Autonomous Robotic Systems Engineering (AURO)

Week 5 practical - Perception and multi-robot systems

Example code

The example code for the Week 4 and Week 5 practicals can be found here:
https://github.com/alanmillard/auro-practicals/tree/main

Specifically, there are two new ROS packages, and two new custom message types:
https://github.com/alanmillard/auro-practicals/tree/main/week_4
https://github.com/alanmillard/auro-practicals/tree/main/week_5
https://github.com/alanmillard/auro-practicals/tree/main/auro_interfaces

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/
```

Random walk with collision avoidance

You can run the example robot controller as follows:

```
ros2 launch week_4 week_4_launch.py

ros2 run week_4 robot_controller
```

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
```

RViz text markers

The `robot_controller` node publishes its current state and pose to a custom node called `rviz_text_marker`, which in turn, publishes a RViz Marker message that can be visualised: http://wiki.ros.org/rviz/DisplayTypes/Marker

You can run the `rviz_text_marker` node and launch RViz as follows:

```
ros2 run week_3 rviz_text_marker

ros2 launch turtlebot3_bringup rviz2.launch.py
```

You can add a visualisation for this as follows:

Add > By topic > `/marker_output`

This marker visualisation will be easier to see if you disable the visualisations for TF and Odometry.

Image processing

Collecting items via teleoperation

You can launch the example world and spawn collectable items as follows:

```
ros2 launch week_4 week_4_launch.py

ros2 run week_5 item_spawner
```

Try using the teleop node to drive the robot around and collect the items:

```
ros2 run turtlebot3_teleop teleop_keyboard
```

Detecting items

The `item_detector` node processes the `/camera/image_raw` topic using OpenCV, and publishes detected items to the `/items` topic:

```
ros2 launch week_4 week_4_launch.py

ros2 run week_5 item_spawner

ros2 run week_5 item_detector
```

Try running the teleop node again, and observe the output of the image processing as you drive the robot around.

Autonomous item collection

Run the following commands while the nodes above are running to see the output of the `item_detector` node:

```
ros2 topic info /items

ros2 topic echo /items
```

Edit the `robot_controller` node from `week_4` to subscribe to `ItemList` messages, and modify the finite state machine to make the robot drive towards an item when it detects one.

Multi-robot simulations

You can launch a multi-robot simulation using launch files from the `nav2_bringup` package as follows:

```
ros2 launch nav2_bringup cloned_multi_tb3_simulation_launch.py robots:="robot1={x: -2.0, y: -0.5, yaw: 1.5707}; robot2={x: 2.0, y: -0.5, yaw: 1.5707}"
```

Run the following command in a separate terminal:

```
ros2 topic list
```

You should see that the topics for each robot are separate by namespace, for example:

```
/robot1/scan
/robot2/scan
```

Remapping topics

To reuse existing nodes with namespaces, you will need to remap topics: https://docs.ros.org/en/humble/How-To-Guides/Node-arguments.html

For example:

```
ros2 run turtlebot3_gazebo turtlebot3_drive --ros-args --remap __ns:=/robot1
```

```
ros2 run turtlebot3_teleop teleop_keyboard --ros-args --remap __ns:=/robot2
```

Read through the corresponding launch file from the nav2_bringup package and make sure you understand how it launches the nodes with different namespaces for each robot: https://github.com/ros-planning/navigation2/blob/humble/nav2_bringup/launch/cloned_multi_tb3_simulation_launch.py

Multi-robot item collection

Port some of the code from this launch file into the launch file from the `week_4` package, to make multiple robots spawn in the `week_4` example world.

Once you have done this, try launching the `robot_controller` node from the `week_4` package, but with the namespace remapped such that it works with `/robot1`.

You can then launch another instance remapped to the namespace for `/robot2`, such that two robots attempt to collect items simultaneously.