Introduction ROS - 3

Jaeseok Kim
The BioRobotics Institute - Service Robotics and Ambient Assisted Living Lab

Contents

- **Parameters**
- **roslaunch**
- **Sensor package**
- **rosserial**
- **TurtleBot 3 development environment**

- **Parameters**
- roslaunch
- Sensor package
- rosserial
- TurtleBot 3 development environment

Parameters

1) Create nodes using parameters

- In this section, we are going to modify the service_server.cpp to use a parameter to select an arithmetic operator, rather than merely adding a and b entered as service requests.
- Let's modify the service_server.cpp source in the following order:

```
$ roscd ros_tutorials_service/src      → Go to the src folder, which is the source code folder of the package
$ gedit service_server.cpp             → Modify source file contents
```

```
#include "ros/ros.h"                                  // ROS basic header file
#include "ros_tutorials_service/SrvTutorial.h"    // SrvTutorial.h" // SrvTutorial Service file header (auto-generated after build)

#define PLUS                1    // Add
#define MINUS               2    // Subtract
#define MULTIPLICATION      3    // Multiply
#define DIVISION            4    // Divide

int g_operator = PLUS;
```

Parameters

```
// If there is a service request, do the following
// service request is req, service response is res

bool calculation(ros_tutorials_service::SrvTutorial::Request &req,
ros_tutorials_service::SrvTutorial::Response &res)
{
// The values of a and b received when the service is requested are changed
according to the parameter value.
// Calculate and store in service response value
switch(g_operator)
{
      case PLUS:
      res.result = req.a + req.b; break;
      case MINUS:
      res.result = req.a - req.b; break;
      case MULTIPLICATION:
      res.result = req.a * req.b; break;
```

Parameters

```
        case DIVISION:
            if(req.b == 0){
                res.result = 0; break;
            }
            else{
                res.result = req.a / req.b; break;
            }
            default:
            res.result = req.a + req.b; break;
            }
        // Display the values of a and b used in the service request and the result value
        corresponding to the service
        response
        ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)req.b);
        ROS_INFO("sending back response: [%ld]", (long int)res.result);
        return true;
}
```

Parameters

```
int main(int argc, char **argv)              // Node main function
{
    ros::init(argc, argv, "service_server");    // Initialize the node name
    ros::NodeHandle nh;                         // Declare a node handle to
    communicate with the ROS system

    nh.setParam("calculation_method", PLUS);     // Parameter Initialization

    // service server declaration, using SrvTutorial service file from
    ros_tutorials_service package
    // Create the service server service_server. The service name is
    "ros_tutorial_srv"
    // Set up to execute a function called calculation when there is a service
    request.
    ros::ServiceServer ros_tutorial_service_server = nh.advertiseService("ros_tutorial_srv",
    calculation);

    ROS_INFO("ready srv server!");
```

Parameters

```
    ros::Rate r(10);
                                        // 10 hz
    while (1)
    {
        nh.getParam("calculation_method", g_operator);    // Change the
        operator to the value received from the parameter
        ros::spinOnce();                // Callback function processing  routine
        r.sleep();                      // sleep processing for routine iteration
    }

    return 0;
}
```

• Note the usage of setParam and getParam regarding parameters.

[Reference] Available as parameter Type

• Parameters can be set as integers, floats, boolean, string, dictionaries, list, and so on.

• For example, 1 is an integer, 1.0 is a float, "Internet of Things" is a string, true is a boolean, [1,2,3] is a list of integers, a: b, c: d is a dictionary.

Parameters

2) Build and run the node

```
$ cd ~/catkin_ws && catkin_make
```

```
$ rosrun ros_tutorials_service service_server
[INFO] [1495767130.149512649]: ready srv server!
```

3) View parameter list

• The "rosparam list" command shows a list of parameters currently used in the ROS network, where /calculation_method is the parameter we used.

```
$ rosparam list
/calculation_method
/rosdistro
/rosversion
/run_id
```

Parameters

4) Example of parameter usage

• Set the parameters according to the following command, and observe the change of service process when requesting the same service each time.
• In ROS, parameters can change the flow, setting, and processing of nodes from outside the node. It's a very useful feature so be familiar with it.

```
$ rosservice call /ros_tutorial_srv 10 5        → Input variables a and b of arithmetic operationresult: 15
                                  → The addition arithmetic sum, which is the default arithmetic operation

$ rosparam set /calculation_method 2        → subtract
$ rosservice call /ros_tutorial_srv 10 5
result: 5

$ rosparam set /calculation_method 3        → multiply
$ rosservice call /ros_tutorial_srv 10 5
result: 50

$ rosparam set /calculation_method 4        → divide
$ rosservice call /ros_tutorial_srv 10 5
result: 2
```

Source code

• We have seen how to communicate services between nodes by creating service servers and client nodes and running them. The relevant source can be found at the address of GitHub below.

• https://github.com/ROBOTIS-GIT/ros_tutorials/tree/master/ros_tutorials_service

• If you want to try it right away, you can clone the source code with the following command in the catkin_ws/src folder and build. Then run the service_server and service_client nodes.

```
$ cd ~/catkin_ws/src
$ git clone https://github.com/ROBOTIS-GIT/ros_tutorials.git
$ cd ~/catkin_ws
$ catkin_make
```

```
$ rosrun ros_tutorials_parameter service_server_with_parameter
```

```
$ rosrun ros_tutorials_parameter service_client_with_parameter 2 3
```

- Parameters
- **roslaunch**
- Sensor package
- rosserial
- TurtleBot 3 development environment

How to use roslaunch

- **rosrun** is a command to execute a node.
- **roslaunch** can run one or more defined nodes.

- In addition, roslaunch command allows you to specify options such as changing package parameters or node names, configuring node namespaces, setting ROS_ROOT and ROS_PACKAGE_PATH, and changing environment variables when running a node.

- roslaunch uses the file '* .launch' to set up an executable node, which is XML-based and provides tag-specific options.
- The execution command is "roslaunch [package name] [roslaunch file]".

How to use roslaunch

1) Use of roslaunch

• Rename the previously created topic_publisher and topic_subscriber nodes and run them. In order to understand the roslaunch, let's setup two separate message communication by running two public nodes and two subscriber Nodes.

• First, let's write a * .launch file. The file used for roslaunch has a file name of * .launch, and you need to create a folder called launch in the package folder and put the launch file in the folder. The folder can be created with the following command and create a new file called union.launch.

```
$ roscd ros_tutorials_topic
$ mkdir launch
$ cd launch
$ gedit union.launch
```

How to use roslaunch

```
<launch>
<node pkg="ros_tutorials_topic" type="topic_publisher" name="topic_publisher1"/>
<node pkg="ros_tutorials_topic" type="topic_subscriber"
name="topic_subscriber1"/>
<node pkg="ros_tutorials_topic" type="topic_publisher" name="topic_publisher2"/>
<node pkg="ros_tutorials_topic" type="topic_subscriber"
name="topic_subscriber2"/>
</launch>
```

• The <**launch**> tag describes required tags to run the node with roslaunch command. The <**node**> tags describe the node to be run by roslaunch. Options include pkg, type, and name.

- **pkg** Package name
- **type** The name of the node to actually execute (node name)
- **name** Set the name (executable name) to be appended when the node corresponding to the above type is executed, usually set to be the same as type, but you can set it to change the name when you need it.

How to use roslaunch

- Once you have created the roslaunch file, run union.launch as follows.

```
$ roslaunch ros_tutorials_topic union.launch --screen
```

- [Reference] How to display status when using roslaunch
  - When the roslaunch command runs several nodes, the output (info, error, etc.) of the running nodes is not displayed on the terminal screen and it makes debugging difficult. If you add the --screen option, the output of all nodes running on that terminal will be displayed on the terminal screen.

How to use roslaunch

- Let’s modify union.launch

```
$ roscd ros_tutorials_service/launch
$ gedit union.launch
```

```
<launch>
<group ns="ns1">
<node pkg="ros_tutorials_topic" type="topic_publisher" name="topic_publisher"/>
<node pkg="ros_tutorials_topic" type="topic_subscriber"
name="topic_subscriber"/>
</group>
<group ns="ns2">
<node pkg="ros_tutorials_topic" type="topic_publisher" name="topic_publisher"/>
<node pkg="ros_tutorials_topic" type="topic_subscriber"
name="topic_subscriber"/>
</group>
</launch>
```

How to use roslaunch

- **Execution result?**

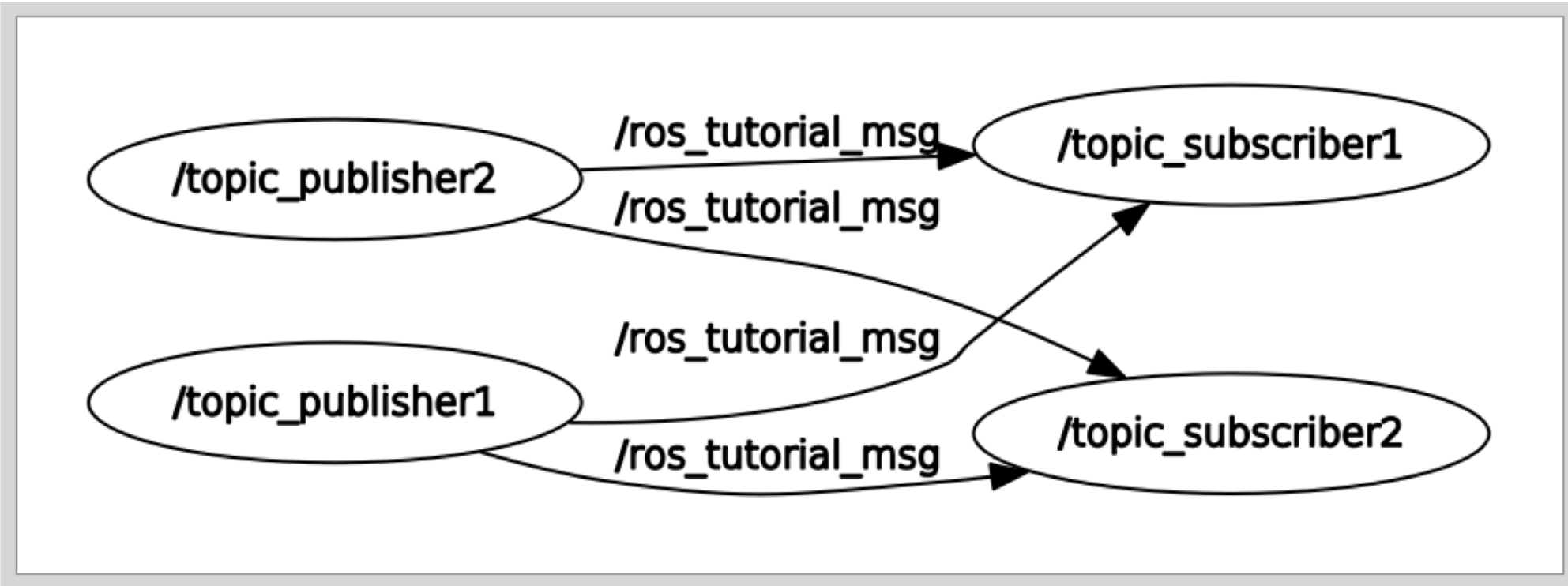
```
$ rosnode list
/topic_publisher1
/topic_publisher2
/topic_subscriber1
/topic_subscriber2
/rosout
```

- As a result, the topic_publisher node was renamed as topic_publisher1 and topic_publisher2, and two nodes are running.

- The topic_subscriber node has also been renamed as topic_subscriber1 and topic_subscriber2.

How to use roslaunch

• The problem is that, unlike the initial intention that "two separate messages are communicated by driving two publisher nodes and subscriber nodes," rqt_graph shows that they are subscribing to each other's messages.
• This is because we simply changed the name of the node to be executed, but did not change the name of the message to be used.
• Let's fix this problem with another roslaunch namespace tag.



How to use roslaunch

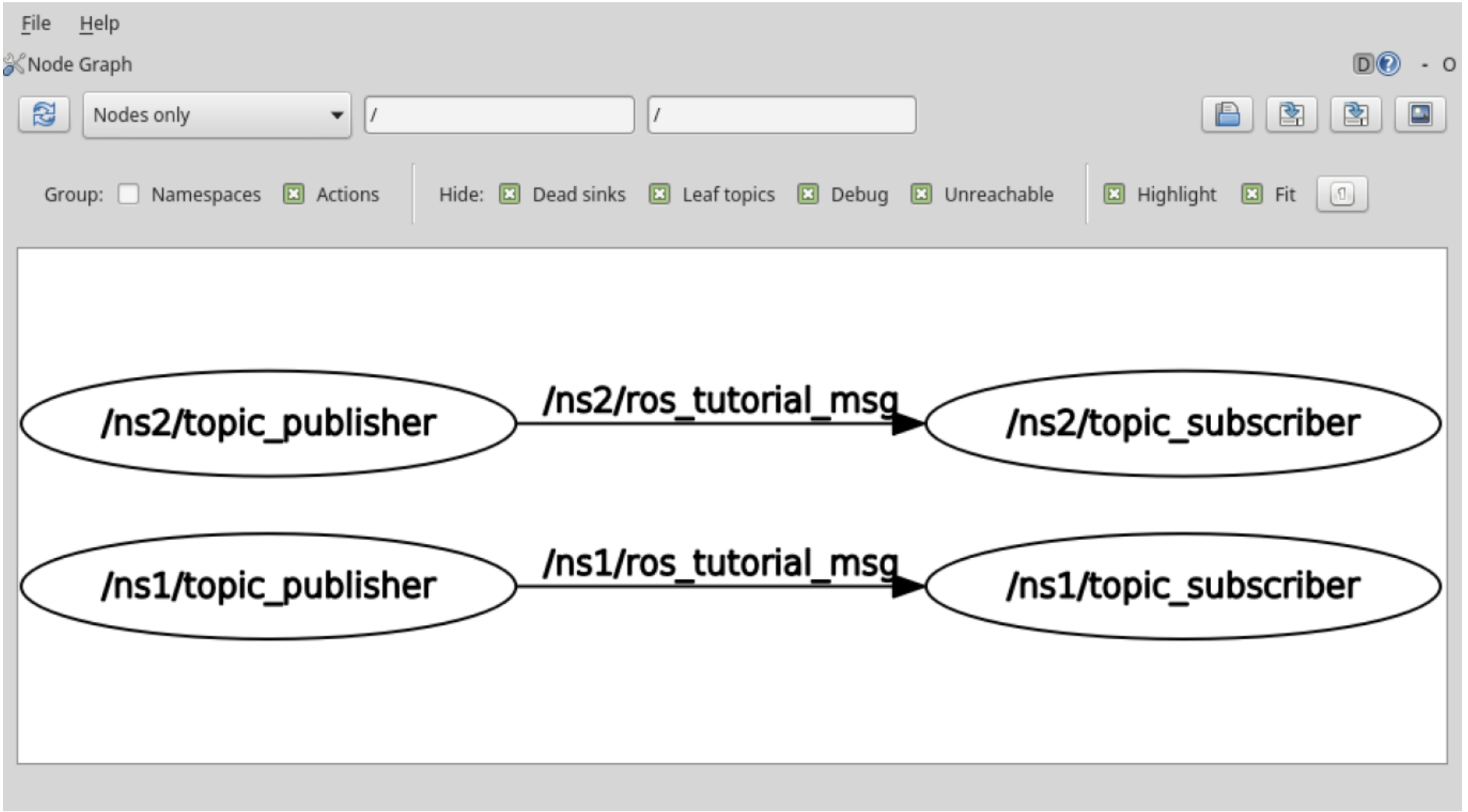
- Let’s modify union.launch

```
$ roscd ros_tutorials_service/launch
$ gedit union.launch
```

```
<launch>
<group ns="ns1">
<node pkg="ros_tutorials_topic" type="topic_publisher" name="topic_publisher"/>
<node pkg="ros_tutorials_topic" type="topic_subscriber"
name="topic_subscriber"/>
</group>
<group ns="ns2">
<node pkg="ros_tutorials_topic" type="topic_publisher" name="topic_publisher"/>
<node pkg="ros_tutorials_topic" type="topic_subscriber"
name="topic_subscriber"/>
</group>
</launch>
```

How to use roslaunch

- The appearance of executed nodes after change



How to use roslaunch

2) launch tag

<launch> Points to the beginning and end of the roslaunch syntax.
<node> This is a tag for node execution. You can change the package, node name, and execution name.
<machine> You can set the name, address, ros-root, and ros-package-path of the PC running the node.
<include> You can import another package or another launch belonging to the same package and run it as a launch file.
<remap> You can change the name of the ROS variable in use by the node name, topic name, and so on.
<env> Set environment variables such as path and IP. (Almost never used)
<param> Set the parameter name, type, value, etc.
<rosparam> Check and modify parameter information such as load, dump, and delete as the rosparam command.
<group> Set the parameter name, type, value, etc.
<test> Used to test the node. Similar to <node>, but with options available for testing.
<arg> You can define a variable in the launch file, so you can change the parameter when you run it like this:.

```
<launch>
<arg name="update_ period" default="10" />
<param name="timing" value="$(arg update_ period)"/>
</launch>
roslaunch my_package my_package.launch update_ period:=30
```

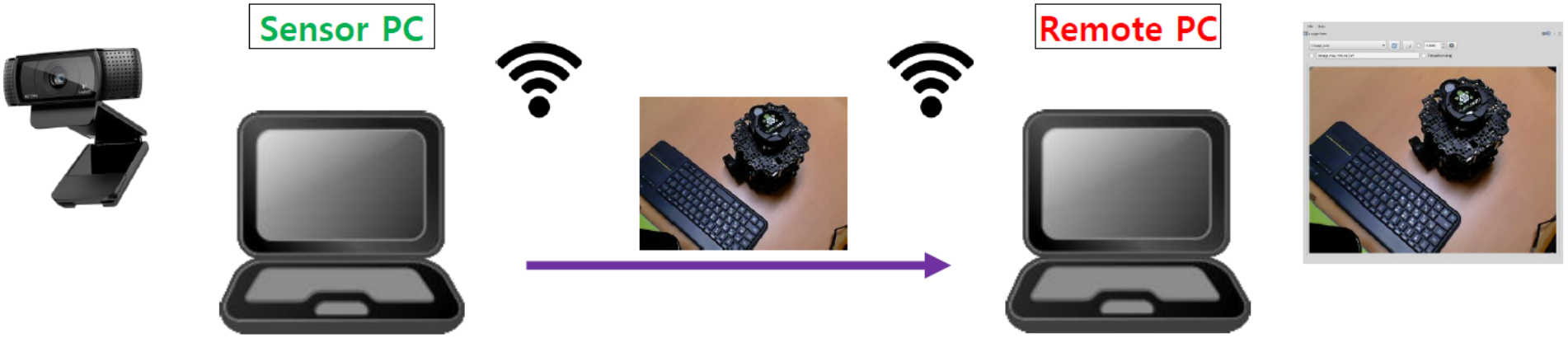
- Parameters
- roslaunch
- **Sensor package**
- rosserial
- TurtleBot 3 development environment

Sensor Package Practice #1 (USB Camera)

```
$ sudo apt-get install ros-kinetic-uvc-camera

$ rosrun uvc_camera uvc_camera_node
$ rosrun uvc_camera uvc_camera_node _device:=/dev/video? (Option)

$ rosrun image_view image_view image:=/image_raw
$ rqt_image_view image:=/image_raw
$ rviz
```

* Change the display options of RViz
  1) Change fixed frame
  Global Options > **Fixed Frame = camera**
  2) Add image display
  Click ‘Add’ in the bottom left corner of Rviz, then select **Image**
  (Add > by display > Rviz > Image)
  3) Change topic value
  Change the value of ‘Image > Image Topic’ to "**/image_raw**"

Sensor Package Practice #2 (Transfer images remotely)



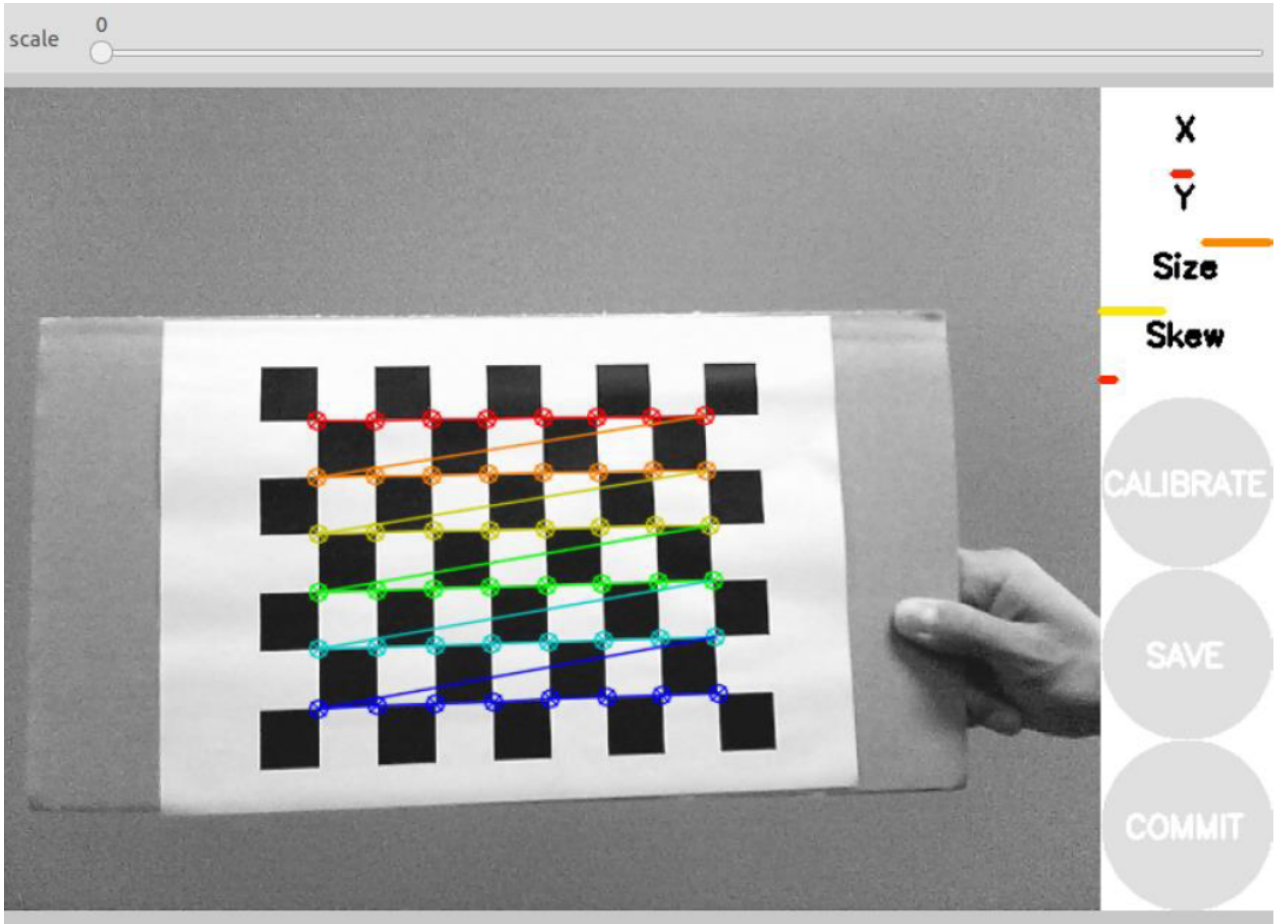
ROS_MASTER_URI = http://IP_OF_REMOTE_PC:11311
ROS_HOSTNAME = IP_OF_SENSOR_PC

ROS_MASTER_URI = http://IP_OF_REMOTE_PC:11311
ROS_HOSTNAME = IP_OF_REMOTE_PC
* Example of running ROS Master on a remote PC

- Modify '~/.bashrc' for each PC (ROS_MASTER_URI and ROS_HOSTNAME)
- Run '**roscore**' & '**rqt_image_view image:=/image_raw**' on the remote PC
- Run '**rosrun uvc_camera**', '**uvc_camera_node**' on the sensor PC

Sensor Package Practice #3 (Camera Calibration)

$ sudo apt-get install ros-kinetic-camera-calibration
$ **rosrun uvc_camera uvc_camera_node**
$ **rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.024 image:=/image_raw camera:=/camera**



Sensor Package Practice #4 (Depth Camera)

```
$ sudo apt-get install ros-kinetic-openni2-camera ros-kinetic-openni2-launch
$ tar -xvf Sensor-Bin-Linux-x64-v5.1.0.41.tar.bz2
$ cd Sensor-Bin-Linux-x64-v5.1.0.41/
$ sudo sh install.sh
$ roslaunch openni2_launch openni2.launch
```

* Change the display options of Rviz

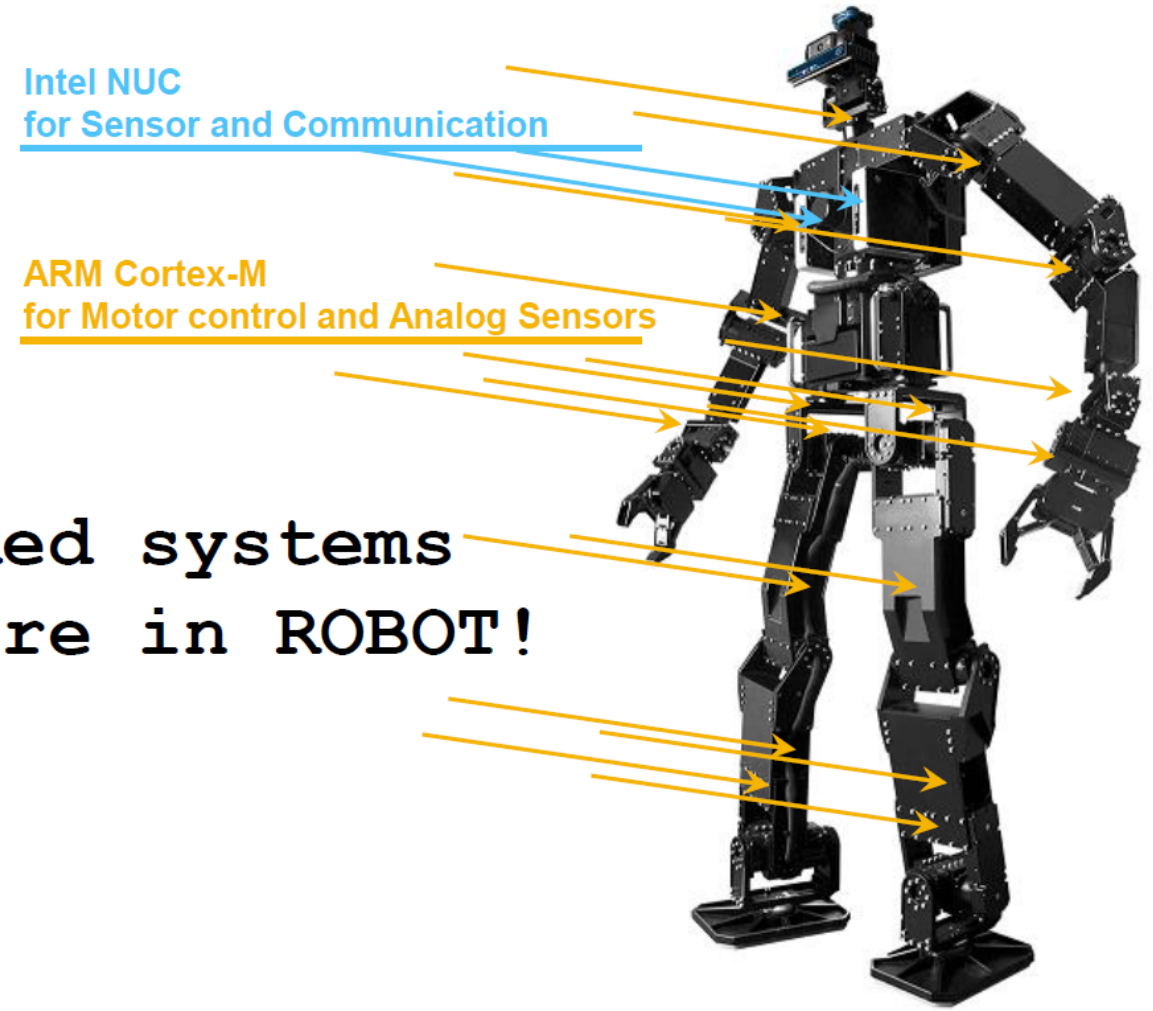
1) Change fixed frame
Change ‘Global Options > Fixed Frame’ to "camera_depth_frame"

2) Add & configure PointCloud2
Click ‘Add’ at the bottom left of rviz, then select PointCloud2

3) Change topic name & detail settings

- Parameters
- roslaunch
- Sensor package
- **Rosserial**
- TurtleBot 3 development environment

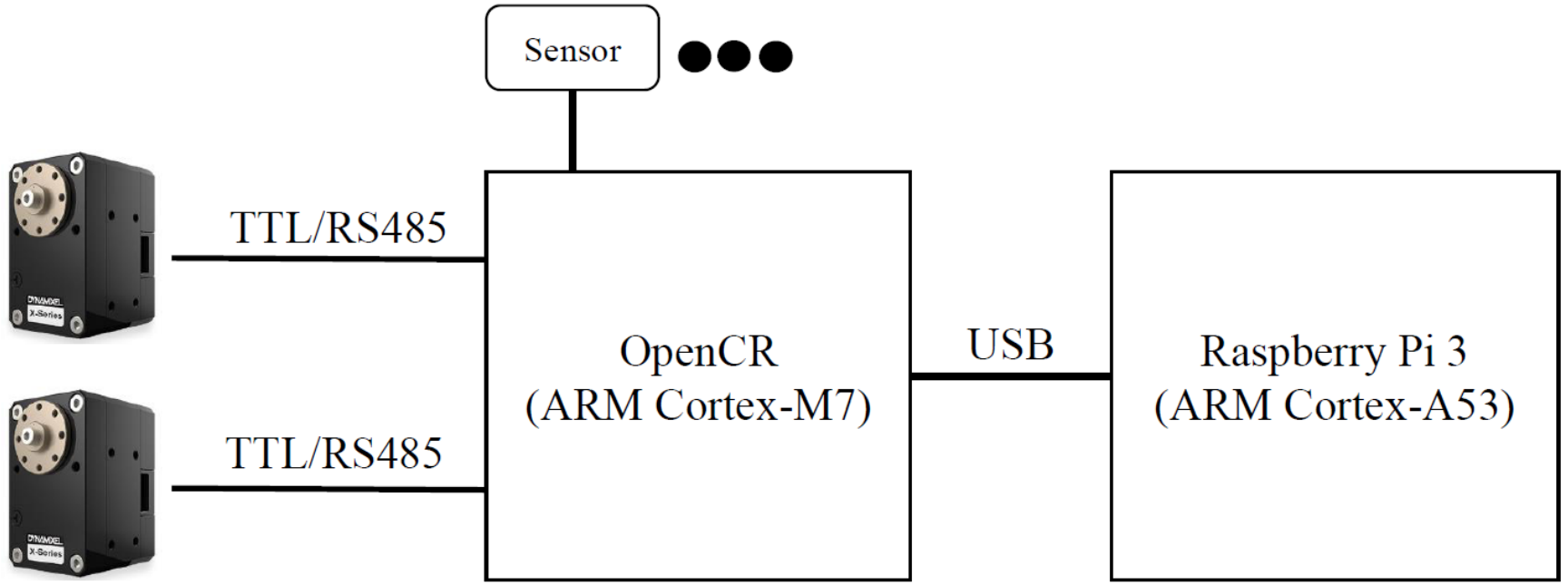
The proportion of embedded systems in robots



Small embedded systems
are everywhere in ROBOT!

Embedded systems in ROS

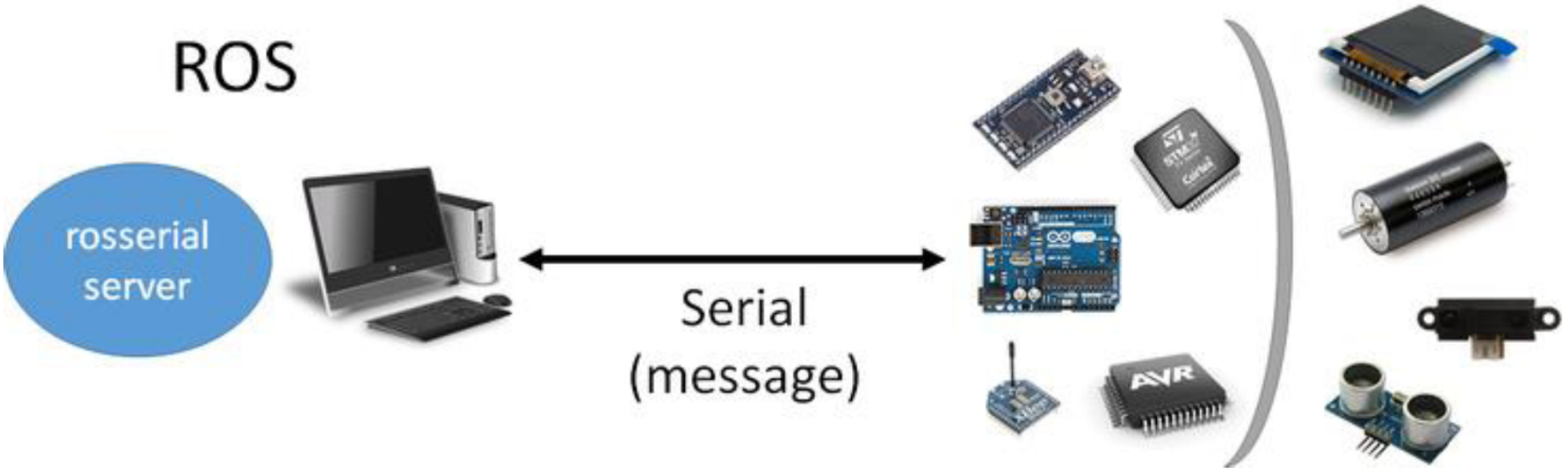
- Unlike PC, ROS can not be installed in embedded system
- For securing real-time factor and hardware control, connection between the embedded system and the ROS installed PC is required.
- ROS provides a package called ‘rosserial’ for this function!



[Configuration of PC and embedded system in TurtleBot3]

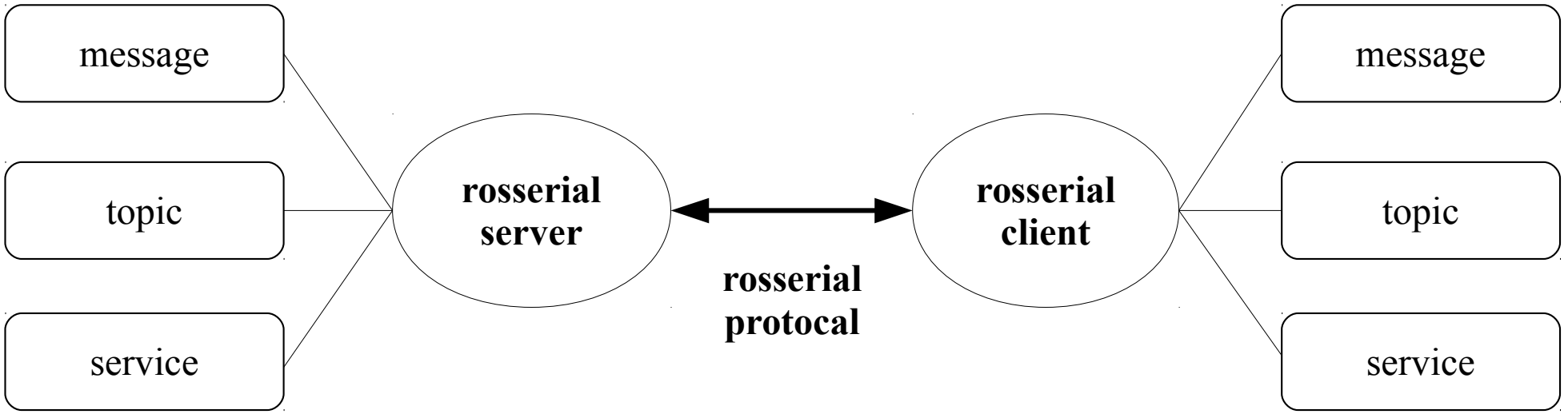
‘rosserial’

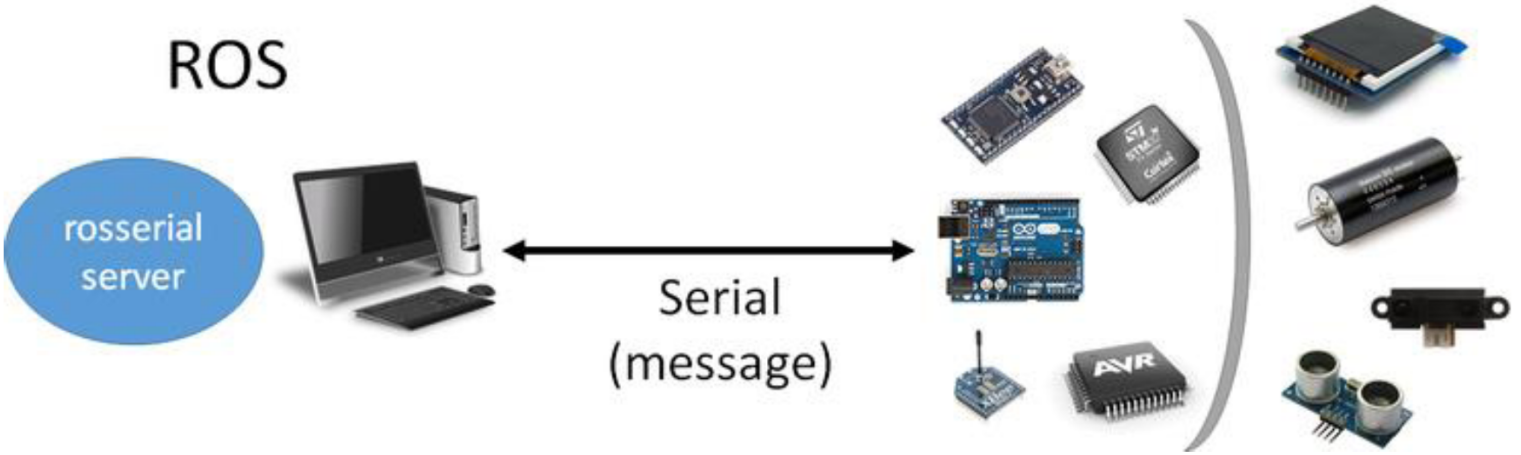
• ROS package acting as an intermediary for message communication between PC & Controller
  • Example) Controller -> Serial(rosserial protocol) -> PC(Retransmission with ROS messages)
  • Example) Controller <- Serial(rosserial protocol) <- PC(Change ROS messages to serial)



http://wiki.ros.org/rosserial

‘rosserial’ server & client





‘rosserial’ server & client

```
$ sudo apt-get install ros-kinetic-rosserial ros-kinetic-rosserial-server ros-kinetic-rosserial-arduino
```

- **rosserial server**
  - **rosserial_python**: Python language based rosserial server, very popular
  - **rosserial_server**: C++ language based rosserial server, Some functions are limited
  - **rosserial**_java: Java language based rosserial server, Used with android SDK

- **rosserial client**
  - **rosserial_arduino**: Support Arduino & Leonardo, OpenCR uses it with few modification
  - **rosserial_embeddedlinux**: Linux Library for Embedded System
  - **rosserial_windows**: Support Windows operating system, Windows application and communication support
  - **rosserial_mbed**: Support ARM’s mbed
  - **rosserial_tivac**: Support TI’s Launchpad

‘rosserial’ Protocol (http://wiki.ros.org/rosserial/Overview/Protocol)

Sync Flag	Header to know the start position of the packet. It is always ‘0xFF’
Sync Flag / Protocol version	A protocol version, ROS Groovy is ‘0xFF’, & ‘ROS Hydro’, ‘Indigo’, ‘Jade’ and ‘Kinetic’ are ‘0xFE’
Message Length (N)	Data length of message, 2 Byte = Low Byte + High Byte, Low Byte are transmitted first, followed by ‘High Byte’
Checksum over message length	Checksum for validating message length headers Checksum = 255 - ((Message Length Low Byte + Message Length High Byte) %256)
Topic ID	It is an ID for identifying the type of message. 2 Byte = Low Byte + High Byte ID_PUBLISHER=0, ID_SUBSCRIBER=1, ID_SERVICE_SERVER=2, ID_SERVICE_CLIENT=4, ID_PARAMETER_REQUEST=6, ID_LOG=7, ID_TIME=10, ID_TX_STOP=11
Serialized Message Data	It is the data to transmit the ROS message in serial form EX) IMU, TF, GPIO
Checksum over Topic ID and Message Data	Topic ID and checksum to validate message data Checksum = 255 - ((Topic ID Low Byte + Topic ID High Byte + data byte values) % 256)

Limitations of ‘rosserial’

- **Memory**
  - The number of publishers, subscribers, and transmit/receive buffer size must be defined in advance
- **Float64**
  - Microcontroller does not support 64-bit real numbers, so it is converted to 32-bit
- **Strings**
  - Instead of storing string data in a string message, only pointer values of externally
    defined string data are stored in the message
- **Arrays**
  - Used with specified array size because of memory constraint
- **Communication Speed**
  - In case of a UART with the speed of 115200 bps, response time becomes slower as the number of messages increases

- Parameters
- roslaunch
- Sensor package
- Rosserial
- **TurtleBot 3 development environment**

Preparation of TurtleBot3 Simulation Development Environment

- Official TurtleBot3 wiki reference
  - http://turtlebot3.robotis.com
- Basic package installation (for the use of 3D simulator Gazebo)

```
$ sudo apt-get install ros-kinetic-joy ros-kinetic-teleop-twist-joy ros-kinetic-teleop-twist-keyboard ros-kinetic-laser-proc ros-kinetic-rgbd-launch ros-kinetic-depthimage-to-laserscan ros-kinetic-rosserial-arduino ros-kinetic-rosserial-python ros-kinetic-rosserial-server ros-kinetic-rosserial-client ros-kinetic-rosserial-msgs ros-kinetic-amcl ros-kinetic-map-server ros-kinetic-move-base ros-kinetic-urdf ros-kinetic-xacro ros-kinetic-compressed-image-transport ros-kinetic-rqt-image-view ros-kinetic-gmapping ros-kinetic-navigation
```

```
$ cd ~/catkin_ws/src/
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3.git
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
$ cd ~/catkin_ws && catkin_make
```

TurtleBot3 Remote control

• roscore operation [Remote PC]

```
$ roscore
```

• turtlebot3_robot.launch :Run the launch file [TurtleBot]

```
$ roslaunch turtlebot3_bringup turtlebot3_robot.launch --screen
```

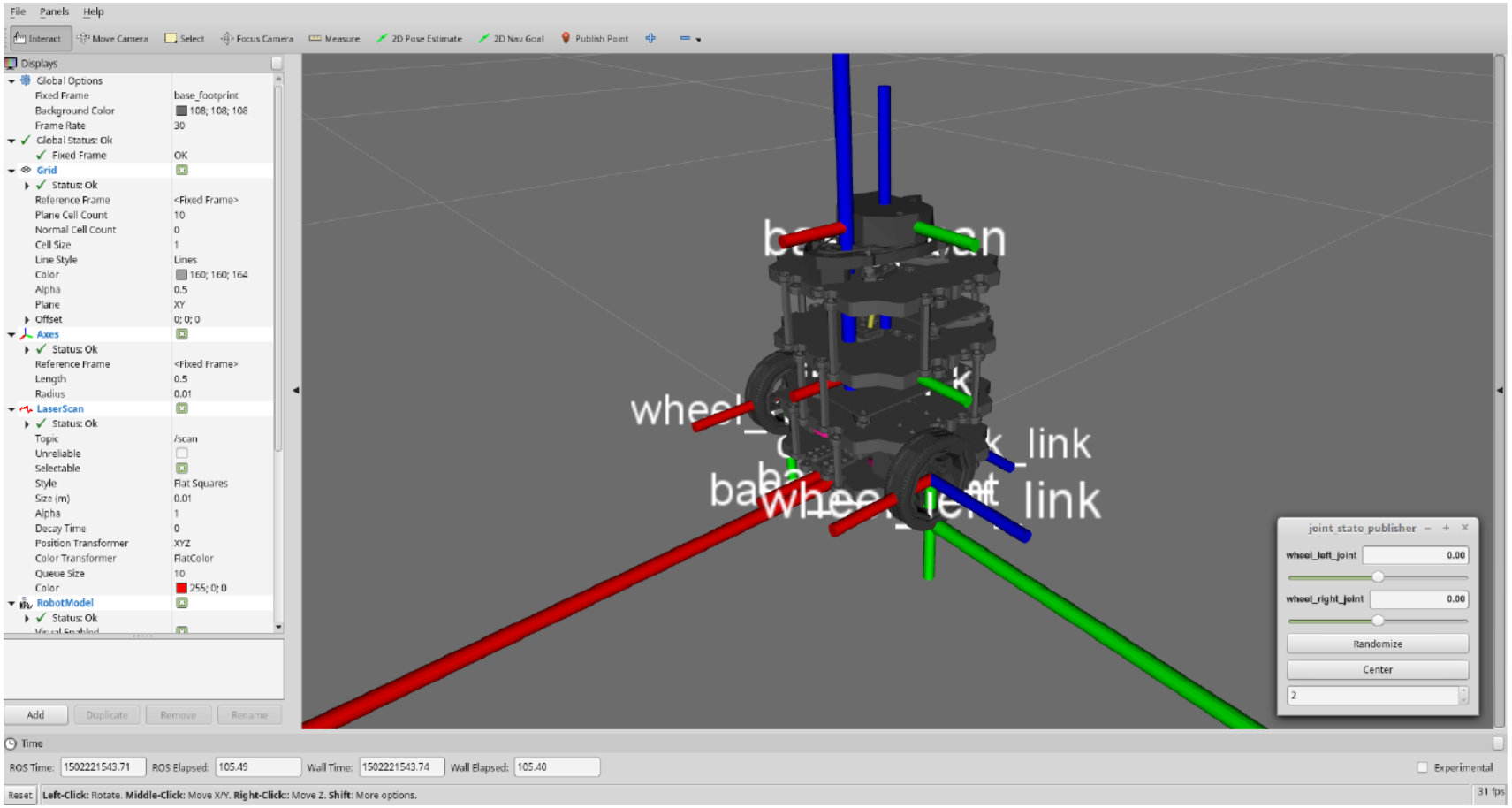
• turtlebot3_teleop_key.launch: Run the launch file [Remote PC]

```
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch --screen
```

TurtleBot3 Visualization

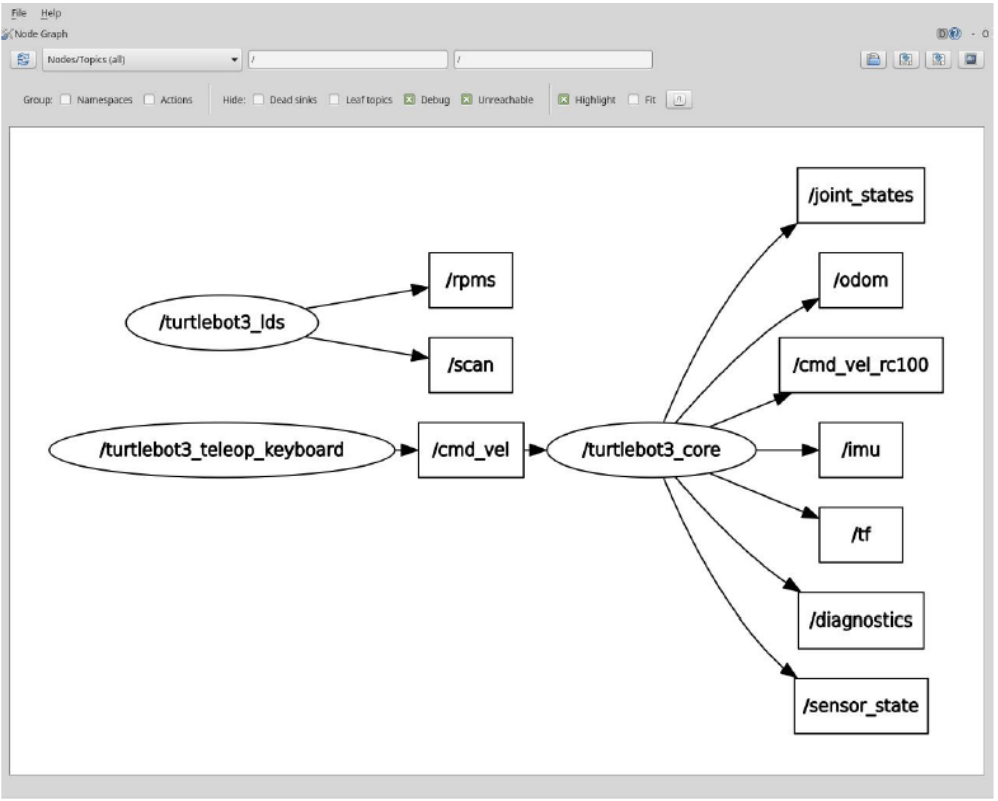
- Run RViz [Remote PC]]

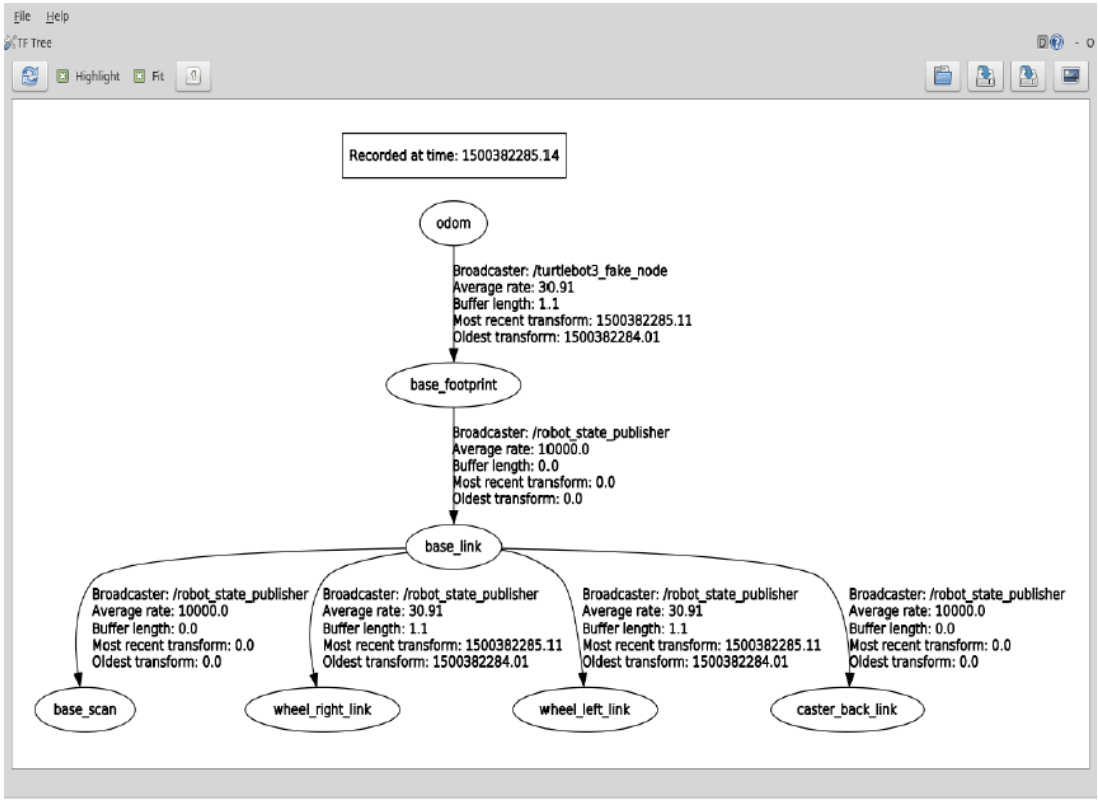
```
$ export TURTLEBOT3_MODEL=burger (or waffle or waffle_pi)
$ roslaunch turtlebot3_bringup turtlebot3_model.launch
```



TurtleBot3 Topic and TF

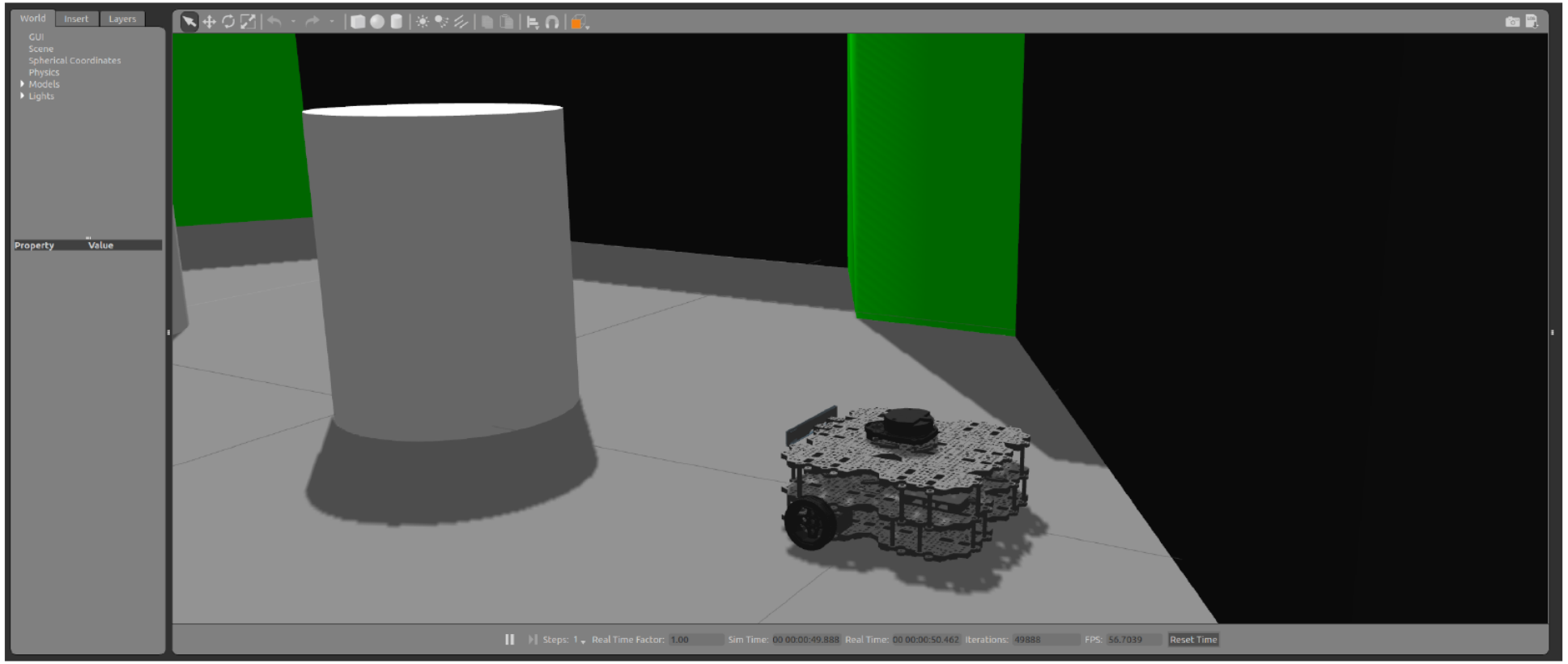
```
$ rqt_graph
$ rqt                [Plugins > Visualization > TF Tree]
```





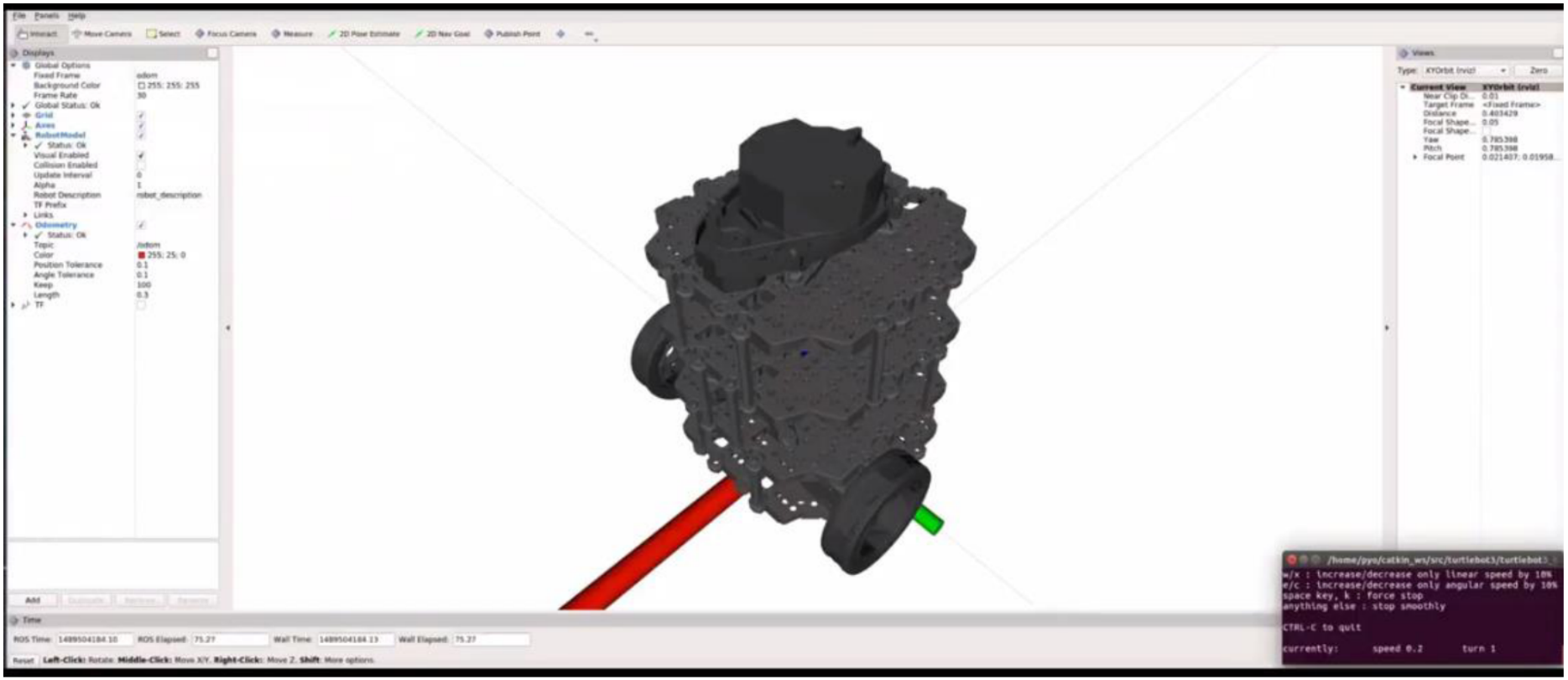
TurtleBot3 + simulation

- Two methods for simulation
  - Using RViz, a 3D visualization tool of ROS
  - Using 3D robot simulator Gazebo
  - http://turtlebot3.robotis.com/en/latest/simulation.html



Simulation with RViz as a viewer

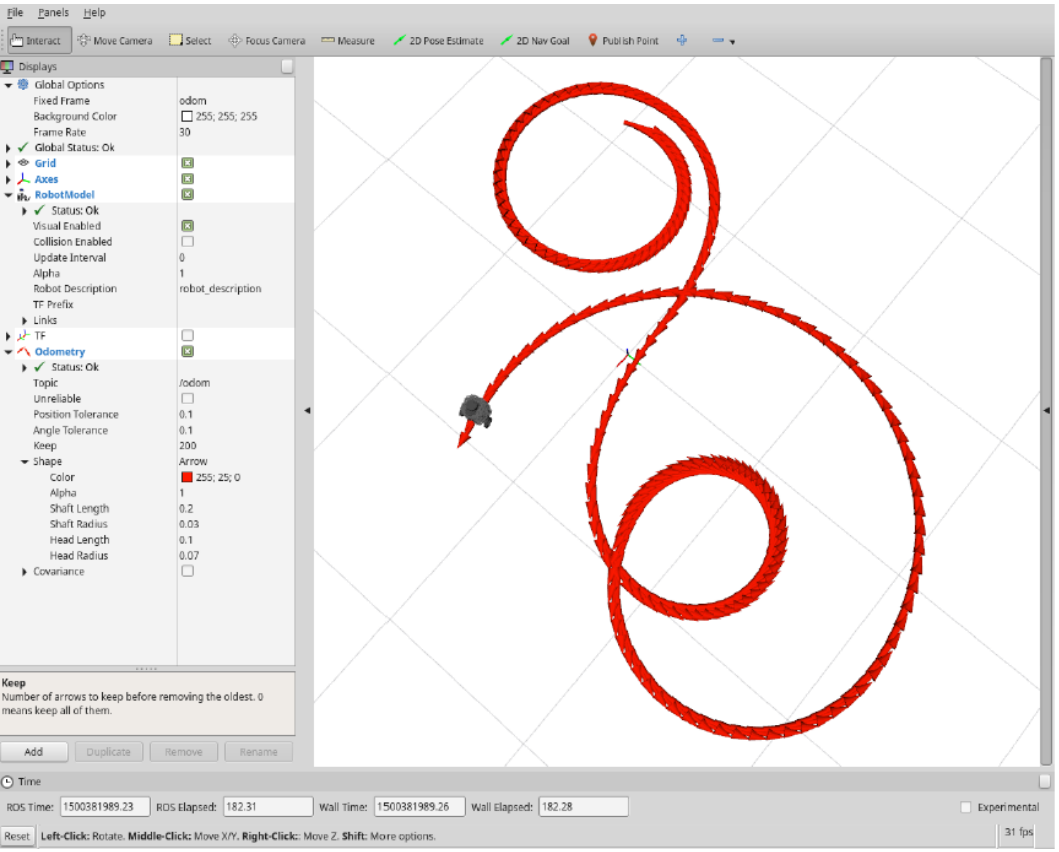
```
$ export TURTLEBOT3_MODEL=burger
$ roslaunch turtlebot3_fake turtlebot3_fake.launch

$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

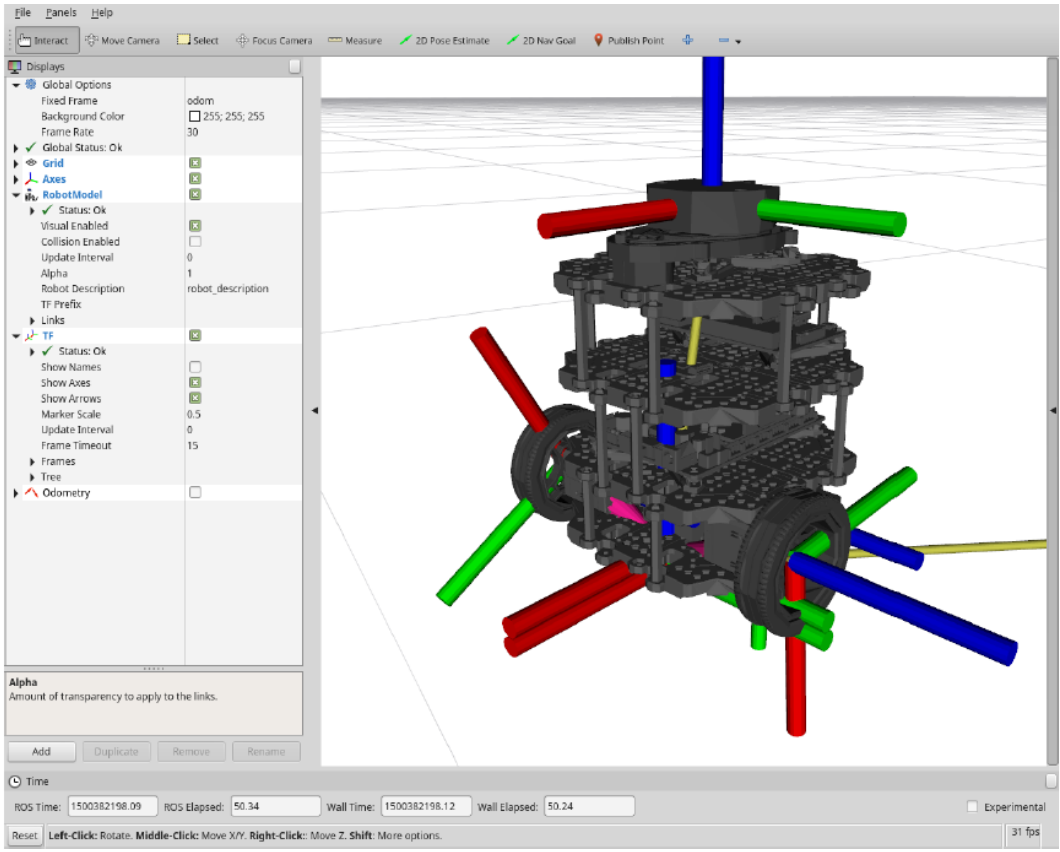


https://youtu.be/iHXZSLBJHMg

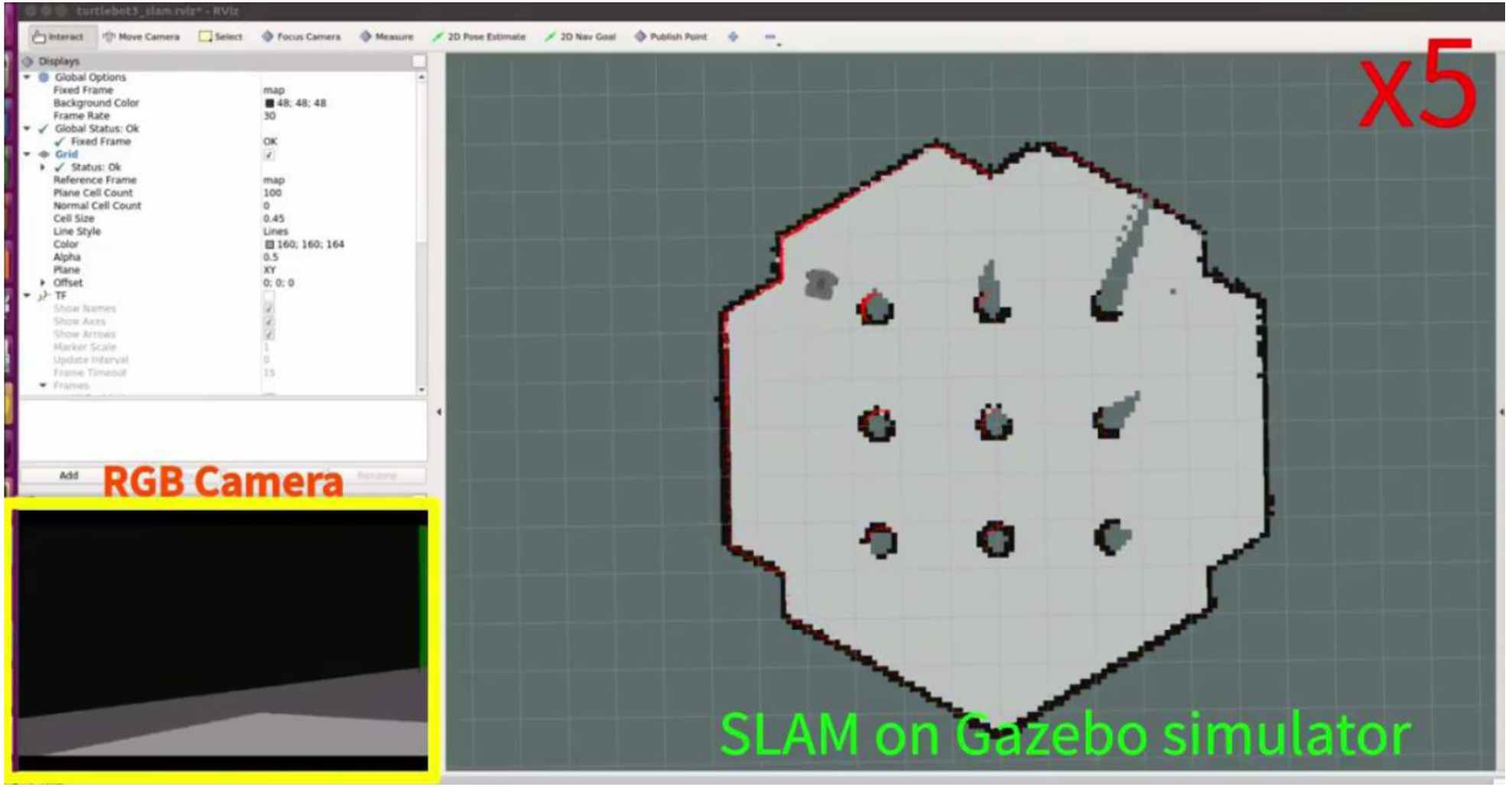
Simulation with RViz as a viewer

- Let's move the robot and check Odometry and tf!





Simulation with Gazebo / TurtleBot3 in Gazebo



https://youtu.be/xXM5r_SVkWM

Reference

- **Book**

  - ROS_Robot_Programming_EN
- http://wiki.ros.org/Documentation
- http://wiki.ros.org/ROS/Tutorials

Question Time!