Autonomous Robotic Systems Engineering (AURO)

Week 7 practical - Mapping, localisation, and navigation

Example code

As usual, the example code can be found here:
https://github.com/alanmillard/auro-practicals/tree/main

Specifically, there is a new ROS package for Week 7:
https://github.com/alanmillard/auro-practicals/tree/main/week_7

Downloading the code

If you have previously cloned the GitHub repository into a workspace, you should be able to run "`git pull`" from the `src` directory to pull the changes.

If you are starting from scratch, create a workspace and download the packages as follows:

```
mkdir -p ~/auro_ws/src
cd ~/auro_ws/src
git clone https://github.com/alanmillard/auro-practicals.git .

git clone https://github.com/DLu/tf_transformations.git
pip3 install transforms3d
```

Building the new packages

To build the packages, run these commands:

```
cd ~/auro_ws
colcon build --symlink-install && source install/local_setup.bash
```

Remember that you will either need to run the following command every time you start a new terminal, unless you add it to your `~/.bashrc` file:

```
source ~/auro_ws/install/local_setup.bash
```

GAZEBO_MODEL_PATH

IMPORTANT: Before you can run the example code, you will need to set an environment variable to tell Gazebo where to find the TurtleBot3 models. You will need to run this command every time you start a new terminal, or you can add it to the end of your `~/.bashrc` file.

```
export
GAZEBO_MODEL_PATH=$GAZEBO_MODEL_PATH:/opt/ros/humble/york/install/turtlebot3_gazeb
o/share/turtlebot3_gazebo/models/
```

Simultaneous Localisation and Mapping (SLAM) with Cartographer

These instructions are based on examples from the TurtleBot3 manual: https://emanual.robotis.com/docs/en/platform/turtlebot3/slam_simulation/

Creating a map with SLAM

First, launch the example world:

```
ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
```

Then run the following launch file, which runs RViz and Cartographer SLAM:

```
ros2 launch turtlebot3_cartographer cartographer.launch.py use_sim_time:=True
```

Now teleoperate the robot and drive it around to create a map (occupancy grid) based on the LiDAR data + odometry data:

```
ros2 run turtlebot3_teleop teleop_keyboard
```

Saving the map

Once you have mapped the entire arena, save your map by running the following command:

```
ros2 run nav2_map_server map_saver_cli -f ~/map
```

This will create `map.pgm` and `map.yaml` in your home directory. The PGM file is the occupancy grid itself, and the YAML file contains metadata.

Now press Ctrl+c in the terminals that you launched the ROS nodes from, to close everything down.

Localisation with Nav2

These instructions are based on examples from the TurtleBot3 manual:
https://emanual.robotis.com/docs/en/platform/turtlebot3/nav_simulation/

Initialising the navigation stack

Now that you have created a map of the world, you can use the ROS 2 navigation stack (Nav2) to localise the robot within that map.

First, launch the example world:

```
ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
```

Then run the following launch file, which runs RViz and Nav2:

```
ros2 launch turtlebot3_navigation2 navigation2.launch.py use_sim_time:=True
map:=$HOME/map.yaml
```

This will initially show the black and white occupancy grid map that you created earlier using Cartographer SLAM.

To initialise Nav2, select "2D Pose Estimate" from the toolbar at the top of the RViz window.

Click and hold somewhere in the map - clicking sets the position, then holding the button and moving the mouse lets you set the orientation (green arrow). Once you are happy with the orientation, let go of the mouse button to set the pose estimate.

Look at where the robot is in the Gazebo simulation, and set a pose estimate in RViz that matches its true pose.

If you set a pose estimate that does not align with the robot's true position, RViz will update to show that the Nav2 costmaps do not align with the map created previously with SLAM.

Improving the pose estimate

Nav2 uses Adaptive Monte Carlo Localisation (AMCL) to localise the robot using a particle filter.

After initialising Nav2, the "Amcl Particle Swarm" should be visible in RViz - this looks like lots of small green arrows around the robot. These will be dispersed until the robot moves, as it does not yet have a reliable estimate of its position within the map.

Teleoperate the robot and observe the effect on the particle swarm:

```
ros2 run turtlebot3_teleop teleop_keyboard
```

The spread of the particle swarm should narrow as the robot drives around, and Nav2 gets a better estimate of the robot's true position within the map.

RViz visualisations

Experiment with the RViz visualisation settings for "Amcl Particle Swarm", "Global Planner", and "Controller", to see how these various parts of Nav2 update as you teleoperate the robot.

The costmaps are based on the map (occupancy grid) you previously created using SLAM, as well as the real-time LiDAR data.

Any obstacles are "inflated" in the costmaps, to increase the "cost" of a robot planning a path through these areas. This allows Nav2 to plan paths such that the robot will not drive too close to obstacles.

See the Nav2 documentation for details on the concepts:
https://navigation.ros.org/concepts/index.html

Note that Nav2 calls the "global planner" a "planner, and the "local planner" a "controller".

Navigation with Nav2

These instructions are based on examples from the TurtleBot3 manual:
https://emanual.robotis.com/docs/en/platform/turtlebot3/nav_simulation/

Setting navigation goals

Once Nav2 has a good estimate of the robot's position, you can set navigation goals via RViz. This will cause Nav2 to plan a path through the world, which the robot can autonomously follow.

Make sure that you have killed the teleop node before starting, otherwise it will fight Nav2 for control of the `/cmd_vel` topic.

In RViz, select "Nav2 Goal" from the toolbar. Like the "2D Pose Estimate", click and hold somewhere to set a position and orientation. When you release the mouse button, the robot should autonomously plan a path and drive to the navigation goal.

The "Navigation 2" panel in RViz gives you some real-time feedback on what the navigation stack is currently doing. Nav2 will also print out further information in the terminal that you launched it from.

Depending on the pose you set for the navigation goal, Nav2 may attempt to "recover" if the robot gets stuck, or even abort the plan entirely.

You can set a new "Nav2 Goal" while the robot is currently executing a plan, and it will cancel the current plan, then make a new one.

You can also explicitly cancel the current plan by pressing "Cancel" in the "Navigation 2" panel.

The Nav2 documentation explains how the robot's behaviour is programmed as a behaviour tree, and how it attempts replanning/recovery:
https://navigation.ros.org/behavior_trees/overview/detailed_behavior_tree_walkthrough.html

Waypoint navigation

To set multiple waypoints for the robot to follow, click the "Waypoint / Nav Through Poses Mode" button in the "Navigation 2" panel.

Use "Nav2 Goal" to set multiple waypoints for the robot to follow, then click "Start Nav Through Poses". The robot should autonomously plan a path that drives through these navigation goals in the sequence that you created them.

Clicking "Start Waypoint Following" will return you to the previous mode, in which the robot will just navigate to a single navigation goal.

Relocalisation

Nav2 is able to relocalise the robot if it loses track of where it is in the world.

You can force this by pressing "Pause" in the "Navigation 2" panel, then teleoperate the robot somewhere else in the world. You will need to watch its position in Gazebo, as RViz will not update while Nav2 is paused.

Once you have driven the robot to a different location, kill the teleop node, then, click "Resume" in RViz. The costmaps will not be misaligned with the map of the world, and the particle swarm should be dispersed.

Teleoperate the robot again for a while, until the AMCL algorithm successfully relocalises the robot within the map. You may need to drive to the top/bottom of the map, where there are more asymmetrical features in the world for the robot to localise against.

If you set a new "Nav2 Goal" before teleoperating again, the robot may not be able to relocalise. The "Navigation 2" panel in RViz and the log messages in the terminal that you launched Nav2 from will tell you what it is currently trying to do.

If relocalisation fails, either provide a new "2D Pose Estimate" based on the robot's true pose in Gazebo.

If the state of Nav2 is irrecoverable, try starting again with "Reset/Startup" in the "Navigation 2" panel, or relaunch Nav2 from scratch in the terminal.

Asymmetrical world

The 3x3 grid of identical cylinders in the example world isn't ideal for localisation, as they will look similar from many angles based on the LiDAR data. This makes it difficult for AMCL to estimate the robot's true position in the world, as there are many possible locations in the map that the same LiDAR data could have been perceived from.

This week's practical code provides a copy of this world with different obstacles, which makes the world more asymmetrical. This should help Nav2 to localise more easily using AMCL.

Repeat the steps above on this asymmetrical world, and see if you notice any difference in localisation performance.

SLAM

Therese are the equivalent commands for creating the SLAM map:

```
ros2 launch week_7 turtlebot3_world_auro.launch.py

ros2 launch turtlebot3_cartographer cartographer.launch.py use_sim_time:=True

ros2 run turtlebot3_teleop teleop_keyboard
```

Save the map to a different location this time:

```
ros2 run nav2_map_server map_saver_cli -f ~/auro_map
```

Nav2

You need to run slightly different commands to launch Nav2 with this custom world:

```
ros2 launch week_7 turtlebot3_world_auro.launch.py

ros2 launch turtlebot3_navigation2 navigation2.launch.py use_sim_time:=True
map:=$HOME/auro_map.yaml
world:=$HOME/auro_ws/src/week_7/worlds/turtlebot3_world_auro.world
```

Note that you cannot specify relative paths to your home directory with `~/`, as Nav2 requires the full absolute path. The environment variable `$HOME` will automatically expand to your home directory path, so is equivalent to `~/`