AA 274A: Principles of Robot Autonomy I
Section 8: Assembling an Autonomy Stack
Case Study: Multiplexing

1 Multiplexing

Multiplexing is the idea of easily switching between different inputs in a software stack. In robotics, this is extremely useful to quickly switch between different sensors if one is faulty[1] or becomes imprecise in challenging environmental conditions (e.g., a drone could stop using its camera if it enters a dark room, and rely on its inertial measurement unit (IMU) instead[2]). Multiplexers are also widely used to switch between different controllers for different terrain[3] or robot operating modes (for instance, a drone may switch to an emergency controller if one motor is faulty)[4].

In this section, you will write a multiplexer to quickly switches between different controllers. **There are multiple approaches to do this, so this section is open-ended**. The goal of today's section is to use multiplexing to put together everything you have learned in previous sections.

We propose two different approaches:

- Listen to a topic where information about the desired controller (e.g., its name) is published.
- Update a ROS parameter in real time.

To start, launch the default robot stack as we've done in previous sections:

```
roslaunch asl_turtlebot project.launch
```

Problem 1: To which topic are the robot's control input sent to? **Hint**: use the commands `rostopic list`, `rosnode list`, `rosnode info`, and `rqt_graph`. What node is currently publishing these control inputs? In your writeup, include the commands that you run to check this, and their output.

Problem 2: What command would you use to control the robot using your keyboard? **Hint:** Take a look at Section 3 and file `/catkin_ws/src/turtlebot3/turtlebot3_teleop/nodes/turtlebot3_teleop_key`, We do not suggest the one in `/asl_turtlebot/scripts` which is extremely hard to use.

A complex control stack may fail when facing new unexpected situations. Thus, it is often a good idea to be able to quickly switch to a backup controller, which is also useful for debugging. In this section, the backup controller will be teleoperated, such that you can control the robot from your keyboard.

Problem 3: Propose a method to efficiently switch from the controller used in the previous section (i.e. with the navigator) to the teleoperated controller. Different methods are possible. **Hint**: One method is to write a node that subscribes to both control topics (one from `navigator.py`, and one from the teleop controller) and publishes to the topic with the control inputs that are sent to the robot.

[1] See for example `https://arxiv.org/pdf/2008.09679.pdf` and `https://player.vimeo.com/video/458021911?autoplay=1`.
[2] See for example `https://arxiv.org/pdf/1909.00079.pdf`.
[3] See for example `https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9172739`.
[4] See for example `https://www.youtube.com/watch?v=P3fM6VwXXFM`.

Problem 4: Implement your idea. For now, you can choose which controller is in charge by hard-coding some parameter. The rest of this section will look at different ways to switch between controllers. Make sure that control commands are only published to the robot from one source.

Problem 5 : Write a launch file that starts both the main software stack (`project.launch`) and your backup keyboard controller.

1.1 Changing your parameter in real-time with a ROS parameter

It may sometimes be more convenient to select the desired controller by changing it using a ROS parameter.

Problem 6: Make the parameter that enables switching between controllers available as a ROS parameter. It should be visible when running the command `rosparam list` and can be set using the `rosparam set` command.

1.2 (Optional, if you have time) Changing your parameter from a GUI

It may be more convenient to change it from a graphical user interface.

Problem 7: Using the content from the previous section, make this parameter reconfigurable in real time using the package `dynamic_reconfigure`. Paste a screenshot of your graphical user interface with the visible parameter into your submission.

2 Final project

Use the remaining time in the section to start planning out your final project. Discuss with your section group (yes, your section group, even if it is not the same as your final project group), possible plans for organizing the high-level behavior of your robot. After each of your final project group members have completed section this week, consolidate your ideas into a plan for your final project. You will hand in this plan in a separate Gradescope assignment from the Section 8 assignment. How you write up this plan is open-ended, however it should be detailed and specific. For example, if you plan on implementing a "delivery" module, your plan could be an FSM detailing how your robot switches between different states in the delivery process. Additionally, briefly describe the extensions you plan on implementing in your write-up.