

INSTALLATION GUIDE

last modified date : 2017.08.25

1. dbw_mkz 패키지를 설치합니다

```
$ sudo sh -c 'echo "deb [ arch=amd64 ] http://packages.dataspeedinc.com/ros/ubuntu $(lsb_release -sc) main"
> /etc/apt/sources.list.d/ros-dataspeed-public.list'

$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys FF6D3CDA

$ sudo apt-get update

$ sudo apt-get install -y ros-indigo-mkz-*
```

2. velodyne 패키지를 설치합니다

```
$ sudo apt-get install -y ros-indigo-velodyne*
```

3. dyros_simulator_ws 파일의 데이터를 catkin_ws로 전부 복사한 후 catkin_make를 수행합니다

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

INSTALLATION GUIDE

4. parking_lot.world의 경로를 바꿔줍니다

- /dyros_simulator_ws/src/dyros_simulator/dyros_simualtor/worlds 폴더 안에 parking_lot.world 파일을 엽니다
- Ctrl + F로 home 단어를 검색하면 어떤 경로가 나옵니다

ex) <uri>file:///home/dyros-vehicle/catkin_ws/src
/dyros_simulator/dyros_simualtor/meshes/parking_lot/parking_lot.dae</uri>

- 위 경로를 현재 컴퓨터의 경로에 맞게 수정해줍니다 (2군데 있습니다)

5. dyros_simulator를 실행합니다

```
$ roslaunch dyros_simulator dyros_simulator.launch
```

6. dyros_teleop_keyboard를 실행합니다

```
$ rosrn dyros_teleop_keyboard dyros_teleop_keyboard.py
```



dyros_teleop_keyboard.py가 나타나지 않는 경우
~/catkin_ws/src/dyros_simulator/dyros_teleop_keyboard 에 들어가서
다음 명령어를 실행합니다

```
$ chmod 755 dyros_teleop_keyboard.py
```

Simulator TEST #1 (motion_planner)

1. dyros_simulator를 실행합니다

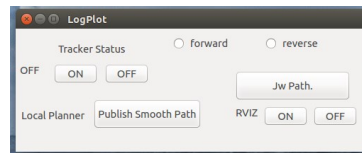
```
$ roslaunch dyros_simulator dyros_simulator.launch
```

2. dyros_teleop_keyboard를 실행합니다

```
$ rosrun dyros_teleop_keyboard dyros_teleop_keyboard.py
```

3. motion_planner를 실행합니다

```
$ rosrun motion_planner motion_planner
```

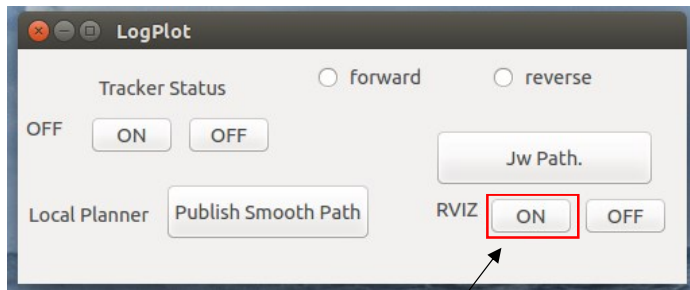


4. waypoints 폴더를 ~/bag_files/ 폴더 안에 넣습니다 (폴더가 없으면 mkdir ~/bag_files로 폴더를 만든 후 넣습니다)

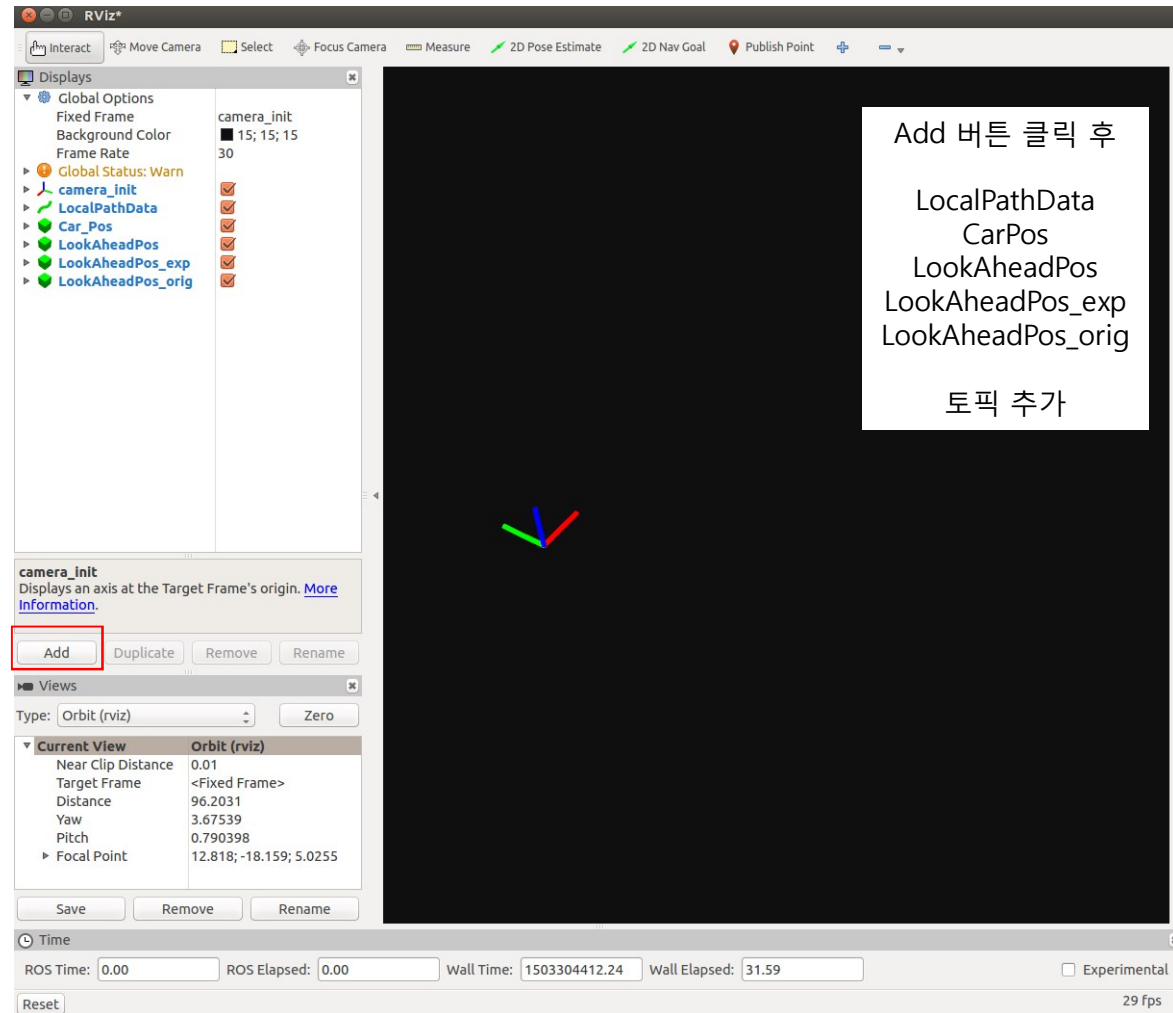
Home bag_files waypoints						
	Name	Size	Type	Modified	Permissions	
ent	 parking_lot_ed.map	1.2 MB	Text	8월 17	-rw-rw-r--	

Simulator TEST #1 (motion_planner)

5. motion_planner에서 RVIZ ON 버튼을 클릭하고 다음과 같이 세팅해줍니다



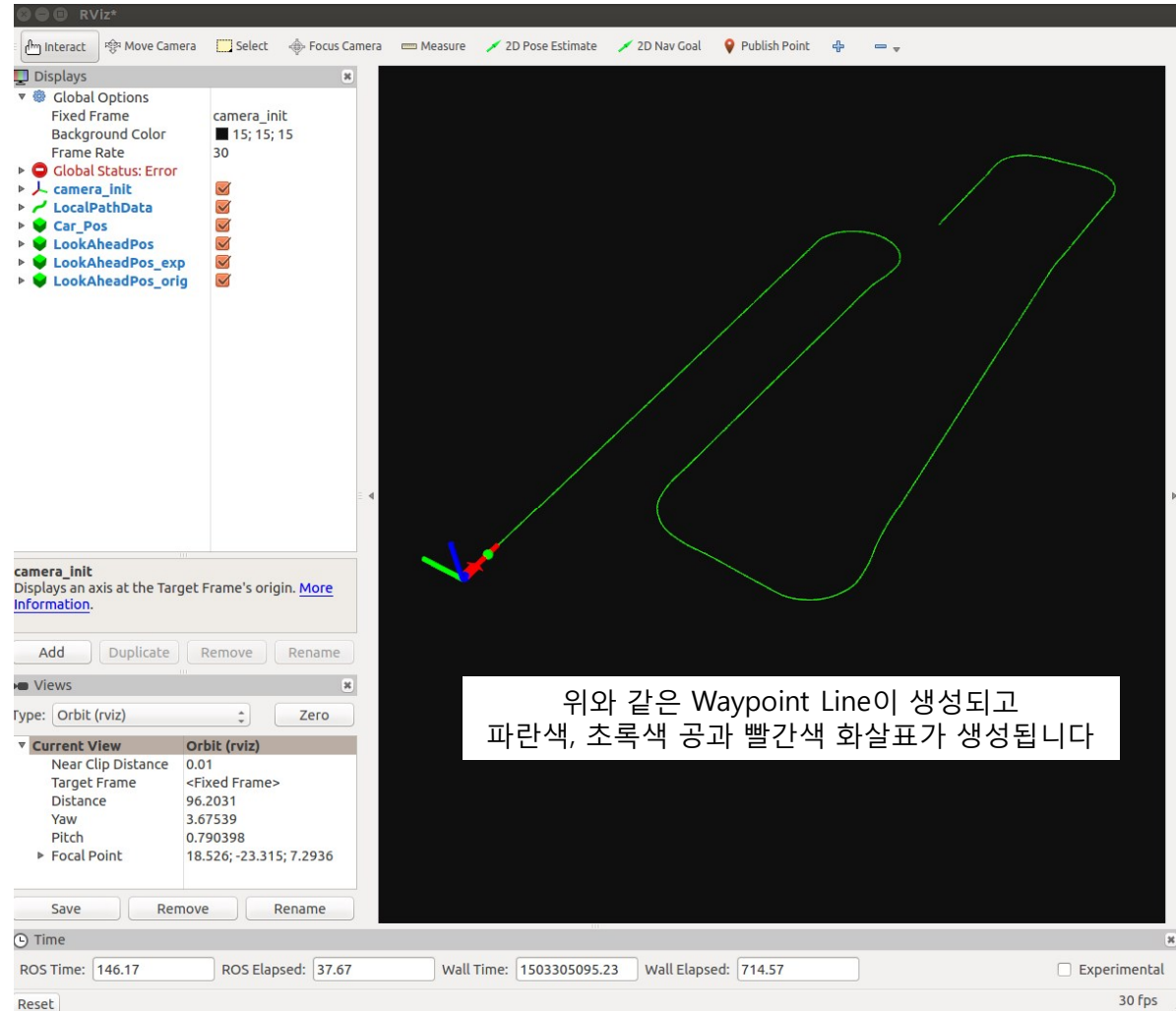
클릭



클릭

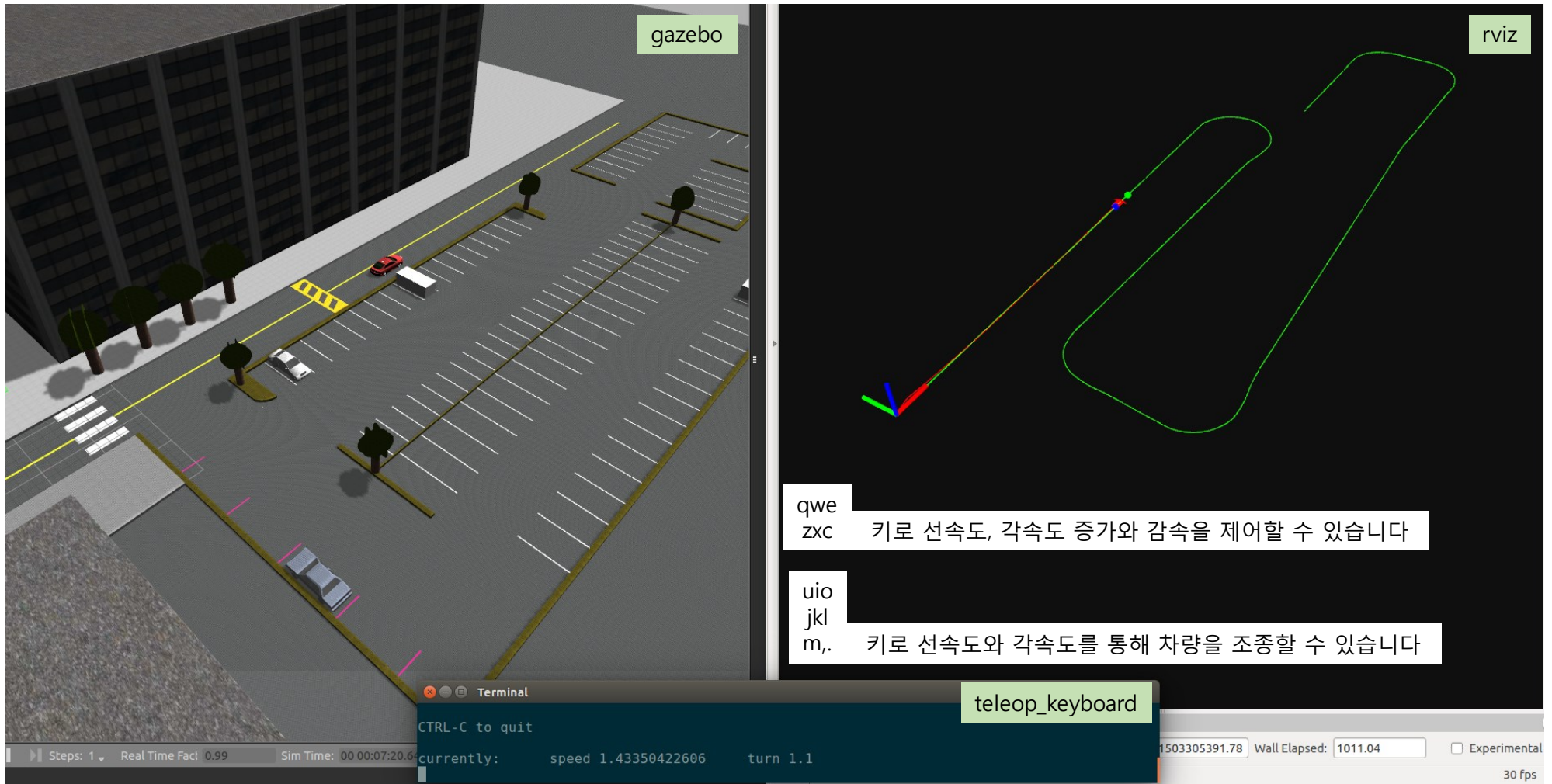
Simulator TEST #1 (motion_planner)

6. motion_planner에서 forward 라디오버튼을 클릭하고 JW Path 버튼을 클릭합니다



Simulator TEST #1 (motion_planner)

7. dyros_teleop_keyboard 터미널창을 클릭하고 i 버튼을 클릭합니다.

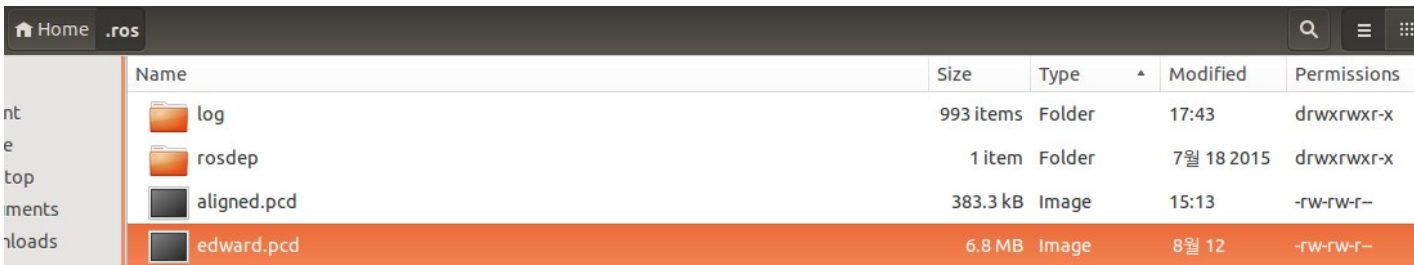


Simulator TEST #2 (LOAM Localization)

1. dyros_simulator를 실행합니다

```
$ roslaunch dyros_simulator dyros_simulator.launch
```

2. edward.pcd(추후 이름 수정 예정)파일을 ~/.ros/ 폴더에 넣습니다



The screenshot shows a file manager window with the title bar 'Home .ros'. The left sidebar has a vertical list of items: 'nt', 'e', 'top', 'ments', and 'hloads'. The main area displays a table of files and folders. The 'edward.pcd' file is highlighted in orange.

Name	Size	Type	Modified	Permissions
log	993 items	Folder	17:43	drwxrwxr-x
rosdep	1 item	Folder	7월 18 2015	drwxrwxr-x
aligned.pcd	383.3 kB	Image	15:13	-rw-rw-r--
edward.pcd	6.8 MB	Image	8월 12	-rw-rw-r--

Simulator TEST #2 (LOAM Localization)

3. gazebo 상에서 T 버튼을 누른 후 차량을 클릭해 임의의 위치에 올려놓습니다

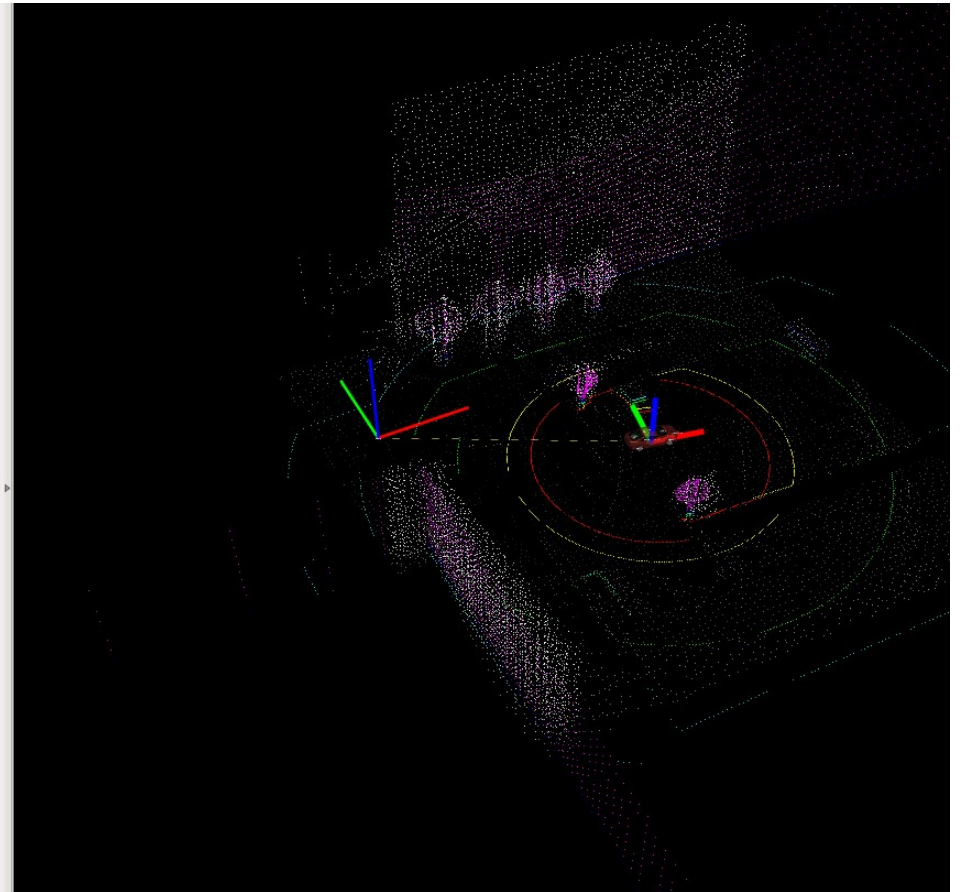
T 버튼 입력 후 상태



Simulator TEST #2 (LOAM Localization)

4. LOAM(modified ver)을 실행합니다

```
$ roslaunch loam_velodyne loam_velodyne_ed.launch
```



코드 수정 부분 : loam_velodyne

기존 코드와 성능을 비교하기 위해 코드를 복사해서 독립적으로 수정했습니다.

scanRegistration.cpp → scanRegistration_ed.cpp

laserOdometry.cpp → laserOdometry_ed.cpp

laserMapping.cpp → laserMapping_ed.cpp

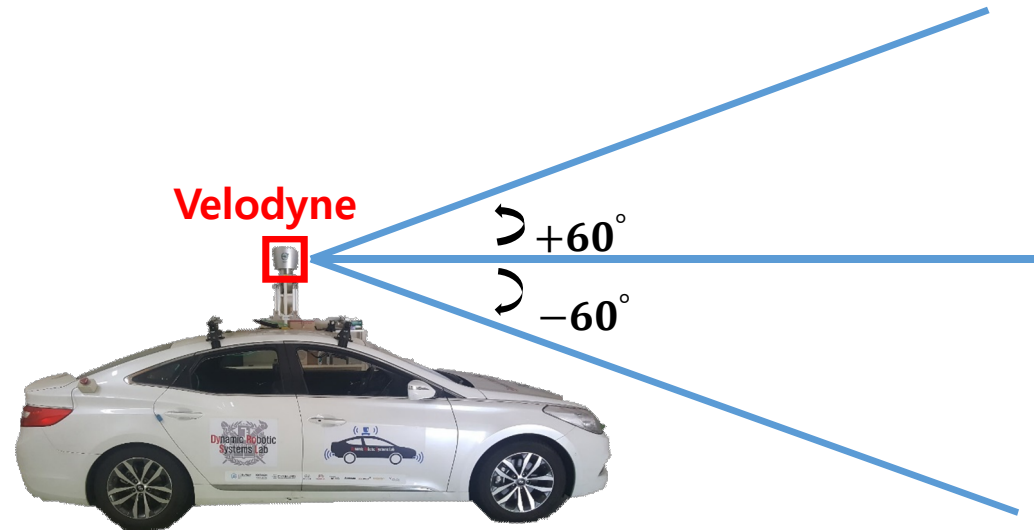
transformMaintenance.cpp → transformMaintenance_ed.cpp

poseInitializer.cpp → poseInitializer_ed.cpp

코드 수정 부분 : loam_velodyne

scanRegistration.cpp → scanRegistration_ed.cpp, 255번줄

```
// ed: 0 ~ -30 degree를 -60 ~ +60 degree로 수정했다
if(angle >= -60 && angle < -29 ) { scanID = 0 ;}
else if(angle >= -28 && angle < -27 ) { scanID = 1 ;}
else if(angle >= -26 && angle < -25 ) { scanID = 2 ;}
else if(angle >= -24 && angle < -23 ) { scanID = 3 ;}
else if(angle >= -22 && angle < -21 ) { scanID = 4 ;}
else if(angle >= -20 && angle < -19 ) { scanID = 5 ;}
else if(angle >= -18 && angle < -17 ) { scanID = 6 ;}
else if(angle >= -16 && angle < -15 ) { scanID = 7 ;}
else if(angle >= -14 && angle < -13 ) { scanID = 8 ;}
else if(angle >= -12 && angle < -11 ) { scanID = 9 ;}
else if(angle >= -10 && angle < -8.5 ) { scanID = 10 ;}
else if(angle >= -8 && angle < -7 ) { scanID = 11 ;}
else if(angle >= -6 && angle < -4.5 ) { scanID = 12 ;}
else if(angle >= -4 && angle < -2.5 ) { scanID = 13 ;}
else if(angle >= -2 && angle < 0 ) { scanID = 14 ;}
else if(angle >= 0 && angle < 60 ) { scanID = 15 ;}
else
continue;
```



Velodyne 데이터를 받는 각도를 0~-30 에서 +60 ~ -60으로 수정

코드 수정 부분 : loam_velodyne

laserOdometry.cpp → laserOdometry_ed.cpp

LOAM을 처음 켜면 (0,0,0)이 되는 좌표계

차량좌표계

기존의 `/camera_init` ↔ `/camera`의 좌표관계를

dyros_simulator 프로그램과 연동하기 위해

`/camera_init` ↔ `/dyros/base_footprint`의 좌표관계로 수정했습니다.

코드 상에서 dyros로 검색하면 수정된 부분이 나옵니다.

코드 수정 부분 : loam_velodyne

laserMapping.cpp → laserMapping_ed.cpp, 349번줄

```
// ed: rostopic pub -1 /loam_map_save std_msgs/String "edward2.pcd" 를 통해 맵파일을 저장하는 함수
void mapSaveHandler(const std_msgs::String::ConstPtr& str){
    ROS_INFO("Saving the current map... to %s", str->data.c_str());

    int cubeI = laserCloudCenWidth;
    int cubeJ = laserCloudCenHeight;
    int cubeK = laserCloudCenDepth;
    int cubeInd = cubeI + laserCloudWidth * cubeJ + laserCloudWidth * laserCloudHeight * cubeK;

    cout << "cubeI : " << cubeI << ", cubeJ : " << cubeJ << ", cubeK : " << cubeK << ", " << cubeInd << endl;
    // ed: 코드 추가
    // 모든 포인트 클라우드 데이터 전체를 저장하기 위해 아래 코드를 추가한다
    pcl::PointCloud<pcl::PointXYZ>::Ptr laserCloudSum_ed(new pcl::PointCloud<pcl::PointXYZ>());
    for (int i = 0 ; i < laserCloudNum ; i++){
        //cout << i << endl;
        // *laserCloudSum_ed += *velo_points_array[i];
        *laserCloudSum_ed += *laserCloudSurfArray[i] + *laserCloudCornerArray[i];
    }

    // ed: 코드 수정
    //pcl::io::savePCDFileASCII(str->data, *laserCloudSurfArray[cubeInd] + *laserCloudCornerArray[cubeInd]);
    pcl::io::savePCDFileASCII(str->data, *laserCloudSum_ed);
}
```

기존의 제한된 PointCloud만 저장하는 코드에서 모든 누적 PointCloud를 맵으로 저장하는 코드로 수정했습니다

코드 수정 부분 : loam_velodyne

transformMaintenance.cpp → transformMaintenance_ed.cpp, 183번줄

```
// ed: roll, pitch를 없애기 위해 아래처럼 설정한다.  
//laserOdometryTrans2.setRotation(tf::Quaternion(geoQuat.x, geoQuat.y, geoQuat.z, geoQuat.w));  
laserOdometryTrans2.setRotation(tf::Quaternion(0, 0, geoQuat.z, geoQuat.w));  
laserOdometryTrans2.setOrigin(tf::Vector3(transformMapped[3], transformMapped[4], transformMapped[5]));  
  
// ed: /dyros/base_footprint tf를 broadcast한다  
//tfBroadcaster2Pointer->sendTransform(laserOdometryTrans2);
```

기존의 **/camera_init ↔ /dyros/base_footprint**의 좌표관계를 퍼블리시할 때 roll, pitch 값 때문에 좌표계가 땅으로 파고 들어가는 문제가 생겨서 평평한 곳에서 실험한다고 가정하고 roll, pitch 값을 0으로 설정했습니다.

하지만

poseInitialize_ed에서도 이미 **/camera_init ↔ /dyros/base_footprint** 좌표관계를 퍼블리시해주고 있으므로 굳이 transformMaintenance_ed 파일에서도 퍼블리시해서 두 개가 겹칠 필요가 없다고 생각되어 해당 좌표관계를 주석처리했습니다. (결론적으로 사용안함)

코드 수정 부분 : loam_velodyne

poseInitializer.cpp → poseInitializer_ed.cpp, 57번줄

```
// ed: gps 데이터를 받는 섭스크라이버 추가
sub_gps = nh_.subscribe<geometry_msgs::Vector3Stamped>("/dyros/gps/utm", 1, &PoseInitializer_ed::gps_callback, this);
sub_gps_heading = nh_.subscribe<std_msgs::Float64>("/dyros/gps/heading", 1, &PoseInitializer_ed::gps_heading_callback, this);
// ed: 실제위치와 예측위치를 비교하기 위해 섭스크라이버 추가
sub_local = nh_.subscribe<std_msgs::Float32MultiArray>("/LocalizationData", 1, &PoseInitializer_ed::local_callback, this);
```

**gps 데이터와 Localization 데이터를 사용해
Map을 Crop해야하므로 위의 섭스크라이버들 추가했습니다**

코드 수정 부분 : loam_velodyne

poseInitializer.cpp ➔ poseInitializer_ed.cpp, 332번줄

위 섹스크라이버들의 콜백함수입니다.

```
// ed: 디버깅용 LocalizationData 데이터를 섹스크라이브하는 콜백함수 추가
void local_callback(const std_msgs::Float32MultiArray::ConstPtr& msg) {
    if(justOnce2){
        // ed: 현재 motion_planner와 연동하기 위해 (x,y) ==> (-y,x)로 좌표축이 틀어져있으므로 이를 반영해준다
        real_pnt[0] = msg->data[0];
        real_pnt[1] = msg->data[1];
        justOnce2 = false;}
}

// ed: /dyros/gps/utm 데이터를 섹스크라이브하는 콜백함수 추가
void gps_callback(const geometry_msgs::Vector3Stamped::ConstPtr& msg) {
    // ed: x : 500000
    // y : 4982950
    translate_pt(0) = msg->vector.x - 500000;
    translate_pt(1) = msg->vector.y - 4982950;
}

// ed: /dyros/gps/heading 데이터를 섹스크라이브하는 콜백함수 추가
void gps_heading_callback(const std_msgs::Float64::ConstPtr& msg) {
    double deg2rad = 0.0174; // ed: 3.14 / 180
    double filtered_yaw = msg->data;
    // ed: /dyros/gps/heading의 컨벤션이 dyros 차량의 각도 컨벤션과 다르므로 아래처럼 데이터처리를 해줘야 한다
    if(msg->data > 0 && msg->data < 180){
        filtered_yaw -= 180;
    }
    else if(msg->data > 180 && msg->data < 360){
        filtered_yaw -= 540;
    }
    // ed: cropbox.setRotation 함수에 맞게 사용하기 위해 90을 더해줘야 한다
    filtered_yaw += 90;
    rotation_yaw(0) = 0;
    rotation_yaw(1) = 0;
    rotation_yaw(2) = filtered_yaw * deg2rad; // yaw}
```

코드 수정 부분 : loam_velodyne

poseInitializer.cpp → poseInitializer_ed.cpp, 448번줄

```
// ed: 아래 코드 추가. z방향을 0으로 설정 안하면 /camera_init과 /dyros/base_footprint가 z방향  
1의 차이를 갖기 때문에 0으로 설정한다
```

```
gh.setOrigin(tf::Vector3(gh.getOrigin().m_floats[0] ,  
                          gh.getOrigin().m_floats[1] ,  
                          0));
```

```
// ed: 차량의 Pitch, Roll 방향으로 좌표계가 기울이지 않도록 x,y축 회전량을 0으로 설정한다
```

```
gh.setRotation(tf::Quaternion(0,0,  
                               gh.getRotation().getZ(),  
                               gh.getRotation().getW()));
```

/dyros/base_footprint 좌표계가 roll, pitch 값을 가지지 않도록
코드를 위처럼 0으로 수정했습니다 (임시방편)

코드 수정 부분 : loam_velodyne

poseInitializer.cpp → poseInitializer_ed.cpp, 491번줄

```
// ed: 코드를 원래대로 수정했다  
// pose2DMsg_.x = -odomMsg_.pose.pose.position.y;  
// pose2DMsg_.y = odomMsg_.pose.pose.position.x;  
pose2DMsg_.x = odomMsg_.pose.pose.position.x;  
pose2DMsg_.y = odomMsg_.pose.pose.position.y;
```

/my_pose로 퍼블리시하는 pose2DMsg_가 (-y,x)였던 코드를 정상좌표계 (x,y)로 수정했습니다.

코드 수정 부분 : loam_velodyne

poseInitializer.cpp → poseInitializer_ed.cpp, 163번줄 match() 함수 내부

```
// ed: 코드 추가
// Set the max correspondence distance to 5cm (e.g., correspondences with higher distances will be ignored)
gicp.setMaxCorrespondenceDistance (500); 최대 일치가능한 거리 (m)
// Set the maximum number of iterations (criterion 1)
gicp.setMaximumIterations (10000); 한 루프당 최대 반복수
//Set the transformation epsilon (maximum allowable difference between two consecutive transformations) in order
for an optimization to be considered as having converged to the final solution.
gicp.setTransformationEpsilon (1e-10); 두 물체 간 변환행렬의 정밀도를 설정하는 파라미터인듯 합니다.
// Set the maximum allowed Euclidean error between two consecutive steps in the ICP loop, before the algorithm is
considered to have converged.
// The error is estimated as the sum of the differences between correspondences in an Euclidean sense, divided by
the number of correspondences.
gicp.setEuclideanFitnessEpsilon (0.5); 한 번의 ICP 루프당 최대허용가능한 Euclidean 거리 에러 값으로 작을 수록 좋다고 합니다.
```

GICP의 여러 파라미터들을 수정할 수 있어서 위처럼 코드를 추가했습니다

코드 수정 부분 : loam_velodyne

poseInitializer.cpp → poseInitializer_ed.cpp, 163번줄 match() 함수 내부

```
// ed: CropBox 코드 추가. PointCloud를 원하는 영역만큼 잘라서 사용할 수 있다
pcl::CropBox<pcl::PointXYZ> cropBoxFilter_source (true);
pcl::CropBox<pcl::PointXYZ> cropBoxFilter_target (true);
cropBoxFilter_source.setInputCloud (previousMap_);
cropBoxFilter_target.setInputCloud (currentMapFiltered_);
// ed: 자를 영역을 설정하는 변수들
Eigen::Vector4f min_pt (-50.0f, -50.0f, -50.0f, 0.0f);
Eigen::Vector4f max_pt (50.0f, 50.0f, 50.0f, 0.0f);
```

**Global Map과 현재 Velodyne 데이터를 Crop할 객체를 각각 생성하고
crop할 영역을 min_pt, max_pt로 설정합니다.**

코드 수정 부분 : loam_velodyne

poseInitializer.cpp → poseInitializer_ed.cpp, 163번줄 match() 함수 내부

```
// ed: 예측의 정확도가 높아질 때까지 무한루프를 돌아서 맞춘다
while(true) {
    min_pt(0) += .5f;
    min_pt(1) += .5f;
    min_pt(2) += .5f;
    max_pt(0) -= .5f;
    max_pt(1) -= .5f;
    max_pt(2) -= .5f;
    //translate_pt[0] += 0.5;
    //translate_pt[1] += 0.5;
    //rotation_yaw(2) += 0.1;
    // Cropbox slightly bigger than bounding box of points
    cropBoxFilter_source.setMin (min_pt);
    cropBoxFilter_source.setMax (max_pt);
    cropBoxFilter_target.setMin (min_pt);
    cropBoxFilter_target.setMax (max_pt);
    // ed: GPS의 데이터를 사용해 특정지역에서 Crop하기 위해 아래 코드를 추가한다
    cropBoxFilter_source.setTranslation (translate_pt);
    cropBoxFilter_target.setRotation (rotation_yaw);
    //cout << "rotation : " << rotation_yaw(2) << endl;
    // ed : 위의 제약조건에 의해 필터링된 포인트클라우드를 생성한다
    cropBoxFilter_source.filter (cloud_out_source);
    cropBoxFilter_target.filter (cloud_out_target);
    cloud_out_ptr_source = cloud_out_source.makeShared();
    cloud_out_ptr_target = cloud_out_target.makeShared();
    .....
}
```

.....
...} **길어서생략**

소스와 타겟간의 중심점들간의
Euclidean 거리로 값이 작을수록
좋습니다.

무한루프를 돌면서 getFitnessScore() 값이
1.5 이하로 떨어질 때까지
Crop 영역을 점점 작게하면서
GICP 알고리즘을 수행합니다.

여기서 translate_pt, rotation_yaw 값으로
GPS 데이터가 사용됩니다.

코드 수정 부분 : loam_velodyne

poseInitializer.cpp → poseInitializer_ed.cpp, 163번줄 match() 함수 내부

```
// ed: 코드 추가, gps heading값을 통해 GICP를 수행하였으므로 initTf_ = gicp.getFinalTransformation()에는  
rotation_yaw각도 만큼의 회전성분이 안 들어가 있다.  
// 따라서 추가해줘야한다  
rotation_matrix = Eigen::AngleAxisf(0, Eigen::Vector3f::UnitX())  
                  * Eigen::AngleAxisf(0, Eigen::Vector3f::UnitY())  
                  * Eigen::AngleAxisf(-rotation_yaw(2), Eigen::Vector3f::UnitZ());  
  
initTf_.block<3,3>(0,0) *= rotation_matrix;
```

getFitnessScore() 값이 1.5 이하로 떨어져서 매칭이 완료된 다음에

변환행렬을 구하게 되는데 이 때 **GPS Heading 값을 통해**

임의로 Current Velodyne PointCloud를 돌려서 맞췄으므로

해당 yaw 각만큼의 rotation_matrix를 만들어 추가해줘야합니다.

(직관적이지 않은 설명이지만 해당 코드를 빼고 해보면 바로 알 수 있습니다.)

코드 수정 부분 : loam_velodyne

launch 파일을 수정된 버전용으로 따로 만들었습니다.

loam_velodyne_ed.launch : 위의 수정된 ..._ed 코드를 실행하는 launch 파일

loam_velodyne_for_slam_ed.launch : Global Map을 만들기 위해 SLAM을 수행하는 launch 파일



Global Map을 만드는 방법은

위 launch 파일을 실행시킨 후 차량으로 원하는 지역을 왕복이동하고
터미널창에서

```
$ rostopic pub -1 /loam_map_save std_msgs/String "map_name.pcd"
```

를 타이핑하면 ~/.ros/ 에 파일이 저장됩니다.

코드 수정 부분 : dbw_mkz_twist_controller

TwistControllerNode.cpp, 60번줄

```
// ed: 휠베이스, 트랙길이, 스티어링비를 설정하는 코드인듯
// Ackermann steering parameters
ackер_wheelbase_ = 2.780; // for grandeur
ackер_track_ = 1.58; // for grandeur
//steering_ratio_ = 14.8;
steering_ratio_ = 18.6; // for grandeur
```

현재 Grandeur 차량에 맞는 wheelbase, track, steering_ratio 값으로 수정했습니다.

코드 수정 부분 : dbw_mkz_twist_controller

TwistControllerNode.cpp, 85번줄

```
// ed: /LocalizationData 토픽 데이터를 저장해서 웨이포인트를 만드는 섭스크라이버 추가
sub_save_localization_data = n.subscribe("waypoint_save", 1, &TwistControllerNode::saveWaypoint, this);

// ed: motion_planner와 연동하기 위한 퍼블리셔, 섭스크라이버 추가
sub_motion_planner = n.subscribe("SteerAngleData", 1, &TwistControllerNode::SteeringAngle_callback, this);
sub_gazebo_model_states = n.subscribe("/gazebo/model_states", 1, &TwistControllerNode::Gazebo_modelStates_callback, this);

pub_localization = n.advertise<std_msgs::Float32MultiArray>("LocalizationData", 1);
pub_gear = n.advertise<dbw_mkz_msgs::GearCmd>("gear_cmd", 1);
```

**waypoint를 저장하는 섭스크라이버와
motion_planner와 연동하기 위한 여러 퍼블리셔, 섭스크라이버를 추가했습니다**

(각각 콜백함수는 너무 길어서 여기서는 생략했고 실제 코드에 자세하게 주석을 첨부했습니다)

코드 수정 부분 : dbw_mkz_twist_controller

waypoint를 저장하는 방법

10Hz, 1초에 10번 기록합니다.

dyros_simulator가 켜져있는 상태에서 다른 터미널창을 켜 다음

\$ rostopic pub -r 10 /dyros/waypoint_save std_msgs/String "mapname.map"

를 입력하면 그 순간부터 차량을 움직이는 모든 차량의 (x,y,heading,velocity) 값이 csv 파일로 저장됩니다.

종료는 Ctrl+C를 누르시면 파일 저장이 완료되고 경로는 ~/.ros/ 에 저장됩니다.



This document was created with the Win2PDF "print to PDF" printer available at
<http://www.win2pdf.com>

This version of Win2PDF 10 is for evaluation and non-commercial use only.

This page will not be added after purchasing Win2PDF.

<http://www.win2pdf.com/purchase/>