Simulador Stage

ROS Stage Simulator

- http://wiki.ros.org/stage_ros?distro=hydro
- Un simulador que provee un mundo virtual poblado por objetos, robots móviles y sensores. Los objetos pueden se detectados y manipulados por los robots.
- Stage provee modelos para varios tipos de sensores y actuadores:
  - sonar or infrared rangers
  - scanning laser rangefinder
  - color-blob tracking
  - bumpers
  - grippers
  - odometric localization
  - and more

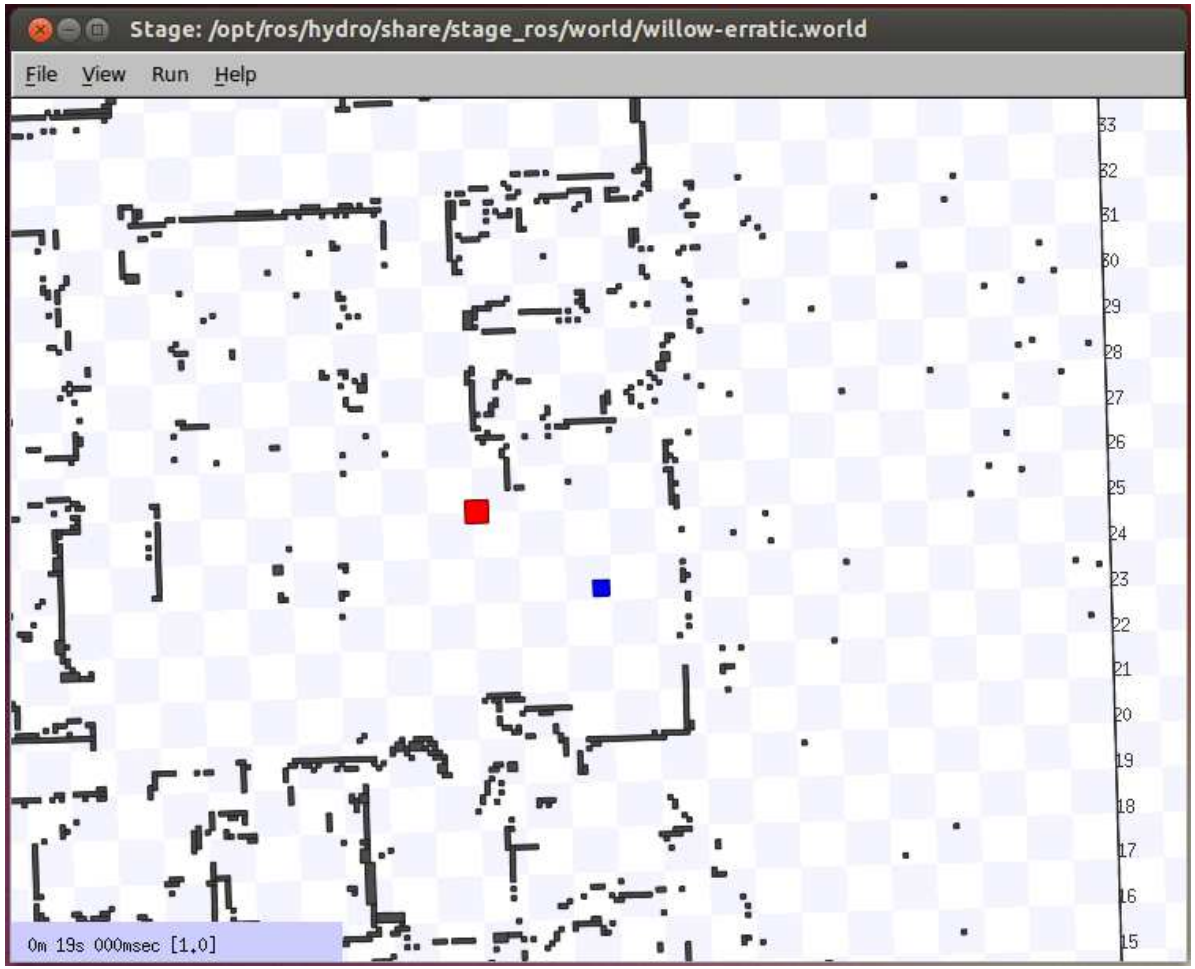
Run Stage with an existing world file

- Stage viene instalado con ROS Indigo
- Stage necesita como entrada ficheros de mundo (world files) con extensión .world.
  - Hay ejemplos: hacer **`roscd stage_ros/world`**
- Stage trae algunos ejemplos de *world* files incluyendo uno que pone un robot en un entorno similar al laboratorio de Willow Garage-like environment.
- Ejecutar:

```
rosrun stage_ros stageros `rospack find stage_ros`/world/willow-erratic.world
```

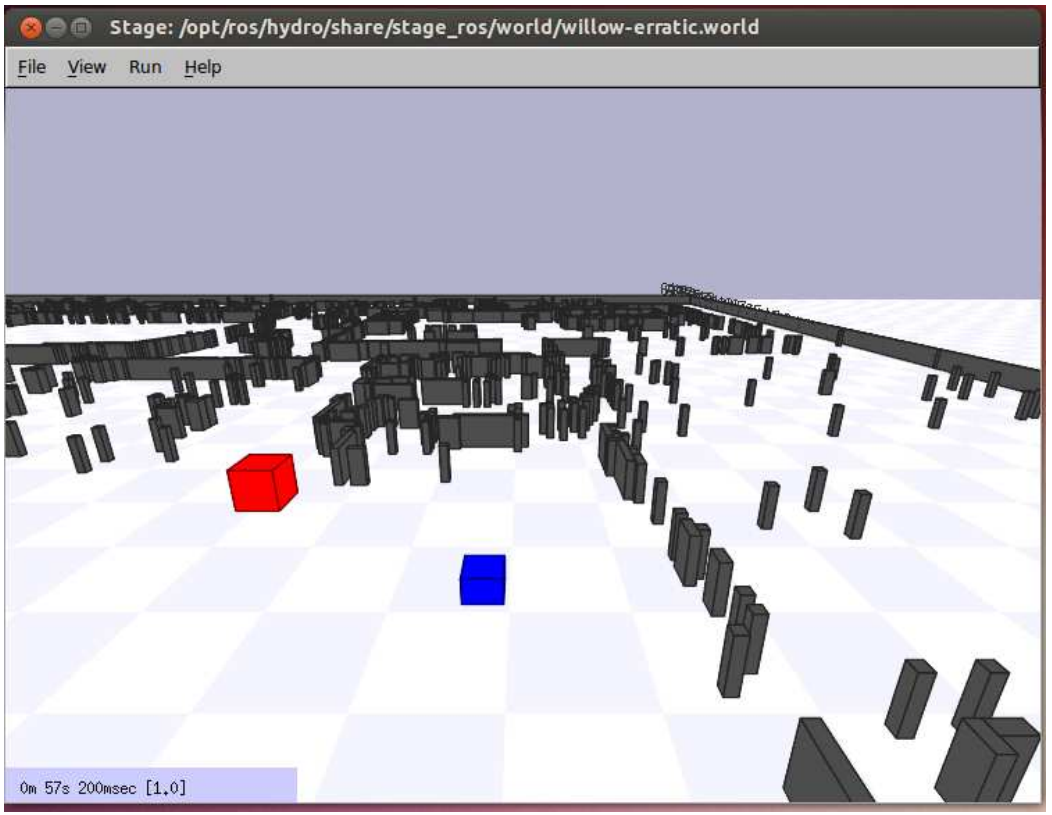
- Navegar por la ventana de Stage hasta ver dos cuadrados: el rojo es una caja, el azul un robot.
- En el material de prácticas hay un tutorial sobre cómo construir ficheros .world en Stage para dibujar objetos/robots más realistas.

Ejecutar Stage con un “world file” existente



Ejecutar Stage con un “world file” existente

- Click en la ventana de Stage y presionar “R” para una vista de perspectiva.



Moviendo el robot con teleoperación

- Para teleoperar un robot desde el teclado usaremos un ***paquete*** llamado *teleop_twist_keyboard*

- **Instalar**
  - `sudo apt-get install ros-<distro>-teleop-twist-keyboard`
- **Ejecutar**
  - `rosrun teleop_twist_keyboard teleop_twist_keyboard.py`

Move the robot around

- Now run teleop_twist_keyboard:

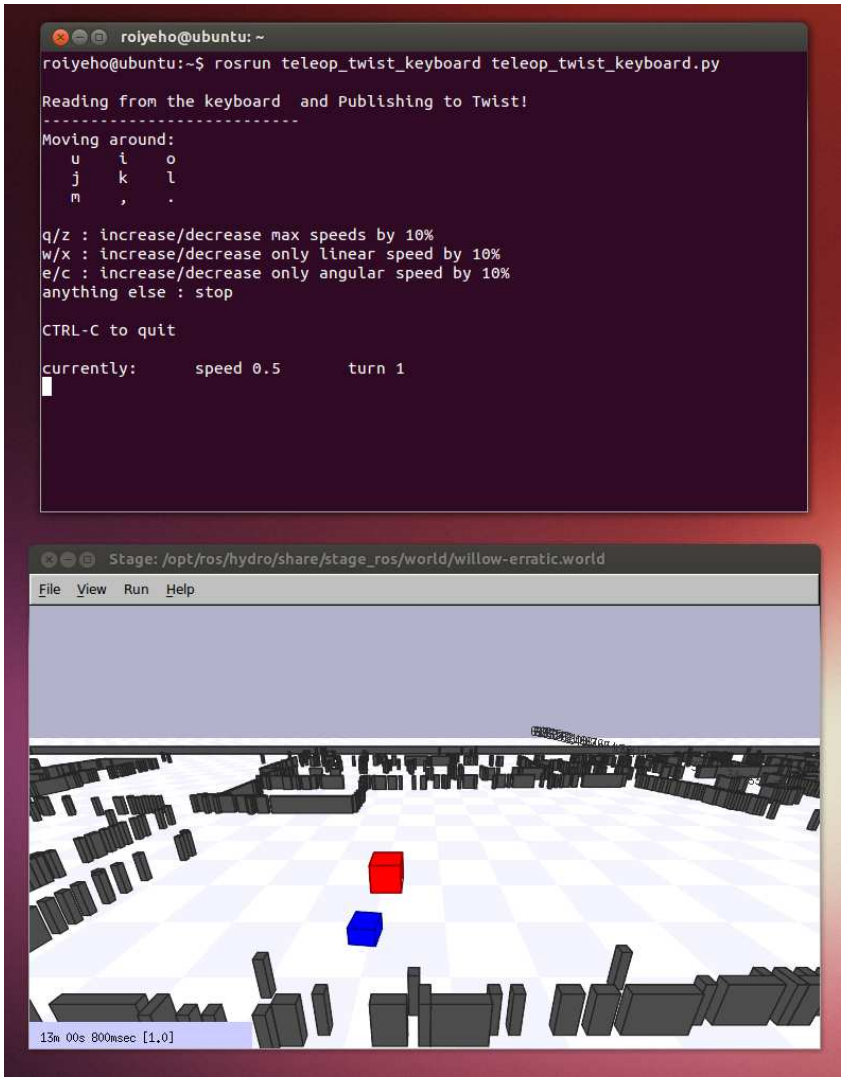
```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

- You should see console output that gives you the key-to-control mapping, something like this:

```
Reading from keyboard
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%

anything else : stop
---------------------------
```

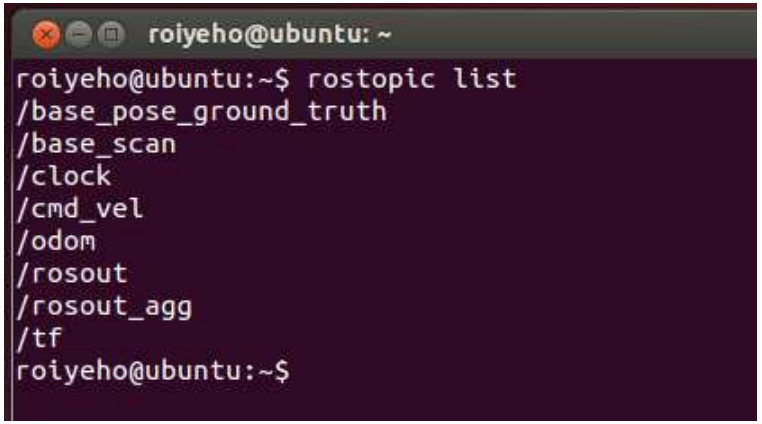
  - Hold down any of those keys to drive the robot. E.g., to drive forward, hold down the **i** key.
  - Mantener el foco en la terminal donde se ejecuta el nodo de teleoperación, si no no recibe las señales del teclado.

Move the robot around



Topics publicados por Stage

- Ver los *topics* disponibles con Stage con **rostopic list**



- Usar **rostopic echo -n 1** para ver una instancia de los datos en uno de los topics.

Mensajes de Odometría

roiyeho@ubuntu: ~

```
roiyeho@ubuntu:~$ rostopic echo /odom -n 1
header:
  seq: 11485
  stamp:
    secs: 1148
    nsecs: 600000000
  frame_id: odom
child_frame_id: ''
pose:
  pose:
    position:
      x: 1.16596142952
      y: -0.133586349782
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: -0.389418342309
      w: 0.921060994003
  covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
twist:
  twist:
```

Nodo "*Move Forward*"

- Vamos a implementar un nodo que guía al robot hasta que choca con un obstáculo.
- Para ello necesitamos publicar mensajes tipo ***Twist messages*** bajo el topic **cmd_vel** (al que está suscrito Stage)
  - Este *topic* es el responsable de enviar órdenes de velocidad al robot.

- Hacer
  - `rosrun rqt_graph rqt_graph`
  - `Observar nodes y topics.`

Twist Message

- http://docs.ros.org/api/geometry_msgs/html/msg/Twist.html
- This message has a **linear** component for the (x,y,z) velocities, and an **angular** component for the angular rate about the (x,y,z) axes.

```
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

Crear un ROS package nuevo

- Creamos un paquete ROS llamado my_stage

```
$cd ~/catkin_ws/src
$ catkin_create_pkg my_stage std_msgs rospy roscpp
```

- El resto de argumentos son los paquetes de los que depende *my_stage*

move_forward.cpp

```
#include "ros/ros.h"
#include "geometry_msgs/Twist.h"

int main(int argc, char **argv)
{
    const double FORWARD_SPEED_MPS = 0.2;

    // Initialize the node
    ros::init(argc, argv, "move_forward");
    ros::NodeHandle node;

    // A publisher for the movement data
    ros::Publisher pub = node.advertise<geometry_msgs::Twist>("cmd_vel", 10);

    // Drive forward at a given speed.  The robot points up the x-axis.
    // The default constructor will set all commands to 0
    geometry_msgs::Twist msg;
    msg.linear.x = FORWARD_SPEED_MPS;

    // Loop at 10Hz, publishing movement commands until we shut down
    ros::Rate rate(10);
    ROS_INFO("Starting to move forward");
    while (ros::ok()) {
        pub.publish(msg);
        rate.sleep();
    }
}
```


```
cmake_minimum_required(VERSION 2.8.3)
project(beginner_tutorials)

## Find catkin macros and libraries
find_package(catkin REQUIRED COMPONENTS roscpp rospy std_msgs genmsg)

## Declare ROS messages and services
# add_message_files(FILES Message1.msg Message2.msg)
# add_service_files(FILES Service1.srv Service2.srv)

## Generate added messages and services
# generate_messages(DEPENDENCIES std_msgs)

## Declare catkin package
catkin_package()

## Specify additional locations of header files
include_directories(${catkin_INCLUDE_DIRS})

## Declare a cpp executable
add_executable(move_forward src/move_forward.cpp)

## Specify libraries to link a library or executable target against
target_link_libraries(move_forward ${catkin_LIBRARIES})
```

Move Forward Demo

- Compilar y ejecutar el nodo

```
cd ~/catkin_ws
catkin_make
```

```
$ cd ~/catkin_ws
$ source ./devel/setup.bash
```

```
rosrun stage_ros stageros `rospack find stage_ros`/world/willow-erratic.world
```

```
$ rosrun my_stage move_forward
```

- El robot se mueve constantemente en el simulador hasta que choca con un obstáculo.

Nodo Stopper

- Haremos que el robot se detenga antes de chocar con un obstáculo.
- Necesitamos leer datos de sensores (subscribirnos al topic que represente información de sensor láser)
- Necesitamos enviar datos de velocidad (publicar el *topic* cmd_vel como el nodo *MoveForward.*
- Crearemos un nodo llamado *stopper.*

Datos de sensores

- La simulación produce datos de sensores (sin ruido)
  - Publicados por **Stage** bajo el *topic* **/base_scan**
- El tipo de mensaje usado por Stage para publicar datos de láser es sensor_msgs/LaserScan
- Puede verse directamente la estructura del mensaje:

```
$rosmsg show sensor_msgs/LaserScan
```

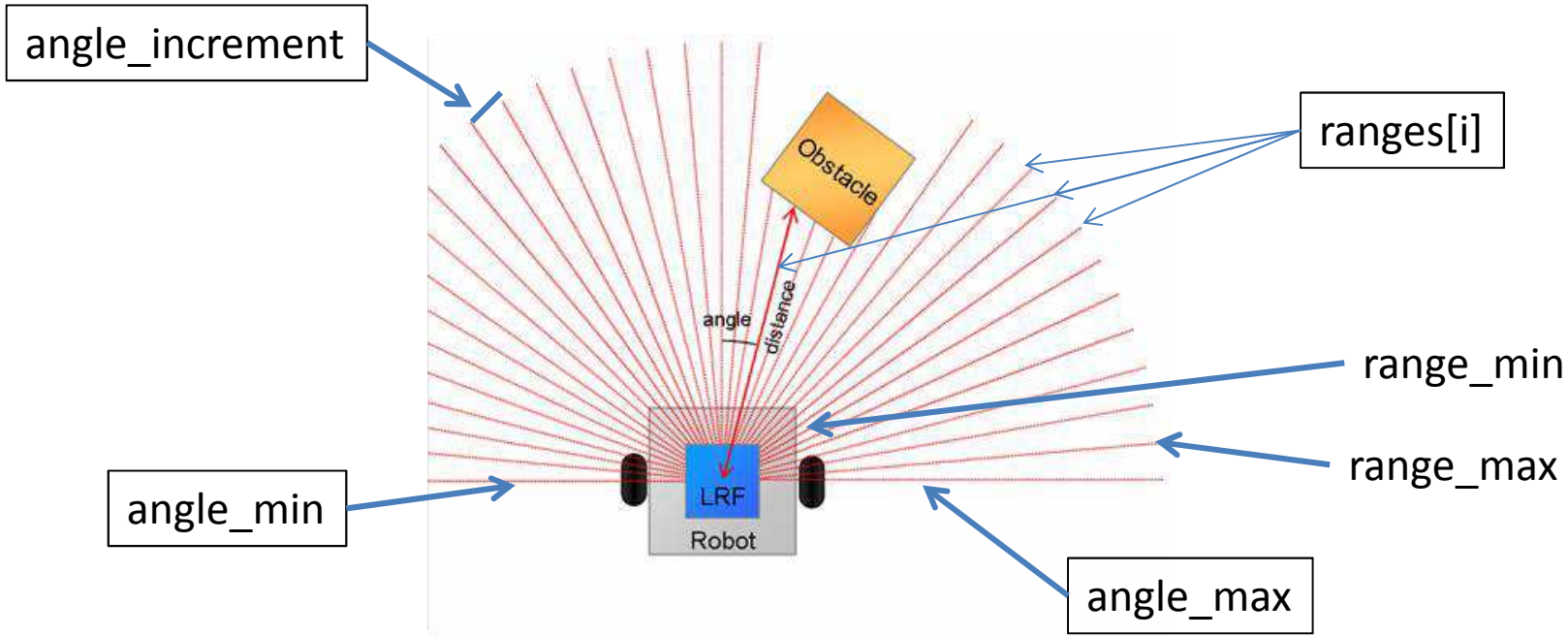
- Stage produce lasers scans *perfectos*
  - Los robots reales y los lásers reales incorporan ruido real que Stage no está simulando.

LaserScan Message

- http://docs.ros.org/api/sensor_msgs/html/msg/LaserScan.html

```
# Single scan from a planar laser range-finder

Header header
# stamp: The acquisition time of the first ray in the scan.
# frame_id: The laser is assumed to spin around the positive Z axis
# (counterclockwise, if Z is up) with the zero angle forward along the x axis

float32 angle_min # start angle of the scan [rad]
float32 angle_max # end angle of the scan [rad]
float32 angle_increment # angular distance between measurements [rad]

float32 time_increment # time between measurements [seconds] - if your scanner
# is moving, this will be used in interpolating position of 3d points
float32 scan_time # time between scans [seconds]

float32 range_min # minimum range value [m]
float32 range_max # maximum range value [m]

float32[] ranges # range data [m] (Note: values < range_min or > range_max should be
discarded)
float32[] intensities # intensity data [device-specific units]. If your
# device does not provide intensities, please leave the array empty.
```

LaserScan Message



LaserScan Message

- Ejemplo de un scan laser del simulador Stage:

```
roiyeho@ubuntu: ~
---
header:
  seq: 1594
  stamp:
    secs: 159
    nsecs: 500000000
  frame_id: base_laser_link
angle_min: -2.35837626457
angle_max: 2.35837626457
angle_increment: 0.00436736317351
time_increment: 0.0
scan_time: 0.0
range_min: 0.0
range_max: 30.0
ranges: [2.427844524383545, 2.42826247215271, 2.4287266731262207, 2.4292376041412354, 2.429795026779175, 2.430398941
040039, 2.4310495853424072, 2.4317471981048584, 2.4324913024902344, 2.4332826137542725, 2.4341206550598145, 2.435005
6648254395, 2.4359381198883057, 2.436917543411255, 2.437944173812866, 2.439018487930298, 2.4401402473449707, 2.44130
94520568848, 2.4425265789031982, 2.443791389465332, 2.4451043605804443, 2.446465253829956, 2.4478745460510254, 2.449
3319988250732, 2.450838088989258, 2.452392816543579, 2.453996419906616, 2.455648899078369, 2.457350492477417, 2.4591
01438522339, 2.460901975631714, 2.462752103805542, 2.4646518230438232, 2.466601848602295, 2.468601942062378, 2.47065
23418426514, 2.4727535247802734, 2.474905490875244, 2.4771084785461426, 2.479362726211548, 2.481668472290039, 2.4840
259552001953, 2.4864354133605957, 2.4888970851898193, 2.4914112091064453, 2.4939777851104736, 2.4965975284576416, 2.
4992706775665283, 2.5019969940185547, 2.504777193069458, 2.5076115131378174, 2.510500192642212, 2.5134434700012207,
2.516441822052002, 2.5194954872131348, 2.5226047039031982, 2.5257697105407715, 2.5289909839630127, 2.53226900100708,
 2.5356037616729736, 2.5389959812164307, 2.542445659637451, 2.5459535121917725, 2.5495197772979736, 2.55314469337463
4, 2.5568289756774902, 2.560572624206543, 2.56437611579895, 2.568240165710449, 2.572165012359619, 2.576151132583618,
 2.5801987648010254, 2.584308624267578, 2.5884809494018555, 2.5927164554595947, 2.597015380859375, 2.601378202438354
5, 2.6058056354522705, 2.610297918319702, 2.6148557662963867, 2.6194796562194824, 2.6241698265075684, 2.628927230834
961, 2.6337523460388184, 2.63478422164917, 2.6436073780059814, 2.6486384868621826, 2.6537396907806396, 3.44798207283
02, 3.4547808170318604, 3.461672306060791, 3.4686577320098877, 3.4757378101348877, 3.4829134941101074, 3.49018549919
1284, 3.4975550174713135, 3.5050225257873535, 3.5125889778137207, 3.5202558040618896, 3.5280232429504395, 3.53589296
3409424, 3.543865442276001, 3.5519418716430664, 3.5601232051849365, 3.568410634994507, 3.5768051147460938, 3.5853075
```

Esquema Stopper

- Clase Stopper
  - Público:
    - Parámetros configurables de la lectura láser
    - Constructor
      - Crea un publisher del topic *cmd_vel* con mensajes tipo *geometry_msgs/Twist*
      - Crea un subscirber del topic *base_scan* con mensajes del tipo *sensor_msgs/LaserScan*
    - Método *startMoving*
      - Implementa un bucle cerrado: mientras puede avanzar (dependiendo de la lectura del sensor) llama a la función *MoveForward.*
  - Privado:
    - Manejador del nodo
    - Publisher y Subscriber
    - Método *MoveForward*
    - Método *callback* para manejar lectura de sensor suscrito.

Stopper.h

```
#include "ros/ros.h"
#include "sensor_msgs/LaserScan.h"

class Stopper
{
public:
    // Tunable parameters
    const static double FORWARD_SPEED_MPS = 0.2;
    const static double MIN_SCAN_ANGLE_RAD = -30.0/180*M_PI;
    const static double MAX_SCAN_ANGLE_RAD = +30.0/180*M_PI;
    const static float MIN_PROXIMITY_RANGE_M = 0.5; // Should be smaller than
sensor_msgs::LaserScan::range_max

    Stopper();
    void startMoving();

private:
    ros::NodeHandle node;
    ros::Publisher commandPub; // Publisher to the simulated robot's velocity command topic
    ros::Subscriber laserSub; // Subscriber to the simulated robot's laser scan topic
    bool keepMoving; // Indicates whether the robot should continue moving

    void moveForward();
    void scanCallback(const sensor_msgs::LaserScan::ConstPtr& scan);
};
```

Stopper.cpp (1)

```
#include "Stopper.h"
#include "geometry_msgs/Twist.h"

Stopper::Stopper()
{
    keepMoving = true;

    // Advertise a new publisher for the simulated robot's velocity command topic
    commandPub = node.advertise<geometry_msgs::Twist>("cmd_vel", 10);

    // Subscribe to the simulated robot's laser scan topic
    laserSub = node.subscribe("base_scan", 1, &Stopper::scanCallback, this);
}

// Send a velocity command
void Stopper::moveForward() {
    geometry_msgs::Twist msg; // The default constructor will set all commands to 0
    msg.linear.x = FORWARD_SPEED_MPS;
    commandPub.publish(msg);
};
```

Stopper.cpp (2)

```
// Process the incoming laser scan message
void Stopper::scanCallback(const sensor_msgs::LaserScan::ConstPtr& scan)
{
    // Find the closest range between the defined minimum and maximum angles
    int minIndex = ceil((MIN_SCAN_ANGLE_RAD - scan->angle_min) / scan->angle_increment);
    int maxIndex = floor((MAX_SCAN_ANGLE_RAD - scan->angle_min) / scan->angle_increment);

    float closestRange = scan->ranges[minIndex];
    for (int currIndex = minIndex + 1; currIndex <= maxIndex; currIndex++) {
        if (scan->ranges[currIndex] < closestRange) {
            closestRange = scan->ranges[currIndex];
        }
    }

    ROS_INFO_STREAM("Closest range: " << closestRange);

    if (closestRange < MIN_PROXIMITY_RANGE_M) {
        ROS_INFO("Stop!");
        keepMoving = false;
    }
}
```

Stopper.cpp (3)

```
void Stopper::startMoving()
{
    ros::Rate rate(10);
    ROS_INFO("Start moving");

    // Keep spinning loop until user presses Ctrl+C or the robot got too close to an
obstacle
    while (ros::ok() && keepMoving) {
        moveForward();
        ros::spinOnce(); // Need to call this function often to allow ROS to process
incoming messages
        rate.sleep();
    }
}
```

run_stopper.cpp

```cpp
#include "Stopper.h"

int main(int argc, char **argv) {
    // Initiate new ROS node named "stopper"
    ros::init(argc, argv, "stopper");

    // Create new stopper object
    Stopper stopper;

    // Start the movement
    stopper.startMoving();

    return 0;
};
```

Stopper Output

Problems Tasks Console Properties Call Graph

<terminated> stopper Configuration [C/C++ Application] /home/roiyeho/catkin_ws/devel/lib/my_stage/stopper (10/25/13, 3:35 AM)

```
[0m[ INFO] [1382661340.576015273, 18109.700000000]: Closest range: 0.760001[0m
[0m[ INFO] [1382661340.667596025, 18109.800000000]: Closest range: 0.740001[0m
[0m[ INFO] [1382661340.768342773, 18109.900000000]: Closest range: 0.720001[0m
[0m[ INFO] [1382661340.867396332, 18110.000000000]: Closest range: 0.680001[0m
[0m[ INFO] [1382661340.966313085, 18110.100000000]: Closest range: 0.680001[0m
[0m[ INFO] [1382661341.067020022, 18110.200000000]: Closest range: 0.660001[0m
[0m[ INFO] [1382661341.172239501, 18110.300000000]: Closest range: 0.640001[0m
[0m[ INFO] [1382661341.269401730, 18110.400000000]: Closest range: 0.620001[0m
[0m[ INFO] [1382661341.371178013, 18110.500000000]: Closest range: 0.600001[0m
[0m[ INFO] [1382661341.465327863, 18110.600000000]: Closest range: 0.580001[0m
[0m[ INFO] [1382661341.565325407, 18110.700000000]: Closest range: 0.540001[0m
[0m[ INFO] [1382661341.668388784, 18110.800000000]: Closest range: 0.520001[0m
[0m[ INFO] [1382661341.771353545, 18110.900000000]: Closest range: 0.500001[0m
[0m[ INFO] [1382661341.868324993, 18111.000000000]: Closest range: 0.500001[0m
[0m[ INFO] [1382661341.967389002, 18111.100000000]: Closest range: 0.480001[0m
[0m[ INFO] [1382661341.967489121, 18111.100000000]: Stop![0m
```

Ejemplo Launch File

- Fichero launch para lanzar el simulado Stage y el nodo *stopper*:

```
<launch>
  <node name="stage" pkg="stage_ros" type="stageros" args="$(find
stage_ros)/world/willow-erratic.world"/>
  <node name="stopper" pkg="my_stage" type="stopper"/>
</launch>
```

- Para ejecutarlo usar:

```
$roslaunch package_name file.launch
```

Trabajo opcional

- Implementar un algoritmo de navegación aleatoria, de manera que el robot nunca se quede atascado, girando hacia la zona en la que tenga mayor espacio libre. Ayudará basarse en *Stopper*.

Mapeo con gmapping y Stage

Construyendo un mapa

- Antes de empezar a usar la navegación con mapa en ROS, 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.

Construyendo un mapa: gmapping

- http://wiki.ros.org/gmapping
- El paquete *gmapping* implementa un nodo ROS que proporciona SLAM basado en láser llamado **slam_gmapping**.

- Entradas: laser scans y odometría

- Salidas: Un mapa de ocupación basado en cuadrículas (**occupancy grid map** OGM)
  - 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

- 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 implementa FastSLAM 2.0: un algoritmo basado en filtrado de partículas
- Detalles en este artículo:

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

gmapping

- Arrancar mapping con el fichero launch_gmapping.launch en PRADO

```
$ roslaunch launch_gmapping.launch
```

  - Esto nos permite ejecutar un robot simulado en stage y simultáneamente un proceso de SLAM para este robot simulado

```
$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

  - Podemos usar la teleoperación para mover el robot en su entorno.

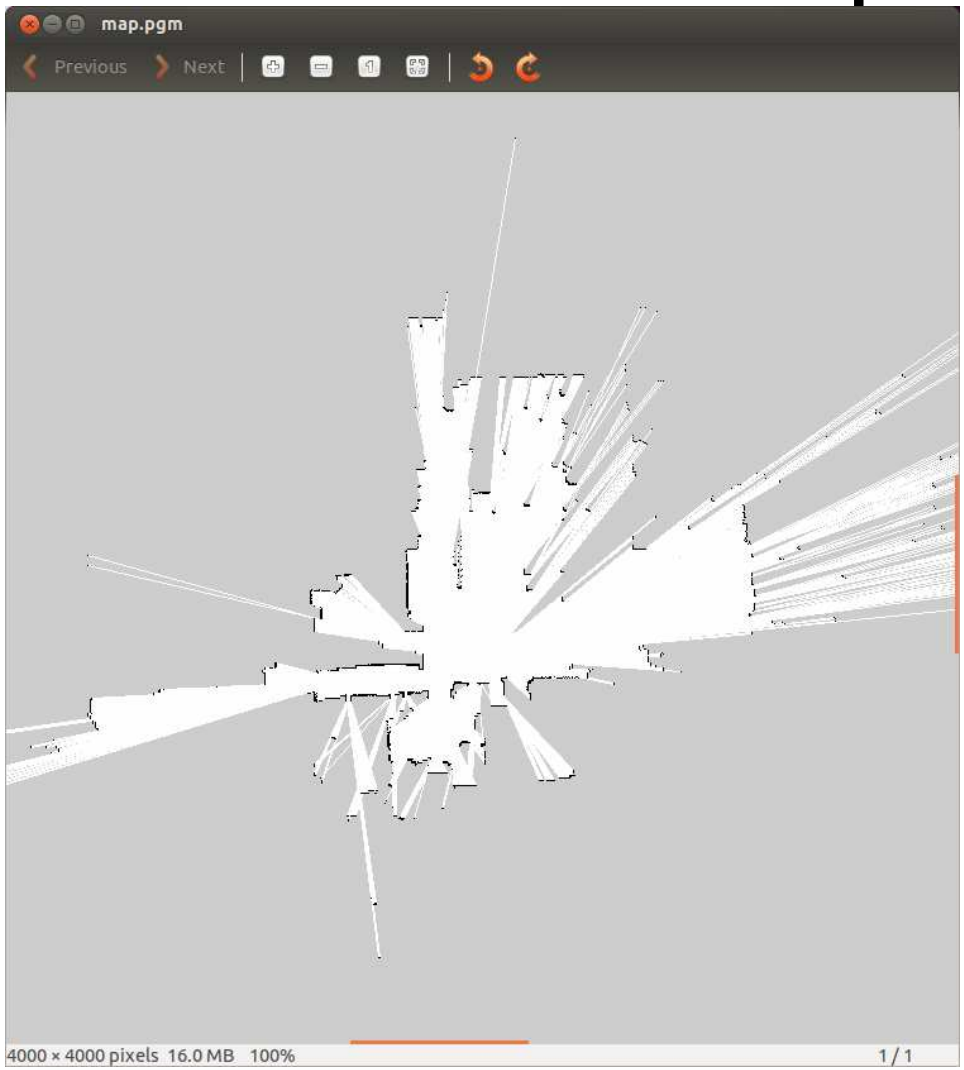
Guardar el mapa con *map_server*

- ***ROS map_server node*** permite guardar mapas generados durante el proceso de SLAM 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:~$
```

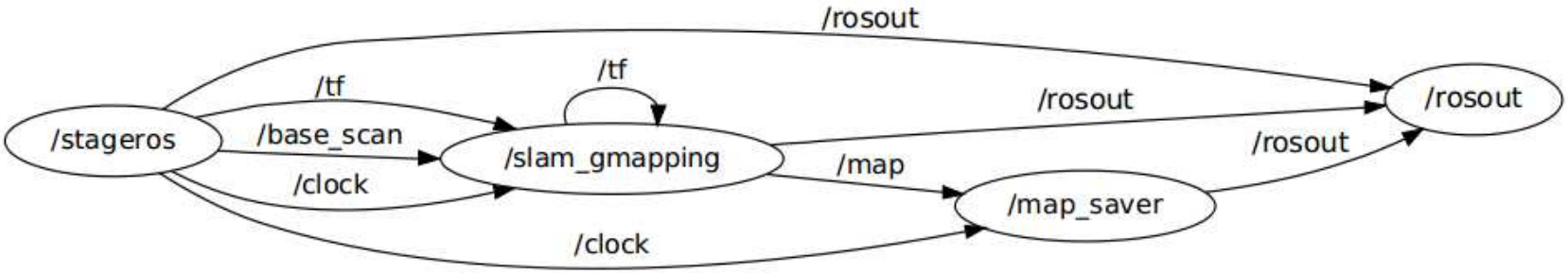
Formato de la imagen

- 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



Visualización con RVIZ

RVIZ: visualización

- **rviz** es una herramienta de visualización 3D de ROS que nos permite ver el mundo desde la perspectiva del robot.
- **rviz** permite ver, el proceso de mapeo.
- Tutoriales y guía de usuario de rviz

  http://wiki.ros.org/rviz

RVIZ: visualización

- Para ejecutar **rviz para visualizar el proceso de mapeo**: `$ roslaunch launch_gmapping+rviz.launch`

```
<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>
```

- Observar (ejecutar rqt_graph):
  - Stage lanza un robot simulado
  - Gmapping lanza el proceso de mapeo
  - Rviz es un nodo que permite visualizar información del ecosistema de ROS.
    - Hay un fichero de configuración de rviz en el paquete ***stage_ros***

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

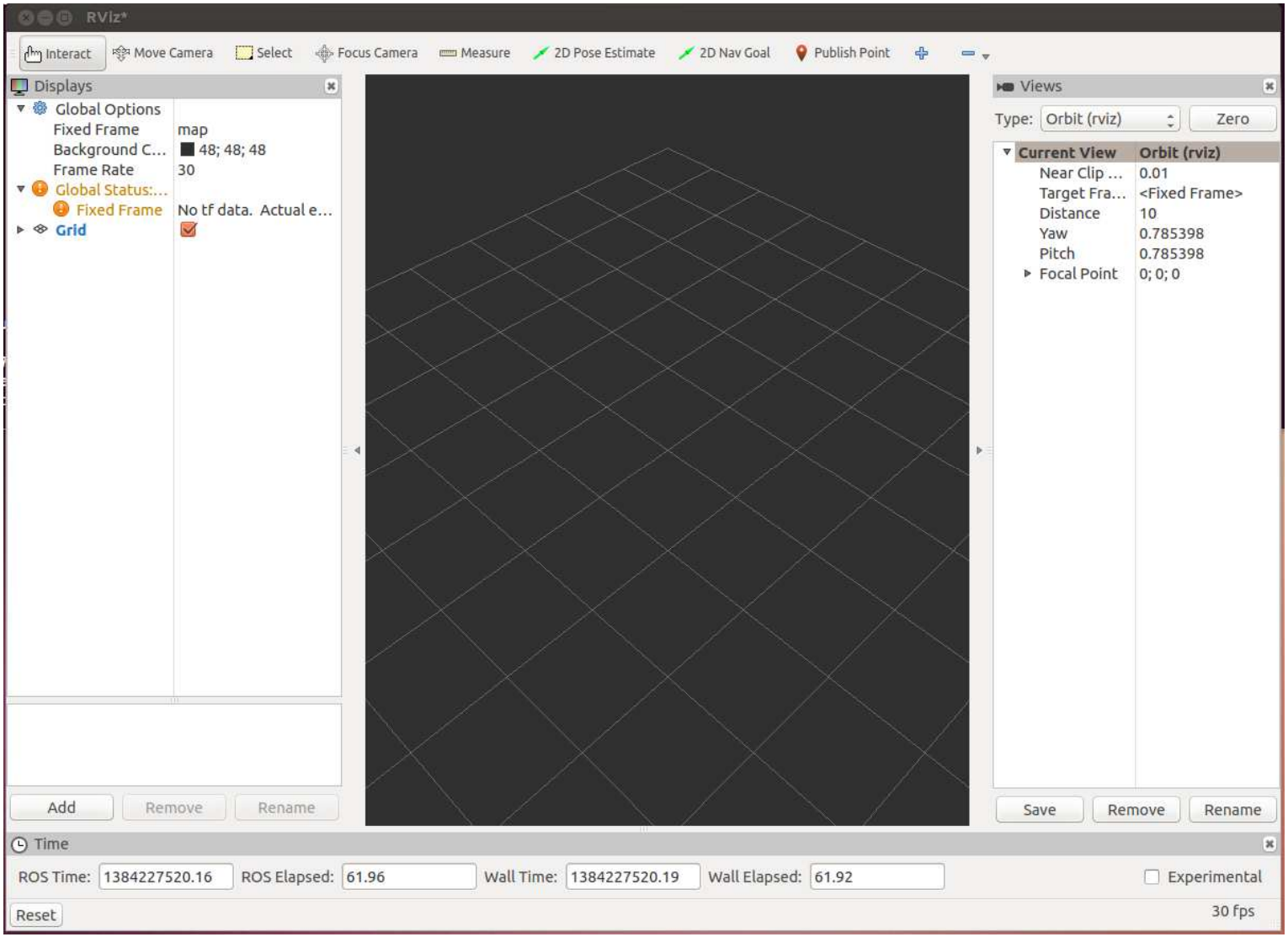
```
$ export LIBGL_ALWAYS_SOFTWARE=1
$ rosrun rviz rviz
```

- 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 general asociados con los topics y/o mensajes que publican los nodos.
- Todos los posibles tipos de display en las siguientes diapositivas, más información en la wiki de Rviz.

rviz Displays

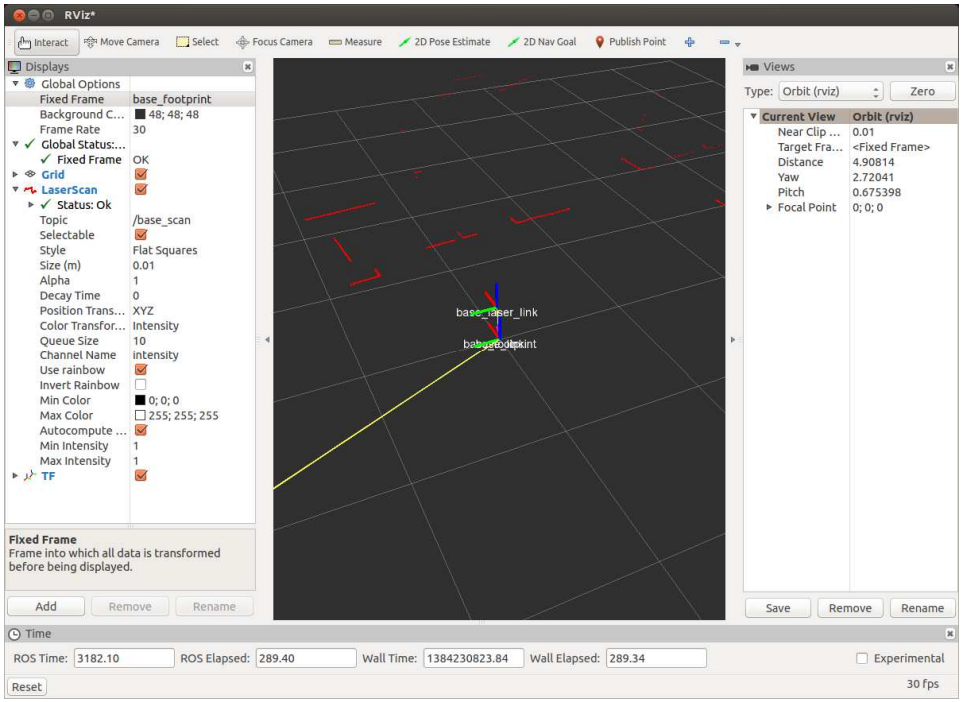
Display name	Description	Messages Used
Axes	Displays a set of Axes	
Effort	Shows the effort being put into each revolute joint of a robot.	sensor_msgs/JointStates
Camera	Creates a new rendering window from the perspective of a camera, and overlays the image on top of it.	sensor_msgs/Image sensor_msgs/CameraInfo
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 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 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.	

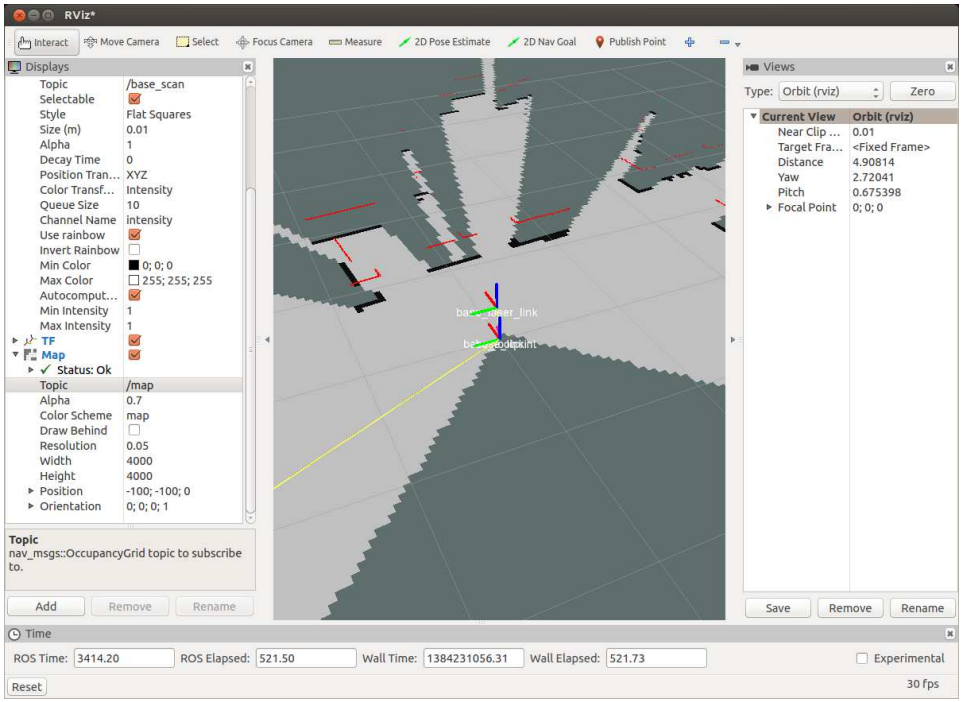
LaserScan Display

- Añadir un display **“LaserScan”** (si no aparece el Display en la lista de displays a la izquierda):
  - 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

- Añadir un display tipo **“Map”,** para visualizar el mapa descubierto por gmapping.
  - Set the **topic** to /map
  - Now you will be able to watch the mapping progress in rviz



Ejercicio propuesto (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?.