SLAM and Navigation

Objectives

Modified from CMSC 20600

Your goal in this project is to experience building your own world, build a map using SLAM, and then navigate Turtlebot3 in the map you built, that is, give your robot a goal location and have your robot automatically move there. If you find yourself stuck on something, email jason27146913@gmail.com or join the TA office hour.

Learning Goals

- Learn how to build your own gazebo world
- Use SLAM to build maps
- Learn how to navigate to a desired location on the map

Writeup

Please add the `README.md` file as your write-up for this project. Please add pictures, Youtube videos, and/or embedded animated gifs to showcase and describe your work. Your writeup should:

- **Room Design** (1 paragraph): Describe how you designed your room.
- **Challenges** (1 paragraph): Describe the challenges you faced while running SLAM and how you overcame them.
- **Future work** (1 paragraph): If you had more time, what do you think could be improved?

Gif/MP4

At the beginning or end of your writeup please include two gif/mp4 files, the first one is a video of you using SLAM to find the map, and the second is a video of you using navigation.

Note that, unfortunately, there are no recording options directly on rviz. If you are running on Windows/MacOS with NoMachine, we recommend that you record your NoMachine's rviz window directly from your base OS since CSIL machines don't come with screen recorders.

If you are on MacOS, QuickTimePlayer would be a great out-of-the-box option for screen recording. On Windows, a few options work great such as OBS Studio, or the Windows 10 builtin Game bar. On Ubuntu, one option would be Simple Screen Recorder

Creating & Mapping Gazebo Environments Using SLAM

In this section, you will

1) build a new world in Gazebo for your Turlebot3 to navigate,

2) use SLAM to generate a map of the world you've created, and

3) save & load the map - visualizing it in RViz.

Setup

To start out today's lab, we'll want to create a new ROS package:

```
$ cd ~/catkin_ws/src

$ catkin_create_pkg lab2_slam rospy std_msgs geometry_msgs sens
or_msgs

$ cd ~/catkin_ws && catkin_make

$ source devel/setup.bash

$ sudo apt-get update

$ sudo apt-get install ros-noetic-dwa-local-planner

$ sudo apt-get install ros-noetic-gmapping
```

Just to keep everything organized, we'll want to create directories called `launch`, `map`, `worlds` within the `lab2_slam` directory:

```
~/catkin_ws/src/lab2_slam/launch

~/catkin_ws/src/lab2_slam/map

~/catkin_ws/src/lab2_slam/worlds
```

Building & Saving a New Gazebo World

Your first objective is to build a new Gazebo world. We'll use the building editor to create an **enclosed room** (that's the only constraint we'll ask you to follow) - anything else you want to add is up to you!

Please start up Gazebo (run `gazebo` in a new terminal) and utilize the building editor (here's a helpful page detailing how to use the building editor) to build your own custom Gazebo world. You can see what using the building editor should look like in the gif.

Once you've finished creating your enclosed room (in whatever shape and texture you like):

1. Save your enclosed room model by clicking on `File/Save` (you can name it whatever you like)
2. Exit the building editor - `File/Exit Building Editor`
3. Save your Gazebo world by clicking on `File/Save World As`. You can name it whatever you like as long as:
    - You don't include any whitespaces in your world filename
    - You use the `.world` file extension
    - You put it within the `~/catkin_ws/src/lab2_slam/worlds` directory
4. Exit Gazebo (`Ctrl+C` in the terminal where you ran `gazebo`)

Incorporating Your New Gazebo World into a New ROS Lauchfile

What we want to do next is start up a world file that has our Turtlebot3 within it, so we can use SLAM to map the environment. We'll do this by building a **roslaunch file**. [roslaunch] is a tool that easily enables you to launch multiple ROS nodes from the same terminal and also set various parameters.

Copy and paste the following code and put it in a file named `turtlebot3_in_enclosed_room.launch` within the `~/catkin_ws/src/lab2_slam/launch` directory. **The one edit you'll want to make to this file is replacing `your_gazebo_world_name.world` with the name of the world file you saved in the last step.**

```
<launch>

  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="mode
l type [burger, waffle, waffle_pi]"/>

  <arg name="x_pos" default="0.0"/>

  <arg name="y_pos" default="0.0"/>

  <arg name="z_pos" default="0.0"/>


  <include file="$(find gazebo_ros)/launch/empty_world.launch">

    <arg name="world_name" value="$(find lab2_slam)/worlds/your
_gazebo_world_name.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>


  <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" a
rgs="-urdf -model turtlebot3_$(arg model) -x $(arg x_pos) -y $
(arg y_pos) -z $(arg z_pos) -param robot_description"/>


</launch>
```

Now we'll go through each part of this roslaunch file to understand what's going on:

- First, for roslaunch files, we use the XML format and always start and end roslaunch files with the `<launch>` and `</launch>` tags at the beginning and end of our file.
- Next we define several arguments for our roslaunch file: `model`, `x_pos`, `y_pos`, and `z_pos`:

```
- <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="
  model type [burger, waffle, waffle_pi]"/>
- <arg name="x_pos" default="0.0"/>
- <arg name="y_pos" default="0.0"/>
```

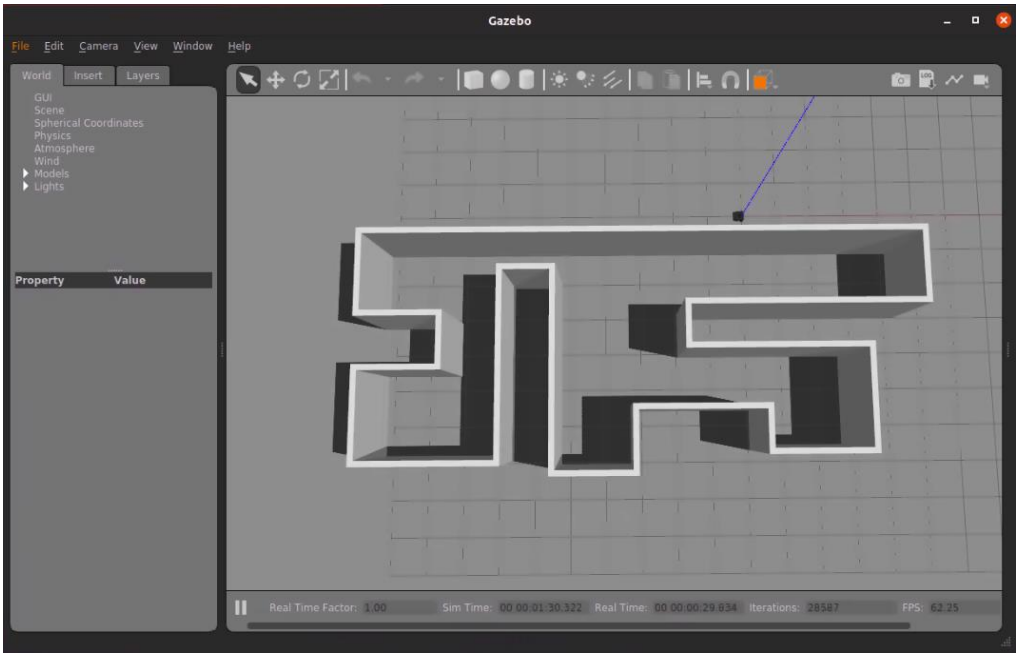
```
<arg name="z_pos" default="0.0"/>
```

Each of these arguments has a default value, where if they are not specified in the terminal when the roslaunch file is run, those arguments will take on the default value. If you want to specify the arguments in the terminal when you run roslaunch, it would look like this:

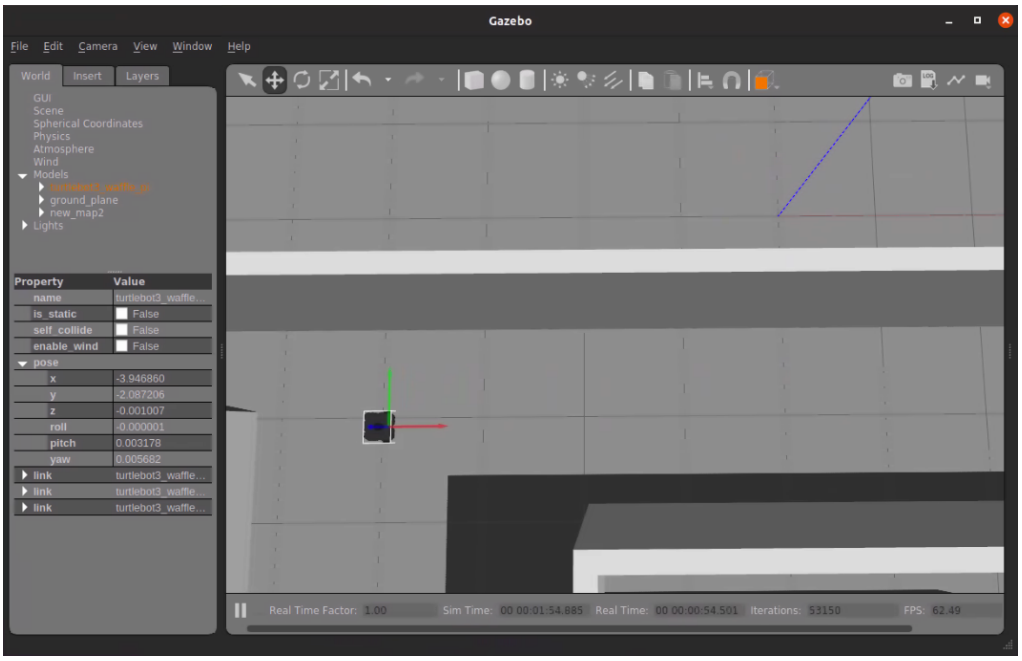
```
roslaunch lab2_slam turtlebot3_in_enclosed_room.launch
model:=waffle_pi x_pos:=1.0 y_pos:=-1.0 z_pos=0.0
```

The arguments specified in this roslaunch file set the type of Turtlebot3 model we're working with (we already have the `TURTLEBOT3_MODEL` environment variable set to `waffle_pi`) as well as the x, y, and z specifications for where the Turtlebot3 robot should spawn in the Gazebo world.

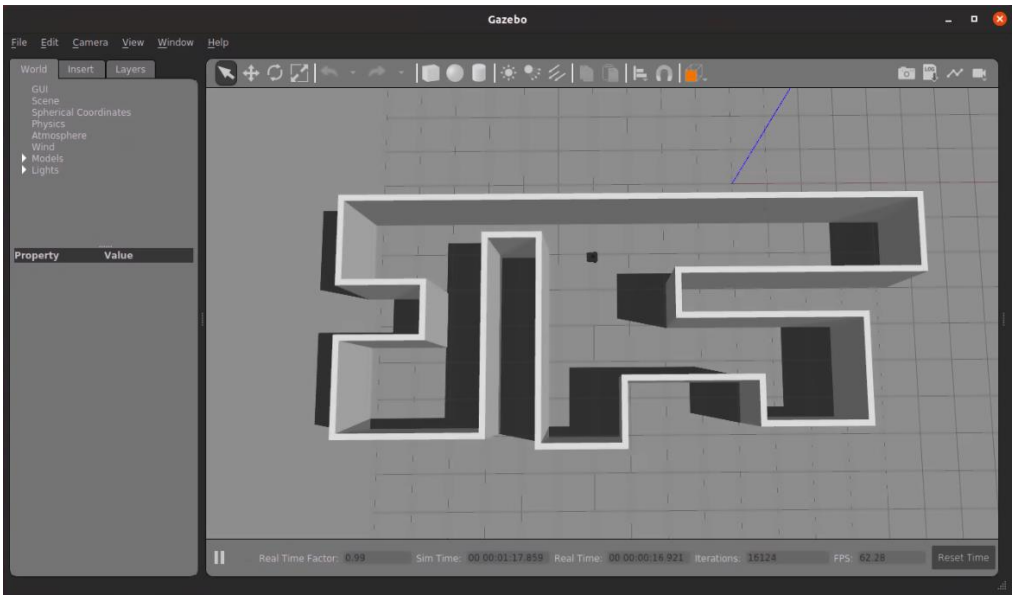
Note: You may notice when you run the roslaunch file that your robot is not within your enclosed space:



If you observe this, you can manually move your Turtlebot3 into the room and examine it's new pose (see the panel view on the left of the screenshot below). In this example, the robot location is `x = -3.95`, `y = -2.09`, `z = 0.0`.



Equipped with this new pose information, I can modify the roslaunch arguments of `x_pos`, `y_pos`, and `z_pos` to ensure that when I run the roslaunch file, my Turtlebot3 appears within the enclosed space (see below).



- Next, the `<include>` tag serves the purpose of launching a different roslaunch file (`empty_world.launch` from the [built-in ROS simulator library ROS packages]) from within this roslaunch file.

```
<include file="$(find gazebo_ros)/launch/empty_world.launch">
  <arg name="world_name" value="$(find lab2_slam)/worlds/your_gazebo_world_name.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>
```

When we launch `empty_world.launch`, we also give it several arguments, including the name of the world file we want it to launch. In order to make this file runnable from any directory, we use a handy `$(find lab2_slam)` command to grab the full file path of the `lab2_slam` ROS package. From the ROS package path, we specify the path to the world file that we

created: `/worlds/your_gazebo_world_name.world`.

- Finally, the `<param>` and `<node>` tags within this roslaunch file enable the Turlebot3 robot to 1) spawn within the Gazebo world and 2) start publishing and subscribing to all of the topics we expect it to (e.g., `/cmd_vel`, `/scan`).

```
- <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_pos) -y $(arg y_pos) -z $(arg z_pos) -param robot_description"/>
```

The file specifying the robot parameters, `turtlebot3_waffle_pi.urdf.xacro`, is located within the ROS packages we cloned when we setup our computer environment for Turtlebot3 robot programming (if you're really curious, you can check out the file's contents on ROBOTIS-GIT's git repo). It's also worth pointing out that the `<node>` tag is used when you want to run a ROS node (when you would want to `rosrun` in a new terminal). Each ROS node must have a unique `name` parameter and you must specify the ROS package the script (specified by the `type` parameter) is found within (in this case it's found in the `gazebo_ros` package). You can also see that we can pass arguments into this ROS node, where we're passing in the arguments we set at the top of the roslaunch file as well as the `robot_description` parameter.

Note: Let's say that you write a Python script `my_ros_node.py` within the `lab2_slam` ROS package and you want to add it to your launchfile. You would add the following line to your launchfile:

```
<node pkg="lab2_slam" type="my_ros_node" name="my_ros_node_1"/>
```

While the `pkg` and `type` fields are specific to the ROS package and Python script you want to run, the `name` field can be anything you

want (as long as no other ROS node running on the same network has the same `name`).

Using SLAM to Create a Map of Your Gazebo Environment

The Turtlebot3 ROS packages have nodes that can perform SLAM and output a map of an environment that the Turtlebot3 navigates. For more details on executing Turtlebot3's SLAM nodes, feel free to check out the Turtlebot3 SLAM documentation.

Now, let's run SLAM and generate a map of your Turtlebot3's environment. Do this by executing the following:

Terminal 1: `roscore`

```
roscore
```

Terminal 2: Launch the roslaunch file we've been working on

```
roslaunch lab2_slam turtlebot3_in_enclosed_room.launch
```

Terminal 3: Run the slam node

```
roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=
gmapping
```

Terminal 4: Teleoperate the Turtlebot3 around the environment until you get a complete map

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

Terminal 5: **After you have created a full map of your environment** (exploring the entire space), you can save your map (please save it in the `~/catkin_ws/src/intro_robo/lab2_slam/map` directory) using the following command (do not add a file extension, it will do so automatically):

```
rosrun map_server map_saver -f filepath_and_filename
```

Here's an example of what you should see in RViz when you're generating your map (my computer was lagging a bit during the recording of this gif, yours should look a bit more continuous):

Loading a Saved Map

If you want to load a map that you've saved and visualize it in RViz, you'll need to create a new roslaunch file (`visualize_map.launch`) that contains the following:

```
<launch>


  <arg name="open_rviz" default="true"/>

  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="mode
l type [burger, waffle, waffle_pi]"/>

  <arg name="map" default="$(find lab2_slam)/map/your_map_file.
yaml" />


  <!-- Map server -->

  <node pkg="map_server" name="map_server" type="map_server" ar
gs="$(arg map)"/>


  <!-- Run a transformation between the map and odom frames -->

  <node pkg="tf" type="static_transform_publisher" name="link1_
broadcaster" args="0 0 0 0 0 0 /map /odom 100" />


  <include file="$(find turtlebot3_bringup)/launch/turtlebot3_r
emote.launch">

    <arg name="model" value="$(arg model)"/>
```

```
  </include>


  <!-- rviz -->
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(find turt
lebot3_gazebo)/rviz/turtlebot3_gazebo_model.rviz"/>


</launch>
```

In the launch file code above, you'll either need to 1) change your default map location in your launchfile code to the location where you saved your map or 2) set the map argument when you run the launchfile, e.g., `roslaunch lab2_slam visualize_map.launch map:=your_map_location`.

Once you've saved `visualize_map.launch`, you'll need to have running in your terminals:

Terminal 1:

```
roscore
```

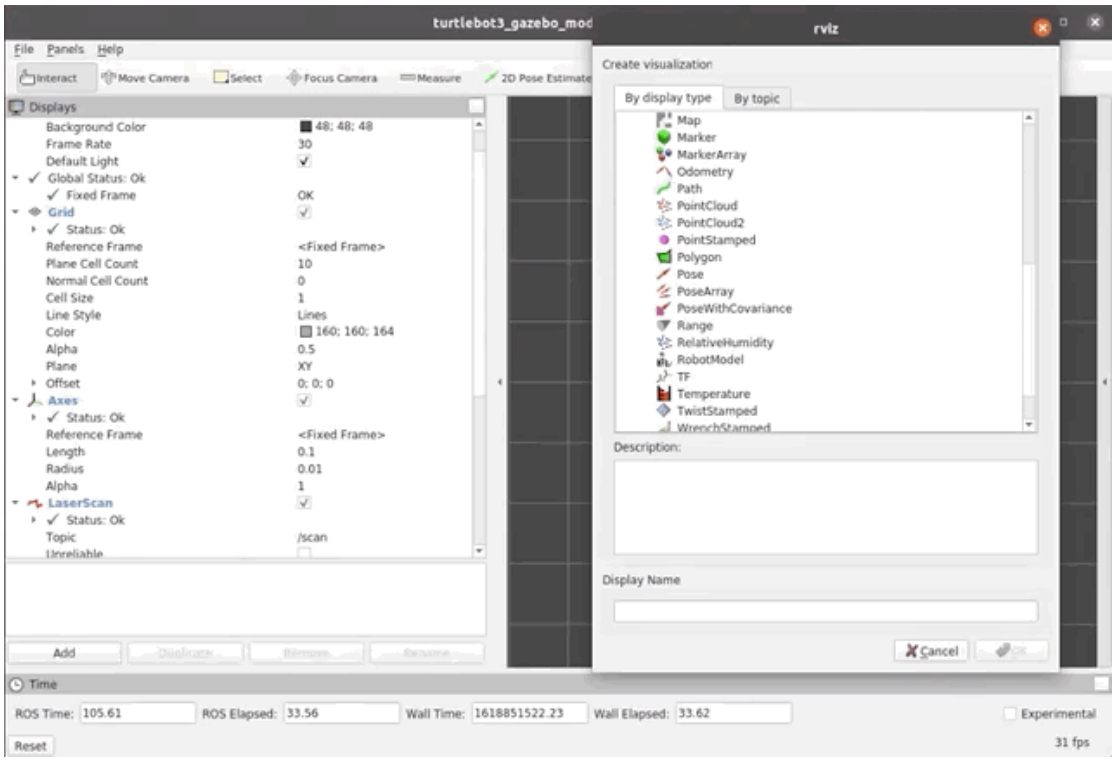
Terminal 2:

```
roslaunch lab2_slam turtlebot3_in_enclosed_room.launch
```

Terminal 3:

```
roslaunch lab2_slam visualize_map.launch
```

Once you have RViz open, you will also need to add a map object and have it subscribe to the `/map` topic in order to visualize the map. To do that you'll need to hit the `Add` button at the bottom left and specify the `/map` topic (see gif).



Navigation to a Goal Location

Once you have successfully localized your robot, you can also give your robot a goal location and have your robot move there (with the help of some Turtlebot3 navigation libraries). **Your goal is to get out of the house (navigate out the front door) once your robot has localized itself.** Here's how to do it:

Terminal 1:

```
roscore
```

Terminal 2:

```
roslaunch lab2_slam turtlebot3_in_enclosed_room.launch
```

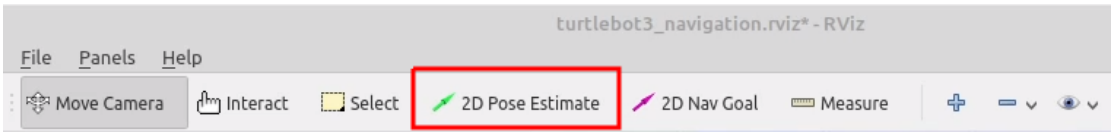
Terminal 3:

```
roslaunch turtlebot3_navigation turtlebot3_navigation.launch ma
p_file:=$(rospack find lab2_slam)/map/your_map_path.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.

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

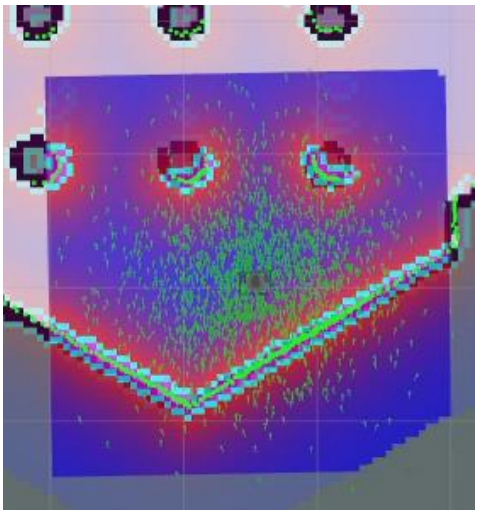


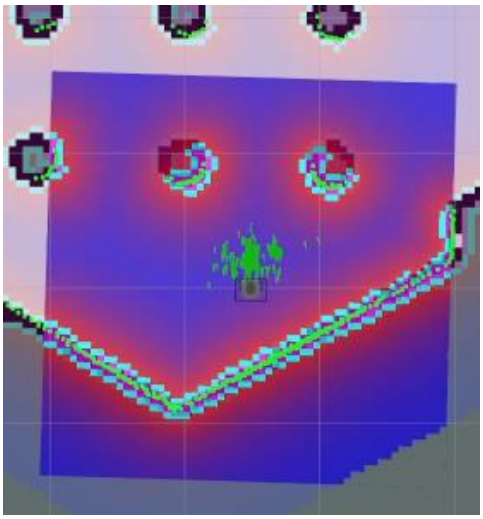
2. Click on the map where the actual robot is located and drag the large green arrow toward the direction where the robot is facing.
3. Repeat step 1 and 2 until the LDS sensor data is overlayed on the saved map.
4. Launch keyboard teleoperation node to precisely locate the robot on the map.

```
$ roslaunch turtlebot3_teleopturtlebot3_teleop_key.launch
```

5. 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.

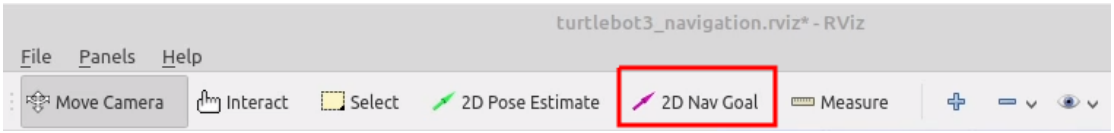
   Terminate the keyboard teleoperation node by entering `Ctrl` + `C` to the teleop node terminal in order to prevent different **cmd_vel** values are published from multiple nodes during Navigation.





Set Navigation Goal

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



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

Grading

The Particle Filter Project will be graded as follows:

- 30% Writeup
    - 5% Room Design
    - 15% Challenges
    - 10% Future Work
- 70% MP4/GIF
    - 35% SLAM
    - 35% Navigation

Submission Deadline and Method

Language : python3

Packages : If you use any package, please mention it in README.md

Deadline : Thu 06 Apr 2023 13:00 (UTC+8)

Delay policy : One Day -> points * 0.7
Two Days -> points * 0.5
After Two Days -> 0 point

Everyone has a chance for a one-day late submission this semester!!!

Submission Method:

Upload a zip file to CGU Teams with the format :

m1234567_lab2.zip
+ m1234567_lab2
- launch
+ turtlebot3_in_enclosed_room.launch
+ visualize_map.launch
- map
+ $(name_of_your_map_file).yaml
+ $(name_of_your_map_file).pgm
- worlds
+ $(name_of_your_world_file).world
+ README.md
+ slam.mp4/gif
+ navigation.mp4/gif
+ other files in the lab2_slam folder

**Do not include other files other than those mentioned above*

All file names should be **only zip files (No rar and 7zip)*

*****Incompatible format will not be graded.***

Notice : Zero points for plagiarism!!!
(either from the internet or copy from classmate)