Research Track I – Summary and Assignment

Carmine Tommaso Recchiuto, Phd

Packages updated

- ✓ The proposed exercise has been implemented by using the packages robot_description and slam_gmapping. If you have cloned the two repositories, robot_description (git clone https://github.com/CarmineD8/robot_description) and slam_gmapping (git clone https://github.com/CarmineD8/slam_gmapping), you can just run “git pull” with the terminal in the path corresponding to the two packages. Otherwise, just download them again.

- ✓ Check the ROS version. With git, you can just run “git checkout noetic” to switch to the noetic branch of the repository

- ✓ Also, some parameters for gmapping, amcl and move_base have been tuned. Please consider the updated parameters.

Exercise (previous week)

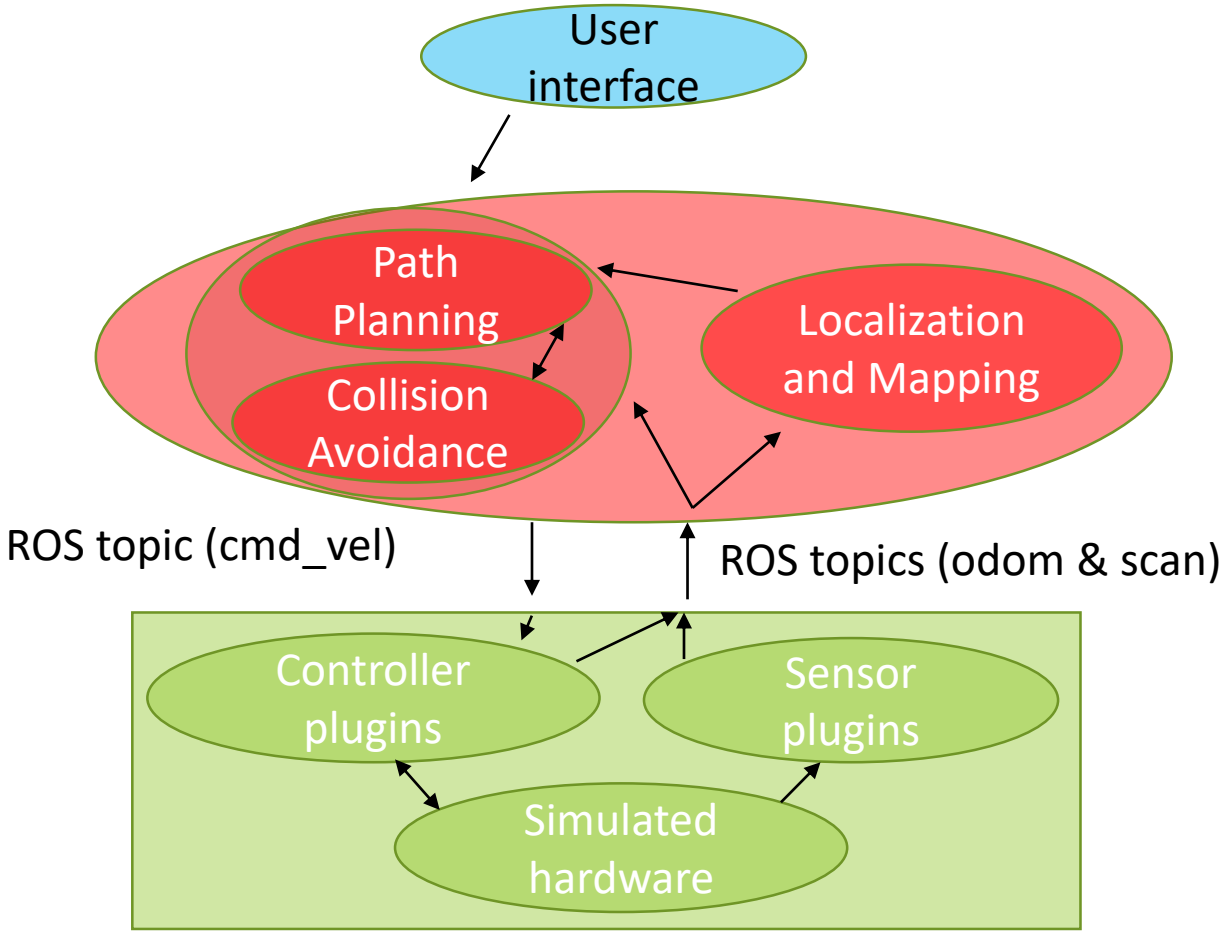
- Using the gmapping algorithm, try to implement the exercise of the previous week by using the *real_odom_robot* and relying on the position of the robot corrected by the mapping algorithm

-> The tf_echo node has been modified by adding a publisher of the *corrected* robot position. Since it has been done with a geometry_msgs::Twist, also bug.py and go_to_point_service.py have been accordingly modified (they have now a callback on a Twist message)

-> The final launch file include the sim_w1.launch (which brings up the simulation corresponding to the robot with the unprecise odometry, together with the gmapping algorithm) and all the necessary scripts file (bug_m2.py, go_to_point_service_m2.py, wall_follow_service_m2.py and tf_echo, for fixing the robot position).

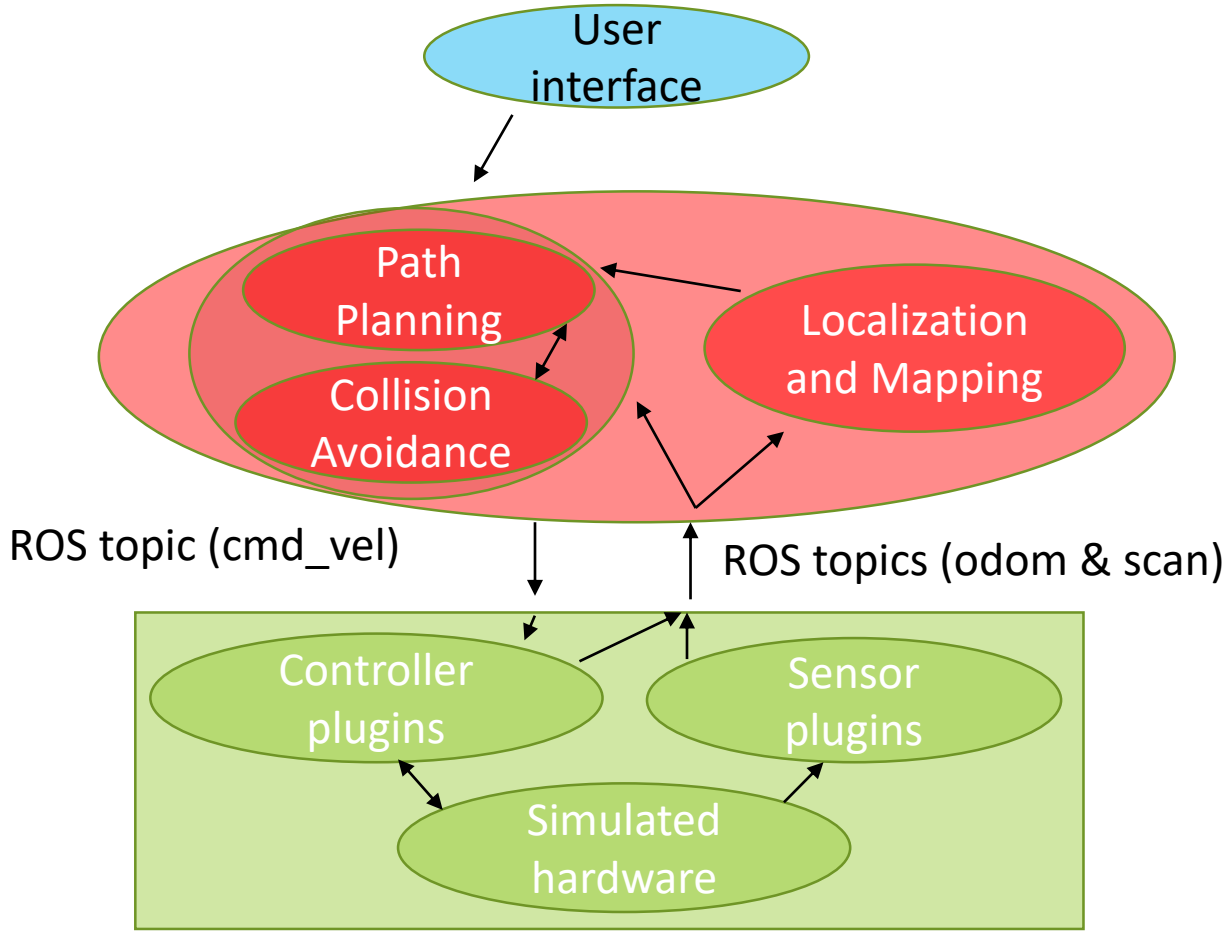
-> On another terminal, we run rosrun robot_description user_interface.py. We run it on another terminal just so that we can see the user interface more easily.

Summary



- In the last two weeks we have analyzed how to implement all typical modules of a mobile robot in simulation
- At first we have seen how to use Gazebo and Rviz for launch a robotic simulation, also developing a simple algorithm for moving a robot in the environment
- Then we have underlined the limitation of the previous approach, and added some modules for localization, mapping, and path planning.
- In particular, we have used the packages gmapping, amcl and move_base

Summary



- Es:

  - ✓ roslaunch gmapping display2.launch
  - ✓ rosrun map_server map_server $(rospack find gmapping)/map.yaml
  - ✓ roslaunch gmapping amcl.launch
  - ✓ roslaunch gmapping move_base.launch

Or

  - ✓ roslaunch gmapping sim_w2.launch
  - ✓ roslaunch gmapping move_base.launch

In the first case, we are using a pregiven map (amcl), in the second case, we adopt a SLAM approach.

Assignment

- On these basis, the final assignment is the following:

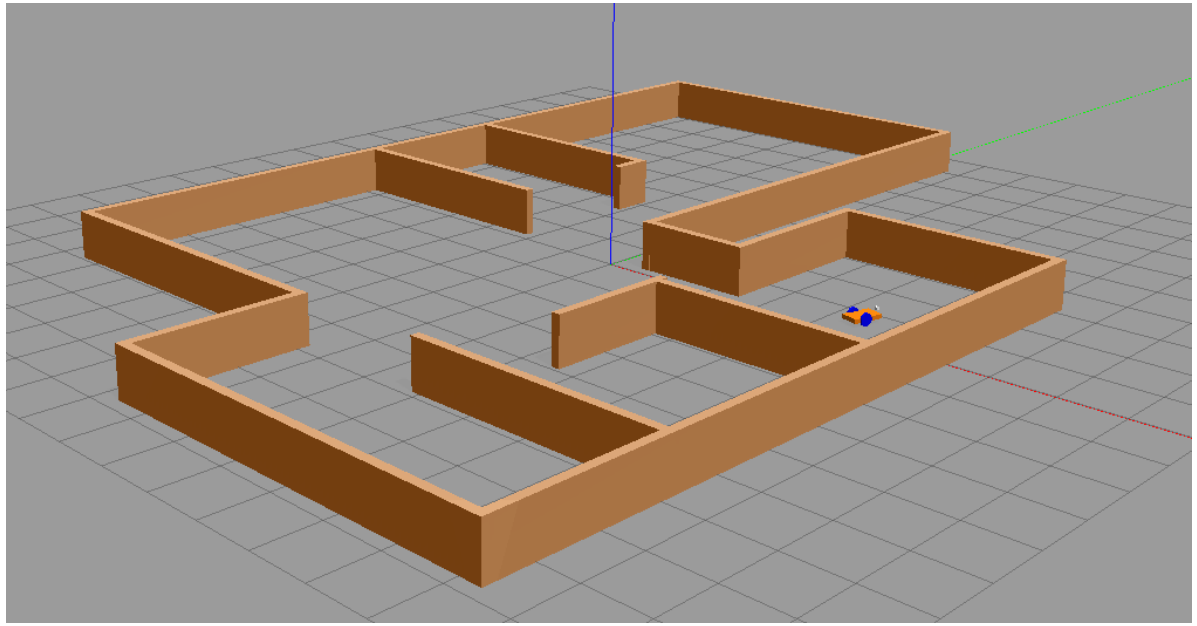
  Development of a software architecture for the control of a mobile robot.

  What will you need?

  - The package final_assignment
  - The slam_gmapping package
  - The ros navigation stack

  Apt-get install ros-<your_ros_distro>-navigation

  Please check your ros distribution and use the corresponding branch



Assignment

Develop a software architecture for the control of the robot in the environment. The software will rely on the move_base and gmapping packages for localizing the robot and plan the motion.

The architecture should be able to get the user request, and let the robot execute one of the following behaviors (depending on the user's input):

- 1) move randomly in the environment, by choosing 1 out of 6 possible target positions: **[(-4,-3);(-4,2);(-4,7);(5,-7);(5,-3);(5,1)],** implementing a random position service as in the assignment 1
- 2) directly ask the user for the next target position (checking that the position is one of the possible six) and reach it
- 3) start following the external walls
- 4) stop in the last position
- 5) (optional) change the planning algorithm to dijkstra (move_base) to the bug0

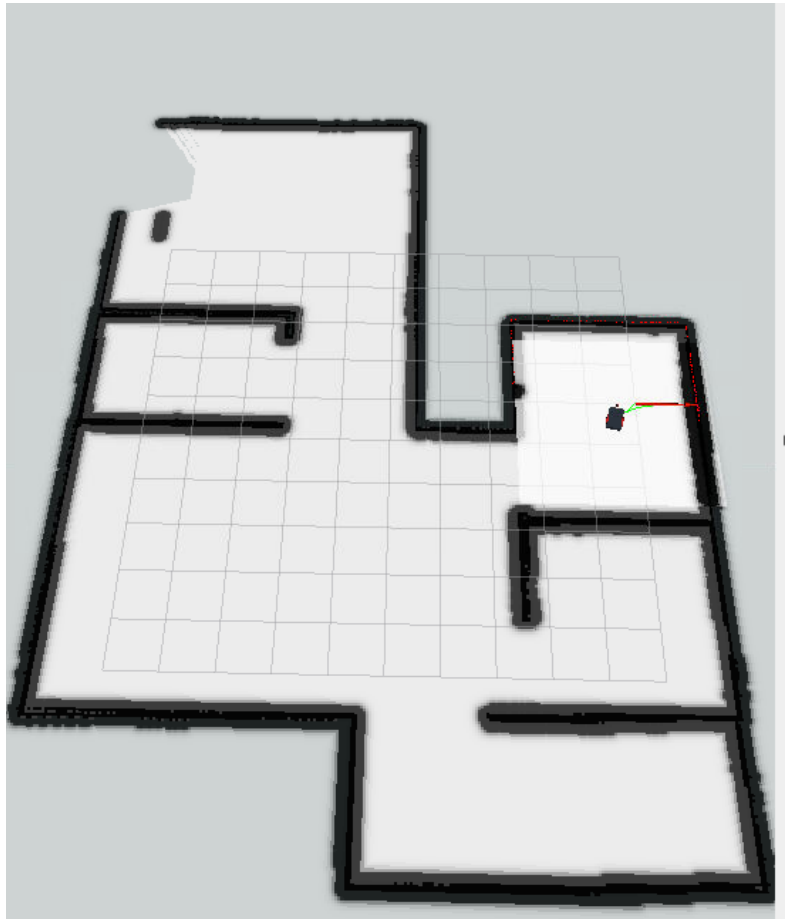
If the robot is in state 1, or 2, the system should wait until the robot reaches the position in order to switch to the state 3 and 4 or (if implemented) to change the planning algorithm.

Assignment

Regarding the user interface, it should:

- acquire the user command (when possible)
- display some information (e.g. the robot position, the distance from the obstacle, the robot state, …)

Of course you a command line interface will be perfectly fine!



Assignment

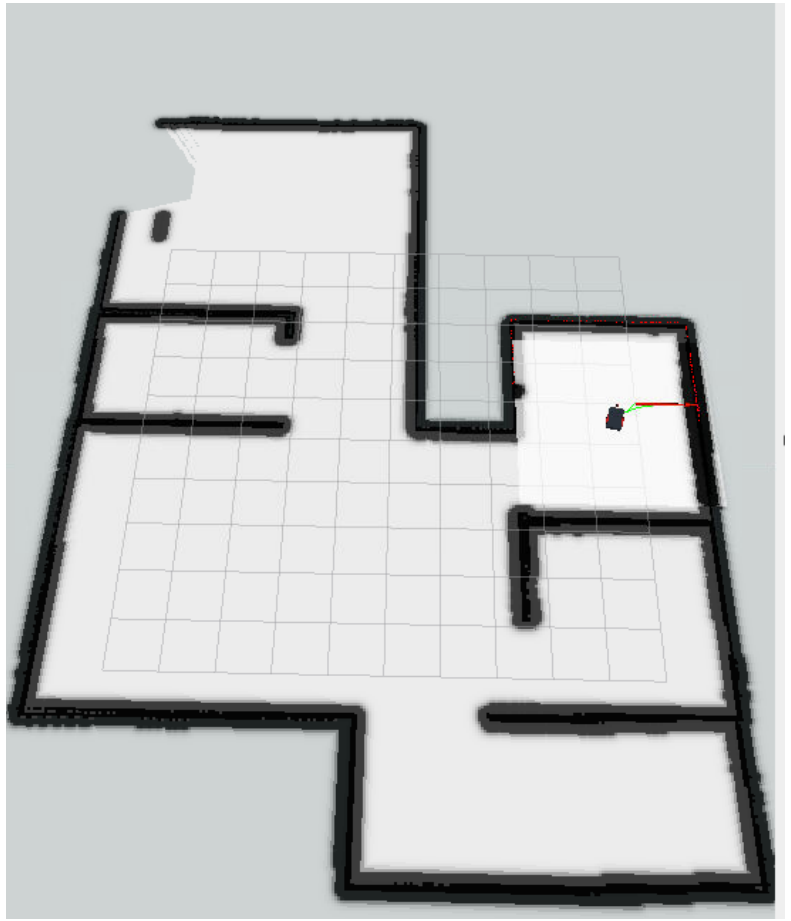
Regarding 1) and 2), the move_base pakage requires goal to be sent to the topic move_base/goal, by sending a message of type move_base_msgs/MoveBaseActionGoal

Please remember to:

set the field goal.target_pose.header.frame_id to *map*
set the field goal.target_pose.pose.orientation.w to 1

the target x and y position coordinates should be set in:

goal.target_pose.pose.position.x
goal.target_pose.pose.position.y

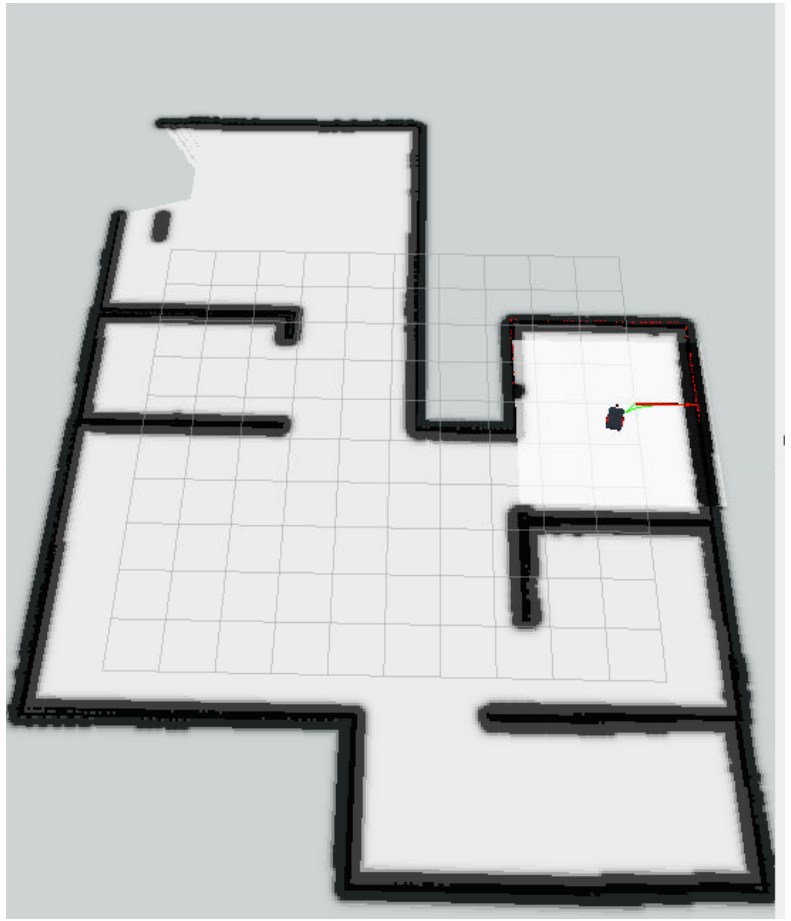


Assignment

Regarding 3) and 5), you can reuse scripts developed in the course. They are also in the scripts folder of the final_assignment package.

However, regarding 5) please notice that, depending on the robot's position, not all targets are reachable with the bug0 algorithm! A strategy for dealing with unreachable targets should be then implemented (e.g. setting a timeout)

There are no particular requirements on the final architecture: the only requirement from the point of view of the software architecture is to use a position server for the random position as in the assignment I.



Requirements – how to submit the assignment

- The link of a github repository containing the developed ROS package should be given;

- The repo should have a README.md file with:
  - Description of the content of the package (nodes, custom messages or services (if any)).
  - Computational graph of the system (how do nodes communicate? You may use rqt_graph).
  - Instructions about how to run the code.
  - Roslaunch (if needed, even more than one roslaunch files) should be used to start the simulation and all necessary nodes.

  The README will also constitute the report of the assignment, so please add:
  - 1 or 2 paragraph about the robot behaviors implemented
  - 1 or 2 paragraphs describing the software architecture and the architectural choices made
  - 1 or 2 paragraphs describing system's limitations and possible improvements

- Functions and source files should be documented (optional: you can create a docs folder with DoxyGen documentation).

Deadline

The deadline of the assignment (and also of the previous one) is 5 days before your oral exam: (so for the JEMARO students it will be 10 January 2021).