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

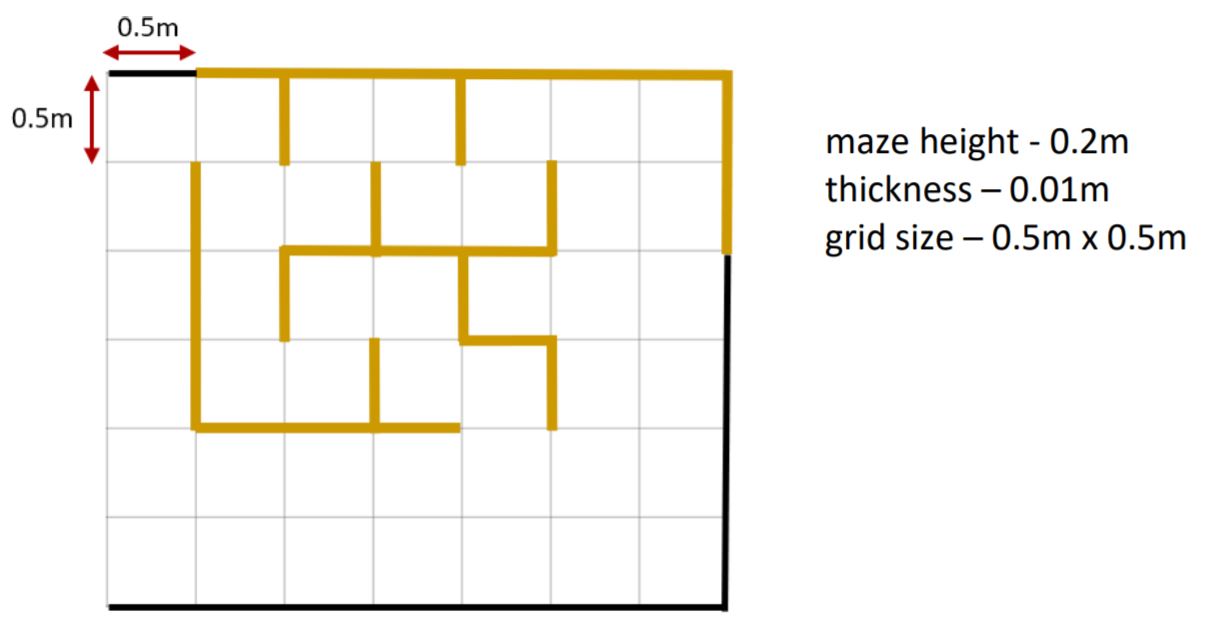
1. **Introduction to the Project**

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.

1. **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 && cakin\_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

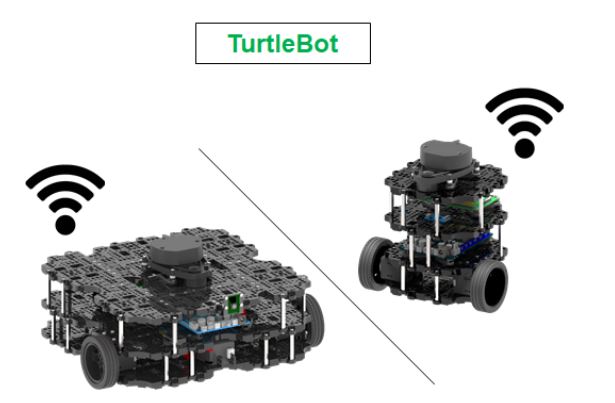
Next, you should modify your world file to include the new my\_maze model.

1. **Turtlebot3 Simulation**
   1. Turtlebot3 Simulation using Fake Node

<https://youtu.be/iHXZSLBJHMg>

To launch the virtual robot, execute the turtlebot3\_fake.launch file in the turtlebot3\_fake package as shown below. The turtlebot3\_fake is a very simple simulation node that can be run without having an actual robot. You can even control the virtual TurtleBot3 in RViz with a teleoperation node.

Before executing this command, you have to specify the model name of TurtleBot3. The ${TB3\_MODEL} is the name of the model you are using in burger, waffle, waffle\_pi.



So everytime before you run a ROS node, you need to run export TURTLEBOT3\_MODEL=${TB3\_MODEL}, where ${TB3\_MODEL} is burger, waffle or waffle\_pi. To permanently set this environment variable, you can attach the following line in your .bashrc hidden file.

export TURTLEBOT3\_MODEL=burger

Let’s run the following commands to do the simulation. In a shell, run the following:

export TURTLEBOT3\_MODEL=burger

roslaunch turtlebot3\_fake turtlebot3\_fake.launch

Open a new shell, run the following to teleoperate the robot:

export TURTLEBOT3\_MODEL=burger

roslaunch turtlebot3\_teleop turtlebot3\_teleop\_key.launch

3.2 Turtlebot3 Simulation using Gazebo

<https://youtu.be/UzOoJ6a_mOg>

There are two ways to simulate using Gazebo. first method is to use with ROS through turtlebot3\_gazebo package and second method is to use only gazebo and turtlebot3\_gazebo\_plugin plugin without using ROS. We will focus on the first way.

3.2.1 Simulation in Empty World

The following command can be used to test the virtual TurtleBot3 on the empty world of the gazebo default environment.

export TURTLEBOT3\_MODEL=burger

roslaunch turtlebot3\_gazebo turtlebot3\_empty\_world.launch

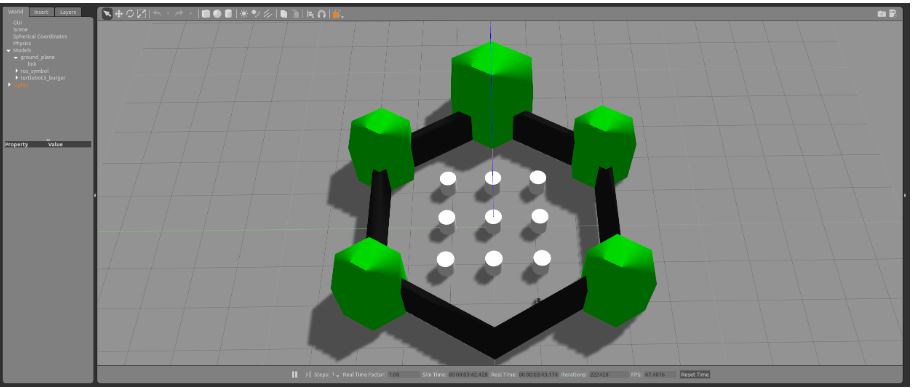
****

**3.2.2 Simulation in Turtlebot3 World**

TurtleBot3 world is a map consists of simple objects that makes up the shape of TurtleBot3 symbol. TurtleBot3 world is mainly used for testing such as SLAM and Navigation.

export TURTLEBOT3\_MODEL=burger

roslaunch turtlebot3\_gazebo turtlebot3\_world.launch

****

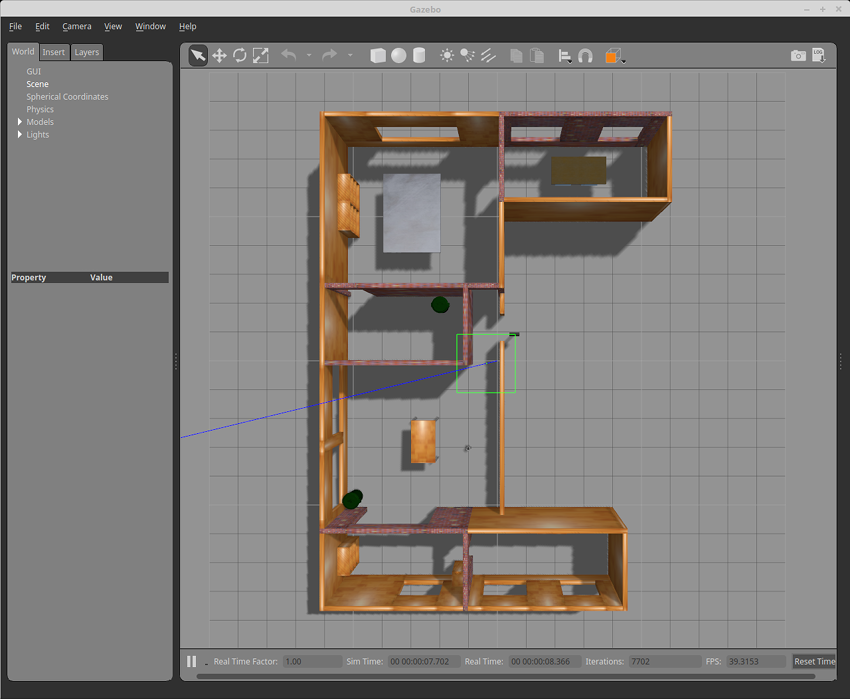
**3.2.3 Simulation in Turtlebot3 House**

TurtleBot3 House is a map made with house drawings. It is suitable for testing related to more complex task performance.

export TURTLEBOT3\_MODEL=burger

roslaunch turtlebot3\_gazebo turtlebot3\_house.launch

**NOTE** : If TurtleBot3 House is excuted for the first time, downloading the map file takes a couple of minutes or more depending on download speed. You need internet connection.



* 1. **Drive Turtlebot3**
     1. **Teleoperation on Gazebo**

In order to control a TurtleBot3 with a keyboard, please launch teleoperation feature with below command in a new terminal window.

roslaunch turtlebot3\_teleop turtlebot3\_teleop\_key.launch

* + 1. **Collision Avoidance**

In order to autonomously drive a TurtleBot3 around the **TurtleBot3 world**, open a new terminal window and enter below command.

export TURTLEBOT3\_MODEL=burger

roslaunch turtlebot3\_gazebo turtlebot3\_world.launch

Open a new terminal window and enter below command.

export TURTLEBOT3\_MODEL=burger

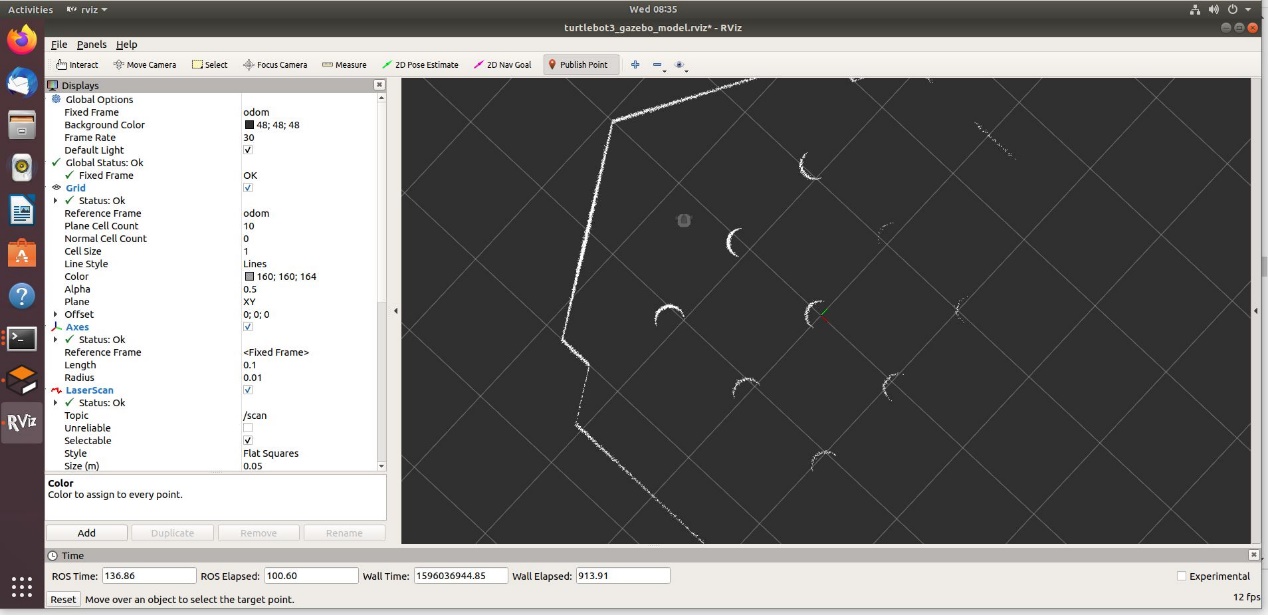
roslaunch turtlebot3\_gazebo turtlebot3\_simulation.launch

* 1. **Execute rviz**

RViz visualizes published topics while simulation is running. You can launch RViz in a new terminal window by entering below command.

export TURTLEBOT3\_MODEL=burger

roslaunch turtlebot3\_gazebo turtlebot3\_gazebo\_rviz.launch

****

* 1. **Virtual SLAM with Turtlebot3**

For virtual SLAM in Gazebo, instead of running the actual robot, you can select the various environments and robot models mentioned above, and the SLAM-related commands will use the ROS packages used in the [SLAM](https://emanual.robotis.com/docs/en/platform/turtlebot3/slam/#slam) section.

* + 1. Virtual SLAM Execution Procedure

The following commands are examples of using the TurtleBot3 Waffle Pi model and the turtlebot3\_world environment.

* Launch Gazebo

export TURTLEBOT3\_MODEL=waffle\_pi

roslaunch turtlebot3\_gazebo turtlebot3\_world.launch

* Launch SLAM

export TURTLEBOT3\_MODEL=waffle\_pi

roslaunch turtlebot3\_slam turtlebot3\_slam.launch slam\_methods:=gmapping

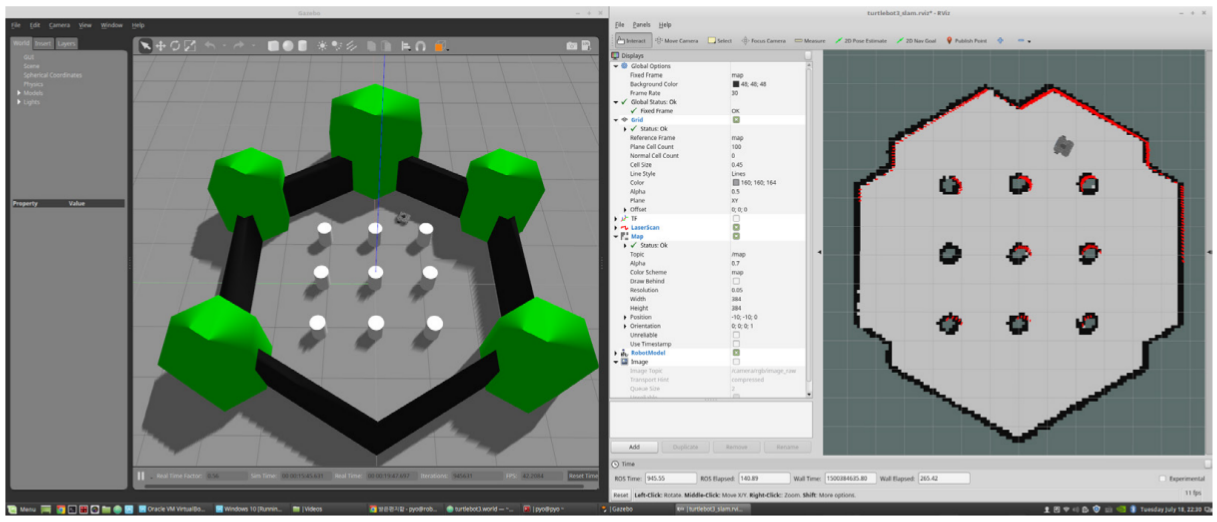
* Remotely Control TurtleBot3

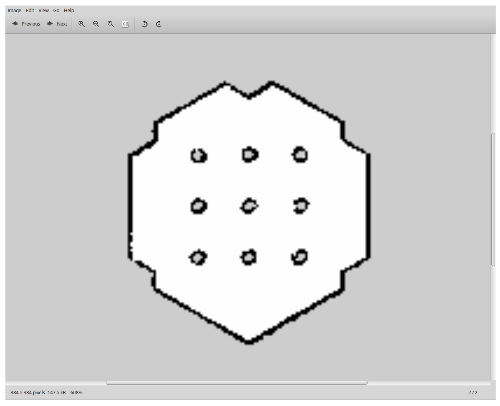
roslaunch turtlebot3\_teleop turtlebot3\_teleop\_key.launch

* Save the Map

rosrun map\_server map\_saver -f ~/map

When you run the dependent packages and move the robot in virtual space and create a map as shown above, you can create a map as shown in figure below.

****

****

* 1. **Virtual Navigation with Turtlebot3**

For virtual Navigation in Gazebo, instead of running the actual robot, you can select the various environments and robot models mentioned above, and the Navigation-related commands will use the ROS packages used in the [Navigation](https://emanual.robotis.com/docs/en/platform/turtlebot3/navigation/#navigation) section.

* + 1. Virtual Navigation Execution Procedure

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.

* Execute Gazebo

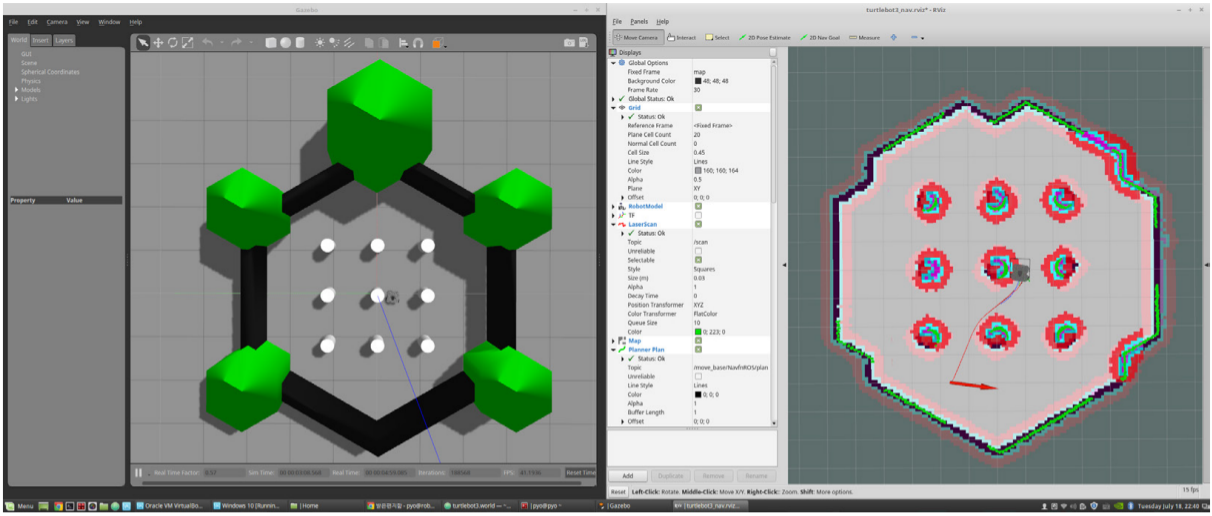
export TURTLEBOT3\_MODEL=waffle\_pi

roslaunch turtlebot3\_gazebo turtlebot3\_world.launch

* Execute Navigation

export TURTLEBOT3\_MODEL=waffle\_pi

roslaunch turtlebot3\_navigation turtlebot3\_navigation.launch map\_file:=$HOME/map.yaml



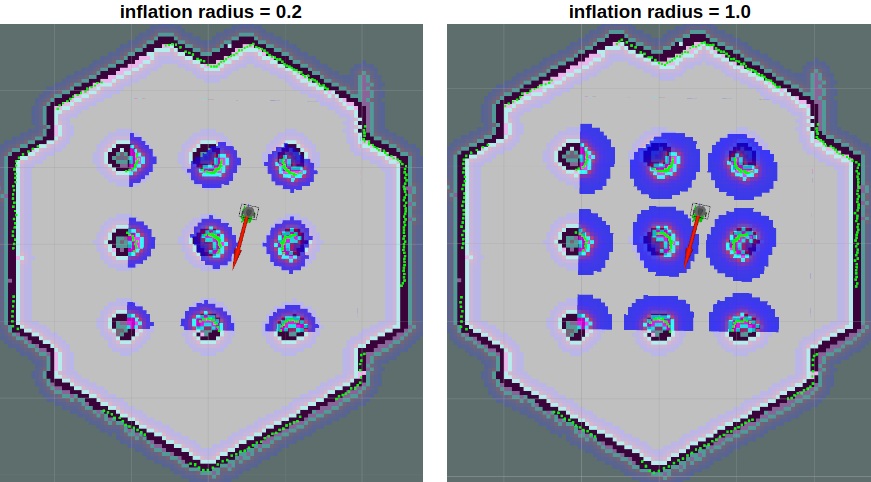
* 1. **Testing Virtual Turtlebot3 by Tuning Parameters**

Navigation stack has many parameters to change performances for different robots. You can get an information about it in [ROS Wiki](http://wiki.ros.org/navigation). This tuning guide give some tips for you to configue important parameters. If you want to change performances depends on your environments, this tips might be help you and save your time.

**3.7.1 inflation\_radius**

turtlebot3\_navigation/param/costmap\_common\_param\_$(model).yaml

This parameter makes inflation area from the obstacle. Path would be planned in order that it doesn’t cross this area. It is safe that to set this to be bigger than robot radius.



**3.7.2 cost\_scaling\_factor**

turtlebot3\_navigation/param/costmap\_common\_param\_$(model).yaml

This factor is multiplied by cost value. Because it is a reciprocal proportion, this parameter is increased, the cost is decreased. The best path is for the robot to pass through a center of between obstacles. Set this factor to be smaller in order to far from obstacles.

**3.7.3 Other Parameters**

***max\_vel\_x***

turtlebot3\_navigation/param/dwa\_local\_planner\_params\_$(model).yaml

This factor is set the maximum value of translational velocity.

***min\_vel\_x***

turtlebot3\_navigation/param/dwa\_local\_planner\_params\_$(model).yaml

This factor is set the minimum value of translational velocity. If set this negative, the robot can move backwards.

***max\_trans\_vel***

turtlebot3\_navigation/param/dwa\_local\_planner\_params\_$(model).yaml

Actual value of the maximum translational velocity. The robot cannot be faster than this.

***min\_trans\_vel***

turtlebot3\_navigation/param/dwa\_local\_planner\_params\_$(model).yaml

Actual value of the minimum translational velocity. The robot can not be slower than this.

***max\_rot\_vel***

turtlebot3\_navigation/param/dwa\_local\_planner\_params\_$(model).yaml

Actual value of the maximum rotational velocity. The robot can not be faster than this.

***min\_rot\_vel***

turtlebot3\_navigation/param/dwa\_local\_planner\_params\_$(model).yaml

Actual value of the minimum rotational velocity. The robot can not be slower than this.

***acc\_lim\_x***

turtlebot3\_navigation/param/dwa\_local\_planner\_params\_$(model).yaml

Actual value of the translational acceleration limit.

***acc\_lim\_theta***

turtlebot3\_navigation/param/dwa\_local\_planner\_params\_$(model).yaml

Actual value of the rotational acceleration limit.

***xy\_goal\_tolerance***

turtlebot3\_navigation/param/dwa\_local\_planner\_params\_$(model).yaml

The x,y distance allowed when the robot reaches its goal pose.

***yaw\_goal\_tolerance***

turtlebot3\_navigation/param/dwa\_local\_planner\_params\_$(model).yaml

The yaw angle allowed when the robot reaches its goal pose.

**Exercise: Please change the inflation\_radius to see the navigation effect.**

1. **Conclusion**

SLAM and navigation are two important areas in intelligent robotics. ROS has realized the SLAM algorithm, namely gmapping, and the navigation algorithm, namely acml. In this project, we have done a range of exercises to study how to use the related ROS packages to do SLAM and navigation for the Robotis Turtlebot3. It is expected that you will be able to know the basic knowledge of the ROS SLAM and navigation applications so that you will be able to do the similar exercises using other third party packages.

Reference:

1. 表允晳 赵汉哲 郑黎蝹 林泰勋， “ROS机器人编程”， Robotis Co.Ltd., 2017
2. <https://emanual.robotis.com/docs/en/platform/turtlebot3>