****

**Embedded Visual Control**

Assignment 1

|  |
| --- |
| **Grade: Individual (10%)** |
| **Title:** Introduction to ROS |

**Instructions**

After launching the template ROS node, you can develop your own nodes. Create a copy of the template\_ws directory and name it worshop1\_<student\_id> under the workshops folder. Inside this dedicated workspace for the first Workshop, you will create a new package. This package will contain two nodes, one publishing node and one subscribing node. The nodes functionalities will be the following:

1. Publisher node: The publisher will send topic messages of type UInt8 using a publishing rate of 1Hz and a queue size of 1. The topic value will be an incrementing value from 0 to the maximum representable UInt8 value. When reaching the maximum representable number, the message should start again from 0.
2. Subscriber node: The subscriber will listen to the same topic with a queue size of 1.

|  |  |
| --- | --- |
| **Assessment Criteria:**   * Short and concise answers * Arguments must be supported by comprehensive visuals (and logging information) * Code quality | **Submission Guidelines:**   * Each student must submit their personal workshop directory (in zipped format) with their report inside * Answers are written in the space between questions |

**Questions**

1. **Subscriber Functionality:** Make the callback of the subscriber trying to predict the newcoming value of the publisher knowing it will be always incrementing by 1.
   1. Illustrate the ROS architecture. (**1pts**)

|  |
| --- |
| Architecture using rqt\_graph |

* 1. Log the expected and received values. For visualization purposes, log unexpected values as warnings. Plot the received message values w.r.t. time. (**1pts**)

|  |
| --- |
| Plotted messages w.r.t time  \*\* Of note: in the legend the Receiver and Subscriber refer to the same node |

* 1. Are they what you expect? Calculate the drop rate based on the expected messages and the received ones. (**1pts**)

|  |
| --- |
| This is expected since the frequency at which the publisher publishes is low enough for the subscriber to be able to receive all messages correctly.  The drop rate is 0%, because every message was received through the topic (none were lost). |

1. **Increasing Publish Rate:** Now increase the publish rate to 10Hz.
   1. Does anything change related to the messages received? (**1pts**)

|  |
| --- |
| The only difference is that the messages are received 10 times faster.  Plotted messages with respect to time |

* 1. Calculate the drop rate. (**1pts**)

|  |
| --- |
| The drop rate is 0% as no message are lost (no warnings) |

1. **Increasing Subscriber Execution Time:** Add artificial delay to the subscriber’s callback to achieve a service rate of 1Hz.
   1. Visualize the received message values. (1pts)

|  |
| --- |
| Visualised received messages: |

* 1. Calculate the drop rate. (**1pts**)

|  |
| --- |
| The drop rate (for when the publisher is sending at 10 hertz) defined as number of messages not received divided by the number of messages sent.  General equation: 100 \* No.messages.lost / No.messages.sent  For this case it can be seen that the drop rate is 90% (for every 1 message received, 9 messages are dropped, therefore the equation is 100\* 9/10) |

1. **Increasing Subscriber’s queue:** Increase the size of the subscriber queue to 10 items.
   1. What happens to the logged messages? Calculate the drop rate. (**1pts**)

|  |
| --- |
| In the logged messages, the Subscriber saves the first 10 messages that are published while the subscriber is artificially delayed (when the subscriber is delayed).  The drop rate in this case would be also 90% as in the previous example, because for every 10 messages that are received by the subscriber, 90 are dropped. |

* 1. Visualize the received message values. Can you notice a pattern in the messages correctly predicted and dropped? (**2pts**)

|  |
| --- |
| The main pattern that I can see is that the first 11 messages are always correct, and after that the incoming message values are incorrectly classified. 11 messages are correct because after receiving the first message, the Subscriber is delayed artificially, but then it still stores the next 10 incoming messages it detected.    \*\* Of note, was unable to find a way to showcase the visulalisation from the beginning because of how I think rqt\_plot works. As such I also showed the messages from the terminal. |

****