Faculty 3ME
DELFT UNIVERSITY OF TECHNOLOGY

ROS Exercise 2
2017-2018

Priadi Teguh Wibowo (4625862)

December 11, 2017

1 Intro

This report explains exercise in Chapter 4 and 6: Writing ROS programs from a set of exercises for the book "A Gentle Introduction to ROS". The first exercise asks the students to log the total message sent by `pubvel` node, total message dropped by `vel_filter` node and total message received by `vel_printer` node. The interval to log their message is 5 seconds wit `INFO` as their severity level. The second exercise asks the students create a launch file for all three nodes in the first exercise.

2 Implementation

We use `ROS_INFO_STREAM_THROTTLE` macro to generate log message because it has *interval* parameter which is specified in seconds. This parameter is set 5.0 seconds as requested by the exercise.
In `pubvel` node, the total message is incremented and the log message is generated inside the while loop after the node publishes its message. The implementation in `pubvel` node is shown by code snippet in Listing 1.

Listing 1: Log message in `pubvel` node

```
int total_msg = 0;
while(ros::ok())
{
  // Create and fill in the message. The other four
  // fields, which are ignored by turtlesim , default to 0.
  geometry_msgs::Twist msg;
  msg.linear.x = double(rand()) / double(RAND_MAX);
  msg.angular.z = 2 * double(rand()) / double(RAND_MAX) - 1;
  // Publish the message.
  pub.publish(msg);
  total_msg++;
```

```
    // Log message about total sent messages.
    ROS_INFO_STREAM_THROTTLE(5.0,"Total published messages
        pubvel: " << total_msg);
    // Wait until it's time for another iteration.
    rate.sleep();
  }
```

Meanwhile in `vel_filter` node, the total message is incremented every time the angular velocity is negative or if this node does not published any message. And the log message is generated inside the call back function `twistMessageFilter`. The implementation in `vel_filter` node is shown by code snippet in Listing 2.

Listing 2: Call back function in `vel_filter`

```
void twistMessageFilter(const geometry_msgs::Twist& msg)
{
  static int total_msg = 0;
  if(msg.angular.z >= 0)
  {
    // Publish the message.
    pub->publish(msg);
  } else {
    total_msg++;
  }
   // Log message about total dropped messages
  ROS_INFO_STREAM_THROTTLE(5.0,"Total dropped messages
      vel_filter: " << total_msg);
}
```

In `vel_printer` node, the total message is incremented and the log message is generated inside the call back function. Its implementation is shown by code snippet in Listing 3.

Listing 3: Call back function in `vel_printer`

```
void twistMessageReceived(const geometry_msgs::Twist& msg)
{
  static int total_msg_print = 0;
  total_msg_print++;
  ROS_INFO_STREAM_THROTTLE(5.0,"Total received messages
      vel_printer: " << total_msg_print);
}
```

The second exercise ask us to create a launch file for all three nodes before. The launch file is show by Listing 4.

Listing 4: Launch file for all three nodes

```
<launch>
  <node
```

```
    pkg="exercise_4"
    type="exercise_4_pubvel"
    name="publish_velocity"
    output="screen"
    launch-prefix="xterm -e"
  />
  <node
    pkg="exercise_4"
    type="exercise_4_vel_filter"
    name="filter_velocity"
    output="screen"
    launch-prefix="xterm -e"
    required="true"
  />
  <node
    pkg="exercise_4"
    type="exercise_4_vel_printer"
    name="subscribe_velocity"
    output="screen"
    launch-prefix="xterm -e"
  />
</launch>
```

Each attributes in launch file above are explained below.

- `pkg` specifies the package
- `type` specifies the executable file name
- `name` assigns the node's name
- Assigning attribute `output` by `"screen"` allows the node to display their standard output on console instead in log files.
- Assigning attribute `launch-prefix` by `"xterm -e"` allows the node to create separate terminal for each nodes.
- Using attribute `required` at certain node allows terminating other nodes when that node is terminated.

3 Result

Figure 1 shows the log message in `pubvel`, `vel_filter` and `vel_printer` nodes respectively. As the figures show, The total reported by `pubvel` is not always equal as the total reported by `vel_filter` and `vel_printer` as shown by table 1. This inequality happens because the publisher-subscriber relationship between `vel_printer`, `vel_filter` and `pubvel` or event-based trigger to print the messages. Thus, sometimes log messages `vel_printer` or `vel_filter` are not fully synchronized or not

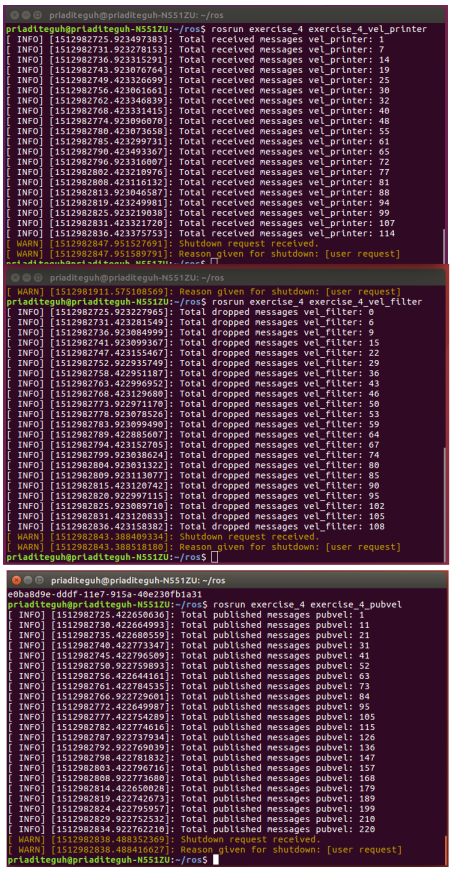

Figure 1: Each terminal running the three nodes

Table 1: Difference total message between `pubvel` and `vel_filter+vel_printer`

No.	Total message vel_filter + vel_printer	Total message pubvel	Difference
1	1	1	0
2	13	11	2
3	23	21	2
4	34	31	3
5	47	41	5
6	59	52	7
7	68	63	5
8	83	73	10

on the ”same state” compared to log messages `pubvel`. Somehow there should be a acknowledgement system between the nodes.
For second exercise, a new package is created without copying all the previous codes because the previous package and executable files are specified in attribute `pkg` and `type` in which ROS program is expected to run.
Figure 2 shows three separated terminal created by the launch file. The total reported by `pubvel` and the total of messages reported by `vel_filter` and `vel_printer` are still not equal or roughly same as result without using launch file as shown by table 2.

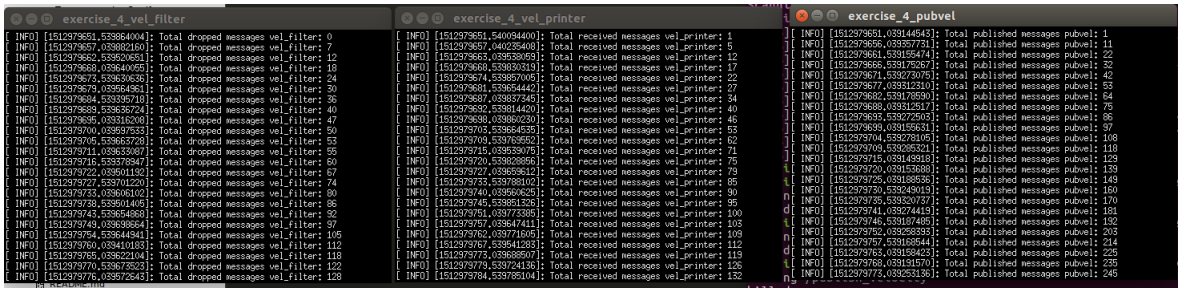

Figure 2: Output generated in console after `vel_filter` node terminated

Table 2: Difference total message between `pubvel` and `vel_filter`+`vel_printer` using launch file

No.	Total message vel_filter+vel_printer	Total message pubvel	Difference
1	1	1	0
2	12	11	1
3	24	22	2
4	35	32	3
5	46	42	4
6	57	53	5
7	70	64	6
8	80	75	5

If the terminal where `vel_filter` resides is closed, the other nodes will terminate. It happens because the attribute `required` is set `"true"` in `vel_filter` node. When that node is terminated, output is generated on console to notice that it will kill other nodes, as shown by figure 3.

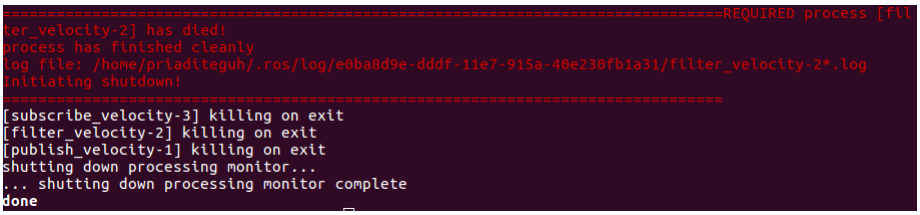

Figure 3: Output generated in console after `vel_filter` node terminated

4 Conclusion

This exercise introduces several fundamental concepts in ROS, which are :

- Logging system in ROS using `ROS_..._STREAM_..` macro can be viewed in log files or console.
- Launch file in ROS can start the master and many nodes at once.