Homework 1

ME 500 A4 - Prof. Tron

Friday 16th February, 2018

The goal of this homework is to get some practice with writing and running ROS nodes in Python.

General instructions

The assignment is intended to be completed in pairs using *pair programming* (remember, two people working on *one* laptop). See the syllabus for details, and see Blackboard for the group assignments.

Homework report (required). Solve each problem, and then prepare a small PDF report containing comments on what you did for each problem, including whether you tackled the **optional** questions.

Demo video (required). You need to prepare a one to three minutes video with the following elements:

- A foreground framing of yourself (both members of the group) saying your names and the name(s) of the robot(s).
- For each question marked as **video**, a short segment showing the laptop and robot demonstrating that you completed the requested work. If you could not complete a question, you must explain what were the problems that you encountered.
- A final conclusion where you explain what you learned with the activity.

Submitting multiple video segments instead of a single video is allowed but not encouraged (in this case, all the segments must be recorded in the same session, e.g., not on different days or hours apart).

Submission. Compress all code you wrote for the assignment into a single ZIP archive with name `<Last1><First1>-<Last2><First1>-hw<Number>.zip`, where `<First1>`, `<First2>` and `<Last1>` , `<Last2>` are the first and last names of the group, and `<Number>` is the homework number (for instance, `TronRoberto-BeeVang-hw1.zip`). Submit the report, the video and the ZIP archive through Blackboard. Both members of the group need to submit a copy of this material. Please refer to the Syllabus on Blackboard for late homework policies.

Grading. Each question is worth 1 point unless otherwise noted. Questions marked as **optional** are provided just to further your understanding of the subject, and not for credit. If you submit an answer I will correct it, but it will not count toward your grade. See the Syllabus on Blackboard for more detailed grading criteria.

Maximum possible score. The total points available for this assignment are 5.5 (5.0 from questions, plus 0.5 that you can only obtain with beauty points). Points for the video questions are assigned separately on Blackboard.

Hints Some hints are available for some questions, and can be found at the end of the assignment (you are encouraged to try to solve the questions without looking at the hints first). If you use these hints, please state so in your report (your grading will not change based on this information, but it is a useful feedback for me).

Use of external libraries and toolboxes All the problems can be solved using the basic ROS Python facilities. You are **not allowed** to use functions or scripts from external libraries or packages unless explicitly allowed.

Problem 1: Listener-talker node

As a starting point, this question will ask you to make a node that subscribes and publishes on a topic.

Question 1.1. Implement a node that combines the templates provided in the `talker.py` and `listener.py` files.

> File name: `talker_listener.py`
> Subscribes to topics: `chatter` (type `std_msgs/String`)
> Publishes to topics: `chatter` (type `std_msgs/String`)
> Description: Combine the scripts in the files `talker.py` and `listener.py`. The result should be a script for a single node that publishes regularly a message on the topic `chatter` and prints on screen what it hears on the same topic.

You can solve the next question below in lieu of this one. Note: for this question, the node can "work alone", i.e., it will produce a continuous output even if only one instance is launched. However, you should be able to launch multiple instances at the same time; in this case, each instance would "hear" messages from every other instance.

Question 1.2 (optional). Same as the previous question, but instead of using the files `talker.py` and `listener.py`, use the files `talkerClass.py` and `listenerClass.py`. That is, the node from your new script should be wrapped in a Python class.

Question 1.3. Modify your node so that it publishes something more interesting than the current time or a count (e.g., the verses of a poem or a row of the table of elements).

Question 1.4 (optional). Transform the `talker_listener.py` node into a `talker_listener_chatbot.py` node that *1)* publishes on the topic `/chatter` when it is started, *2)* replies every time something is heard on the topic `/chatter`, but *3)* does not reply to its own messages. The result of this question is essentially the same as the chat bot from the mini-hackaton. You should get a "continous" stream of messages on the topic only if you launch two instances of the node.

Question 1.5 (video). Show the results after launching two nodes, both in the nodes' terminal windows, and also the output of the command `rostopic echo /chatter`.

Question 1.6 (optional). Modify your node so that it uses `rospy.get_name()` to get its name assigned by the system, and changes its behavior accordingly (e.g., it publishes counts on different ranges or from a different poem).

Problem 2: Keyboard teleoperation

In this question you will make a generic keyboard teleoperation node that translates key presses into messages that define the commanded linear and angular speeds for the robot. Before starting this question, you should have completed the previous question, and also looked at the scripts `key_terminate.py` and `zigzag_twist.py` provided in the repository. The velocities published by the node `key_op` in this question will be used as a reference to command the motors by the node in the next question.

Question 2.1. Implement a node according to the following specification.

> File name: `key_op.py`
> Subscribes to topics: None
> Publishes to topics: `motor_twist` (type `geometry_msgs/Twist`)
> Description: The node should maintain, internally, two variables: `speed_linear` and `speed_angular`. These variables should be initialized at zero, must always stay in the interval between -1.0 and 1.0, and will be adjusted with increments of ± 0.2. The node should regularly check for key presses (see `key_terminate.py` for an example of how this can be implemented), and then map the following keys to the following actions:
>
> - `'W'`: increment `speed_linear`;
> - `'S'`: decrement `speed_linear`;
> - `'D'`: increment `speed_angular`;
> - `'A'`: decrement `speed_angular`.
>
> At the beginning, the script should print on screen a brief description of the commands above, together with the current values of `speed_linear` and `speed_angular`. Every time either `speed_linear` or `speed_angular` changes, the node should perform the following actions:
>
> *1)* Print on screen the updated values of `speed_linear` and `speed_angular`.
>
> *2)* Prepare a `Twist` message where the `linear.x` field is set to `speed_linear` and the `angular.z` field is set to `speed_angular` (all other fields are left to zero).
>
> *3)* Publish the `Twist` message on `motor_twist`.

The message `Twist` is defined for 3-D velocities (linear and angular); however, since our robot is essentially 2-D, can rotate, but cannot move sideways, we use only two fields out of the six available. Note: it is suggested that you define a constant `SPEED_DELTA=0.2` and use it for the increments; in this way, if later you want to adjust this amount, it will be easily accessible.

Question 2.2 (video). Show the results on the terminal while giving different sequences of commands, and the result of the command `rostopic echo /motor_twist`.

Problem 3: Motor command

In this question you will make a node that translates motor commands into actual motions of the robot. You will have to use the theory from the class to figure out how to translate the commands in motor speeds. You can use the node from the previous question to test it.

Before starting this question, you should look at the script `scripts/zero_motors.py`.

Question 3.1. Implement a node according to the following specification.

> File name: `motor_command.py`
> Subscribes to topics: `motor_twist` (type `geometry_msgs/Twist`)
> Publishes to topics: None.
> Description: As initialization, the node should import the module `me416_utilities` provided in the repository, and then create two objects from the classes `MotorSpeedLeft` and `MotorSpeedRight` in that module (see the file `scripts/zero_motors.py` in the provided repository for an example). Then, every time a new `Twist` message is published on the topic `motor_twist`, the node should compute the speeds for the left and right motors, and then command the motors using the method `.set_speed(speed)` of the two objects you create. Recall that the argument `speed` needs to be a number between -1.0 (full speed backward) and 1.0 (full speed forward).

Note that the constructors of these two classes accept an optional argument `speed_offset` (which is intended to be between 0.0 and 1.0) that acts as a multiplier for the set velocity; for instance, if the set speed for a motor is 1.0, but `speed_offset` for that motor is 0.9, then the effective velocity of that motor will be 0.9 of its max speed.

Question 3.2 (optional). Add constants `SPEED_MULTIPLIER_LINEAR` and `SPEED_MULTIPLIER_ANGULAR` that are used to change the mapping between

Question 3.3. Adjust `speed_offset` for one of the motors so that, on a level non-slippery surface, the robot will actually go straight when commanded to do so.

Question 3.4 (video). Show the robot following various commands at different speeds.

Hint for question 1.4: The easiest way to satisfy the requirement of not replying to the node's own publishing, is to keep an internal variable `last_str` with the last message published. If the message received is equal to `last_str`, then the node should not publish a reply. Additionally, you will need to insert a pause to avoid replying too quickly to a message.

Hint for question 2.1: Since this node needs to keep some internal state, it is the best to wrap your callbacks and main functions in an object.