Project 2：Creating Maze Simulation with ROS and Gazebo for Turtlebot3

1. Introduction to the Project

In this project, you are creating a maze simulation in Gazebo simulator. When this simulator is ready you can test your robot navigation code with the simulator. This will save you a lot of time and resources.

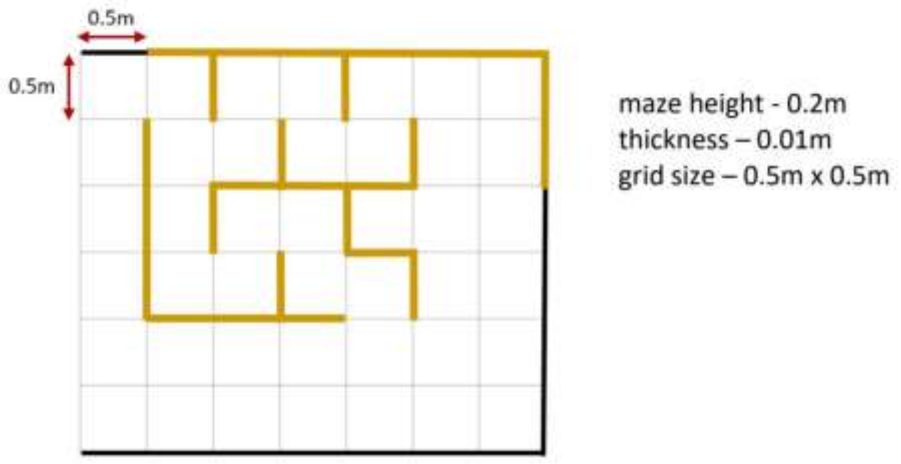
This tutorial provides a further opportunity to study simulating a ROS controlled TurtleBot3 in Gazebo. Three tasks are involved in this work, as listed below.

(1) Build a new Gazebo environment, here a maze, for robot to do SLAM and navigation;

(2) Turtlebot3 SLAM and navigation;

(3) Use program to drive a robot in a map by creating waypoints.

A video showing how to develop the simulation is shared with you. The map of the maze is provided as a .png file. You need a map file (not mandatory but helpful) to create a maze in Gazebo.



Use the existing turtlebo3_empty_world simulation to create your own simulator. First, you have to clone the turtlebo3_empty_world, and then, change it as required.

In this tutorial, you are creating a custom model (maze) and add to an existing world. Model is a keyword used to refer a virtual object in Gazebo. You can add any model from the existing models in Gazebo. Many models are available to be used in simulations. You can experiment with these models, and they will be useful for your project. Some example models are cars, bicycles, trees, houses, walls, person (static and moving), many objects, drones, furniture etc.

Note: (1) Always compile your packages after making changes to your files;
(2) Ensure Gazebo Turtlebot3 simulations packages are already installed.

2. Experimental Procedure

Open a terminal, create a new catkin workspace and inside the workspace, create a new package called my_simulation.

```
mkdir -p ~/project2_ws/src
cd ~/project2_ws/src
catkin_init_workspace
catkin_create_pkg my_simulation rospy roscpp std_msgs
cd ~/project2_ws && mkdir launch && mkdir worlds
roscd turtlebot3_gazebo/launch
cp turtlebot3_empty_world.launch ~/project2_ws/src/my_simulation/launch/my_world.launch
cd ../worlds
cp empty.world ~/project2_ws/src/my_simulation/worlds/empty_world.world
cd ~/project2_ws/src/my_simulation/launch && gedit my_world.launch
```

Modify the 7th line, change turtlebot3_gazebo to my_simulation, change empty.world to empty_world.world, save it. Now compile your new package in your new workspace, and run your launch file script.

```
cd ~/project2_ws && catkin_make && source devel/setup.bash
roslaunch my_simulation my_world.launch
```

So far, you have successfully cloned the turtlebot3_empty_world simulation to your own simulation. Next, we will create a maze model in your new simulation.

Now let's create a maze using a map. The map is just a guide. If you want your can create the maze without using a map. The map shown above in Introduction section can be downloaded in our project website. Maze height is 0.2m, and thickness is 0.01m.

Given your new simulation has been launched, go to Editor => Building Editor, then click on Import at the bottom of the left panel, in the pop-up menu, choose the map you've just downloaded, click Next. In the maze map, use your mouse to specify the left and right borders, change the distance in the left Distance item to 10m, click OK.

The map will be shown on top of the world. Now, click on Create Walls on the left panel, and drag along the walls of the maze, you will see the corresponding virtual walls in the Gazebo simulator. Click on the wall of the map, you can adjust its properties. You can then choose a suitable color in the left panel Add Color, and click on the object in Gazebo simulator for its effects.

Complete all walls (you need to be patient!), and save the maze model in ~/.gazebo/models directory with the name my_maze, you will see two files inside my_maze folder (.config and .sdf). They are configuration and simulation description format files. Any object in Gazebo is described with these two files. If you want to modify an object you have to edit these files manually or modify the object in a GUI and save it updating these files. For this tutorial, you are creating only four files to complete your maze.

Your simulation package

a. ~/project2_ws/src/my_simulation/worlds/empty_world.world

b. ~/project2_ws/src/my_simulation/launch/my_world.launch

Your maze model

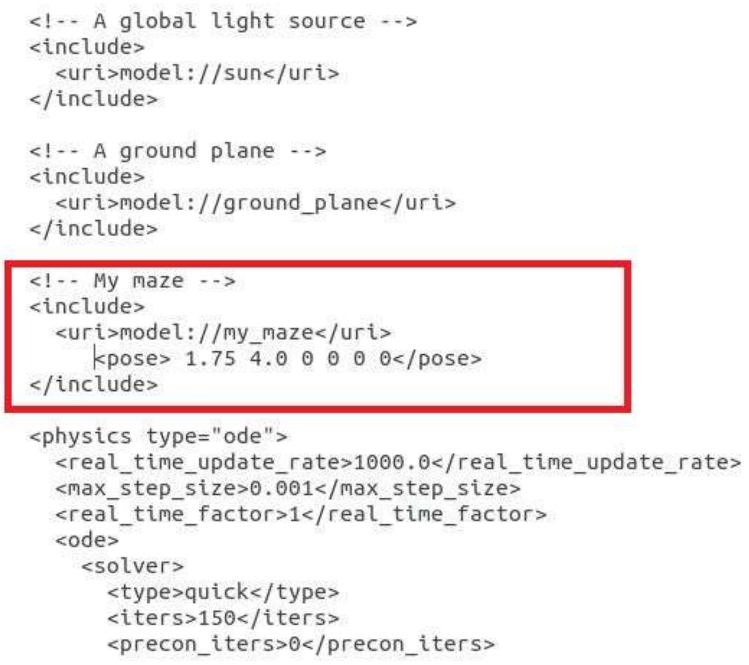
c. ~/.gazebo/models/my_maze/model.config

d. ~/.gazebo/models/my_maze/model.sdf

Next, you should modify your world file to include the new my_maze model. Open your empty_world.world, and do the following change.

```
cd ~/project2_ws/src/my_simulation/worlds && gedit empty_world.world
```

Add a snippet of XML code as shown in the red block below.



```
<!-- A global light source -->
<include>
  <uri>model://sun</uri>
</include>

<!-- A ground plane -->
<include>
  <uri>model://ground_plane</uri>
</include>

<!-- My maze -->
<include>
  <uri>model://my_maze</uri>
     <pose> 1.75 4.0 0 0 0 0</pose>
</include>

<physics type="ode">
  <real_time_update_rate>1000.0</real_time_update_rate>
  <max_step_size>0.001</max_step_size>
  <real_time_factor>1</real_time_factor>
  <ode>
    <solver>
      <type>quick</type>
      <iters>150</iters>
      <precon_iters>0</precon_iters>
```

After you modification, you need to save it, and run the launch file as follows. Select waffle, and you will see the turtlebot in a maze world.

```
export TURTLEBOT3_MODEL=waffle
roslaunch my_simulation my_world.launch
```

Keep this running. Now that you've created your own world with a turtlebot inside, let's follow the steps in our project 1 to build a map of this world, then based on the map, we have two options to command the robot:

(1). Specify the destination in rviz, and make the robot autonomously move to that point, as we did in our first project;
(2). Design a set of waypoints, and a python script including these points to drive the robot moving along these points.

First of all, we need to build the map using the following commands. Launch SLAM

```
export TURTLEBOT3_MODEL=waffle
roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
```

Remotely control turtlebot3 to move

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

After you move the robot sufficiently around its environment, you can save the map

```
rosrun map_server map_saver -f ~/map_proj
```

Now you can use the map to autonomously navigate the robot. Terminate all applications that were executed during the virtual SLAM practice and execute related packages in the following instruction, the robot will appear on the previously generated map. After setting the initial position of the robot on the map, set the destination to run the navigation as shown in figure below. The initial position only needs to be set once.

```
export TURTLEBOT3_MODEL=waffle
roslaunch my_simulation my_world.launch
```

Then open another terminal, and input

```
export TURTLEBOT3_MODEL=waffle
roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/map_proj.yaml
```

Now, after you've done the first option, we do the second, i.e., use a python script and waypoints to drive the robot.

Keep the rviz map open, click on Publish Point on the top panel (if this button does not exist, click on + symbol on the top panel, then choose PublishPoint, and click OK), use your mouse to point on the place you want your robot to pass (do not click down your mouse), you will find the [x,y,z] data on the bottom of rviz window beside Reset. You only need X and Y axes data so just record [x,y] data, ignore z value. Carry on this process repetitively along the path till the destination you want your robot to pass through, and stop until you get sufficient points for your robot to form its trajectory.

After you have recorded a set of positions you want the robot to go through, create a python script by the following commands.

```
cd ~/project2_ws/src/my_simulation && mkdir scripts && cd scripts
gedit waypoint.py
```

Then, go to my github site to copy the python script code.
https://github.com/Zhijun2/project2/blob/master/scripts/waypoint.py

Paste the code in the waypoint.py, in Line 9th, you need to modify the WAYPOINTS list with your recordings, and save it, and do not forget to chmod its property. Go to your catkin workspace to compile your package again.

Now you are ready to use your python program to drive the robot to follow the trajectory you just defined. Try the following commands.

```
export TURTLEBOT3_MODEL=waffle
roslaunch my_simulation my_world.launch
```

You can now try to move the robot following the waypoints.

```
rosrun my_simulation waypoint.py
```

Hopefully your robot can now follow the points you specified to move! Well done.

3. Conclusion

In this work you have learned how to build a tailor designed world, and use the tailor designed models in your new world. You also learned the way of recording the waypoints and use your program to drive your robot to follow the waypoints. This is a very useful achievement which you are sure to benefit in doing your future robot project.

Reference:

1. wiki.ros.org/ROS/Tutorials
2. https://emanual.robotis.com/docs/en/platform/turtlebot3
3. http://wiki.ros.org/Books/ROS_Robot_Programming_English