9.1 Playing with ROS

January 19, 2021

9.1 Playing with ROS In this chapter we are going to play a bit with different robotic techniques in the context of the Robot Operating System (ROS) ecosystem.

For that, we are going to use the Stage simulator:

> Stage is a robot simulator, it provides a virtual world populated by mobile robots and sensors, along with various objects for the robots to sense and manipulate. ROS provides a node, called stage_ros, which wraps the core functionality of Stage, like the utilization of 2D laser scanners or robotic bases yielding odometry information.

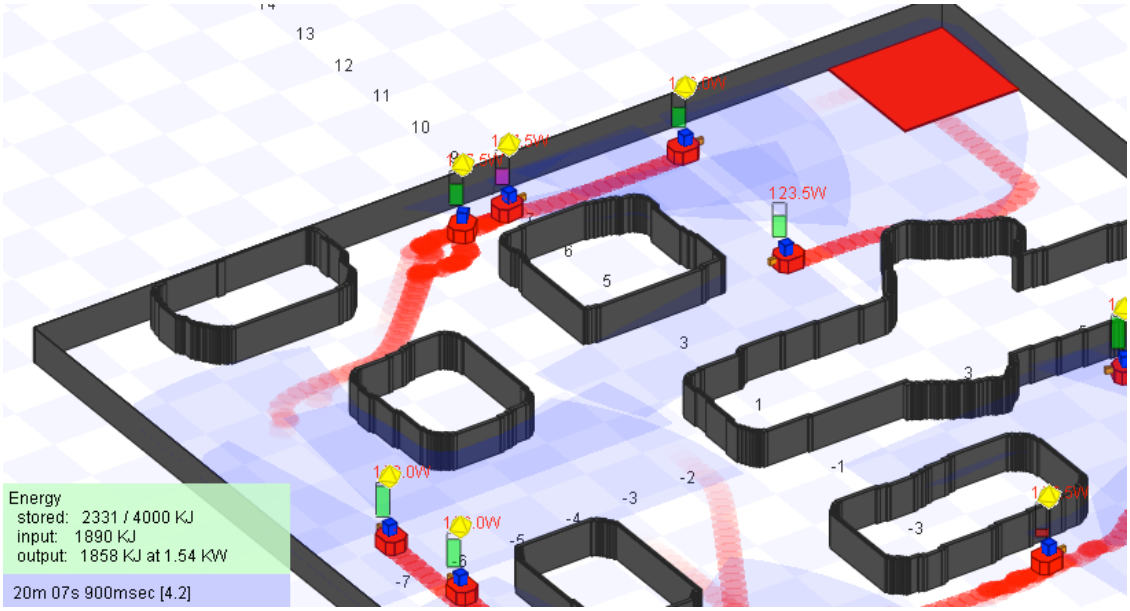


Fig 1. Example of a simulation with Stage showing several robots moving around an environment.

0.1 9.1.1 Getting ready: the provided Virtual Machine and the pre-installed ros-pkgs

For those not willing to install and configure ROS, a virtual machine (Virtual Box) is provided together with this notebook holding a Ubuntu 18.04 OS. It comes with ROS Melodic installed, and a catkin workspace ready to use.

Some indications: - User of the OS: robotics, password: robotics. - ROS installation path: /opt/ros/melodic - Catkin workspace path: /home/robotics/catkin_ws

Note-> If you want to install ROS in your computer, you can follow the detailed guide in: http://wiki.ros.org/melodic/Installation, and install the Desktop-Full version of ROS. If you choose this option, you have to know that the `open_gmapping` package is not available for ROS Melodic in the Ubuntu package repository. That package is a wrapper for **GMapping**, a SLAM method **based on particle filters** that we are going to use. However, it is easy to compile it from source. For doing that, execute the following commands:

```
cd ~/catkin_ws/src
git clone https://github.com/ros-perception/openslam_gmapping src/openslam_gmapping
git clone https://github.com/ros-perception/slam_gmapping src/slam_gmapping
cd ..
Catkin_make
```

You would also need to install the other packages that are already provided with the virtual machine. You can find them in the *Your first robotic explorer pkgs* file in the virtual campus. To install them you just have to copy its content in your `~/catkin_ws/src` directory and compile them with `catkin_make`.

0.2 9.1.2 Insight into Stage

Stage comes also pre-installed into the virtual machine. Stage simulates a world as defined in a `.world` file. This file tells stage everything about the world, from obstacles (usually represented via a bitmap to be used as a kind of background), to robots and other objects in the virtual environment. The node `stage_ros` only exposes a subset of Stage's functionality via ROS. Specifically, it finds the Stage models of type laser, camera (rgbd) and position, and maps these models to the ROS topics:

- laser_scan (`sensor_msgs/LaserScan`)
- image (`sensor_msgs/Image`)
- depth (`sensor_msgs/Image`)
- odom (`nav_msgs/Odometry`)
- base_pose_ground_truth (`nav_msgs/Odometry`) *NOT USE!!!*

You can run this node by executing the following command:

```
rosrun stage_ros stageros $(rospack find missions_pkg)/world/robotics-house4.world
```

Note→ Remember that the ROS-MASTER must always be launched before any other node (executing `roscore`)

You will see an interface like the one shown below, where the robot is represented as a blue box, and it is equipped with a 2D laser scanner. Walls are represented as gray, tall rectangles, which define the rooms within the environment where the robot is going to operate. Spend some minutes taking a look at the `View` options within the Stage interface, since they can be useful for checking different types of information.

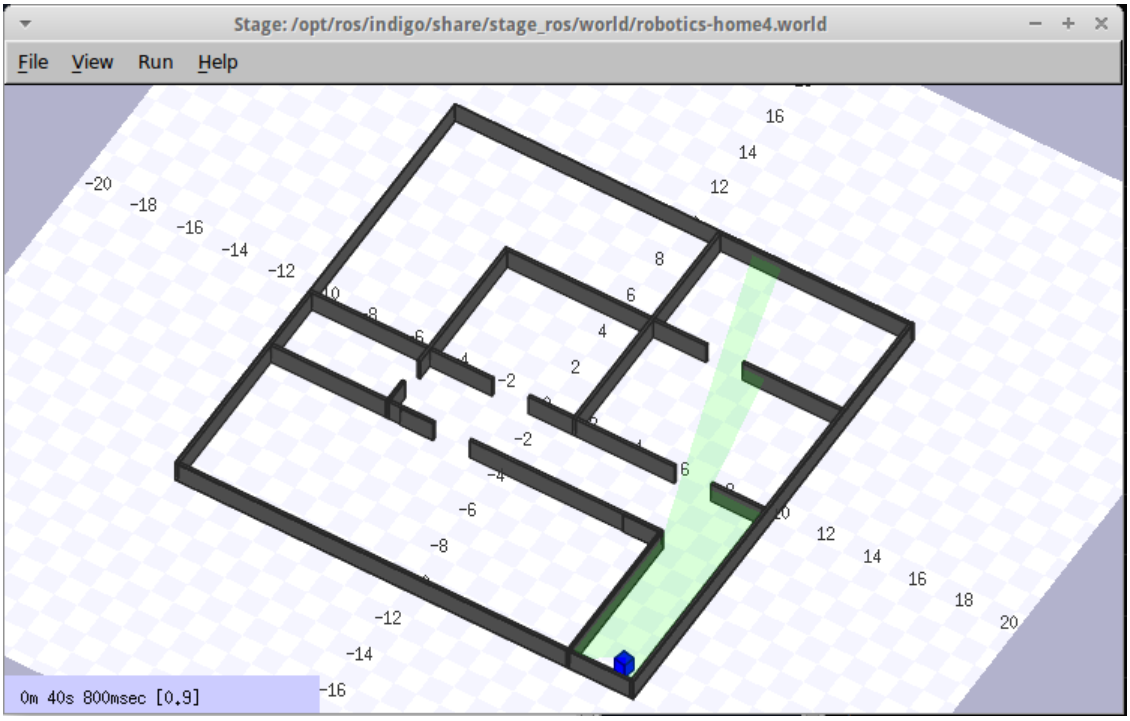


Fig 2. Example of a robot (blue square) and the readings from the laser scanner (green).

0.3 9.1.3 Some basic movement commands

You can open a new terminal and command the robot directly by publishing in the `cmd_vel` topic:

```
rostopic pub /cmd_vel geometry_msgs/Twist '[-0.5, 0, 0]' '[0, 0, 0]' -r 100
```

Since this is a very tedious form of controlling a robot, you can make use of the community and employ different packages that control a robot through the keyboard. For example, you can use the `keyboard_control` pkg that is also provided with this notebook.

You can launch this simple example with the following command, which makes use of `roslaunch`, a tool for easily launching multiple ROS nodes, as well as setting parameters on the Parameter Server:

```
roslaunch missions_pkg demo_1_stage_keyboard.launch
```

Take care commanding the robot, it can crash otherwise!

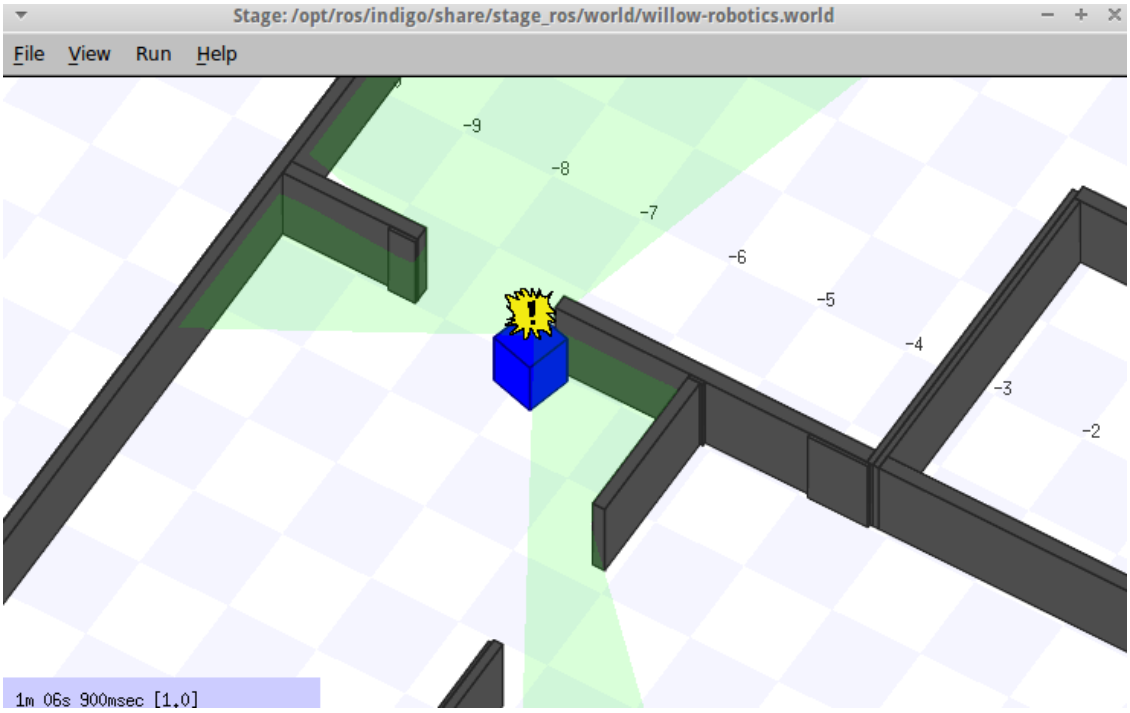
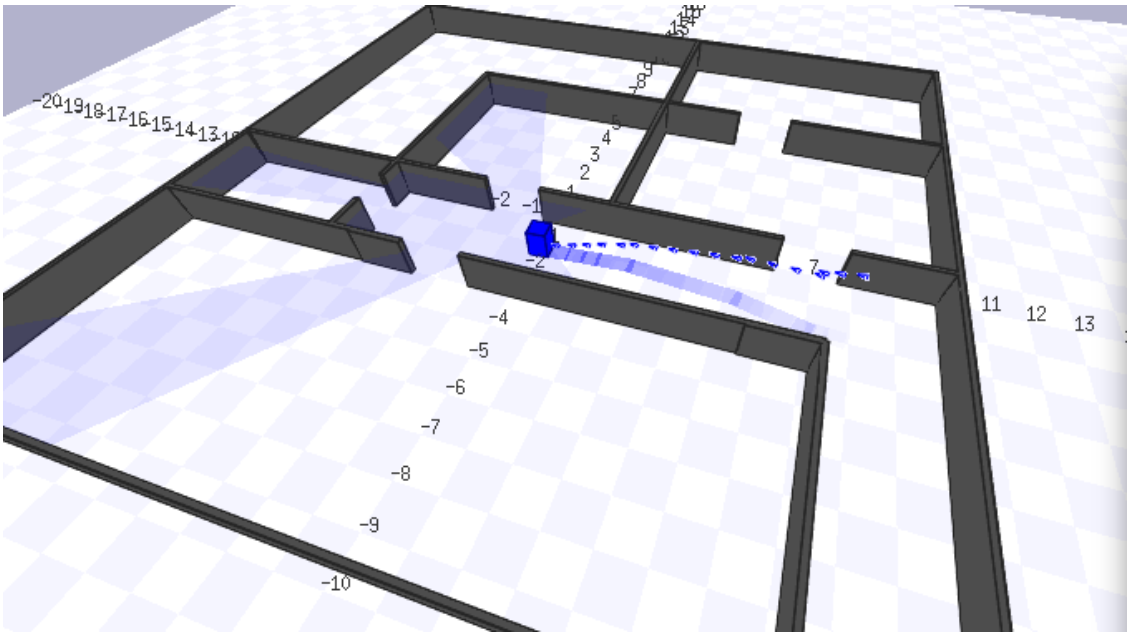


Fig 3. Example of a sad robot.

0.3.1 ASSIGNMENT 1: Manually controlling the robot

What to do?

- Command the robot using the *launch file* provided above,
- activate the needed options for visualizing the path followed by the robot in Stage, and
- take a picture of such simulation and provide it below.



1 9.1.4 SLAM demo

Although we will use Stage as a robotic simulator, it is a good practice to visualize the data coming from the robot outside the simulator. That way, when we use a real robot, the interface will be the same. RVIZ is a 3D visualization tool for ROS with a lot of pre-configured ROS message types.

As a demo for introducing ROS and RVIZ, let's take a look at a very employed package for SLAM: **Gmapping**. This package contains a ROS wrapper for **OpenSlam's Gmapping**. The `gmapping` package provides laser-based SLAM (Simultaneous Localization and Mapping) as a ROS node called `slam_gmapping`. Using `slam_gmapping`, you can create a 2-D occupancy grid map (like a building floorplan) from laser and pose data collected by a mobile robot.

As a simple exercise, you can load the Stage simulator, then launch the `gmapping` node, and finally move manually the robot to see how the map builds up while localizing the robot (SLAM). For example, you can launch this demo with the command:

```
roslaunch missions_pkg demo_2_slam.launch
```

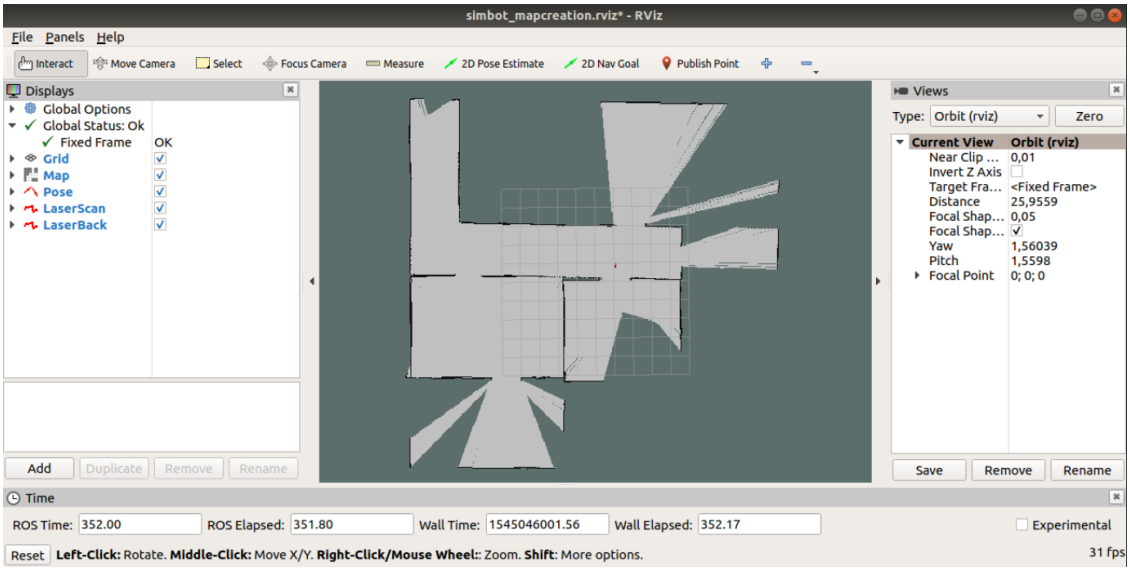


Fig 4. Example of a map under construction.

1.0.1 ASSIGNMENT 2: Building maps

Launch the previous *launch file* for every environment provided. Said environments are defined in the *world files*:

- `robotics-house1.world`
- `robotics-house2.world`
- `robotics-house3.world`
- `robotics-house4.world`

- `robotics-offices1.world`

The content of the `demo_2_slam.launch` file is as follows:

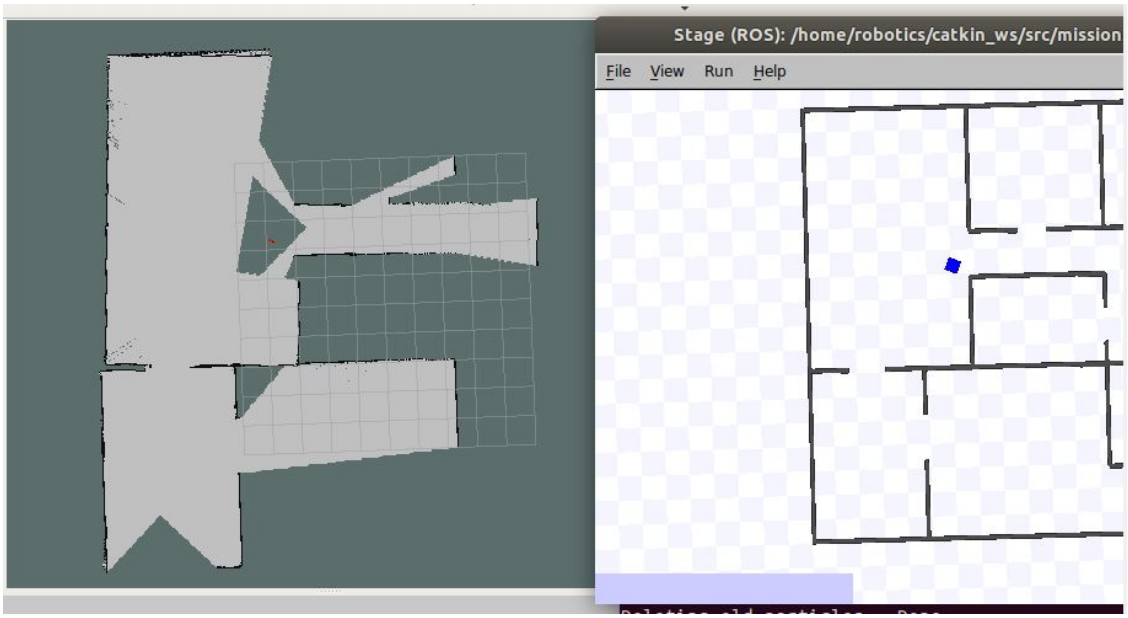
```
<!-- Launch file for the Robotics exercices with ROS -->

<launch>

    ### ROBOT SIMULATION ###
    <param name="use_sim_time" value="true" />
    <include file="$(find missions_pkg)/launch/simbot_stage.launch" >
        <arg name="world_file" value="-d $(find missions_pkg)/world/robotics-house4.world" />
    </include>

    ### URDF model "giraff" robot ###
    <include file="$(find missions_pkg)/launch/simbot_urdf.launch" />

    ### NAVIGATION ###
    <include file="$(find missions_pkg)/launch/simbot_keyboard_control.launch" />

    ### MAPPING/SLAM ###
    <include file="$(find missions_pkg)/launch/simbot_gmapping.launch" />

    ### RVIZ  ###
    <node name="rviz" pkg="rviz" type="rviz" respawn="false" output="screen" args="-d $(find mis

</launch>
```

So you have to modify the *world file* in the `world_file` parameter for launching the SLAM demo with each environment.

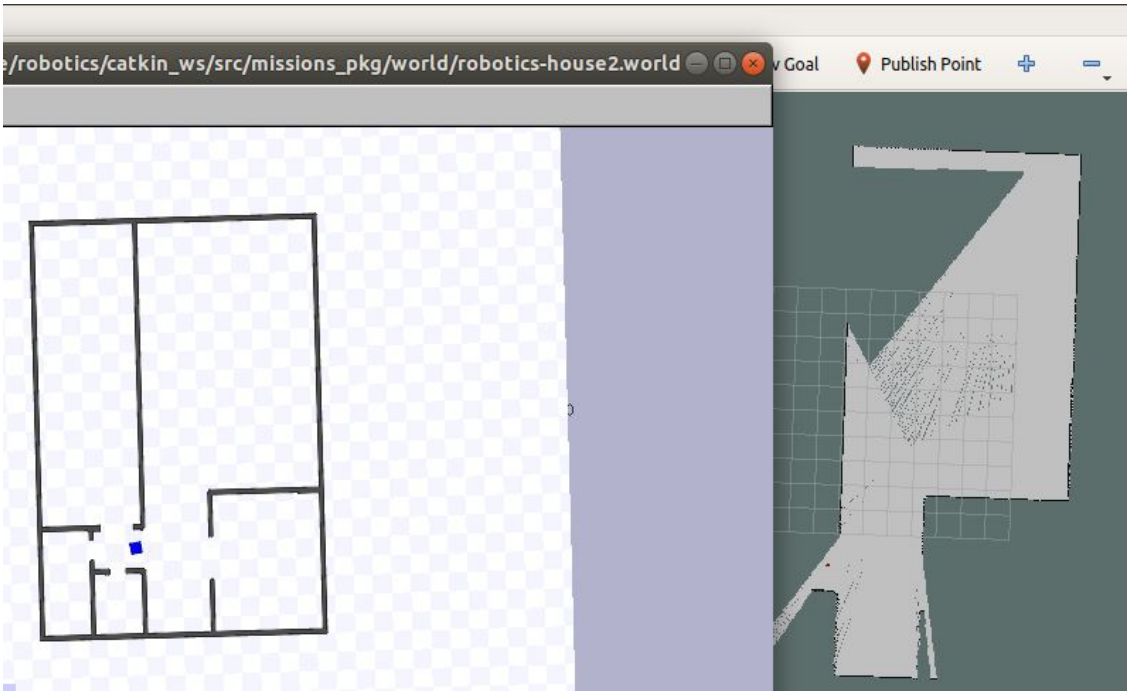
What to provide?

Include here an image of each resultant map.

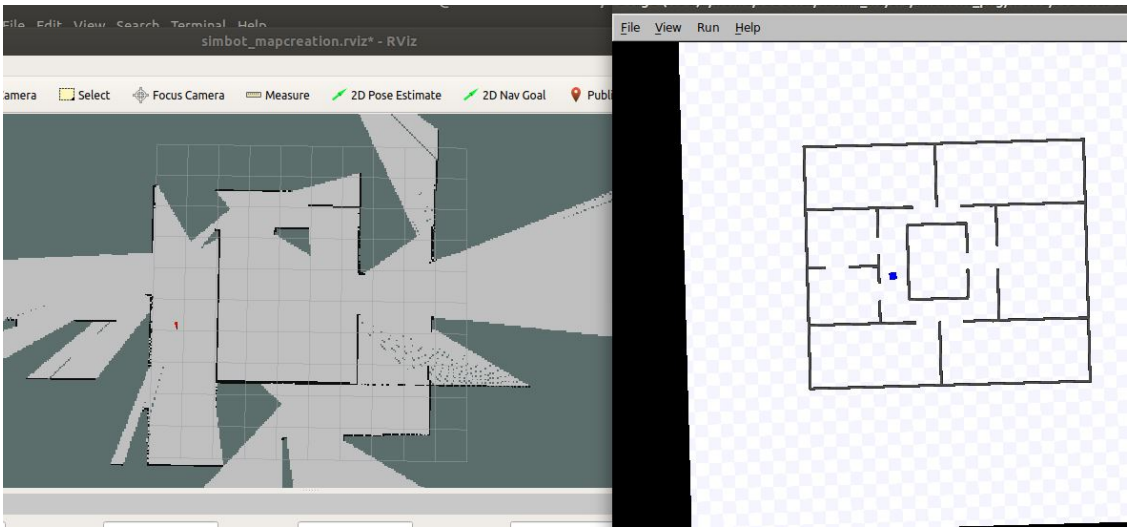
robotics-house1.world



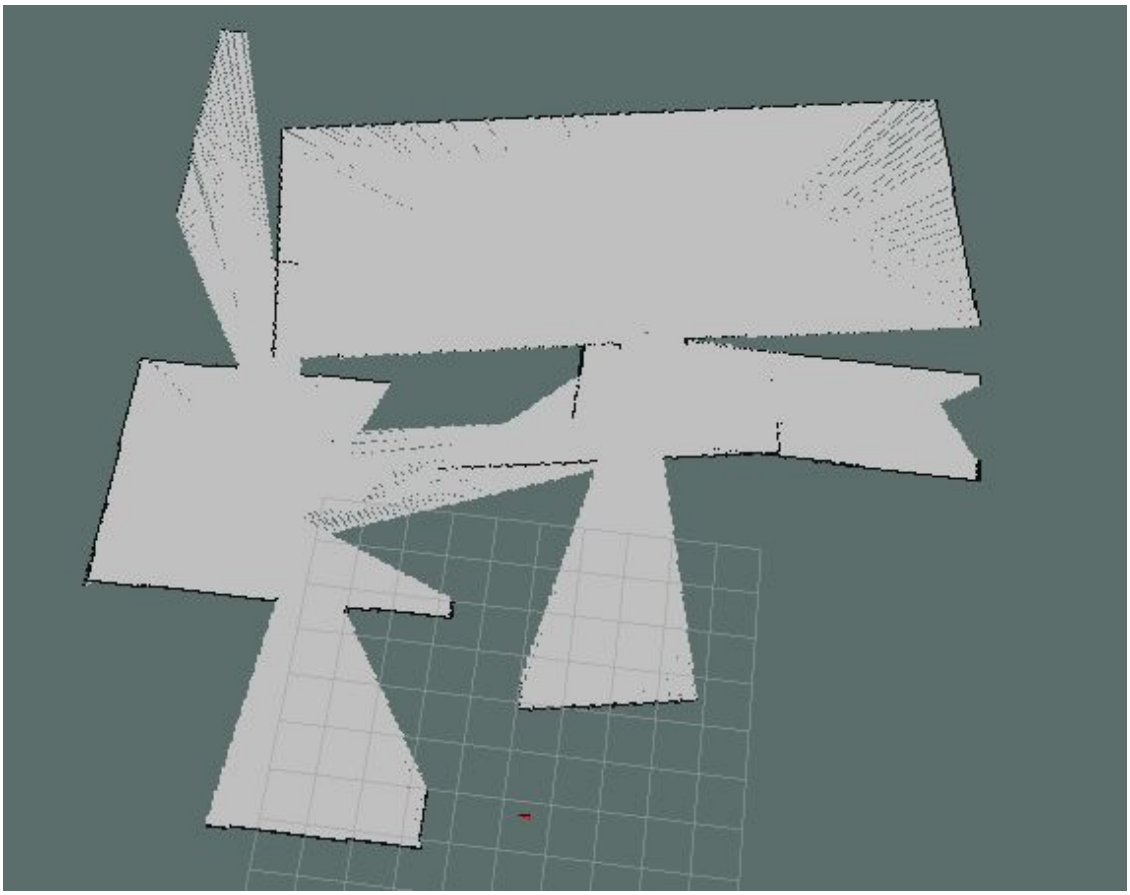
robotics-house2.world



robotics-house3.world



robotics-house4.world



robotics-offices1.world