EE3305/ME3243
Robotic System Design
Manual of Project 2
Implementation of Path Planning in ROS

Teaching Assistant: Guo Haoren haorenguo_06@u.nus.edu
Instructor 1: Dr. Andi Sudjana Putra andi_sp@nus.edu.sg
Instructor 2: A/Prof. Prahlad Vadakkepat prahlad@nus.edu.sg

©National University of Singapore

AY 2023/2024 Semester 1
October 15, 2023

Contents

1 About the Project

1. Students are expected to simulate a Turtlebot3 robot manoeuvering a given maze. The **size** of the maze is **known** to Turtlebot3. The maze is divided into cells, where each cell has four directions. The **obstacle** of the maze is **unknown** to Turtlebot3.

2. Students are expected to simulate Turtlebot3 moves autonomously from the initial cell to the destination cell. The destination cell is according to the last alphabet of a student's matriculation number as assigned in Figure 1.

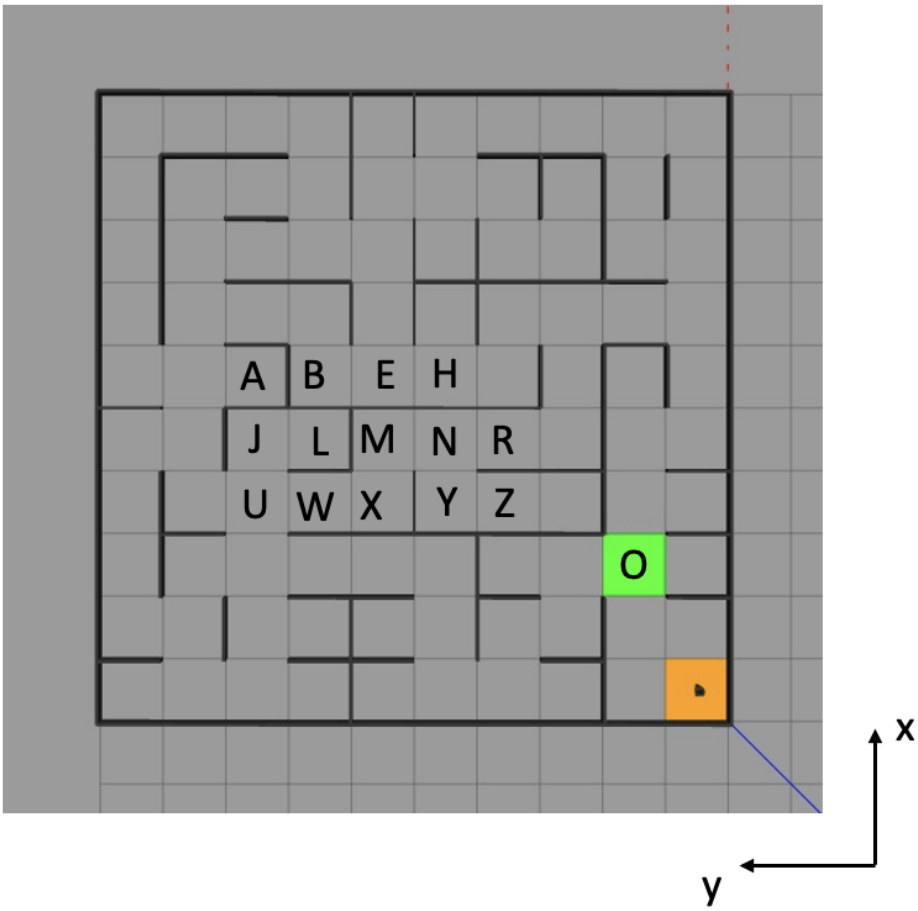

Figure 1: Assignment of destination cell

3. Information about Turtlebot3 is available here https://emanual.robotis.com/docs/en/platform/turtlebot3/features/#specifications).

The robot operates as follows:

1. The sensors detect the walls around the current cell. The information is then passed on to another *node* that plans the path.

2. Robot's knowledge of the map is kept being updated as more information is received. Using a path planning algorithm, the path of the robot is computed. Information about where the robot is heading to is then passed on to another *node* that control the motion.

3. After receiving the next destination, the robot moves accordingly using a control algorithm.

2 Objectives

At the end of the project, students are expected to demonstrate their ability to:

1. Apply a control system in a path planning problem
2. Apply a path planning algorithm in a ROS simulation
3. Present and analyse the performance of the navigation system

For your report and presentation, you need to produce the following:

1. A figure showing the original cell, the destination cell and the Turtlebot3.
2. A short video showing the Turtlebot3 robot reaching the destination cell. Use the view that can best represent your work.
3. One graph of the *nodes* and *topics* of the ROS simulation.

3 Submissions (Demo, Report and Code)

A **demo** is required to show students running ROS. In the demo, students may be asked to show his/her original simulation and/or to change the location of the destination cell.

Report should be titled "2_Path_[your_name].pdf". The content of the report is as follows:

1. Initial conditions. Show the initial and assigned destination cells in the maze.
2. Path planning.
   a) Discuss how the path planning works, in particular the lines titled *"Poll the queue"* and *"Find the path if the goal is found and break the loop"*.
   b) Identify (1) the path planned at the start cell, (2) the path planned at cell O (refer to Figure 1) and (3) the eventual path. Discuss why they are the same or different.
3. Navigation. Discuss how the PID control is implemented, including:
   a) The code that you have added,
   b) The control gains.
4. Performance enhancement. Discuss what you planned to improve, how you did it and the results.
5. ROS structure. Include the graph of ROS nodes and topics of the simulation. Identify and describe the 3 main *nodes* and *topics* pertaining to those nodes.

The **code** submission should contain the zip of the entire package (`a345b_path` in the above example). As you may be sharing a machine, it is possible that the name of the package is the same between you and your pair. However, you should indicate your name and NUSNET ID clearly in the following files:

1. The launch files, i.e., `a345b_path.launch` and `start_all_nodes.launch`.

2. The PID control node, i.e., `bot_control_node.cpp` file.
3. The path planning node, i.e., `path_plan_node.cpp` file.
4. The simulation and PID parameters, i.e., `config.yaml`.
5. `predefine.hpp` file.

Additionally, for the presentation, capture a short **video** showing the Turtlebot3 reaching the destination cell.

4 Preparation (for personal machines, not applicable to machines in the lab)

1. Install ROS Noetic.
2. Install Turtlebot3 robot in ROS Noetic as follows:

```
# husky and turtlebot dependencies
$ cd ~
$ sudo apt install ros-noetic-husky-gazebo ros-noetic-turtlebot3-* --yes

# write to .bashrc
$ echo "export TURTLEBOT3_MODEL=burger">> ~/.bashrc
$ source ~/.bashrc
$ source ~/.bashrc
```

3. Optional: Install Sublime Text editor using the command:

```
$ sudo snap install sublime-text --classic
```

5 File Structure of ROS Simulation

The file structure of ROS simulation is presented in Figure 2.

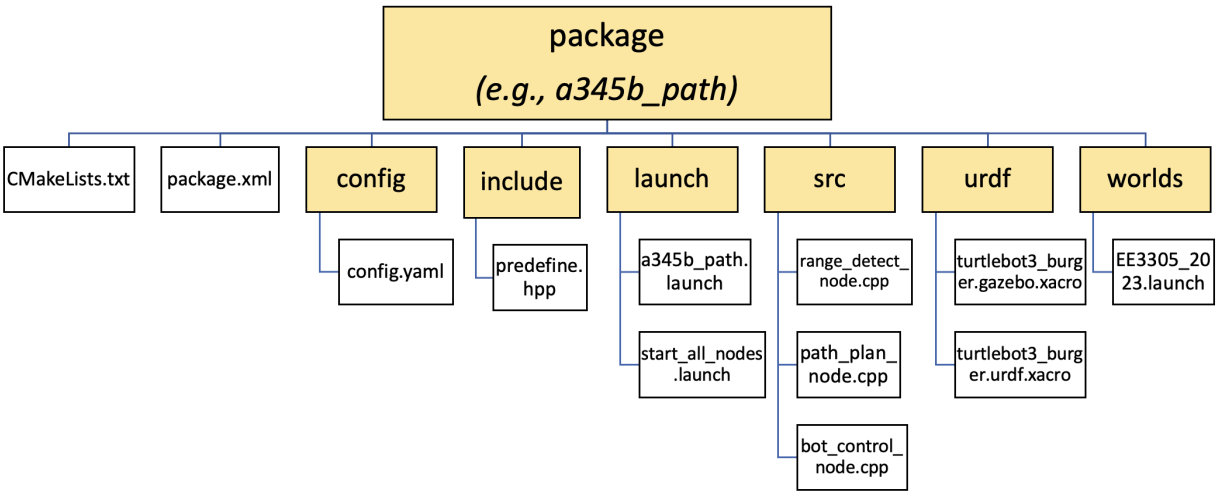

Figure 2: File structure of ROS simulation

The files to download are available at Canvas as follows:

1. predefine.hpp
2. range_detect_node.cpp
3. template_path_plan_node.cpp, where you need to modify and rename to path_plan_node.cpp
4. template_bot_control_node.cpp, where you need to modify and rename to bot_control_node.cpp
5. turtlebot3_burger.gazebo.xacro (same as the one in Project 1)
6. turtlebot3_burger.urdf.xacro (same as the one in Project 1)
7. EE3305_2023.launch

6 Steps

1. Create a workspace and build it.

   At the Home directory, create your own **workspace** with a unique name, such as using a letter and the last 4 characters of matriculation number. (If your matriculation number is A0012345B, the workspace name is `a345b_path_ws`, where `path` is the title of Project 2 and `ws` is the customary ending for a workspace.) Use the following commands:

```
$ cd ~
$ mkdir a345b_path_ws
$ cd a345b_path_ws
$ mkdir src
$ cd ~/a345b_path_ws
$ catkin_make
```

2. Create a package.

   Inside the `/src` directory of the workspace, create a package using unique name, such as using a letter and the last 4 characters of matriculation number as above. If your matriculation number is A0012345B, the package name is `a345b_path`. Use command `$ catkin_create_pkg package_name` as follows:

   ```
   # In ~/a345b_path_ws/src
   $ catkin_create_pkg a345b_path std_msgs roscpp
   ```

   When the package has been successfully built:

   - A directory whose name is `a345b_path` will be generated.
   - Inside the package, two files are generated: `CMakeLists.txt` and `package.xml`.

3. Generate a maze.

   Inside the package, create a directory called `worlds` (hint: use command `$ mkdir`). In the above example:

   ```
   # In ~/a345b_path_ws/src/a345b_path
   $ mkdir worlds
   ```

   Download the world file `EE3305_2023.world` as provided and place it in the `worlds` directory.

4. Prepare to launch the maze.

   Inside the package, create a directory called `launch` (hint: use command `$ mkdir`). In the above example:

   ```
   # In ~/a345b_path_ws/src/a345b_path
   $ mkdir launch
   ```

   Create a launch file in the `launch` directory using a unique name, such as using a letter and the last 4 characters of matriculation number as above. (If your matriculation number is A0012345B, the launch file name is `a345b_path.launch`). (hint: use command `$ touch`). Write the following lines:

```
<?xml version="1.0"?>

<!--
  EE3305/ME3243
  Name: YOUR NAME
  NUSNET ID: YOUR NUSNET ID
-->

<launch>
  <arg name="model" default="burger"/>
  <arg name="x" default="0.5"/>
  <arg name="y" default="0.5"/>
  <arg name="yaw" default="1.57"/>

  <!-- Launch Gazebo. -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find
    ↪  a345b_path)/worlds/EE3305_2023.world"/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>

  <!-- Use Gazebo service to spawn the robot. -->
  <param name="robot_description" command="$(find xacro)/xacro --inorder
  ↪  $(find turtlebot3_description)/urdf/turtlebot3_$(arg
  ↪  model).urdf.xacro"/>
  <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf" args="-urdf
  ↪  -model turtlebot3_$(arg model) -x $(arg x) -y $(arg y) -Y $(arg yaw)
  ↪  -param robot_description"/>

</launch>
```

5. Copy the robot model of Turtlebot3.

   Create a directory called `urdf` in the package (hint: use command `$ mkdir directory_name`) as follows:

   ```
   # In ~/a345b_path_ws/src/a345b_path
   mkdir urdf
   ```

   Download the model files `turtlebot3_burger.gazebo.xacro` and `turtlebot3_burger.urdf.xacro` and place them in the `urdf` directory.

6. Build the package.

   In the workspace, build the package. In the above example, the workspace is `a345b_path_ws`, hence:

```
# In ~/a345b_path_ws
$ catkin_make
```

7. Source own workspace.

   Source own workspace. To do this, go to workspace and use command `$ source ./devel/setup.bash` as follows:

   ```
   # In ~/a345_path_ws
   $ source ./devel/setup.bash
   ```

   To check that the sourcing to your workspace has happened, you can go to the workspace and use command `$ echo ROS_PACKAGE_PATH`. It should point to your workspace.

8. Launch the environment.

   From the workspace, launch the environment using command

   `$ roslaunch package_name launch_file_name.launch`. In this example, the workspace is `~/a345_path_ws`, the package is `a345b_path` and the launch file name is `a345b_path`, hence:

   ```
   # In ~/a345b_path_ws
   $ roslaunch a345b_path a345b_path.launch
   ```

   A window should appear showing the environment of a maze with the Turtlebot3 robot as shown in Figure 3. The initial cell is marked orange in cell (0,0) and the destination cell is marked green in cell (2,1). Note that the maze orientation is turned 90°. Turtlebot3 is in the initial cell.

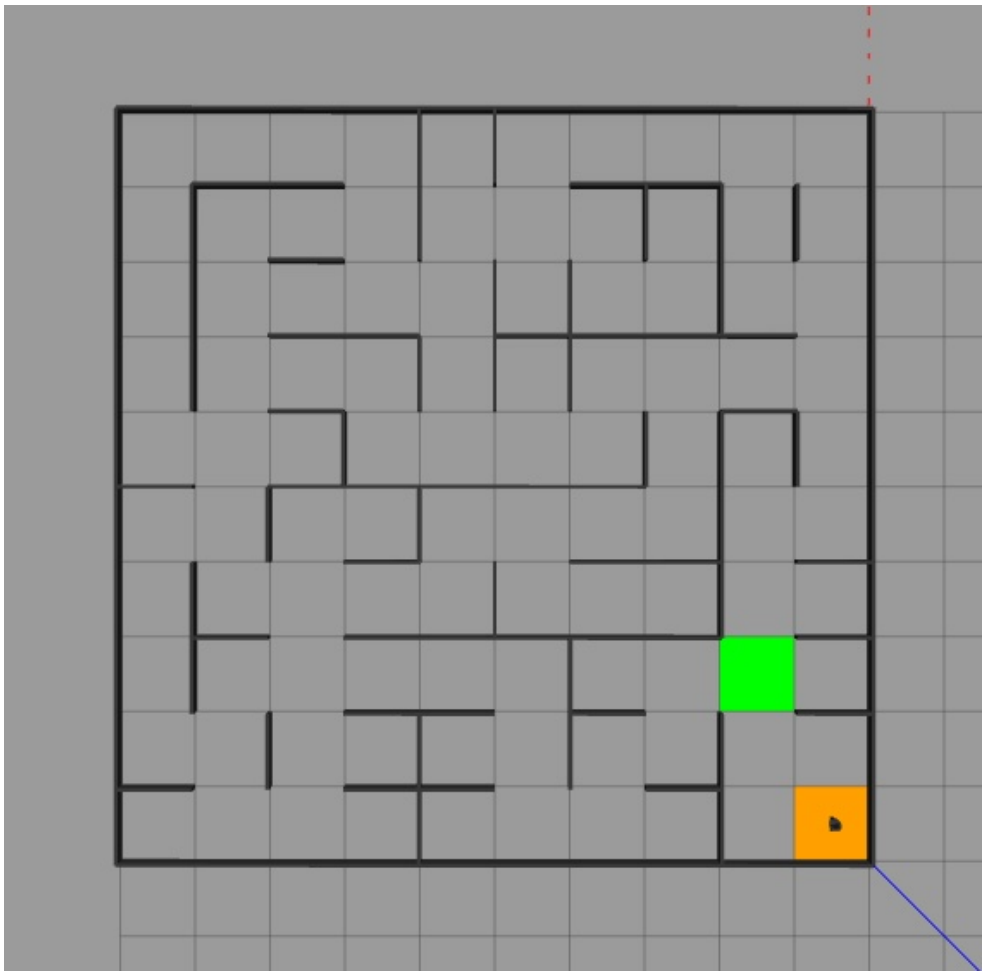

Figure 3: The default maze

Loading the world for the first time may take some time (e.g., 10 minutes is possible). Loading it the second time onward will be much faster.

Once the maze is produced, close the simulation and proceed to the next steps.

9. Change the location of the destination cell to the assigned location and take a screenshot.

   The **visual** of the location of the destination cell is assigned in `EE3305_2023.world`.

   Open `EE3305_2023.world`. In the above example:

```
# In ~/a345b_path_ws/src/a345b_path/worlds
$ subl EE3305_2023.world
```

   The location of the original cell is defined on line 96, 99 and 654. It is currently as follows:

```
<pose> 0.5 0.5 0.001 0 -0 0 </pose>
```

   where `(0.5, 0.5)` refers to the coordinate of the centre of the cell.

   The location of the destination cell is defined on line 106, 109 and 714. It is currently as follows:

```
<pose> 2.5 1.5 0.001 0 -0 0</pose>
```

where `(2.5, 1.5)` refers to the centre coordinate of the cell.

Change the location of the destination cell according to the one assigned to you.

Launch the environment again (refer to Step 8) and take a screenshot, showing the assigned destination cell. **You will need the screenshot for your report.**

10. Copy header files to an `include` directory and define the destination cell in the header file.

    Inside the `include` directory of the package, copy the file `predefine.hpp`. This file contains the location of the cell that Turtlebot3 is supposed to go to. The location is defined as `GOAL_X` and `GOAL_Y`. Open `predefine.hpp`. (Hint: use command `subl`).

    a) **Insert your name and NUSNET ID in the space provided at the start of the code.**

    b) Change the destination cell according to the one assigned to you.

11. Copy nodes to the `src` directory.

    Copy the following files to the `/src` directory of the package according to the 3 nodes in this simulation.

    a) For the node to detect the environment, copy `range_detect_node.cpp` to the `/src` directory.

    b) For the node to plan the path, copy `template_path_plan_node.cpp` to the `/src` directory. Rename `template_path_plan_node.cpp` into `path_plan_node.cpp`.

    c) For the node to control the movement of the robot, copy `template_bot_control_node.cpp` to the `/src` directory. Rename `template_bot_control_node.cpp` into `bot_control_node.cpp`.

12. Define the path in `path_plan_node.cpp`

    The file `path_plan_node.cpp` is supposed to define the path planning.

    a) **Insert your name and NUSNET ID in the space provided at the start of the code.**

    b) The code is structured as follows:

        i. Initialising the map size and walls.

        ii. Calling subscribers: the position of the robot and the walls.

        iii. Initialising the vertices for Breadth First Search (BFS).

        iv. Initialising the queue for BFS.

        v. Updating the queue **continuously**.

        vi. Finding the path when the goal is found (where you need to enter your comments to some lines of code).

        vii. Note that the goal is defined with `goal_x` and `goal_y`. They need to be set in `config.yaml` in the subsequent step.

    c) Find the lines `// WHAT DOES THIS LINE OF CODE DO?`. There are 5 such lines. **This is where you should enter your comment to the the lines of code to complete the path finding algorithm.**

13. Define the PID control in `bot_control_node.cpp`

    The file `bot_control_node.cpp` is to define the PID control.

a) **Insert your name and NUSNET ID in the space provided at the start of the code.**

b) Referring to Project 1, study the code to understand the structure. Note the variables of the PID parameters. They need to be tuned in `config.yaml` in the subsequent step.

c) You need to insert your own code to implement PID control. Find the lines `// ENTER YOUR PID CODE HERE` and `// END OF YOUR PID CODE HERE`. **This is where you should enter your code to implement the PID control.**

14. Create a `config` directory.

In the package, create a directory called `config` (hint: use command `$ mkdir`) and then create the file `config.yaml` (hint: use command `$ touch`).

Like in Project 1, `config.yaml` contains the parameters to tune PID controller and the destination. Note that the name of the parameters is slightly different, e.g., `Kp_x` instead of `Kp_f`. **Tune the PID gains and explain about it in the report.** The destination point is defined as `goal_x` and `goal_y`.

Open the file `config.yaml` (hint: use command `$ subl`). Write as follows:

a) Write your name and NUSNET ID.

b) Key in the parameters to tune PID controller and the destination.

The syntax is as follows:

```
# EE3305/ME3243
# Name: YOUR NAME
# NUSNET ID: YOUR NUSNET ID

Kp_x: insert a value
# enter the rest of the variables (9 variables total)
...
```

15. Modify the `CMakeLists.txt` file.

In the package, open `CMakeLists.txt` file (hint: use command `$ subl`) as follows (refer to the example):

```
# In ~/a345b_path_ws/src/a345b_path
$ subl CMakeLists.txt
```

Most of its statements are commented, i.e. inactive. You will uncommment statements that you need. Like in Project 1, the following packages are added in `CMakeLists.txt`: `roscpp`, `sensor_msgs`, `nav_msgs`, `geometry_msgs` and `std_msgs`.

First, find `find_package`. Make sure it is uncommented (by removing the # sign) and add the abovementioned packages. The resulting statements should be as follows:

```
find_package(catkin REQUIRED COMPONENTS
    roscpp
    sensor_msgs
    nav_msgs
    geometry_msgs
    std_msgs
)
```

Second, find `catkin_package` and add the same dependency packages as in `find_package`. The resulting statements should be as follows:

```
catkin_package(
    INCLUDE_DIRS include
    CATKIN_DEPENDS
        roscpp
        sensor_msgs
        nav_msgs
        geometry_msgs
        std_msgs
)
```

Third, as you have `include` directory in the package, uncomment `include_directories`. The resulting statements should be as follows:

```
include_directories(
    include
    ${catkin_INCLUDE_DIRS}
)
```

Fourth, uncomment `add_executable` and `target_link_libraries` and include the files that define the 3 nodes. The resulting statements should be as follows:

```
add_executable(
    range_detect_node
    src/range_detect_node.cpp
)

target_link_libraries(
    range_detect_node
    ${catkin_LIBRARIES}
)

add_executable(
    path_plan_node
    src/path_plan_node.cpp
)

target_link_libraries(
    path_plan_node
    ${catkin_LIBRARIES}
)

add_executable(
    bot_control_node
    src/bot_control_node.cpp
)

target_link_libraries(
    bot_control_node
    ${catkin_LIBRARIES}
)
```

The `CMakeLists.txt` file should look as follows (excluding comments and unnecessary lines):

```
cmake_minimum_required(VERSION X.X.X)
project(a345b_path)

find_package(catkin REQUIRED COMPONENTS
    roscpp
    sensor_msgs
    nav_msgs
    geometry_msgs
    std_msgs
)

catkin_package(
    INCLUDE_DIRS include
    CATKIN_DEPENDS
        roscpp
        sensor_msgs
        nav_msgs
        geometry_msgs
        std_msgs
)

include_directories(
    include
    ${catkin_INCLUDE_DIRS}
)

add_executable(
    range_detect_node
    src/range_detect_node.cpp
)

target_link_libraries(
    range_detect_node
    ${catkin_LIBRARIES}
)

add_executable(
    path_plan_node
    src/path_plan_node.cpp
)

target_link_libraries(
    path_plan_node
    ${catkin_LIBRARIES}
)
```

```
add_executable(
    bot_control_node
    src/bot_control_node.cpp
)

target_link_libraries(
    bot_control_node
    ${catkin_LIBRARIES}
)
```

16. Modify the `package.xml` file.

    In the package, open `package.xml` file (hint: use command `$ subl`) as follows (refer to the example):

```
# In ~/a345b_path_ws/src/a345b_path
$ subl package.xml
```

    Add `<build_depend>`, `<build_export_depend>` and `<exec_depend>` for each package included in `CMakeLists.txt`.

    The `package.xml` file should look as follows (excluding comments and unnecessary lines):

```xml
<?xml version="1.0"?>
<package format="2">
    <name>a345b_path</name>
    <version>0.0.0</version>
    <description>The a345b_path package</description>

    <maintainer email="maluser@todo.todo">maluser</maintainer>
    <license>TODO</license>

    <buildtool_depend>catkin</buildtool_depend>

    <build_depend>roscpp</build_depend>
    <build_depend>sensor_msgs</build_depend>
    <build_depend>nav_msgs</build_depend>
    <build_depend>geometry_msgs</build_depend>
    <build_depend>std_msgs</build_depend>

    <build_export_depend>roscpp</build_export_depend>
    <build_export_depend>sensor_msgs</build_export_depend>
    <build_export_depend>nav_msgs</build_export_depend>
    <build_export_depend>geometry_msgs</build_export_depend>
    <build_export_depend>std_msgs</build_export_depend>

    <exec_depend>roscpp</exec_depend>
    <exec_depend>sensor_msgs</exec_depend>
    <exec_depend>nav_msgs</exec_depend>
    <exec_depend>geometry_msgs</exec_depend>
    <exec_depend>std_msgs</exec_depend>

    <export>
        <!-- Other tools can request additional information be placed here
        ↪  -->
    </export>
</package>
```

17. Create a launch file to launch all nodes at the same time.

    Inside the `launch` directory, create a file to launch all nodes at the same time (hint: use command `$ touch`). The name can be `start_all_nodes.launch`. In the above example, this can be done as follows:

```
# In ~/a345b_path_ws/src/a345b_path/launch
$ touch start_all_nodes.launch
```

    Once created, open the file (hint: use command `$ subl`) and write the command to launch the 3 nodes and `config.yaml`. Enter the following lines (remember to use **your** package name):

```
<?xml version="1.0"?>

<!--
  EE3305/ME3243
  Name: YOUR NAME
  NUSNET ID: YOUR NUSNET ID
-->

<launch>
  <arg name="student_pkg" value="a345b_path"/>

  <!-- Launch the parameters. -->
  <rosparam command="load" file="$(find a345b_path)/config/config.yaml"/>

  <!-- Launch all nodes in the student's package. -->
  <node pkg="$(arg student_pkg)" type="range_detect_node"
  ↪  name="range_detect_node" output="screen"/>
  <node pkg="$(arg student_pkg)" type="path_plan_node" name="path_plan_node"
  ↪  output="screen"/>
  <node pkg="$(arg student_pkg)" type="bot_control_node"
  ↪  name="bot_control_node" output="screen"/>

</launch>
```

18. Build the package.
    Refer to Step 6.

19. Source own workspace.
    Refer to Step 7.

20. Launch the environment.
    Refer to Step 8.

    A window should appear showing the maze and the robot, although the robot does not yet move.

21. Launch the simulation.
    Open another terminal and **source** your own workspace in the new terminal (refer to Step 7). Launch all nodes from the workspace using command `$ roslaunch package_name launch_file`. In the above example, it will be as follows:

    ```
    # In ~/a345b_path_ws
    $ roslaunch a345b_path start_all_nodes.launch
    ```

    The robot in the maze window should start moving to reach the destination.

22. Enhance the simulation.
    At this stage, you may want to enhance the simulation. Some ideas to enhance the simulation are as follows:

    a) Applying physical parameters of Turtlebot3 to make the simulation much more realistic.

    b) Tuning the PID gains to achieve better navigation (while defining what "better navigation" means).

c) Other ideas that you may have.

23. Capture the video.
    After you have achieved a satisfactory result, capture a short video showing the Turtlebot3 reaching the destination cell. You do not need to record the entire journey of Turtlebot3. Use the recording menu in Ubuntu for better results. In case the record button does not appear, recording with other means, including a camera phone, is acceptable.

24. Explore ROS.
    Open a new terminal while the simulation is running. In the new terminal, use command `$ rqt_graph` to get an overview of the nodes and topics of the simulation. **For your report, select "Nodes/Topics (active)". Take a screenshot, attach it in your report and describe it.**