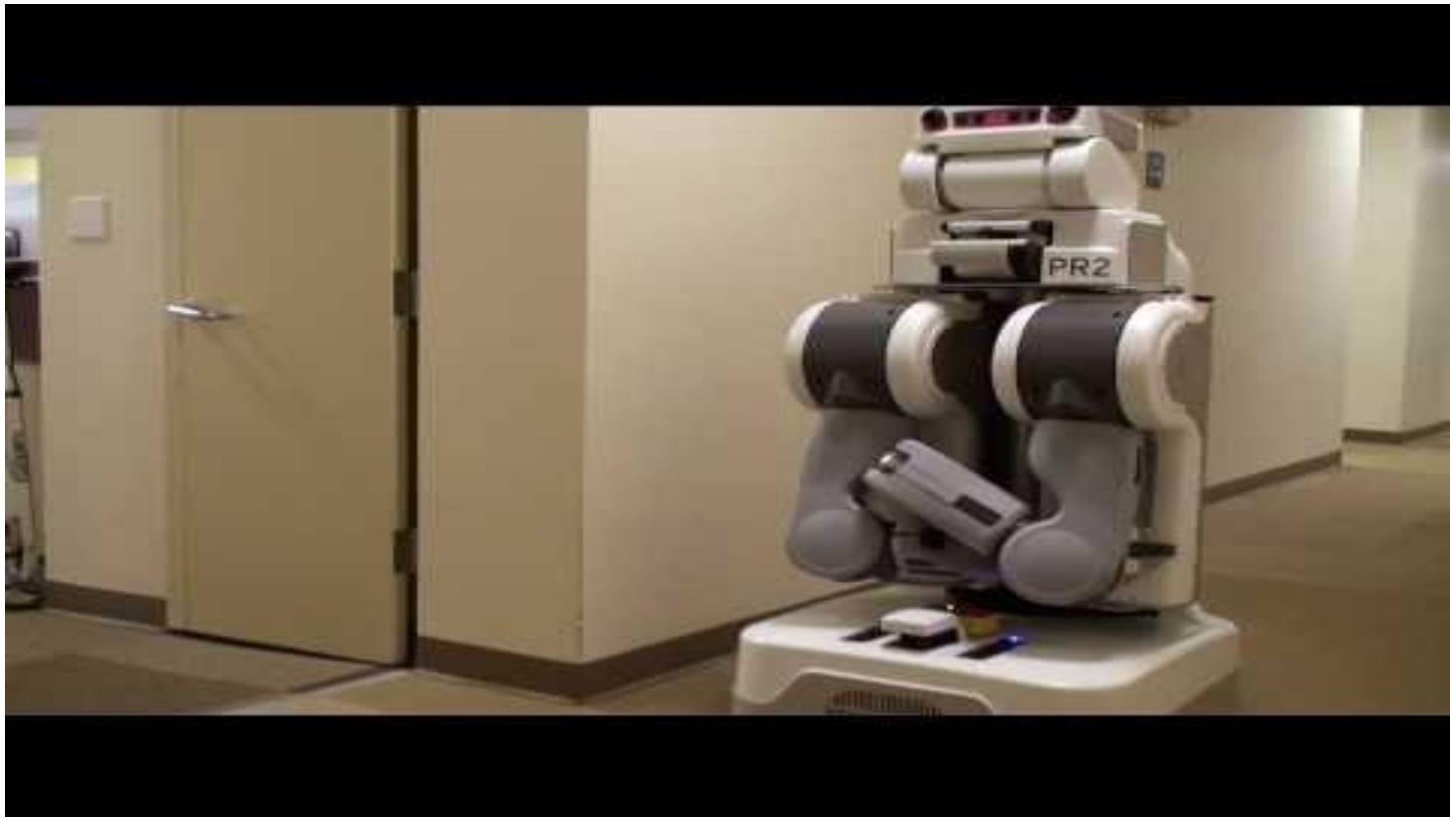
11주차

Youngjoon Han
young@ssu.ac.kr

Contents

- Navigation Stack
  1. 개요
  2. 맵 작성
  3. Localization
  4. Navigation

Navigation Stack – 1. 개요



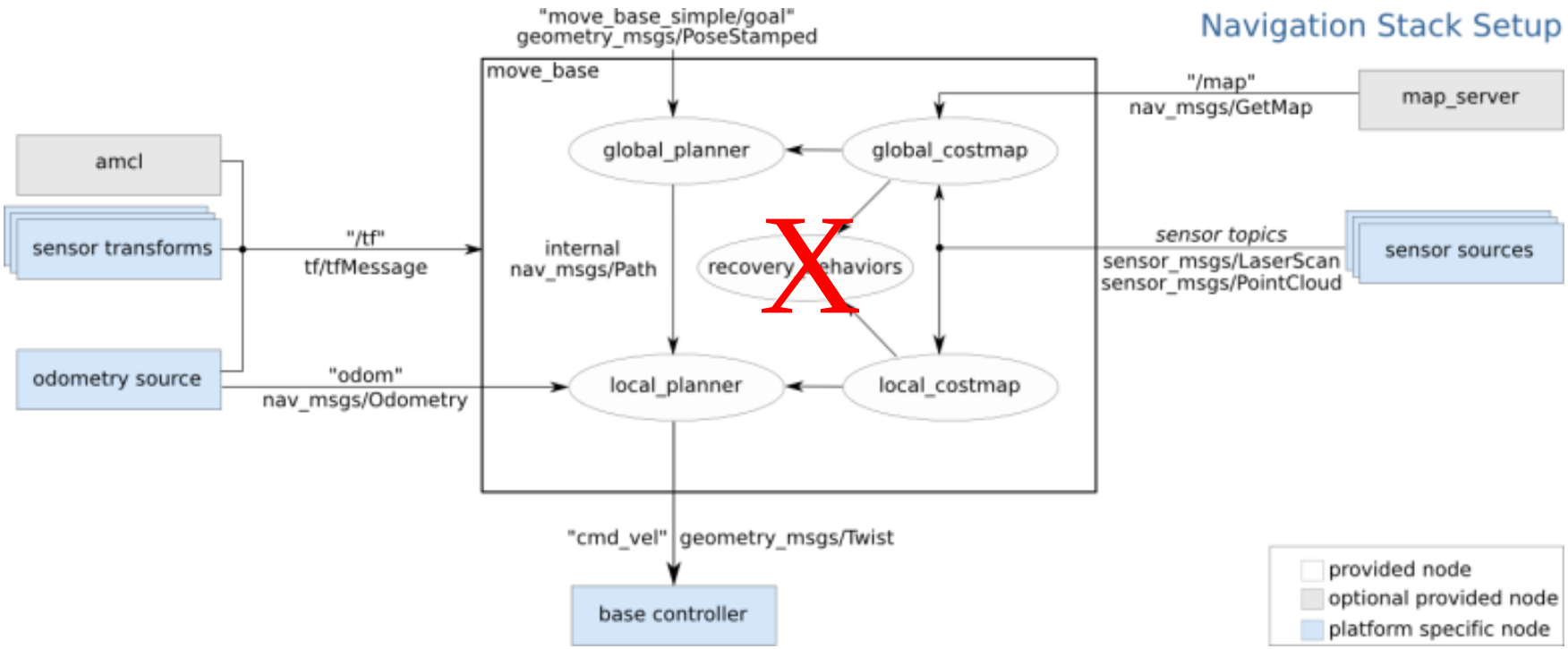
https://youtu.be/qziUJcUDfBc

Navigation Stack – 1. 개요

- Navigation Stack
  - 자율주행을 위한 오픈소스들의 추상화
  - SLAM/Localization/Navigation 오픈 소스를 제공하며, 기본적인 자율주행 소프트웨어 스택을 제공한다.
  - ROS 코드 형태로 배포되며, 임의의 로봇에 적용될 수**는** 있다.
- Navigation Stack을 사용하기 위해
  - 센서와 오도메트리 정보를 input으로 받으며, cmd_vel 토픽을 output으로 내보낸다.
  - 따라서 사용자는 tf/Odometry/Laser 데이터를 세팅해야 한다.
  - input/output만 맞춰 준다면 plug-in 방식으로 기존 소스를 다른 소스로 대체할 수 있다.
    - ex)2륜 로봇 path planner를 car-like 로봇 path planner로 대체할 수 있다.

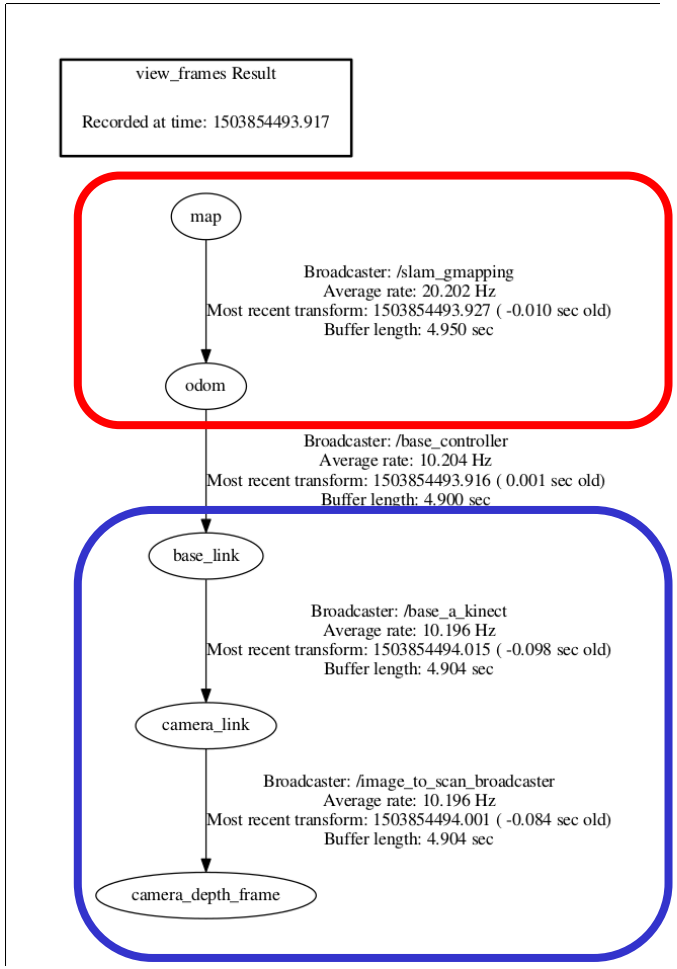
Navigation Stack – 1. 개요

- http://wiki.ros.org/navigation/Tutorials/RobotSetup



Navigation Stack – 1. 개요

- simple Navigation Stack TF tree example

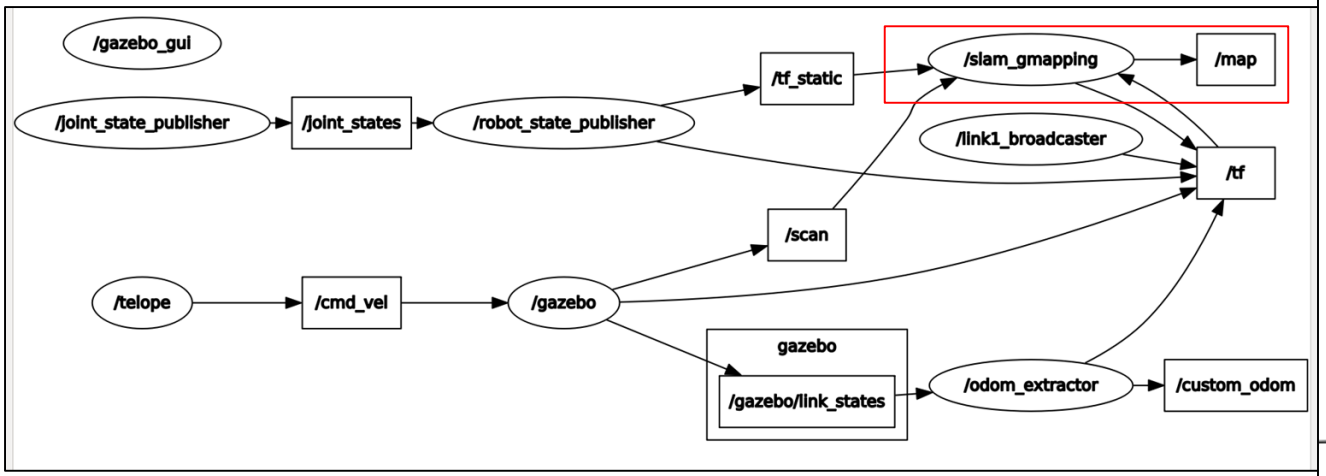


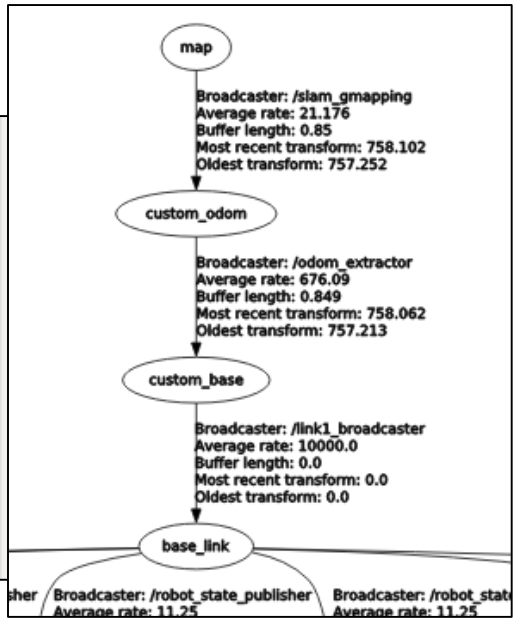
Navigation Stack – 1. 개요

- Navigation 관련 용어
  - SLAM(Simultaneous Localization And Mapping) : 임의 공간에서 이동하면서 주변을 탐색할 수 있는 로봇에 대해 그 공간의 지도 및 현재 위치를 추정하는 문제
    - 참고 : https://www.youtube.com/watch?v=khSrWtB0Xik&t=156s
    - Navigation Stack에서는 2d SLAM을 위해 Gmapping 노드를 제공한다.
  - Navigation : 항공기나 선박, 차량 등을 한 장소에서 다른 장소로 이동시키기 위한 방법 또는 그러한 기술을 가리킨다.
    - Navigation Stack에서는 navigation 문제를 global/local로 나누어 해결한다. 이를 위해 global/local costmap이 필요하다.

Navigation Stack – 2. 맵 작성

- Navigation Stack에서는 로봇을 주행시키기 위해 현재 로봇 주변의 맵을 이용한다.
- 따라서 로봇 주변의 맵을 로봇이 이해할 수 있는 형태로 입력해야 한다.
- 이를 위해 SLAM알고리즘을 이용하며, Navigation Stack에서는 Gmapping 노드를 제공한다.
  (아래 사진은 예시용. 이번 강의와는 관련 X)





Navigation Stack – 2. 맵 작성

- 로봇 주변 지도 작성: 로봇 하드웨어/오도메트리 구성□ gmapping 노드실행 □맵 작성 후 map_server패키지의 map_saver 노드로 맵을 pgm과 yaml파일로 저장
- 처리를 위해 slam_gmapping 노드는 scan□base_link, base_link□odom tf가 필요함.

> 4.1.5 Required tf Transforms
>
> `<the frame attached to incoming scans>` → `base_link`
>
> usually a fixed value, broadcast periodically by a robot_state_publisher, or a `tf` static_transform_publisher.
>
> `base_link` → `odom`
>
> usually provided by the odometry system (e.g., the driver for the mobile base)
>
> 4.1.6 Provided tf Transforms
>
> `map` → `odom`
>
> the current estimate of the robot's pose within the map frame

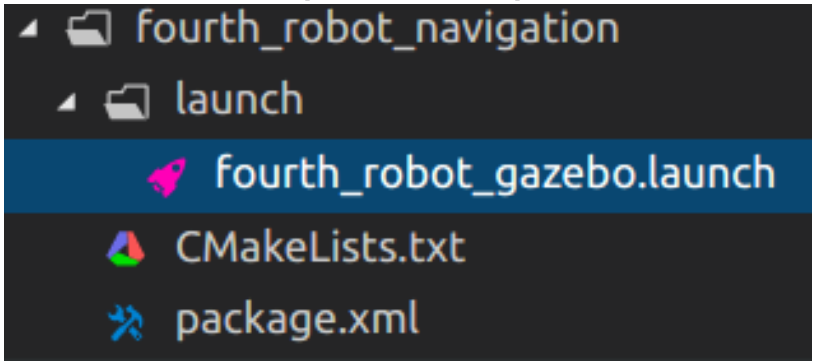
- 만들어진 맵은 이후 map_server패키지의 map_server노드로 불러올 수 있다.

Navigation Stack – 2. 맵 작성

- 오픈소스 다운로드

```
## https://github.com/CIR-KIT/fourth_robot_pkg.git
```

- 네비게이션 스택 실습을 위한 패키지를 작성한다.
- fourth_robot_navigation 패키지 작성 후 다음 슬라이드의 런치 파일 작성
- fourth_robot 오픈소스는 수정 금지



Navigation Stack – 2. 맵 작성

fourth_robot_gazebo.launch | playpen_map.yaml | fourth_robot_mapping.launch

```
<launch>

  <!-- these are the arguments you can pass this launch file, for example paused:=true -->
  <arg name="model" default="$(find fourth_robot_description)/robots/fourth_robot.urdf.xacro"/>
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="gui" default="true"/>
  <arg name="headless" default="false"/>
  <arg name="debug" default="false"/>

  <!-- We resume the logic in empty_world.launch, changing only the name of the world to be launched -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find fourth_robot_gazebo)/worlds/clearpath_playpen.world"/>
    <arg name="debug" value="$(arg debug)" />
    <arg name="gui" value="$(arg gui)" />
    <arg name="paused" value="$(arg paused)"/>
    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
    <arg name="headless" value="$(arg headless)"/>
  </include>

  <!-- lrf merger -->
  <include file="$(find fourth_robot_bringup)/launch/sensors/lrf_merger.launch"/>

  <!-- Load the URDF into the ROS Parameter Server -->
  <param name="robot_description"
         command="$(find xacro)/xacro.py '$(arg model)'" />

  <!-- Run a python script to the send a service call to gazebo_ros to spawn a URDF robot -->
  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model" respawn="false" output="screen"
        args="-urdf -model fourth_robot -param robot_description"/>

  <!-- ros_control motoman launch file -->
  <include file="$(find fourth_robot_control)/launch/fourth_robot_control.launch"/>
</launch>
```

0 0 | Ln 33, Col 84 | Spaces: 4 | UTF-8 | LF | XML

Navigation Stack – 2. 맵 작성

- 런치파일 실행(아래 사진 에러는 무시)

```
## https://github.com/CIR-KIT/fourth_robot_pkg.git
```

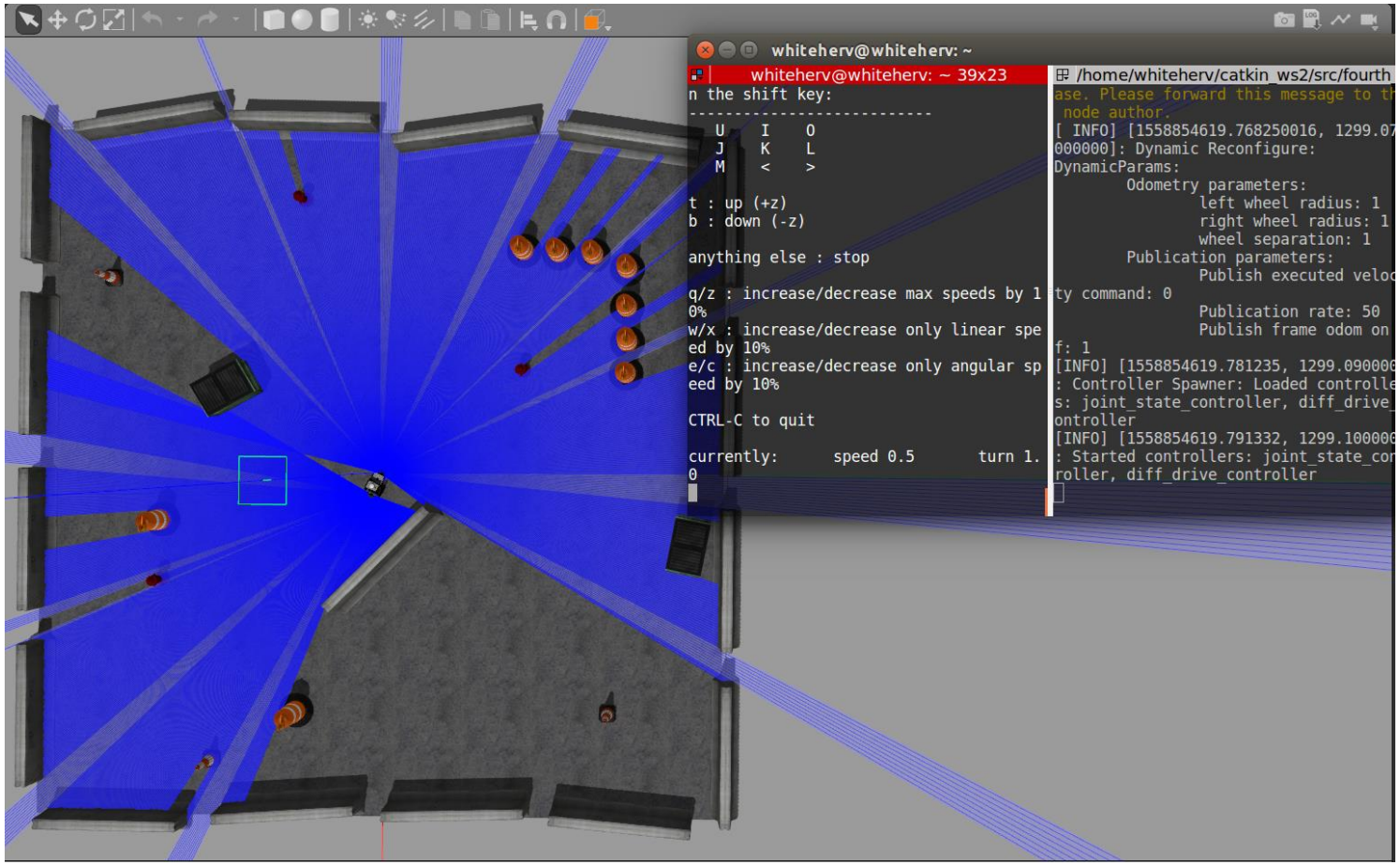
```
process[gazebo-2]: started with pid [24881]
process[gazebo_gui-3]: started with pid [24886]
ERROR: cannot launch node of type [ira_laser_tools/laserscan_multi_merger]: can't locate n
ode [laserscan_multi_merger] in package [ira_laser_tools]
process[urdf_spawner-5]: started with pid [24891]
process[controller_spawner-6]: started with pid [24892]
process[robot_state_publisher-7]: started with pid [24893]
```

- 로봇 조종 노드 실행(커맨드 중 :=은 해당 노드가 publish하거나 subscribe하는 토픽의 이름을 바꾸는 명령어이다. https://answers.ros.org/question/9248/how-do-you-remap-a-topic/ 참고)

```
## rosrun teleop_twist_keyboard teleop_twist_keyboard.py
/cmd_vel:=/diff_drive_controller/cmd_vel
```

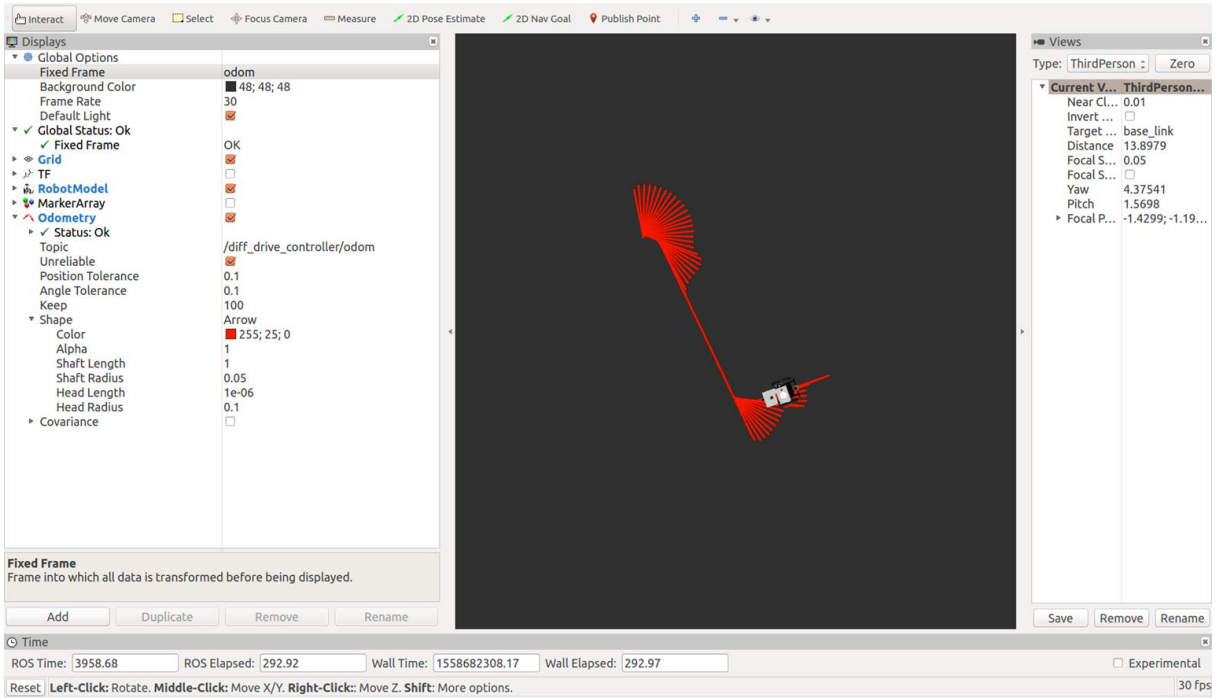
Navigation Stack – 2. 맵 작성

- 로봇 조종 확인



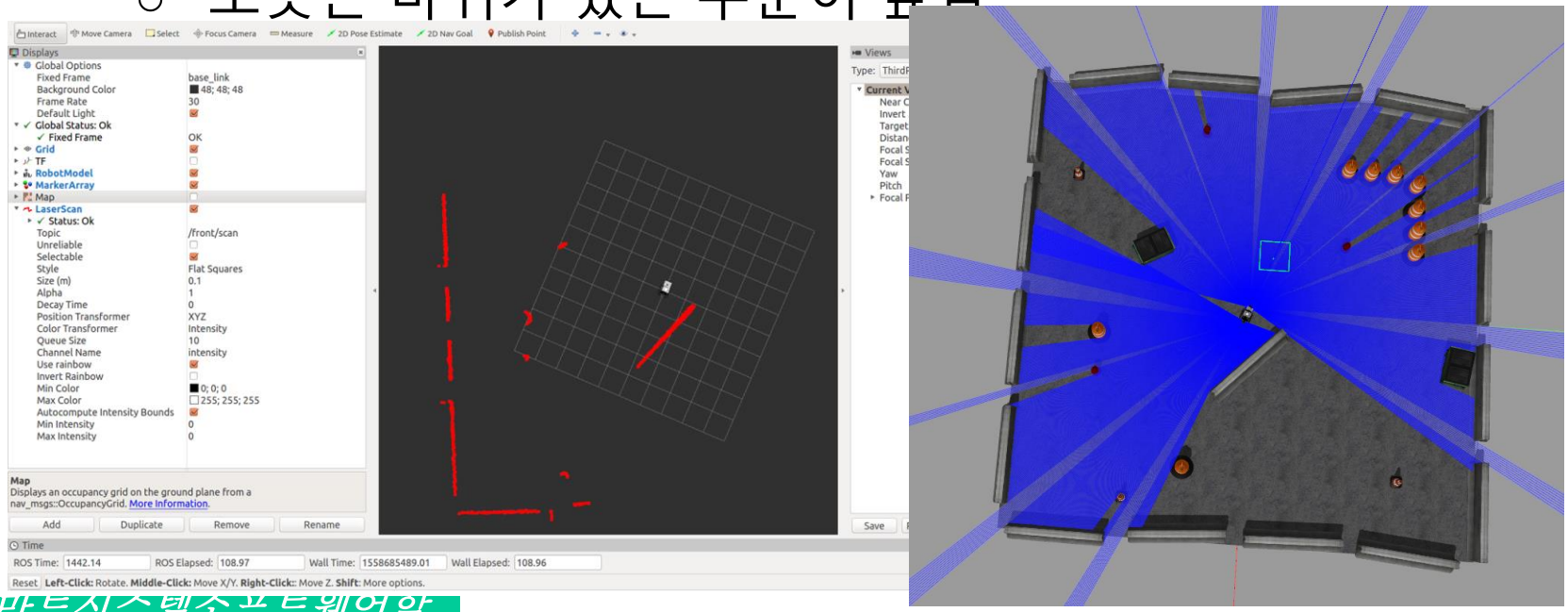
Navigation Stack – 2. 맵 작성

- 실습을 진행하기 전 odometry, laser데이터, tf를 확인한다.
- mobile_robot의 odometry 과제 영상과 같이 세팅한다. fourth robot은 /diff_drive_controller/odom 명으로 odometry가 publish 된다.



Navigation Stack – 2. 맵 작성

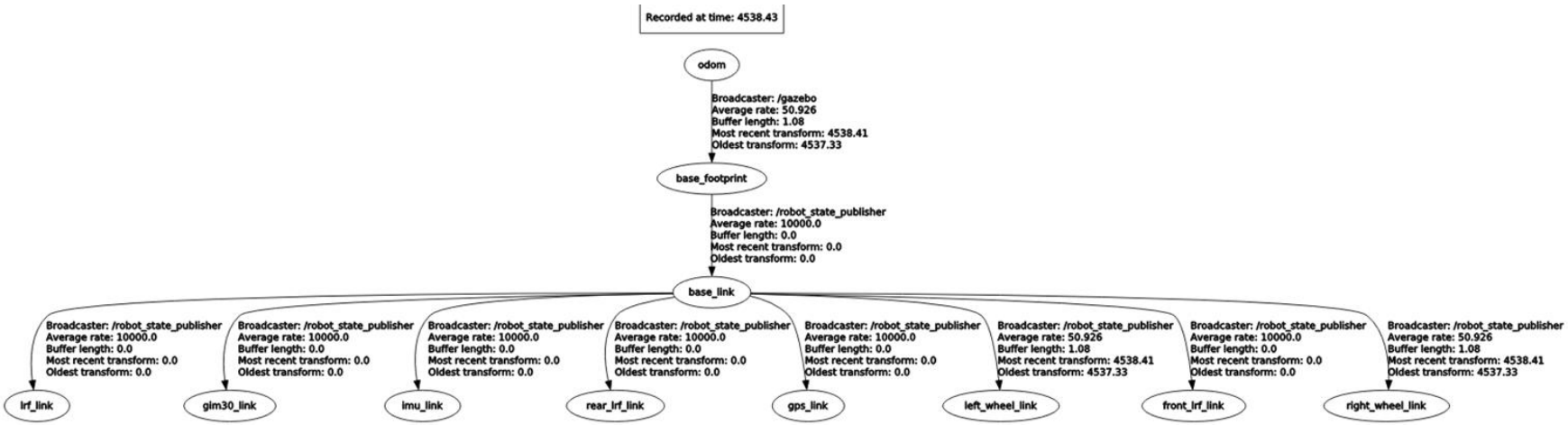
- 실습을 진행하기 전 odometry, laser데이터, tf를 확인한다.
- rviz에서
  - Global Options에서 fixed frame은 base_link
  - LaserScan 추가 후 /front/scan으로 세팅, size는 0.1로 세팅
  - 로봇은 바퀴가 있는 부분이 앞임



Navigation Stack – 2. 맵 작성

- 실습을 진행하기 전 odometry, laser데이터, tf를 확인한다.

```
## rosrun rqt_tf_tree rqt_tf_tree
```



Navigation Stack – 2. 맵 작성

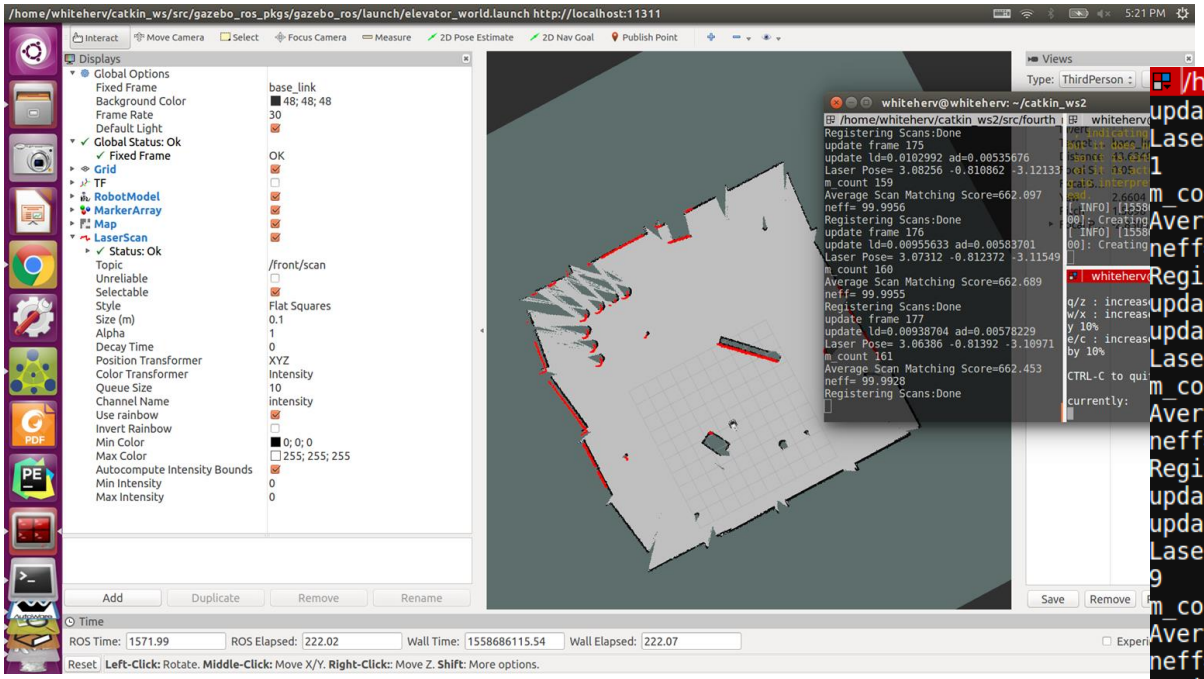
- fourth_robot_gazebo.launch에 gmapping 노드를 **추가**로 실행시키는 fourth_robot_mapping.launch를 작성한다.아래 링크 참고 -> **fourth_robot에 맞게 수정 필요!**
  https://github.com/ros-perception/slam_gmapping/blob/hydro-devel/gmapping/launch/slam_gmapping_pr2.launch
  - 로봇의 scan 토픽은 /front/scan 이다. 이를 /scan으로 remapping시키기 위해 아래와 참고해 런치파일을 수정한다.

```
<node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" output="screen">
  <remap from="scan" to="/front/scan"/>
```

  - 로봇의 base_frame, odom_frame에 유의
  - gmapping 파라미터 중 maxUrange의 값을 50으로 수정
- gmapping에서 사용되는 모든 파라미터는 아래 페이지를 참고한다 : http://wiki.ros.org/gmapping

Navigation Stack – 2. 맵 작성

- rviz→ Add→ map 토픽 추가
- 로봇을 적절히 움직여 사진과 같은 맵을 작성하도록 한다. 런치 파일이 잘 작성되었다면 오른쪽과 같은 메시지가 출력된다.
- 맵 작성 후 저장하지 않으면 데이터가 사라지니 주의할 것




```
/home/whiteherv/catkin_ws2/src/fourth_
update ld=0.00468729 ad=0.00539363
Laser Pose= 0.504303 0.0945183 0.27065
1
m_count 80
Average Scan Matching Score=445.185
neff= 99.9996
Registering Scans:Done
update frame 97
update ld=0.00341835 ad=0.00475928
Laser Pose= 0.50715 0.0964104 0.27541
m_count 81
Average Scan Matching Score=446.173
neff= 99.9996
Registering Scans:Done
update frame 98
update ld=0.00454182 ad=0.0047781
Laser Pose= 0.511107 0.0986398 0.28018
9
m_count 82
Average Scan Matching Score=447.22
neff= 99.9996
Registering Scans:Done
```

Navigation Stack – 2. 맵 작성

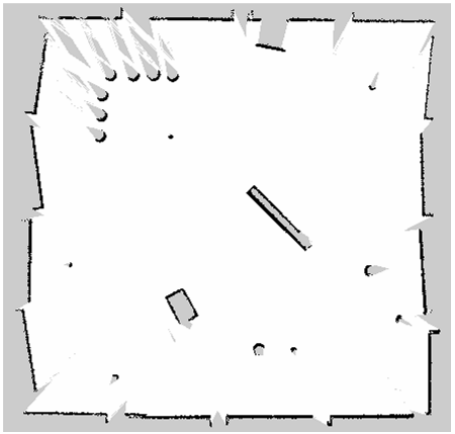
- map_server 패키지가 없다면 다운받는다.

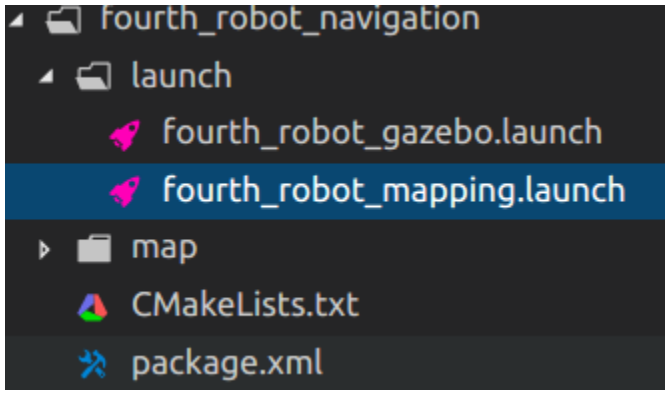
```
## sudo apt-get install ros-kinetic-map-server
```

- (map이 다 그려진 이후)map파일과 메타데이터 저장

```
## rosrun map_server map_saver -f playground
```

- fourth_robot_navigation 패키지에 map 폴더를 만들어, 생성된 맵파일과 yaml파일을 이동시킨다.





Navigation Stack – 3. Localization

- fourth_robot_navigation 패키지에 fourth_robot_localization.launch 파일을 생성하고 기본세팅 + 맵 로드 노드를 구성한다.

fourth_robot_localization.launch

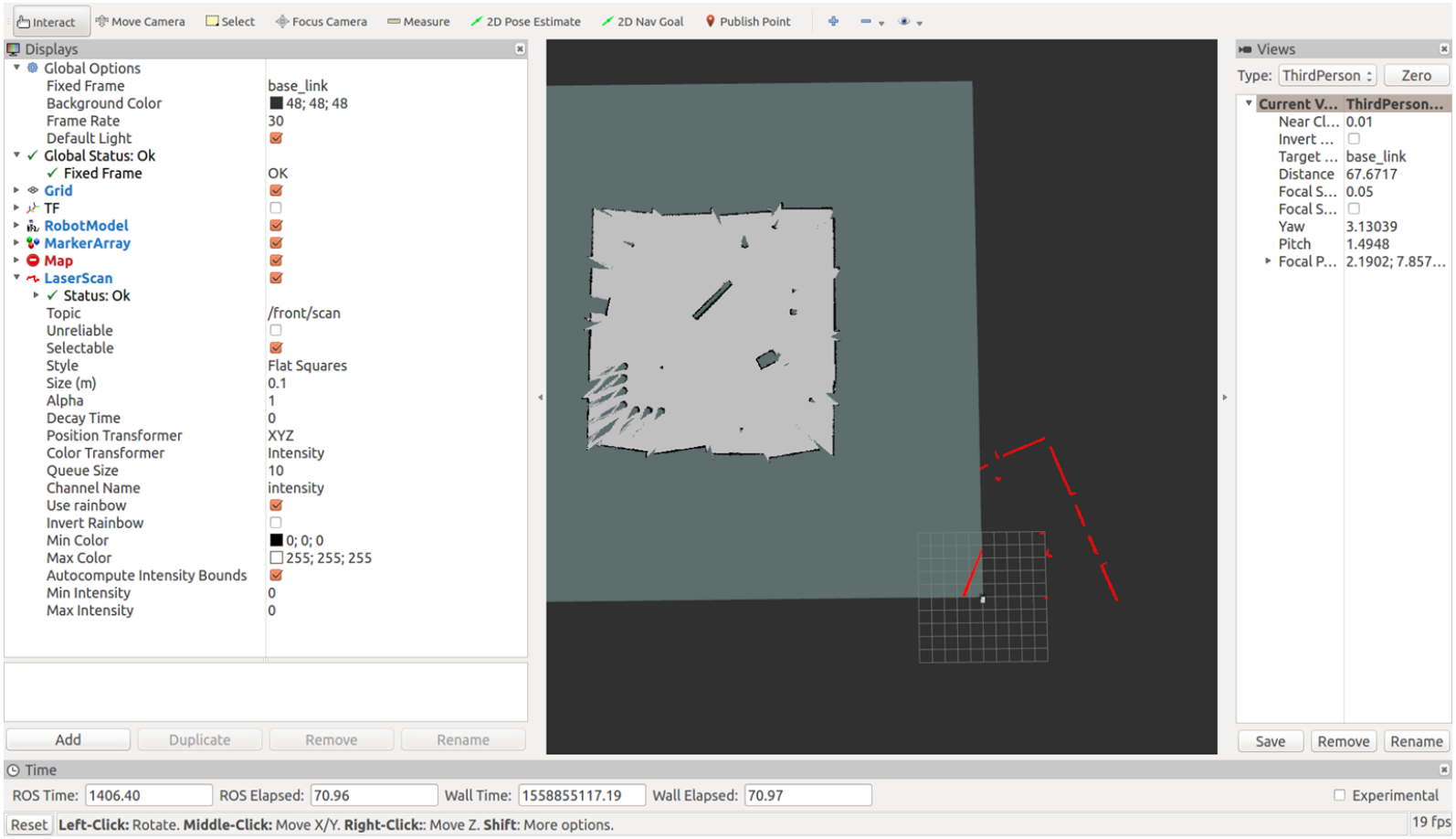
```
<launch>

  <!-- these are the arguments you can pass this launch file, for example paused:=true -->
  <arg name="model" default="$(find fourth_robot_description)/robots/fourth_robot.urdf.xacro"/>
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="gui" default="true"/>
  <arg name="headless" default="false"/>
  <arg name="debug" default="false"/>

  <!-- We resume the logic in empty_world.launch, changing only the name of the world to be launched -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
  <arg name="world_name" value="$(find fourth_robot_gazebo)/worlds/clearpath_playpen.world"/>
    <arg name="debug" value="$(arg debug)" />
    <arg name="gui" value="$(arg gui)" />
    <arg name="paused" value="$(arg paused)"/>
    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
    <arg name="headless" value="$(arg headless)"/>
  </include>

  <!-- lrf merger -->
  <include file="$(find fourth_robot_bringup)/launch/sensors/lrf_merger.launch"/>

  <!-- Load the URDF into the ROS Parameter Server -->
  <param name="robot_description"
     command="$(find xacro)/xacro.py '$(arg model)'" />

  <!-- Run a python script to the send a service call to gazebo_ros to spawn a URDF robot -->
  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model" respawn="false" output="screen"
      args="-urdf -model fourth_robot -param robot_description"/>

  <!-- ros_control motoman launch file -->
  <include file="$(find fourth_robot_control)/launch/fourth_robot_control.launch"/>

  <arg name="map_file" default="$(find fourth_robot_navigation)/map/playground.yaml"/>
  <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)" />

</launch>
```

fourth_robot_gazebo 참고

map load 코드 추가

Navigation Stack – 3. Localization

- rviz에 map이 표시되는지 확인



Navigation Stack – 3. Localization

- 로봇이 주어진 맵에서 자신의 위치와 속도를 파악하기 위해서는 Localization 이 필요하다. 이를 위해 Adaptive Monte Carlo Localization (Described by Dieter Fox) 알고리즘을 이용한다.
- AMCL의 구체적인 알고리즘과 파라미터는 Thrun, Burgard, and Fox이 집필한 Probablic Robotics에 소개되어있다. 이 강의에서는 간단히 **로봇의 위치와 속도**를 **particle filter 기반의 확률적 알고리즘을 이용해 근사**시킨 다는 점만 이해하고 넘어가도록 한다.
- 참고 : https://www.youtube.com/watch?v=aDiTGWIoPL0

Navigation Stack – 3. Localization

- turtlebot3 오픈소스를 참고해 amcl 소스를 mobile_robot_localization.launch 하단에 추가하도록 한다.
  https://github.com/ROBOTIS-GIT/turtlebot3/blob/master/turtlebot3_navigation/launch/amcl.launch
- amcl에서 사용하는 모든 파라미터는 로스 위키에서 확인할 수 있다.
  http://wiki.ros.org/amcl
- turtlebot3 오픈소스를 참고해 gmapping 런치파일 세팅과 같은 방식으로 프레임과 scan 토픽명을 맞춘다.

Navigation Stack – 3. Localization

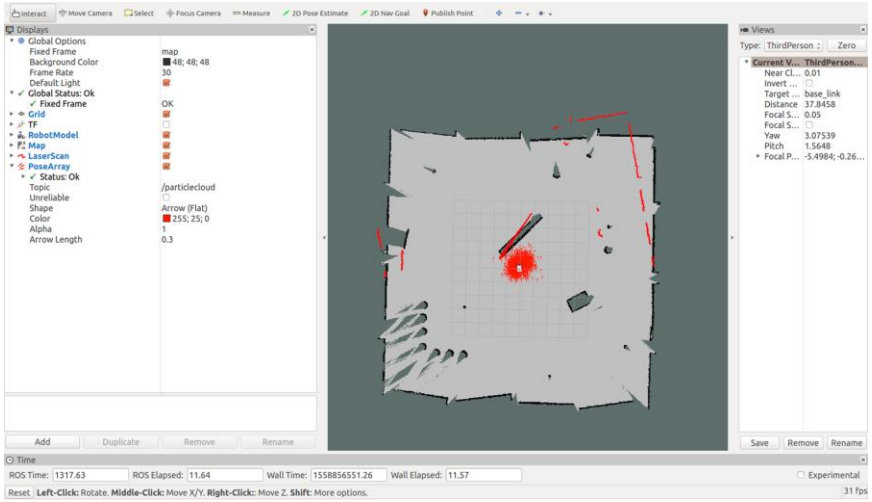
- amcl 패키지가 없다면 다운받는다.

```
## sudo apt-get install ros-kinetic-amcl
```

- Localization 런치파일 실행

```
## roslaunch mobile_robot mobile_robot_localization.launch
```

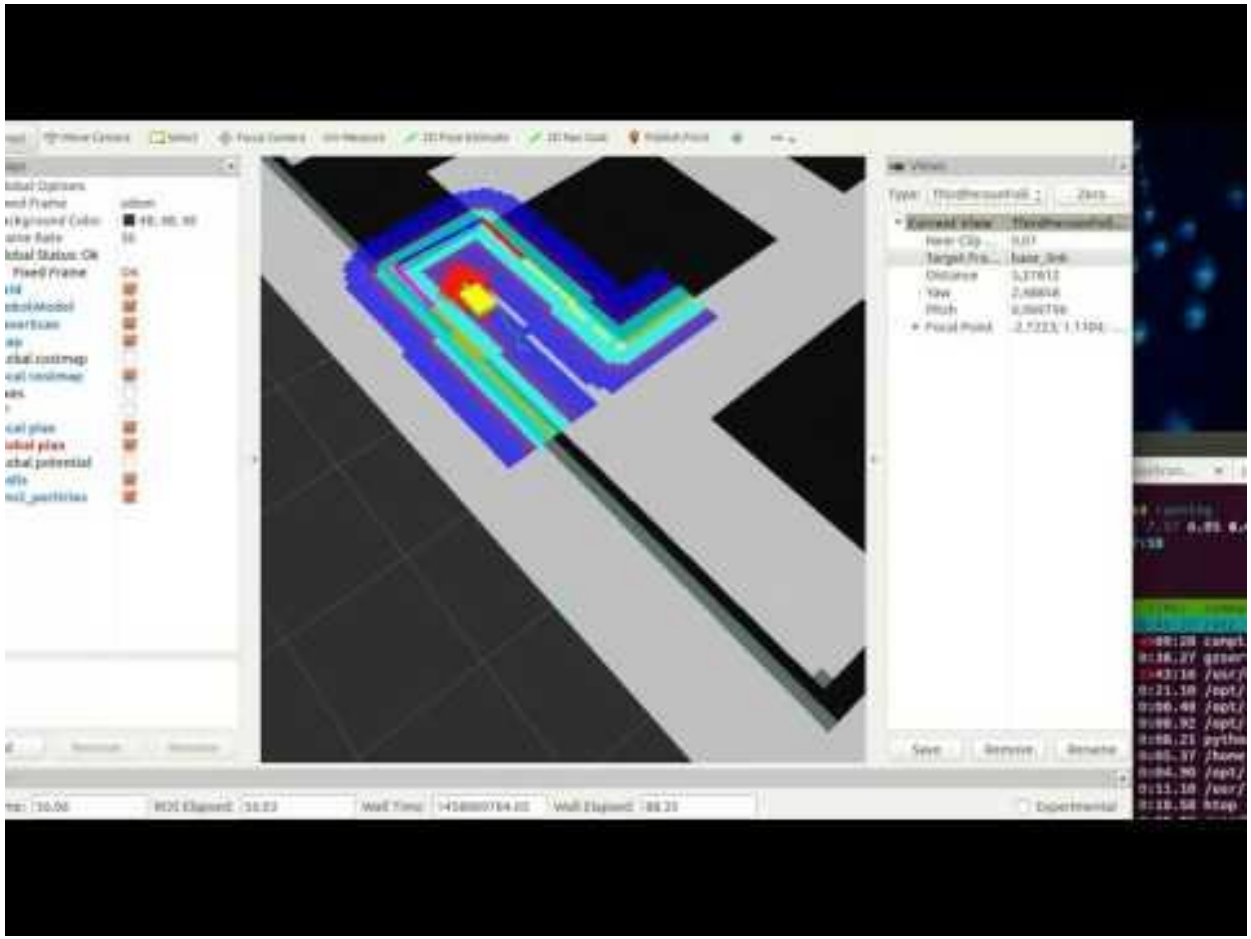
- rviz에서 PoseArray 토픽을 추가한다.



Navigation Stack – 3. Localization

- 과제 - 로컬라이징 동영상 촬영
- 동영상만 제출한다. 아래의 내용이 포함되어야 한다.
  - localization 런치파일/teleop/rviz실행화면(하나의 런치파일에 나머지를 포함시켜도 상관없음. 단 아무것도 켜지 않은 상태에서 시작해야함)
  - rosrun rqt_tf_tree rqt_tf_tree 실행화면
  - 비전랩 홈페이지 영상 참고

Navigation Stack – 4. Navigation



Navigation Stack – 4. Navigation

- 로봇이 Goal에 도착하기 위해서는 경로를 계획하고 모터와 같은 엑츄에이터에 명령을 내려야 한다. 이러한 경로계획 문제를 Navigation이라 한다.
- Navigation Stack은 Navigation 문제를 해결하기 위해 **갈 수 없는 위치, 가능하면 피하고 싶은 위치를 표시한 Occupancy Grid map**(costmap)을 구성하고 이러한 맵을 기반으로 로봇의 부피, 최대가속도나 최대각속도 등 **로봇 파라미터를 고려해 Planning**을 수행하는 인터페이스를 제공한다.
- 또한 Navigation Stack은 문제를 Global/Local을 나누어 해결한다. Navigation Stack의 Global Planning에서는 목표 지점까지 도달하기 위한 대략적인 경로를 계산한다. Local Planning에서는 로봇 파라미터를 기반으로 Global Path를 수정하고 가능한 여러개의 경로를 생성한다. 그리고 각각의 경로에 대해 충돌 등을 조사해 scoring을 수행하고, 가장 점수가 높은 경로를 최종 Local Planning 경로로 선택, cmd_vel을 발행한다.

Navigation Stack – 4. Navigation

- turtlebot3_navigation/params의 파라미터 파일들의 내용을 fourth_robot_navigation/cfg 안에 복사한다.
  - costmap 관련
    - costmap_common_params_burger.yaml□ costmap_common_params
    - global_costmap_params.yaml
    - local_costmap_params.yaml
  - planner 관련
    - move_base_params.yaml
    - dwa_local_planner_params_burger.yaml□ dwa_local_palnner_params.yaml

Navigation Stack – 4. Navigation

- fourth_robot_navigation 패키지에
  fourth_robot_move_base.launch를 작성한다.

```
<launch>
  <arg name="odom_topic" default="odom" />

  <!-- move_base -->
  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
    <param name="base_local_planner" value="dwa_local_planner/DWAPlannerROS" />
    <rosparam file="$(find fourth_robot_navigation)/cfg/costmap_common_params.yaml" command="load" ns="local_costmap" />
    <rosparam file="$(find fourth_robot_navigation)/cfg/local_costmap_params.yaml" command="load" />
    <rosparam file="$(find fourth_robot_navigation)/cfg/global_costmap_params.yaml" command="load" />
    <rosparam file="$(find fourth_robot_navigation)/cfg/move_base_params.yaml" command="load" />
    <rosparam file="$(find fourth_robot_navigation)/cfg/dwa_local_planner_params.yaml" command="load" />
    <remap from="odom" to="$(arg odom_topic)"/>
    <remap from="cmd_vel" to="diff_drive_controller/cmd_vel"/>
  </node>
</launch>

```

Navigation Stack – 4. Navigation

- move_base가 동작하도록 토픽/프레임 명을 fourth_robot_navigation에 맞게 수정한다. 필요한 파라미터의 값은 rviz나 rqt_graph, 지금까지 실행시킨 런치파일등을 활용해 찾아낸다.
  - costmap_common_params.yaml의 footprint : 로봇의 충돌 부피 계산을 위해 사용하는 정보이다. custom_base 프레임을 기준으로 로봇의 면적을 기술한다. 여기서 면적값은 로봇을 xy-plane에 사영시켰을 때의 면적보다 조금 크게 설정하는 것이 좋다.
  - 여러가지 프레임
  - costmap_common_params의 scan 관련 파트

Navigation Stack – 4. Navigation

- localizing 실행

```
## roslaunch fourth_robot_navigation
fourth_robot_localization.launch
```

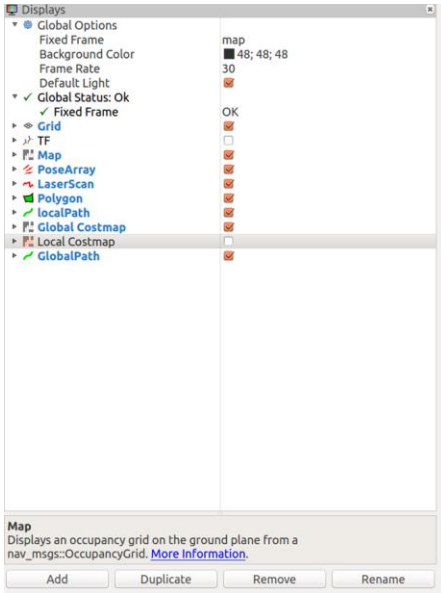
- move_base 실행

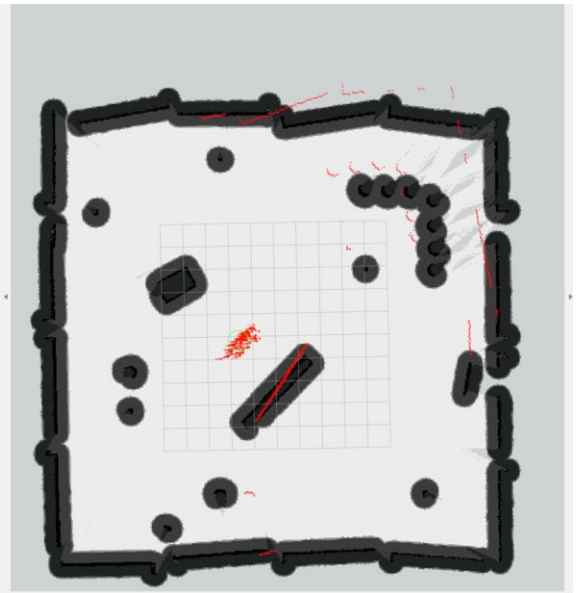
```
## roslaunch mobile_robot mobile_robot_move_base.launch
```

move_base를 실행시킨 터미널에서 warning이 발생하면 않도록 한다.

Navigation Stack – 4. Navigation

- rviz에서 다음 토픽들을 추가하고, 이름을 적절히 변경한 후 rviz의 세팅을 저장한다.
  - footprint(global, local 상관없음)
  - global_plan(NavfnRos/plan)
  - local_plan(DWAPlannerROS/globalPlan)
  - glocal_costmap
  - local_costmap





Navigation Stack – 4. Navigation

- Global Options에서 FixedFrame을 map 으로 설정한다.
- rviz 상단에 PanelI에서 tool 체크 확인
- 2D Nav Goal로 로봇 골 체크