Autonomous Robotic Systems Engineering (AURO)

Week 3 practical - Control architectures

Finite State Machines

Here are some instructions on how to get started from scratch, rather than cloning an existing GitHub repository:

Creating a workspace, package, and node

These commands will create workspace called `week3_ws`, a package called `week_3`, and a node called `turtlebot3_fsm`:

```
mkdir -p ~/week3_ws/src
cd ~/week3_ws/src
ros2 pkg create --build-type ament_python --node-name turtlebot3_fsm week_3
```

The resulting directory tree should look like this:

```
am567@auro-pc:~$ tree ~/week3_ws
/home/am567/week3_ws
└── src
    └── week_3
        ├── package.xml
        ├── resource
        │   └── week_3
        ├── setup.cfg
        ├── setup.py
        ├── test
        │   ├── test_copyright.py
        │   ├── test_flake8.py
        │   └── test_pep257.py
        └── week_3
            ├── __init__.py
            └── turtlebot3_fsm.py

5 directories, 9 files
```

You can open these files in VS Code by running the following command:

```
code ~/week3_ws/src
```

Open the Python file `~/week3_ws/src/week_3/week_3/turtlebot3_fsm.py` to edit the source code of your newly created node.

I've written a simple ROS node that implements an example finite state machine, which you can use to get started. Copy this example code into the file:
https://github.com/alanmillard/auro-practicals/blob/main/turtlebot3_fsm.py

This node will make the robot alternate between two states - driving forward, and turning - based on a simple counter. It should roughly drive in a square path, based on the counter thresholds (imprecise).

Building

```
cd ~/week3_ws
colcon build --symlink-install
```

Running

```
source ~/week3_ws/install/local_setup.bash
ros2 run week_3 turtlebot3_fsm
```

You will need to run "`source ~/week3_ws/install/local_setup.bash`" whenever you start a new terminal, otherwise your ROS environment won't know about your custom package/node.

If you are comfortable with editing your `~/.bashrc` file, then you can add the command "`source ~/week3_ws/install/local_setup.bash`" to the end of it, so that it runs automatically every time you launch a new terminal.

Terminal 1:

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

Terminal 2:

```
source ~/week3_ws/install/local_setup.bash
ros2 run week_3 turtlebot3_fsm
```

Using Ctrl+c to kill your node will unfortunately not stop the robot from moving, as the wheel speeds will remain set according to the last Twist message received on the `/cmd_vel` topic. If you want to stop it manually, you can run the teleop node:

```
ros2 run turtlebot3_teleop teleop_keyboard
```

However, make sure that you kill the teleop node before running your FSM node again, otherwise they will fight over the `/cmd_vel` topic.

If the robot crashes and becomes irrecoverable, you don't need to close and reopen Gazebo - instead you can run `rqt` and call the `/reset_simulation` service.

Extending the example

Editing the state machine to change the robot's behaviour. There are various things you could try to make the robot do:

- Drive forward by a random amount
- Turn by a random amount
- Randomly choose the direction that it turns
- Turn for an amount of "wall-clock" time, rather than a counter
- Turn by a specific angle
- Add separate states for turning left/right
- Detect and avoid obstacles

You will need to subscribe to the `/scan` and/or `/odom` topics to implement some of these.

It may help to take a look at the Python port of the C++ collision avoidance node from last week's practical:
https://github.com/alanmillard/auro-practicals/blob/main/week_2/week_2/turtlebot3_drive_python.py

ROBOTIS also provide an example Python node for "obstacle detection" - see here for a description (select the Dashing tab):
https://emanual.robotis.com/docs/en/platform/turtlebot3/basic_examples/#obstacle-detection

https://github.com/ROBOTIS-GIT/turtlebot3/tree/humble-devel/turtlebot3_example/turtlebot3_example/turtlebot3_obstacle_detection

Launch files

Instead of running Gazebo/RViz and your FSM node in separate terminals, you can create a launch file that does this automatically with one command.

Follow this tutorial from the official ROS 2 documentation to get used to simple launch files with Turtlesim:
https://docs.ros.org/en/humble/Tutorials/Intermediate/Launch/Creating-Launch-Files.html

Once you're comfortable with the basic concepts, take a look at the TurtleBot3 launch files for the empty and turtle worlds:

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

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

Try creating a custom launch file that is mostly the same, but spawns the TurtleBot3 in a different location.

You can then try editing the launch file to run your custom FSM node at the same time as Gazebo.

If you want to launch RViz automatically as well, take a look at this launch file for an example:

https://github.com/ROBOTIS-GIT/turtlebot3/blob/humble-devel/turtlebot3_cartographer/launch/cartographer.launch.py