Tutorial on ROS

Olivier Aycard
Professor
Grenoble INP - PHELMA
GIPSA Lab
https://www.gipsa-lab.grenoble-inp.fr/user/olivier.aycard
olivier.aycard@grenoble-inp.fr

Outline

1. Our first node in ROS;
2. Our second node in ROS;
3. Compile and run nodes.

Our first node

- Edit *laser_text_display_node.cpp* in *~/catkin_ws/src/tutorial_ros*;

```
class laser_text_display {

private:
    ros::NodeHandle n;

    ros::Subscriber sub_scan;

    // to store, process and display laserdata
    int nb_beams;
    float range_min, range_max;
    float angle_min, angle_max, angle_inc;
    float r[1000], theta[1000];
    geometry_msgs::Point current_scan[1000];

    bool new_laser;//to check if new data of laser is available or not
```

I declare a node

This node will subscribe to one type of messages

Variables to store and process the laser data

Our first node

- Our first node will subscribe to the message called « scan »;
- « scan » is the message published by the laser and containing the laser data;
- Each time a new message « scan » is published, our first node will receive and store this message with the method « scanCallback »

```
sub_scan = n.subscribe("scan", 1, &laser_text_display::scanCallback, this);
```

Our first node

```
void scanCallback(const sensor_msgs::LaserScan::ConstPtr& scan) {

    new_laser = true;
    // store the important data related to laserscanner
    range_min = scan->range_min;
    range_max = scan->range_max;
    angle_min = scan->angle_min;
    angle_max = scan->angle_max;
    angle_inc = scan->angle_increment;
    nb_beams = ((-1 * angle_min) + angle_max)/angle_inc;

    // store the range and the coordinates in cartesian framework of each hit
    float beam_angle = angle_min;
    for ( int loop=0 ; loop < nb_beams; loop++, beam_angle += angle_inc ) {
        if ( ( scan->ranges[loop] < range_max ) && ( scan->ranges[loop] > range_min ) )
            r[loop] = scan->ranges[loop];
        else
            r[loop] = range_max;
        theta[loop] = beam_angle;

        //transform the scan in cartesian framework
        current_scan[loop].x = r[loop] * cos(beam_angle);
        current_scan[loop].y = r[loop] * sin(beam_angle);
        current_scan[loop].z = 0.0;
    }

}//scanCallback
```

The type of messages published by the laser

Our first node

- Our first node is an infinite loop that will run at 10 hz

1. It will check if a new message from the laser has been published;
2. It will call the method scanCallback to collect the data of the laser;
3. The method « update » will proceed the data of the laser

```
//INFINTE LOOP TO COLLECT LASER DATA AND PROCESS THEM
ros::Rate r(10);// this node will run at 10hz
while (ros::ok()) {
    ros::spinOnce();//each callback is called once to collect new data: laser
    update();//processing of data
    r.sleep();//we wait if the processing (ie, callback+update) has taken less than 0.1s (ie, 10 hz)
}
```

Our first node

- The method « update » will check if new message of the laser has arrived with *new_laser*;
- It will perform a loop over the beams to display the laser data published;

```
void update() {

    // we wait for new data of the laser
    if ( new_laser )
    {

        ROS_INFO("New data of laser received");

        for ( int loop=0 ; loop < nb_beams; loop++ )
            ROS_INFO( r[%i] = %f, theta[%i] (in degrees) = %f, x[%i] = %f, y[%i] = %f", loop, r[loop], loop, theta[loop]*180/M_PI, loop, current_scan[loop].x, loop, current_scan[loop].

        new_laser = false;

    }

}// update
```

ROS command to have a display in a terminal

Our first node

- Save *laser_text_display_node.cpp* and compile it with catkin_make in ~/catkin_ws
- Run it: rosrun *tutorial_ros laser_text_display_node* in ~/catkin_ws
- Do not forget to run a rosbag to play laser data

Outline

1. Our first node in ROS;
2. Our second node in ROS;
3. Compile and run nodes.

Our second node

- Edit *laser_graphical_display_node.cpp* in *~/catkin_ws/src/tutorial_ros*;

```
class laser_graphical_display {

private:
    ros::NodeHandle n;

    ros::Subscriber sub_scan;

    ros::Publisher pub_laser_graphical_display_marker;

    // to store, process and display laserdata
    int nb_beams;
    float range_min, range_max;
    float angle_min, angle_max, angle_inc;
    float r[1000], theta[1000];
    geometry_msgs::Point current_scan[1000];

    bool new_laser;//to check if new data of laser is available or not

    // GRAPHICAL DISPLAY
    int nb_pts;
    geometry_msgs::Point display[1000];
    std_msgs::ColorRGBA colors[1000];
```

I declare a node

This node will subscribe to one kind of messages

This node will publish one kind of messages

New variables to store and graphically display the laser data

Our second node

- Our second node will publish a message called « laser_graphical_display_marker »;
- The type of this message is a marker used by rviz to have a graphical display;

```
laser_graphical_display() {

    sub_scan = n.subscribe("scan", 1, &laser_graphical_display::scanCallback, this);

    pub_laser_graphical_display_marker = n.advertise<visualization_msgs::Marker>("laser_graphical_display_marker", 1); // Preparing a topic to publish our results.

    new_laser = false;
```

Our second node

- The method « update » will check if a new message of the laser has arrived with *new_laser*;
- It will perform a loop over the beams to display the laser data published and store the laser data with a blue color;

```cpp
void update() {

    // we wait for new data of the laser
    if ( new_laser )
    {

        new_laser = false;
        ROS_INFO("New data of laser received");

        nb_pts = 0;
        for ( int loop=0 ; loop < nb_beams; loop++ )
        {
            ROS_INFO("r[%i] = %f, theta[%i] (in degrees) = %f, x[%i] = %f, y[%i] = %f", loop, r[loop], loop, theta[loop]*180/M_PI, loop, current_scan[loop].x, loop, current_scan[loop].

            display[nb_pts] = current_scan[loop];

            colors[nb_pts].r = 0;
            colors[nb_pts].g = 0;
            colors[nb_pts].b = 1;
            colors[nb_pts].a = 1.0;

            nb_pts++;
        }
        populateMarkerTopic();
```

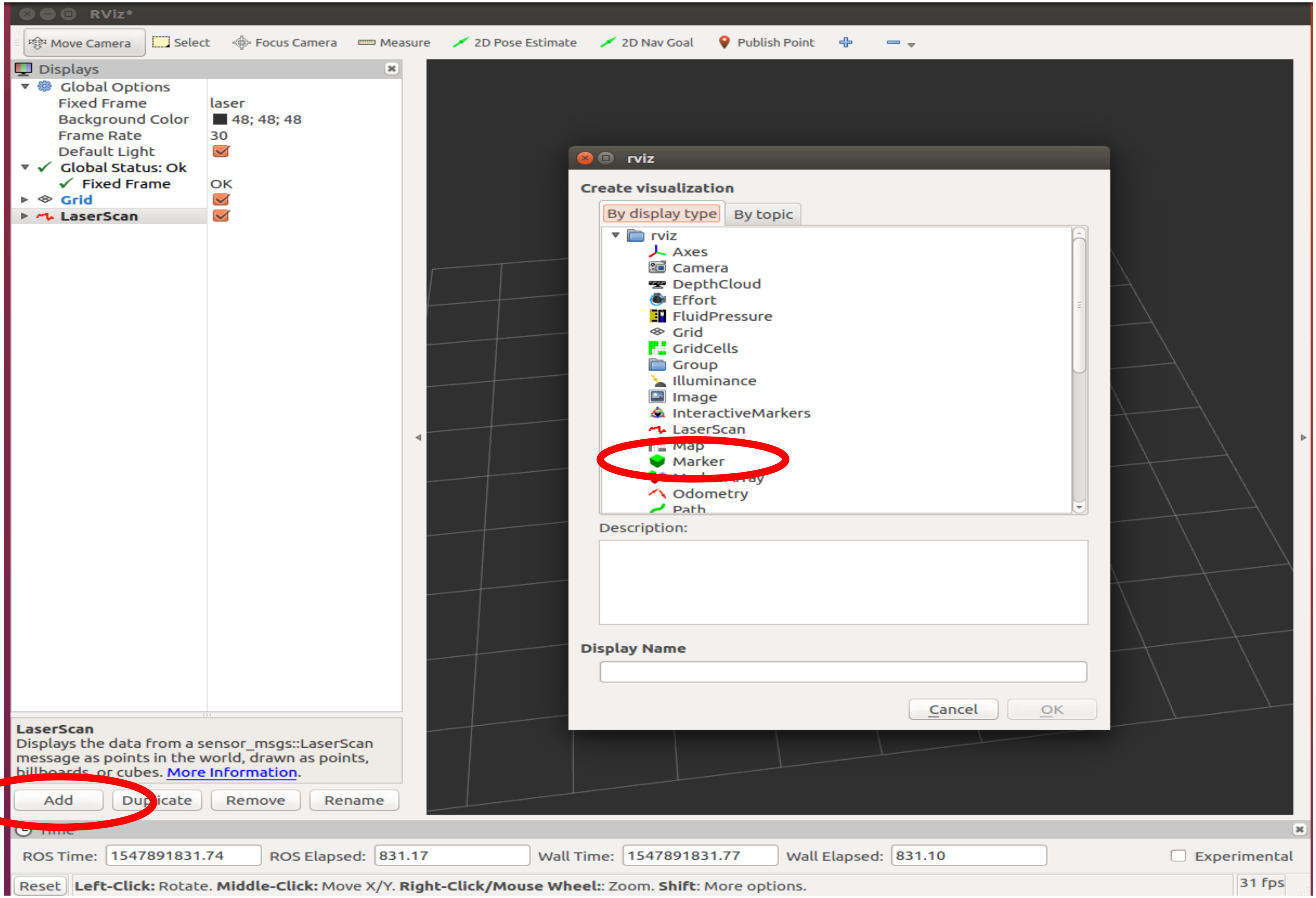
Store the laser data with a blue color

This method will build the marker message and publish it

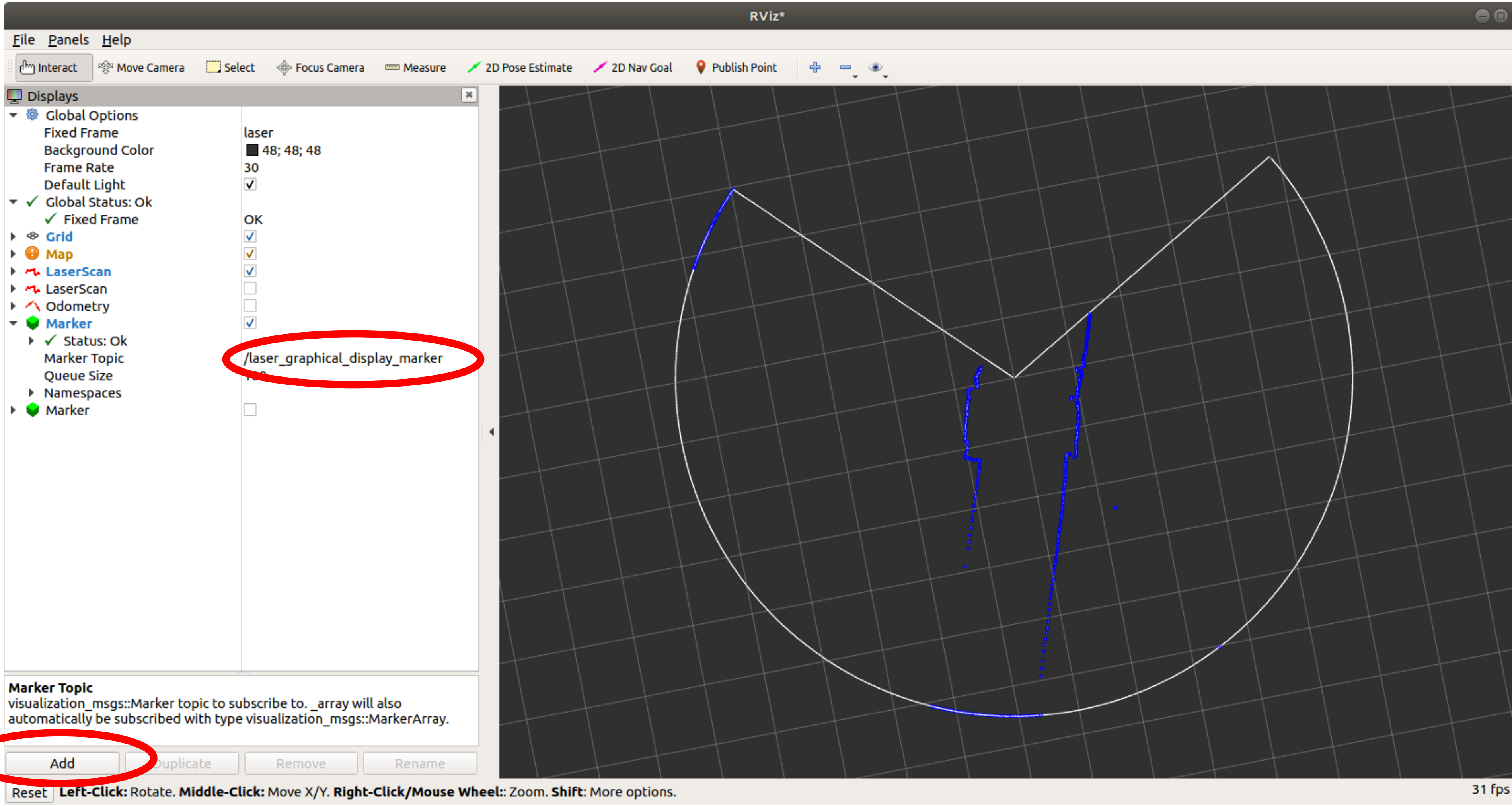
Our second node

- Save *laser_graphical_display_node.cpp* and compile it with catkin_make in ~/catkin_ws
- Run it: rosrun *tutorial_ros laser_graphical_display_node* in ~/catkin_ws
- Do not forget to run a rosbag to play laser data

Our second node



Our second node



Our second node

- Change the color of the laser data in green
- See the difference with rviz

Outline

1. Our first node in ROS;
2. Our second node in ROS;
3. Compile and run nodes.

Compile and run nodes

- A folder in ~/catkin_ws/src is called a package;
- A package contains some source files;
- These source files will be compiled to create nodes;
- To compile packages: run catkin_make in ~/catkin_ws

- For instance, in the package tutorial_ros, there are 3 source files that will generate 3 nodes

- To run a node: rosrun package_name node_name in ~/catkin_ws
- For instance: rosrun *tutorial_ros laser_text_display_node*
  - Run the node *laser_text_display_node* located in the package *tutorial_ros*