Storing and Visualizing ROS Message Data Using Rosbag Package

Moldir Zabirova, Valeriya Kostyukova, Daryn Kenzhebek

I. Node to Publish End-Effector Coordinates Using MoveIt Library

In the previous laboratory, we configured the MoveIt library to control the snake robot in the Cartesian space. We published the coordinates for the robot's end-effector to move in a rectangular path in the XY plane.

In order to get the position of the robot's end-effector we created a C++ script. Using MoveIt, we can retrieve the x- and y-coordinates of the end-effector and store them in a Float64 multi-array, named "new_msg". For the five joint angles, chattedCallback() function subscribes to the "/joint_states" topic where it receives the joint state messages. The joint angles was similarly stored in "new_msg_2" multi-array and continuously published (Fig. 1).

```
std_msgs::Float64MultiArray new_msg;
std_msgs::Float64MultiArray new_msg_2;

void chatterCallback(const sensor_msgs::JointState::ConstPtr& msg) {
        ROS_INFO("I heard: [%f, %f, %f, %f, %f]", msg->position[0],msg->position[1],msg->position[2],msg->position[3],msg->position[4]);
        new_msg_2.data.resize(4);
        new_msg_2.data = msg->position;

}


int main(int argc, char **argv)
{
        ros::init(argc, argv, "publisher");
        ros::NodeHandle node_handle;
        ros::AsyncSpinner spinner(0);
        ros::Rate loop_rate(50); //Frequency

        ros::Publisher pub = node_handle.advertise<std_msgs::Float64MultiArray>("/publisher", 1000);
        ros::Subscriber sub = node_handle.subscribe("/joint_states", 1000, chatterCallback);
        //ros::Subscriber sub_1 = node_handle.subscribe("/joint2/state", 1000, chatterCallback);

        spinner.start(); // For moveit implementation we need AsyncSpinner, we cant use ros::spinOnce()
        //ros::Subscriber sub = node_handle.subscribe("/positions", 1000, chatterCallback);
        static const std::string PLANNING_GROUP = "snake_robot"; /* Now we specify with what group we want work, here group1 is the name of my group controller*/
        moveit::planning_interface::MoveGroupInterface move_group(PLANNING_GROUP); // loading move_group

        const robot_state::JointModelGroup *joint_model_group =move_group.getCurrentState()->getJointModelGroup(PLANNING_GROUP); //For joint control
        geometry_msgs::PoseStamped current_pose;

        current_pose = move_group.getCurrentPose();

        while (ros::ok()){
                // std_msgs::Float64MultiArray array;
                current_pose = move_group.getCurrentPose();
                new_msg.data.resize(2);
                new_msg.data[0] = current_pose.pose.position.x;
                new_msg.data[1] = current_pose.pose.position.y;
                new_msg.data = new_msg.data;
                pub.publish(new_msg);
                new_msg_2.data = new_msg_2.data;
                pub.publish(new_msg_2);

                ros::spinOnce();
                loop_rate.sleep();


        }

        ROS_INFO("Done");
        ros::shutdown();

        return 0;
}
```

Fig. 1. Node to retrieve and store position and joint angles data.

We can see the response of the joints in rqt in a form of a plot (Fig. 2). This helps to visualize the history of the joint angles during the movement.

The next task is to record and plot such log file that will include the joint angles and the positions of the end-effector.

II. Record of Topics into .rosbag and Conversion to .csv

Rosbag is a tool to record and playback the robot simulation. This recorded data can include information such as sensor readings, images, and other messages exchanged between different parts of a robotic system. Once recorded, the rosbag file can be replayed to analyze or test algorithms without the need to have the actual robot or sensors present.

To record the position of the end-effector in X and Y-axes, we run the node to extract the coordinates and the node for the end-effector to draw a rectangular path (Fig. 3).

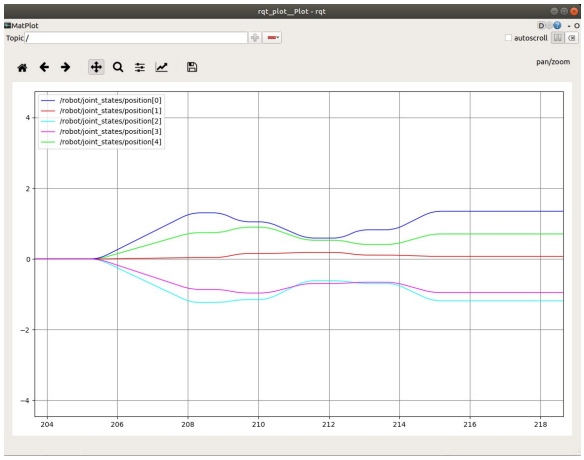

Fig. 2. Robot Response Visualized in rqt

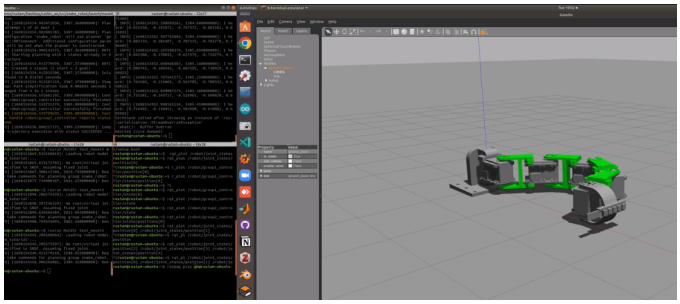

Fig. 3. Snake Robot Drawing a Rectangular Path in Gazebo

Then, using

```
rosbug record /publisher
```

we can record this data to a new folder outside our workspace.

To visualize the data we need to convert rosbag file to CSV (Fig. 4) with

```
rostopic echo /publisher -b joints.bag
-p > joints.csv.
```

III. Plotting X- and Y-Coordinates and Joint Angles in MATLAB

We used MATLAB to plot and visualize the data of end-effector position and joint angles changes (Fig. 5, 6).

The estimated output rectangular trajectory can be seen on Fig. 5.

	A	B	C	D	E	F
1	220239000000	3.22699869063	-0.0100997437935			
2	220239000000	5.74312686918E-05	-6.86614055674E-05	-4.53540495595E-05	-7.35170946449E-05	7.34010098977E-05
3	220239000000	3.22699869063	-0.0100997437935			
4	220239000000	5.7431268691E-05	-6.86614055647E-05	-4.53540495595E-05	-7.35170946458E-05	7.34010098977E-05
5	220338000000	3.22699869063	-0.0100997437935			
6	220338000000	5.7431268691E-05	-6.86614055647E-05	-4.53540495595E-05	-7.35170946458E-05	7.34010098986E-05
7	220438000000	3.22699869063	-0.0100997437935			
8	220438000000	5.7431268691E-05	-6.86614055647E-05	-4.53540495595E-05	-7.35170946458E-05	7.34010098986E-05
9	220539000000	3.22699869063	-0.0100997437935			
10	220539000000	5.7431268691E-05	-6.86614055665E-05	-4.53540495595E-05	-7.35170946458E-05	7.34010098977E-05
11	220539000000	3.22699869063	-0.0100997437935			
12	220539000000	5.7431268691E-05	-6.86614055665E-05	-4.53540495595E-05	-7.35170946458E-05	7.34010098977E-05
13	220638000000	3.22699869063	-0.0100997437935			
14	220638000000	5.7431268691E-05	-6.86614055665E-05	-4.53540495595E-05	-7.35170946458E-05	7.34010098977E-05
15	220638000000	3.22699869063	-0.0100997437935			
16	220638000000	5.74312686918E-05	-6.86614055656E-05	-4.53540495595E-05	-7.35170946458E-05	7.34010098977E-05
17	220739000000	3.22699869063	-0.0100997437935			
18	220739000000	5.74312686918E-05	-6.86614055656E-05	-4.53540495595E-05	-7.35170946458E-05	7.34010098977E-05
19	220837000000	3.22699869063	-0.0100997437935			
20	220837000000	5.74312686901E-05	-6.86614055647E-05	-4.53540495595E-05	-7.35170946458E-05	7.34010098977E-05
21	220838000000	3.22699869063	-0.0100997437935			
22	220838000000	5.7431268691E-05	-6.86614055656E-05	-4.53540495595E-05	-7.35170946449E-05	7.34010098977E-05
23	220939000000	3.22699869063	-0.0100997437935			
24	220939000000	5.7431268691E-05	-6.86614055656E-05	-4.53540495595E-05	-7.35170946449E-05	7.34010098977E-05
25	221037000000	3.22699869063	-0.0100997437935			
26	221037000000	5.74312686901E-05	-6.86614055665E-05	-4.53540495595E-05	-7.35170946449E-05	7.34010098977E-05
27	221037000000	3.22699869063	-0.0100997437935			
28	221037000000	5.74312686901E-05	-6.86614055665E-05	-4.53540495595E-05	-7.35170946449E-05	7.34010098977E-05
29	221138000000	3.22699869063	-0.0100997437935			
30	221138000000	5.7431268691E-05	-6.86614055656E-05	-4.53540495595E-05	-7.35170946458E-05	7.34010098986E-05

Fig. 4. CSV File Data: Positions and Angles Alternate in Rows

```
data_position = readmatrix("joints.csv");

time1 = data_position(1:2:end,1);
x = data_position(1:2:end, 2);
y = data_position(1:2:end,3);

x_norm = normalize(x);
y_norm = normalize(y);

figure(1);

plot(time1,y_norm,time1,x_norm);
legend('x', 'y');
xlabel('time(s)');
ylabel('position');
title('Plot of Normalized Position Values of End-Effector');

% Extract columns
time2 = data_position(2:2:end, 1);
motortom = data_position(2:2:end, 2);
joint2 = data_position(2:2:end, 3);
joint4 = data_position(2:2:end, 4);
joint6 = data_position(2:2:end, 5);
end_effector = data_position(2:2:end, 6);

% Plot the data
figure(2);
plot(time2, motortom, time2, joint2, time2, joint4, time2, joint6,
legend('motortom', 'joint2', 'joint4', 'joint6', 'end');
xlabel('time(s)');
ylabel('angle(rad)');
title('Plot of Joint Angles');
grid on;
```

Fig. 5. MATLAB Code to Visualize CSV Data

IV. Conclusion

In this lab project we used a configured MoveIt library to control a snake robot in Cartesian space, publishing coordinates for the robot's end-effector to follow a rectangular path in the XY plane. Our node written as a C++ script utilized MoveIt and the "/joint_states" topic to retrieve and store the end-effector's x- and y-coordinates as well as the joint angles, respectively. The recorded data was then stored in a rosbag file. This file captures essential information such as sensor readings and message exchanges, enabling offline analysis and testing of algorithms without the need for the physical robot. The rosbag was subsequently converted to a CSV format for visualization. MATLAB was employed to plot and analyze the end-effector's position and joint angles over time, revealing a clear trajectory of the robot's movement. Future work may involve further exploration of different robot

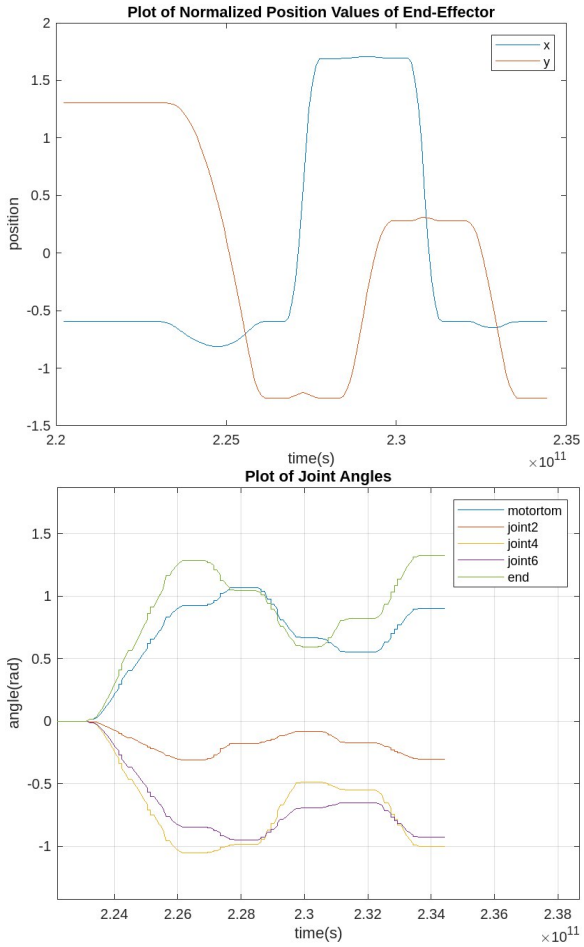

Fig. 6. Plotting X- and Y-Coordinates and Joint Angles in MATLAB.

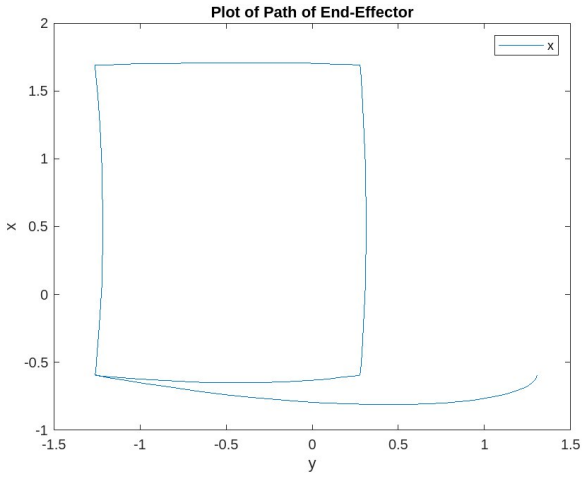

Fig. 7. Plot of the Path of End-Effector

trajectories and the application of advanced control strategies for more complex movements.