Autonomous Robotic Systems Engineering (AURO)

Week 2 practical - Simulation and visualisation

Gazebo simulation

The official TurtleBot3 documentation has instructions on how to launch Gazebo with different world files, as well as different robot models:
https://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/

IMPORTANT: Make sure you select the “Humble” tab at the top of the page. The default is “Kinetic”, which is for ROS 1.

Testing different worlds

You can launch the empty world as follows:

```
ros2 launch turtlebot3_gazebo empty_world.launch.py
```

This is a completely empty world, so isn’t useful for much other than simple testing/debugging of the robot. The launch file itself is a useful template to use as a starting point for custom launch files, though:
https://github.com/ROBOTIS-GIT/turtlebot3_simulations/blob/humble-devel/turtlebot3_gazebo/launch/empty_world.launch.py

Familiarise yourself with the mouse controls for the camera in Gazebo, and try moving/zooming the camera to get a better look at the robot:
https://classic.gazebosim.org/hotkeys

To launch “TurtleBot3 World” or “TurtleBot3 House”, run one of the following commands:

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

```
ros2 launch turtlebot3_gazebo turtlebot3_house.launch.py
```

https://github.com/ROBOTIS-GIT/turtlebot3_simulations/blob/humble-devel/turtlebot3_gazebo/launch/turtlebot3_world.launch.py

https://github.com/ROBOTIS-GIT/turtlebot3_simulations/blob/humble-devel/turtlebot3_gazebo/launch/turtlebot3_house.launch.py

These launch files will automatically spawn a TurtleBot3 in the world, which you can then teleoperate by running the following command in a separate terminal window:

```
ros2 run turtlebot3_teleop teleop_keyboard
```

Try driving the robot around, and observe how the simulated sensors respond to the environment (camera and LiDAR).

Testing different TurtleBot3 models

The ROS environment is set up to default to use the TurtleBot3 Waffle Pi. The launch files above automatically pick up on the value of the environment variable `TURTLEBOT3_MODEL`. To check what this is currently set to, you can run:

```
echo $TURTLEBOT3_MODEL
```

By default, this will output `waffle_pi`, but you can also set it to `burger` or `waffle` with one of the following commands:

```
export TURTLEBOT3_MODEL=burger
```

```
export TURTLEBOT3_MODEL=waffle
```

```
export TURTLEBOT3_MODEL=waffle_pi
```

However, if you launch a new terminal this environment variable will default back to `TURTLEBOT3_MODEL=waffle_pi`, unless you add an `export` command to your `~/.bashrc`.

Try running the `ros2` commands that you learned last week to inspect the nodes, topics, services, and actions that have been created by Gazebo and the TurtleBot3: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools.html

Try running `rqt_graph` in a separate terminal, and inspect the relationship between the nodes and topics: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html#rqt-graph

Are there any differences in the nodes/topics/services/actions for the 3 different TurtleBot3 models?

Collision avoidance

There is a built-in collision avoidance node that you can run after launching Gazebo:

```
ros2 run turtlebot3_gazebo turtlebot3_drive
```

Make sure you are not also running the `teleop_keyboard`, otherwise you will have two ROS nodes simultaneously publishing to the `/cmd_vel` topic!

The source code for this node is written in C++:

https://github.com/ROBOTIS-GIT/turtlebot3_simulations/blob/humble-devel/turtlebot3_gazebo/src/turtlebot3_drive.cpp

https://github.com/ROBOTIS-GIT/turtlebot3_simulations/blob/humble-devel/turtlebot3_gazebo/include/turtlebot3_gazebo/turtlebot3_drive.hpp

If you are feeling brave, you could try porting this to Python (or write a simpler equivalent Python node).

RViz

Much like `rqt`, you can run the command `rviz2` without any parameters. This will launch RViz in an unconfigured state, but you can set it up manually if you wish.

TurtleBot3 configuration

The following command will launch RViz and automatically configure it for use with the TurtleBot3 robot:

```
ros2 launch turtlebot3_bringup rviz2.launch.py
```

The source code of this launch file can be found here: https://github.com/ROBOTIS-GIT/turtlebot3/blob/humble-devel/turtlebot3_bringup/launch/rviz2.launch.py

It specifies this configuration file for RViz: https://github.com/ROBOTIS-GIT/turtlebot3/blob/humble-devel/turtlebot3_description/rviz/model.rviz

LiDAR visualisation

Move/resize the Gazebo and RViz windows so that they can see both of them at the same time. Then use a terminal to teleoperate the robot with the keyboard.

Observe how the LiDAR sensor readings update with respect to obstacles in the simulated environment. Try hiding the Gazebo window, and see if you can teleoperate the robot when only looking at the RViz visualisation of the laser scan data. This is all the robot can see - hopefully it's enough to avoid crashing into obstacles!

Camera visualisation

Make sure that your environment variables are set up to simulate the TurtleBot3 Waffle Pi, then launch Gazebo, and RViz with the launch file above.

You can then add a camera visualisation in RViz by clicking "Add > By topic" > `/camera/image_raw`". The camera visualisation should then appear in the bottom-left corner. You may wish to change the Visibility options, to turn off some of the debugging visualisation.

Manual configuration

Try launching `rviz2` from the command line (without the launch file), and see if you can reproduce the TurtleBot3 configuration manually. The contents of model.rviz (linked above) may give you some hints.

You can add visualisations by clicking “Add” in the bottom-left corner, then “By display type” or “By topic”. Depending on the visualisation you select, you may need to configure parameters that appear in the sidebar, such as the ROS topic to subscribe to. Some fields may look blank, but if you click on them, there is usually a drop-down menu of options that have been detected automatically.

Troubleshooting

ROS/Gazebo nodes sometimes fail to launch properly - if you encounter a problem, it may not be an issue with any of the commands you have run, or the code you have written.

If something doesn't work as expected, try pressing Ctrl+C (sometimes multiple times) in each of your terminals that are running ROS nodes, and relaunch/rerun them.

This may not always work, as ROS/Gazebo processes sometimes fail to exit cleanly, and remain hanging. Closing the terminals that you launched them from may help, but if not you may need to kill them manually (see below).

ROS 2 daemon

The `ros2 daemon` starts in the background automatically whenever you run a `ros2` command. You can check whether it is running as follows:

```
ros2 daemon status
```

If you are experiencing issues, restarting the ROS 2 daemon may help (it may be necessary to close the terminals belonging to hanging processes first):

```
ros2 daemon stop
ros2 daemon start
```

ROS 2 middleware (RMW)

The ROS environment provided for you on the lab PCs is set up to use Cyclone DDS, however the default for ROS is Fast DDS (formerly Fast RTPS):

https://docs.ros.org/en/humble/Installation/DDS-Implementations.html

https://docs.ros.org/en/humble/How-To-Guides/Working-with-multiple-RMW-implementations.html

If you run a ROS command outside of the ROS environment (e.g. in a terminal launched without the "ROS2 Xfce Terminal" shortcut, or if you haven't customised your `~/.bashrc`), this will launch a `ros2 daemon` with the default RMW.

It is possible to get into a situation where you have two different `ros2 daemon` processes running simultaneously, configured with different RMW implementations. ROS nodes associated with one `ros2 daemon` will then not be able to talk to those associated with the other one.

You can use the following command to check for running ROS processes:

```
ps -ef | grep -i ros
```

You should see an output that contains this:

```
/usr/bin/python3 -c from ros2cli.daemon.daemonize import main; main() --name
ros2-daemon --ros-domain-id 0 --rmw-implementation rmw_cyclonedds_cpp
```

If you have run ROS outside of the pre-configured environment, you will see an output more like this:

```
/usr/bin/python3 -c from ros2cli.daemon.daemonize import main; main() --name
ros2-daemon --ros-domain-id 0 --rmw-implementation rmw_fastrtps_cpp
```

This indicates that the default DDS is still being used. In this case, run the commands above to stop the `ros2 daemon` and kill any rogue ROS processes.

Killing rogue Gazebo processes

If Gazebo is misbehaving, you can run the following command to kill all of the associated processes before launching it again:

```
killall -9 gazebo & killall -9 gzserver & killall -9 gzclient
```

Example solutions

Python port of `turtlebot3_drive` (C++)

https://github.com/alanmillard/auro-practicals/tree/main/week_2

If you haven't already got a ROS workspace, you can create one as follows:

```
mkdir ~/auro_ws/src
```

Then clone the GitHub repository into the src directory:

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

IMPORTANT: You will also need to run the following commands to install a library for converting Quaternions to Euler angles:

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

Then build the code **from the workspace root directory**:

```
cd ~/auro_ws
colcon build --symlink-install
```

Using the `--symlink-install` flag causes ROS to pick up on changes to your Python source code, without having to run a separate `colcon build` command every time. However, you will need to rerun `colcon build` every time you change files related to the package configuration.

In 3 separate terminals, run the following commands.

Terminal 1:

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

Terminal 2:

```
source ~/auro_ws/install/local_setup.bash
ros2 run week_2 turtlebot3_drive_python
```

Terminal 3:

```
ros2 launch turtlebot3_bringup rviz2.launch.py
```

RViz manual setup

Instead of running "`ros2 launch turtlebot3_bringup rviz2.launch.py`", you can launch RViz directly by running:

```
ros2 run rviz2 rviz2
```

Or the shortcut:

```
rviz2
```

This opens RViz in an unconfigured state - if you want to manually set it up like the above launch file, try following these steps:

~~**Global Options > Fixed Frame > base_link**~~
This will make the visualisation relative to the robot, so the laser scan data will rotate around the robot.

Global Options > Fixed Frame > odom
This is more intuitive for teleoperation - the robot will move around relative to the world.

Add > By display type > RobotModel
RobotModel > Description Topic > /robot_description

Add > By display type > TF
TF > Show Names > Check this box to see the link names

Add > By topic > /camera/image_raw
Camera > Visibility > Uncheck this box to hide RViz visualisations, or expand the menu and select the visibility of individual features

Add > By topic > /scan/LaserScan
LaserScan > Style > Points (easier to see than Flat Squares, with default settings)

Add > By topic > /odom/Odometry
Odometry > Covariance > Uncheck this box to remove the big yellow circle - see here: https://answers.ros.org/question/282512/rviz-odometry-is-all-yellow/

Odometry > Position/Angle Tolerance > Increase to reduce frequency of arrows drawn

Views > Orbit

Yaw = 3.14 (In radians: pi = 180 degrees)
Pitch = 1.57 (In radians: pi/2 = 90 degrees)
Distance = 8 (zooms in a bit)