Mobile Robot Localization and Navigation

Ivan Marković Matko Orsag Damjan Miklić

Automation and Robotics
Robot Programming and Simulation

UNIVERSITY OF ZAGREB
Faculty of Electrical Engineering and Computing

Before we begin

- Be sure to install the following packages for the class:

```
$ sudo apt-get update
$ sudo apt-get install ros-$ROS_DISTRO-stage
$ sudo apt-get install  \
  ros-$ROS_DISTRO-teleop-twist-keyboard
$ sudo apt-get install ros-$ROS_DISTRO-navigation
```

So you want an autonomous robot?

- A robot is in a completely unknown environment for the first time. What to do next?
- One of the most fundamental problems in mobile robotics is map building of an unknown environment
- Ends up being a prerequisite for autonomous mobile robots (without making structural changes in the environment)
- This problem is known as Simultaneous Localization and Mapping (SLAM)

So you want an autonomous robot?

- SLAM is difficult, and what if you are not a SLAM expert?
- Say thank you to researchers who open-sourced their solution
- At the moment popular SLAM implementations in are GMapping/OpenSLAM, Hector SLAM, Cartographer, and MRPT
- Let's illustrate the SLAM problem on the Board

Simulator

- Launch the simulator with a simple map and a robot inside

```
$ rosrun stage_ros stageros simple_pzros.world
```

- Familiarize yourself with the environment and the robot sensor
- Analyze the topics published by the simulator (`rostopic list, info, ...`)

Controling the robot

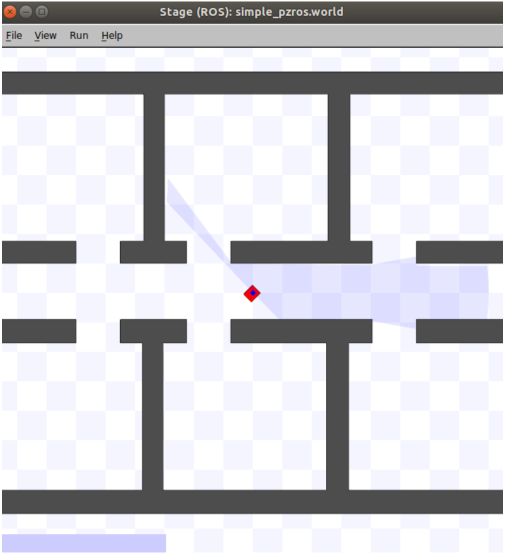
- One of the listed topics is `/cmd_vel`

```
$ rostopic info /cmd\_vel
Type: geometry_msgs/Twist
...
```

- We will send messages to this topic in order to control the robot
- Analyze the message type. What kind of information does the message contain? (hint: `rosmsg show`)
- The package `teleop_twist_keyboard` offers the option of controlling the robot via the keyboard

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

Stage simulator with a robot equpped with a laser



Localization

- We assume that we have the map the environment. Do we have the map published?
- With the map we can start making the robot do useful tasks in the environment
- But we also need an algorithm which will tell us where the robot is located based on the map and received measurements
- For this purpose we will use adaptive Monte Carlo localization (AMCL)
- Let's illustrate how AMCL works on the Board

AMCL

- Let's look at the AMCL documetation
- It requires a laser scan, initial pose, the map, some transformations, and has many parameters to set up (most we can leave default, but some require attention)
- Laser scan and odometry are already being published by the simulator
- Map is just like any other topic AMCL will subsribe to, we will run our true map:

```
$ rosrun map_server map_server simple_rooms.yaml
```

- What about the transforms?

AMCL

- We will not go at this point in the details of the `tf` package
- Launch `rviz`

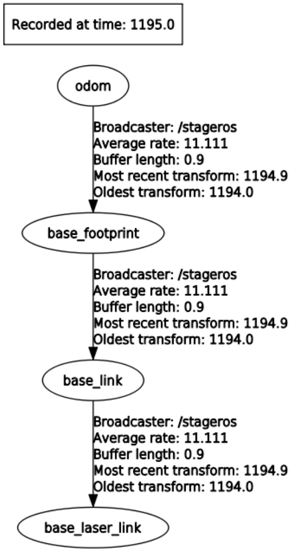
```
$ rosrun rviz rviz
```

- Add LaserScan (/base_scan topic), TF, and map (/map topic) display types
- Move the robot and see how TF coo. sytems position chage
- Now lets look at the available transforms:

```
$ rosrun rqt_tf_tree rqt_tf_tree
```

- Are all the required transforms available?

TF tree



Recorded at time: 1195.0
odom
Broadcaster: /stageros
Average rate: 11.111
Buffer length: 0.9
Most recent transform: 1194.9
Oldest transform: 1194.0
base_footprint
Broadcaster: /stageros
Average rate: 11.111
Buffer length: 0.9
Most recent transform: 1194.9
Oldest transform: 1194.0
base_link
Broadcaster: /stageros
Average rate: 11.111
Buffer length: 0.9
Most recent transform: 1194.9
Oldest transform: 1194.0
base_laser_link

AMCL

- We are missing the transform between the odometry and the map

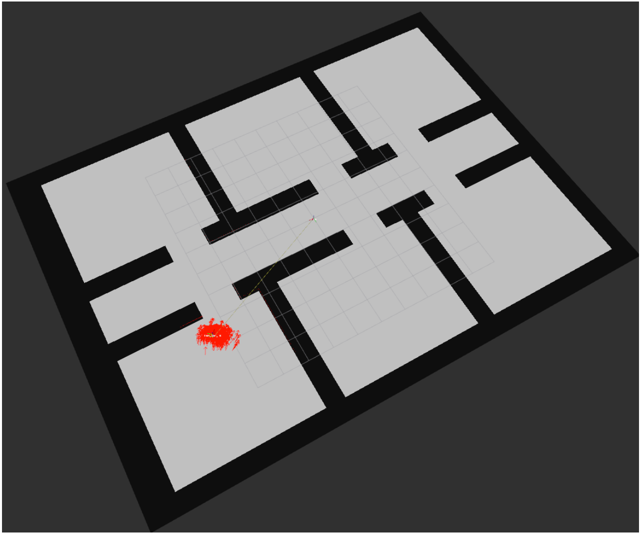
```
$ rosrun tf static_transform_publisher \
0 0 0 0 0 0 map odom 100
```

- We are ready to run the AMCL for which we will need to set the frames

```
$ rosrun amcl amcl _use_sim_time:=true \
_use_map_topic:=true scan:=/base_scan  \
_odom_frame_id:=/odom _base_frame_id:=/base_link \
_global_frame_id:=/map _tf_broadcast:=false
```

- In the RViz add `PoseArray` to visualize `particlecloud` topic and click `2D Pose Estimate` and then click on the map to set up the initial position
- Drive the robot around a bit and see how the algorithm converges to the location

AMCL



Navigation

- Now that you have the map and a way localize your robot in the map we can start with robot navigation
- In essence, the purpose of navigation is to move the robot from point A to point B
- But this stems a plethora of problems: finding the optimal global path, acting locally, avoiding static and dynamic obstacles, recalculating paths due to changes in the environment, controlling the robot etc.
- We will use the Move Base package
- Let's illustrate how MR navigation works on the Board

Starting multiple ROS nodes

- As our applications get more complex, we will need to start multiple nodes, set many parameters etc.
- Solution is to use `roslaunch`, a tool for starting multiple nodes and setting multiple parameters
- Many packages come with already set-up "launch files"

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

- `roslaunch` uses XML files (e.g. `file.launch`) that describe the nodes that should be run, parameters that should be set, and other attributes
- Location of the launch file is resolved by the argument `package_name`; however, this is not necessary

roslaunch XML syntax (simple example)

```
<launch>
  <node name="shove" pkg="turtlecontrol" type="push.py" />
  <remap from="turtle1/cmd\_vel" to="cmd\_vel"/>
</launch>
```

- Starts a node `shove` using the `push.py` executable from the `turtlecontrol` package and call it `shove`
- In the example the simulator is subscribed to `turtle1/cmd_vel`, but our node publishes `cmd_vel`, so we need to remap this
- Visit this link for more info

Useful links

- http://www.ros.org/wiki/gmapping
- http://openslam.org/
- http://www.ros.org/wiki/amcl
- http://www.ros.org/wiki/move_base