

---

# ROS visualization, tf, sensor\_msgs, bag

---

# Overview

---

- ROS bag 소개, 실습
- Rviz 소개, 실습
- ROS Sensor\_msgs 소개, 실습
- ROS tf 소개, 실습



# ROS bag

---

- Rosbag

- Topic 을 기록하는 ROS의 data저장 방식
  - Sensor의 데이터 뿐만 아니라 Publish되는 모든 Topic 기록 가능.

1. ROSMASTER 실행 `$ roscore`
2. Turtlesim node실행 `$ rosrun turtlesim turtlesim_node`
3. Turtlesim Teleop node실행 `$ rosrun turtlesim turtle_teleop_key`
4. Rosbag record실행  
(모든 topic record) `$ rosbag record -a`
5. Teleop node에서 방향키로 조작.
6. Rosbag record 종료 (CTRL + C)



# ROS bag

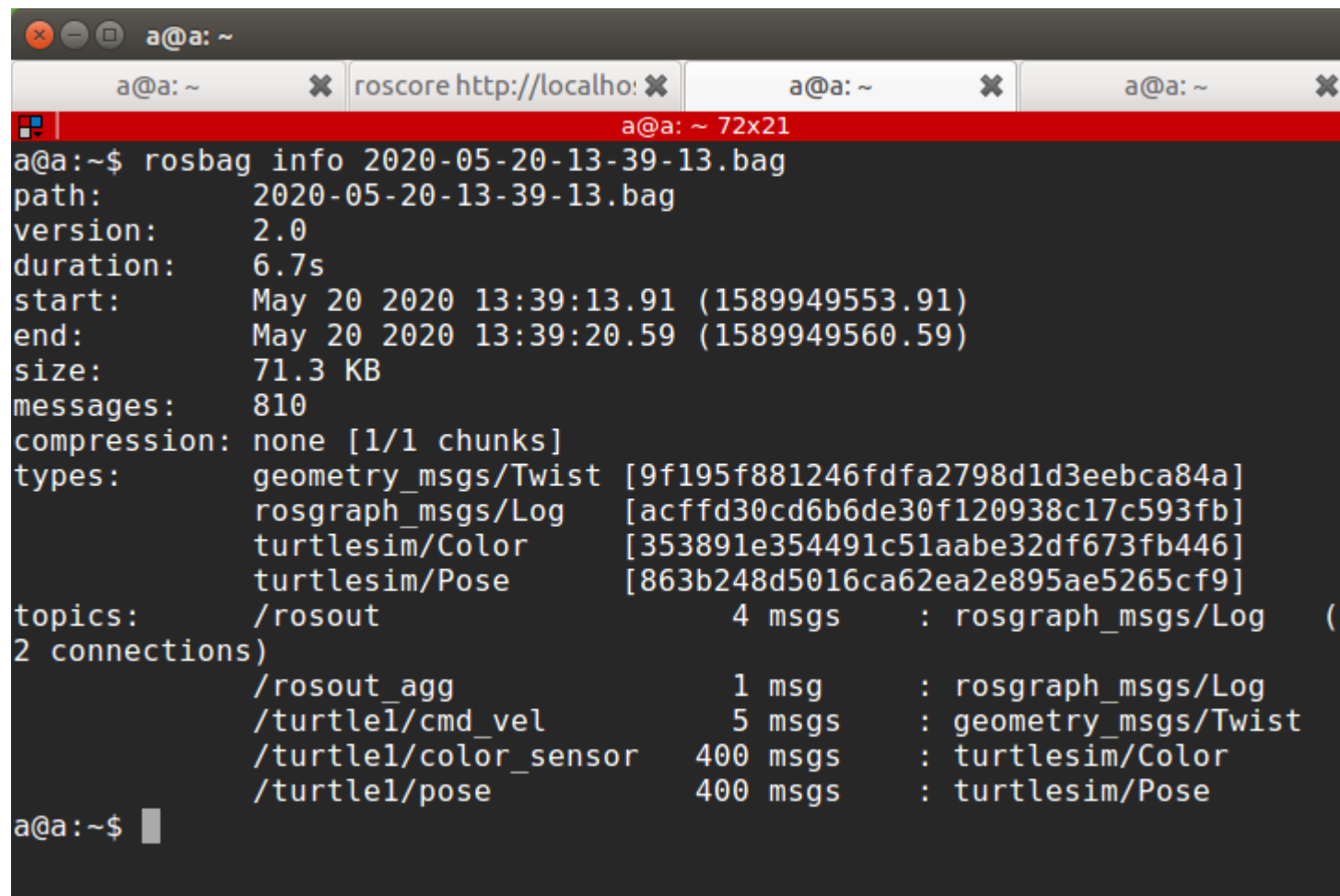
- Rosbag

- Bag file 확인

```
$ rosbag info <bagfilename>
```

- Bagfile의 정보를 확인가능.

- Record된 topic을 topics에서 확인.



```
a@a: ~  
a@a: ~ 72x21  
a@a:~$ rosbag info 2020-05-20-13-39-13.bag  
path:          2020-05-20-13-39-13.bag  
version:       2.0  
duration:      6.7s  
start:         May 20 2020 13:39:13.91 (1589949553.91)  
end:           May 20 2020 13:39:20.59 (1589949560.59)  
size:          71.3 KB  
messages:      810  
compression:   none [1/1 chunks]  
types:         geometry_msgs/Twist [9f195f881246fdfa2798d1d3eebca84a]  
               rosgraph_msgs/Log   [acffd30cd6b6de30f120938c17c593fb]  
               turtlesim/Color     [353891e354491c51aabe32df673fb446]  
               turtlesim/Pose       [863b248d5016ca62ea2e895ae5265cf9]  
topics:        /rosout              4 msgs      : rosgraph_msgs/Log   (  
2 connections)  
               /rosout_agg           1 msg       : rosgraph_msgs/Log  
               /turtle1/cmd_vel      5 msgs      : geometry_msgs/Twist  
               /turtle1/color_sensor 400 msgs     : turtlesim/Color  
               /turtle1/pose         400 msgs     : turtlesim/Pose  
a@a:~$
```

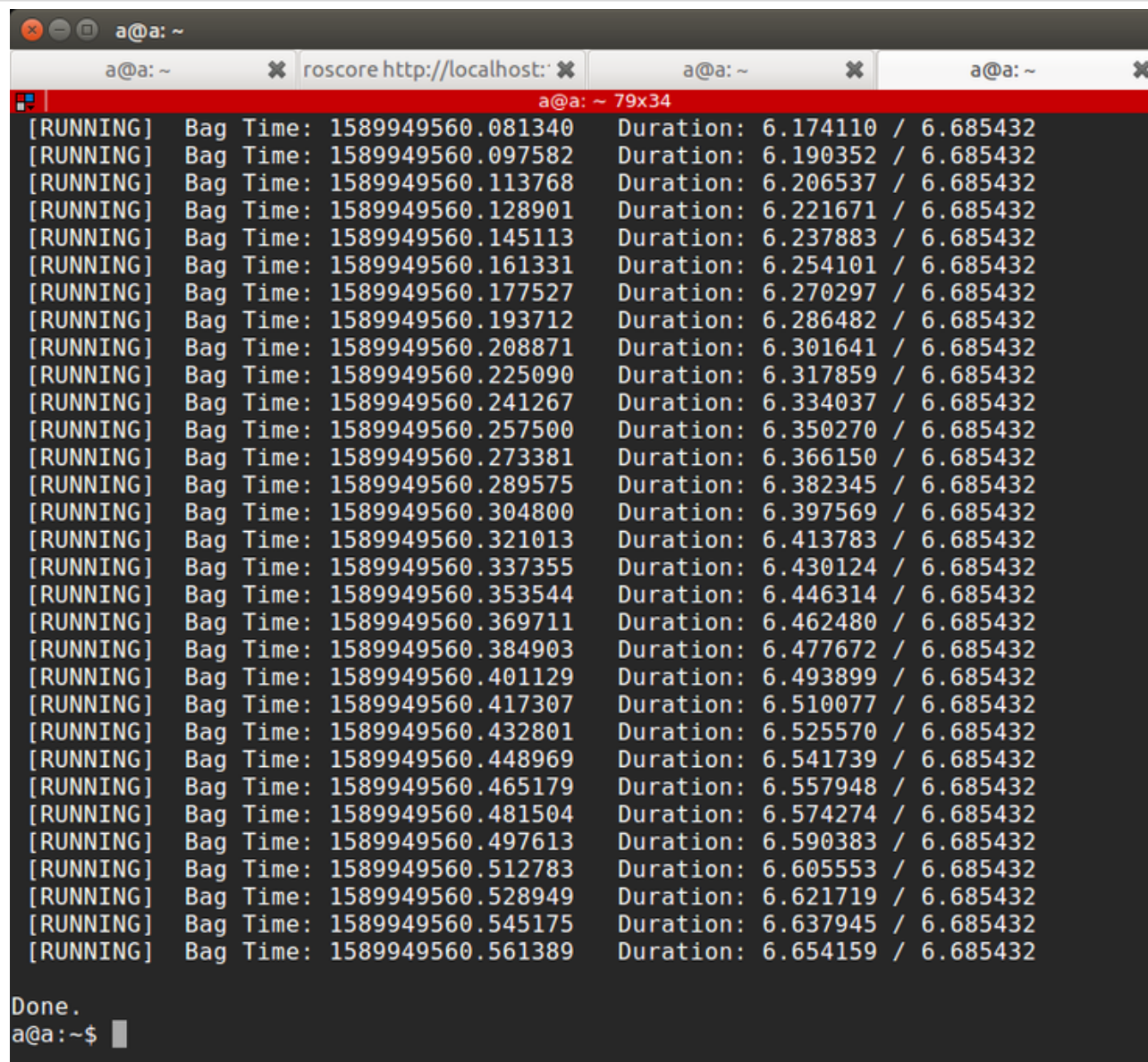
# ROS bag

- Rosbag play

- Bag file 재생

\$ rosbag play <bagfilename>

- 오른쪽 그림과 같이 재생됨을 확인.

A terminal window titled 'a@a: ~' with multiple tabs. The active tab shows the output of the 'rosbag play' command. The output consists of a list of 30 lines, each starting with '[RUNNING]' followed by 'Bag Time' and 'Duration' values. The values are in scientific notation. At the bottom of the terminal, it says 'Done.' and 'a@a: ~\$' with a cursor.

```
a@a: ~
a@a: ~
roscore http://localhost:~
a@a: ~
a@a: ~
a@a: ~ 79x34
[RUNNING] Bag Time: 1589949560.081340 Duration: 6.174110 / 6.685432
[RUNNING] Bag Time: 1589949560.097582 Duration: 6.190352 / 6.685432
[RUNNING] Bag Time: 1589949560.113768 Duration: 6.206537 / 6.685432
[RUNNING] Bag Time: 1589949560.128901 Duration: 6.221671 / 6.685432
[RUNNING] Bag Time: 1589949560.145113 Duration: 6.237883 / 6.685432
[RUNNING] Bag Time: 1589949560.161331 Duration: 6.254101 / 6.685432
[RUNNING] Bag Time: 1589949560.177527 Duration: 6.270297 / 6.685432
[RUNNING] Bag Time: 1589949560.193712 Duration: 6.286482 / 6.685432
[RUNNING] Bag Time: 1589949560.208871 Duration: 6.301641 / 6.685432
[RUNNING] Bag Time: 1589949560.225090 Duration: 6.317859 / 6.685432
[RUNNING] Bag Time: 1589949560.241267 Duration: 6.334037 / 6.685432
[RUNNING] Bag Time: 1589949560.257500 Duration: 6.350270 / 6.685432
[RUNNING] Bag Time: 1589949560.273381 Duration: 6.366150 / 6.685432
[RUNNING] Bag Time: 1589949560.289575 Duration: 6.382345 / 6.685432
[RUNNING] Bag Time: 1589949560.304800 Duration: 6.397569 / 6.685432
[RUNNING] Bag Time: 1589949560.321013 Duration: 6.413783 / 6.685432
[RUNNING] Bag Time: 1589949560.337355 Duration: 6.430124 / 6.685432
[RUNNING] Bag Time: 1589949560.353544 Duration: 6.446314 / 6.685432
[RUNNING] Bag Time: 1589949560.369711 Duration: 6.462480 / 6.685432
[RUNNING] Bag Time: 1589949560.384903 Duration: 6.477672 / 6.685432
[RUNNING] Bag Time: 1589949560.401129 Duration: 6.493899 / 6.685432
[RUNNING] Bag Time: 1589949560.417307 Duration: 6.510077 / 6.685432
[RUNNING] Bag Time: 1589949560.432801 Duration: 6.525570 / 6.685432
[RUNNING] Bag Time: 1589949560.448969 Duration: 6.541739 / 6.685432
[RUNNING] Bag Time: 1589949560.465179 Duration: 6.557948 / 6.685432
[RUNNING] Bag Time: 1589949560.481504 Duration: 6.574274 / 6.685432
[RUNNING] Bag Time: 1589949560.497613 Duration: 6.590383 / 6.685432
[RUNNING] Bag Time: 1589949560.512783 Duration: 6.605553 / 6.685432
[RUNNING] Bag Time: 1589949560.528949 Duration: 6.621719 / 6.685432
[RUNNING] Bag Time: 1589949560.545175 Duration: 6.637945 / 6.685432
[RUNNING] Bag Time: 1589949560.561389 Duration: 6.654159 / 6.685432
Done.
a@a: ~$
```

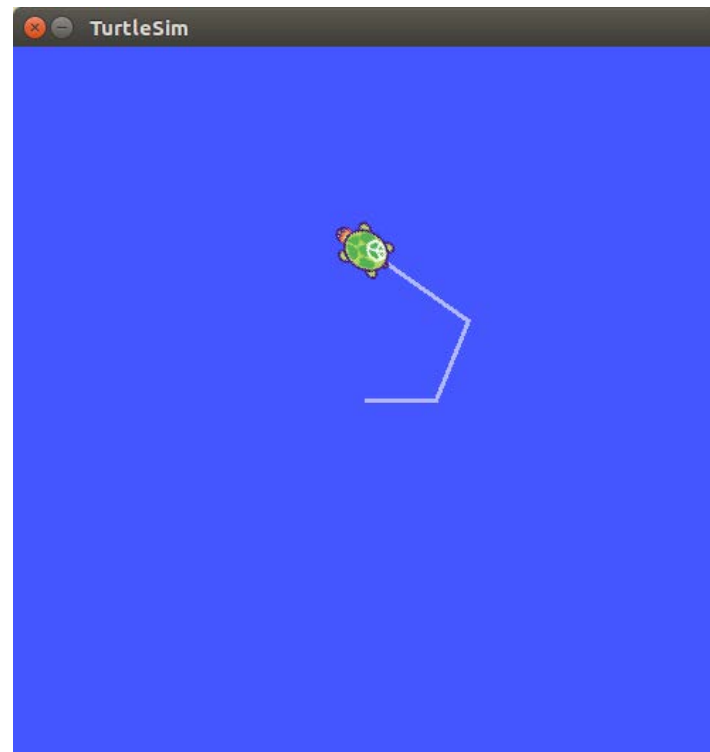
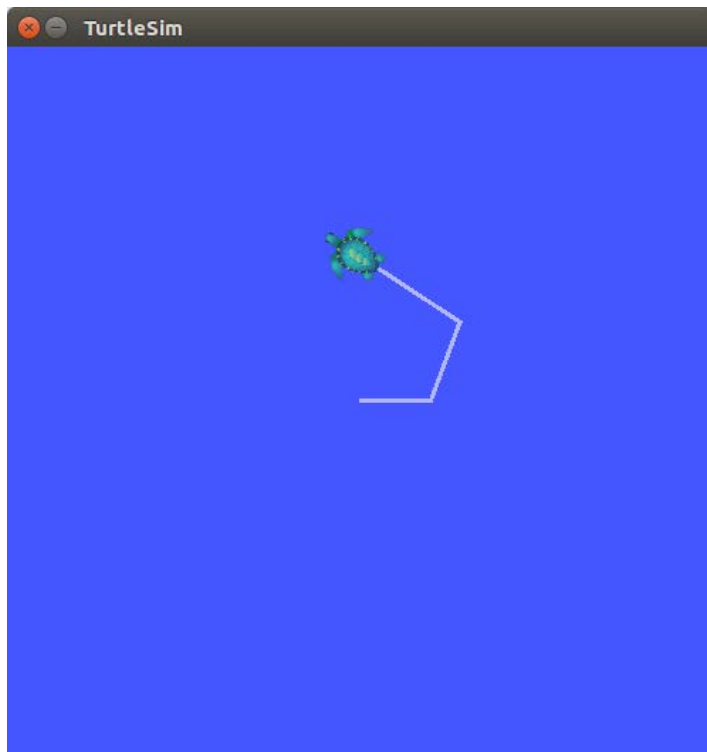
# ROS bag

---

- Rosbag play

- Turtlesim node에서 record된 topic확인
- Turtlesim node를 재시작.→ 동일한 조건
- Rosbag play, 동일한 동작 확인.

→ 동일한 동작.



# ROS bag

---

- Rosbag command line

- 특정한 topic만 record
- 재생시 clock topic(ros time)도 같이 재생
- 반복 재생 (loop)
- 특정구간 재생
- x배속 재생
- 다음과 같이 여러 옵션을 주어 사용가능

```
$ rosbag record /topic1 /topic2 ...
```

```
$ rosbag play bagname.bag --clock
```

```
$ rosbag play bagname.bag -l
```

```
$ rosbag play bagname.bag -s startTime  
-u duration
```

```
$ rosbag play bagname.bag -r x
```

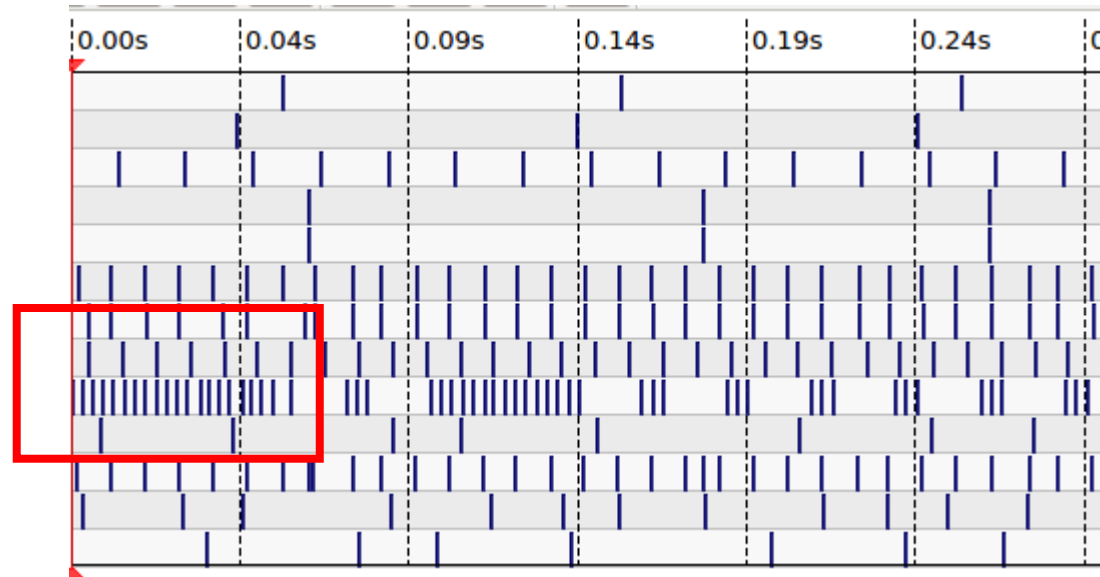
```
$ rosbag play bagname.bag --clock -r 0.2 -l -s 2.4 -u 5
```

---

# ROS bag

- Rosbag 한계점

- Topic이 많은 경우 정확하게 기록되기 어려움  
(Data 손실, timestamp가 동일하지 않음,  
buffer stuck으로 인한 delay...)
- Ros가 아닌 다른환경에서 사용하기 번거로움.  
→ 다른환경에서는 h5와 같은 형식 이용.
- 일정한 frequency를 갖는 topic을 record한  
결과를 확인했을때 오른쪽과 같은 불균일 구  
간 발생.

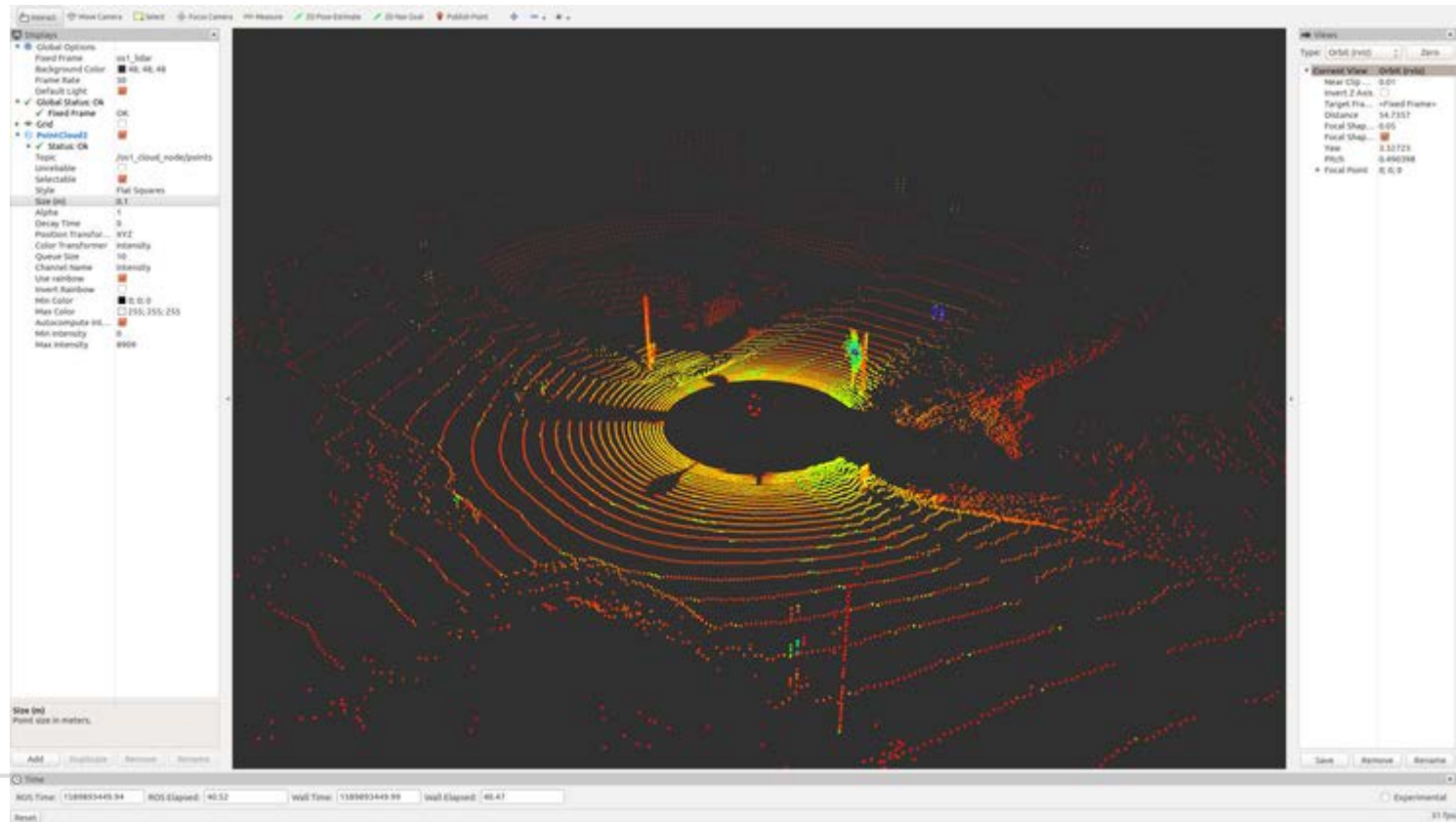


Rqt\_bag으로 topic timestamp를 확인한 결과.



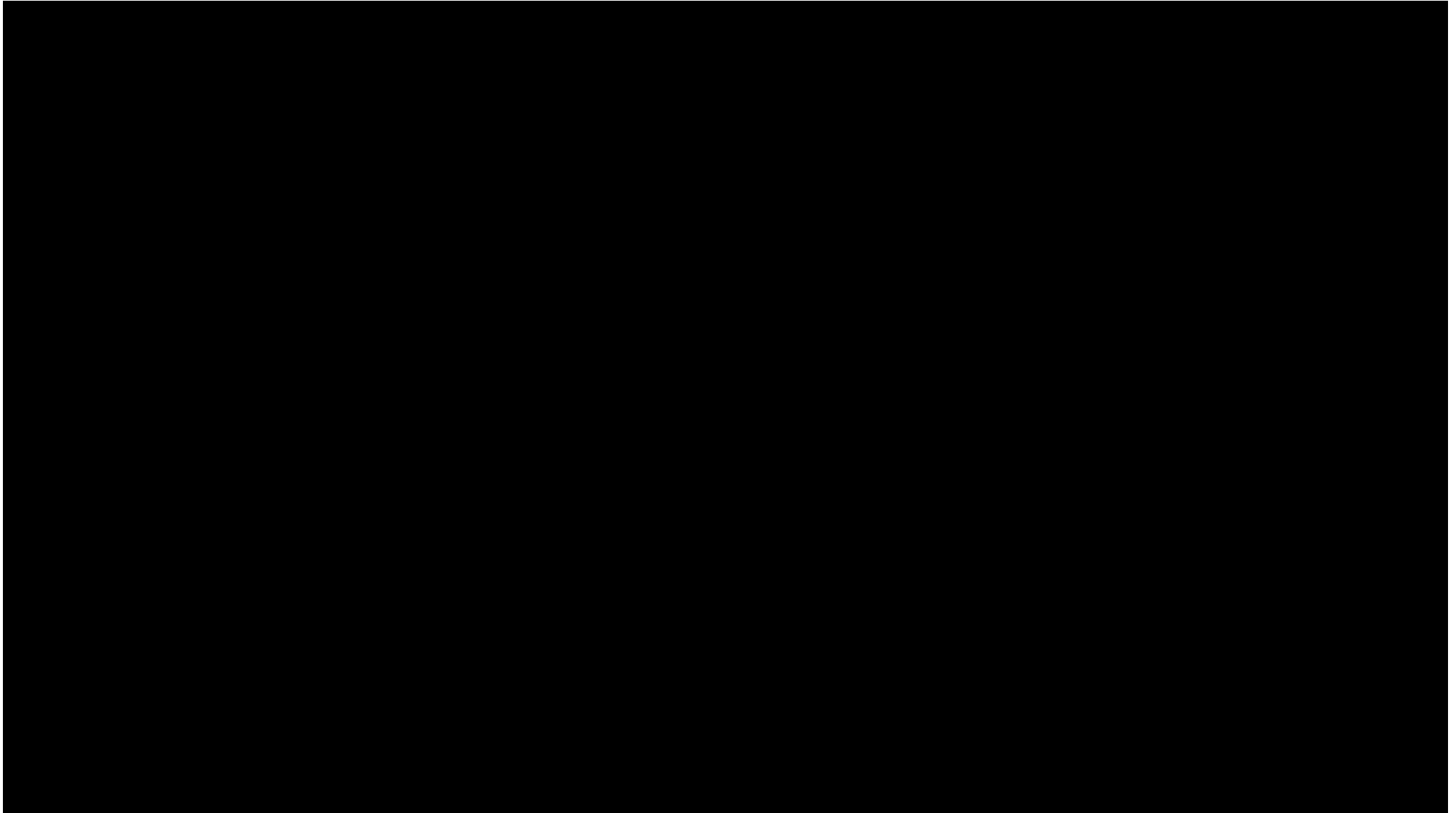
# Rviz

- Rviz
  - ROS의 Visualization Tool
  - 3D 시각화(Point cloud, Robot model...)
  - 간단한 point, navigation goal등의 publish기능



# Rviz

---

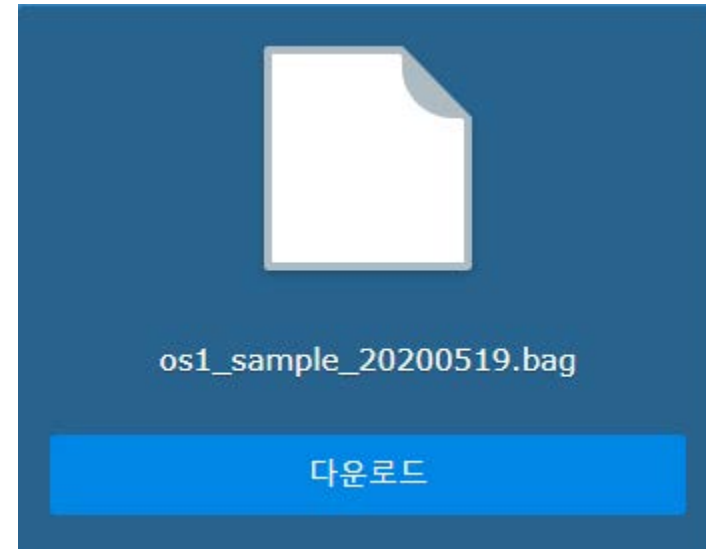


- Rviz 3D Pointcloud visualize
  - 제공되는 Ouster os1-64 LiDAR 센서의 데이터 시각화 예제

<http://irl-nas2.synology.me:5000/sharing/UkEGBDld5>



Ouster os1-64 LiDAR



# Rviz

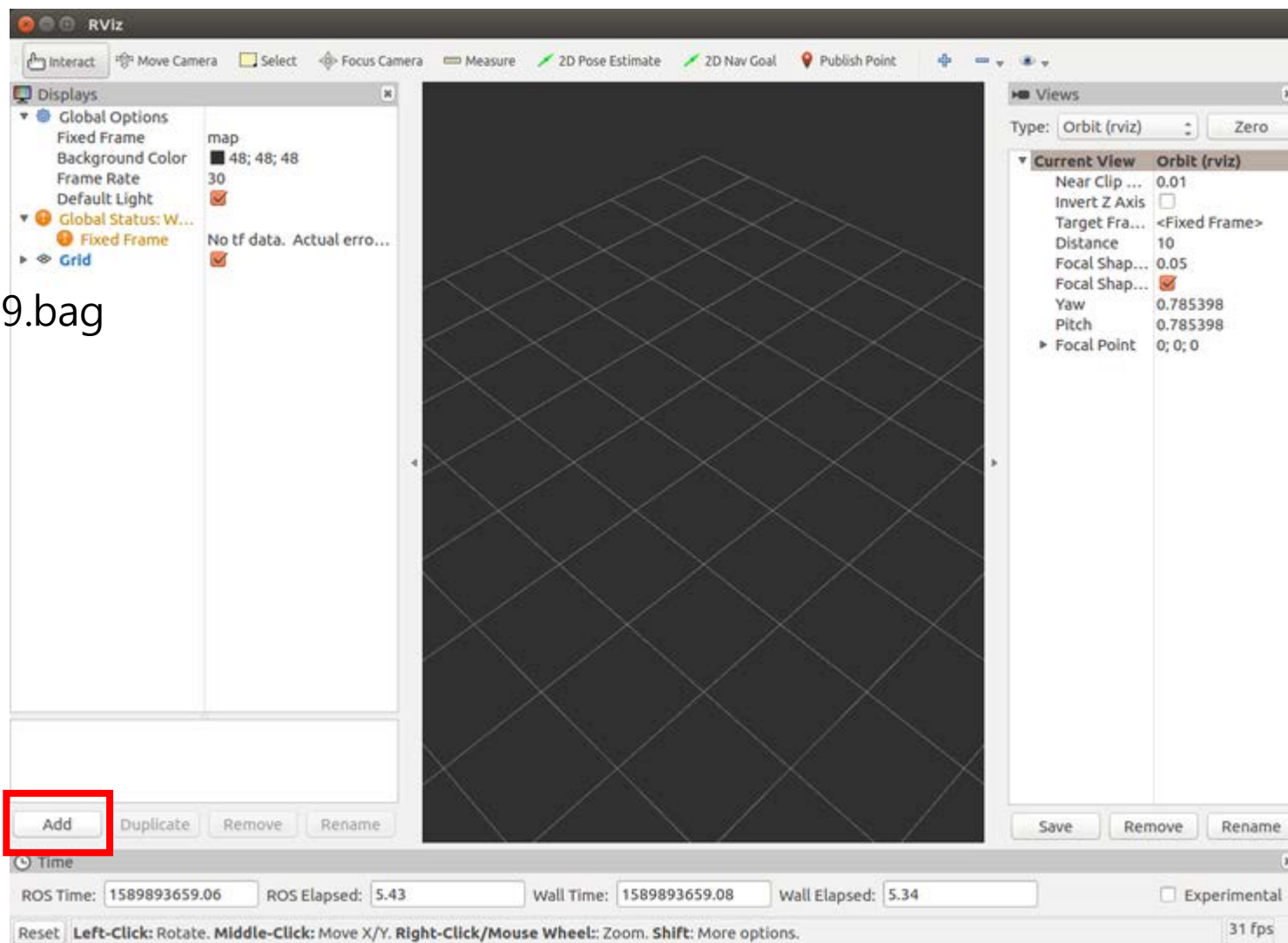
- Rviz

- Rviz 실행      `$ rviz`

- 제공된 bag file을 재생

`$ rosbag play os1_sample_20200519.bag`

- 좌측 하단의 "Add"버튼으로 display topic을 추가할 수 있음.



# Rviz

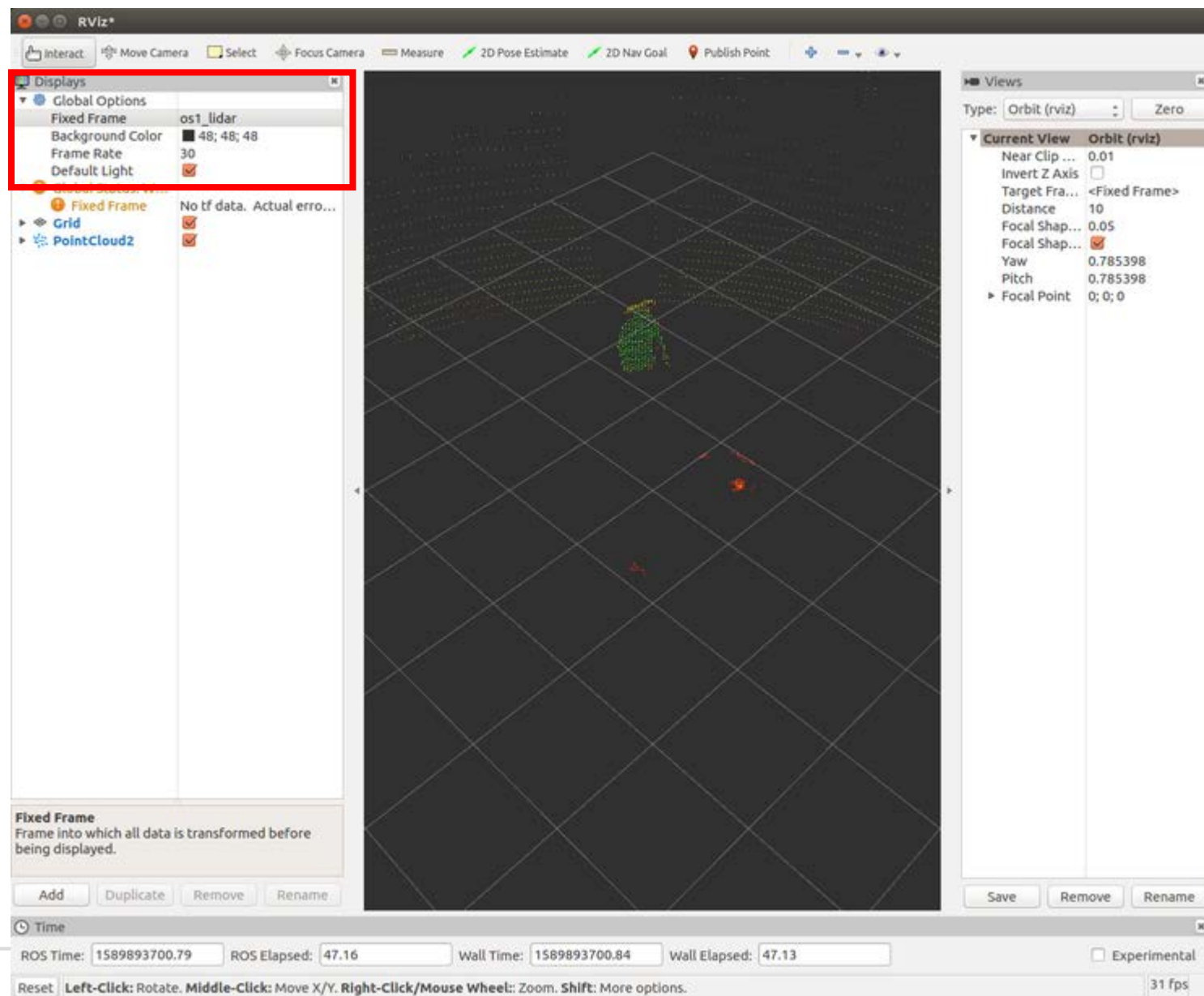
- Rviz
  - Display할 topic을 선택, <OK>



# Rviz

- Rviz

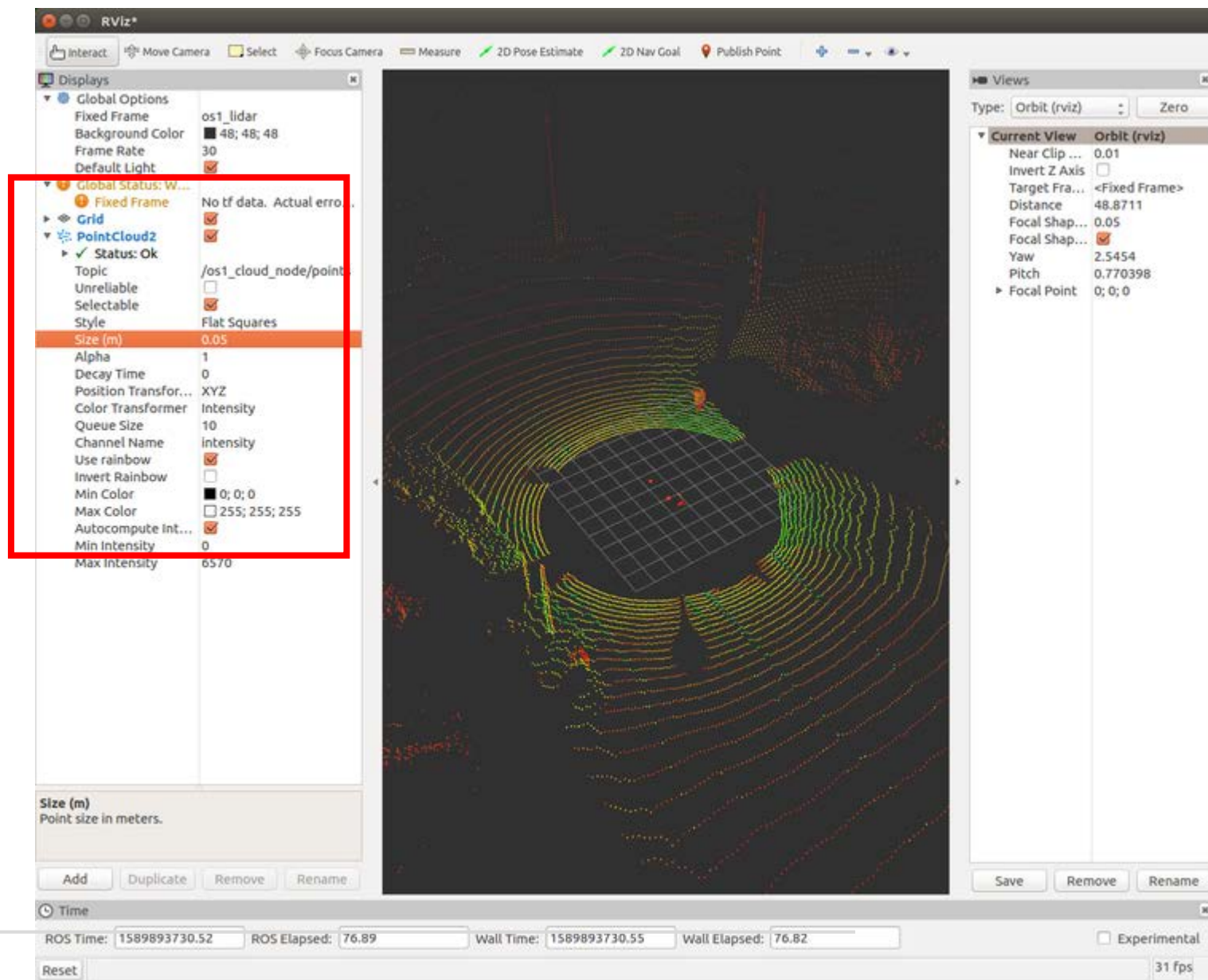
- Display 탭의 global option – Fixed Frame을 os1\_lidar로 변경.
- Fixed Frame은 rviz에서 중심으  
로할 좌표계(tf).



# Rviz

- Rviz

- Topic의 display 옵션을 변경할 수 있음.
- Display화면에서 마우스 좌클릭 드래그, 휠클릭 드래그 등으로 시점 변경 가능.





# Sensor\_msgs

- Sensor\_msgs

- 많이 사용되는 Sensor의 msg type.
- 대표적으로 LiDAR, Camera, IMU 등의 msg type이 정의 되어 있음.
- Rviz에서 시각화하여 확인하기 쉬움.
- 기타 nav\_msgs, geometry\_msgs등 여러 msg들이 정의 되어 있음.
- 제공된 PointCloud2의 데이터 구조

[https://github.com/ros/common\\_msgs](https://github.com/ros/common_msgs)

## sensor\_msgs/PointCloud2 Message

File: `sensor_msgs/PointCloud2.msg`

### Raw Message Definition

```
# This message holds a collection of N-dimensional points, which may
# contain additional information such as normals, intensity, etc. The
# point data is stored as a binary blob, its layout described by the
# contents of the "fields" array.

# The point cloud data may be organized 2d (image-like) or 1d
# (unordered). Point clouds organized as 2d images may be produced by
# camera depth sensors such as stereo or time-of-flight.

# Time of sensor data acquisition, and the coordinate frame ID (for 3d
# points).
Header header

# 2D structure of the point cloud. If the cloud is unordered, height is
# 1 and width is the length of the point cloud.
uint32 height
uint32 width

# Describes the channels and their layout in the binary data blob.
PointField[] fields

bool is_bigendian # Is this data bigendian?
uint32 point_step # Length of a point in bytes
uint32 row_step # Length of a row in bytes
uint8[] data # Actual point data, size is (row_step*height)

bool is_dense # True if there are no invalid points
```

### Compact Message Definition

```
std_msgs/Header header
uint32 height
uint32 width
sensor_msgs/PointField[] fields
bool is_bigendian
uint32 point_step
uint32 row_step
uint8[] data
bool is_dense
```



# Sensor\_msgs

- Std\_msgs/Header

- Timestamp, frame\_id 등을 포함하고 있는 std msg
- 앞서 제공된 bag의 pointcloud2 메시지는 frame\_id가 "os1\_lidar".
- Timestamp는 sec, nano sec단위로 이루어져 있음

## std\_msgs/Header Message

File: `std_msgs/Header.msg`

### Raw Message Definition

```
# Standard metadata for higher-level stamped data types.
# This is generally used to communicate timestamped data
# in a particular coordinate frame.
#
# sequence ID: consecutively increasing ID
uint32 seq
#Two-integer timestamp that is expressed as:
# * stamp.sec: seconds (stamp_secs) since epoch (in Python the variable is called 'secs')
# * stamp.nsec: nanoseconds since stamp_secs (in Python the variable is called 'nsecs')
# time-handling sugar is provided by the client library
time stamp
#Frame this data is associated with
string frame_id
```

### Compact Message Definition

```
uint32 seq
time stamp
string frame_id
```

*autogenerated on Thu, 13 Feb 2020 04:02:12*

# Sensor\_msgs

---

- Sensor\_msg 사용하기

- Pointcloud2 메시지 type은 x,y,z의 point집합으로 이루어져 있지 않음.  
(byte단위로 이루어져 있음)
- Pcl(Point Cloud Library) library를 사용해 xyz로 변경



# Sensor\_msgs

---

- Sensor\_msg 사용하기

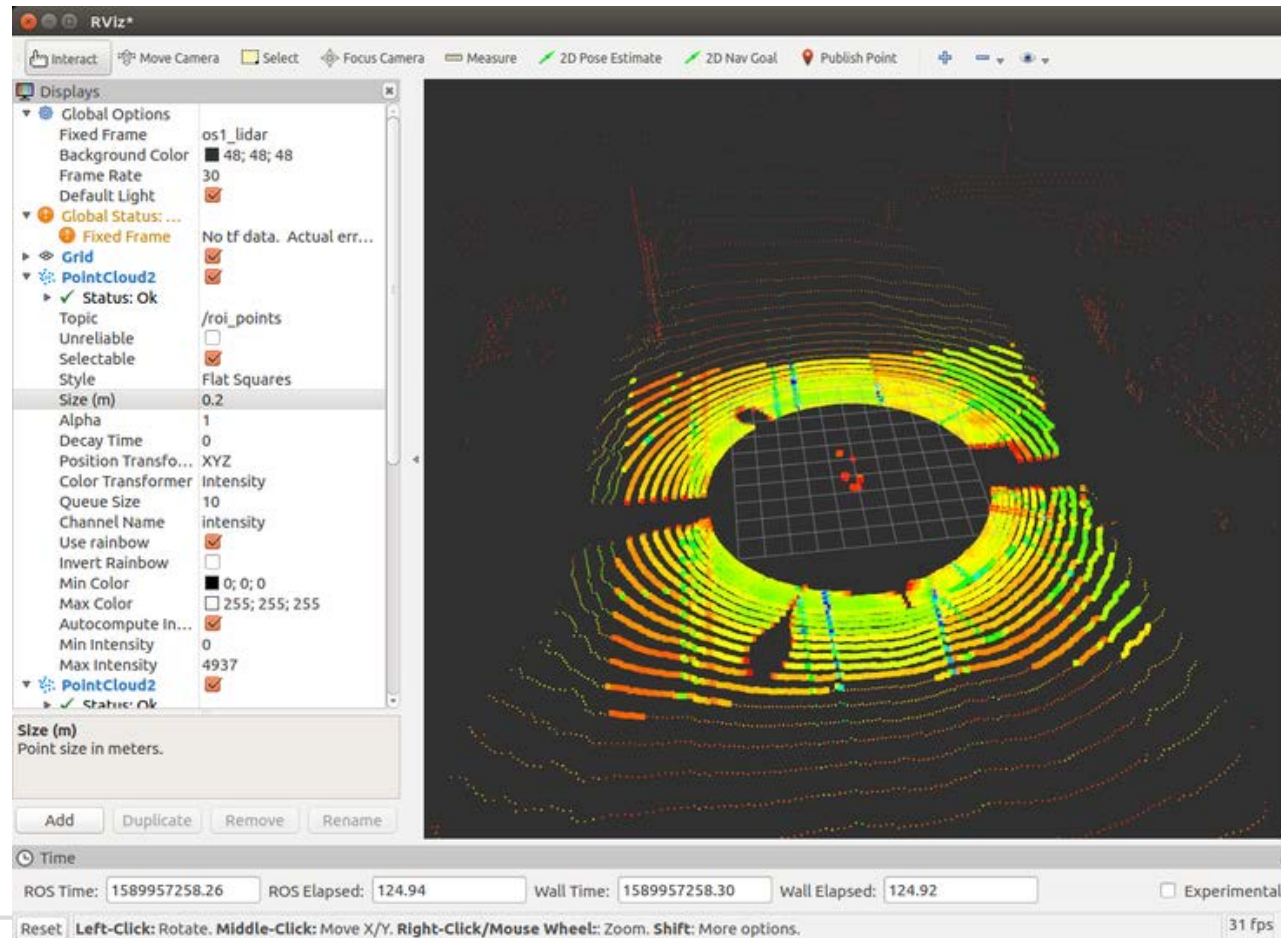
- Sensor\_msgs, Pcl을 사용하기위한 CMakeList와 package xml수정.
- 기존 내용을 수정.

```
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
  message_generation
  pcl_conversions
  pcl_ros
  sensor_msgs
)
find_package(PCL REQUIRED)

include_directories(
  # include
  ${catkin_INCLUDE_DIRS}
  ${PCL_INCLUDE_DIRS}
)
```

# Sensor\_msgs

- Sensor\_msg 사용하기 → ROI Example
  - $\text{abs}(x) < 10$ ,  $\text{abs}(y) < 10$ ,  $\text{abs}(z) < 5$ 인 point를 다시 publish하는 node.



# Sensor\_msgs

---

- Sensor\_msg 사용하기 → ROI Example

- Main 함수 작성, include header
- 제공된 bag의 pcl topic  
"os1\_cloud\_node/points"
- 다시 pointcloud2로 publish하기  
위한 publisher.

```
#include <pcl/point_cloud.h>
```

```
#include <pcl/point_types.h>
```

```
#include <sensor_msgs/PointCloud2.h>
```

```
#include <pcl_conversions/pcl_conversions.h>
```

```
#define ROI_X 10
```

```
#define ROI_Y 10
```

```
#define ROI_Z 5
```

```
ros::Publisher roi_pub;
```

```
int main(int argc, char **argv)
```

```
{
```

```
    ros::init(argc, argv, "pcl_roi");
```

```
    ros::NodeHandle n;
```

```
    ros::Subscriber sub = n.subscribe("/os1_cloud_node/points", 1, pointcloud_callback);
```

```
    roi_pub = n.advertise<sensor_msgs::PointCloud2>("/roi_points", 1);
```

```
    ros::spin();
```

```
    return 0;
```

```
}
```

---

# Sensor\_msgs

---

- ROI Example

- Callback함수 작성.
  - Pcl을 이용,  
sensor\_msgs/pointcloud2를  
pcl::PointXYZ로 변환.
  - pointXYZI의 point가 ROI내  
부에 있는지 검사
  - 원래의 헤더를 복사후 ros  
msg로 변환, publish

```
void pointcloud_callback(sensor_msgs::PointCloud2 msg)
{
    pcl::PointCloud<pcl::PointXYZI> cloud;
    pcl::fromROSMsg(msg, cloud);

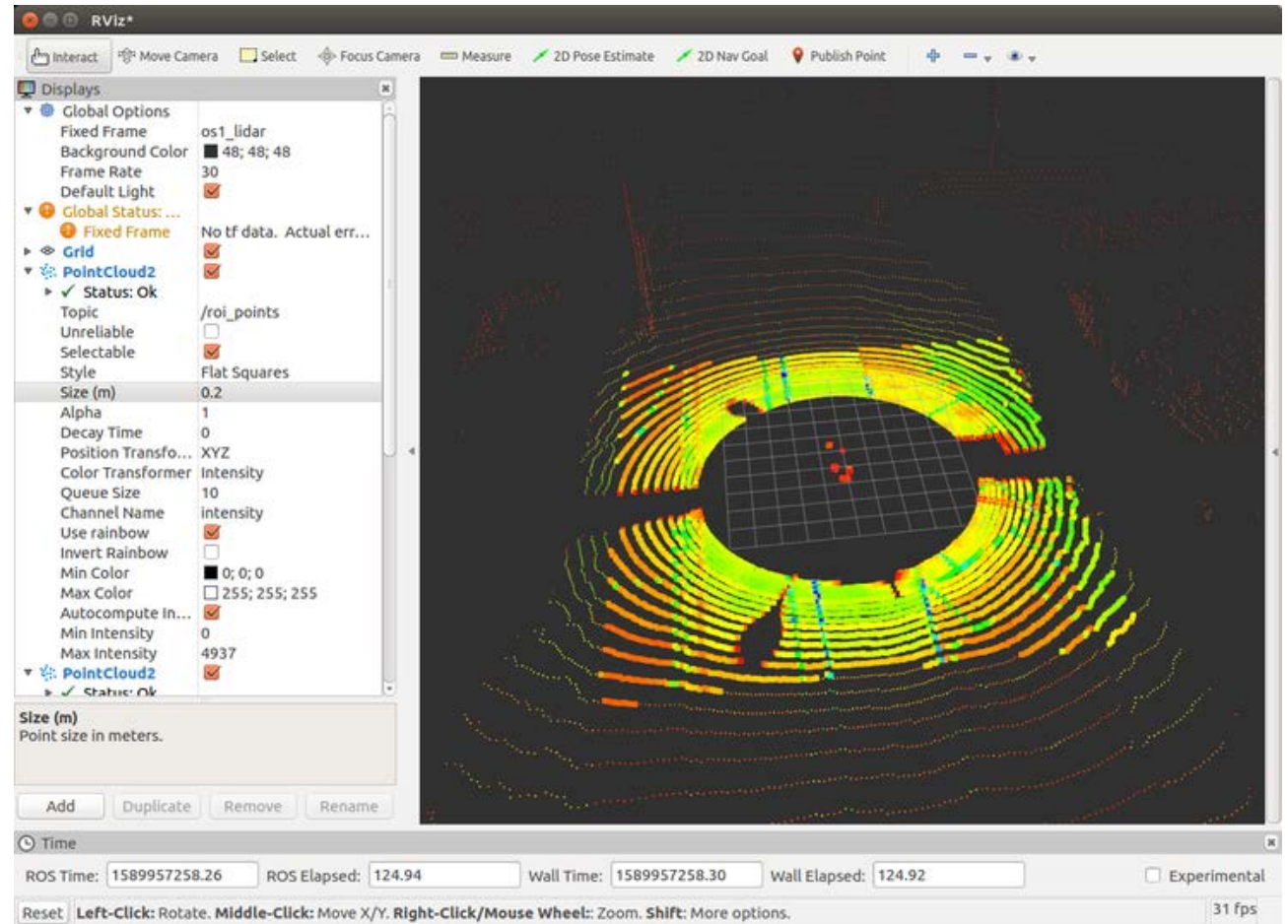
    pcl::PointCloud<pcl::PointXYZI> roi_cloud;
    for (size_t i = 0; i < cloud.size(); i++)
    {
        if(abs(cloud.at(i).x) < ROI_X && (abs(cloud.at(i).y)
< ROI_Y) && abs(cloud.at(i).z) < ROI_Z)
        {
            roi_cloud.push_back(cloud.at(i));
        }
    }
    //sensor header
    roi_cloud.header = cloud.header;
    roi_cloud.is_dense = cloud.is_dense;
    roi_cloud.height = 1;
    roi_cloud.width = roi_cloud.size();
    sensor_msgs::PointCloud2 pub_msg;
    pcl::toROSMsg(roi_cloud, pub_msg);
    roi_pub.publish(pub_msg);
}
```

# Sensor\_msgs

- ROI Example

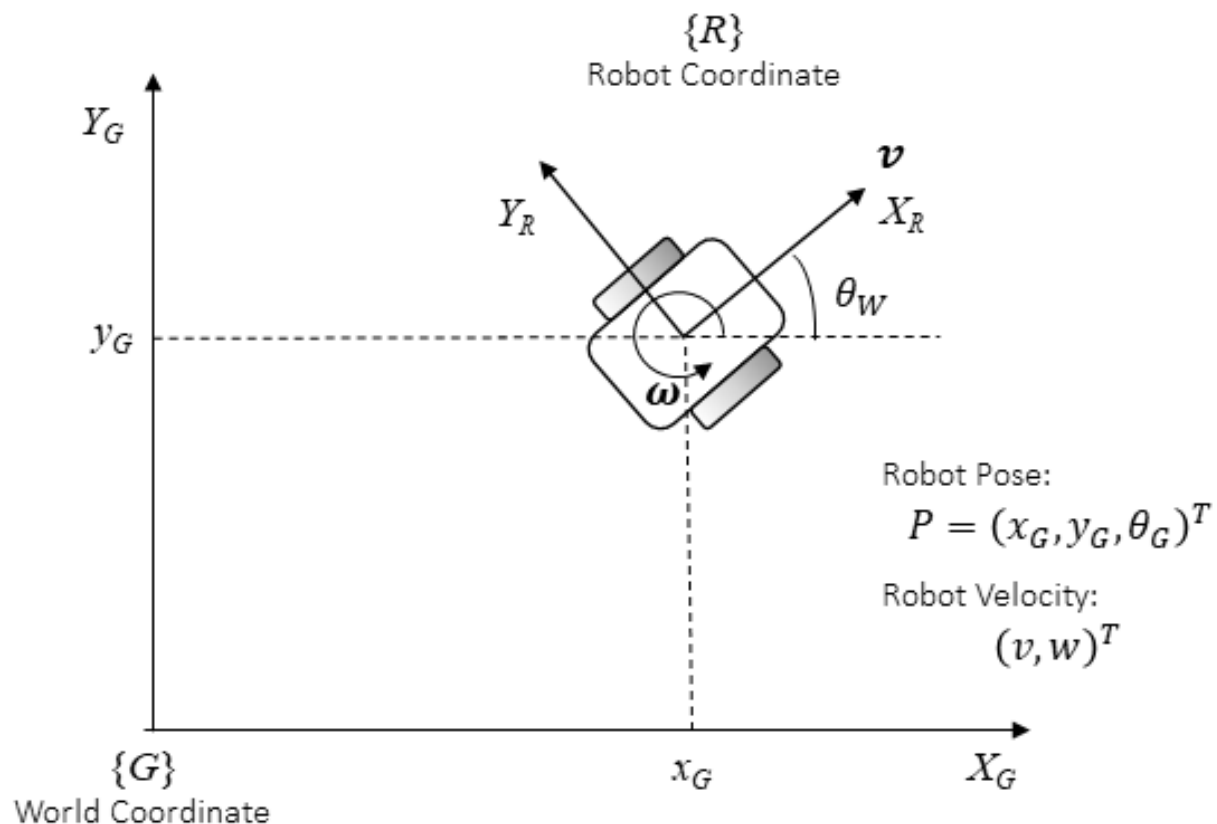
- 결과.

- Fixed Frame = "os1\_lidar"
- 앞서 실습한 "add"로 2가지 topic을 추가.
- 원래의 point size를 더 작게 설정.
- Roi로 publish된 point를 크게 설정해 결과를 확인



- tf

- Robot은 여러 좌표계 (world, robot, sensor...)를 가지고 있음
- 여러 좌표 프레임을 추적 할 수 있는 패키지.
- Rviz위에서 다른 좌표계의 topic을 동시에 볼 수 있게함.
- 몇초전의 프레임을 추적하거나, 쉽게 좌표 변환 행렬을 얻을 수 있음.





- tf – turtle demo

- Dependency 설치

```
$ sudo apt-get install ros-kinetic-ros-tutorials ros-kinetic-geometry-tutorials  
ros-kinetic-rviz ros-kinetic-roslaunch ros-kinetic-rqt-tf-tree
```

- Turtle\_tf demo 실행

```
$ roslaunch turtle_tf turtle_tf_demo.launch
```

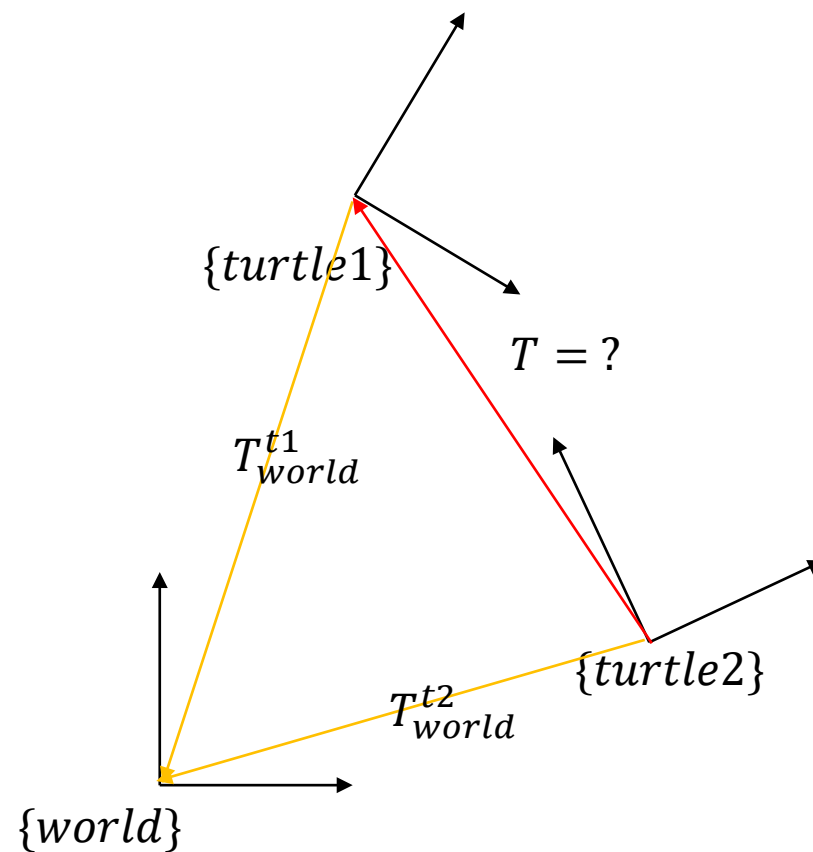
---

- tf – turtle demo
  - Roslaunch 를 실행시킨 터미널에서 방향키로 조작
  - 뒤 거북이가 따라오는것을 볼 수 있음



- tf – turtle demo

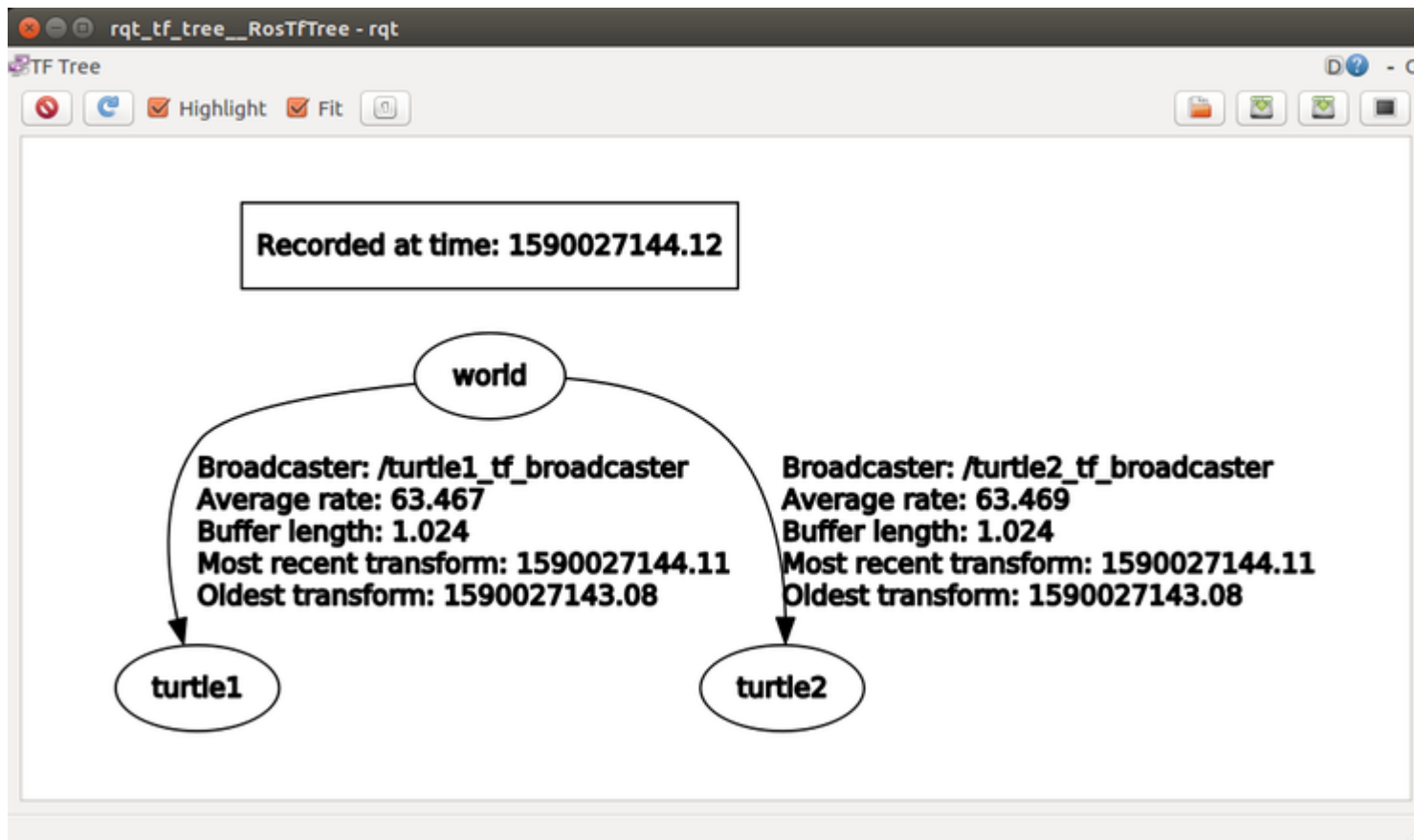
- 3가지의 좌표계가 tf library로 broadcast된 예제(world, turtle1, turtle2)
- Turtle2(따라가는 turtle)가 turtle1(방향키로 움직이는 turtle)의 상대 좌표를 계산후 turtle1을 향해 구동.



- tf – rqt\_tf\_tree

- 현재 broadcast중인 tf tree 확인
- Tree구조 뿐만 아니라  
Broadcaster, rate, buffer length  
등의 정보를 확인 할 수 있음

```
$ rosrun rqt_tf_tree rqt_tf_tree
```



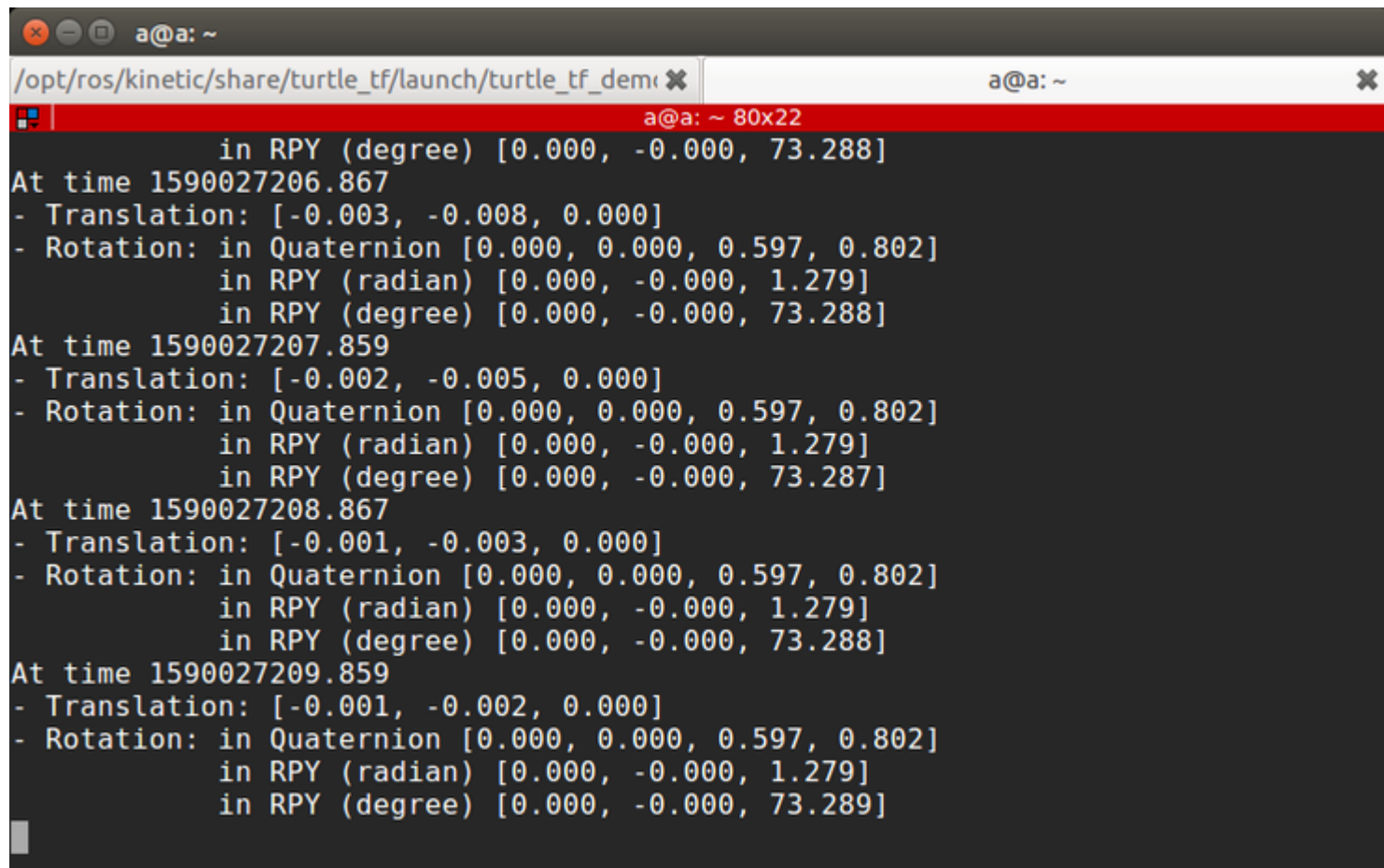
- tf – tf echo

- 이어져 있는 tf 의 정보를 확인 가능.

```
$ rosrun tf tf_echo [reference_frame]  
[target_frame]
```

- 현재 broad cast되는 tf는  
world-turtle1, world-turtle2 지  
만 바로 turtle1-turtle2의  
transform 을 확인가능

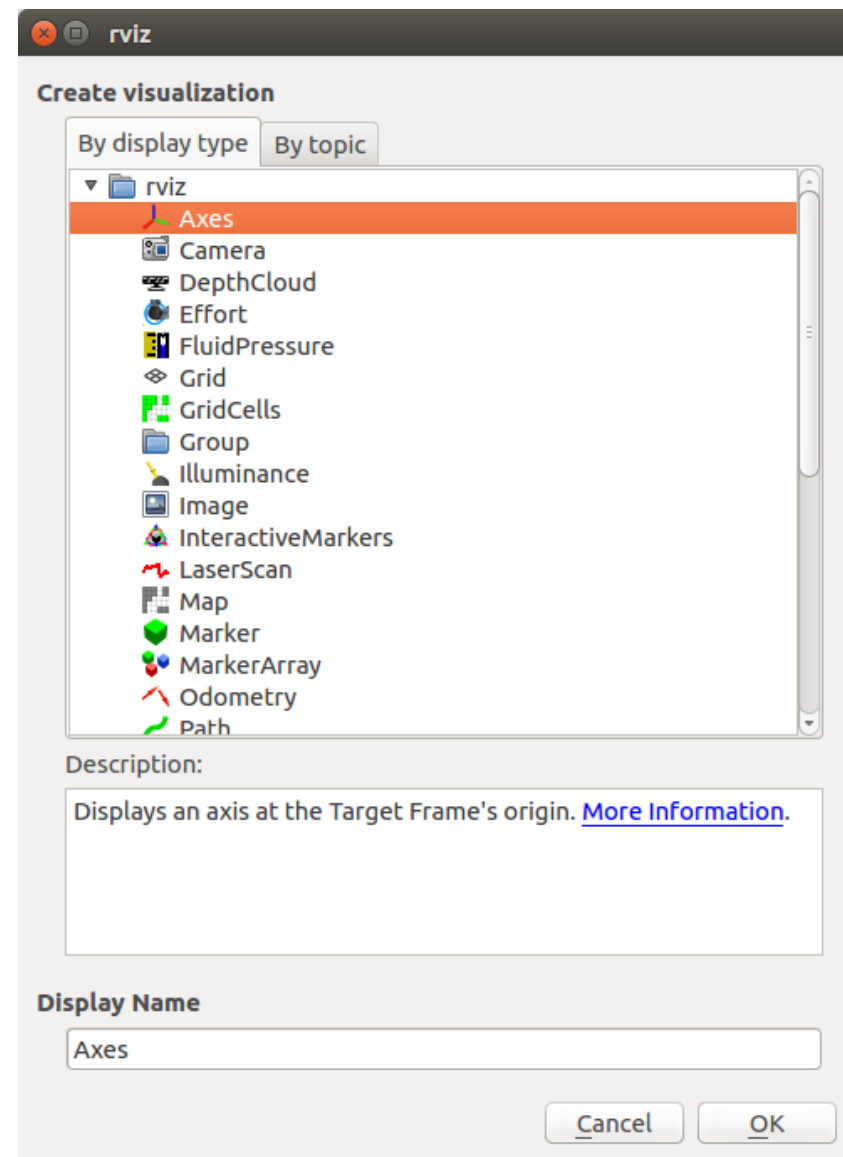
```
$ rosrun tf tf_echo turtle1 turtle2
```



```
a@a: ~  
/opt/ros/kinetic/share/turtle_tf/launch/turtle_tf_demo ⌵ a@a: ~  
a@a: ~ 80x22  
      in RPY (degree) [0.000, -0.000, 73.288]  
At time 1590027206.867  
- Translation: [-0.003, -0.008, 0.000]  
- Rotation: in Quaternion [0.000, 0.000, 0.597, 0.802]  
            in RPY (radian) [0.000, -0.000, 1.279]  
            in RPY (degree) [0.000, -0.000, 73.288]  
At time 1590027207.859  
- Translation: [-0.002, -0.005, 0.000]  
- Rotation: in Quaternion [0.000, 0.000, 0.597, 0.802]  
            in RPY (radian) [0.000, -0.000, 1.279]  
            in RPY (degree) [0.000, -0.000, 73.287]  
At time 1590027208.867  
- Translation: [-0.001, -0.003, 0.000]  
- Rotation: in Quaternion [0.000, 0.000, 0.597, 0.802]  
            in RPY (radian) [0.000, -0.000, 1.279]  
            in RPY (degree) [0.000, -0.000, 73.288]  
At time 1590027209.859  
- Translation: [-0.001, -0.002, 0.000]  
- Rotation: in Quaternion [0.000, 0.000, 0.597, 0.802]  
            in RPY (radian) [0.000, -0.000, 1.279]  
            in RPY (degree) [0.000, -0.000, 73.289]
```

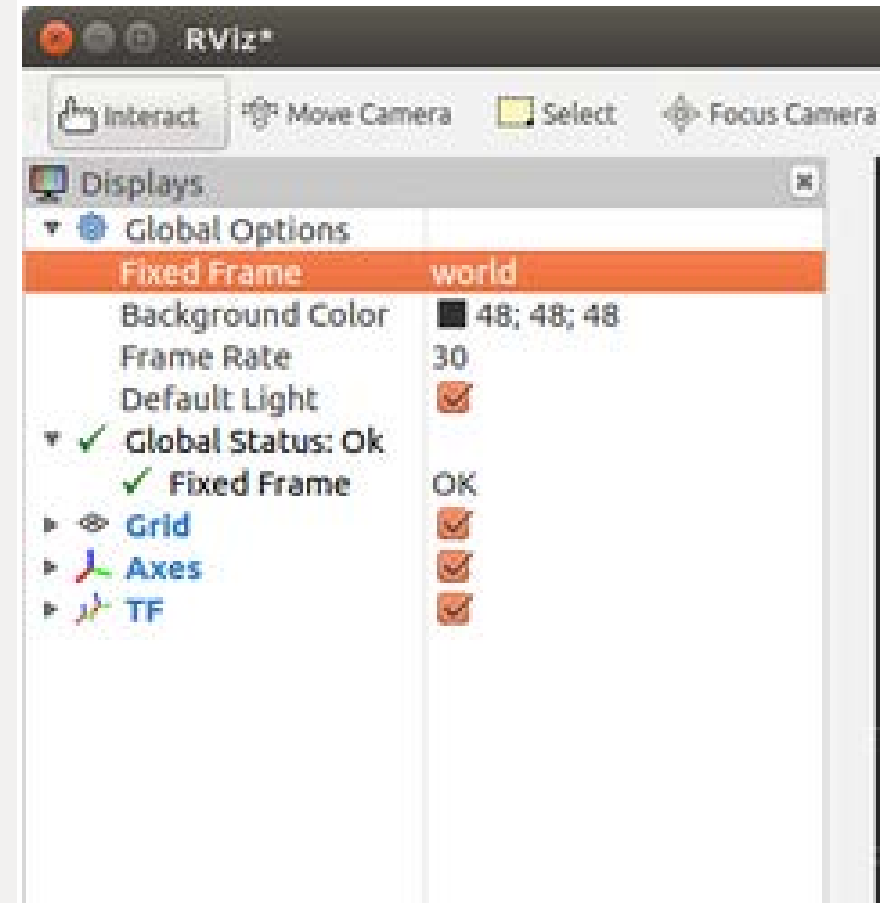
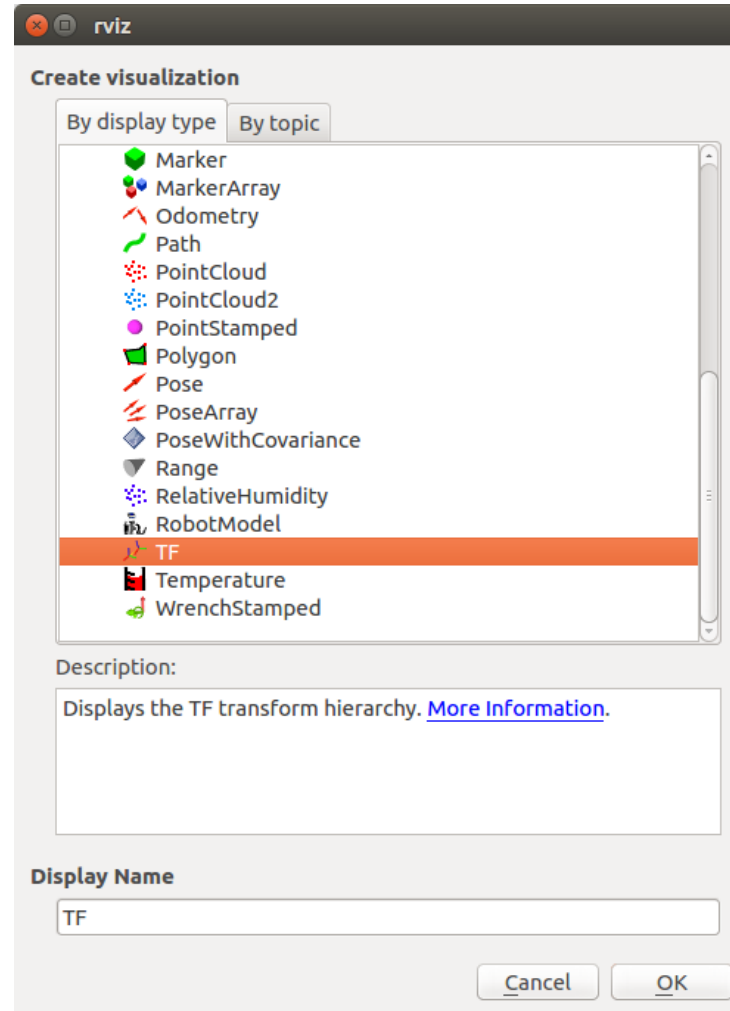
- tf – rviz

- Broad cast되고있는 tf를 visualize.
- 먼저 "Add" – Axes를 추가해 Fixed frame의 축을 확인

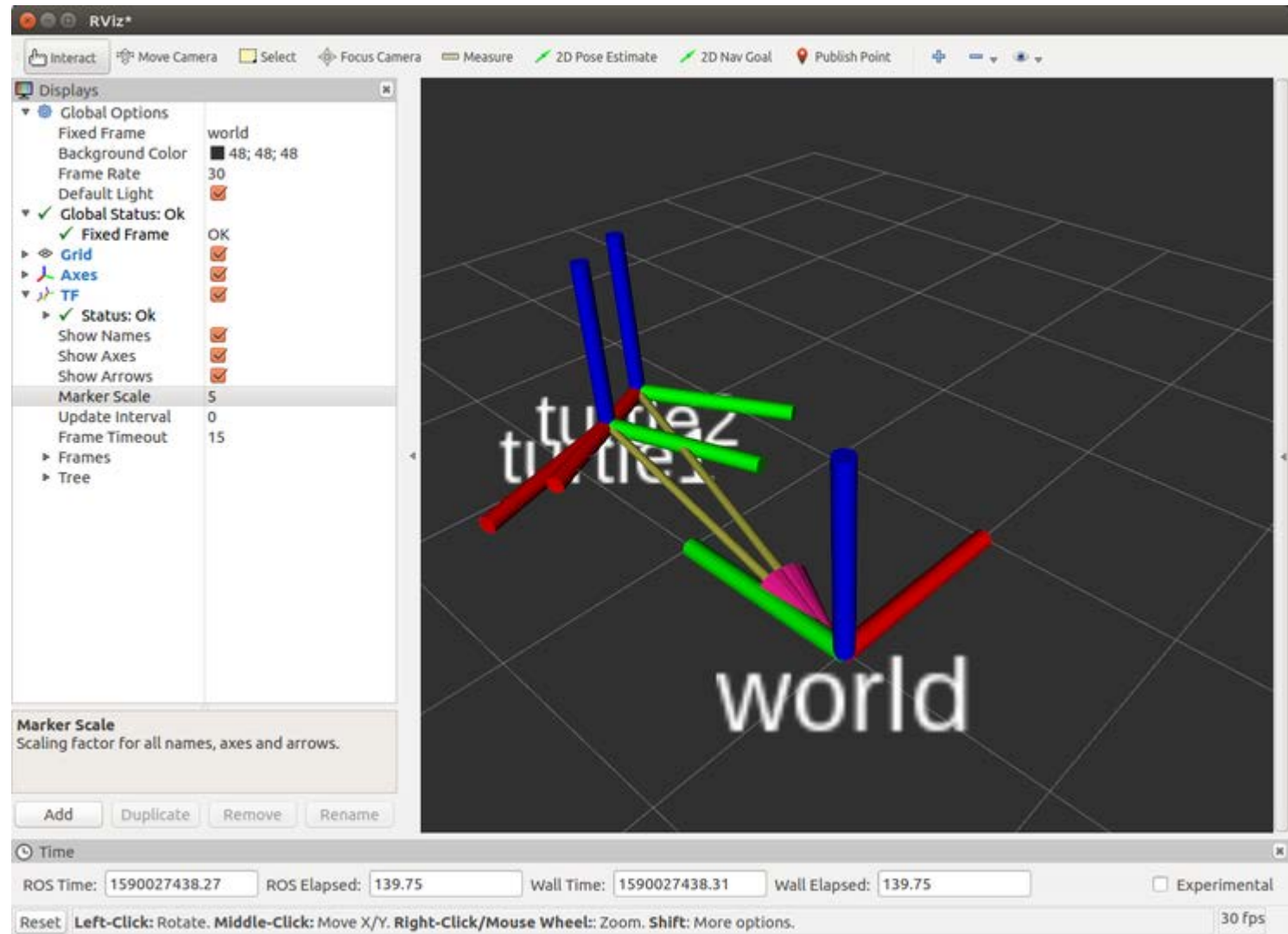


- tf – rviz

- Add – “By display type” 하단에 위치한 TF를 추가.
- Display 탭에서 “Global Options” – “Fixed Frame”을 world로 변경.



- tf – rviz
  - Marker Scale 등의 display option도 조정할 수 있음.





- Tf – broadcaster 예제

- Turtlesim 의 pose를 받아 tf로 publish 하는 예제
- 새로운 패키지를 생성

```
$ cd %YOUR_CATKIN_WORKSPACE_HOME%/src  
$ catkin_create_pkg learning_tf tf roscpp rospy turtlesim
```

- 새로운 패키지를 빌드

```
$ cd %YOUR_CATKIN_WORKSPACE_HOME%/  
$ catkin_make  
$ source ../devel/setup.bash
```

- tf\_broadcaster.cpp작성 <WORKSPACE>/learning\_tf/src/tf\_broadcaster.cpp
-

- Tf – broadcaster 예제

- 새로운 패키지를 생성

```
$ cd %YOUR_CATKIN_WORKSPACE_HOME%/src  
$ catkin_create_pkg learning_tf tf roscpp rospy turtlesim
```

- 새로운 패키지를 빌드

```
$ cd %YOUR_CATKIN_WORKSPACE_HOME%/  
$ catkin_make  
$ source ./devel/setup.bash
```

- tf\_broadcaster.cpp작성 <WORKSPACE>/learning\_tf/src/tf\_broadcaster.cpp
-

- Tf – broadcaster 예제

- tf\_broadcaster.cpp
- Turtlesim pose의 x,y,z, quaternion(회전 변환)을 받아 tf publish.

`br.sendTransform(tf::StampedTransform`  
(설정된 transform , tf의 timestamp,  
부모 프레임, 자식 프레임));

- CMakeList에 add\_executable,  
target\_link\_library를 추가후 빌드.

```
#include <ros/ros.h>
#include <tf/transform_broadcaster.h>
#include <turtlesim/Pose.h>
std::string turtle_name;
void poseCallback(const turtlesim::PoseConstPtr& msg){
    static tf::TransformBroadcaster br;
    tf::Transform transform;
    transform.setOrigin( tf::Vector3(msg->x, msg->y, 0.0) );
    tf::Quaternion q;
    q.setRPY(0, 0, msg->theta);
    transform.setRotation(q);
    br.sendTransform(tf::StampedTransform(transform, ros::Time::now(), "world", turtle_name));
}
int main(int argc, char** argv){
    ros::init(argc, argv, "my_tf_broadcaster");
    if (argc != 2){ROS_ERROR("need turtle name as argument");
return -1;};
    turtle_name = argv[1];
    ros::NodeHandle node;
    ros::Subscriber sub = node.subscribe(turtle_name+"/pose",
10, &poseCallback);
    ros::spin();
    return 0;
};
```

- Tf – broadcaster 예제

- 결과를 확인하기 위한 turtlesim을 포함한 launch file 실행.
- Launch file : 여러 개의 노드, 상수 등을 한번에 설정, 실행하기 위한 실행 명령 파일. 추후에 좀더 자세히 다룰 예정.
- tf\_demo.launch 을 새로 생성하고 오른쪽과 같이 작성.

```
<launch>
  <!-- Turtlesim Node-->
  <node pkg="turtlesim" type="turtlesim_node" name="sim"/>

  <node pkg="turtlesim" type="turtle_teleop_key" name="teleop" output="screen"/>
  <!-- Axes -->
  <param name="scale_linear" value="2" type="double"/>
  <param name="scale_angular" value="2" type="double"/>

  <node pkg="learning_tf" type="turtle_tf_broadcaster"
        args="/turtle1" name="turtle1_tf_broadcaster" />
</launch>
```

- Tf – broadcaster 예제

- 앞장에서 작성한 launch file을  
앞서 생성한 package의 새로운  
폴더 `learning_tf/launch` 에 저  
장후 실행.

```
$ roslaunch learning_tf tf_demo.launch
```

```
<launch>
  <!-- Turtlesim Node-->
  <node pkg="turtlesim" type="turtlesim_node" name="sim"/>

  <node pkg="turtlesim" type="turtle_teleop_key" name="tel
eop" output="screen"/>
  <!-- Axes -->
  <param name="scale_linear" value="2" type="double"/>
  <param name="scale_angular" value="2" type="double"/>

  <node pkg="learning_tf" type="turtle_tf_broadcaster"
    args="/turtle1" name="turtle1_tf_broadcaster" />
</launch>
```

- Tf – broadcaster 예제

- Rviz, rqt\_tf\_tree, tf\_echo 등으로 broadcast된 결과를 확인.

