ME416 Homework 3 Report

Massinissa Bosli, Jilin Zheng

March 29, 2024

Problem 1: Improved twist_to_speeds()

Question 1.1

Expanding the equation $\dot{z} = A(\theta)u$, we get:

$$speed\_linear \cdot \cos(\theta) = \frac{k}{2} \cdot \cos(\theta) \cdot u_{LW} + \frac{k}{2} \cdot \cos(\theta) \cdot u_{RW} \tag{1}$$

$$speed\_linear = \frac{k}{2} \cdot u_{LW} + \frac{k}{2} \cdot u_{RW} \tag{2}$$

$$\frac{2}{k} \cdot speed\_linear = u_{LW} + u_{RW} \tag{3}$$

$$speed\_angular = \frac{k}{2d} \cdot u_{RW} - \frac{k}{2d} \cdot u_{LW} \tag{4}$$

$$\frac{2d}{k} \cdot speed\_angular = -u_{LW} + u_{RW} \tag{5}$$

where equation (3) is the result of the first two rows of $\dot{z} = A(\theta)u$ expanded and equation (5) is the result of the last row of $\dot{z} = A(\theta)u$ expanded.

Setting up equations (3) and (5) in matrix-form, we get:

$$\begin{bmatrix} \frac{2}{k} \cdot speed\_linear \\ \frac{2}{k} \cdot speed\_linear \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} u_{LW} \\ u_{RW} \end{bmatrix} \tag{6}$$

To isolate, $\begin{bmatrix} u_{LW} \\ u_{RW} \end{bmatrix}$, we calculate the inverse of $\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$:

$$\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}^{-1} = \frac{1}{1-(-1)} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix} \tag{7}$$

And finally, we obtain:

$$\begin{bmatrix} u_{LW} \\ u_{RW} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} \frac{2}{k} \cdot speed\_linear \\ \frac{2}{k} \cdot speed\_linear \end{bmatrix} = \begin{bmatrix} \frac{1}{k} \cdot speed\_linear - \frac{d}{k} \cdot speed\_angular \\ \frac{1}{k} \cdot speed\_linear + \frac{d}{k} \cdot speed\_angular \end{bmatrix} \tag{8}$$

Intuitively,

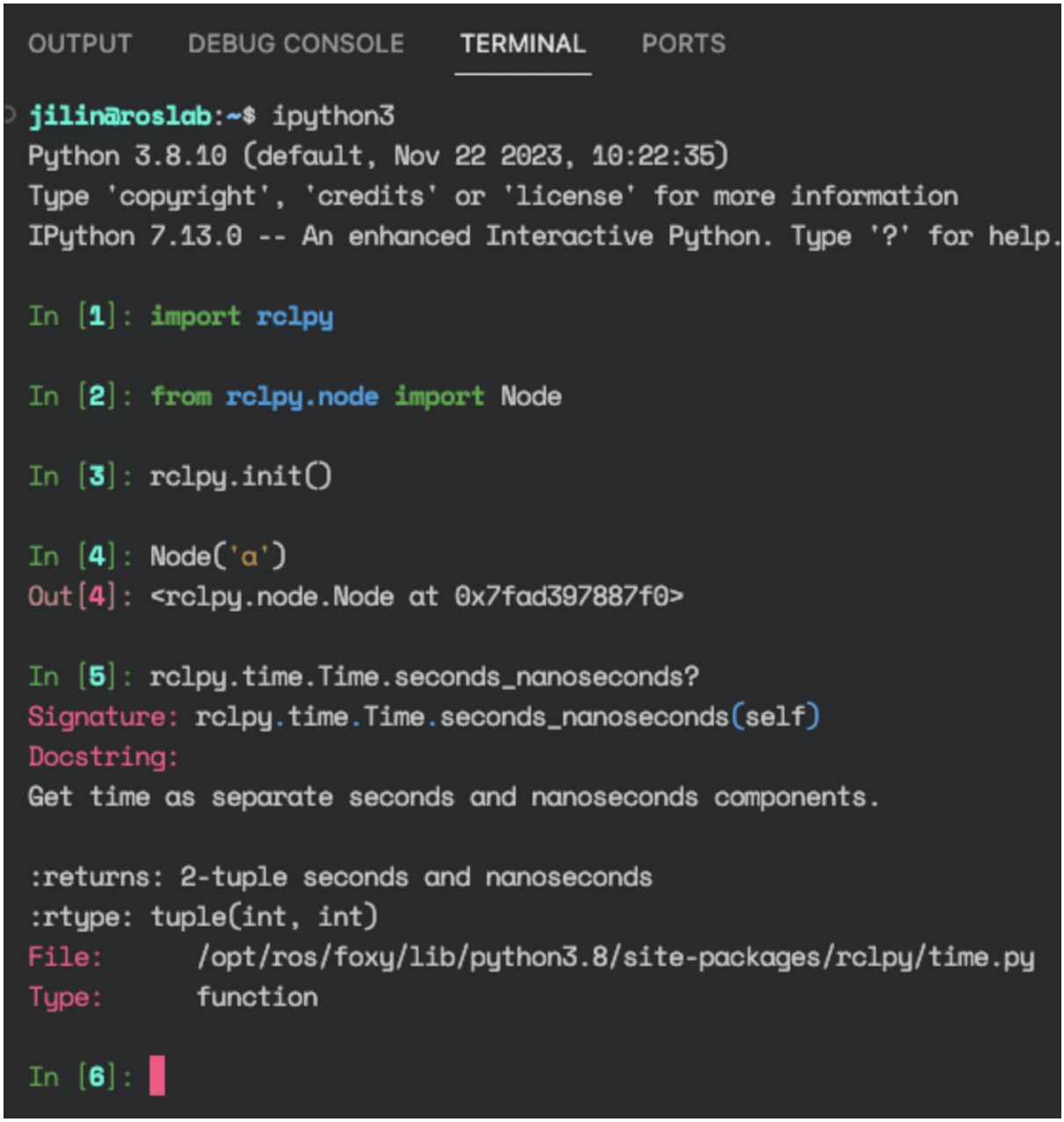
- Decreasing both k and d: will increase the linear speed
- Decreasing k and increasing d: will increase the linear speed
- Increasing k and decreasing d: will decrease the linear speed
- Increasing both k and d: will decrease the linear speed
- And for all the above cases,
  - Angular/turning speed will increase if the resulting $\frac{d}{k} > \frac{1}{2}$
  - Angular/turning speed will stay the same if the resulting $\frac{d}{k} = \frac{1}{2}$
  - Angular/turning speed will decrease if the resulting $\frac{d}{k} < \frac{1}{2}$

Question 1.2

- k=1, d= 0.5
- k=0.75, d=0.35
- k=0.5, d=0.1
- k=0.5, d=0.075
- k=0.5, d=0.09
- k=0.35, d =0.075
- k=0.35, d=0.05
- k=0.35, d=0.06
- k=0.35, d=0.065
- k=0.35, d=0.062
- k=0.35, d=0.075
- k=0.35, d=0.07
- k=0.35, d=0.068
- k=0.35, d=0.067
- k=0.35, d=0.065
- k=0.35, d=0.063
- And...**FINAL CONFIGURATION: k=0.35, d=0.06**

Problem 2: Utility Class StampedMsgRegister

Question 2.1



The method seconds_nanoseconds() gets the time as separate seconds and nanoseconds components (both ints) for an object of type rclpy.time.Time.

Question 2.2

```python
class StampedMsgRegister():
    """ Computes the delay, in seconds, betweeen two ROS messages. """
    def __init__(self):
        """
        Initialize the internal variables msg_previous to None
        """
        self.msg_previous = None

    def replace_and_compute_delay(self, msg):
        """
        Given a new stamped message as input,
        computes the delay (in seconds) between
        the time stamp of this message and the value in time_previous,
        and then replaces the internal copy of the previous message with the current message.
        """
        # save old message to msg_previous
        msg_previous = self.msg_previous

        # only calculate the time delay if msg_previous is not None
        if msg_previous is not None:
            time_delay = stamp_difference(msg.header.stamp, msg_previous.header.stamp)
        # set the time_delay to a tuple of 0 seconds and nanoseconds
        # if there was no previous message
        else:
            time_delay = (0,0)

        # save new message to previous message
        self.msg_previous = msg

        return time_delay, msg_previous

    def previous_stamp(self):
        return None if self.msg_previous is None else self.msg_previous.header.stamp
```

First, we save the old message (self.msg_previous) to a local variable within the function, msg_previous. Then, we calculate the time_delay via the stamp_difference() function ONLY if the msg_previous is NOT None, i.e., we had a previous message stored. If msg_previous is None, we just set the time_delay to a tuple (0,0) for 0 seconds and 0 nanoseconds since the last message (there never existed a message). Finally, we save the new msg to the object internal variable self.msg_previous and return the time_delay and msg_previous.