ROS Homework

Autonomous Robotics Labs

Labs 02 (26.2./28.2. 2019)

Plotting service

1. Create a custom service definition file (let's call it Plot.srv) with two request variables: **xdata** and **ydata** (you can leave the response field empty). Both variables should be arrays of arbitrary length with base type float32.
2. Create a script of a node providing a plotting service. The service should use the created service definition. The xdata will contain values on the X-axis and ydata will contain the corresponding Y-axis values.
3. Plot the data from the service request using the `plot(x, y)` function of the `pyplot` module from the `matplotlib` package. You might also need to use the `show()` function to show the plot.

Setting up the parameters

1. Next, you will need to set a couple of parameters.
2. First, set a global parameter with the name "**plot_length**" specifying the number of values that shall be collected and plotted (i.e. this will be the length of xdata and ydata)
3. Set a second global parameter called "**start_time**" that will hold the starting timestamp of the ROS bag recording (more details below).

Reading messages from a bagfile

1. Download the bagfile from the labs web page (link should be next to the link to this document)
2. Check the contents of the bagfile:

```
rosbag info <bag_name>
rqt_bag <bag_name>
```

3. Find topic name and message type corresponding to data from a laser scanner. This is a device that takes range measurements using laser ranging technique. The laser is incrementally (but very quickly) rotated and a measurement is taken after each increment. Thus, a single scan contains multiple range measurements taken at different angles.
   Based on the data (ROS message) type, try to find documentation for the laser scanner data (e.g. http://docs.ros.org/melodic/api/).

4. Create a node that will listen for the messages from the laser scanner and process them in the following way:

   (a) Filter out possibly erroneous data (i.e. outside of the normal value range – see the message documentation)

   (b) Discard measurements taken at an angle greater than 30° or lower than −30° (i.e. $abs(angle) \leqq 30°$)

   (c) Compute the mean of the remaining values and store it into a buffer (you can use *Python list* or *numpy array*)

   (d) Store the message timestamp as well (you can store it as a single float using the `to_sec()` function)

   (e) If the number of stored values had reached the number specified in the *plot_length* global parameter:

      i. Stop accumulating the data
      ii. Create an instance of the PlotRequest
      iii. Fill the xdata with the timestamps minus the value in the global parameter *start_time* – that is, each value should represent the number of seconds from the beginning of the bagfile.
      iv. Fill the ydata with the computed means
      v. Call the plotting service

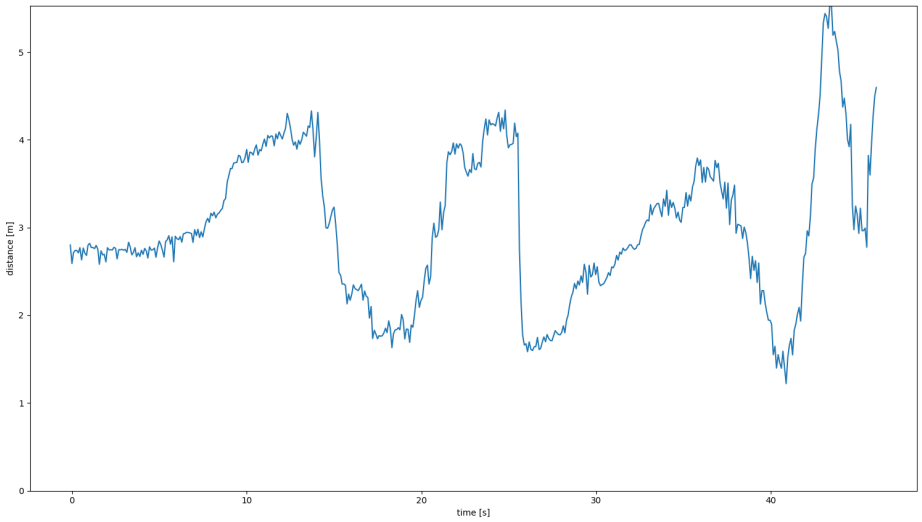
5. Run all the nodes and then play the bagfile (`rosbag play`).

Hints

- You can find most of the necessary information within the presentations from Lab 01 and Lab 02.
- The rest could be found using the mentioned commands or by following the provided links.
- Check http://wiki.ros.org/rospy/Overview/Time if you need help with the timestamps.
- If you have problem with plotting, search for matplotlib plot examples or documentation
- When dealing with more complex nodes and persistent variables, it is useful to create a class that will handle all of the data processing. In the case of the listener node, the structure of your script should look like this:

```
#!/usr/bin/env python2
import rospy
import numpy as np
# TODO import the service definitions

class ScanCollector():
    SERVICE_NAME = "plot"

    def __init__(self):
        # Initialize the node here
        # wait until the self.SERVICE_NAME service is available
        # create service proxy and store it as this object's variable ~ object property
        # (i.e. self.something = ...)
        # retrieve the necessary parameters from the parameter server
        # and store them into variables
        # create the listener object and assign a class method as the callback
        # possibly some additional stuff

    def scan_callback(self, msg):
        # process the message
        # if enough data has been collected, call the plotting service

if __name__ == '__main__':
    sc = ScanCollector()
    rospy.spin()
```

- Here is an example of the resulting plot (extracted from the bag with *plot_length* set to 500):



- Your plot might differ a little, depending on the parameters. You also don't need to tune value ranges of the plot axis and the labels (but the plot will look nicer if you do!)