NATIONAL UNIVERSITY OF SINGAPORE

CEG5303 Intelligent Autonomous Robotic Systems

Project 2: Design and Operate your own dream Mobile Robot (30%)

+Final Presentation (10%)

I. Instructions

1. This project is group work. Students are required to complete it in groups of 3. The tasks include both group tasks and individual tasks that you need to complete independently.
2. Zip all files into one compressed file. Name your file as project2 with your group number. Submit the project on canvas.
   Example: If your group number is 'group1' and then the file name must be project2_group1.zip.
3. Put your names and student ID on the cover page. There should be one report. Recommended pages: 8–16 pages. 1.5-line spacing, 12-point' Times New Roman' font,1-inch margins, upload the zipped file including report in PDF, Simulation videos and your ROS/Python Project file. Include proper citations for all sources.
4. The submission deadline is **23:59 on April 15th 2025**; any submissions after that will be subject to the late submission policy.
5. Any questions, feel free to contact TA Zhang Aoqian, e1144122@u.nus.edu.

II. Objective

· **Installation and use of ROS and related software packages:** You can learn how to install the ROS Noetic version on the Ubuntu system, as well as some dependent software packages, such as dynamixel-sdk, turtlebot3, gmapping, etc. You'll also learn how to use the catkin tool to organize your code and how to run different ROS nodes and launch files.

· **Sensor selection and application:** You can learn how to select appropriate sensors, such as cameras, lidar, inertial measurement units, etc., based on the functions and needs of the robot. You can also learn how to use sensor data to collect and fuse environmental information, and how to communicate and coordinate with other modules.

· **The principles and implementation of SLAM and navigation:** You can learn how to use SLAM (simultaneous localization and mapping) technology to allow robots to position themselves and build environmental maps in unknown environments. You'll also learn how to use navigation technology to allow the robot to plan and execute an appropriate path based on a map and target location, while avoiding obstacles and uncertainty.

· **Principles and Applications of Kalman Filtering:** You can learn how to use

Kalman filtering, a commonly used state estimation algorithm, to deal with sensor noise and inaccuracies to obtain optimal position and velocity estimates for the robot. You can also learn how to use Python to write Kalman filter code, and how to draw and analyze the robot's motion trajectory and speed changes.

III. Task and Guide

Theory requires practice. Now let's implement map building and navigation using our own robot.

i. Install ROS and dependent packages

ROS (Robot Operating System) is an open source meta-operating system for robots. It provides services that an operating system should have, including hardware abstraction, low-level device control, implementation of common functions, inter-process message passing, and package management.

ROS not only provides a rich function package, but also supports code reuse, multi-language implementation, convenient testing and scalability.

There are many versions of ROS. We recommend you install ROS Noetic on Ubuntu 20.04. For installation of other versions, please refer to this website. Then I will guide you to install ROS.

Please follow the steps in this website to install ROS noetic: https://varhowto.com/install-ros-noetic-ubuntu-20-04/#Before_Installing_ROS_Noetic

Then we install Dependent ROS Packages.

```
$ sudo apt-get install ros-noetic-joy ros-noetic-teleop-twist-joy \
  ros-noetic-teleop-twist-keyboard ros-noetic-laser-proc \
  ros-noetic-rgbd-launch ros-noetic-rosserial-arduino \
  ros-noetic-rosserial-python ros-noetic-rosserial-client \
  ros-noetic-rosserial-msgs ros-noetic-amcl ros-noetic-map-server \
  ros-noetic-move-base ros-noetic-urdf ros-noetic-xacro \
  ros-noetic-compressed-image-transport ros-noetic-rqt* ros-noetic-rviz \
  ros-noetic-gmapping ros-noetic-navigation ros-noetic-interactive-markers
```

Catkin is the recommended way to organise your code, it uses more standard CMake conventions and provides more flexibility especially for people wanting to integrate external code bases or who want to release their software. Please refer to this website to install catkin.

ii. Choose the Sensor for your robot.

Individual

In 3.1 and 3.2 we learned and got to know many various sensors, so how should we choose when we actually use them? Now assuming that our robot only has the functions of displacement and rotation, we ultimately need to complete

the functions of map construction and navigation, so what sensors do we need? How do robots sense distance? How to avoid obstacles? How to confirm my current position, speed, acceleration and direction? I hope you can fully consider various issues and emergencies.

You can choose the appropriate sensor type, such as camera, lidar, inertial measurement unit, etc., as well as the specific sensor model and parameters based on the functions and needs of your robot. You can compare the advantages and disadvantages of different sensor models, such as performance, accuracy, price, reliability, compatibility, etc., and why you chose your sensor model. You can describe how your sensors collect and fuse environmental information, and how they communicate and coordinate with other modules, such as SLAM, navigation, Kalman filtering, etc.

This section is **individual**, meaning that each member of the group should choose different sensor. Please do research on the choice of sensor. In the report, the type, model and parameter information of the selected sensor should be clearly stated, and what advantages does this model have compared with other models and what are its price advantages.

iii. Install TurtleBot3 and Gazebo

TurtleBot3 is a popular open source robot designed for use in education, research, product prototyping, and hobby applications. It aims to significantly reduce the size and price of the robot without sacrificing performance, functionality, and quality.

It is a standard ROS platform mobile robot suitable for teaching and scientific research. It can be used to verify SLAM and navigation algorithms.

TurtleBot3 can be used as a simulation tool in the absence of real robots and hardware. We can verify and test algorithms in a simulation environment and then apply them to real robots. For more information about it please refer to this website.

Then we install TurtleBot3 via Debian Packages.

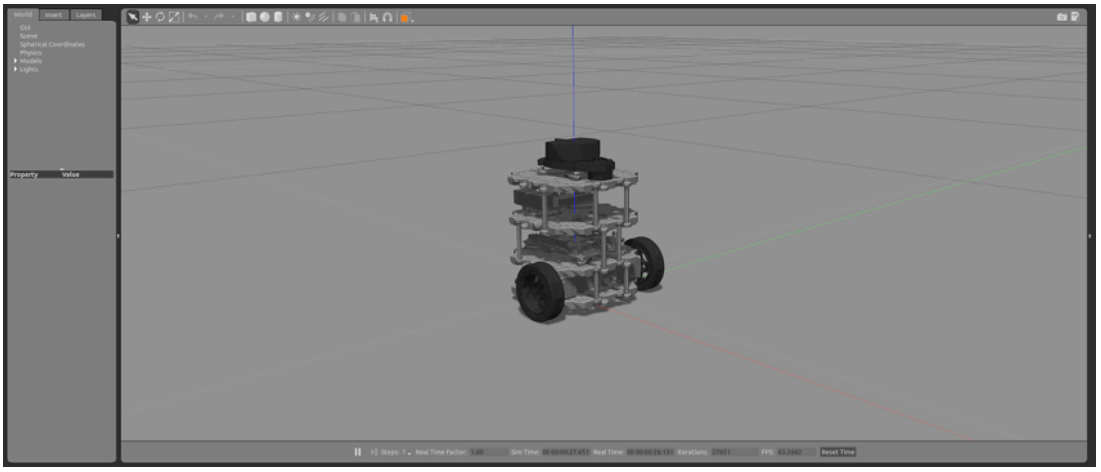
```
$ sudo apt install ros-noetic-dynamixel-sdk
$ sudo apt install ros-noetic-turtlebot3-msgs
$ sudo apt install ros-noetic-turtlebot3
```

The TurtleBot3 Simulation Package requires turtlebot3 and turtlebot3_msgs packages as prerequisite. Without these prerequisite packages, the Simulation cannot be launched.

```
$ cd ~/catkin_ws/src/
$ git clone -b noetic-devel
https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
$ cd ~/catkin_ws && catkin_make
```

Three simulation environments are prepared for TurtleBot3. Please select one of these environments to launch Gazebo.

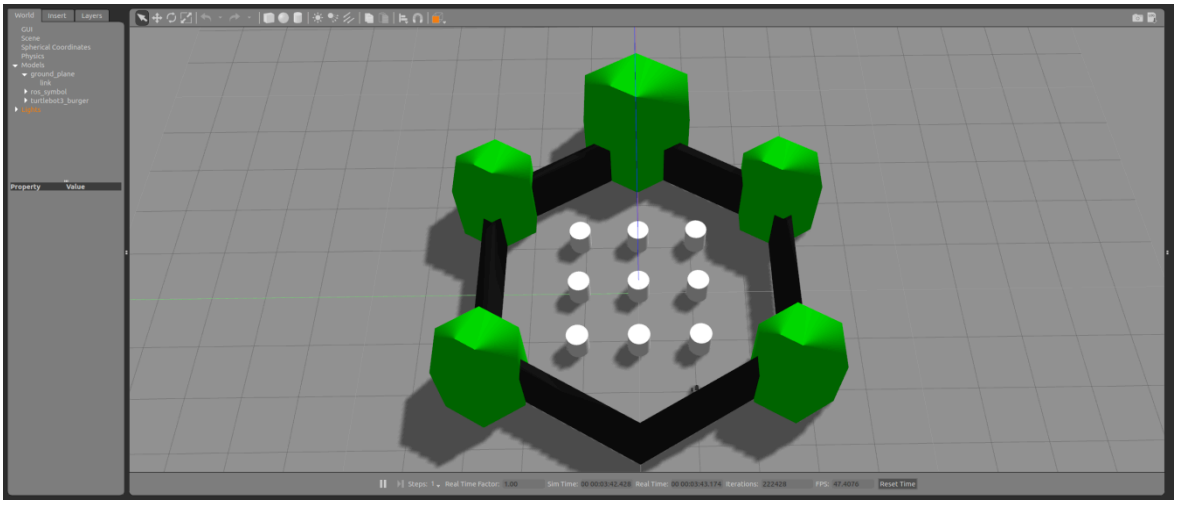
Empty World



$ export TURTLEBOT3_MODEL=burger

$ roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch

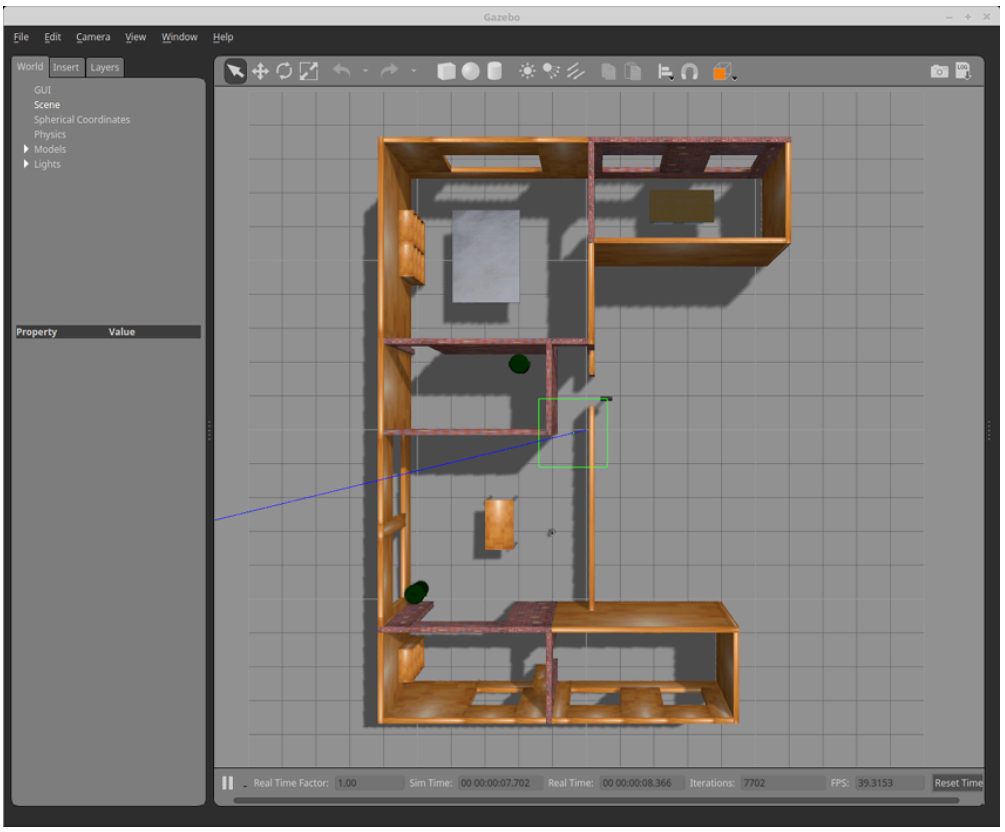
TurtleBot3 World



$ export TURTLEBOT3_MODEL=waffle

$ roslaunch turtlebot3_gazebo turtlebot3_world.launch

TurtleBot3 House



$ export TURTLEBOT3_MODEL=waffle_pi

$ roslaunch turtlebot3_gazebo turtlebot3_house.launch

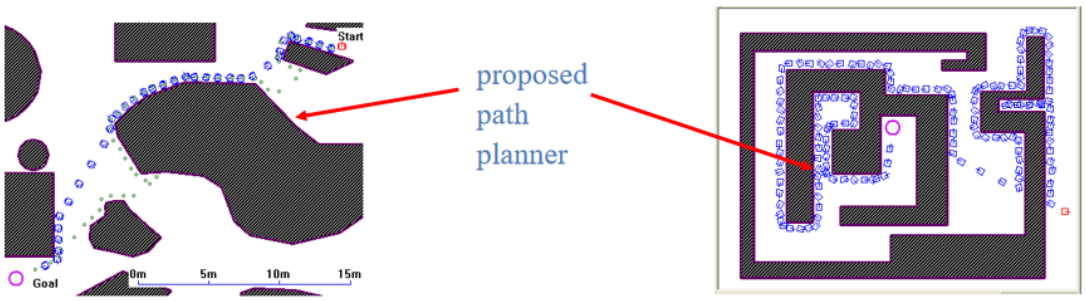
In order to teleoperate the TurtleBot3 with the keyboard, launch the teleoperation node with below command in a new terminal window.

$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch

If you encounter error regarding exit code 255, try killall gzserver to kill all process and relaunch ROS with roslaunch

iv. Build Map Using SLAM

In this step, you need to design the map yourself and construct its 2D floor plan through SLAM. Please note that the map you design should be more complex than the image below. You can use this section to describe the characteristics of the map you design, such as size, shape, obstacles, texture, etc.



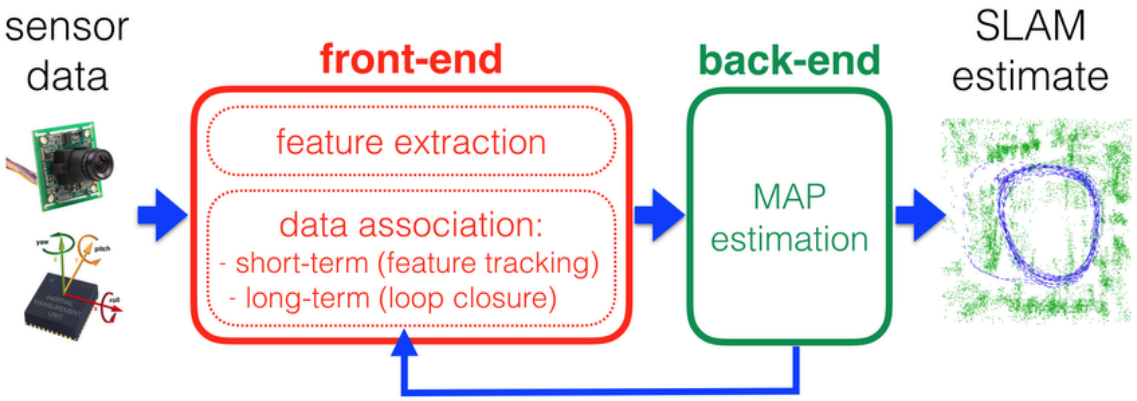
SLAM (Simultaneous Localization and Mapping) is a technology used to simultaneously realize the robot's own positioning and environmental map construction. Its principle is to use sensors such as cameras, lidar, and inertial

measurement units to collect environmental information, and then fuse this information through algorithms to determine the robot's position in the unknown environment and build an environmental map. Through SLAM technology, robots can autonomously explore and navigate in unknown environments.

The working principle of SLAM involves two types of technologies:

Sensor signal processing: includes front-end processing, depending on the type of sensor used.

Pose graph optimization: including back-end processing, independent of sensors.



Here is an example to help you understand how to use SLAM to build a map, and you should apply it to your map.

Please use the proper keyword among burger, waffle, waffle_pi for the TURTLEBOT3_MODEL parameter.

```
$ export TURTLEBOT3_MODEL=burger
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

Open a new terminal from Remote PC with Ctrl + Alt + T and run the SLAM node. Gmapping SLAM method is used by default.

```
$ export TURTLEBOT3_MODEL=burger
$ roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
```

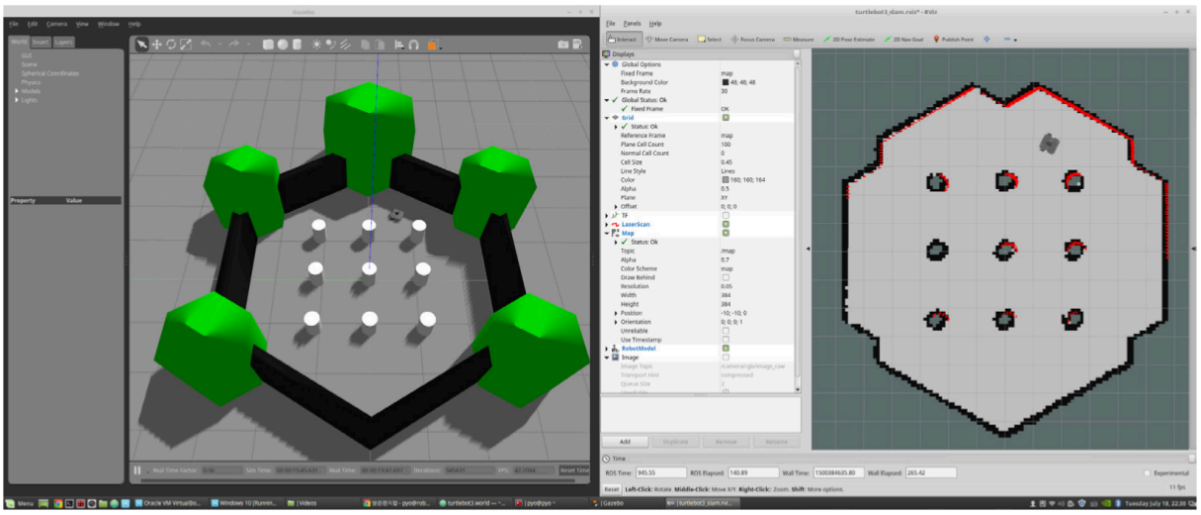
Open a new terminal from Remote PC with Ctrl + Alt + T and run the teleoperation node from the Remote PC.

```
$ export TURTLEBOT3_MODEL=burger
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
 Control Your TurtleBot3!
 ---------------------------
 Moving around:
        w
   a    s    d
        x
 w/x : increase/decrease linear velocity
 a/d : increase/decrease angular velocity
```

space key, s : force stop

CTRL-C to quit

When the map is created successfully, open a new terminal from Remote PC with Ctrl + Alt + T and save the map.



$ rosrun map_server map_saver -f ~/map

Individual

v. Design Navigation Law

Now that you've created your map, the next step is navigation. What do you need to do to make the robot go to the corresponding location according to your instructions?

All the information you have available to you is the information provided to you by the sensor you selected in the second mission. How do you plan your route given the global information? How do you avoid obstacles? How do you determine your current position, speed and direction? Please describe in detail the process or logic of your robot from the starting point to the end point in the form of pseudo-code.

You can refer to some commonly used navigation algorithms, such as A* algorithm, Dijkstra algorithm, RRT algorithm, etc., to design your path planning and obstacle avoidance strategies, or improve and innovate them. You can use mathematical formulas and charts to illustrate the principles and effects of your algorithm, which can more intuitively demonstrate the advantages and limitations of your algorithm.

This task is **individual**, meaning that each member of the group should be different.

vi. Navigation Simulation

Just like the SLAM in Gazebo simulator, you can select or create various environments and robot models in the virtual Navigation world. However, a

proper map needs to be prepared before running the Navigation.

In the previous SLAM section, TurtleBot3 World is used to create a map. The same Gazebo environment will be used for Navigation.

Please use the proper keyword among burger, waffle, waffle_pi for the TURTLEBOT3_MODEL parameter.

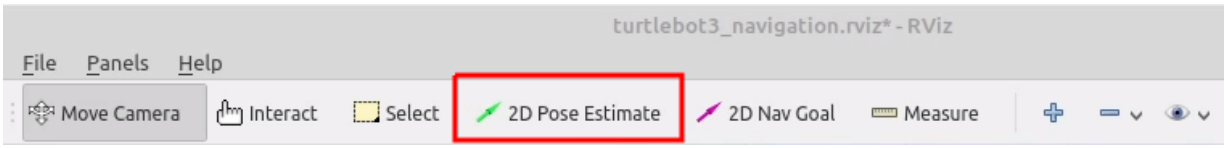
```
$ export TURTLEBOT3_MODEL=burger
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

Open a new terminal from Remote PC with Ctrl + Alt + T and run the Navigation node.

```
$ export TURTLEBOT3_MODEL=burger
$ roslaunch turtlebot3_navigation turtlebot3_navigation.launch
map_file:=$HOME/map.yaml
```

Initial Pose Estimation must be performed before running the Navigation as this process initializes the AMCL parameters that are critical in Navigation. TurtleBot3 has to be correctly located on the map with the LDS sensor data that neatly overlaps the displayed map.

Click the 2D Pose Estimate button in the RViz menu.

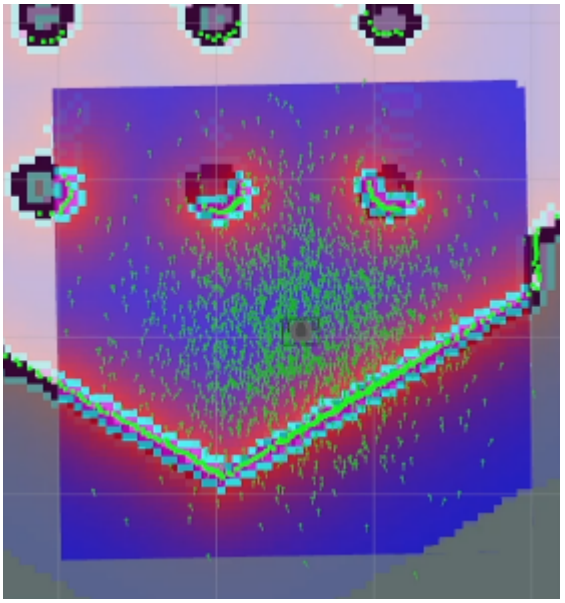


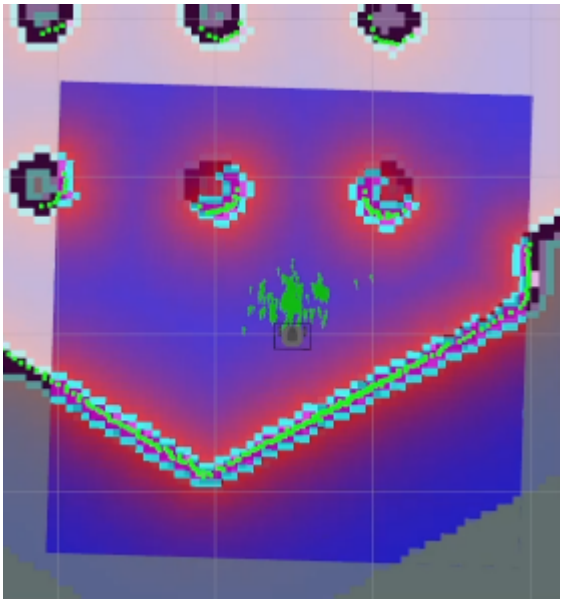
Click on the map where the actual robot is located and drag the large green arrow toward the direction where the robot is facing.

Launch keyboard teleoperation node to precisely locate the robot on the map.

```
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

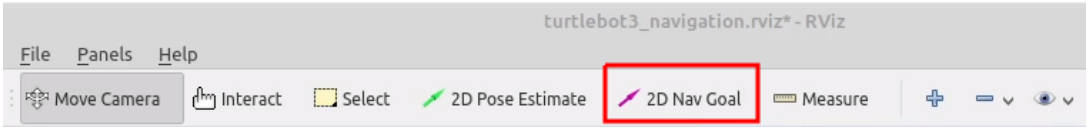
Move the robot back and forth a bit to collect the surrounding environment information and narrow down the estimated location of the TurtleBot3 on the map which is displayed with tiny green arrows.





Terminate the keyboard teleoperation node by entering Ctrl + C to the teleop node terminal to prevent different cmd_vel values are published from multiple nodes during Navigation.

Click the 2D Nav Goal button in the RViz menu.



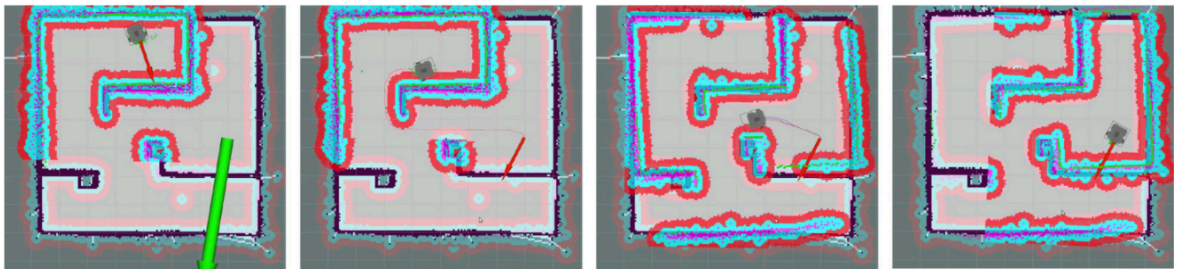
Click on the map to set the destination of the robot and drag the green arrow toward the direction where the robot will be facing.

This green arrow is a marker that can specify the destination of the robot.

The root of the arrow is x, y coordinate of the destination, and the angle θ is determined by the orientation of the arrow.

As soon as x, y, θ are set, TurtleBot3 will start moving to the destination

immediately.



vii. **Kalman Filter in Navigation**

If we could get all the accurate data, we could accomplish our navigation easily. But in fact, the data transmitted from the sensor is not necessarily accurate.

Now we assume that our robot is equipped with a GPS sensor, through which we can obtain the real-time position (x-y) of the robot. These precise data can be obtained from the simulation of the previous task.

Then we add Gaussian noise to our observation data. Please specify the noise you added. The starting point of the robot is at the origin, please clarify its initial direction and final direction.

Kalman filter uses the dynamic information of the target to try to remove the influence of noise and obtain a good estimate of the target position. This estimate can be an estimate of the current target position (filtering), or an estimate of the future position (prediction).

You need to describe the algorithm steps of your Kalman filter in detail, including initialization, prediction, update, and related mathematical formulas and matrix operations. You also need to explain how the parameters of your Kalman filter are selected, such as state transition matrix, observation matrix, process noise covariance matrix, observation noise covariance matrix, etc.

You need to display and analyze the results of your Kalman filter, including drawing the original coordinate trajectory, the trajectory after adding noise, and the optimal estimated trajectory after Kalman filter processing. You also need to plot the velocity change curves in the x and y directions, and compare the differences and pros and cons of the three situations. You can also calculate and display the error and accuracy metrics of your Kalman filter, such as root mean square error, Kalman gain, etc. Python is recommended. You may use any open sourced code for this task.

viii. Analyse and Think Individual

Do your map construction and localization systems accurately reflect the characteristics of the environment and the robot's position? Can your system adapt to different environments and robot models? Can your system handle sensor noise and uncertainty?

Can your Kalman filter effectively eliminate noise in the observation data and obtain optimal position and velocity estimates? Can your Kalman filter adapt to different initial conditions and noise parameters? Does your Kalman filter work well with your navigation algorithm?

Is your navigation algorithm able to plan an appropriate path based on the map and target location while avoiding obstacles and uncertainty? Can your navigation algorithm adapt to different maps and target locations? Does your navigation algorithm enable real-time feedback and adjustments?

Are you able to support your analysis and discussion with data and graphs? Are you able to measure the performance and robustness of your system using appropriate evaluation metrics and methods? Are you able to compare and analyze with other systems or methods?

Can you use artificial intelligence methods to replace or improve your Kalman filter and navigation algorithms? Can you explain the rationale and advantages of your chosen AI approach? Were you able to demonstrate experimentally or theoretically the performance and robustness of your AI approach?

This part is **individual.**

IV. Score Distributions

The distribution of the scores for the task is listed below.

Score Distributions

Task	Score Distribution
i	5%
ii	10%(individual)
iii	5%
iv	25%
v	10%(individual)
vi	15%
vii	25%
viii	5%(individual)

V. Final Presentation

In week 12 and 13 of this semester, you are required to give a 6 minute presentation in groups to illustrate what you learned from this course and your outlook for the future development of intelligent autonomous robotic systems. This part will count for 10% in your final grade.

Requirements:

1. Your presentation should contain the following contents:

a. What you learned from this course.

b. Your outcome and findings in mini-Lab, project1 and project2.

c. Your points of view about current intelligent autonomous robotic systems like chatgpt, sora, Apple Vision Pro, etc.

d. your outlook for the future development of intelligent autonomous robotic systems.

2. Each group is required to make a PPT and every member needs to do the presentation (Try to distribute everyone's speaking time evenly). **Group 1 - 25 will present on week 12, and group 26 - 50 will present on week 13.**

3. The time of this presentation needs to be strictly limited to 6 minute.

Reference

ROBOTIS e-Manual: https://emanual.robotis.com/

GMapping SLAM: https://openslam-org.github.io/gmapping.html

GMapping Package: http://wiki.ros.org/gmapping

Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard: Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters, IEEE Transactions on Robotics, Volume 23, pages 34-46, 2007 (link)

Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard: Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling, In Proc. of the IEEE International Conference on Robotics and Automation (ICRA), 2005 (link)