ROS: Data visualization

The objective of this lab is to learn how to create data publisher for graphical visualization in RViz.

Setting up your computer

Open terminal window.

Clone your git repository to your home folder:

```
$ git clone URL_for_<yourname>-rtech_repository
```

Use `ls` to confirm that ***<yourname>-rtech*** has been downloaded to your home folder.

Brief introduction

This lab will be the first of two labs that focus on creating a sensor driver in a ROS-based system.

On a general level, the pipeline for graphically visualizing sensor data is depicted in Figure 1. The physical sensor is connected to your computer through a supported communication interface. ROS driver node reads the data from that communication interface and publishes it in a proper format as ROS messages. Even though the ROS driver node can publish the data in a format that is suitable for visualization, it might be smart to create a separate node that handles visualization (i.e. keeping ROS nodes as small functional units that can be reused as the hardware/sensor changes). For example, ROS driver node is often specific to manufacturer whereas the data visualizer node is specific to message type.

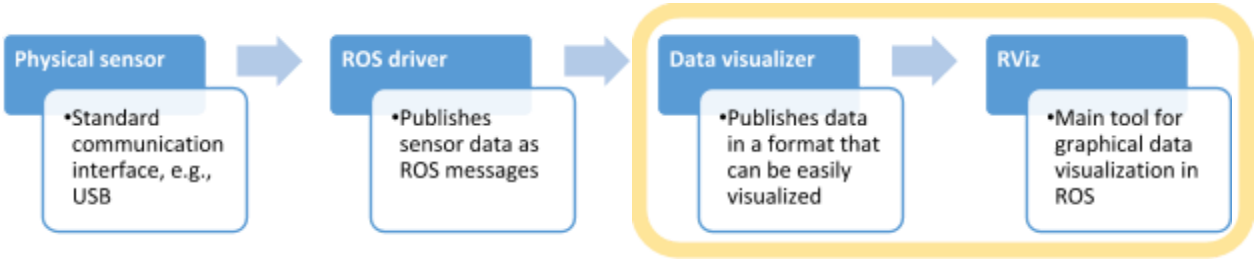

Figure 1. *The pipeline for visualizing sensor data in RViz.*

Creating a data visualization package that uses RViz

In this lab you will be working on the data visualization part in Figure 1 (the inside of yellow frame). The tasks in this labs are as follows.

(1) Create a ROS package for publishing messages containing ranging information
(2) Create an RViz configuration for visualizing the messages (range) graphically
(3) Set up an easy-to-use launch file for all of it

First, create a new ROS package, name it `range_visualizer` and have the package depend on `std_msgs`, `roscpp`, `rospy`, and `sensor_msgs`.

Since RViz has a built-in display type for visualizing range (http://wiki.ros.org/rviz/DisplayTypes), create a new ROS node that publishes data as sensor_msgs/Range (http://wiki.ros.org/sensor_msgs).

Let us assume that we have a ultrasonic sensor with a range of 2 cm to 4 m and a field of view of 45°. As we are currently only interested in getting the publishing and visualization to work, use randomly generated values for the actual range.

After you have successfully created the node (i.e. it continuously publishes Range messages with random values for range), run RViz and add Range display type to your RViz configuration. Are you now able to graphically see the data that is being published by your data visualizer node? Did you define a frame_id for your Range messages? Save the configuration file in within your `range_visualizer` package.

Set up a launch file that simultaneously runs your visualizer node and RViz with the saved configuration.

Even though the visualization should be working as-is, you may notice that RViz also displays a warning message related to TF. The warning is displayed because your frame of reference does not have a clearly defined place in the transform tree. We can solve this problem by publishing a transform between our reference frame and a virtual world frame. Update your launch file to publish a static transform (http://wiki.ros.org/tf#static_transform_publisher) between *map* and your frame of reference. You can set this transform at zero translation and rotation.

Play around with field of view, radiation type, and min and max range variables. Try to understand how they influence your visualization.

At the end of the lab, you should have the following solution to present. By using a single launch file, the RViz opens and displays a cone of range that represents the messages being published by your visualization node. No errors or warnings related to TF should be present.

Can you explain the meaning and effect of every variable in the sensor_msgs/Range?

Show the final result to the instructor.

CLEAN UP YOUR WORKSPACE

NB! Before you leave the lab, make sure you have pushed all the files in your catkin workspace to your git cloud service.

In terminal, `cd` to **<yourname>-rtech**

Type

```
git config user.email "youremail@example.com"
```

Type

```
git status
```

You should now see all the new and modified files in red.

Prepare the relevant files for the commit.

```
git add file_name_in_red1 file_name_in_red2
```

When you now type

```
git status
```

you should see all the added files in green. You are now ready to commit changes. Type

```
git commit –m "Insert a brief explanation"
```

Your changes have now been committed but not yet uploaded to the cloud. To upload your files, type

```
git push
```

In your web browser, **verify that all the files** have been uploaded to the **<yourname>-rtech** repository.

Delete the **<yourname>-rtech** folder and any other files you created from the lab's computer.