Homework #6

due Wednesday 2/14/24 11:59pm

This week, I would like to explore mapping using ROS. ROS started with exactly these types of mobile robot applications - navigation, localization, mapping, etc. So it is useful to know and could be a platform for projects.

The set uses Gazebo to simulate a mobile robot including a laser scanner, which lets us drive around and build maps. I know a few folks (using Macs with the M1/2 chips) are unable to install/run Gazebo. If you can't run Gazebo (or it is simply too slow or you don't want to drive the robot), we also have several pre-recorded simulations that you can play back (without Gazebo). That means you won't be able to control how the robot drives, but you can still build maps in ROS.

We are providing skeleton and support code in the form of a ROS package `turtlebot`. As with all our ROS packages, please place under `~robotws/src`. And then build in the `~/robotws` folder:

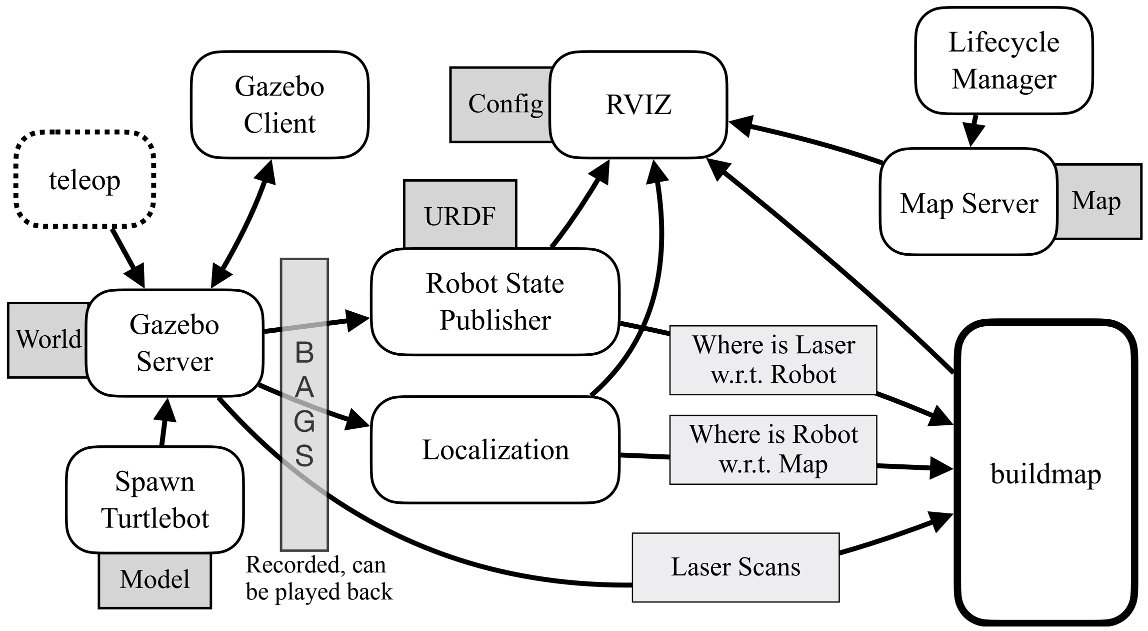
```
cd ~/robotws
colcon build --symlink-install --packages-select turtlebot
source install/setup.bash
```

where the `--packages-select` argument means you are only building that package (faster). And the last command tells the prompt where to find ROS files it hasn't seen before. So it will only be necessary the first time - not necessary if only file contents has changed.

Everything is started with a single launch file `turtlebot.launch.py`. So you can run things using

```
ros2 launch turtlebot turtlebot.launch.py
```

Or use `turtlebot_nogazebo.launch.py` if you are unable to run Gazebo. Please take a look at the launch file. We will ask you to edit this depending on the problem below. It sets up the following system of ROS nodes:



Changing the Launch File

Within the launch file, the first section locates/selects the files to define: (1) the Gazebo world, (2) the Gazebo robot model, (3) the URDF, (4) the pre-saved good map, (5) the RVIZ configuration, and (6) the ROS bag to be played if Gazebo doesn't work. You may need to change some of these.

The second section defines all elements, a subset of which the last section then includes to be launched. You will need to **comment/uncomment the elements in the last section**:

(a) *Simulation.* The Gazebo server `gzserver` is the main simulation engine, reading a world file. You will also need to spawn one robot in the world. Choose either the robot with a clean lidar or with a noisy lidar. Replace with playback if you are skipping Gazebo.

(b) *Visualization.* You can directly view the simulation using the Gazebo client `gzclient` but that does not show the map. Or use RVIZ to visualize the robot, the scans, and the map. You could start both, but they are both graphics-intense and will slow down the computer.

(c) *Transforms.* ROS locates everything by placing frames on all objects (using the ROS TF library). The `robot_state_publisher` reads the URDF file and places the lidar sensor relative to the robot's base. The localization tracks the robot is relative to the map. You can run either perfect localization or a noisy version.

(d) *Map.* To publish a known map (and see in RVIZ), you may use the `map_server` together with a `lifecycle` manager (necessitated by the ROS navigation stack). OR, as we are building our own maps, start your `buildmap` node instead.

NOTE, **after you change the launch file, you will have to rebuild** using the above command.

Avoiding Gazebo

If you can not (or opt not to) run Gazebo, then do *not* launch the Gazebo server, nor the spawn nodes or the Gazebo GUI/client. Instead select one of the pre-recorded simulations (in the first section) and include the playback in the last section. The recordings provide the turtlebot with a clean/perfect lidar or a noisy lidar. The drive through a `singleroom`, the `leftside` house, or the `rightside` of the house. Or they drive through the `alternate` house (for the last problem).

If you would like to record your own runs, the record and playback commands are:

```
ros2 bag record --use-sim-time -o NAME_OF_RECORDING /joint_states /odom /scan /tf
ros2 bag play --clock 10  NAME_OF_RECORDING
```

where the `--use-sim-time` and `--clock` are needed as we want to record/replay the simulator time instead of normal time. And the topics to be recorded are `/joint_states`, `/odom`, `/scan`, and `/tf`.

Driving the Robot

If you are using Gazebo, you will need to drive the robot. For this, in a separate terminal (separate window or tab), run:

```
ros2 run turtlebot teleop
```

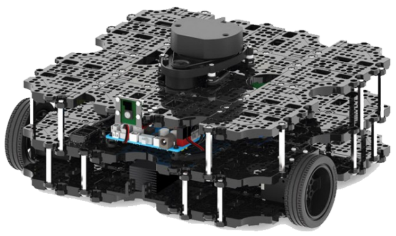
This will read the keyboard and send ROS messages with the appropriate driving commands. As it uses the keyboard, it must have its own window and can *not* be part of the launch file.

Problem 1 (Welcome to Gazebo and Turtlebot) - 7 points:

First, let's view the Gazebo simulator by itself. In particular, we are going to simulate a Turtlebot, which is supported "out of the box". The real versions are made by Robotis

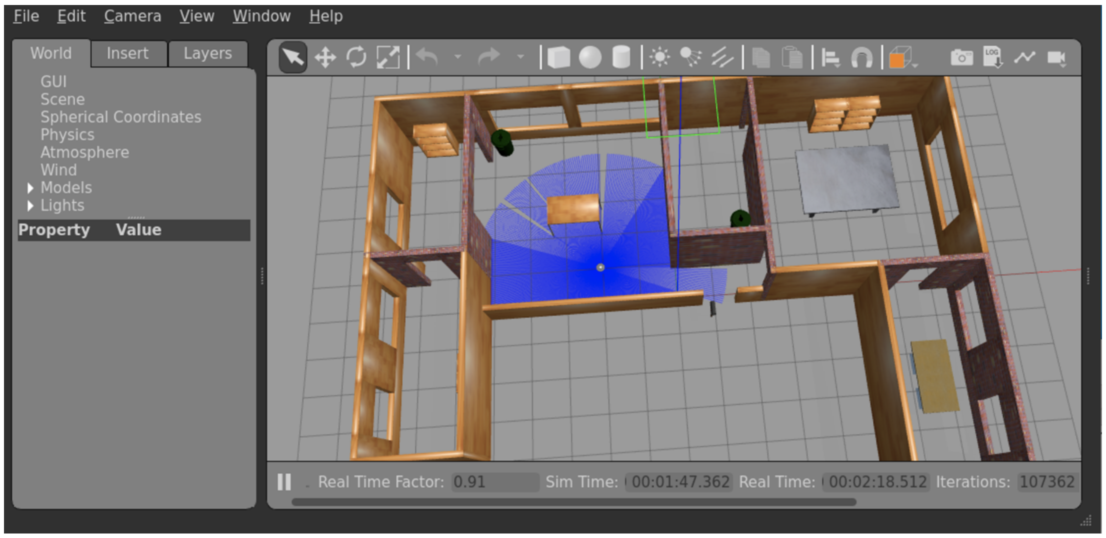
`https://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/#gazebo-3d`

Turtlebots are capable but simple little mobile robots without arms. We'll focus on the 3rd generation (`turtlebot3`) "waffle" model, which includes a regular and depth camera, as well as a laser range finder.



Please edit the launch file to start only `gzserver`, `gzclient`, and one of the `spawn` nodes. Rebuild and then run the launch command (see above). Note Gazebo may take a long time to start (up to 3 or more minutes!), as it needs to download the house model the first time. Subsequent starts will be faster. Finally, open a second terminal to drive using the keyboard (again see above).

You should see the Gazebo GUI showing the house and the robot. Note in particular the rays of the laser scanner. And, if you zoom in, you will see the robot (faint under the rays).



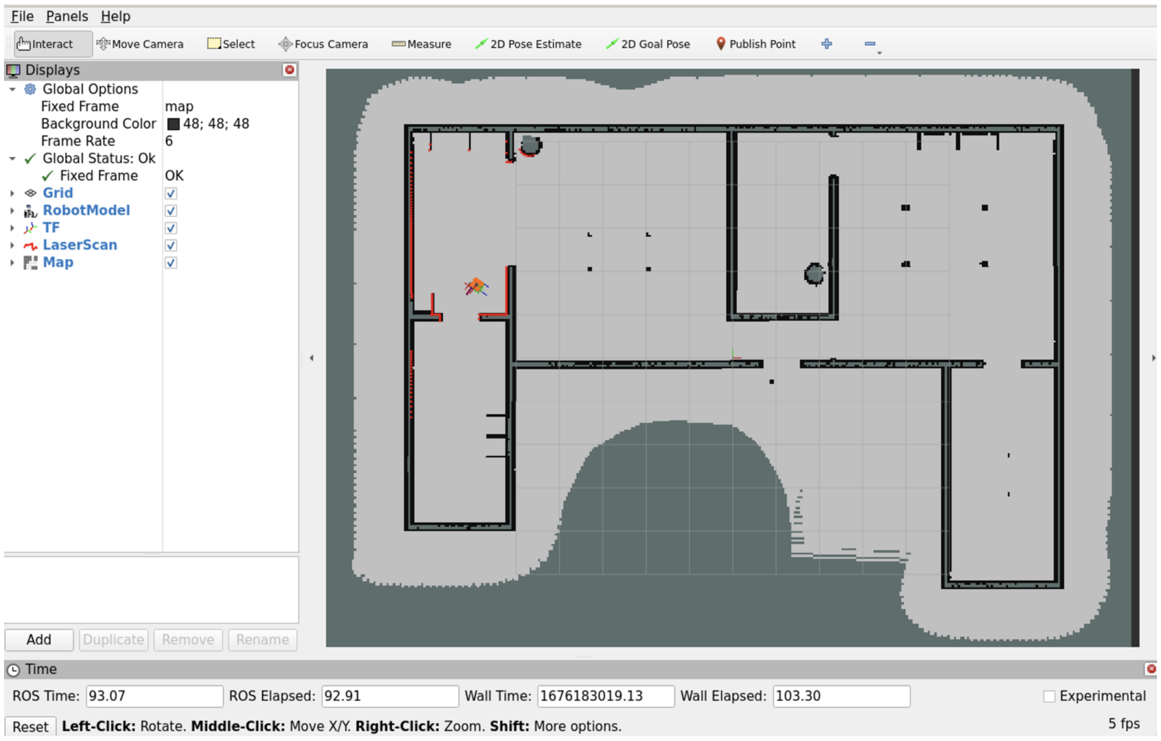
Drive around and get a feel for the UI. Please submit a screen shot with the robot in a different room. **If you are unable to use Gazebo, let us know and skip to the next problem, always substituting the playback command for Gazebo.**

Problem 2 (Visualize in RVIZ) - 7 points:

While the Gazebo GUI is nice, we will switch to RVIZ as this allows us to also visualize the map. To start we will use the known (good) map. So please edit the launch file to

- launch the Gazebo server and spawn the turtlebot with clean/perfect lidar
- not launch the Gazebo client
- launch RVIZ and the robot state publisher
- launch the perfect localization
- and launch the map server (together with the lifecycle manager).

Please rebuild, launch, and again drive around (using a second terminal, see above). Notice the laser scans should pick up the walls perfectly.



Global Options
Fixed Frame map
Background Color ■ 48; 48; 48
Frame Rate 6
Global Status: Ok
Fixed Frame OK

ROS Time: 93.07 ROS Elapsed: 92.91 Wall Time: 1676183019.13 Wall Elapsed: 103.30

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:** Zoom. **Shift:** More options.

5 fps

If you are interested, try the commands (in a different terminal), which are hopefully self-explanatory?

```
ros2 service call /pause_physics   std_srvs/srv/Empty
ros2 service call /unpause_physics std_srvs/srv/Empty
ros2 service call /reset_world     std_srvs/srv/Empty
```

Again, please submit a screen shot with the robot in a different room

Problem 3 (Running your Code) - 10 points:

Now let's get down to business. In the launch file, comment out the map server (and lifecycle manager) and instead launch you `buildmap` node. Then edit the `buildmap.py` file! Notice:

(i) The code sets up a 18m by 12m map, with 5cm resolution. For this it uses a grid (NumPy array) of 360 by 240 elements being Log-Odds-Ratios. Specifically, the variable `self.logoddsratio` is a 360x240 NumPy array.

(ii) It places the origin (lower left corner) of the grid (-9m, -6m) relative to the `map` coordinate frame. This puts the `map` frame in the center of the grid (center of the map).

(iii) You will need to convert (x,y) coordinates (in the `map` frame) into grid (u, v) elements.

(iv) The code is built around two functions:

- `sendMap()`. This publishes the map (as is currently known), thus sending it to RVIZ to be visualized. You will need to edit a bit of this.
- `laserCB()`. This is called every time a new laser scan is available and needs to update the map (specifically the log odds ratio). You will need to write most of this.

(v) To see the details of the `LaserScan` message being received, the `OccupancyGrid` message being published, or the `Transform` it uses for relate the laser location to the map, see

```
https://docs.ros.org/en/api/sensor_msgs/html/msg/LaserScan.html
https://docs.ros.org/en/api/nav_msgs/html/msg/OccupancyGrid.html
https://docs.ros.org/en/api/geometry_msgs/html/msg/TransformStamped.html
```

(vi) As always, scan for "FIXME" for things that need editing.

As a first step fix the `sendMap()` function to create probabilities from the log odds ratio. This should be sufficient to run the code. The log odds ratio is initialized to zero, which should lead to a probability of 50% and hence a medium gray map in RVIZ.

Try initializing the map (the log odds ratio) to +1 (more occupied) or -1 (more free). This should appear as more black and more white in RVIZ! Restore the initialization to 0 when done testing.

Please submit the 2 or 3 lines of edited code.

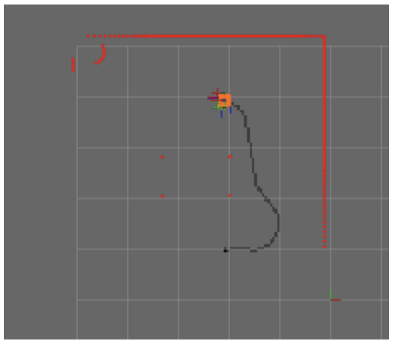
Problem 4 (Tracing the Robot) - 14 points:

Before we build the map, let's make sure we know how to transform (x,y) coordinates into (u,v) grid elements. So instead of creating the map, simply mark where the robot has been and create a trace.

You will notice the `laserCB()` function queries ROS's TF library to determine where the laser scanner (and hence the robot) currently is. Take those (x,y) coordinates and set the equivalent log odds ratio element to +1.

Practically, this will leave a trace as shown to the right.

For THIS, and ALL SUBSEQUENT problems, please submit the code section you edited along with a screen shot after a successful run, for example with the robot having moved to a new room.



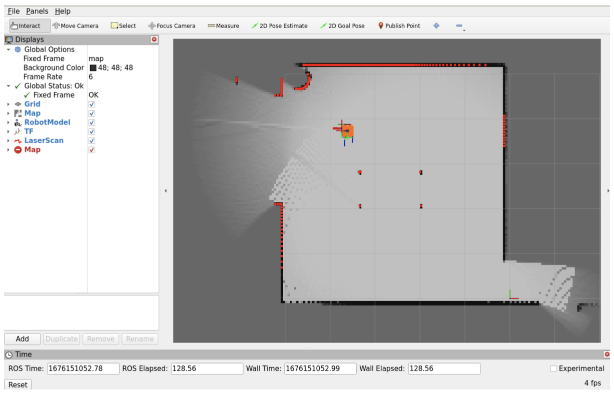
Problem 5 (Marking the Laser Contact Points) - 17 points:

For one more preliminary step, continue to edit `laserCB()`, this time to mark where the laser detects an object. That is, for each ray, if the reported range r shows a valid contact ($r_{\text{min}} < r < r_{\text{max}}$), set the equivalent logs odds ratio element to +1.

Now, as you drive, you should see where the walls are (and remain after the laser scans have moved on). This does not yet clear free space or filter the readings, but should confirm that the coordinate calculations are correct.

Problem 6 (Actual Mapping) - 25 points:

Which brings us to the main problem: creating a map. Edit `laserCB()` to update the log odds ratio as the algorithm properly dictates: increasing the value if contact is detected, reducing the value if free space is detected. You should see the map develop, similar to



```
Global Options
  Fixed Frame          map
  Background Color     48; 48; 48
  Frame Rate           6
Global Status: Ok
  Fixed Frame          OK

ROS Time: 1676151052.78   ROS Elapsed: 128.56   Wall Time: 1676151052.99   Wall Elapsed: 128.56
4 fps
```

If you are unsure how to determine the grid element that lie along a ray, feel free to use Bresenham's algorithm encoded in the given utility function:

```
for (u,v) in self.bresenham((xs,ys), (xe,ye)):
```

where `(xs,ys)` and `(xe,ye)` are the start/end points in grid coordinates and the algorithm returns the list of `(u,v)` pixels, excluding the end point.

Please also answer:

(a) What value should the log odds ratio have to indicate an 90% probability of a grid element being occupied?

(b) Approximately how many times will a wall be hit by a ray before the robot reaches the wall, if driving at a nominal forward speed? The scanner runs at 5Hz.

What is the number if we can only process half the scans and drive at double (high) speed?

(c) Therefore, how much will you increase the log odds ratio (l_{occupied}), every time a ray sees a contact in a grid element?

(d) Equivalently, how much will you decrease the log odds ratio (l_{free}), every time a ray passes through a grid element? Keep in mind that rays are much more tightly spaced near the robot and hence the grid elements nears the robot are seen by many more rays.

Indeed these numbers indicate how much filtering you are doing and how easy it is to change from occupied to free and vice versa. To test, change l_{occupied} or l_{free} by a factor 3.

Problem 7 (Testing with Noisy Sensors) - 8 points:

Please update the launch file to spawn the turtlebot with the noisy lidar. Re-run the above (fully functional) mapping code.

(a) What effect does this have on the map?

(b) Change the two log odds ratio numbers, in particular their ratio $l_{\text{occupied}}/l_{\text{free}}$. In particular, increase and decrease l_{occupied} by a factor 3. What effect does this have on the map?

Keep in mind, if you are tired of driving yourself, you can replace Gazebo and the driving (`teleop`) with a pre-recorded simulation run. In the launch file, do not launch Gazebo, or spawn a turtlebot. Instead start the playback, for example using the `leftside_noisylaser` recording.

Problem 8 (Testing with Bad Localization) - 8 points:

As a second test, please return to a clean lidar - update the launch file to spawn the clean turtlebot or use a clean pre-recorded run. But also update the launch file to use noisy localization.

You are now starting to see the SLAM challenges, if the map is unknown and the localization is poor or unknown.

(a) What happens after a run, in particular after the robot moves to a new room and then returns to the old room?

(b) Besides fixing the localization, can you think of a way/adjustment to the algorithm to more easily allow the system to recover/overcome/forget/undo the now inconsistent/incorrect map data?

Please do not implement anything (unless you are excited to do so). I would simply like you to make a reasonable suggestion.

Problem 9 (OPTIONAL Try Building Map of Unknown Space) - 0 points:

If you still have the energy (entirely optional), try changing the world in the launch file to see what it's like to build a map of a space you haven't seen before.

Problem 10 (Time Spent) - 4 points:

Approximately how much time did you spend on this set? Any particular problems with ROS/Gazebo?