EPD 4: Introduction to SLAM

Objectives

- Introduction to localization/map-building employed in the deliberative architectures for navigation.
- Use of a simple SLAM algorithm in ROS to build a map (Gmapping).
- Use of the ROS map_server package.
- Introduction to the ROS base localization algorithm (AMCL).

Assignments

1 Introduction

In the previous EPDs we have been working without any previous knowledge of the environment. We developed our navigation system as a reactive solution based on just sensory input from a 2D laser range finder.

In this EPD we will present the problem of localization in Robotics. We will introduce the concept of geometrical models of the environment (maps) and their relevance in robot navigation (see Figure 1).

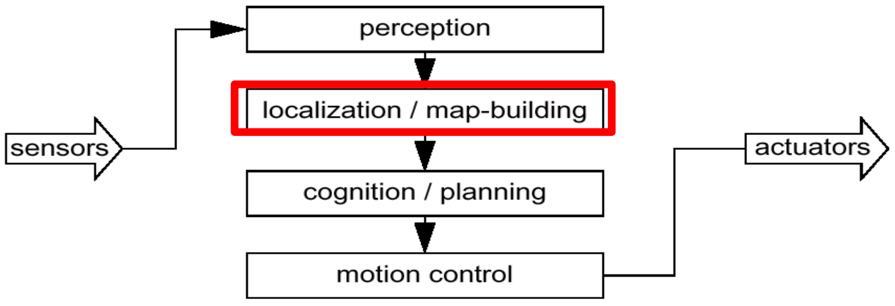


Figure 1. Deliberative architecture for robot navigation

In our current system (EPD3) we relied on the wheel odometry of our Turtlebot to estimate the robot position relative to a starting location at each time step (localization). However, in real robot applications, the computation of robot odometry leads to small position errors due wheels drifts and the integration of velocity measurements over time. These errors are accumulated so that the precision of the localization is decreasing over time (see Figure 2).

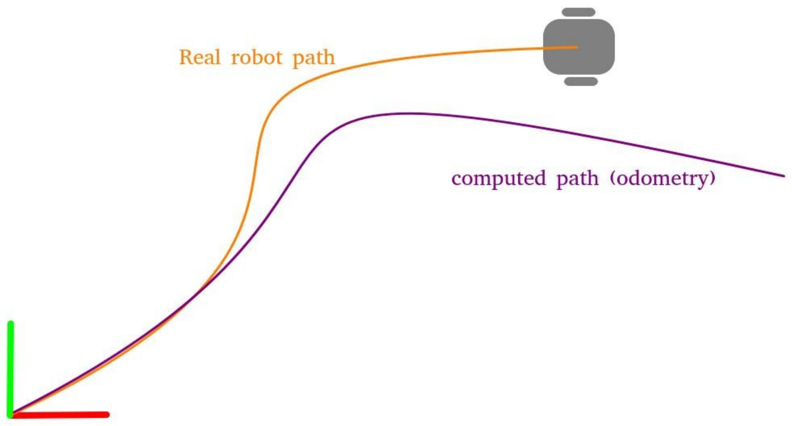


Figure 2. Example of odometry error.

Although there exist more advanced methods to compute the robot odometry (visual odometry for instance), there still exist some drawbacks and complex situations that affect negatively to the odometry computation. Beyond that, the robot localization is still a complex problem to solve in Robotics.

One way to deal with the odometry problems is to build a "memory" of the environment based on our sensing input. This way, we can "store" the sensed information (map) so that the robot may use this data to better compute its position along with the current sensor input. The family of techniques to build a model of the environment, and to localize the robot in it, is called Simultaneous Localization and Mapping (SLAM).

In the EPD5, we will also see the relevance of the environment maps in path planning.

2 Building a geometrical map with ROS Gmapping

In order to be able to plan a collision free path that the robot should follow from a starting point to a goal point, we should model the environment of the robot. In this way, we can distinguish the safe (without obstacles) positions of the robot from the unsafe ones. Besides we can overcome the problem of local minima to reach a goal. To this end, we will model the environment as a 2D grid of occupied and free cells.

We are going to use a SLAM method based on 2D laser called Slam Gmapping. Further information at: http://wiki.ros.org/gmapping. In order to build a 2D occupancy grid (like a building floorplan), this method will use:

- the robot poses provided by the odometry data of our robot (through the coordinates and relations in the TF tree).
- the horizontal laser data with the distance ranges to the obstacles detected.

To learn this tool and to test it, we will use another ROS package. **From the epd4 directory in the Aula Virtual you can dowload the zip file localization.zip**. It contains a ros package called localization that you must download, unzip and put it in the source folder of your ROS workspace. Then compile your workspace

```
$ cd /home/rva_container/rva_exchange/rva_ws
$ catkin_make
```

To try Gmapping, we can launch:

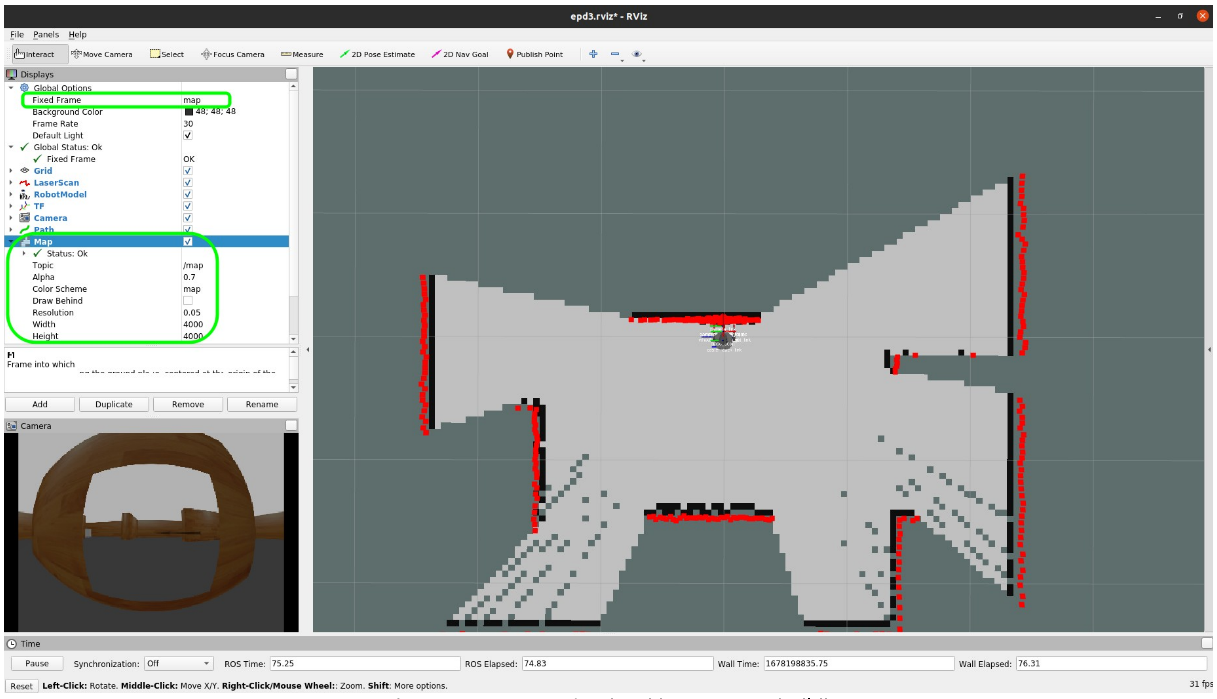
```
$ roslaunch localization gmapping.launch
```

This launch file will run:

- A Gazebo simulation with our Turtlebot robot
- Rviz
- The Gmapping node
- The teloperation node

Once everything is running, we can control our Turtlebot from our keyboard. What we must do is to walk our robot around the whole environment. This way, the algorithm can take data from all the available space and to build the map properly.

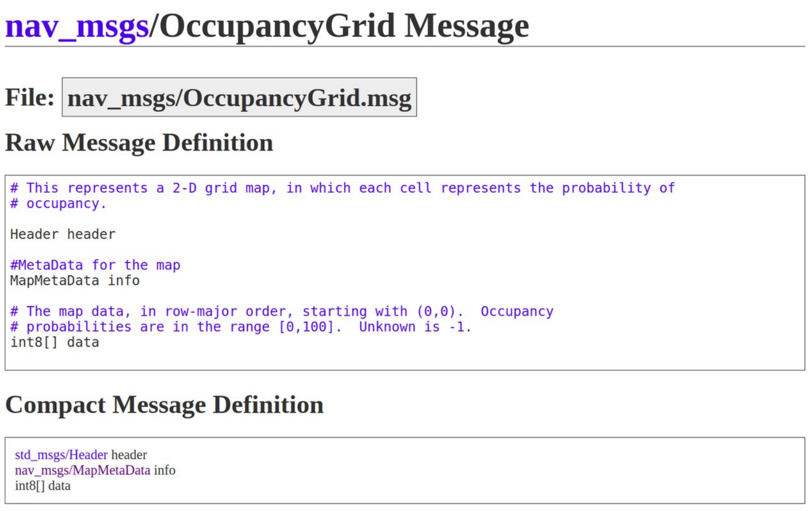
We can check the progress of the map creation in Rviz. First, we must indicate the frame "map" as a fixed frame, and adding a new display of type "map" with the name topic "/map" (see Figure 3). The light gray color indicates the free space detected by the mapping system. The darker gray color indicates"unknown" areas. Finally, the obstacles detected are shown in black color.



```
Displays
  Global Options
    Fixed Frame          map
    Background Color     48; 48; 48
    Frame Rate           30
    Default Light        ✓
  Global Status: Ok
    Fixed Frame          OK
  Grid                   ✓
  LaserScan              ✓
  RobotModel             ✓
  TF                     ✓
  Camera                 ✓
  Path                   ✓
  Map                    ✓
    Status: Ok
    Topic                /map
    Alpha                0.7
    Color Scheme         map
    Draw Behind
    Resolution           0.05
    Width                4000
    Height               4000

ROS Time: 75.25   ROS Elapsed: 74.83   Wall Time: 1678198835.75   Wall Elapsed: 76.31
```

Figure 3. Capture of Rviz with a 2D map building

What we are visualizing in Rviz is the information that the Gmapping algorithm is publishing in ROS by using the following messages:



nav_msgs/OccupancyGrid Message

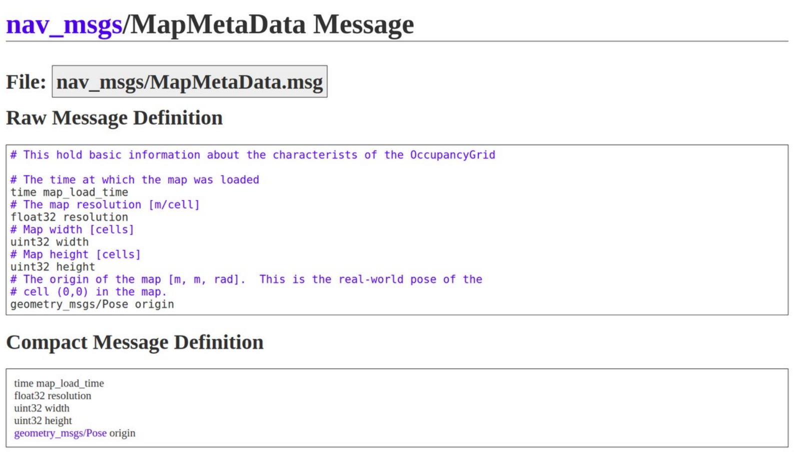
File: nav_msgs/OccupancyGrid.msg

Raw Message Definition

```
# This represents a 2-D grid map, in which each cell represents the probability of
# occupancy.

Header header

#MetaData for the map
MapMetaData info

# The map data, in row-major order, starting with (0,0).  Occupancy
# probabilities are in the range [0,100].  Unknown is -1.
int8[] data
```

Compact Message Definition

```
std_msgs/Header header
nav_msgs/MapMetaData info
int8[] data
```



nav_msgs/MapMetaData Message

File: nav_msgs/MapMetaData.msg

Raw Message Definition

```
# This hold basic information about the characterists of the OccupancyGrid

# The time at which the map was loaded
time map_load_time
# The map resolution [m/cell]
float32 resolution
# Map width [cells]
uint32 width
# Map height [cells]
uint32 height
# The origin of the map [m, m, rad].  This is the real-world pose of the
# cell (0,0) in the map.
geometry_msgs/Pose origin
```

Compact Message Definition

```
time map_load_time
float32 resolution
uint32 width
uint32 height
geometry_msgs/Pose origin
```

At the same time, the algorithm creates a new coordinates system (map) which is added to our TF tree (see Figure 4):

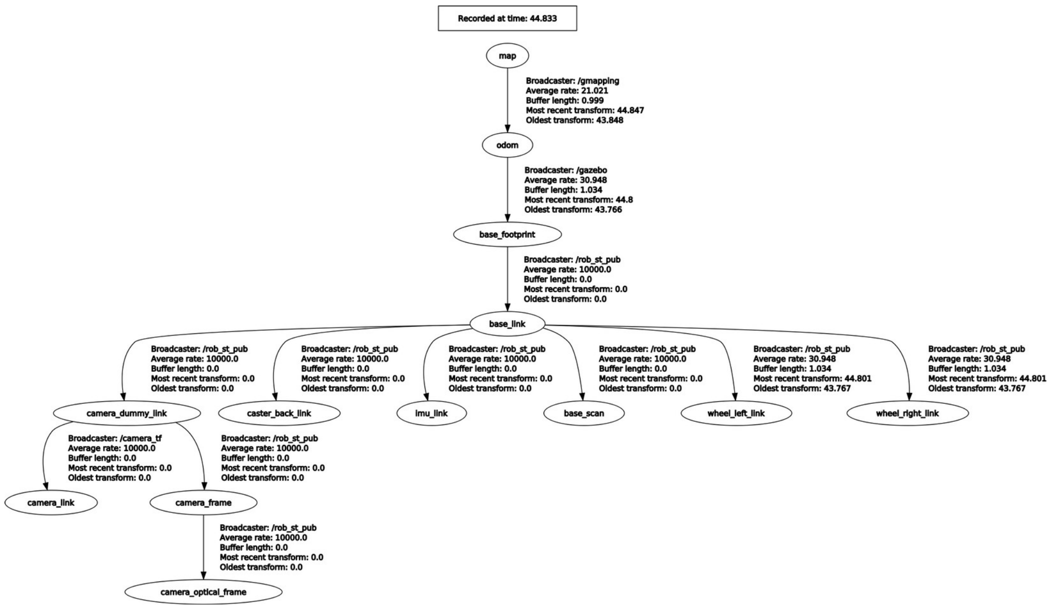


Figure 4. TF tree

Therefore, while we move the robot around, the Gmapping algorithm is building a model of the environment and it is also localizing the robot in that scenario. Once we have explored the whole space, we can proceed to store the map. To do so, we use the node *map_saver*. This node is part of the package *map_server* which will be explained in the following section.

To run the map_saver, we can move to the directory in which we want to store the map (*rva_ws/src/localization/maps*), and execute:

```
$ rosrun map_server map_saver -f map
```

With the *-f* option, we indicate the name that we want to use for our map.

This node will generate two files:

1. The map image that represents occupied, unknown and free cells (usually in pgm format).
2. Each map should be accompanied by a .yaml file, which describes the map meta-data. The YAML format of the file is as follows:

```
image: map.pgm
resolution: 0.1
origin: [0.0, 0.0, 0.0]
occupied_thresh: 0.65
free_thresh: 0.196
negate: 0
```

   Every field of the YAML has the following meaning:

   - **image:** Path to the image file containing the occupancy data; can be absolute, or relative to the location of the YAML file
   - **resolution:** Resolution of the map, in meters / pixel
   - **origin:** The 2-D pose of the lower-left pixel in the map, as (x, y, yaw), with yaw as counterclockwise rotation (yaw=0 means no rotation)
   - **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.
   - **negate:** Whether the free/occupied semantics should be reversed (i.e. greater converts to less and viceversa in the two former definitions)

3 Use of the geometrical model: the map_server package

The map server package loads a geometric grid map that represents the free and occupied 2D cells in the environment. Therefore, the map server will read our map information (map image and map metadata) and it will publish this information into ROS. To do that, it will use the same ROS messages than Gmapping (*nav_msgs/OccupancyGrid*). Further info: http://wiki.ros.org/map_server

We will use this information for localization, as we will see in the next section, and for path planning, as we will explain in the following EPD5.

4 Introduction to localization: Adaptive Monte Carlo Localization (AMCL)

There are plenty of different SLAM and/or localization techniques (we just saw SLAM Gmapping for instance). It is out of the scope of this course to go into a throughout explanation of SLAM. We will simply introduce here the basic concepts behind a well-known method available in ROS, AMCL. We will also show how to use it in our navigation system.

AMCL is based on a particle filter. The very basic idea behind particle filters is to spread particles in the environment. Each particle represents a possible pose of the robot in the environment (a pose of the laser sensor to be more specific). Then, the 2D laser data is represented in that pose on the map, and a correspondence between the laser data and the map's obstacles is computed. The particle (sensor pose) that better fits the map obstacles is chosen as the most likely robot pose in the map. More information can be found in: http://wiki.ros.org/amcl and http://www.probabilistic-robotics.org/

Checking the AMCL documentation we see that the algorithm needs the following input information:

- 2D laser data (by default in the topic */scan*).
- The relations and poses of the different coordinates systems through the topic */tf*.
- A map.
- An initial pose (0 by default).

The AMCL node will provide as output the localization of the robot in the map. That is the estimation of the map coordinate frame regarding the odometry and robot frames (See Figure 5). A topic /amcl_pose with the pose of the robot in the map will be also published.

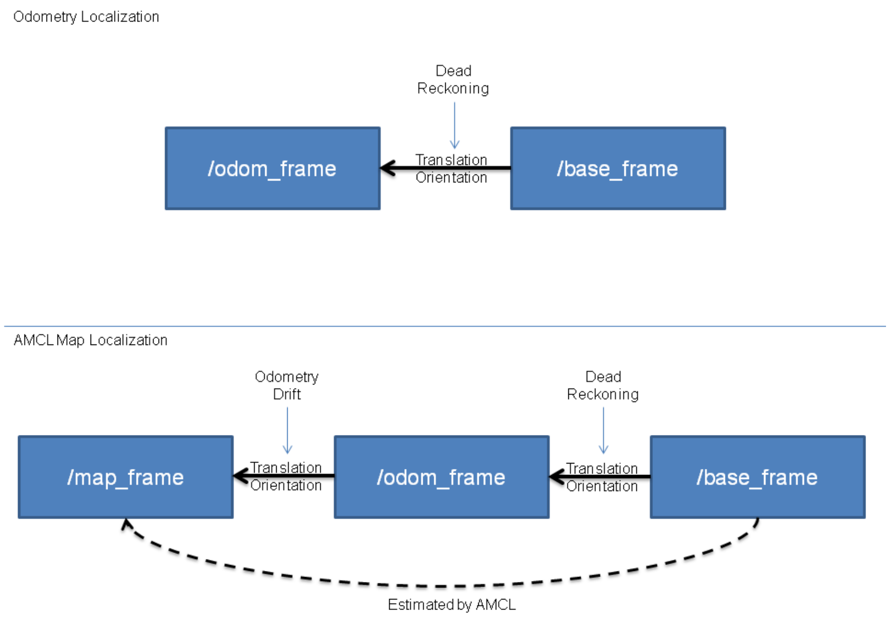


Figure 5. AMCL localization through coordinate frames.

To test the localization system you can firstly examine and secondly execute the launch file amcl.launch:

```
$ roslaunch localization amcl.launch
```

Using the keyboard you can move the robot in the environment while the localization system (AMCL) will try to keep the best robot pose in the map frame.