Homework 1

ME 416 - Prof. Tron

2024-02-25

The goal of this homework is to get some practice with writing and running ROS nodes in Python, and to start laying the groundwork make the robot move. For this assignment, it is acceptable to test all your code on `roslab.bu.edu` alone. Please read the instructions on Blackboard carefully.

Figure 1 gives an overview of the nodes and files involved in this Homework, and how they relate to each other and with the nodes already provided in the `git` repository.

Preparation. Please make sure you have the latest version of the provided code, by running the command `rastic_cl ws update`.

🔎 Under the hood, this command uses `git pull` on each of the subdirectories under `~/rastic_ws/src`, stashing your changes if they conflict with the changes we provided. If you are working on the robot, it will need an Internet connection (i.e., it needs to be connected to the WiFi network).

Problem 1: Listener_accumulator node

Goal This question will ask you to make a node that subscribes to string messages, and then publishes them, in blocks, at a lower rate. The goal is to get more familiar with publishers and subscribers, and to get a better understanding of when messages are published.

Question node 1.1. Implement a node that combines the templates provided in `talker.py` and `repeater.py`. It is suggested that you start by copying one of these files, and then adding/changing the code to obtain the requested functionality.

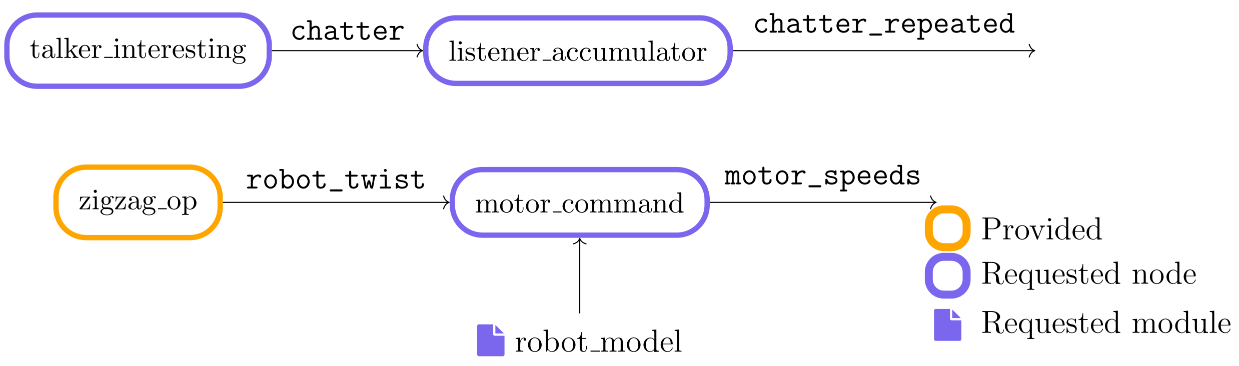

Figure 1: ROS graph of the nodes and files used in this assignment

File name: `listener_accumulator.py`
Subscribes to topics: `chatter` (type `std_msgs/String`)
Publishes to topics: `chatter_repeated` (type `std_msgs/String`) Frequency: 1/3 Hz.
Description: The function of this node is similar to the one of `repeater_class.py`; however, instead of publishing the received messages after each callback, it concatenates them, and publishes them at a (lower) rate. The result should be a node that regularly (at the frequency given above) publishes a message on the topic `chatter_repeated`, with a message having as a content all the messages that it has received on the topic `chatter` between updates. The published messages should also be printed on screen.

As an example, assume that there are messages on the topic `/chatter` appearing with the times and content shown in Table 1a; then, assuming that the node `listener_accumulator.py` publishes at times 3.0 s and 6.0 s seconds, the messages on the topic `/chatter_repeated` should appear as shown in Table 1b. See the hints for this question for additional details.

Table 1: Example illustrating how `listener_accumulator.py` should work.

(a) Topic `/chatter`

Time	Content
0.1 s	`Message 1`
1.3 s	`Message 2`
2.1 s	`Message 3`
3.2 s	`Message 4`
5.0 s	`Message 6`

(b) Topic `/chatter_repeated`

Time	Content
3.0 s	`Message 1 Message 2 Message 3`
6.0 s	`Message 4 Message 6`

Question report 1.1. Using the example in Table 1, and referring to your code, explain the flow of execution (i.e., which parts of `listener_accumulator.py` are executed and when). Add snippets of your code as you deem necessary.

Question node 1.2. Copy the file `talker.py` into the file `talker_interesting.py`. Modify `talker_interesting.py` so that it publishes the contents of a list of strings that you define (e.g., with the verses of a poem, the lines of a song, Pokemon names, or the elements of the periodic table). When the node reaches the end of the list, it should continue from the top. Ensure that the frequency of the publisher is 2 Hz.

Question video 1.1. Launch the nodes `talker_interesting.py`, `listener_accumulator.py`, and the commands `ros2 topic hz /chatter`, `ros2 topic hz /chatter_repeated` on side-by-side terminals (i.e., you should see the four terminals at the same time). Records a few seconds of the outputs (as already mentioned, ideally you should use a screen recorder for this).

Problem 2: Motor command

In ROSBot, you can control the speed of the two motors to change the overall linear and angular speed of the entire robot. See Figure 2 for an illustration of the axes and robot velocities. **Goal** In this question you will make

1) A function that transforms high-level desired robot velocities to actual motor speeds (file `rastic_ws/src/me416_lab/robot_model/robot_model.py`);

2) A node that uses this function to actually move the robot given ROS messages, and that publishes the speeds used (file `motor_command.py`).

3) Test your node with a provided node (`zigzag_op.py`) which repeatedly publishes velocity commands to make the robot turn left and right.

Question **code** **2.1.** Create the function specified below. Note that you should already have the file `robot_model/robot_model.py` from a previous in-class activity; add content to it.

File name: `robot_model/robot_model.py`

. Function name: `twist_to_speeds`

Description: Given the desired linear and angular velocity of the robot, returns normalized speeds for the left and right motors. Speeds needs to be thresholded to be between -1.0 (backward at maximum speed) and 1.0 (forward at maximum speed).

Input arguments

- `speed_linear` (type `float`) Linear speed of the robot (i.e., along the x-axis of the robot).
- `speed_angular` (type `float`) Angular speed of the robot (i.e., around the z axis of the robot).

Returns arguments

- `speed_left` (type `float`) The speed for the left motor.
- `speed_right` (type `float`) The speed for the right motor.

In general, the inputs (`speed_linear` and `speed_angular`) are both in the range [`-1.0`, `1.0`], and could specify a mix of linear and angular velocities (e.g., `speed_linear=0.5`, and `speed_angular=-0.5`); i.e., the desired trajectory of the robot could be an arc, not simply going forward/backward or rotating in place.

Note: For this question, think about how different combinations of motor speeds can affect the overall linear and angular velocity of the robot. However, you do not need to be too formal (i.e., solve the kinematics problem). For this assignment, just write something sensible; you will revise this function and make the model more formal in the next homework after discussing the robot kinematics in class. Whatever logic you choose to implement, the following points should be true (these are tested by the autograder on Gradescope):

1) If `speed_angular` is close to zero, the two motor speeds should be approximately equal.

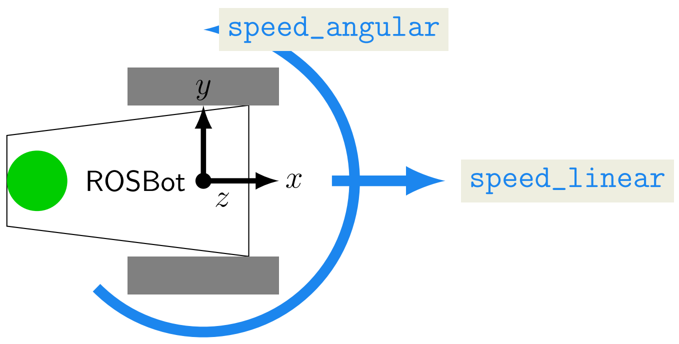

Figure 2: Axes on ROSBot to define linear and angular velocities.

2) If `speed_angular` is positive (respectively, negative), `right` should be greater (respectively, lower) than `left`.

3) If `speed_linear` is positive, the average of `left` and `right` should be positive.

Question **report** **2.1.** Add comments to the function `twist_to_speed`( ) that explain the logic you implemented for Question code 2.1. Include a screenshot or copy and paste the code in your report (the score in this question will be given based on how easy is to understand the logic from the comments).

Question **report** **2.2.** First, visit the page `https://docs.ros2.org/latest/api/geometry_msgs/msg/Twist.html`. Take a screenshot. Then, run the command `ros2 interface show geometry_msgs/msg/Twist`, Take a screenshot.. Finally, explain the meaning of `Vector3`, `Twist`, `float64`, and `geometry_msgs`. In particular, explain how they hierarchically relate to each other and to the concept of message types seen in class.

Question **code** **2.2.** Copy the file `~/rastic_ws/src/me416_lab/me416_lab/twist_example_template.` to the file `twist_example.py`, and modify the stub of the function `twist_fill`( ) as described below.

> File name: `twist_example.py`
>
> Function name: `twist_fill`
>
> Description: Create an object for a message of type `geometry_msgs/Twist`, fill *all* attributes with non-zero, floating point values (you can pick arbitrary values, as long as they are not zero).
>
> Returns arguments
>
> - `msg` (type `geometry_msgs.msg.Twist`) A message with all the attributes filled.

The goal of this question is just to get you familiar with how to create message objects, and how nested message types are handled.

Question **report** **2.3.** Open the file `~/rastic_ws/src/me416_msgs/msg/MotorSpeedsStamped.msg`. Take a screenshot. Run the command `ros2 interface show me416_msgs/msg/MotorSpeedsStamped`.

Take a screenshot. Using what we discussed in class about ROS message types, write in your report an explanation of what the output of the last command means.

Preparation. Look at the script `scripts/zero_motors.py` as an example on how to set the motor speed.

Preparation. The next questions refer to the following node specification, which you will need to write through the next few questions.

> You will have to start this file from scratch, no template is provided.

> File name: `motor_command.py`
> Subscribes to topics: `robot_twist` (type `geometry_msgs/Twist`)
> Publishes to topics: `motor_speeds` (type `me416_lab/MotorSpeedsStamped`)
> Description: See questions below for the expected behavior of the node.

Question **report** **2.4.** As initialization, the node should perform three steps:

1) Import the modules `robot_model` (from Question code 2.1) and `me416_utilities` (provided in the repository).

2) Create two objects `motor_left` and `motor_right` from the classes `MotorSpeedLeft` and `MotorSpeedRight`, respectively, which are provided under the `me416_utilities` module.

3) Initialize the ROS node, and then create a subscriber and a publisher for, respectively, the topics `robot_twist` and `motor_speeds`.

Note that the constructors of `MotorSpeedLeft` and `MotorSpeedRight` 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. In future assignments you might have to change this to make the robot go straight when a zero angular velocity is requested. Include a copy of this part of the code in your report.

Question **report** **2.5.** In the callback for the topic `robot_twist`, the node should transform the content of the `Twist` message into the speeds for the left and right motors using the function `twist_to_speed`( ) from the module `robot_model`, and then set those speeds on the motors using the methods `.set_speed`(`speed`) of the objects `motor_left` and `motor_right`. Include a copy of this part of the code in your report.

Question **report** **2.6.** In the same callback for the topic `robot_twist`, create a message `MotorSpeedsStamped`, fill in the time in the `header.stamp` attribute using `self.get_clock().now().to_msg()`, and fill the `left` and `right` fields with the speeds you set on the respective motors. Include a copy of this part of the code in your report.

Question **video** **2.1.** Run the `motor_command` node together with the node `zigzag_op` (provided in the repository). This should make the robot follow forward-moving arches. If you are using a real robot, make sure to have plenty of space before running the nodes.

Question **video** **2.2.** Same as the previous question, but this time run also the commands `ros2 topic echo /robot_twist` and `ros2 topic echo /motor_speeds`, and record their output; make sure that the outputs are readable in the video. For best results, use a screen recorder.

Hint for question node 1.1: In general, the frequency of the messages on the published topic should constant independently of the frequency of the messages on the subscribed topic. For instance, in the example of Table 1b, the second message contains only two strings, since, in the previous three seconds, only two messages were given on the subscribed topic, see Table 1a. Moreover, the timing of the published messages is exactly once every 3 s, despite the fact that the messages on the subscribed topic are received at irregular times.

To achieve this behavior, the commands for publishing should not appear in the callback.

Hint for question video 1.1: See hint for question node 1.1.

Hint for question report 2.5: The values of `speed_linear` and `speed_angular` will be found, respectively, in the `linear.x` and `angular.z` field of the `Twist` message received from the callback. These value should be used as inputs to the function `twist_to_speed()`. The results then will be used in the two calls to the method `.set_speed()` of the two motors, and also to populate the fields of a message of type `MotorSpeedsStamped`. You will need to import the message type in order to be able to use it:

```
from me416_msgs.msg import MotorSpeedsStamped
```

Hint for question report 2.6: See hint above. Moreover, note that the message type `MotorSpeedsStamped` expects `double` values for the fields `left` and `right`, and are not related to objects of the class `MotorSpeed`.