

# WeMMI\_Gazebo

# 목차

- Gazebo\_패키지 구성 및 실행
- UR 간단한 동작
- Tracer 간단한 동작
- 전체 시스템 동작하기 및 lidar data

# Gazebo 패키지 구성 및 실행

먼저 workspace를 만들고 wemmi패키지를 다운받으시면 되겠습니다.

```
$ mkdir -p ~/wego_ws/src
```

```
$ cd ~/wego_ws/src
```

```
$ git clone https://github.com/WeGo-Robotics/WeMMI\_Gazebo.git
```

디펜던시가 있는 package를 다운받습니다.

```
$ cd ~/wego_ws
```

```
$ rosdep install --from-paths src --ignore-src -r -y
```

그 후에 해당 workspace를 빌드해주고 해당 workspace를 셋업해 줍니다.

```
$ catkin_make
```

```
$ source devel/setup.bash
```

Wemmi 시뮬레이터의 전체적인 패키지 구성은 다음과 같습니다.

- fmauch\_universal\_robot: universal robot에 관련된 package로 universal robot에 대한 드라이버와 하드웨어 파라미터 등이 있는 package입니다
- mobile\_manipulator\_description: mobile manipulator의 전체적인 하드웨어 관한 정보를 볼 수 있는 패키지 입니다.
- Mobile\_manipulator\_moveit\_config: 로봇 팔을 쉽게 제어할 수 있게 해주는 Package입니다. 추후 ROS manipulator 강의에서 자세하게 다뤄집니다.
- Tracer\_ros: 하부 플랫폼인 tracer에 관한 package입니다.
- Ugv\_sdk: 하부 플랫폼에 관련된 package이며 실제 trace와 PC의 통신에 관한 코드가 들어 있습니다.
- Mobile\_manipulator\_example: 이번 강의에서 준비한 example 코드입니다.

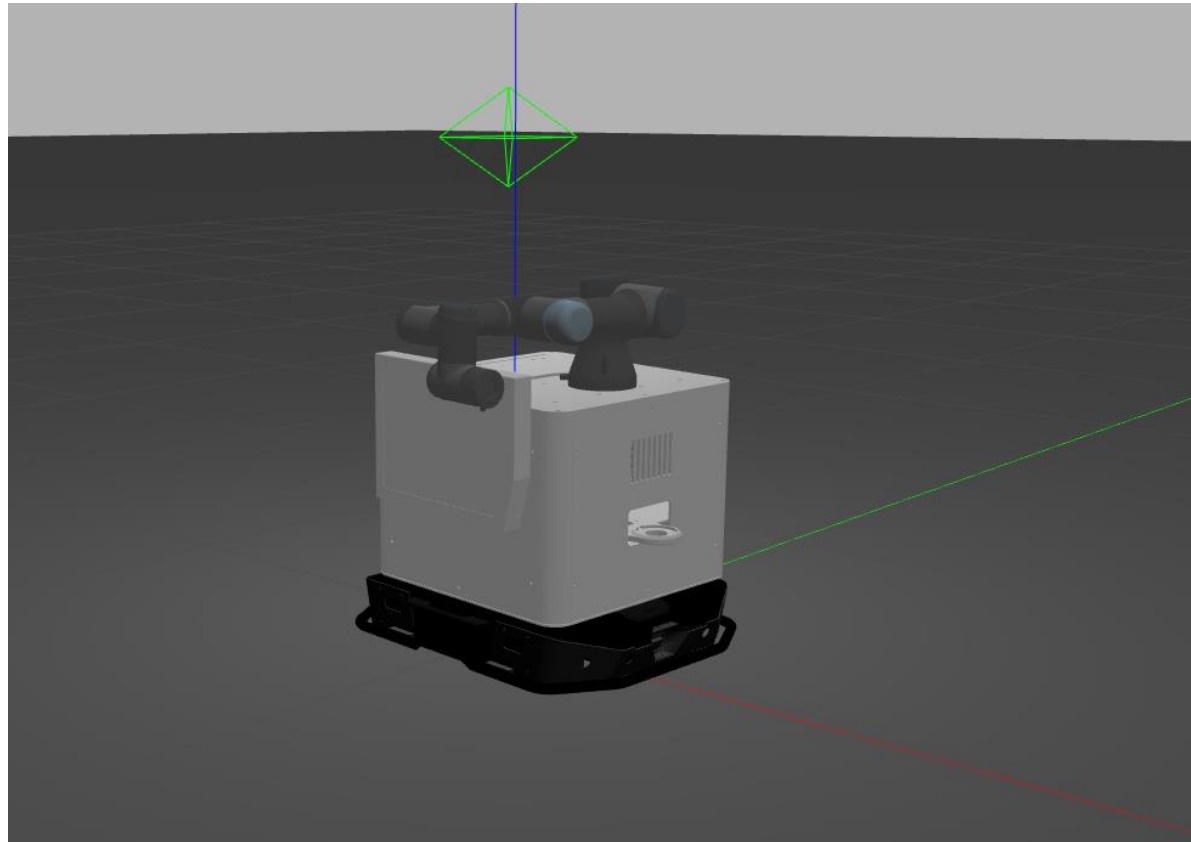
- hector\_slam: hector slam과 gmapping을 수행할 수 있는 package입니다.
- Navigation: Navigation의 기본 package입니다
- scout\_2dnav: Navigation에서 wemmi 에 맞는 parameter들이 튜닝 되어 있는 package입니다.

// scout이란 저희 wego의 하나의 platform이름입니다. 이 이름을 바꾸면 dependency가 걸려있어서 바꾸기가 어렵습니다.

- Wego: navigation에 필요한 launch 파일들을 실행하는 packag입니다.(ex. Navigation 하는데, acml, rviz, move\_base 등 여러 packag가 한번에 켜져야 하는데 이를 하나의 launch 파일에 모아둠)

다음 명령어를 통해서 wemmi gazebo 시뮬레이션을 실행 할 수 있습니다.

```
$ roslaunch tracer_gazebo_sim tracer_empty_world.launch //map0이 없는 gazebo
```

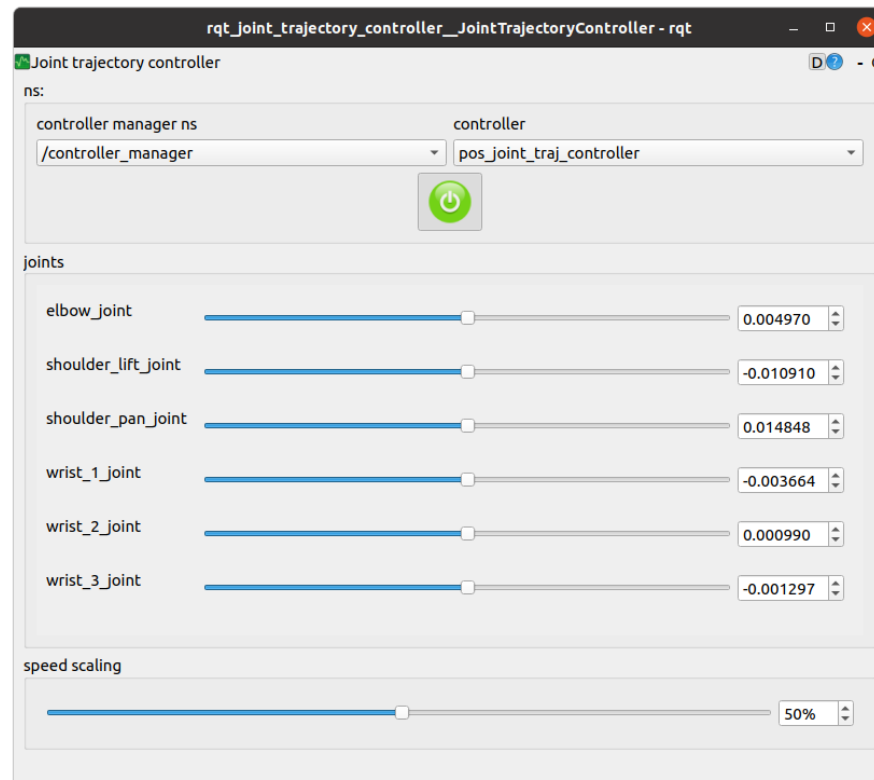


# UR 간단한 동작



Gazebo가 켜져 있는 상황에서 rqt를 통해서 UR을 간단하게 동작시켜 보겠습니다. 각각의 joint들을 움직여 보고 Gazebo상에서 UR이 어떻게 움직이는지 확인해 봅니다.

```
$ rosrn rqt_joint_trajectory_controller rqt_joint_trajectory_controller
```

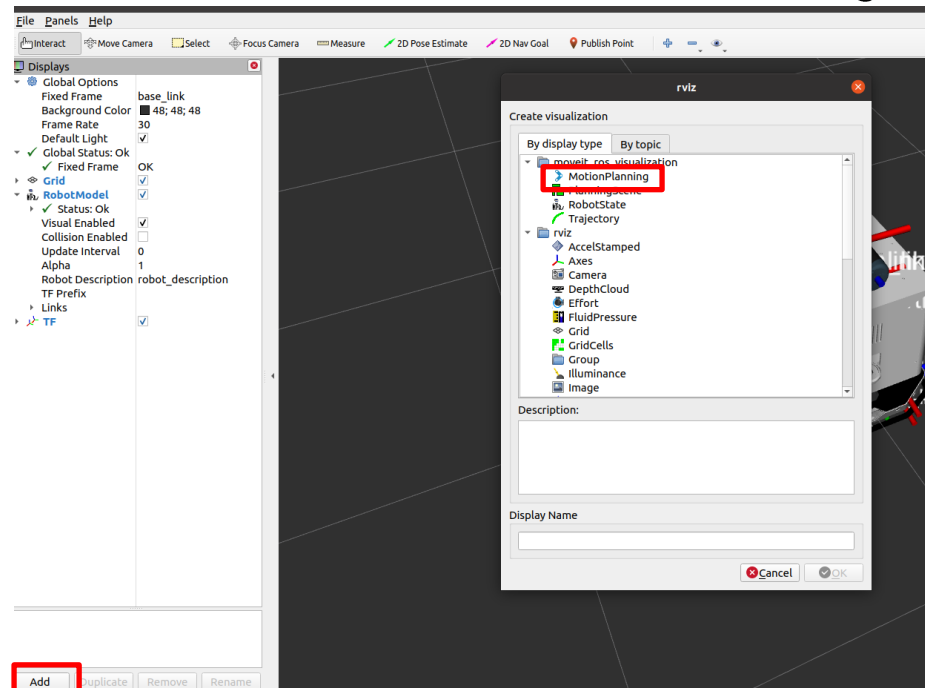


또한 Moveit이란 Package를 사용하면 좀 더 편하게 다음과 같이 움직일 수 있습니다.

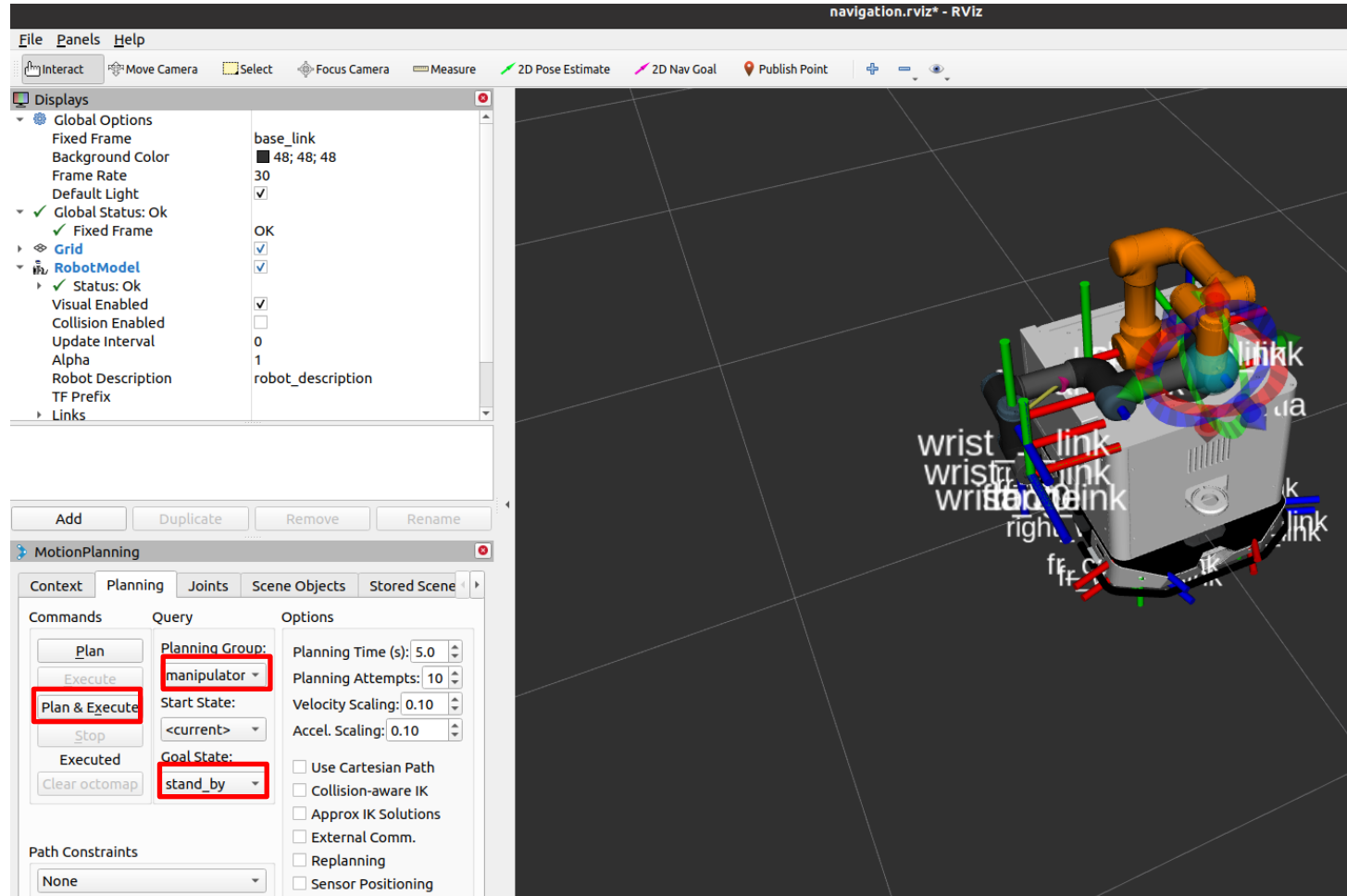
```
$ roslaunch mobile_manipulator_moveit_config move_group.launch
```

```
$ rviz
```

- 이 명령어 후에 rviz 상에서 add 버튼을 누르고 MotionPlanning을 누릅니다.



그 후에 Planning Group을 Manipulator를 선택할 후 GoalState를 stand\_by로 설정하고 Plan & Execution을 눌러줍니다. (주의 Gazebo 상에서 움직일 때만 이렇게 동작을 시키며, 실제 로봇을 동작할 때는 Plan 버튼과 Execution 버튼을 따로 눌러주시기 바랍니다)



이제 마지막으로 c++로 작성된 코드로 간단하게 동작시켜 보겠습니다.

- Gazebo와 Moveit이 시작된 상태, rviz는 꺼져 있어도 됨

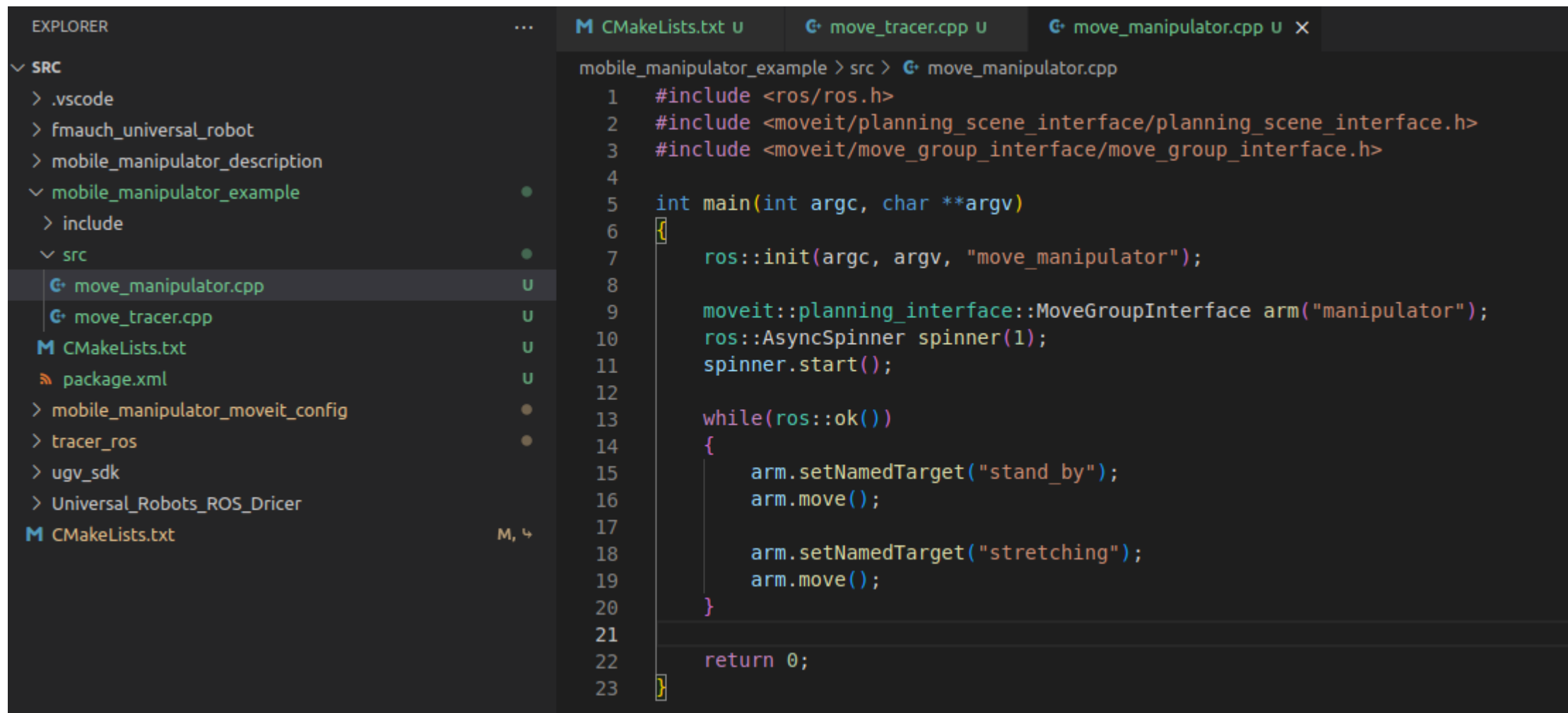
```
$ rosrun mobile_manipulator_example move_manipulator
```

- 다음 파일을 실행시키면 manipulator가 stan\_by와 stretching을 반복하며 움직일 것입니다.

- 코드는 다음 위치에 있습니다.

```
~/catkin_ws/src/mobile_manipulator_example/src/move_manipulator.cpp
```

만약에 코드를 수정하시면 catkin\_make로 빌드를 다시 진행해 주시기 바랍니다.



```
mobile_manipulator_example > src > move_manipulator.cpp
1  #include <ros/ros.h>
2  #include <moveit/planning_scene_interface/planning_scene_interface.h>
3  #include <moveit/move_group_interface/move_group_interface.h>
4
5  int main(int argc, char **argv)
6  {
7      ros::init(argc, argv, "move_manipulator");
8
9      moveit::planning_interface::MoveGroupInterface arm("manipulator");
10     ros::AsyncSpinner spinner(1);
11     spinner.start();
12
13     while(ros::ok())
14     {
15         arm.setNamedTarget("stand_by");
16         arm.move();
17
18         arm.setNamedTarget("stretching");
19         arm.move();
20     }
21
22     return 0;
23 }
```

# Tracer 간단한 동작

Tracer는 cmd\_vel이라는 topic을 통해서 제어 됩니다. Gazebo가 실행된 상태에서 다음 명령을 통해서 cmd\_vel이라는 topic이 발행되고 있는 것을 확인할 수 있습니다.

\$ rostopic list

```
wego@wego-GS66-Stealth-10SD:~/ros_workspace/wemmi_gazebo_ws$ rostopic list
/clicked_point
/clock
/cmd_vel
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/performance_metrics
/gazebo/set_link_state
/gazebo/set_model_state
/initialpose
/joint_group_pos_controller/command
/joint_states
/move_base_simple/goal
/pos_joint_traj_controller/command
/pos_joint_traj_controller/follow_joint_trajectory/cancel
/pos_joint_traj_controller/follow_joint_trajectory/feedback
/pos_joint_traj_controller/follow_joint_trajectory/goal
/pos_joint_traj_controller/follow_joint_trajectory/result
/pos_joint_traj_controller/follow_joint_trajectory/status
/pos_joint_traj_controller/state
/rosout
/rosout_agg
/tf
/tf_static
/tracer_motor_l_controller/command
/tracer_motor_r_controller/command
```

- cmd\_vel 이라는 토픽을 다음과 같이 발행하여 Wemmi를 움직여 봅니다. cmd\_vel이라는 토픽은 Tracer의 속도를 조절할 수 있는 토픽입니다.

```
$ rostopic pub -r 10 /cmd_vel geometry_msgs/Twist "linear:
```

```
x: 0.5
```

```
y: 0.0
```

```
z: 0.0
```

```
angular:
```

```
x: 0.0
```

```
y: 0.0
```

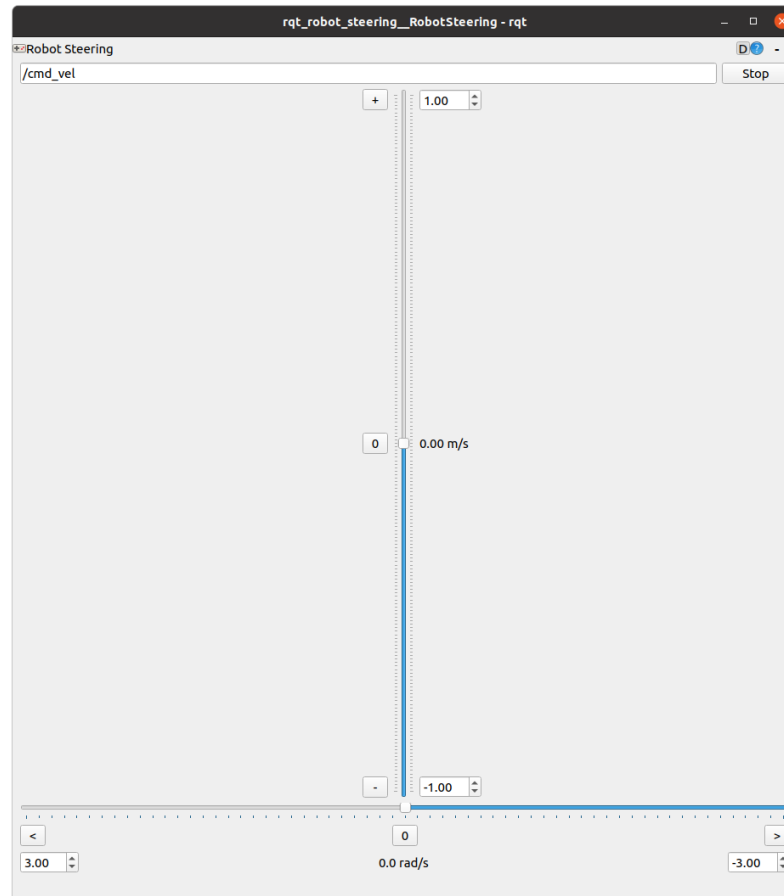
```
z: 0.5“
```

해당 명령어는 다음과 같이 해석이 됩니다. /cmd\_vel이라는 이름의 토픽의 msg타입은 geometry\_msgs/Twist이고 이 값의 linear의 x 값에는 0.5, angular 값의 z는 0.5의 값을 넣어서 10HZ의 속도로 토픽을 발행해라, 이 명령어를 통해서 Gazebo 상에서의 Wemmi가 원을 그리면서 움직이는 것을 볼 수 있습니다.



- 또는 rqt를 통해서도 trace를 움직일 수 있습니다.

**\$ rosrun rqt\_robot\_steering rqt\_robot\_steering**



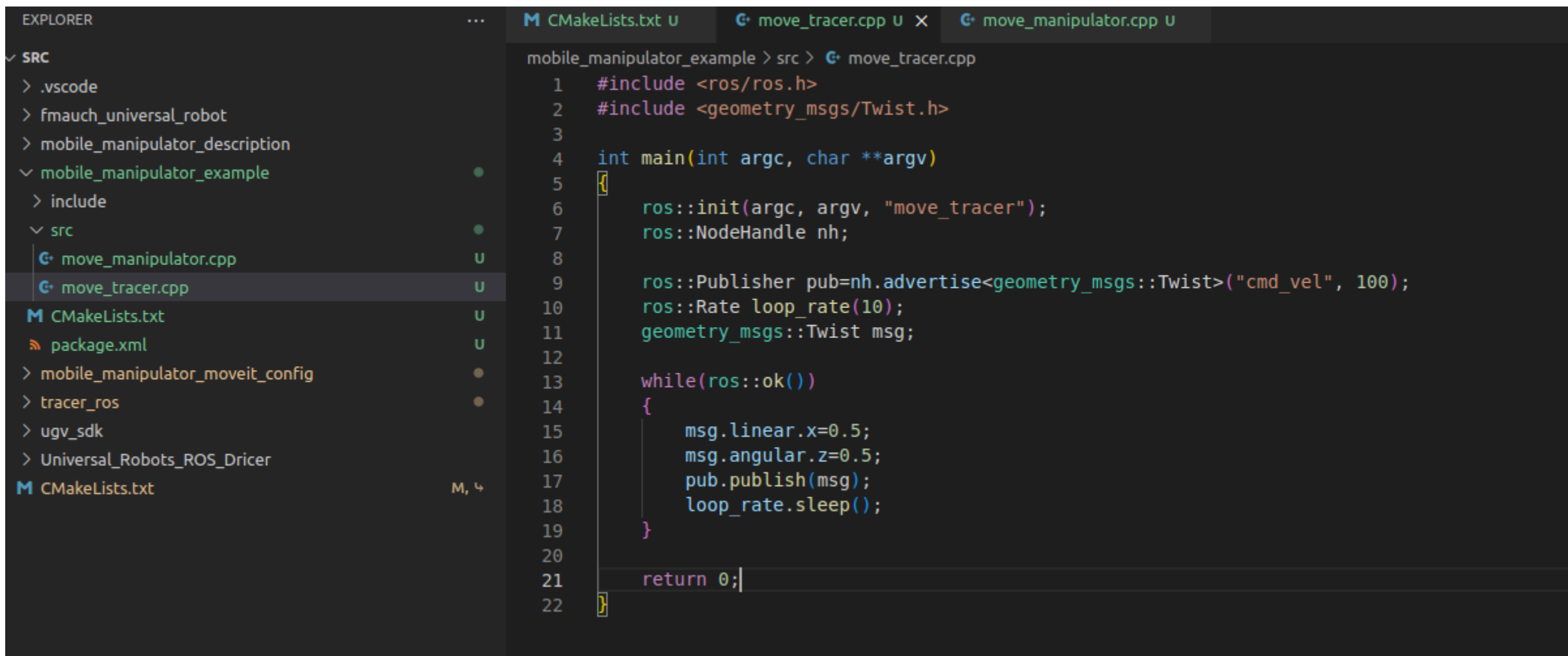
이제 마지막으로 c++로 작성된 코드로 간단하게 동작시켜 보겠습니다.

```
$ rosrun mobile_manipulator_example move_tracer
```

- 다음 파일을 실행시키면 wemmi가 원을 그리며 움직일 것입니다
- 코드는 다음 위치에 있습니다.

```
~/catkin_ws/src/mobile_manipulator_example/src/move_tracer.cpp
```

만약에 코드를 수정하시면 catkin\_make로 빌드를 다시 진행해 주시기 바랍니다.



The image shows a VS Code editor window with a project structure on the left and a C++ source file on the right.

**EXPLORER (Left Panel):**

- ✓ SRC
  - > .vscode
  - > fmauch\_universal\_robot
  - > mobile\_manipulator\_description
  - ✓ mobile\_manipulator\_example
    - > include
    - ✓ src
      - move\_manipulator.cpp
      - move\_tracer.cpp
  - CMakeLists.txt
  - package.xml
  - > mobile\_manipulator\_moveit\_config
  - > tracer\_ros
  - > ugv\_sdk
  - > Universal\_Robots\_ROS\_Driver
- CMakeLists.txt

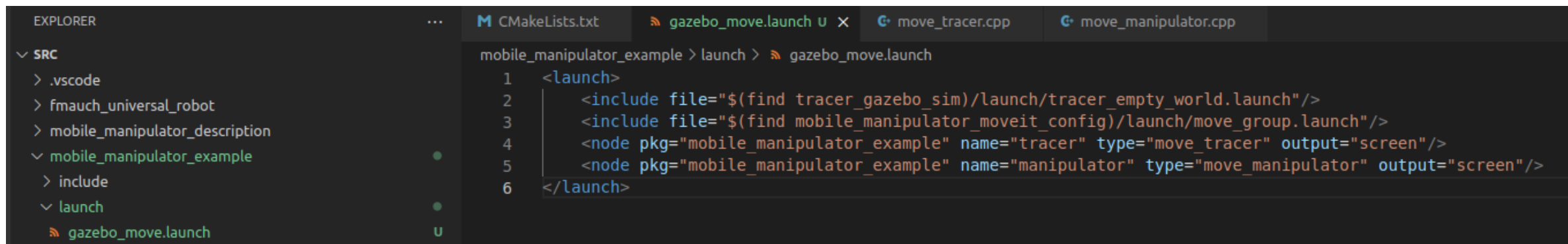
**move\_tracer.cpp (Right Panel):**

```
mobile_manipulator_example > src > move_tracer.cpp
1  #include <ros/ros.h>
2  #include <geometry_msgs/Twist.h>
3
4  int main(int argc, char **argv)
5  {
6      ros::init(argc, argv, "move_tracer");
7      ros::NodeHandle nh;
8
9      ros::Publisher pub=nh.advertise<geometry_msgs::Twist>("cmd_vel", 100);
10     ros::Rate loop_rate(10);
11     geometry_msgs::Twist msg;
12
13     while(ros::ok())
14     {
15         msg.linear.x=0.5;
16         msg.angular.z=0.5;
17         pub.publish(msg);
18         loop_rate.sleep();
19     }
20
21     return 0;
22 }
```

# 전체 시스템 동작 및 Lidar data

이제는 launch 파일을 작성하여 전에 작성했던 코드를 한번에 실행시키는 launch 파일을 실행시키겠습니다.

mobile\_manipulator\_example 패키지에 launch 라는 폴더를 만들고 안에 gazebo\_move.launch를 만들어 봤습니다.



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows the directory structure of the mobile\_manipulator\_example package, with the launch folder selected. The code editor shows the content of the gazebo\_move.launch file, which is a ROS launch file that includes two other launch files and runs two nodes: move\_tracer and move\_manipulator.

```
mobile_manipulator_example > launch > gazebo_move.launch
1 <launch>
2   <include file="$(find tracer_gazebo_sim)/launch/tracer_empty_world.launch"/>
3   <include file="$(find mobile_manipulator_moveit_config)/launch/move_group.launch"/>
4   <node pkg="mobile_manipulator_example" name="tracer" type="move_tracer" output="screen"/>
5   <node pkg="mobile_manipulator_example" name="manipulator" type="move_manipulator" output="screen"/>
6 </launch>
```

그리고 이 launch 파일을 실행해 봅니다.

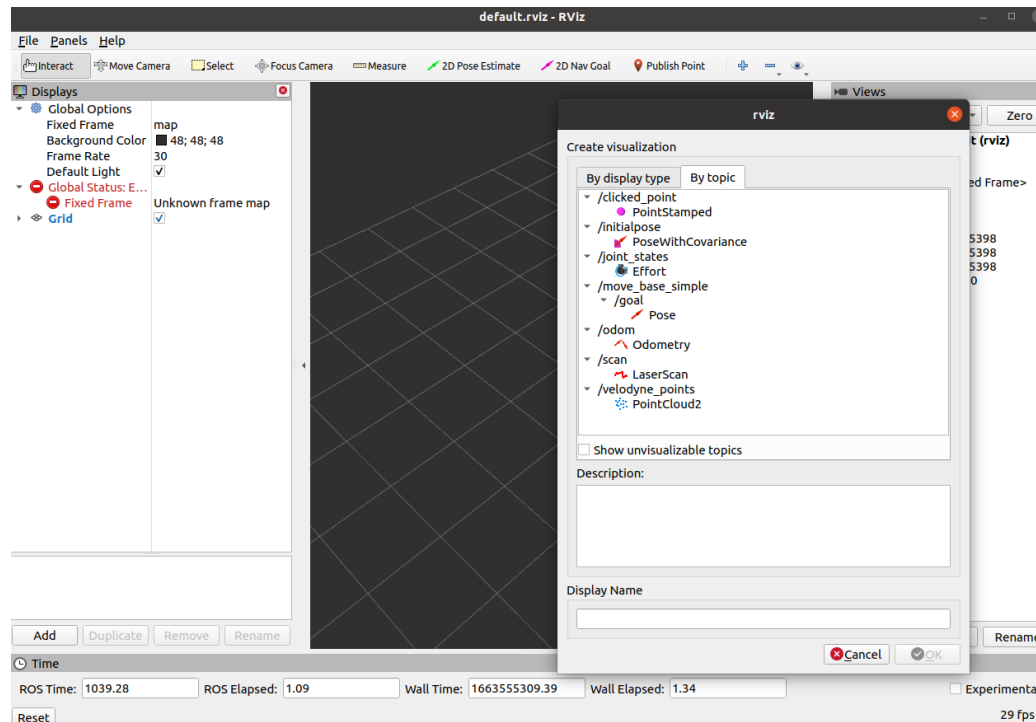
```
$ roslaunch mobile_manipulator_example gazebo_move.launch
```

Rviz를 켜고 Lidar data를 확인해 보실 수 있습니다.

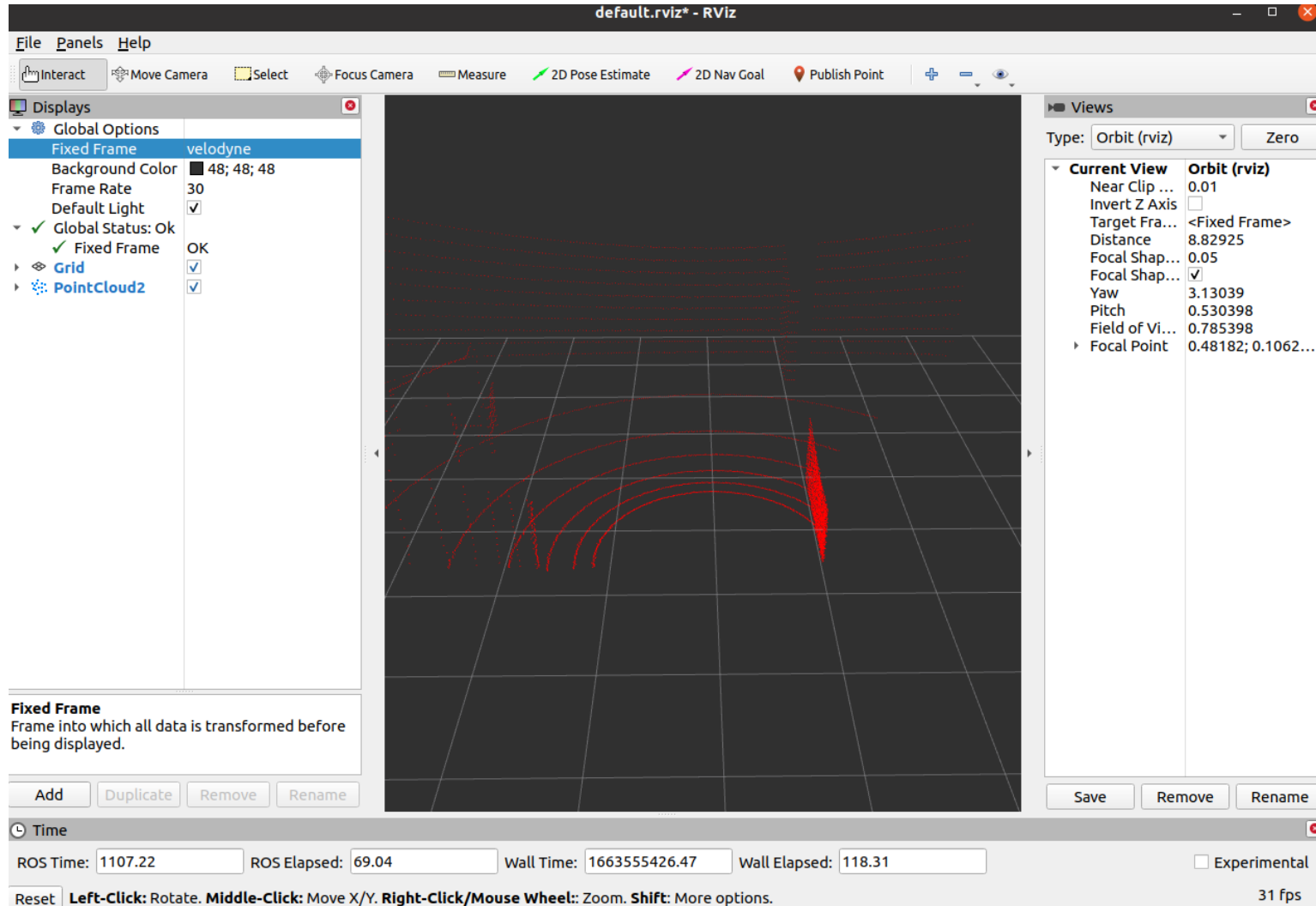
- Gazebo키시고 아무 물건이나 gazebo에 추가해 주시면 되겠습니다.

\$ rviz

Add 버튼을 누르고 PointCloud2를 눌러주시기 바랍니다.



이제 Fixed frame을 velodyne으로 두면 Lidar data가 들어 오는 것을 확인 할 수 있습니다.





본사(기술연구소 및 사무실) : 16914 경기도 용인시 기흥구 구성로 357(청덕동) 용인테크노밸리 B동 513호

기술연구소(서울) : 04799 서울특별시 성동구 성수동2가 280-13, 삼환디지털벤처타워 401호

대표전화 : 031 – 229 – 3553

팩스 : 031 – 229 – 3554

제품문의: [go.sales@wego-robotics.com](mailto:go.sales@wego-robotics.com)

기술문의: [go.support@wego-robotics.com](mailto:go.support@wego-robotics.com)