crea dos costmaps.

```
#include "ros/ros.h"
#include "tf/transform_listener.h"
#include "costmap_2d/costmap_2d_ros.h"

int main(int argc, char** argv){
  ros::init(argc, argv, "miscostmaps_node");
  ros::NodeHandle nh;
  tf::TransformListener tf(ros::Duration(10));
  costmap_2d::Costmap2DROS localcostmap("local_costmap", tf);
  costmap_2d::Costmap2DROS globalcostmap("global_costmap", tf);
  ros::spin();
return(0);
}
```


- Lanzamos el fichero miscostmaps_fake_5cm.launch.
- En la guía de costmaps de PRADO está más detallado el uso de ficheros de configuración de costmaps.
- Hay ejemplos de cómo usar costmaps en el cliente.

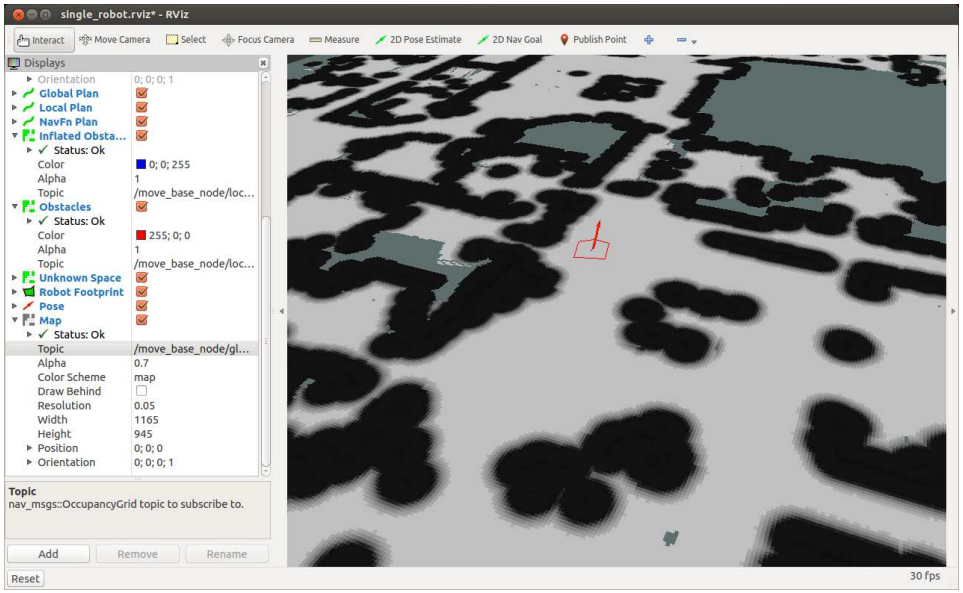
```
<launch>
  <master auto="start"/>
  <param name="/use_sim_time" value="true"/>

   <node name="miscostmaps_node" pkg="miscostmaps" type="miscostmaps_node" output="screen">
      <rosparam file="$(find miscostmaps)/configuration/costmap_common_params.yaml" command="load"
    ns="global_costmap" />
        <rosparam file="$(find miscostmaps)/configuration/costmap_common_params.yaml" command="load"
    ns="local_costmap" />
        <rosparam file="$(find miscostmaps)/configuration/local_costmap_params.yaml"  command="load" />
        <rosparam file="$(find miscostmaps)/configuration/global_costmap_params.yaml" command="load" />
      </node>
   <node name="map_server" pkg="map_server" type="map_server" args="$(find miscostmaps)/maps/simple_rooms.yaml"
    respawn="false"/>


  <node pkg="stage_ros" type="stageros" name="stageros" args="$(find mistage1516)/configuracion/mundos/mi-
    simplerooms.world" respawn="false">
            <param name="base_watchdog_timeout" value="0.2"/>
      </node>
 <node name="fake_localization" pkg="fake_localization" type="fake_localization" respawn="false" />
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(find navigation_stage)/single_robot.rviz" />
</launch>
```

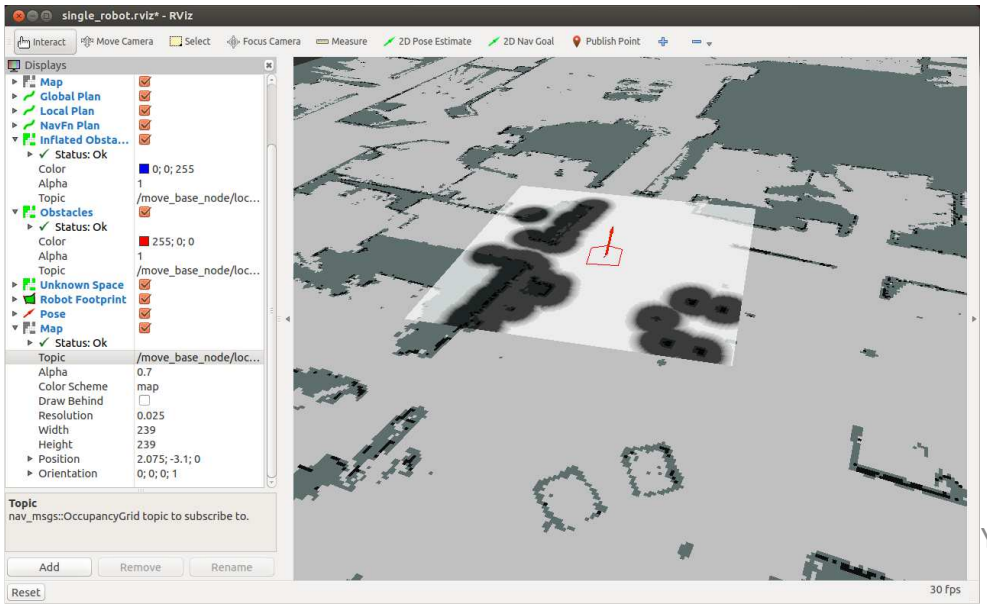
# Visualización de costmaps

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

  /miscostmaps_node/local_costmap/costmap
- Global costmap topic:

  /miscostmaps_node/global_costmap/costmap



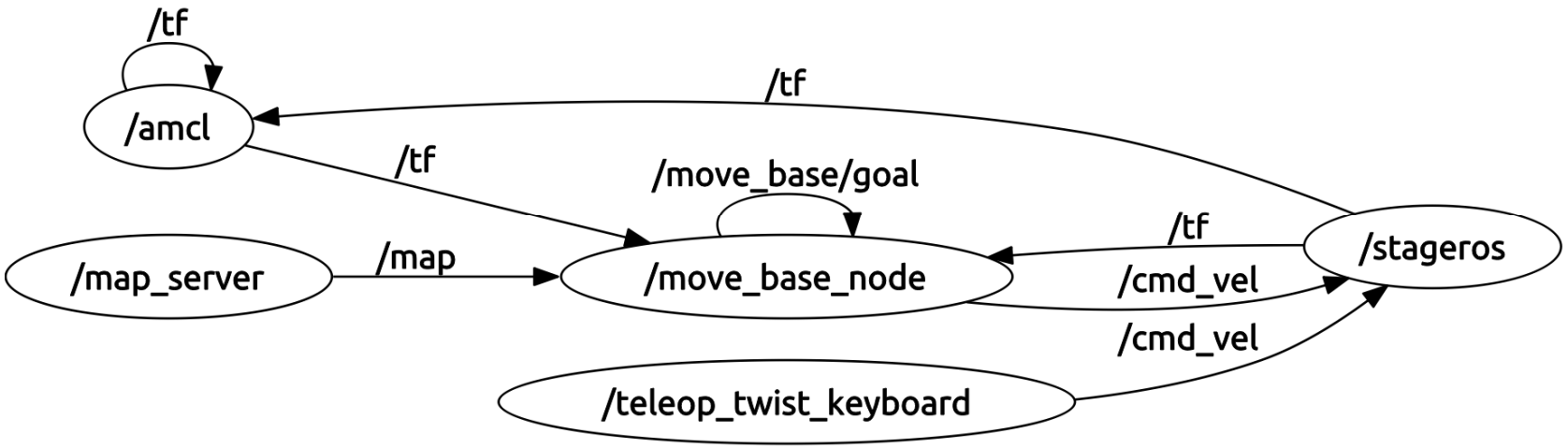




- Configurar el footprint del robot.
- Poner como topic:
- …/local_costmap/footprint
- Ejecutar

  rosrun rqt_reconfigure rqt_reconfigure

Yehoshua

# Nodes Graph



# Las 4 cuestiones de la navegación

- ¿A dónde voy?
  - Objetivo: determinado por un humano o un planificador de misiones
- ¿Cuál es el mejor camino?
  - Problema de planificación de caminos (path planning), el que recibe más atención
    - Métodos cualitativos
    - Métodos cuantitativos
- ¿Por dónde he pasado?
  - Creación de mapas.
- ¿Dónde estoy?
  - Localización y creación de mapas.

# Introducción

- En particular un robot necesita planificar un camino que le especifique cómo alcanzar una localización particular, mediante la consecución de varios hitos intermedios.
- Hay distintas técnicas para Navegación Global:
  - Navegación topológica o cualitativa
  - Navegación métrica o cuantitativa
  - En cualquier caso, para la navegación global es necesario tener un mapa del entorno
    - Siempre aparece la misma pregunta: ¿Cómo construyo mi mapa, o de dónde puedo obtenerlo?

# Navegación topológica o cualitativa

- Basada en encontrar relaciones de conectividad entre objetos del mundo
- Representación del conocimiento
  - (habitación-tiene habitacion1 puerta1) (conecta puerta1 pasillo1) ….
- Paso previo a la planificación automática de tareas.
  - **Práctica 3**
- Distinguir:
  - Planificación de tareas (STRIPS)
    - Muchos tipos de acciones: mover, coger, soltar, apilar,….
    - Representación sofisticada de distintos efectos y precondiciones
  - Planificación de caminos
    - Las acciones planificadas son sólo movimientos del tipo:
      - (move-to x, y, theta)
    - Efectos de acciones: el robot cambia de posición

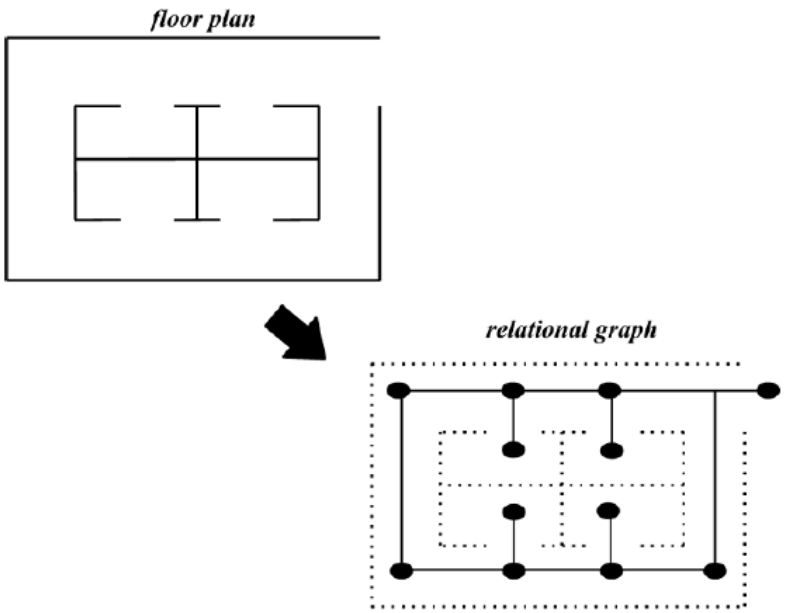
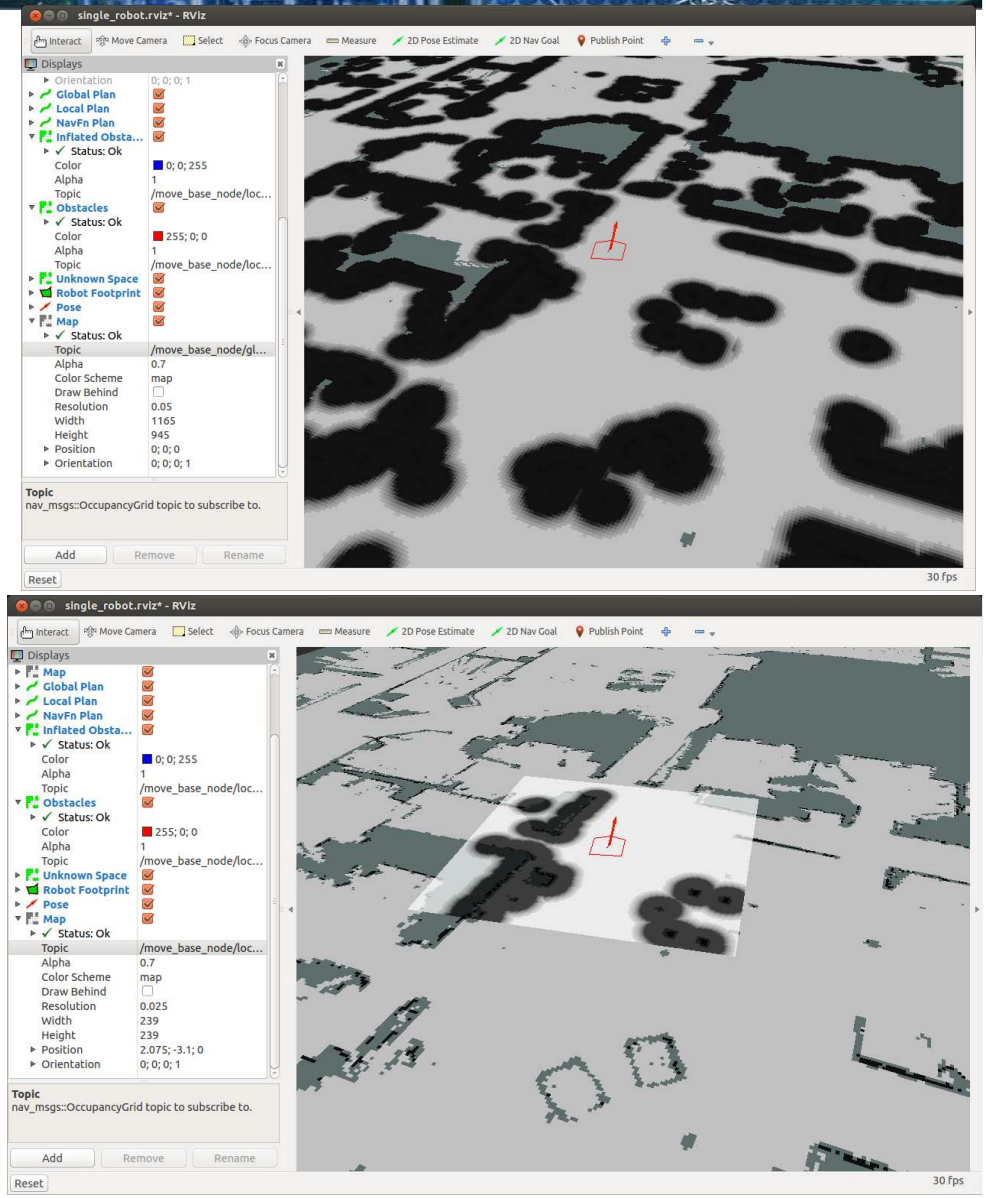



Figure 9.2 Representation of a floor plan as a relational graph.

# Navegación métrica o cuantitativa

- A partir de una representación del mundo como un "mapa“ discretizado de posibles posiciones (espacio de configuraciones) del robot
- Tratar de encontrar un camino, si es posible óptimo.
- Ejecutar el plan con el navegador local.



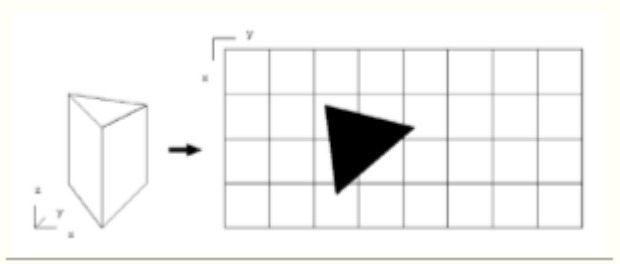
# ¿Cómo construyo mi mapa, o de dónde puedo obtenerlo?

- No conozco el mapa: necesito técnicas de localización (siguiente sesión)
- Sí lo conozco: necesito técnicas de "discretización", porque trabajamos con dominios contínuos.

# REPRESENTACIÓN

# Representación: Espacio de configuraciones (CSPace)

- Espacio del mundo
  - robots, objetos, obstáculos...
- Cspace (configuration space)
  - Una estructura de datos que permite especificar la posición (pose) de cualquier objeto y el robot (por ejemplo: array bidimensional).
  - Una buena representación del Cspace reduce el número de grados de libertad con los que se tiene que enfrentar un planificador
    - Por ejemplo, aunque el robot pueda trabajar con 6 grados de libertad, un robot móvil terrestre sólo necesita 3 (x,y,yaw)



# Representación: Espacio de configuraciones (CSPace) (2)

- Representaciones de un Cspace
  - Varios tipos de representaciones, todas orientadas a crear particionamiento (discretización) del espacio
    - Espacio libre (configuraciones en las que puede estar el robot)
    - Espacio ocupado (configuraciones inalcanzables - obstáculos, muros)
  - Métodos de descomposición en celdas
  - Métodos de esqueletización

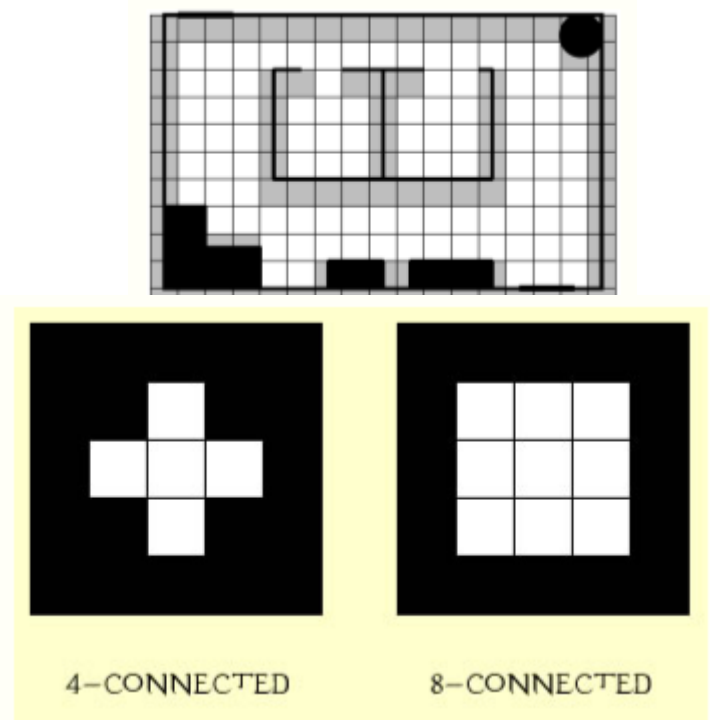
# Descomposición en celdas

- Discretizar el espacio libre en un número finito de regiones contiguas, llamadas celdas.
- El problema de búsqueda de un camino se convierte fácilmente en un problema de búsqueda en grafos (de los que ya conocemos)
- En general recibe el nombre de Matriz de Ocupación (Occupancy Grid)

# Cuadrículas regulares

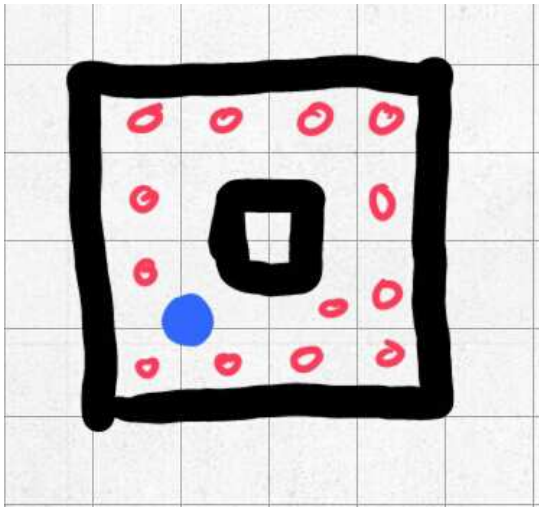
- Superponer una cuadrícula cartesiana sobre el espacio del mundo (lo que hace un costmap)

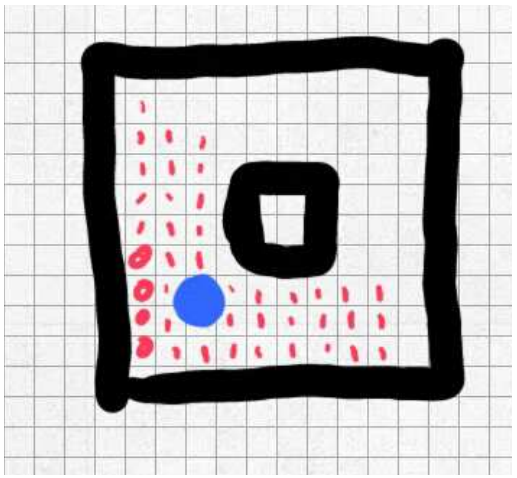
- Referidas como "matrices o mallas de ocupación" (occupancy grid)
  - Casilla = ocupada (0), si hay objeto en el área contenida por la casilla
  - Casilla = libre (1), si no hay objeto.
  - Casilla = desconocida (-1), si se desconoce
    - En problemas de localización es útil.
- Su aplicación en búsqueda de grafos es inmediata.
  - El centro de cada celda es un nodo en el grafo.
  - 4-conexas u 8-conexas (se permite diagonal)



# Cuadrículas regulares (2)

- Inconvenientes
  - Sesgo en la digitalización
    - Si un objeto cae en un área muy pequeña de la celda, se marca como ocupada
    - Una solución:
      - cuadrículas muy finas (6 a 10 cm/cuadrado)
    - Costo muy alto de almacenamiento y un número muy alto de nodos en procesos de búsqueda.

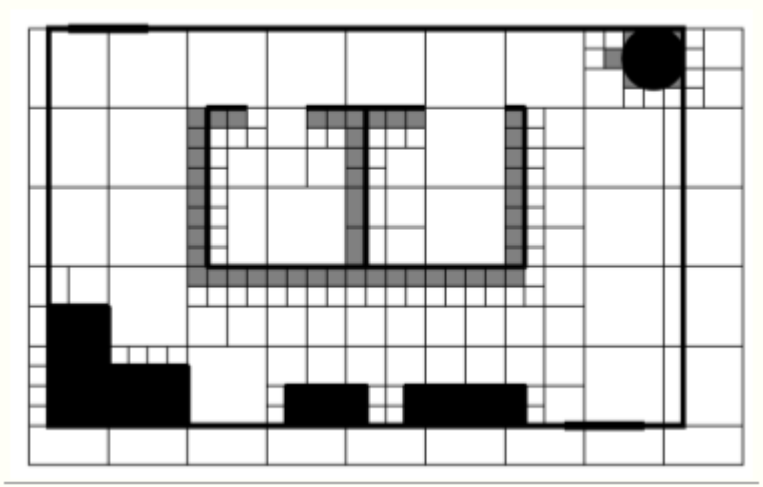




# Cuadrículas regulares recursivas (Quadtrees)

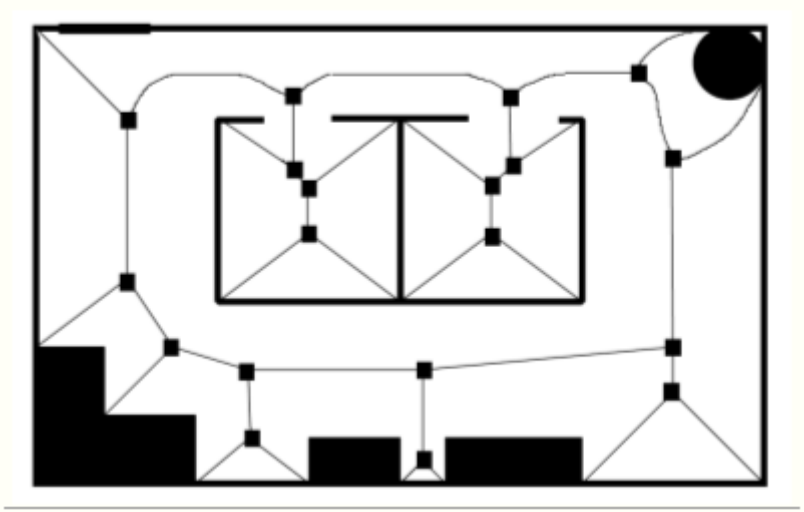
- Variante de cuadrícula regular

- Evitar espacio desperdiciado
  - Tamaño celda inicial (p.ej: 16x16cm)
  - Si un objeto ocupa parte, pero no toda la celda
  - Entonces el algoritmo de Cspace divide esa celda en 4 (quad) celdas.
  - Hacerlo recursivamente.
- Ver la opción "View> Data> Occupancy grid" de Stage



# Esqueletización

- Grafos de Voronoi



  - Arco de Voronoi: línea equidistante a todos los puntos de obstáculos.
    - Esta idea genera líneas que van por el centro del los pasillos
  - Vértice (nodo) de Voronoi: punto donde se encuentran varios arcos de Voronoi
  - Un grafo de Voronoi, construido a partir de un mapa, involucra una estrategia de control implícita
    - Permite a un robot navegar de un vértice a otro permaneciendo equidistante a todos los obstáculos.
    - Si sigue un arco de Voronoi, no colisionará.
  - Requieren más tiempo para construirse, pero la planificación es más eficiente que en descomposición de celdas.

# RVIZ: visualización

- **rviz** permite ver el proceso de mapeo.
- **rviz** es una herramienta de visualización 3D de ROS que nos permite ver el mundo desde la perspectiva del robot.
- Rviz user guide and tutorials

  http://wiki.ros.org/rviz
- Para ejecutar **rviz**

```
$ rosrun rviz rviz
```

# Si RVIZ no arranca a la primera...

- Desactivar aceleración hardware
  - Si vuestro sistema usa Mesa graphics drivers (e.g. para Intel GPUs, dentro de una VM), la aceleración hardware puede causar problemas.
  - Antes de ejecutar rviz hacer

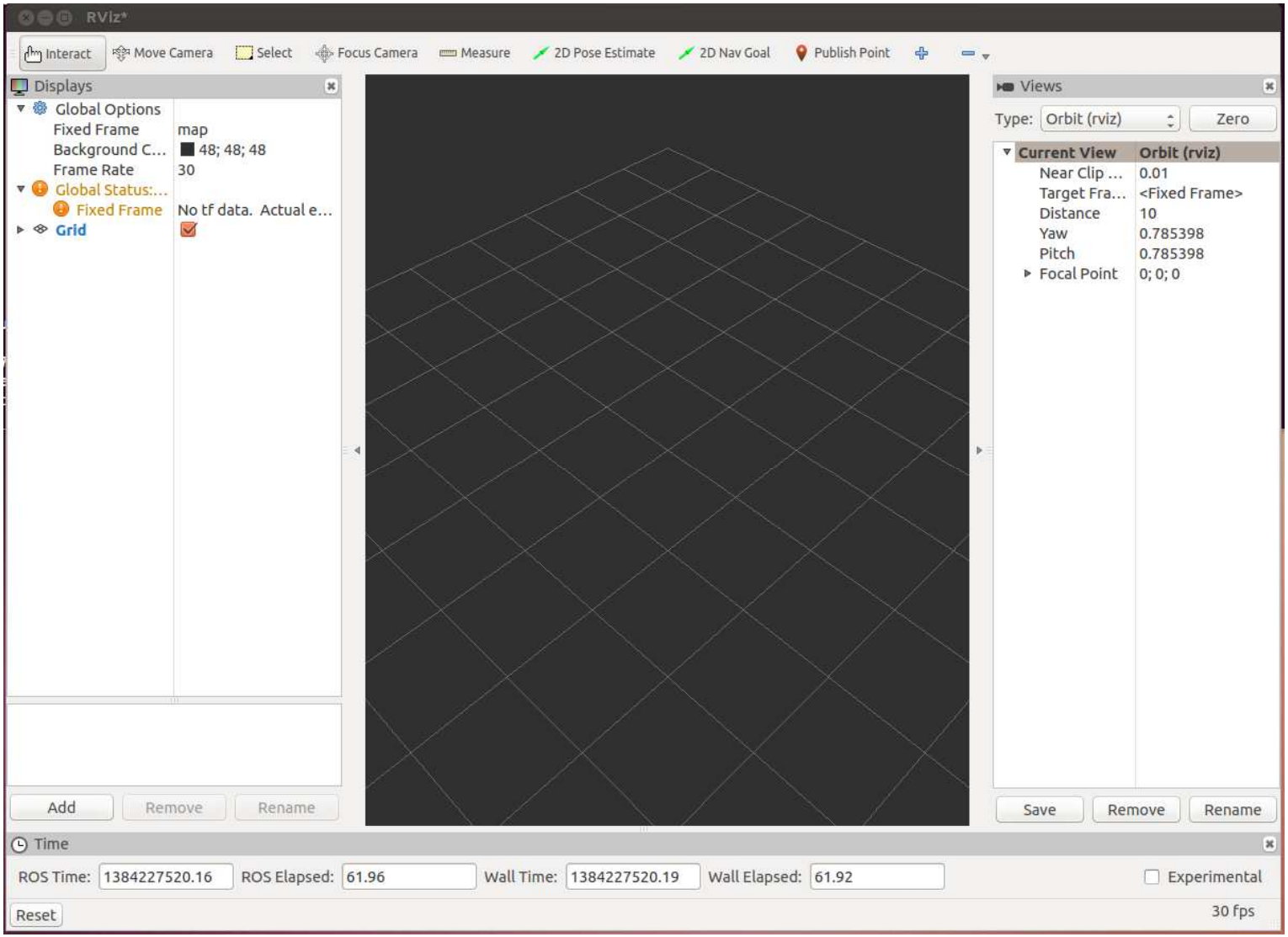
```
$ export LIBGL_ALWAYS_SOFTWARE=1
$ rosrun rviz rviz
```

- Si persiste, usar opción -sync

```
$ export LIBGL_ALWAYS_SOFTWARE=1
$ rosrun rviz rviz -sync
```

- Si persiste, probar a borrar cualquier contenido de ~/.rviz:

```
$ rm -R  ~/.rviz/*
```

- La primera vez se ve una vista 3D vacía
- A la izquierda hay un área de **Displays,** que contiene una lista de varios elementos en el mundo.
  - Ahora solo contiene opciones globales y la rejilla (grid).
- Debajo de el área de Displays hay un botón **Add** button que permite añadir más elementos que visualizar
  - En generla asociados con los topics y/o mensajes que publican los nodos.


# rviz Displays

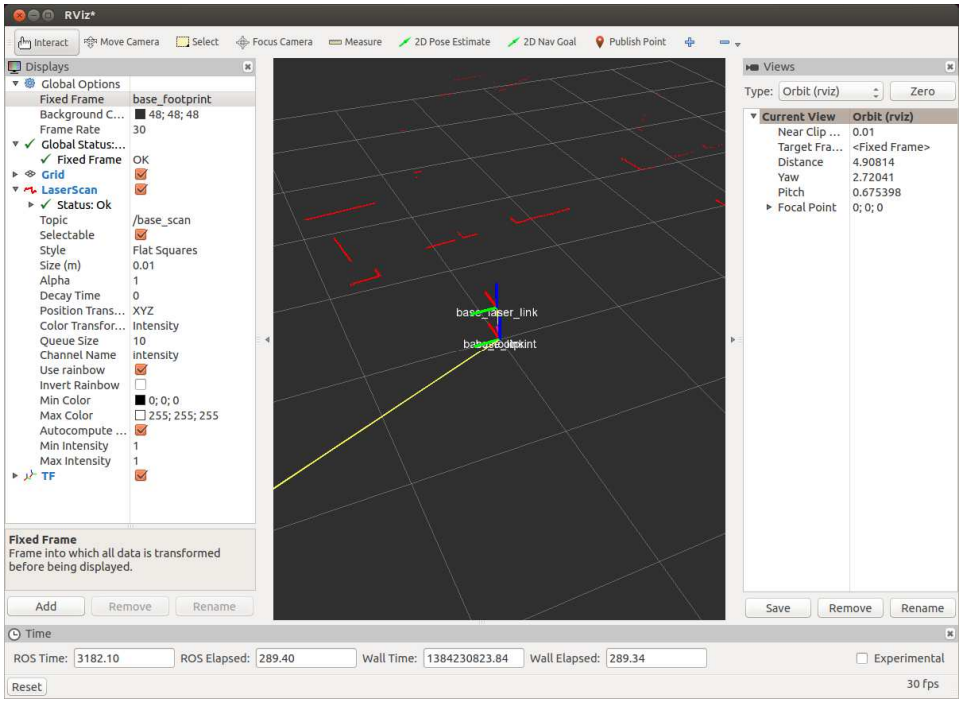
| Display name | Description | Messages Used |
|---|---|---|
| Axes | Displays a set of Axes | |
| Effort | Shows the effort being put into each revolute joint of a robot. | sensor_msgs/JointStates |
| Camera | Creates a new rendering window from the perspective of a camera, and overlays the image on top of it. | sensor_msgs/Image<br>sensor_msgs/CameraInfo |
| Grid | Displays a 2D or 3D grid along a plane | |
| Grid Cells | Draws cells from a grid, usually obstacles from a costmap from the navigation stack. | nav_msgs/GridCells |
| Image | Creates a new rendering window with an Image. | sensor_msgs/Image |
| LaserScan | Shows data from a laser scan, with different options for rendering modes, accumulation, etc. | sensor_msgs/LaserScan |
| Map | Displays a map on the ground plane. | nav_msgs/OccupancyGrid |


# rviz Displays

| Display name | Description | Messages Used |
|---|---|---|
| Markers | Allows programmers to display arbitrary primitive shapes through a topic | visualization_msgs/Marker<br>visualization_msgs/MarkerArray |
| Path | Shows a path from the navigation stack. | nav_msgs/Path |
| Pose | Draws a pose as either an arrow or axes | geometry_msgs/PoseStamped |
| Point Cloud(2) | Shows data from a point cloud, with different options for rendering modes, accumulation, etc. | sensor_msgs/PointCloud<br>sensor_msgs/PointCloud2 |
| Odometry | Accumulates odometry poses from over time. | nav_msgs/Odometry |
| Range | Displays cones representing range measurements from sonar or IR range sensors. | sensor_msgs/Range |
| RobotModel | Shows a visual representation of a robot in the correct pose (as defined by the current TF transforms). | |
| TF | Displays the tf transform hierarchy. | |

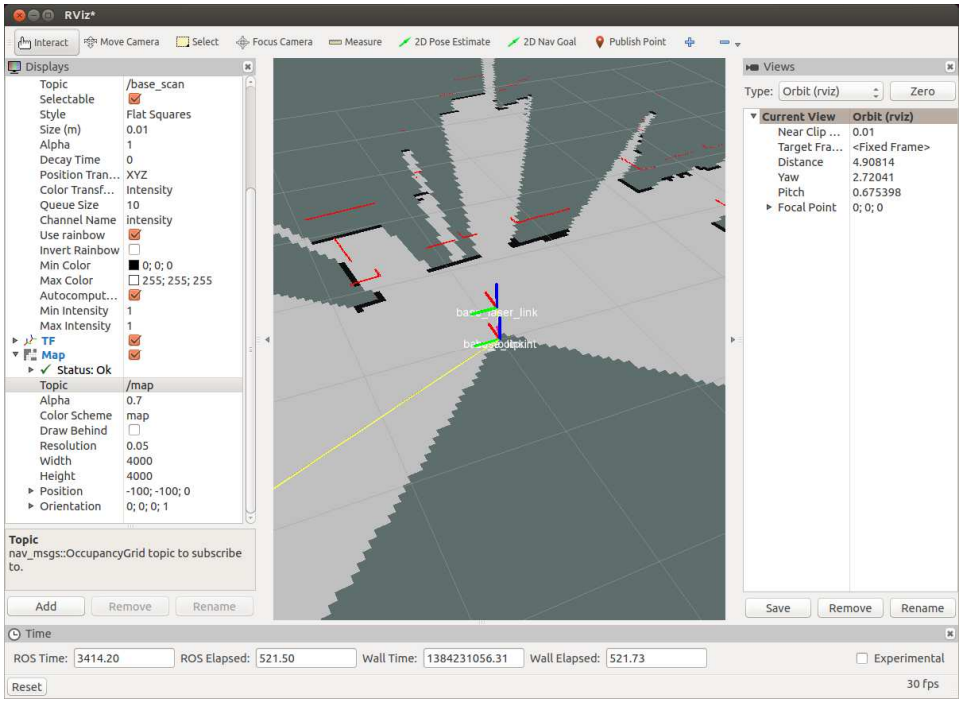
# LaserScan Display

- Click the Add button under Displays and choose the LaserScan display
- In the LaserScan display properties change the topic to /base_scan
- In Global Options change Fixed Frame to odom
- To see the robot's position also add the TF display
- The laser "map" that is built will disappear over time, because rviz can only buffer a finite number of laser scans



# Map Display

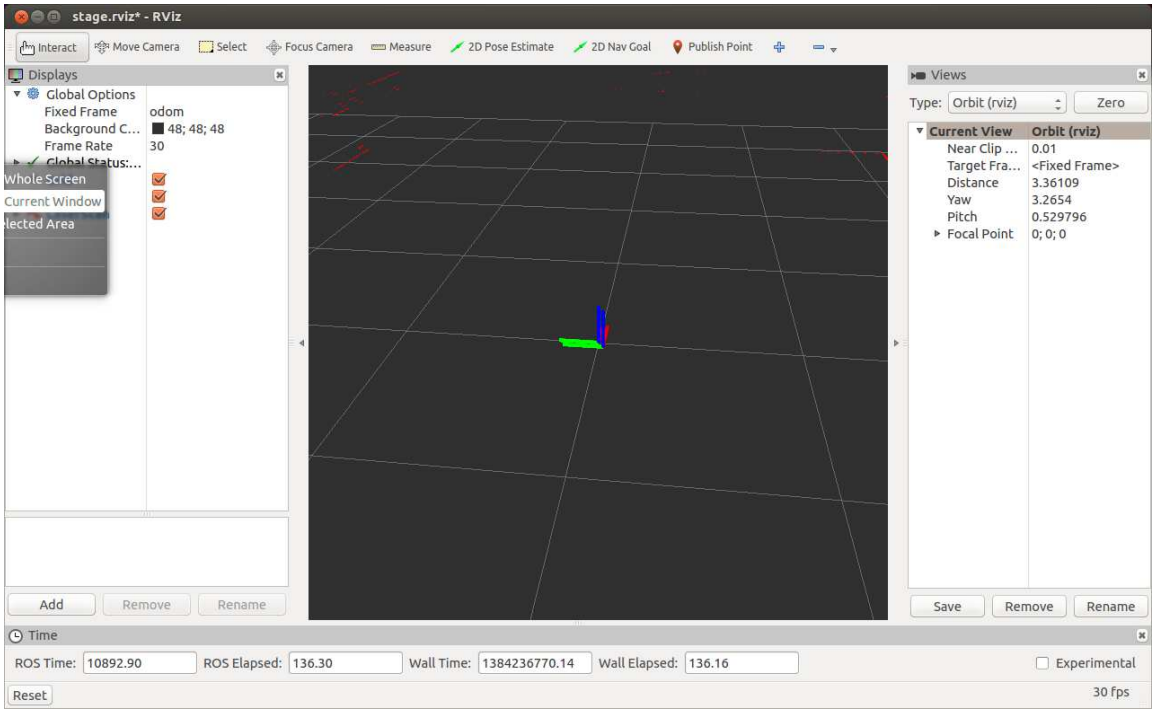
- Add the Map display
- Set the **topic** to /map
- Now you will be able to watch the mapping progress in rviz



# Rosviz con configuración predefinida

- Hay un fichero de configuración de rviz en el paquete ***stage_ros***

```
$ rosrun rviz rviz -d `rospack find stage_ros`/rviz/stage.rviz
```

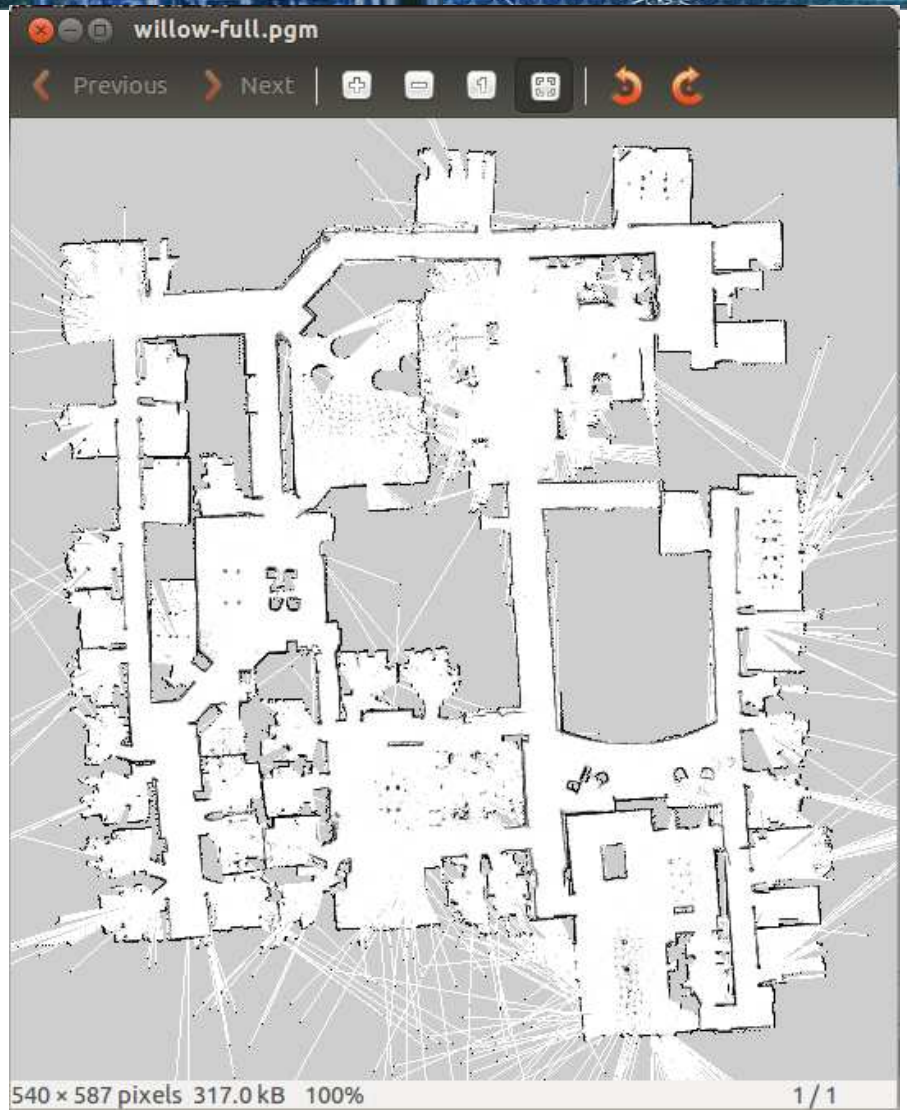


# Fichero launch para gmapping

```
<launch>
  <node name="stage" pkg="stage_ros" type="stageros" args="$(find
stage_ros)/world/willow-erratic.world"/>
  <node name="slam_gmapping" pkg="gmapping"
type="slam_gmapping">
    <remap from="scan" to="base_scan"/>
 </node>
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
stage_ros)/rviz/stage.rviz"/>
</launch>
```

# Ejercicio (no necesario entregar)

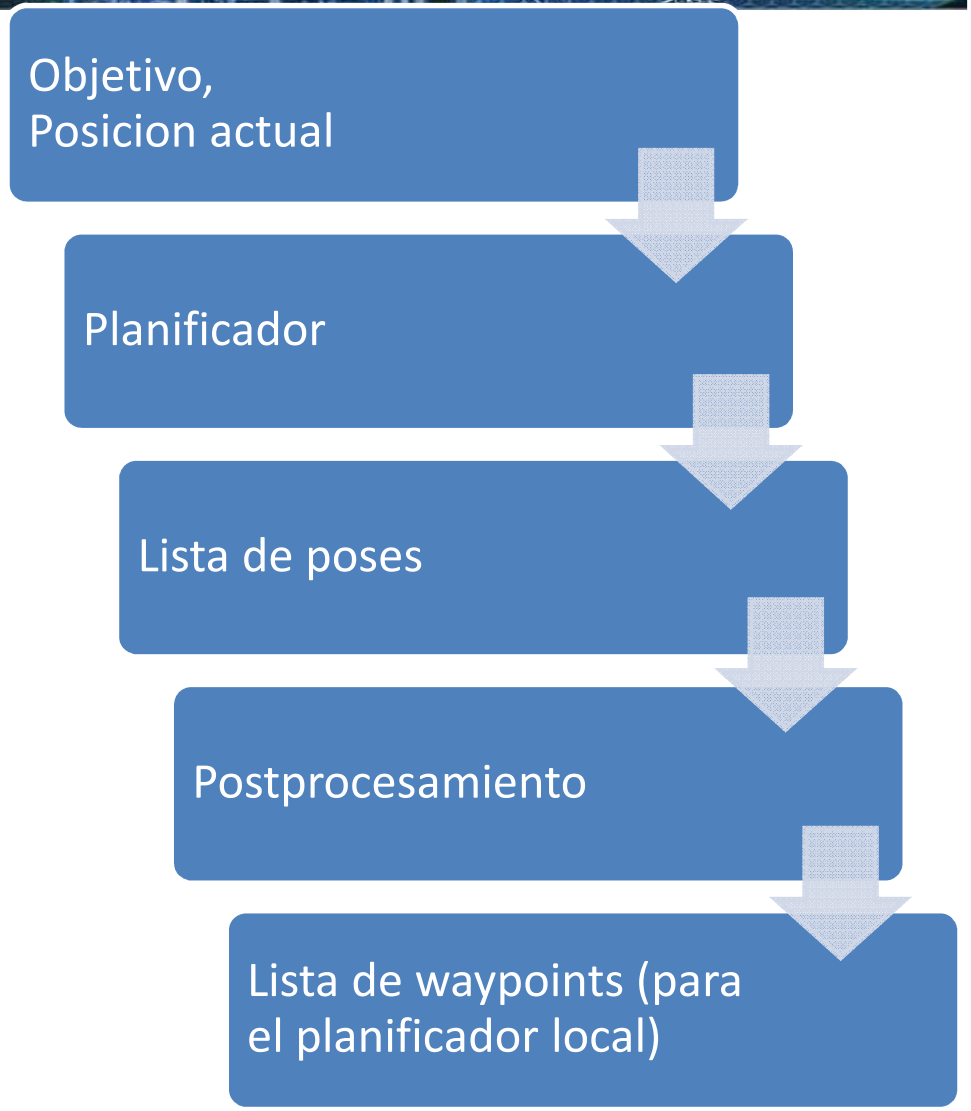
- Crear un mapa usando vuestro propio robot deambulante aleatorio
- Comparar el resultado con el mapa original en /opt/ros/hydro/share/stage_ros/world/willow-full.pgm
- ¿Cuánto tarda en crear un mapa preciso de la zona?.



# PLANIFICACIÓN GLOBAL (PLANIFICACIÓN DE CAMINOS MÉTRICA)

# Planificación de caminos

- Objetivo de la planificación de caminos:
  - Determinar un camino hacia un objetivo especificado.
  - Camino: secuencia de "poses"
  - Objetivo: una pose.
- Principales características
  - Tratan de encontrar un camino óptimo.
  - Waypoint (hito)
    - La secuencia de configuraciones se "postprocesa" y se obtiene un conjunto de subobjetivos
    - Cada punto guía (subobjetivo) es una pose (x,y, theta).
    - Ubicaciones donde el robot podría cambiar su orientación



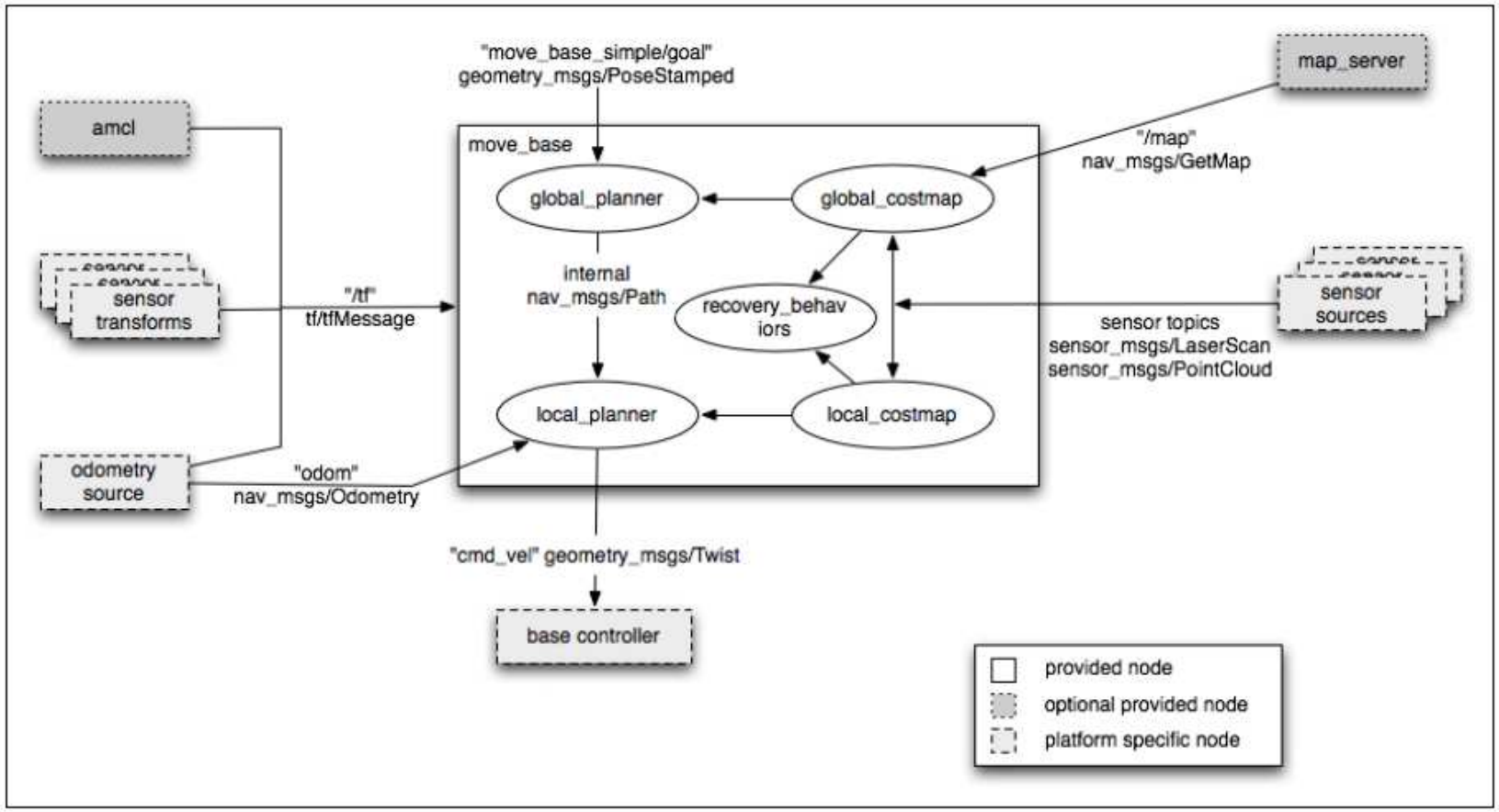
# Planificación de caminos (2)

- Componentes de un planificador de caminos global
  - La representación
    - Normalmente cuadrículas regulares, en ROS se utiliza el concepto de **costmap** para representar el mundo discretizado.
    - http://wiki.ros.org/costmap_2d
  - El algoritmo
    - Problema de búsqueda en grafos (por ejemplo A*)
    - Tratamiento directo de la cuadrícula, similar a un problema de "coloreado de grafos".
  - Un problema que siempre aparece
    - ¿Cuándo uso el planificador y cuando me muevo?
    - Entrelazado de planificación y ejecución, este problema está resuelto en ROS con el paquete move_base http://wiki.ros.org/move_base dentro de navigation stack.

# ROS Navigation Stack

- 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-indigo-navigation**
- ROS Navigation Introductory Video

# Navigation Stack



# Navigation Stack Requirements

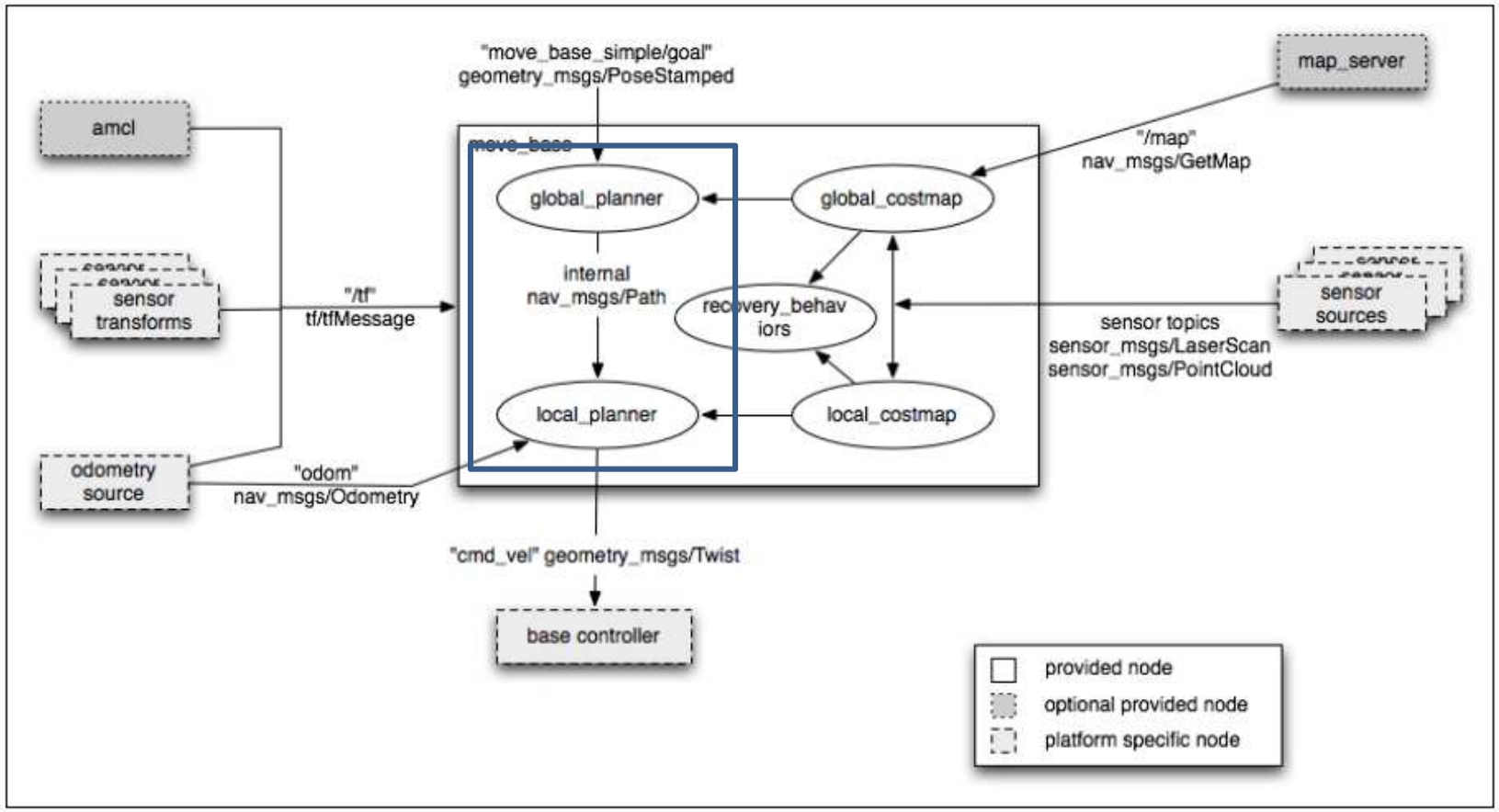
- Tres requisitos fundamentales:
  - *Navigation stack* solo maneja robots con ruedas con conducción diferencial y holonómicos.
    - Puede hacer algo más con robots bípedos, como localización, pro siempre que el robot no se mueva de lado
  - La base tiene que tener un laser montado para poder crear 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



# Ejecutar ROS navigation_stack con *Stage*

- Descargar los tutoriales de navegación en git
  - 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
```

# move_base Configuration File

Ver en src/navigation_tutorials-hydro-devel/navigation_stage/move_base_config

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

# Move_base.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.
-->
  <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="10.0" />

    <param name="oscillation_timeout" value="30.0" />
    <param name="oscillation_distance" value="0.5" />
    <!--
    <param name="base_local_planner" value="dwa_local_planner/DWAPlannerROS" />
    -->


    <param name="planner_patience" value="20.0" />
    <param name="base_global_planner" value="my_astar_planner/MyAstarPlanner"/>


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


# Running ROS Navigation Stack in Stage

| 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 |

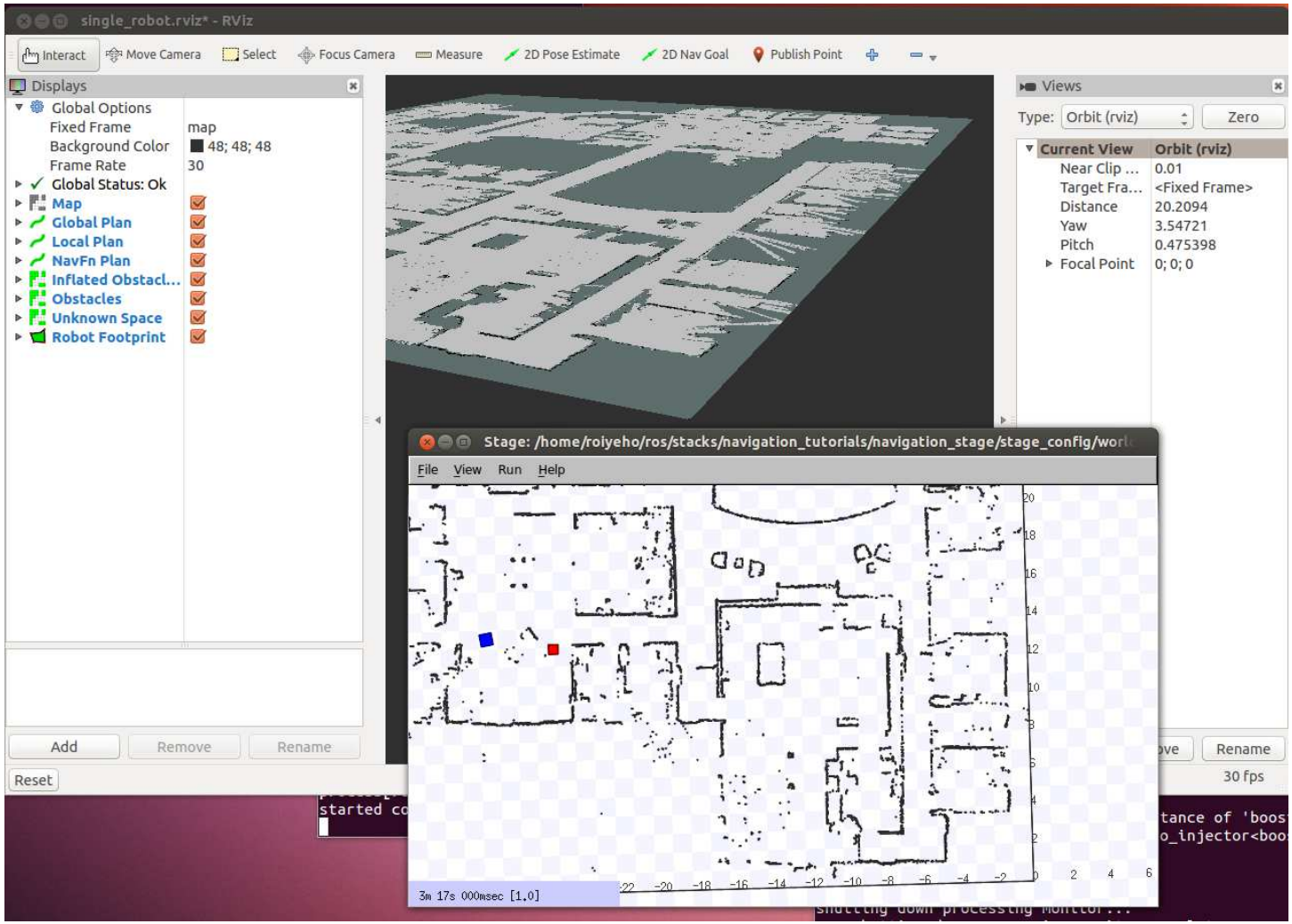
# move_base_amcl_5cm.launch

```
<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:

```
$ cd ~/catkin_ws_navegacion/navigation_tutorials-hydro-devel/navigation_stage/launch
$ roslaunch move_base_amcl_5cm.launch
```

# Ejecutando fichero launch

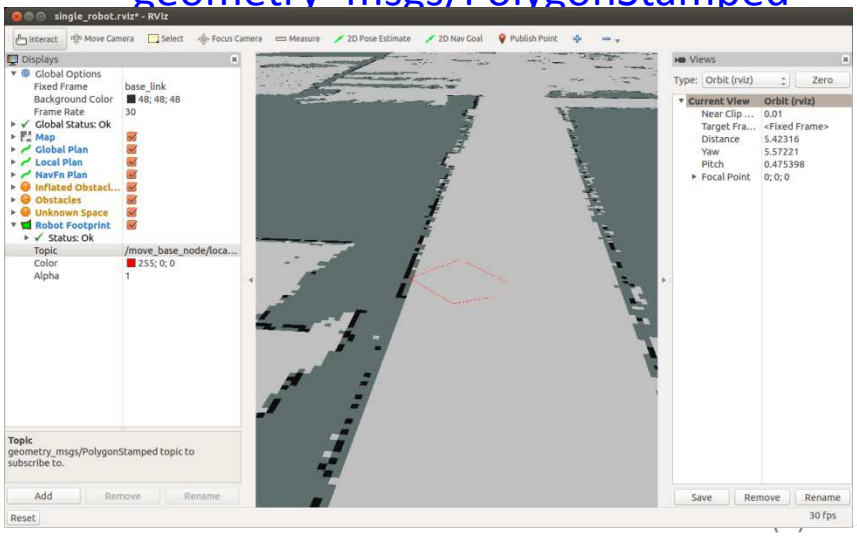



- Configurar *rviz* con *navigation stack*:
  - Asignar la pose del robot para el sistema de localización.
  - Visualizar toda la información que suministra ***navigation stack***.
  - Enviar goals desde rviz.

# 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_layer/footprint_stamped
- Type:
  - geometry_msgs/PolygonStamped

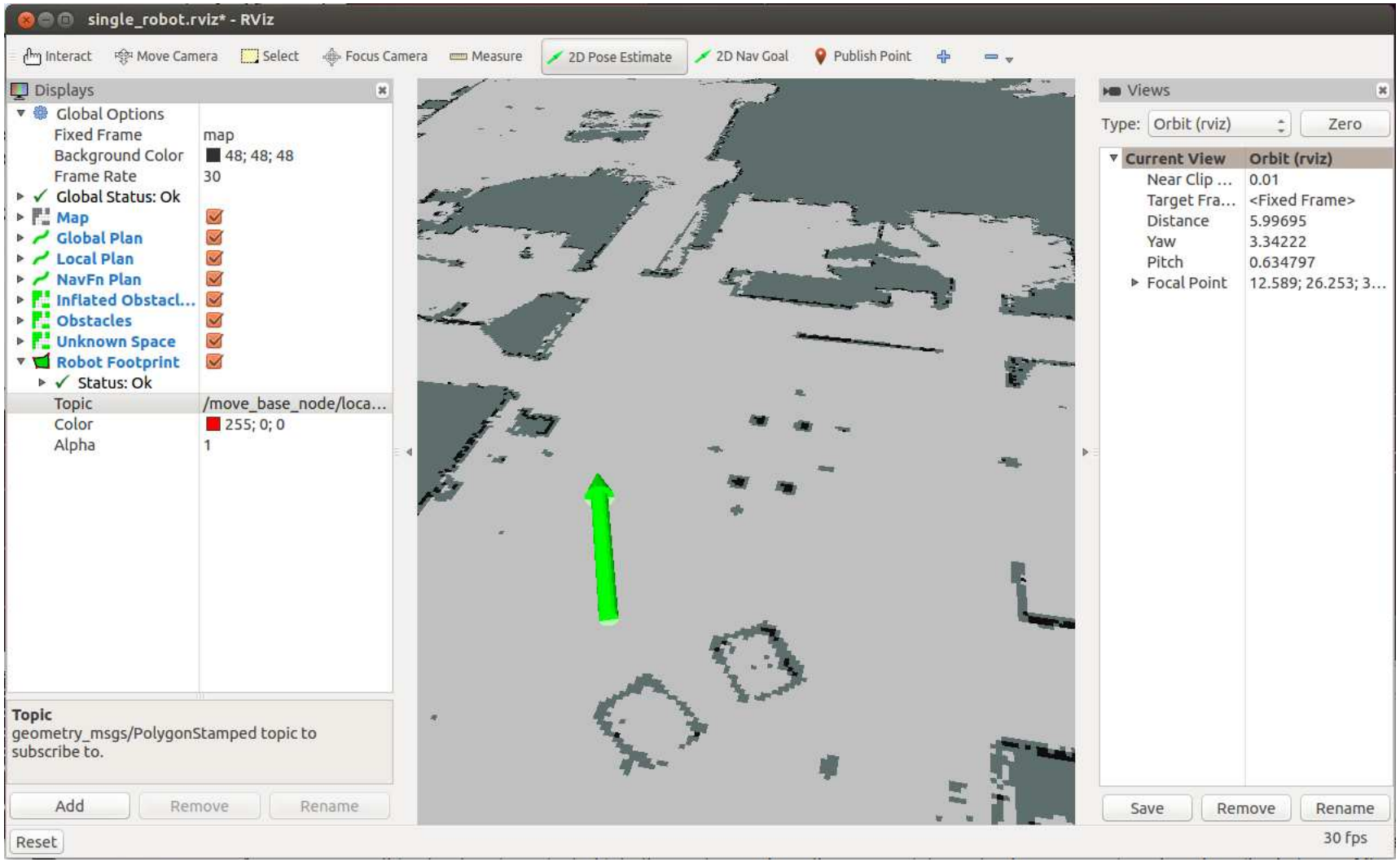
```
roiyeho@ubuntu: ~
roiyeho@ubuntu:~$ rostopic echo move_base_node/local_costmap/footprint_layer/foo
tprint_stamped -n1
header:
  seq: 2175
  stamp:
    secs: 438
    nsecs: 0
  frame_id: odom
polygon:
  points:
    -
      x: -0.334999978542
      y: -0.334999978542
      z: 0.0
    -
      x: -0.334999978542
      y: 0.334999978542
      z: 0.0
    -
      x: 0.334999978542
      y: 0.334999978542
      z: 0.0
    -
      x: 0.469999998808
      y: 0.0
      z: 0.0
    -
      x: 0.334999978542
      y: -0.334999978542
      z: 0.0
---
roiyeho@ubuntu:~$
```



# 2D Pose Estimate

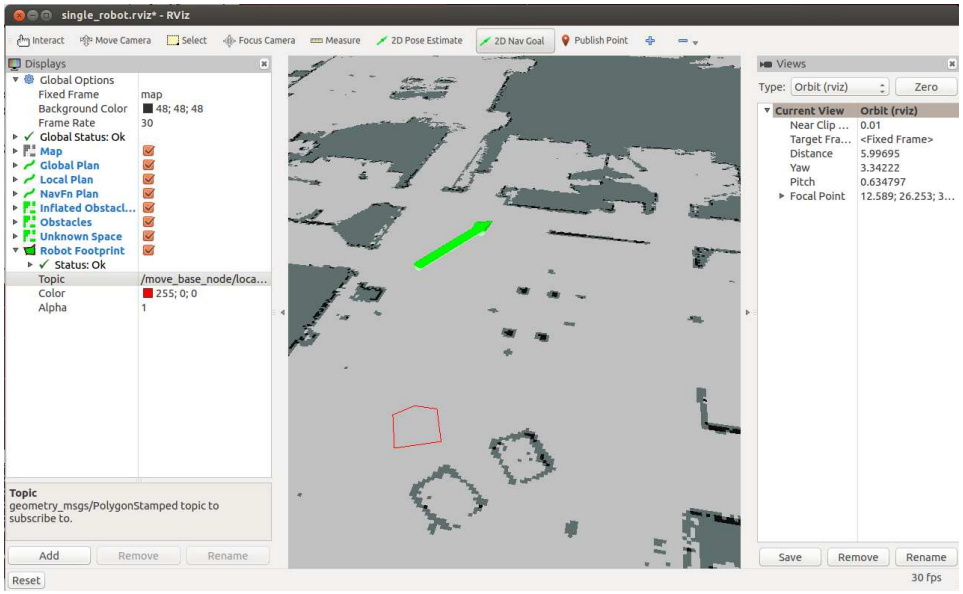
- La estimación de la pose en 2D (P shortcut) nos permite inicializar el sistema de localización usado por *navigation* stack mediante la asignación de la *pose* del robot.
- *Navigation* stack **espera** a que se le asigne esta nueva pose en un *topic* llamado **initialpose**.
- Para asignar la *pose*
  - 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

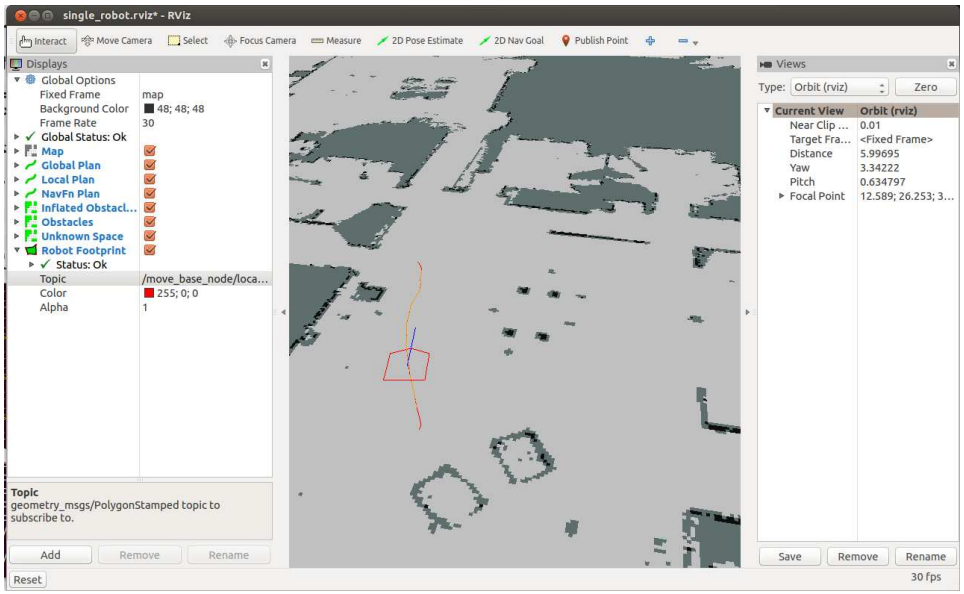


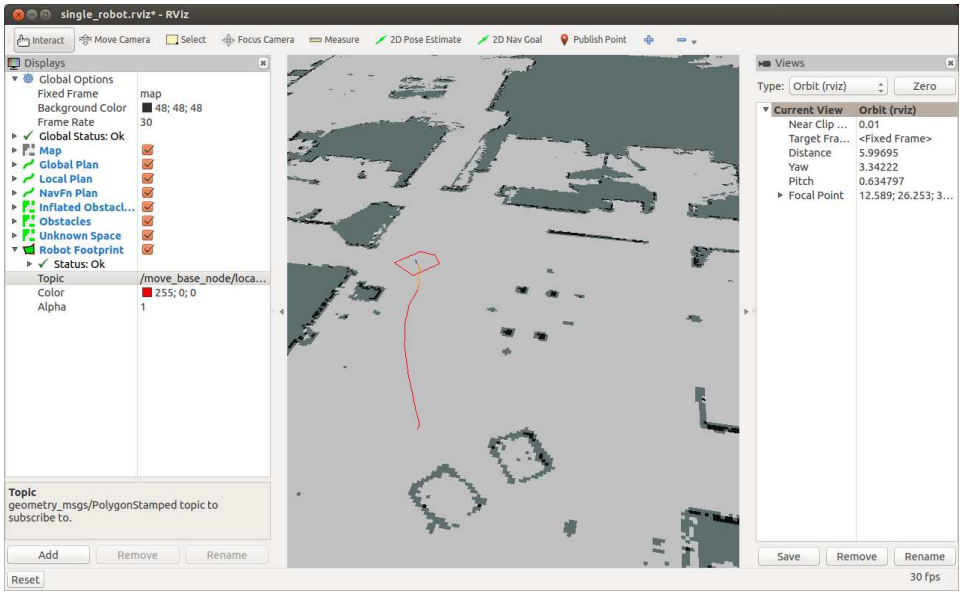
# 2D Nav Goal

- 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 la posición (x,y) y la orientación (theta) del robot deseados.



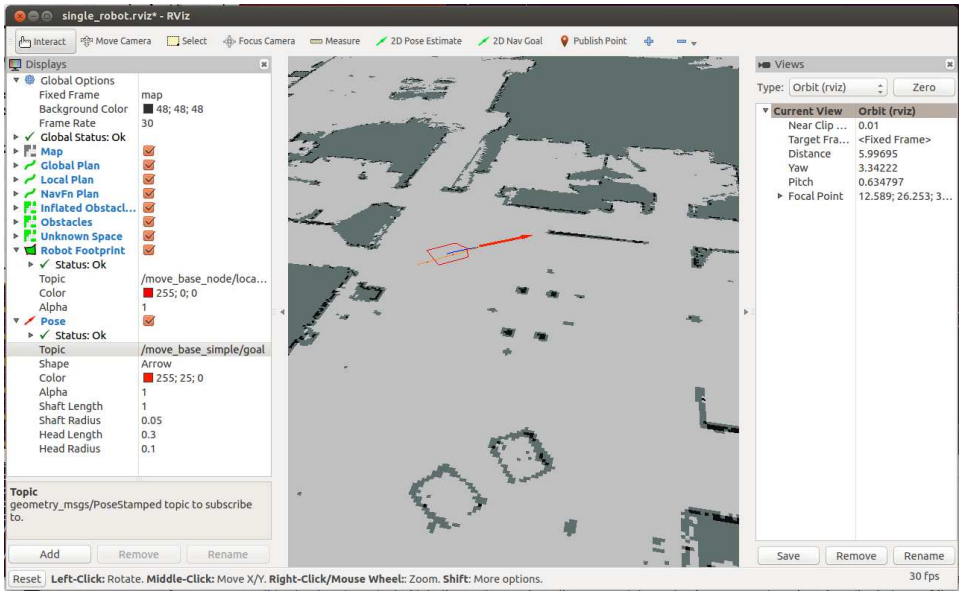
# El robot se mueve al destino



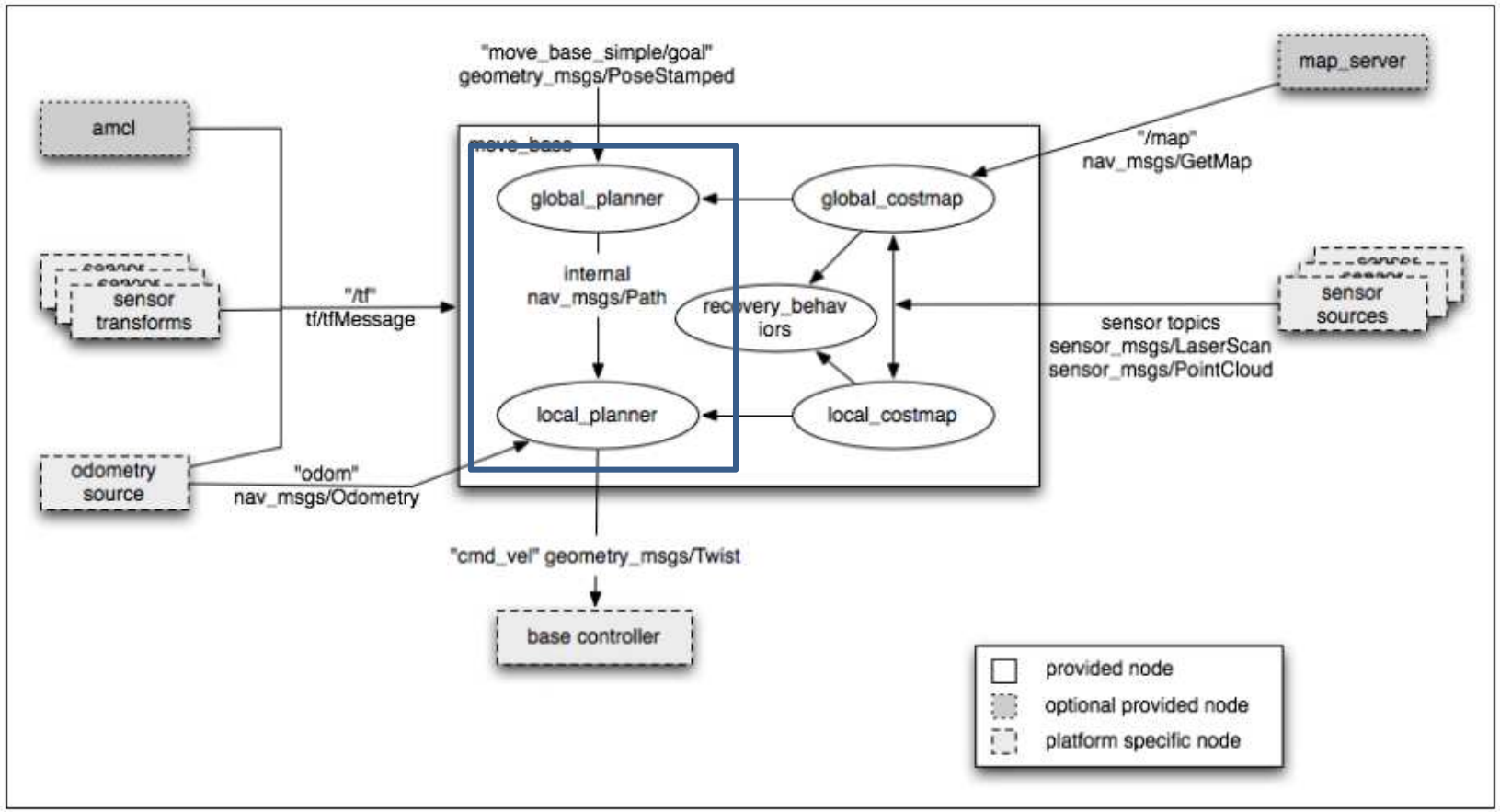



# Current Goal

- 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.



# Navigation Stack



# Global and Local Planner

**Global planner:** *navfn*

http://wiki.ros.org/navfn

- este paquete provee 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 caminio de coste mínimo en una cuadrícula.
- La función de navegación (por eso ***navfn )*** se calcula según el algoritmo de Dijkstra's
  - aun no tienen soporte para A*.

**Local planner:** *base_local_planner*

http://wiki.ros.org/base_local_planner

- este paquete provee 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.

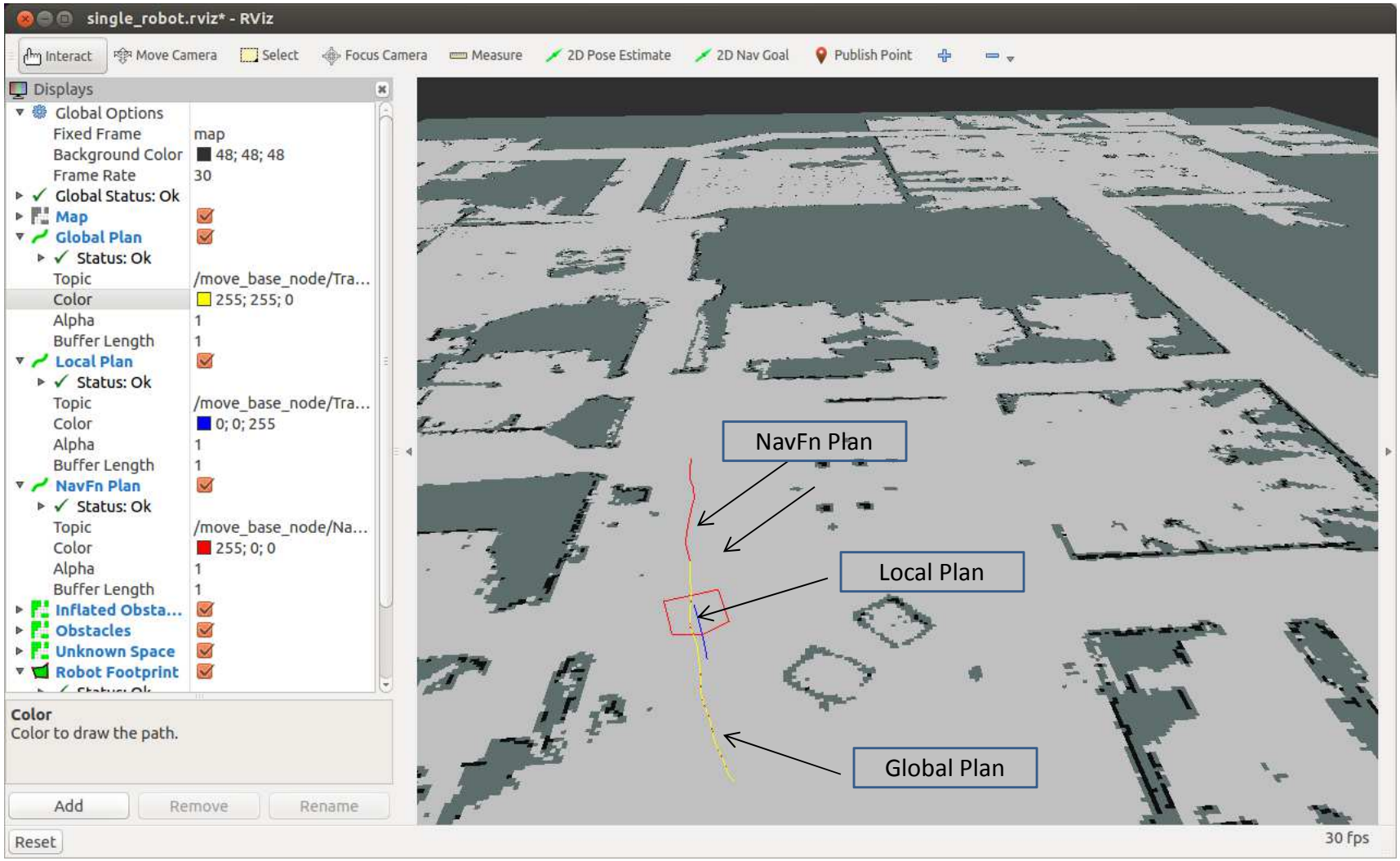
Ver documentación en: http://wiki.ros.org/navigation/Tutorials/Writing%20A%20Global%20Path%20Planner%20As%20Plugin%20in%20ROS

## 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

# Navigation Plans in rviz



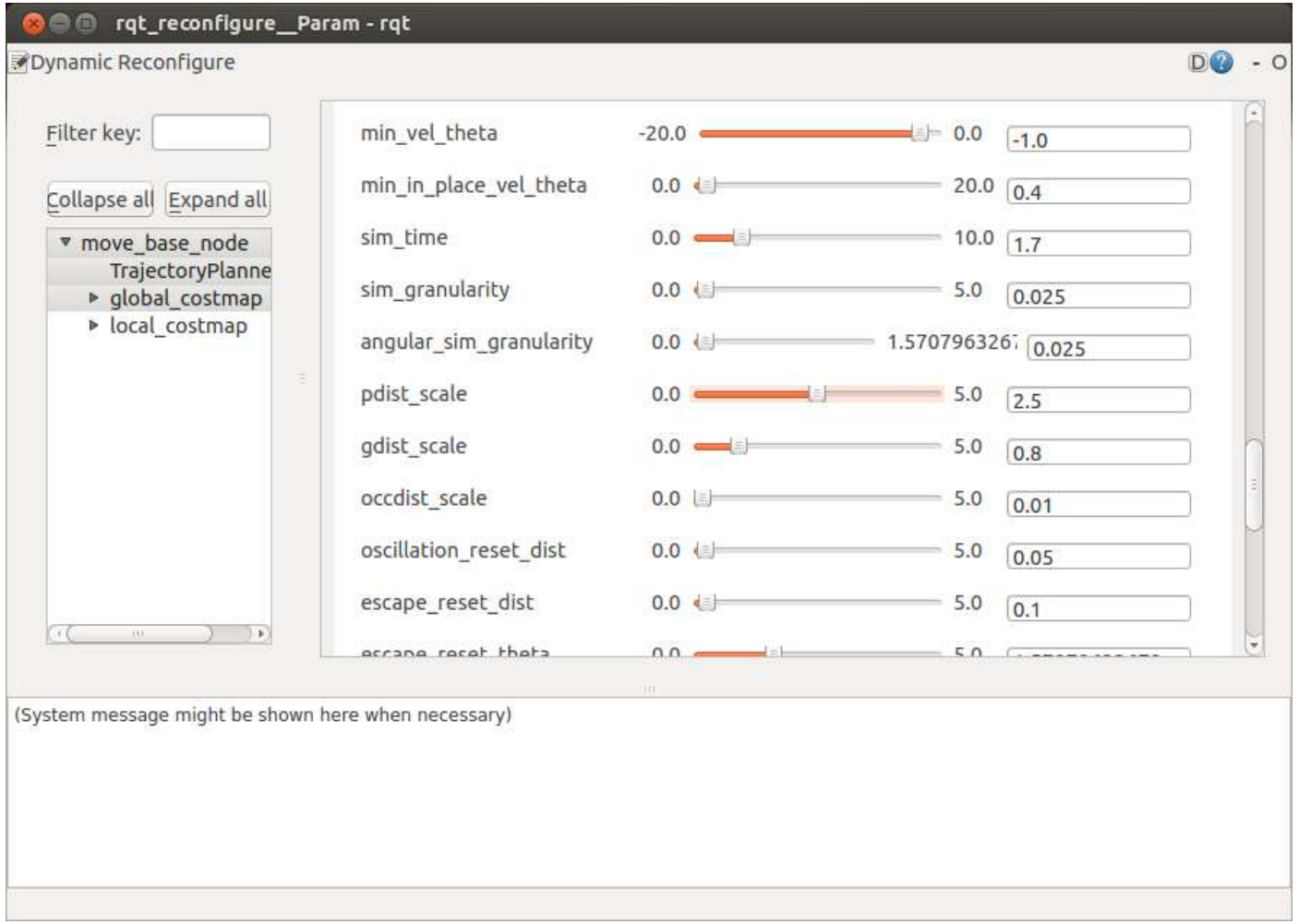
# Changing Trajectory Scoring

- Run the rqt_reconfigure tool

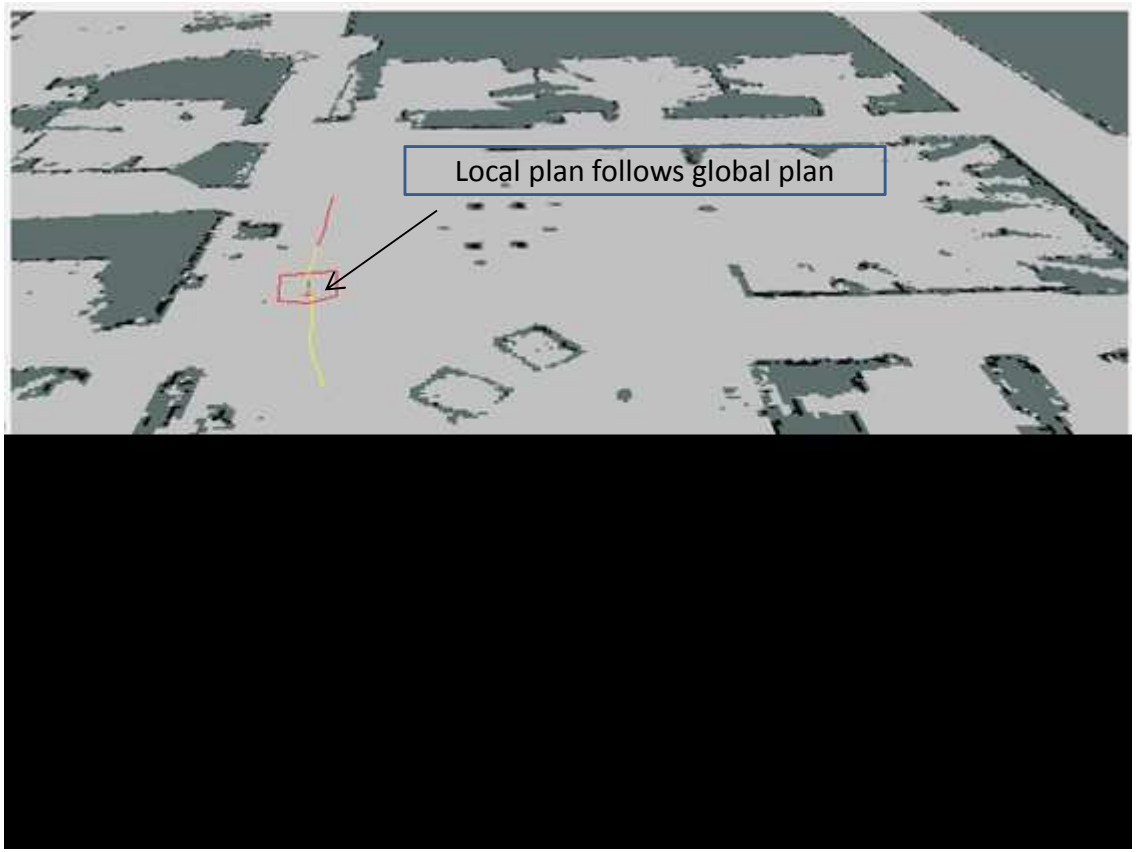
```
$ rosrun rqt_reconfigure rqt_reconfigure
```

  - This tool allows changing dynamic configuration values
- 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



# CONFIGURAR MOVE_BASE CON UN PLANIFICADOR GLOBAL DE COSECHA PROPIA

# ¿Qué tengo que saber?

- Move_Base funciona con un planificador global por defecto pero puede cambiarse por otro planificador global implementado, 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 desarrollo un planificador global como plugin?
- ¿Cómo configurar move_base para que ejecute mi plugin?


- 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.

# 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>
```

Siguiente transparencia

fichero:
catkin_ws_navegacion/src/navigation_tutorials-hydro-devel/navigation_stage/launch/move_base*.launch

# Move_base.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.
-->
  <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="10.0" />

    <param name="oscillation_timeout" value="30.0" />
    <param name="oscillation_distance" value="0.5" />
    <!--
    <param name="base_local_planner" value="dwa_local_planner/DWAPlannerROS" />
    -->


    <param name="planner_patience" value="20.0" />
    <param name="base_global_planner" value="my_astar_planner/MyAstarPlanner"/>


    <rosparam file="$(find navigation_stage)/move_base_config/costmap_common_params.yaml" command="load" ns="global_costmap" />
    <rosparam file="$(find navigation_stage)/move_base_config/costmap_common_params.yaml" command="load" ns="local_costmap" />
    <rosparam file="$(find navigation_stage)/move_base_config/local_costmap_params.yaml" command="load" />
    <rosparam file="$(find navigation_stage)/move_base_config/global_costmap_params.yaml" command="load" />
    <rosparam file="$(find navigation_stage)/move_base_config/base_local_planner_params.yaml" command="load" />
    <!--
    <rosparam file="$(find navigation_stage)/move_base_config/dwa_local_planner_p
    -->
  </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 declaro que **myastar_planner/MyastarPlanner** es un plugin para move_base?
  - En el fichero xml del plugin que he desarrollado previamente.
- Descargar la carpeta my_astar_planner.zip y descomprimirla en
  - catkin_ws_navegacion/src/
  - comprobar que hay un directorio my_astar_planner en src.
- Observar que es un directorio de paquete, pero que contiene un xml adicional
  - astar_planner_plugin.xml
  - (el nombre da igual, lo importante es el contenido)
  - Observar que el valor del parámetro de move_base.xml que llama al 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

```
<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>
```

# 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 catkin_ws_navegacion
$catkin_make
(debería compilar sin error)
para comprobar que el plugin está exportado con éxito hacer
$rospack plugins --attrib=plugin nav_core
(debería salir una lista de ficheros xml y entre ellos)
/home/<tuhome>/catkin_ws_navegacion/src/my_astar_planner/astar_planner_plugin.xml
```

# Configurar move_base

- Descargar y descomprimir en /tmp la carpeta comprimida recursos.zip
- asumiendo que estamos en **src/navigation_tutorials-hydro-devel/navigation_stage** Hacer lo siguiente:
  - Copiar los *.jpg a stage_config/maps
  - Copiar los *.world a stage_config/world
  - Copiar move_base+global_planner.xml a move_base_config. Observar en este fichero los siguientes parámetros.

```
<param name="planner_patience" value="20.0" />

<param name="base_global_planner" value="my_astar_planner/MyAstarPlanner"/>
```

  - Ir a launch
    - Editar move_base_amcl5cm.launch, guardarlo como **move_base_amcl5cm+global_planner.launch** , sustituyendo la referencia a “move_base.xml” por “move_base+global_planner.xml”

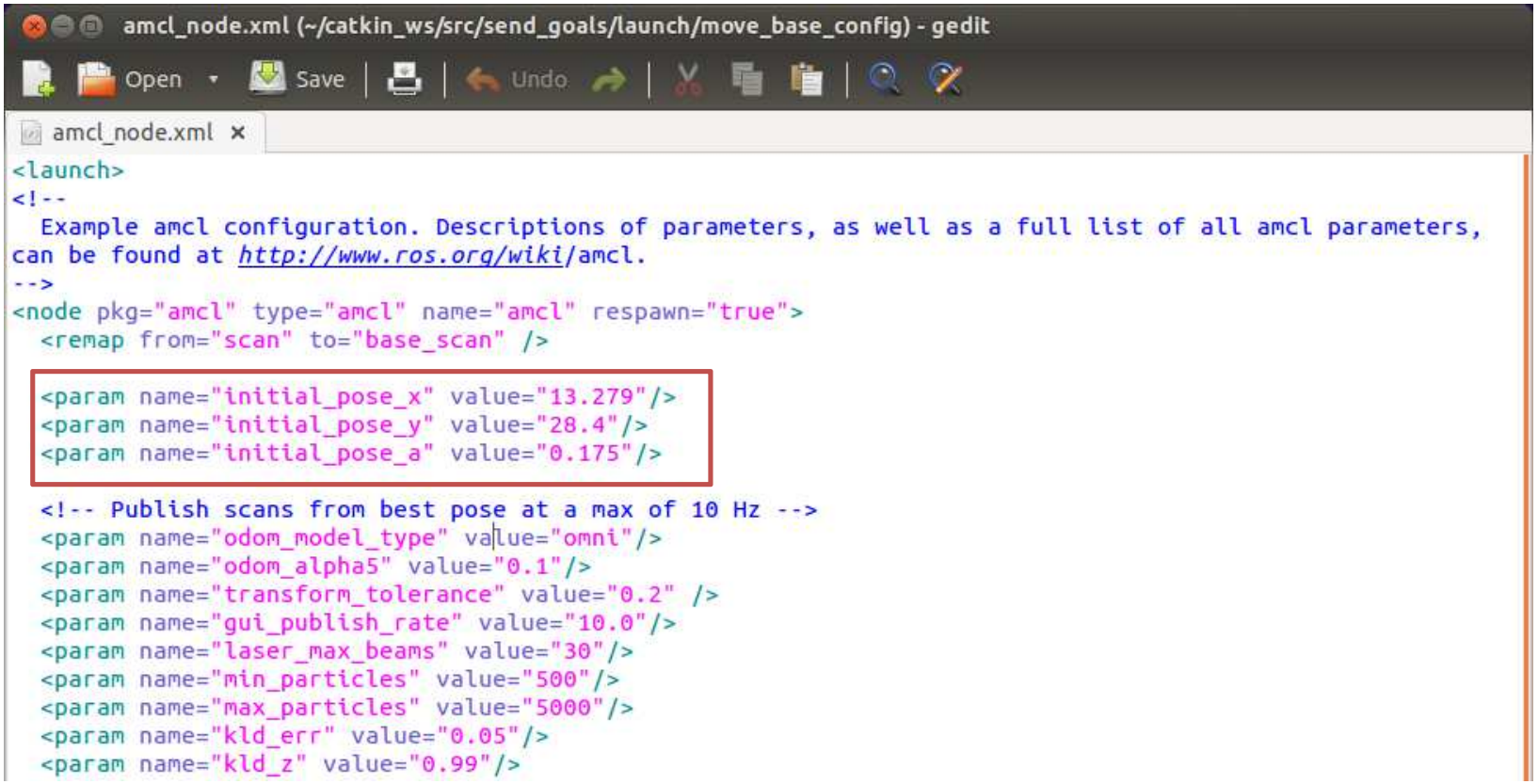
# ejecutar el launch

- Ir a navigation_stage/launch
- Ejecutar:
- roslaunch **move_base_amcl5cm+global_planner.launch**
- Debería salir stage con rviz

- Poner:
  - Poner Initial Pose
  - Observar la footprint
  - Añadir topic de plan_total
  - Poner un objetivo (cercano)

# 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.




- Implementar A* a partir del código fuente entregado para ejecutarlo como plugin de *global_planner* en *move_base*.
  - El objetivo es obtener el camino **más corto y más seguro** desde la posición actual del robot hasta la posición dada como objetivo.
    - Camino más seguro: el que garantiza en todo paso que la distancia del robot a los obstáculos es la más amplia.
  - En el código fuente se proveen funciones que calculan heurística en función de la distancia pero no de la seguridad del camino.
- El resultado final tiene que ser no solo que se obtenga el mejor camino, además se tiene que visualizar en rviz y demostrar que el robot es capaz de seguirlo al menos algunos ciclos del planificador local sin dar error.
- Entregar:
  - Código fuente del A*, implementado modifcando el fichero my_astar.cpp.
  - Presentación powerpoint para la defensa (que será el 10 y 13 de Abril).
- Fecha de entrega para ambos grupos: 9 de Abril a las 23:59.

# MATERIAL ADICIONAL (VARIANTES DE ALGORITMOS PARA BÚSQUEDA EN COSTMAPS)

# Planificación de caminos basada búsqueda en grafos

- Un CSpace puede convertirse en un grafo
  - Nodo: una configuración (una celda)
  - Arco: celdas adyacentes
  - Nodo inicial: configuración actual
  - Nodo objetivo: configuración objetivo
- Cualquier técnica de búsqueda que ya conocéis puede aplicarse

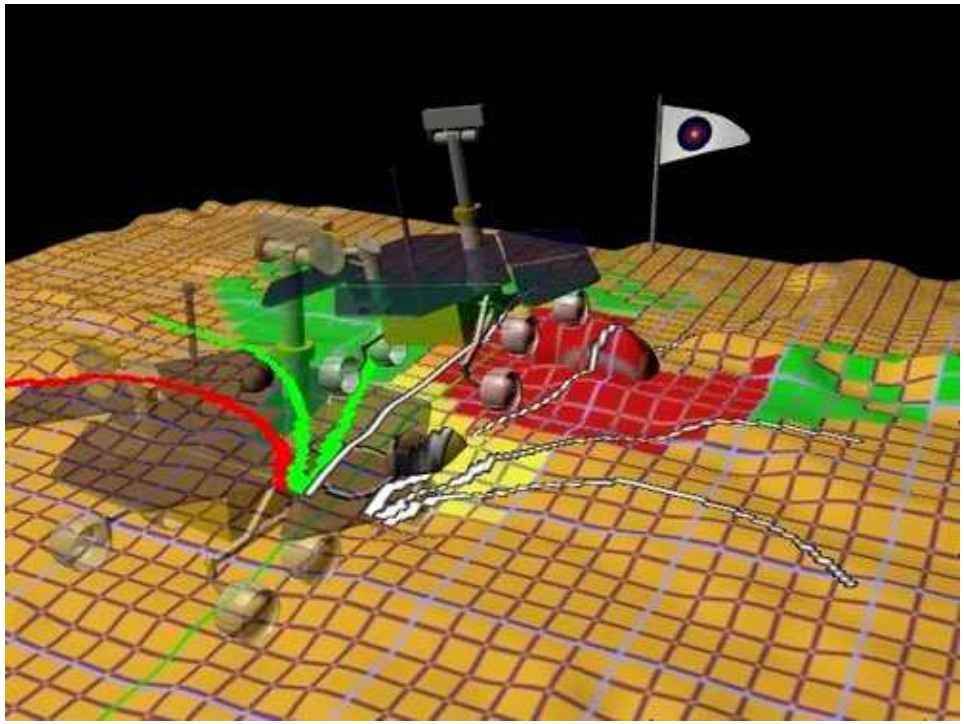
# A* es muy usado

- Produce un camino óptimo:
  - Comenzando por el nodo inicial n_0
  - Explorando sus descendientes e incrementa el camino seleccionando siempre el mejor nodo
  - Selección basada en la función f(n) = g(n) + h(n)
    - g(n) = d(n_0,n-1) + d(n-1,n)
    - h(n) =d_linearecta(n, n_f)
  - Si h(n) es admisible y monótona entonces en una búsqueda de grafos A* encuentra el camino óptimo
  - La distancia en línea recta es una heurística admisible porque nunca sobreestima el coste real hasta el objetivo

# A* para planificación de caminos

- Ventaja:
  - Puede usarse en cualquier CSpace que pueda convertirse a un grafo (cuadriculas regulares, Voronoi, conectividad,....)
- Inconveniente
  - En entornos donde hay más factores o información a tener en cuenta (p.ej: gasto de energía) hay que afinar mucho en la heurística para que siga siendo admisible.

# D*



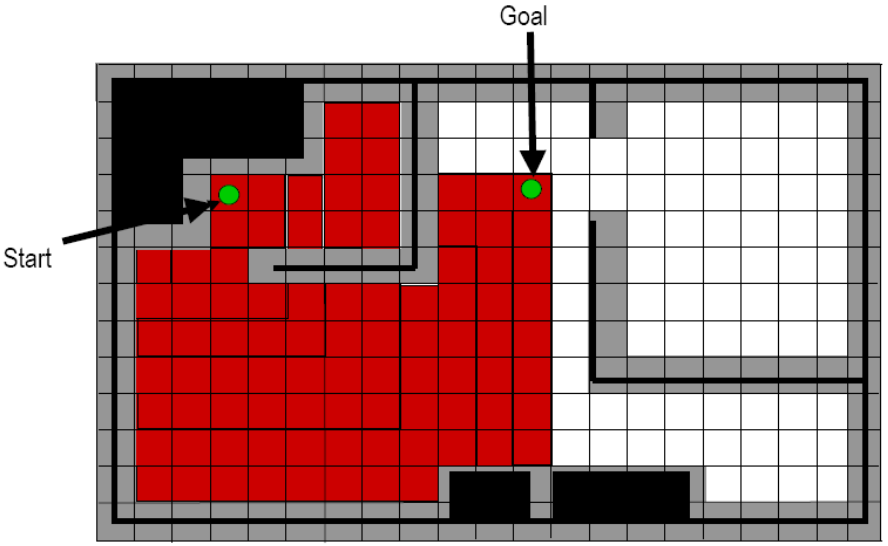
- Variante de A*
  - Planifica previamente un camino (A*) desde cualquier posición hasta el objetivo
  - A* : el camino más corto desde una sola fuente (estilo Dijkstra)
  - D*: el caminio más corto para todos los pares (situación, objetivo) (estilo Floyd-Warshall)
  - h* se basa en la idea de traversabilidad
    - la traversabilidad se actualiza dinámicamente con nueva información detectada
    - D* replanifica contínuamente, modificando su mapa con información en tiempo real

# Basados en "frente de onda" (wavefront)

- A* requiere construir un grafo (estructuras de datos adicionales)
- La técnica Wavefront es muy adecuada para trabajar directamente en la cuadrícula (en la matriz)

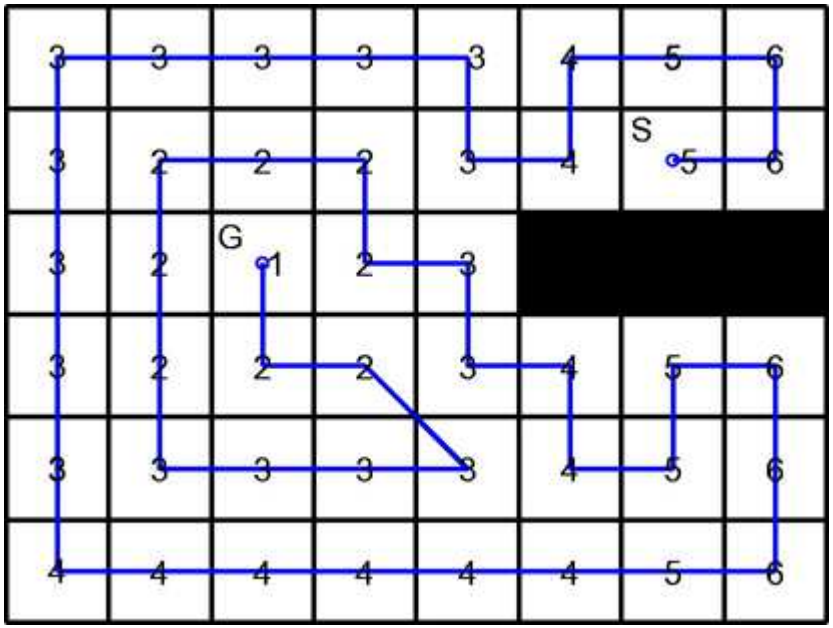
# Wavefront

- Idea
  - considerar CSpace como un material conductivo con el calor radiando desde la posición actual hasta el objetivo.
  - Si hay camino, se supone que el calor llega al objetivo.
  - Se puede obtener el camino óptimo desde todas las celdas hasta el objetivo.
  - Al final tenemos un mapa que parece un campo de potencial



# Wavefront (2)

- Algoritmo wavefront
  - 1. Propagar la onda desde el objetivo hasta el estado inicial
    - Cuadrícula binaria: 1 ocupado, 0 libre
    - Celda objetivo = 2
    - Celdas adyacentes a 2, que tengan 0, poner 3
    - Celdas adyacentes a 3, que tengan 0, poner 4
    - ....
    - Hasta llegar a la celda objetivo.
  - 2. Extraer el camino usando gradiente descendiente
    - Si la celda inicial tiene x, buscar una adyacente que tenga x-1 (grad. descendiente). Marcarla como nodo del camino (waypoint).
    - Después buscar las vecinas de esta, con x-2
    - ...
    - Hasta que llegue al nodo objetivo.
  - 3. Suavizar el camino (extraer waypoints)
    - Iterativamente, eliminar el waypoint i, si de i-1 a i+1 (en línea recta) no se atraviesa ningún obstáculo.
    - repetir hasta que no se puedan eliminar más waypoints
    - Devolver los waypoints.



# Wavefront (3)

- Puede usarse en diferentes tipos de terrenos
  - Obstáculo: conductividad 0
  - Espacio libre: conductividad infinita
  - Terreno no deseable (areas rocosas): baja conductividad -> camino más costoso.
  - Se puede usar bidireccional (para ahorra coste computacional)

# ENTRELAZANDO PLANIFICACIÓN (DELIBERATIVA) Y EJECUCIÓN (REACTIVA)

# Aproximación inicial:

- Primero planifico, luego ejecuto.
  - Para cada waypoint (subobjetivo)
    - Lo tomo como vector direccional de entrada
    - Aplico un comportamiento reactivo "move-to-goal", basado en campos de potencial, por ejemplo
  - Hasta que llego al objetivo global.


- Obsesión por el subobjetivo
- No hay replanificación oportunística


- Gastar tiempo y energía intentando alcanzar la posición exacta del subobjetivo.
- Porque la condición de terminación se ha puesto con una tolerancia no realista
  - Por ejemplo: ¿ya estoy en la (6,3, 90º)?
  - Estando en la (6,2.7, 90º) sigue moviéndose y llega a la (6,3.2,90º)....
  - Solución1: tolerancia
    - (abs(y_actual - y_goal) < epsilon)
  - Solución2: timeout
    - si no alcanzo el objetivo en un tiempo dado, o devuelvo error, o me quedo en un nuevo estado ("estoy perdido")

# Replanificación contínua

- La obsesión puede ser porque ha cambiado el entorno y el mapa "a priori" ya no es correcto
  - P.ej: el robot tiene una cámara que detecta terreno fangoso
  - Solución3:
    - Actualizar el mapa
    - Replanificar (con el D* por ejemplo)
    - En cada posición, conoce el camino óptimo
- Esto nos lleva a una replanificación contínua
  - Muy costosa (p.ej: en un rover planetario)
  - Sujeta a errores por sensores con ruido.

# Replanificación basada en eventos

- En lugar de replanificar en cuanto se detecta un pequeño desvío del plan original
- Tomar una medida de "cómo de importante" es el desvío
- Solo replanificar en este caso.