Laboratory Report 3

Altynbek Aidarbekov, Azat Balapan, Azamat Turganbayev

1. Introduction

During this laboratory session, the main aim was to work with the 5 DOF planar robot and control a single joint. There are 4 main tasks that should be completed:

- Create a ROS node that will “listen” for std_msgs/Float64 type data and “publish” this data to the joint of the planar robot. The node should send the command to move if any new incoming value is higher than the previous one
- Get the step response of the base’s and end effector’s joint
- Get the sine-wave response of the base’s and end effector’s joint
- Decrease the Proportional gain of PID in the joints and repeat 2 previous steps

The task can be performed with the real robot or on the Gazebo simulator. The work was completed with the **real robot** during 2 laboratory sessions.

2. “Listener” ROS node

During the task 1, “listener.cpp” and “publisher.cpp” were used to execute the joint. An ID number’s digits were sent 1 by 1. The first number was saved in the *prevNum* variable and compared with the new one. If a new number was greater than the previous one, **joint4** was moved to the left by the value of 1. In order to avoid the clashing after several executions, during the next iteration, the direction was changed to the right. The algorithm was in progress until it had not been terminated in the terminal. The explanation of the code with robot movement can be found here.

53 lines (42 sloc) | 996 Bytes Raw Blame

```cpp
#include "ros/ros.h"
#include "std_msgs/Float64.h"
#include "std_msgs/String.h"

#include <math.h>

class SubscribeAndPublish
{
public:
        SubscribeAndPublish()
        {
                // Topic you want to publish
                pub_ = n_.advertise<std_msgs::Float64>("/motortom2m/command", 1000);

                // Topic you want to subscribe
                sub_ = n_.subscribe("chatter", 1, &SubscribeAndPublish::chatterCallback, this);
        }

        void chatterCallback(const std_msgs::String::ConstPtr &msg)
        {
                std_msgs::Float64 msg_to_send;

                double angle = 1;
                if (turnLeft)
                {
                        angle = -angle;
                }
                msg_to_send.data = angle;
                pub_.publish(msg_to_send);
                ROS_INFO("moving");
                turnLeft = !turnLeft;
        }

private:
        ros::NodeHandle n_;
        ros::Publisher pub_;
        ros::Subscriber sub_;

        bool turnLeft = true;
};

int main(int argc, char **argv)
{
        // Initiate ROS
        ros::init(argc, argv, "listener_rotater");

        // Create an object of class SubscribeAndPublish that will take care of everything
        SubscribeAndPublish SAPObject;

        ros::spin();

        return 0;
}
```

Figure 1. Screenshot of the listener.cpp code

3. Step Response of the joint

a. Base Joint

In order to obtain the step response of the base joint, the algorithm used in the previous task was simplified, where the joint moved from left to right by the value of 1. To move the base joint, the topic was published to *motortom2m/command.* The problem occurred and was explained by the Professor at the end of the lab. session. The thing is, during the execution of the motor on the base joint, the base was not fixed. As a result, there were considerable **deviations** between the goal and current position. The same issue was observed with sin wave response.

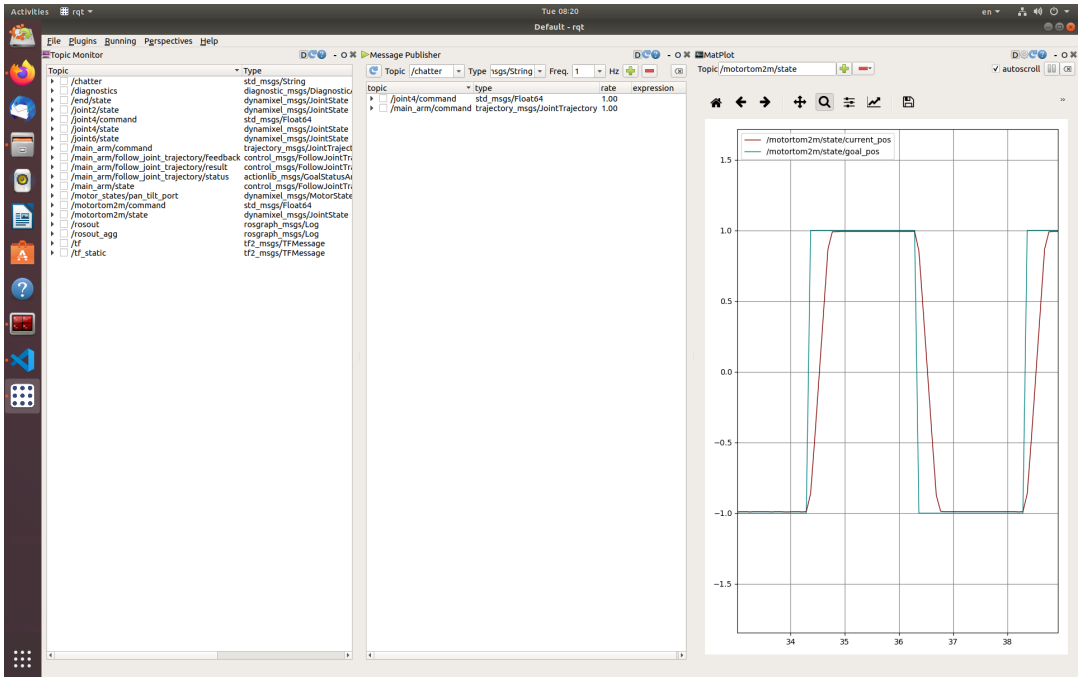

Figure 2. Step Response of Base Joint

b. End Effector Joint

During this part, the same procedure was done as with **Base Joint,** except the topic was published to *end/command.* The code explanation for both parts and the moving joint can be found here.

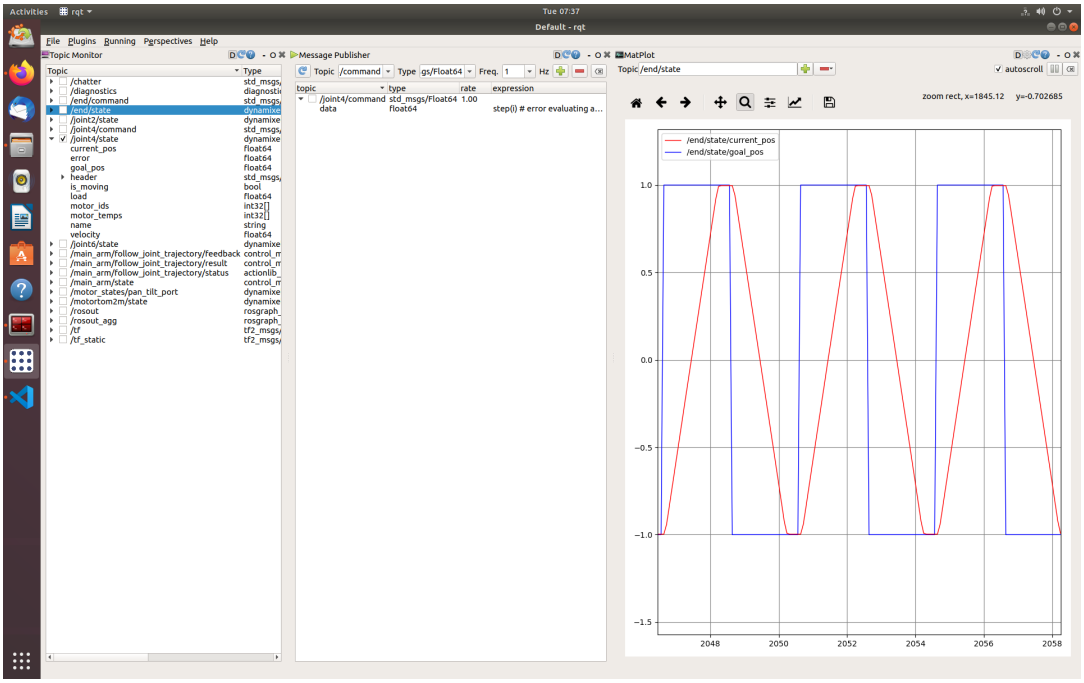

Figure 3. Step Response of End Effector Joint

4. Sine-wave response of the joint

a. Base Joint

To obtain the sine-wave response of the joint the expression for *motortom2m/command* was assigned as **sin(i)** at the rate with value **7**. The results can be seen below.

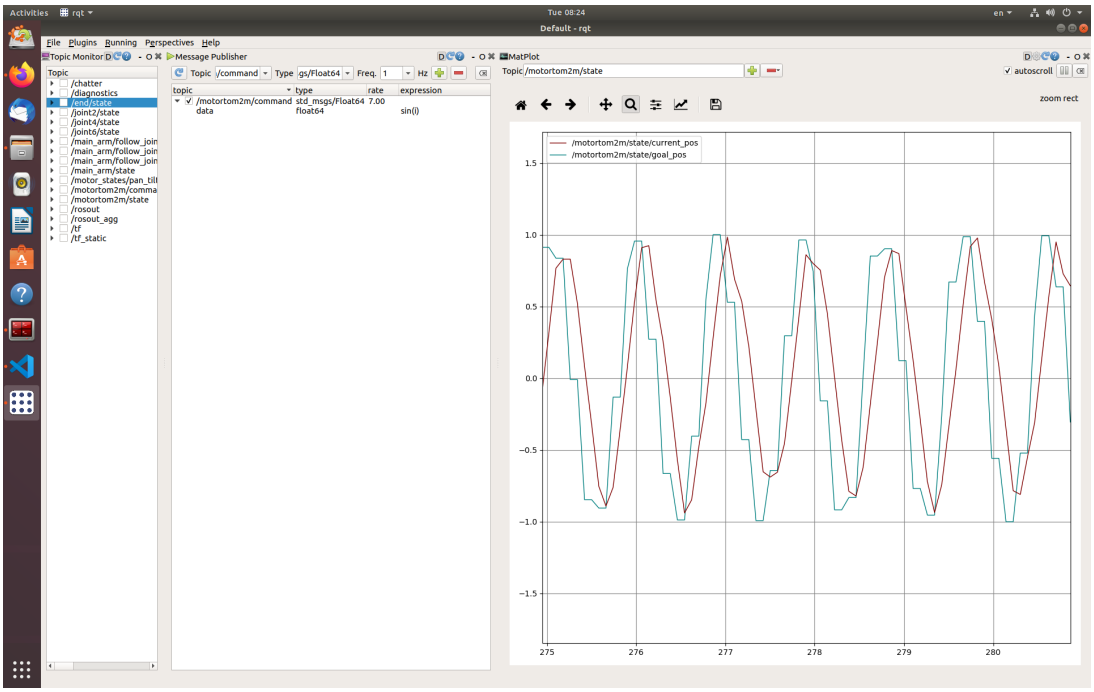

Figure 4. Sine-Wave Response of Base Joint

b. End Effector Joint

Getting the sine-wave response of the end effector joint was done the same as to the base joint. The assigned values were the same.

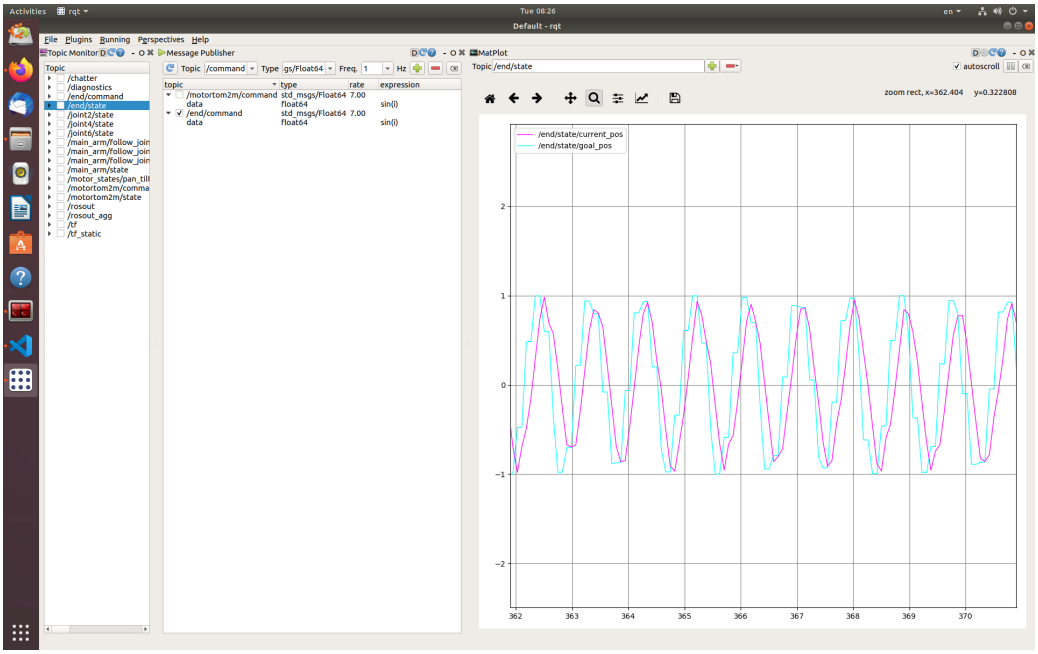

Figure 5. Sine-Wave Response of End Effector Joint

5. Proportional gain of PID

This task was not done due to the fact that all tasks were done on the real robot. It can be possible to do it on Gazebo simulation. However, in order to perform it in real life, the tuning of the motors is required.

6. Conclusion

To sum up, all tasks except for decreasing the proportional gain were done successfully. All work was done during the laboratory sessions on *06/09/2022* and *13/09.2022*. The issues occurred with gaining the step response of the base joint since the base itself was not fixed. The code for tasks can be found in the GitHub repository.