

# SLAM & Navigation

Department of Electrical Engineering, Incheon National University  
**Hwasu Lee, Munkyu Bae**

2024.07.02~05

# 강사 소개



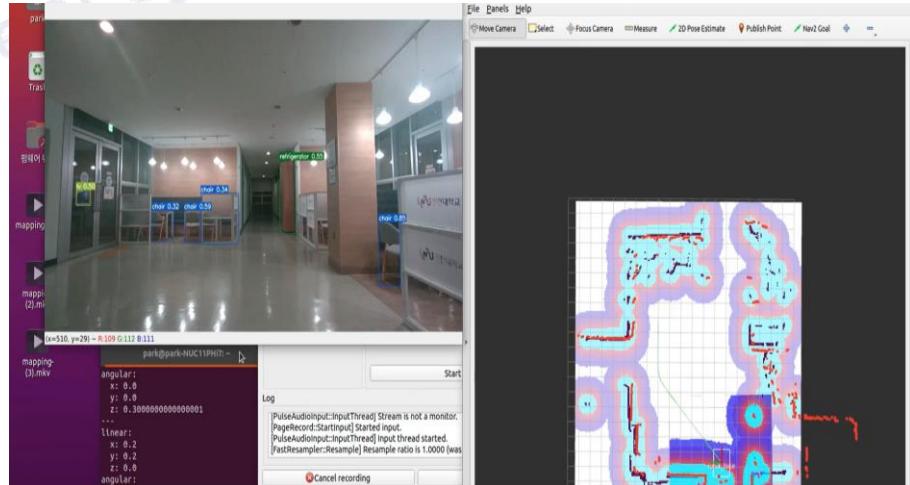
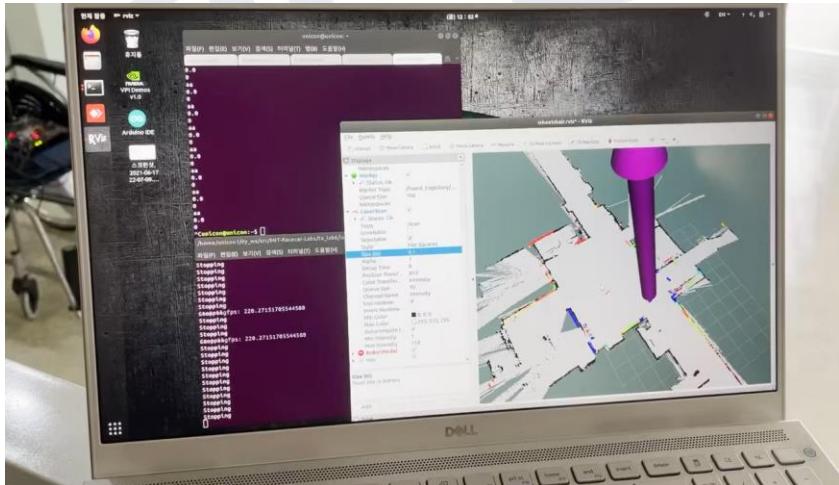
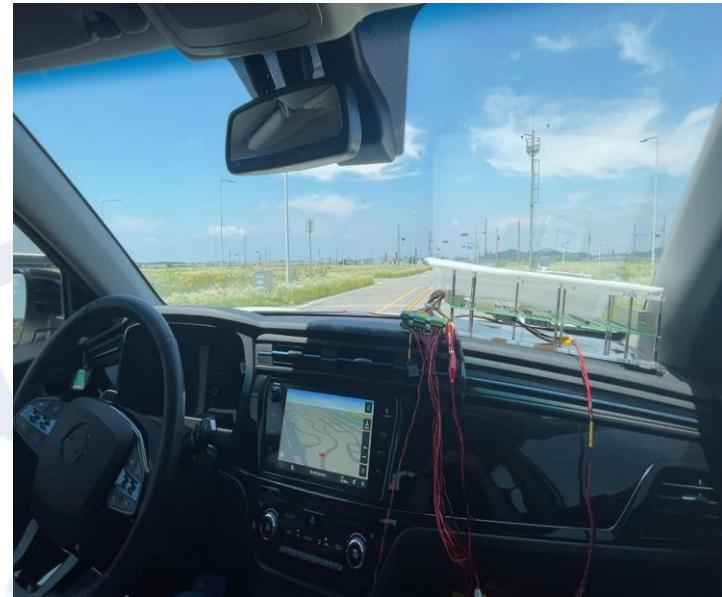
- 이화수 (Hwasu Lee)
- 2015.03 ~ 2022.02: 인천대학교 전기공학과 학사  
2022 ~ 현재: 인천대학교 전기공학과 석사 과정 재학 중
- 무인지능 시스템제어 연구실 소속  
(지도교수: 강창묵 교수, <https://uniconlab.wixsite.com/main>)
- 관심 분야: 예측 제어, 경로 탐색, 강화학습, 심층 강화학습  
ROS 1 & ROS 2 기반 SLAM 및 Navigation 등
- 적용 플랫폼: 자율주행 차량, 모바일 로봇 등
- Projects
  - 자율주행 전동휠체어 핵심 기술 개발
  - 순찰로봇의 도심지 적용을 위한 자율주행 기술 개발
  - 지능형 건물 바닥 청소 로봇 플랫폼 개발
  - 인천공항 터미널 폐기물 수집 및 이송로봇 개발

# 강사 소개



- 배문규 (Munkyu Bae)
- ~ 2022.02: 인천대학교 전기공학과 학사  
2022 ~ 현재: 인천대학교 전기공학과 석사 과정 재학 중
- 무인지능 시스템제어 연구실 소속  
(지도교수: 강창욱 교수, <https://uniconlab.wixsite.com/main>)
- 관심 분야: 선형 제어, 비선형 제어, 예측 제어, 센서 퓨전 등
- 적용 플랫폼: 자율주행 차량, 모바일 로봇 등
- Projects
  - 전동휠체어 자율주행 핵심 기술 개발
  - Single-Channel LiDAR 기반 3D mapping 구현
  - 협동로봇을 활용한 언텍트 스토어 운영시스템 개발
  - 동작 분석을 위한 센서 및 알고리즘 개발
  - 자율주행 차량 토크/조향/제구동 통합제어기 개발

# 연구실 수행 프로젝트



# Contents

## 1. SLAM

- 1) SLAM이란?
- 2) SLAM의 종류
- 3) SLAM의 원리
- 4) ROS 기반 SLAM 실습
- 5) GIMP 기반 Occupancy Grid Map 보정

## 2. Navigation

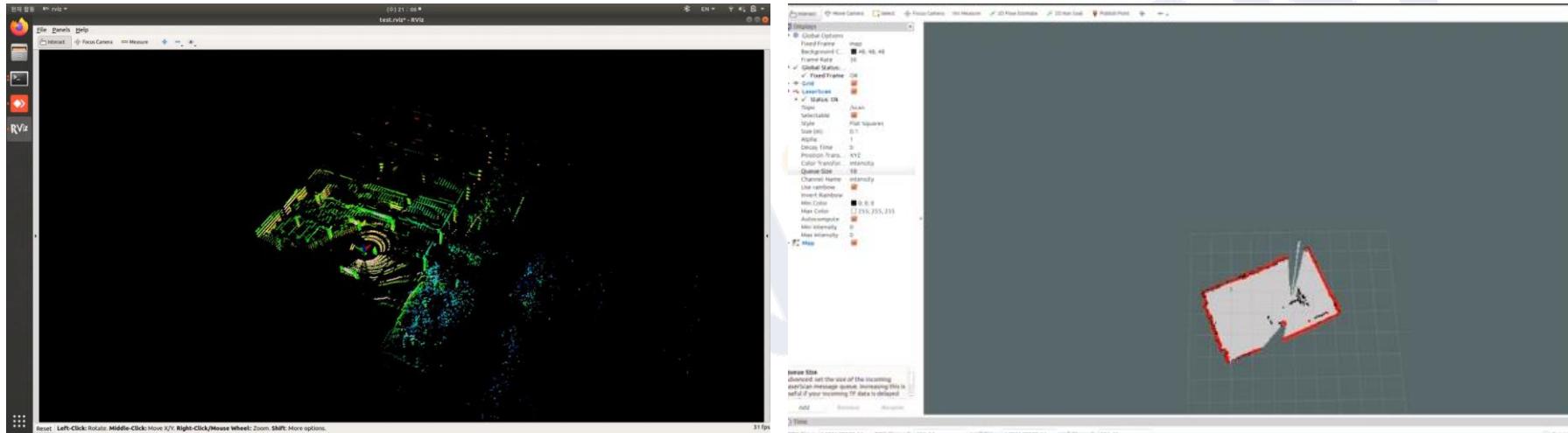
- 1) ROS Navigation Stack이란?
- 2) ROS Navigation Stack의 구조 및 원리
- 3) ROS Navigation Stack 기반 자율주행 실습

# **1. SLAM**

uni-UNICON  
Unmanned & Intelligent systems CONtrol LAB

# 1. SLAM

## 1) SLAM이란?

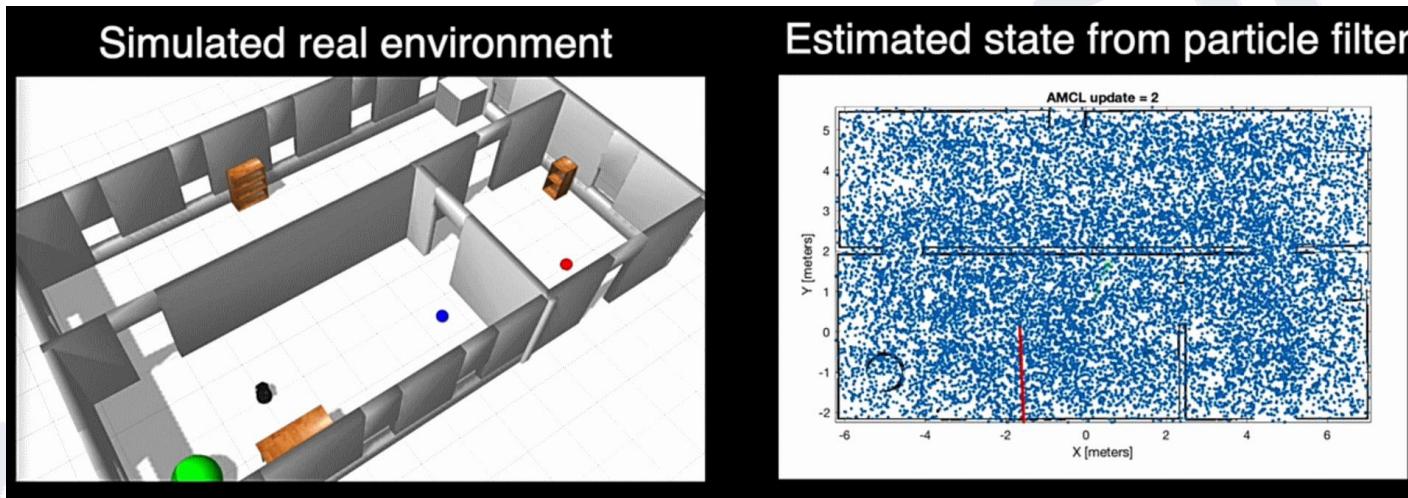


자료 출처. <https://wiki.ros.org/rplidar>

- SLAM(Simultaneous Localization And Mapping)은 로봇 공학과 컴퓨터 비전 분야에서 로봇 또는 자율주행 차량이 알려지지 않은 환경을 탐색할 때, 해당 환경의 지도를 만들어 냄과 동시에 지속적으로 자신의 위치를 추정하는 기술.
- SLAM은 로봇이 센서를 통해 주변 환경을 지속적으로 관찰하고 그 관찰 결과를 바탕으로 지도를 생성하며, 생성된 지도를 기반으로 자신의 위치를 업데이트.
- SLAM의 주요 구성 요소는 ‘Localization’과 ‘Mapping’으로 구성.

# 1. SLAM

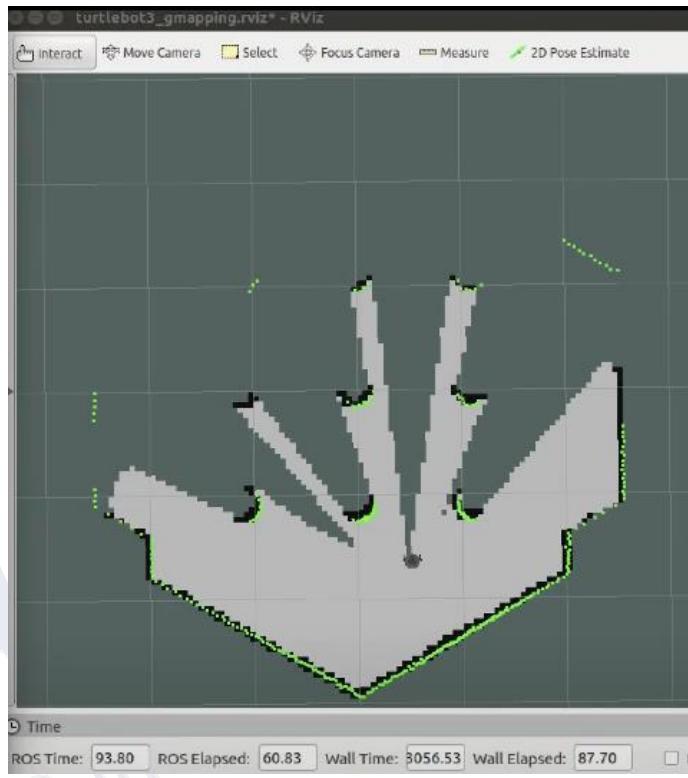
## 1) SLAM이란?



- Localization : Localization은 로봇이 현재 자신이 어디에 있는지를 파악하는 과정.  
이는 로봇이 지도 상에서 자신의 위치를 정확하게 추정하고, 센서 데이터와 지도 데이터를 매칭하여 지속적으로 업데이트하는 과정을 포함함.

# 1. SLAM

## 1) SLAM이란?



자료 출처. <https://www.youtube.com/shorts/Y1VXlrZEPSk>

- **Mapping** : Mapping은 로봇이 환경을 탐색하면서 주변의 물리적인 특징들을 추출하여 지도를 생성하는 과정.

생성된 지도는 로봇이 나중에 같은 환경을 탐색할 때나 다른 로봇이 해당 환경을 파악하는 데 사용될 수 있음.

# 1. SLAM

## 2) SLAM의 종류

- SLAM은 크게 LiDAR 센서 기반의 SLAM과, Camera 센서 기반의 SLAM으로 구분.
- LiDAR 센서 기반의 SLAM은, LiDAR 센서의 Scan data(거리 데이터)와 위치 추정을 위한 IMU 및 Odometry의 data를 기반으로 환경을 매핑함과 동시에 위치를 추정.
- Camera 센서 기반의 SLAM(=Visual SLAM)은, Camera 센서의 Image data를 통해 환경을 매핑하고 위치를 추정.
- LiDAR 기반 SLAM은, Camera 기반 SLAM 대비 고정밀의 환경 정보 생성 가능.
- Camera 기반 SLAM은, LiDAR 기반 SLAM 대비 비용이 저렴하며, 환경의 시각적 세부 정보를 취득 가능.

# 1. SLAM

## 2) SLAM의 종류 – Camera 센서 기반 SLAM (Visual SLAM)

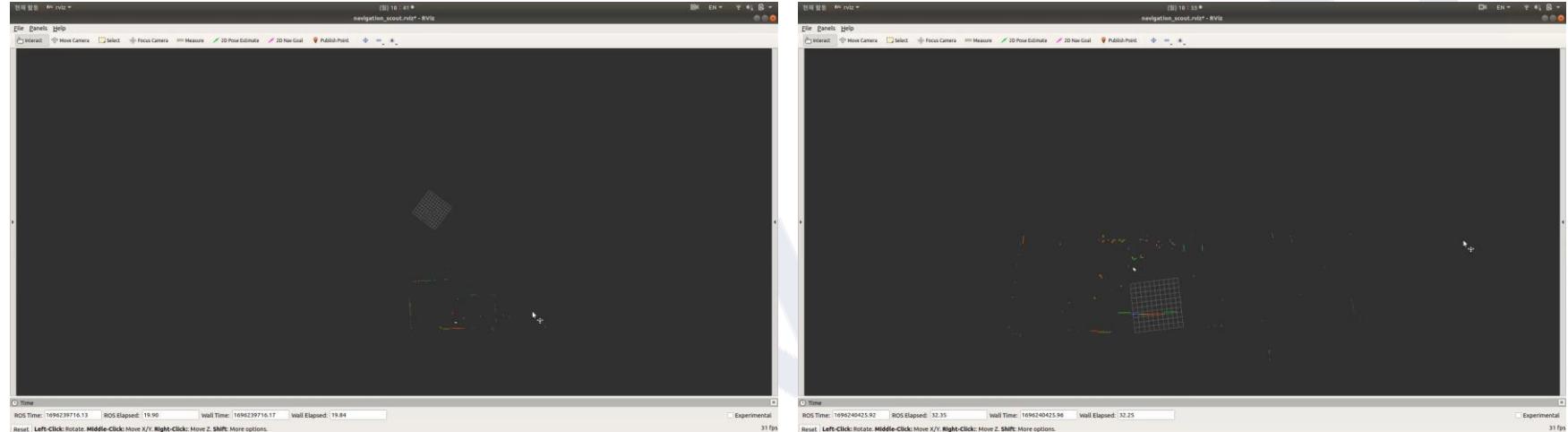


자료 출처. <https://www.youtube.com/watch?v=8DISRmsO2YQ>

- 위 영상은 KITTI Dataset을 기반으로 수행된 Visual SLAM 예시 영상.
- 사용된 Visual SLAM 기법은 ORB(Oriented FAST and Rotated BRIEF) SLAM 기법으로, 가장 대표적인 Visual SLAM 기법 중 하나.
- ORB SLAM은 **single camera**, **stereo camera**, **RGB-D camera** 모두에서 사용 가능.
- ORB SLAM 외에도 LSD(Large-Scale Direct Monocular) SLAM 등 다양한 Visual SLAM 기법이 존재하며, 현재에도 계속하여 개선된 Visual SLAM 기법이 연구되어지고 있음.

# 1. SLAM

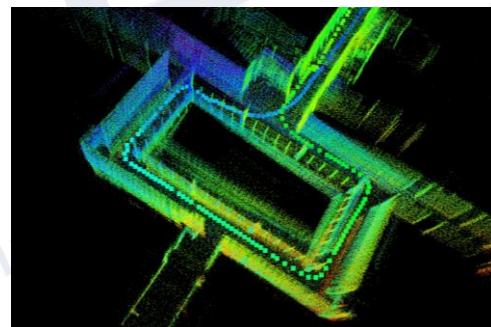
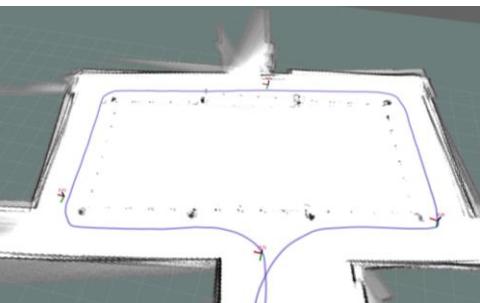
## 2) SLAM의 종류 – LiDAR 센서 기반 SLAM (LiDAR SLAM)



- 위 영상은 ‘인천공항 터미널 폐기물 수집 및 이송로봇’ 과제에서 수행한 인천공항 3층 일부를 2D LiDAR 기반 SLAM을 통해 Mapping한 영상.
- 사용된 LiDAR SLAM 기법은 Google에서 개발한 Cartographer 기법으로, Loop Closure를 자동으로 탐지하고 처리함으로써 높은 정밀도 및 신뢰도를 가지는 기법.
- Cartographer 외에도, Hector, Gmapping, LOAM, LeGo-LOAM 등 다양한 SLAM 기법이 존재

# 1. SLAM

## 2) SLAM의 종류 – LiDAR 센서 기반 SLAM (2D/3D LiDAR SLAM)



- **2D Mapping**

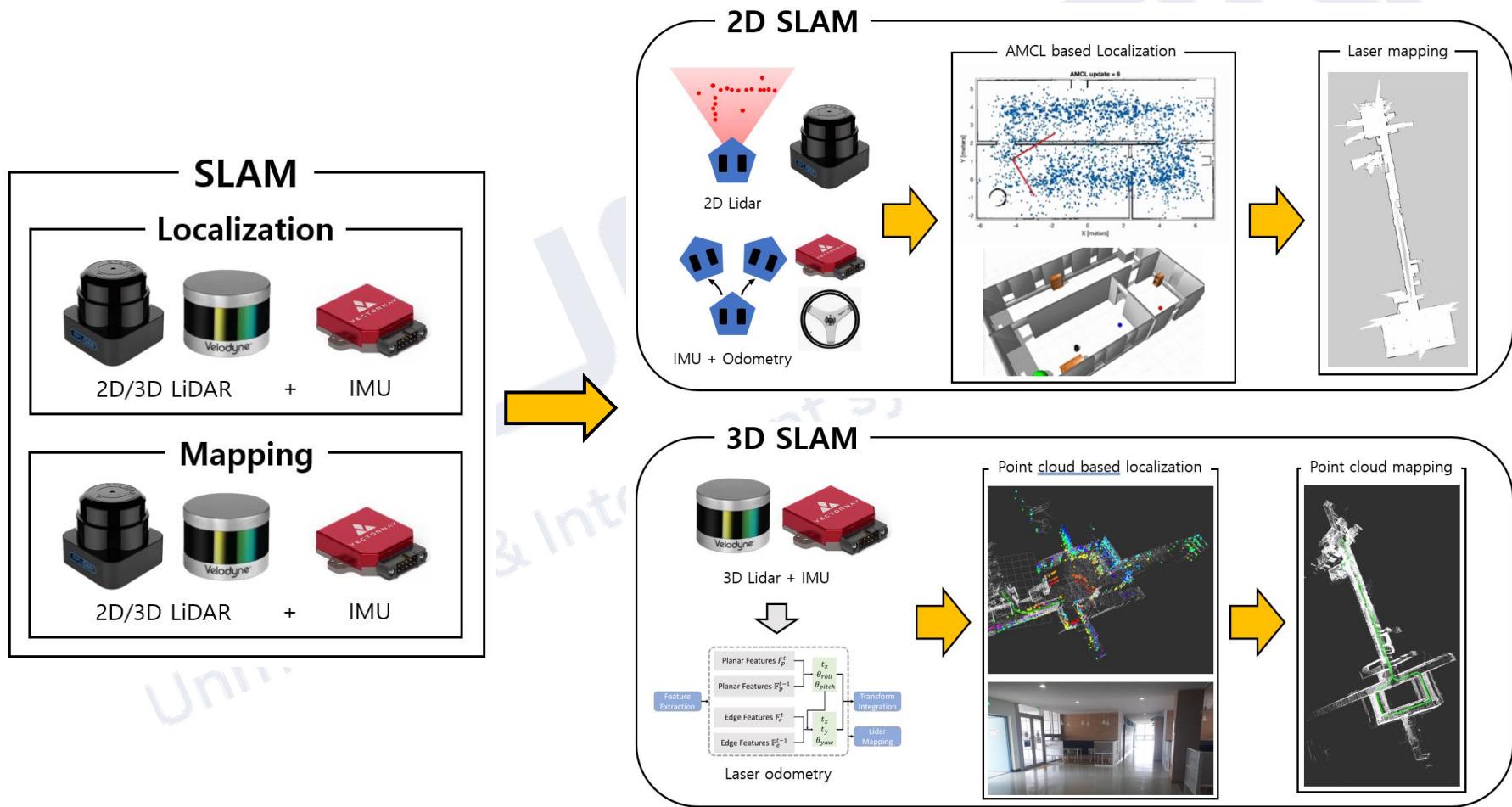
- Hector mapping : 2D LIDAR
- Gmapping : 2D LIDAR, Odometry
- Lifelong : 2D LIDAR, Odometry
- **2D Cartographer** : 2D LIDAR, IMU, (Odometry)
- ...

- **3D Mapping**

- LOAM : 3D LIDAR, IMU
- **LeGo-LOAM** : 3D LIDAR, IMU
- 3D Cartographer : 3D LIDAR, IMU, (Odometry)
- Octomap : 3D LIDAR, Depth camera
- ...

# 1. SLAM

## 2) SLAM의 종류 – LiDAR 센서 기반 SLAM (2D/3D LiDAR SLAM)



# 1. SLAM

## 3) LiDAR SLAM의 원리

### ① Initialization (초기화)

→ 로봇은 초기 위치를 (0, 0) 또는 임의의 위치로 설정하고, 초기 지도 상태를 설정.

### ② Sensor Data Acquisition (센서 데이터 획득)

→ 로봇은 LiDAR 및 IMU 센서 등을 바탕으로 주변 환경에 대한 정보를 수집.

→ 센서 데이터를 통해 주변의 장애물, 랜드마크, 벽 등의 정보를 획득.

### ③ Feature Extraction (특징점 추출)

→ 센서 데이터(ex. LiDAR 데이터)를 처리하여 장애물의 경계, 코너, 벽과 같은 유의미한 특징적인 점들을 추출.

→ 해당 과정은 센서의 raw 데이터를 더 유용한 형태로 변환하는 과정.

### ④ Scan Matching (스캔 매칭)

→ 현재 스캔 데이터와 이전 스캔 데이터를 비교하여 두 스캔 데이터 간의 최적의 정합(Alignment) 결과 도출.

→ 스캔 매칭을 통해 로봇의 상대적인 이동(이동 거리 및 방향)을 계산하며, 로봇의 현재 위치를 추정.

# 1. SLAM

## 3) LiDAR SLAM의 원리

### ⑤ Feature Matching (특징점 매칭)

- 스캔 매칭 결과를 바탕으로 현재 지도와 비교하여 특징점을 매칭.
- 스캔 매칭 결과는 피쳐 매칭 과정의 중요한 입력으로 사용되며,  
기존 지도에 존재하는 특징점과 새로 추출된 특징점을 비교하여 일치하는 특징점을 찾음.

### ⑥ Pose Estimation (로봇 위치 추정)

- 스캔 매칭과 피쳐 매칭 결과를 종합하여 로봇의 위치를 더욱 정확하게 추정.
- 여러 센서 데이터를 융합하여 위치 추정 정확도를 향상시킬 수 있음.

### ⑦ Map Update (지도 업데이트)

- 로봇의 위치와 새로운 특징점 정보를 기반으로 지도 업데이트 수행.
- 새로운 피쳐를 추가하거나 기존 피쳐의 위치를 조정하여 지도를 갱신.

### ⑧ Loop Closure (루프 클로징)

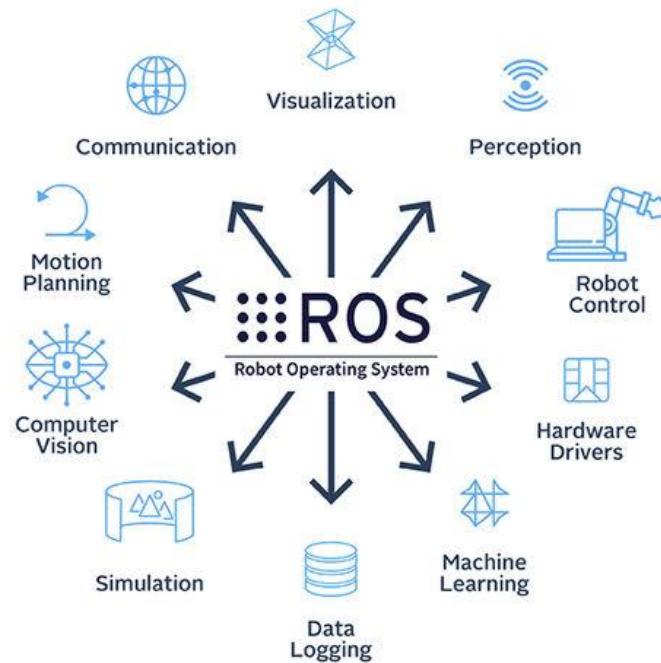
- 로봇의 위치 추정 오차를 줄이고 지도의 일관성을 유지하기 위해, 로봇이 이미 탐색한 지역으로 돌아왔을 때 스캔 매칭 기반의 루프 클로징 과정을 수행하여 지도의 일관성 유지 및 정확도 향상.

### ⑨ Iteration (반복)

- 로봇이 주어진 환경을 완전히 탐색할 때까지 위 ① ~ ⑧ 단계를 반복함으로써 지도 작성 및 위치 추정 수행.

# 1. SLAM

## 4) ROS 기반 SLAM 실습 (1/19)



자료 출처: <https://www.directindustry.com/prod/automationware/product-192516-2394360.html>

- ROS(Robot Operating System)는 오픈 소스이며, 메타 운영 시스템(meta-operating system)으로 주로 Linux 환경에서 사용됨.
- ROS는 소프트웨어 개발자들이 로봇 애플리케이션을 쉽게 만들 수 있도록 도와주는 라이브러리와 도구를 제공.

# 1. SLAM

## 4) ROS 기반 SLAM 실습 [2/19]

### catkin\_ws 생성 및 빌드

```

jay@ubuntu: ~/catkin_ws
jay@ubuntu: ~/catkin_ws 80x24
jay@ubuntu: $ mkdir -p ~/catkin_ws/src
jay@ubuntu: $ cd ~/catkin_ws/src/
jay@ubuntu:~/catkin_ws/src$ catkin_init_workspace
Creating symlink "/home/jay/catkin_ws/src/CMakeLists.txt" pointing to "/opt/ros/melodic/share/catkin/cmake/toplevel.cmake"
jay@ubuntu:~/catkin_ws/src$ ls
CMakeLists.txt
jay@ubuntu:~/catkin_ws/src$ cd ..
jay@ubuntu:~/catkin_ws$ catkin_make → = $ cd ~/catkin_ws && catkin_make
Base path: /home/jay/catkin_ws
Source space: /home/jay/catkin_ws/src
Build space: /home/jay/catkin_ws/build
Devel space: /home/jay/catkin_ws/devel
Install space: /home/jay/catkin_ws/install
#####
##### Running command: "cmake /home/jay/catkin_ws/src -DCATKIN_DEVEL_PREFIX=/home/jay/catkin_ws/devel -DCMAKE_INSTALL_PREFIX=/home/jay/catkin_ws/install -G Unix Makefiles" in "/home/jay/catkin_ws/build"
#####
-- The C compiler identification is GNU 7.5.0
-- The CXX compiler identification is GNU 7.5.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
자료 출처. https://velog.io/@cjh2626002/catkin-catkin-workspace-%EC%83%9D%EC%84%B1

```

- catkin\_ws는 Catkin workspace의 줄임말로, ROS에서 사용하는 패키지에 대한 환경 설정 관리, 빌드 등 ROS 패키지를 효율적으로 개발·관리하기 위한 workspace를 의미

\$ mkdir -p ~/catkin\_ws/src

(→ catkin\_ws 생성 후, 하위 폴더에 src 폴더 생성)

\$ cd ~/catkin\_ws/src

(→ catkin\_ws 내에 위치한 하위 폴더인 src 폴더로 이동)

\$ catkin\_init\_workspace

(→ catkin\_ws 초기화 및 빌드 환경 설정)

\$ cd ~/catkin\_ws && catkin\_make

(→ catkin\_ws 빌드)

\$ source devel/setup.bash

(→ 환경 변수에 ROS 패키지 경로 추가)

# 1. SLAM

## 4) ROS 기반 SLAM 실습 [3/19]

### Turtlebot3 관련 패키지 설치

```
unicon3@unicon3: ~ /catkin_ws/src /  
unicon3@unicon3: ~/catkin_ws/src $ git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3.git  
Cloning into 'turtlebot3'...  
remote: Enumerating objects: 6636, done.  
remote: Counting objects: 100% (1328/1328), done.  
remote: Compressing objects: 100% (207/207), done.  
remote: Total 6636 (delta 1163), reused 1172 (delta 1121), pack-reused 5308  
Receiving objects: 100% (6636/6636), 119.99 MiB | 4.59 MiB/s, done.  
Resolving deltas: 100% (4151/4151)... done.  
unicon3@unicon3: ~/catkin_ws/src $ git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git  
Cloning into 'turtlebot3_simulations'...  
remote: Enumerating objects: 3167, done.  
remote: Counting objects: 100% (1168/1168), done.  
remote: Compressing objects: 100% (236/236), done.  
remote: Total 3167 (delta 1004), reused 932 (delta 932), pack-reused 1999  
Receiving objects: 100% (3167/3167), 15.36 MiB | 4.01 MiB/s, done.  
Resolving deltas: 100% (1878/1878), done.  
unicon3@unicon3: ~/catkin_ws/src $
```



\$ cd ~/catkin\_ws/src/ (→ catkin\_ws로 이동)

\$ git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3.git

(→ Turtlebot3 관련 패키지 다운로드)

\$ git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3\_simulations.git

(→ Turtlebot3 시뮬레이션 환경 관련 패키지 다운로드)

# 1. SLAM

## 4) ROS 기반 SLAM 실습 (4/19)

### catkin\_ws 빌드

```

unicon3@unicon3:~/catkin_ws$ cd ~/catkin_ws && catkin_make
Base path: /home/unicon3/catkin_ws
Source space: /home/unicon3/catkin_ws/src
Build space: /home/unicon3/catkin_ws/build
Devel space: /home/unicon3/catkin_ws/devel
Install space: /home/unicon3/catkin_ws/install
=====
===== Running command: "make cmake_check_build_system" in "/home/unicon3/catkin_ws/build"
=====
Using CMAKE_DEVEL_PREFIX: /home/unicon3/catkin_ws/devel
-- Using CMAKE_PREFIX_PATH: /home/unicon3/catkin_ws/devel;/opt/ros/noetic
-- This workspace overlays: /home/unicon3/catkin_ws/devel;/opt/ros/noetic
-- Found PythonInterp: /usr/bin/python3 (found suitable version "3.8.10", minimum required is "3")
-- Using PYTHON_EXECUTABLE: /usr/bin/python3
-- Using debian Python package layout
-- Using empty: /usr/lib/python3/dist-packages/_em.py
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CMAKE_TESTS_RESULTS_DIR: /home/unicon3/catkin_ws/build/test_results
-- Forcing gtest/gmock from source, though one was otherwise available.
-- Found gtest sources under '/usr/src/googletest': gtests will be built
-- Found gmock sources under '/usr/src/googletest': gmock will be built
CMake Deprecation Warning at /usr/src/googletest/CMakeLists.txt:4 (cmake_minimum_required):
  Compatibility with CMake < 2.8.12 will be removed from a future version of
  CMake.

  Update the VERSION argument <min> value or use a ...<max> suffix to tell
  CMake that the project does not need compatibility with older versions.

CMake Deprecation Warning at /usr/src/googletest/googletest/CMakeLists.txt:45 (cmake_minimum_required):
  Compatibility with CMake < 2.8.12 will be removed from a future version of
  CMake.

  Update the VERSION argument <min> value or use a ...<max> suffix to tell
  CMake that the project does not need compatibility with older versions.

CMake Deprecation Warning at /usr/src/googletest/googletest/CMakeLists.txt:56 (cmake_minimum_required):
  Compatibility with CMake < 2.8.12 will be removed from a future version of
  CMake.

  Update the VERSION argument <min> value or use a ...<max> suffix to tell
  CMake that the project does not need compatibility with older versions.

-- Found PythonInterp: /usr/bin/python3 (found version "3.8.10")
-- Using Python nosetests: /usr/bin/nosetests3
-- catkin 0.8.10
-- BUILD_SHARED_LIBS is on
-- BUILD_SHARED_LIBS is on
-- 
-- traversing 16 packages in topological order:
--   detection_node
[ 64%] Generating Lisp code from turtlebot3_example/Turtlebot3ActionGoal.msg
[ 64%] Generating Javascript code from turtlebot3_example/Turtlebot3ActionGoal.msg
[ 65%] Generating Javascript code from turtlebot3_example/Turtlebot3ActionResult.msg
[ 68%] Built target yolov5_ros_generate_messages_py
[ 69%] Generating Javascript code from turtlebot3_example/Turtlebot3ActionFeedback.msg
[ 70%] Generating Euslisp code from turtlebot3_example/Turtlebot3Goal.msg
[ 72%] Building CXX object turtlebot3_simulations/turtlebot3_gazebo/CMakeFiles/turtlebot3_drive.dir/simulation/turtlebot3_drive.cpp.o
[ 72%] Building CXX object turtlebot3_simulations/turtlebot3_fake/CMakeFiles/turtlebot3_fake_node.dir/simulation/turtlebot3_fake.cpp.o
[ 73%] Generating Lisp code from turtlebot3_example/Turtlebot3ActionResult.msg
[ 73%] Generating C++ code from turtlebot3_example/Turtlebot3ActionResult.msg
[ 74%] Generating Python from MSG turtlebot3_example/Turtlebot3ActionFeedback
[ 75%] Generating Python from MSG turtlebot3_example/Turtlebot3Goal
[ 76%] Generating Javascript code from turtlebot3_example/Turtlebot3Goal.msg
[ 76%] Generating Javascript code from turtlebot3_example/Turtlebot3Result.msg
[ 77%] Generating Python from MSG turtlebot3_example/Turtlebot3Result
[ 78%] Generating Lisp code from turtlebot3_example/Turtlebot3ActionFeedback.msg
[ 79%] Generating Euslisp code from turtlebot3_example/Turtlebot3Result.msg
[ 80%] Generating Euslisp code from turtlebot3_example/Turtlebot3Feedback.msg
[ 81%] Generating Javascript code from turtlebot3_example/Turtlebot3Feedback.msg
[ 82%] Generating Lisp code from turtlebot3_example/Turtlebot3Goal.msg
[ 82%] Generating Python from MSG turtlebot3_example/Turtlebot3Feedback
[ 83%] Generating C++ code from turtlebot3_example/Turtlebot3ActionResult.msg
[ 84%] Generating C++ code from turtlebot3_example/Turtlebot3ActionFeedback.msg
[ 84%] Built target turtlebot3_example_generate_messages_nodejs
[ 86%] Generating Lisp code from turtlebot3_example/Turtlebot3Result.msg
[ 88%] Built target turtlebot3_example_generate_messages_eus
[ 89%] Built target yolov5_ros_generate_messages_cpp
[ 91%] Built target yolov5_ros_generate_messages_lisp
[ 92%] Generating C++ code from turtlebot3_example/Turtlebot3Goal.msg
[ 92%] Generating C++ code from turtlebot3_example/Turtlebot3Result.msg
[ 93%] Built target yolov5_ros_generate_messages_nodejs
[ 95%] Built target yolov5_ros_generate_messages_eus
[ 95%] Built target detection_msgs_generate_messages
[ 95%] Built target beginner_tutorials_generate_messages
[ 96%] Generating C++ code from turtlebot3_example/Turtlebot3Feedback.msg
[ 96%] Built target detection_2d_generate_messages
[ 96%] Generating Lisp code from turtlebot3_example/Turtlebot3Feedback.msg
[ 96%] Built target yolov5_ros_generate_messages
[ 96%] Built target turtlebot3_example_generate_messages_lisp
[ 97%] Generating Python msg _int_.py for turtlebot3_example
[ 97%] Built target turtlebot3_example_generate_messages_cpp
[ 97%] Built target turtlebot3_example_generate_messages_py
[ 97%] Built target turtlebot3_example_generate_messages
[ 97%] Linking CXX executable /home/unicon3/catkin_ws/devel/lib/turtlebot3_slam/flat_world_imu_node
[ 97%] Built target flat_world_imu_node
[ 98%] Linking CXX executable /home/unicon3/catkin_ws/devel/lib/turtlebot3_gazebo/turtlebot3_drive
[ 98%] Built target turtlebot3_drive
[ 98%] Linking CXX executable /home/unicon3/catkin_ws/devel/lib/turtlebot3_bringup/turtlebot3_diagnostics
[ 98%] Built target turtlebot3_diagnostics
[ 98%] Linking CXX executable /home/unicon3/catkin_ws/devel/lib/turtlebot3_fake/turtlebot3_fake_node
[ 98%] Built target turtlebot3_fake_node
unicon3@unicon3:~/catkin_ws$ 

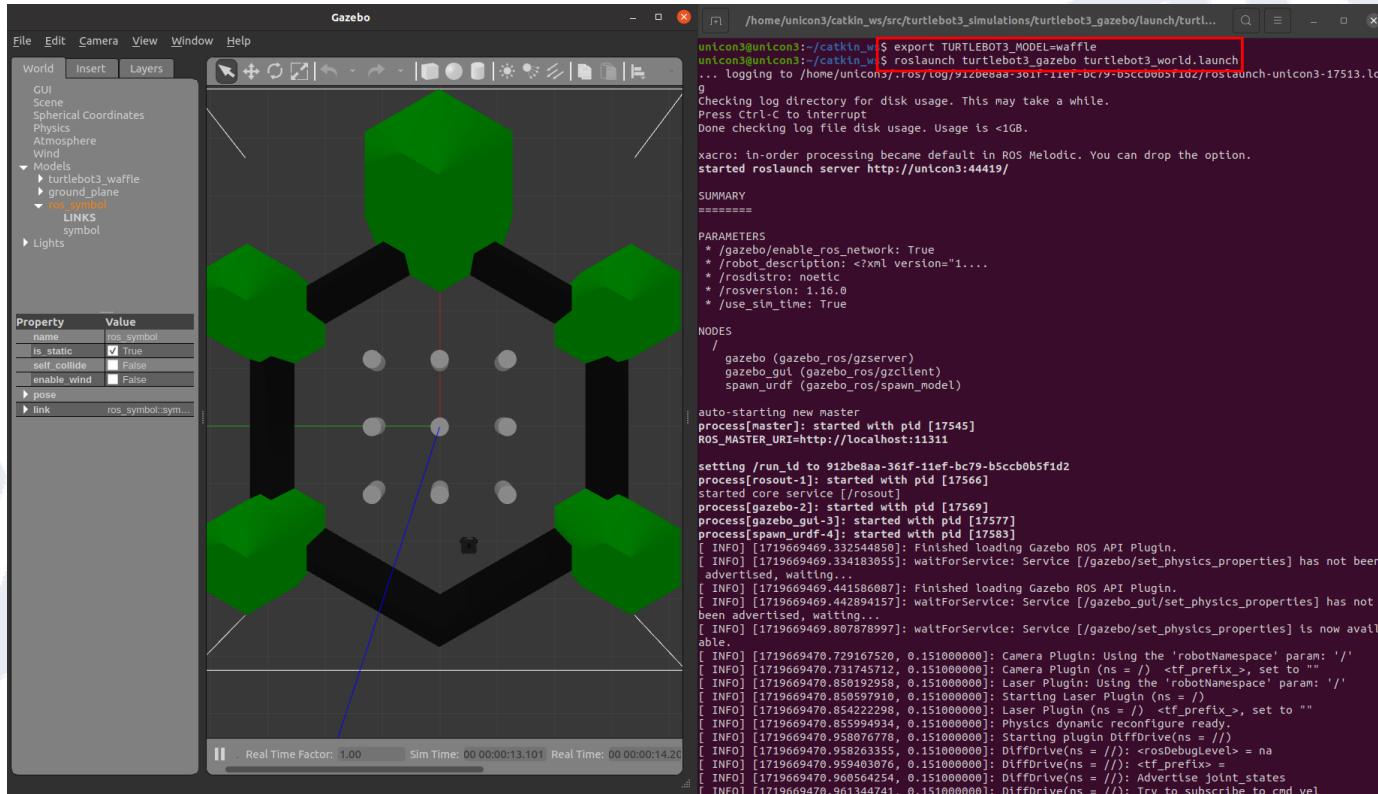
```

\$ cd ~/catkin\_ws && catkin\_make (→ 관련 패키지 다운로드 후, catkin\_ws 빌드)

# 1. SLAM

## 4) ROS 기반 SLAM 실습 [5/19]

### Turtlebot3 모델 정의 및 시뮬레이션 환경 실행



\$ export TURTLEBOT3\_MODEL=burger (또는 \$ export TURTLEBOT3\_MODEL=waffle)

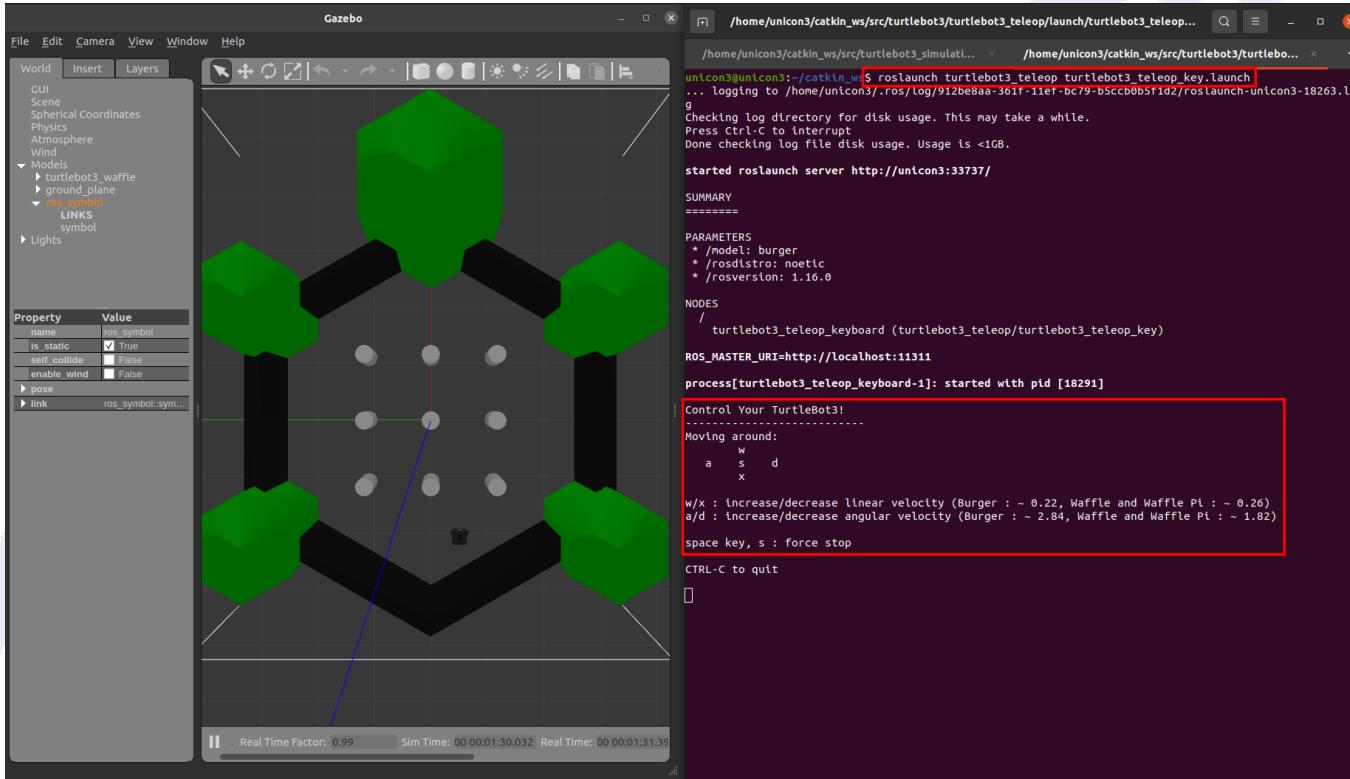
(→ 빌드 후, 환경 실행 전에 사용할 Turtlebot3 모델 정의)

\$ roslaunch turtlebot3\_gazebo turtlebot3\_world.launch (→ Turtlebot3 시뮬레이션 환경 실행)

# 1. SLAM

## 4) ROS 기반 SLAM 실습 [6/19]

### Turtlebot3 모델 정의 및 Turtlebot3 제어기 실행



`$ export TURTLEBOT3_MODEL=burger` (또는 `$ export TURTLEBOT3_MODEL=waffle`)

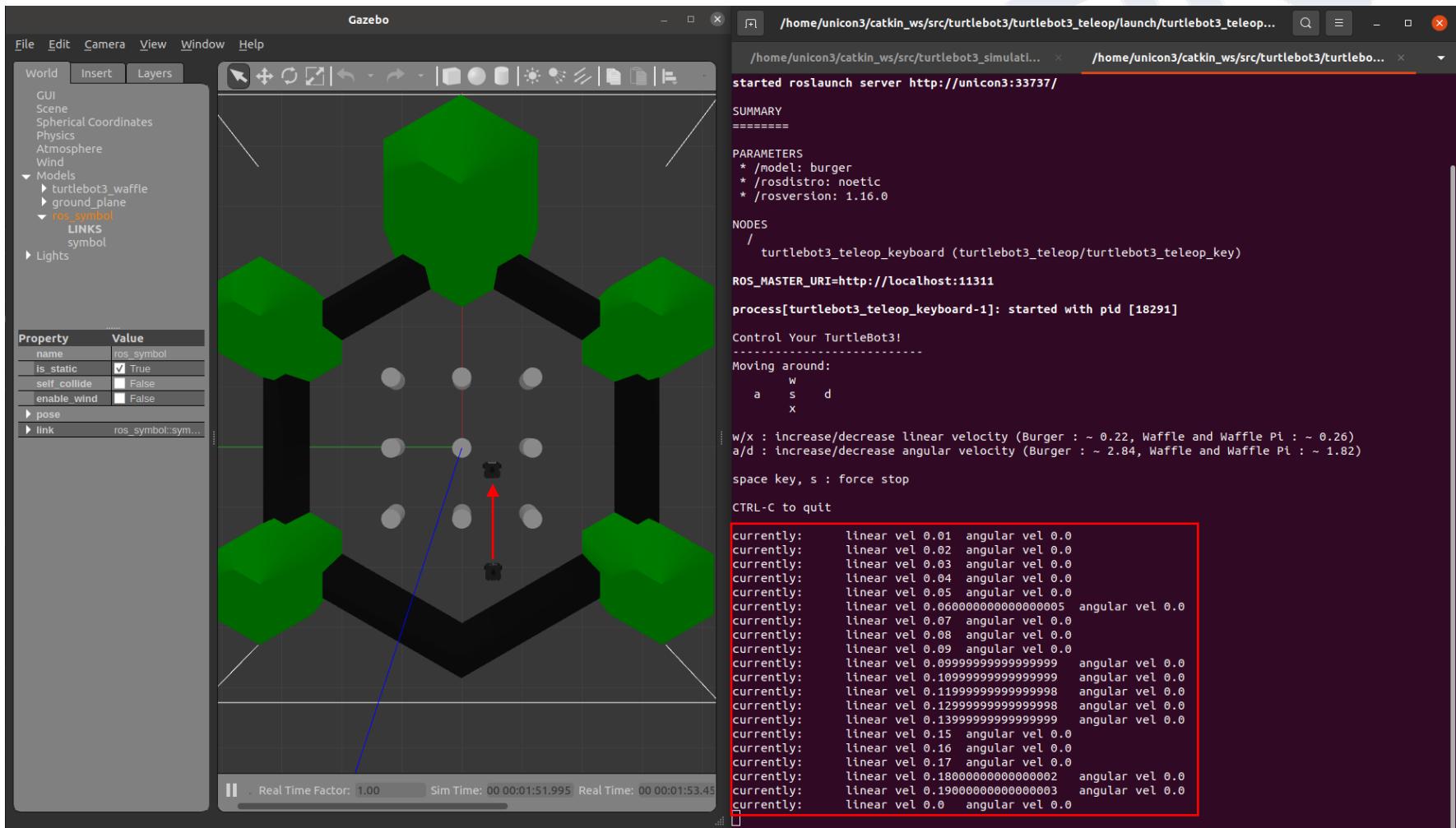
(→ 빌드 후, 환경 실행 전에 사용할 Turtlebot 모델 정의)

`$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch` (→ Turtlebot3 키보드 제어)

# 1. SLAM

## 4) ROS 기반 SLAM 실습 [7/19]

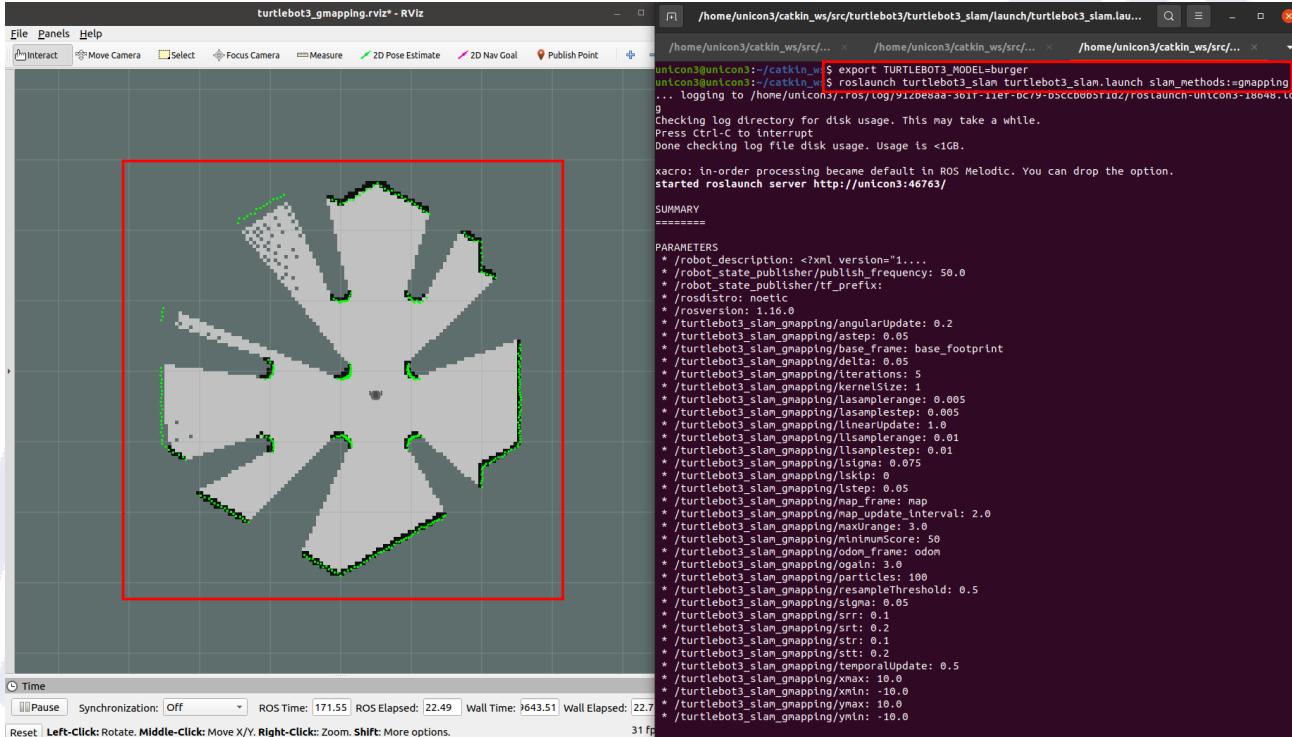
### Turtlebot3 모델 정의 및 Turtlebot3 제어기 실행



# 1. SLAM

## 4) ROS 기반 SLAM 실습 [8/19]

### Turtlebot3 모델 정의 및 SLAM(Gmapping) 실행



\$ export TURTLEBOT3\_MODEL=burger (또는 export TURTLEBOT3\_MODEL=waffle)

(→ 빌드 후, 환경 실행 전에 사용할 Turtlebot3 모델 정의)

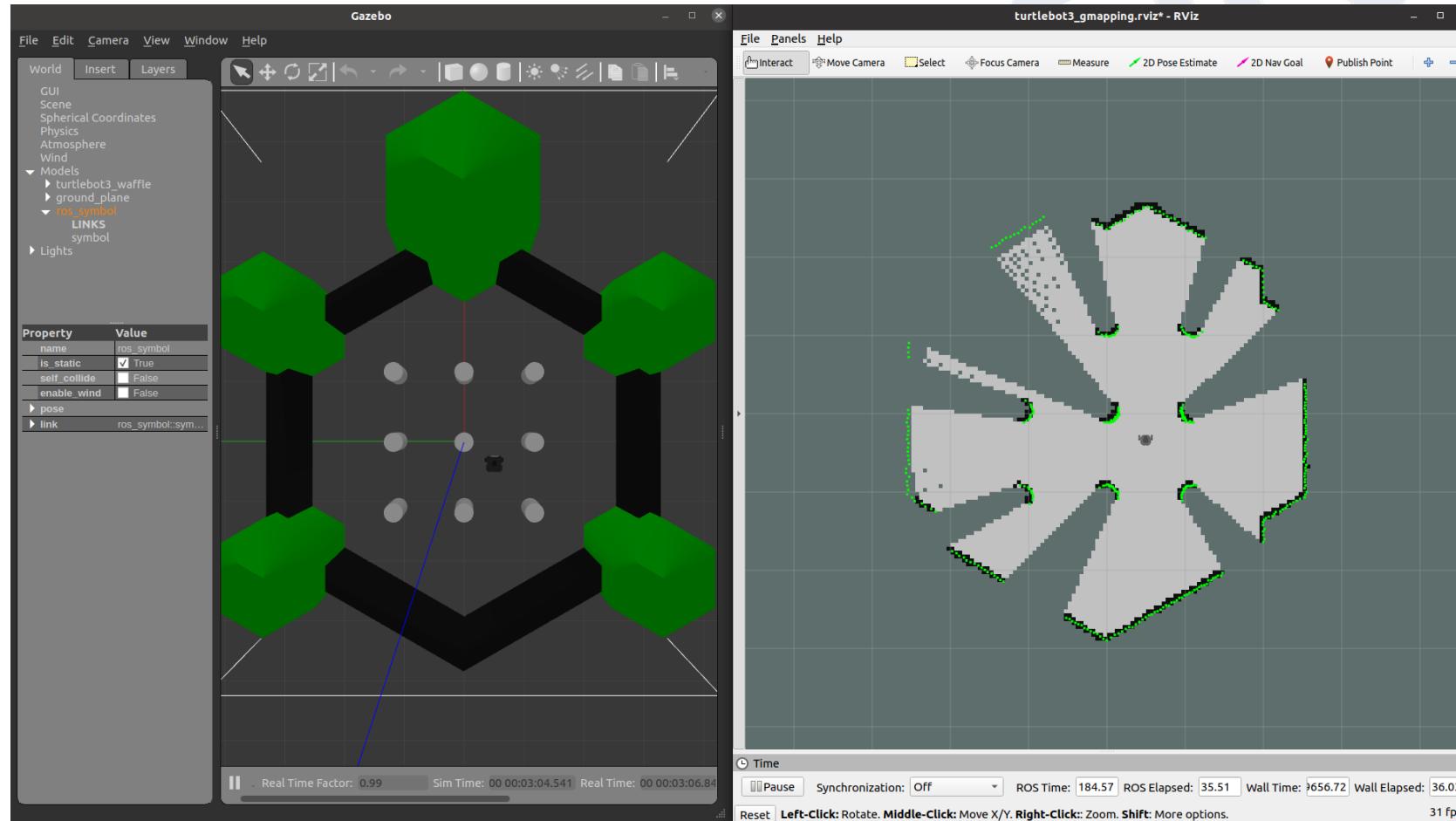
\$ roslaunch turtlebot3\_slam turtlebot3\_slam.launch slam\_methods:=gmapping

(→ Turtlebot3 기반 Gmapping SLAM 실행)

# 1. SLAM

## 4) ROS 기반 SLAM 실습 [9/19]

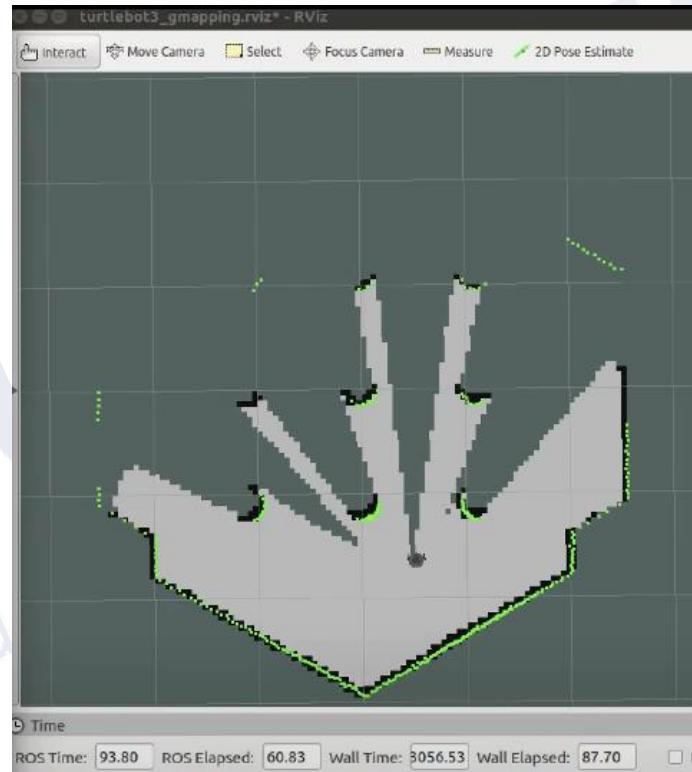
### Turtlebot3 모델 정의 및 SLAM(Gmapping) 실행



# 1. SLAM

## 4) ROS 기반 SLAM 실습 (10/19)

### SLAM(Gmapping) 예시 영상



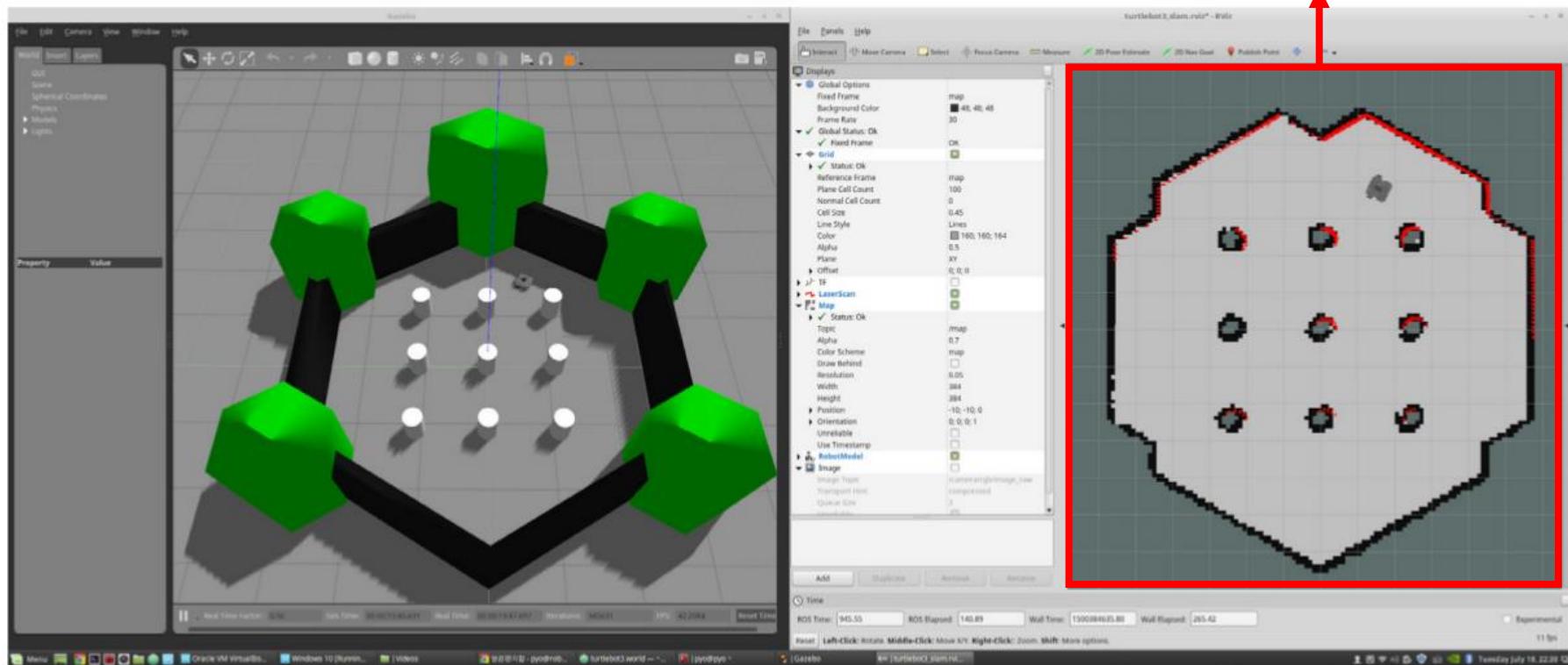
- 위 영상과 같이, Gmapping을 통해 Gazebo simulation 환경에 위치한 Turtlebot3 모델을 직접 제어함으로써 Occupancy Grid Map을 생성.

# 1. SLAM

## 4) ROS 기반 SLAM 실습 (11/19)

SLAM(Gmapping)을 통해 도출한 Map 저장

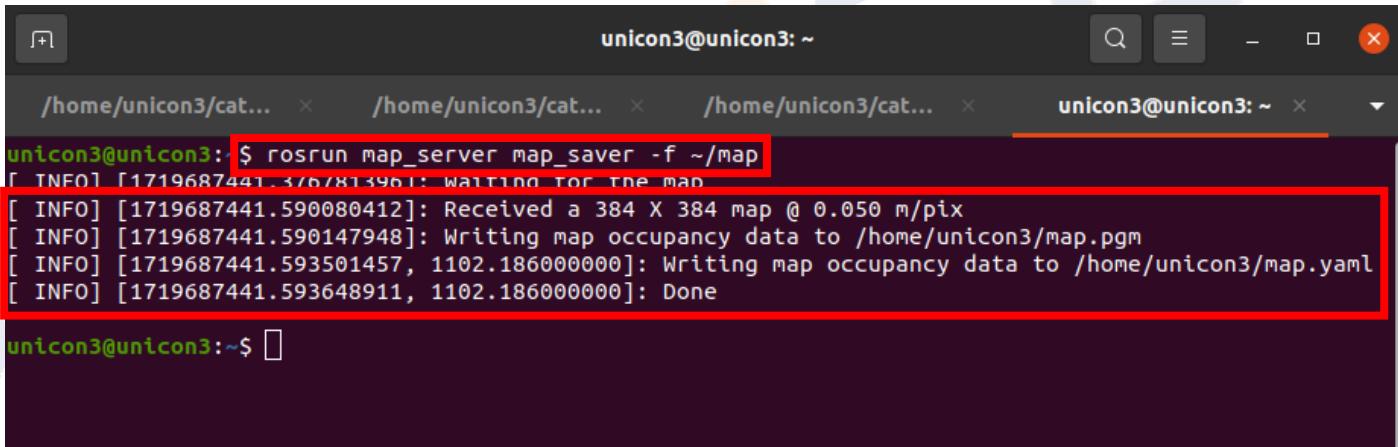
“Occupancy Grid Map”



# 1. SLAM

## 4) ROS 기반 SLAM 실습 (12/19)

SLAM(Gmapping)을 통해 도출한 Map 저장



A terminal window titled "unicon3@unicon3: ~" showing the command \$ rosrun map\_server map\_saver -f ~/map being run. The output shows the process of saving the map to both a PGM file and a YAML file. The last line of output is "[ INFO] [1719687441.593648911, 1102.186000000]: Done". The entire output block is highlighted with a red rectangle.

```
unicon3@unicon3: $ rosrun map_server map_saver -f ~/map
[ INFO] [1719687441.376781346]: WAITING FOR THE MAP
[ INFO] [1719687441.590080412]: Received a 384 X 384 map @ 0.050 m/pix
[ INFO] [1719687441.590147948]: Writing map occupancy data to /home/unicon3/map.pgm
[ INFO] [1719687441.593501457, 1102.186000000]: Writing map occupancy data to /home/unicon3/map.yaml
[ INFO] [1719687441.593648911, 1102.186000000]: Done
```

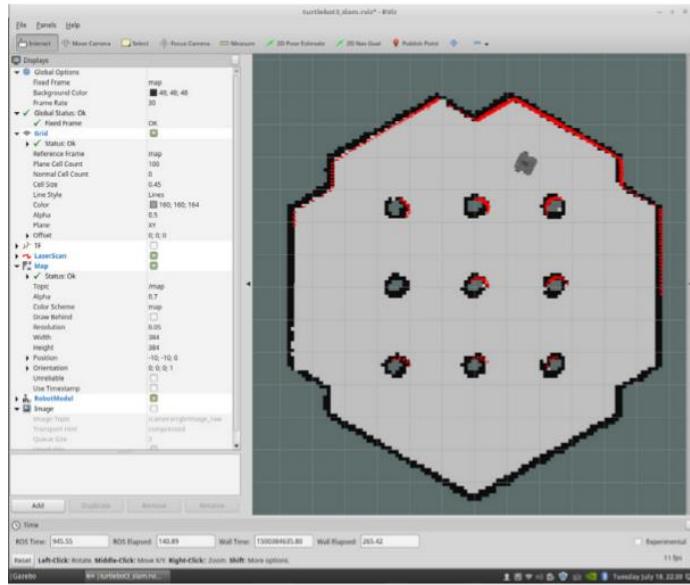
\$ rosrun map\_server map\_saver -f ~/map

(→ Gmapping을 기반으로 생성한 Occupancy Grid Map을 “map.pgm” 파일로 저장)

# 1. SLAM

## 4) ROS 기반 SLAM 실습 (13/19)

### Occupancy Grid Map



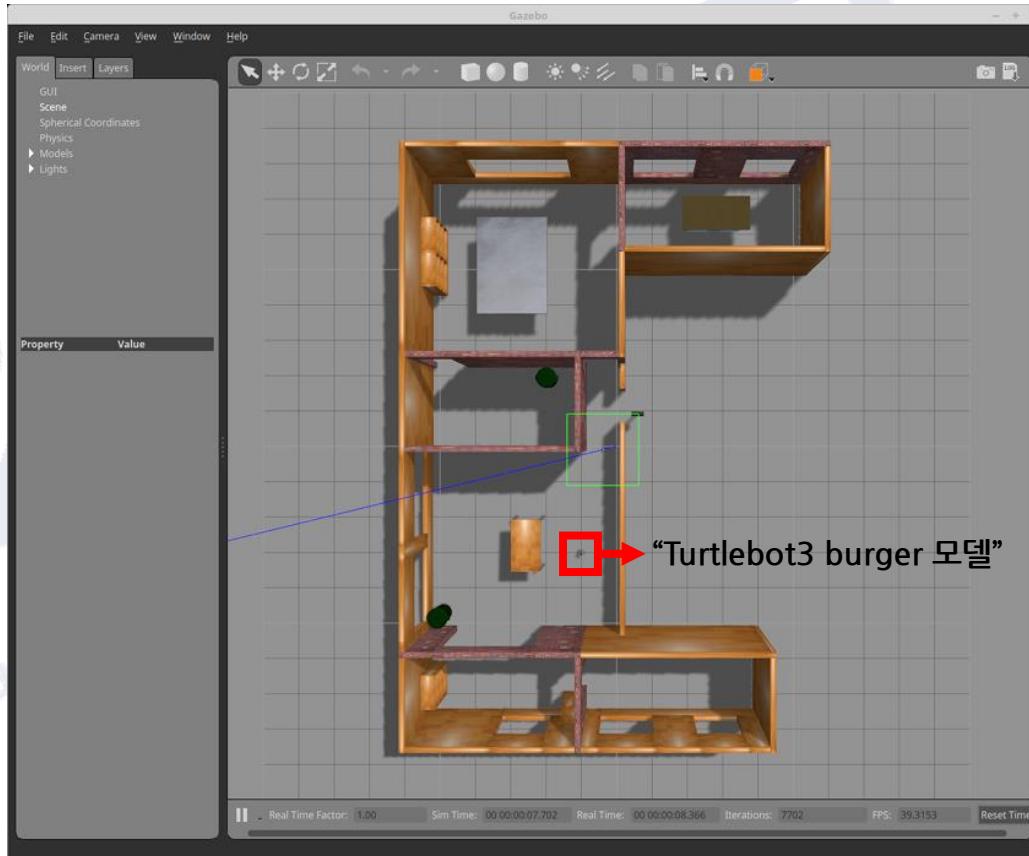
- Occupancy Grid Map은 이산적인 셀로 구성된 지도로, 각 셀은 특정 공간의 점유 상태를 나타냄.
- 각 셀은 비어 있을 확률과 점유되어 있을 확률을 가지고 있으며, 이는 센서의 불확실성과 노이즈를 반영한 것.
- SLAM 과정에서 지도는 지속적으로 업데이트되며, 새로운 센서 데이터가 수집될 때마다 수정 및 개선됨.
- Occupancy Grid Map을 구성하는 하나의 셀은 0부터 100까지의 정수 값으로 이루어짐.
- 0은 해당 셀이 완전한 **free space**임을 나타내며, 해당 셀 영역에 장애물이 없다는 것을 의미.  
100은 해당 셀이 완전히 점유되어 있음을 나타내며, 해당 셀 영역에 장애물이 있다는 것을 의미.  
-1 또는 **unknown**은 해당 셀의 상태가 알려지지 않았음을 나타내며, 로봇은 해당 셀 영역의 정보를 취득하지 못했거나 셀에 대한 확신이 없는 경우 해당 값을 사용함.

# 1. SLAM

## 4) ROS 기반 SLAM 실습 (14/19)

### ROS 기반 SLAM(Gmapping) 실습

➤ 아래의 Turtlebot3 환경에서 Gmapping을 통한 Occupancy Grid Map 생성 및 저장하기.

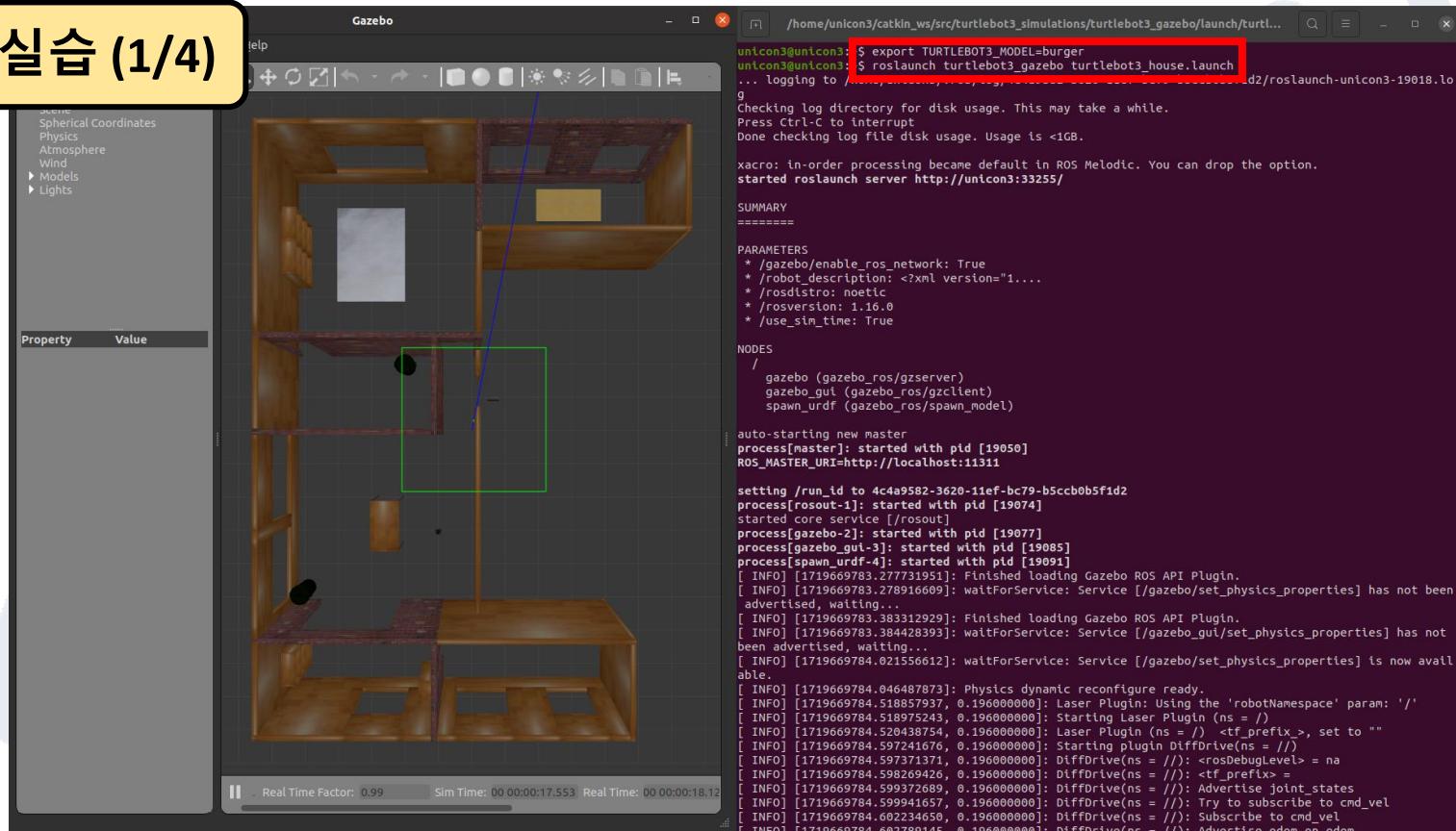


```
$ roslaunch turtlebot3_gazebo turtlebot3_house.launch
```

# 1. SLAM

## 4) ROS 기반 SLAM 실습 (15/19)

실습 (1/4)



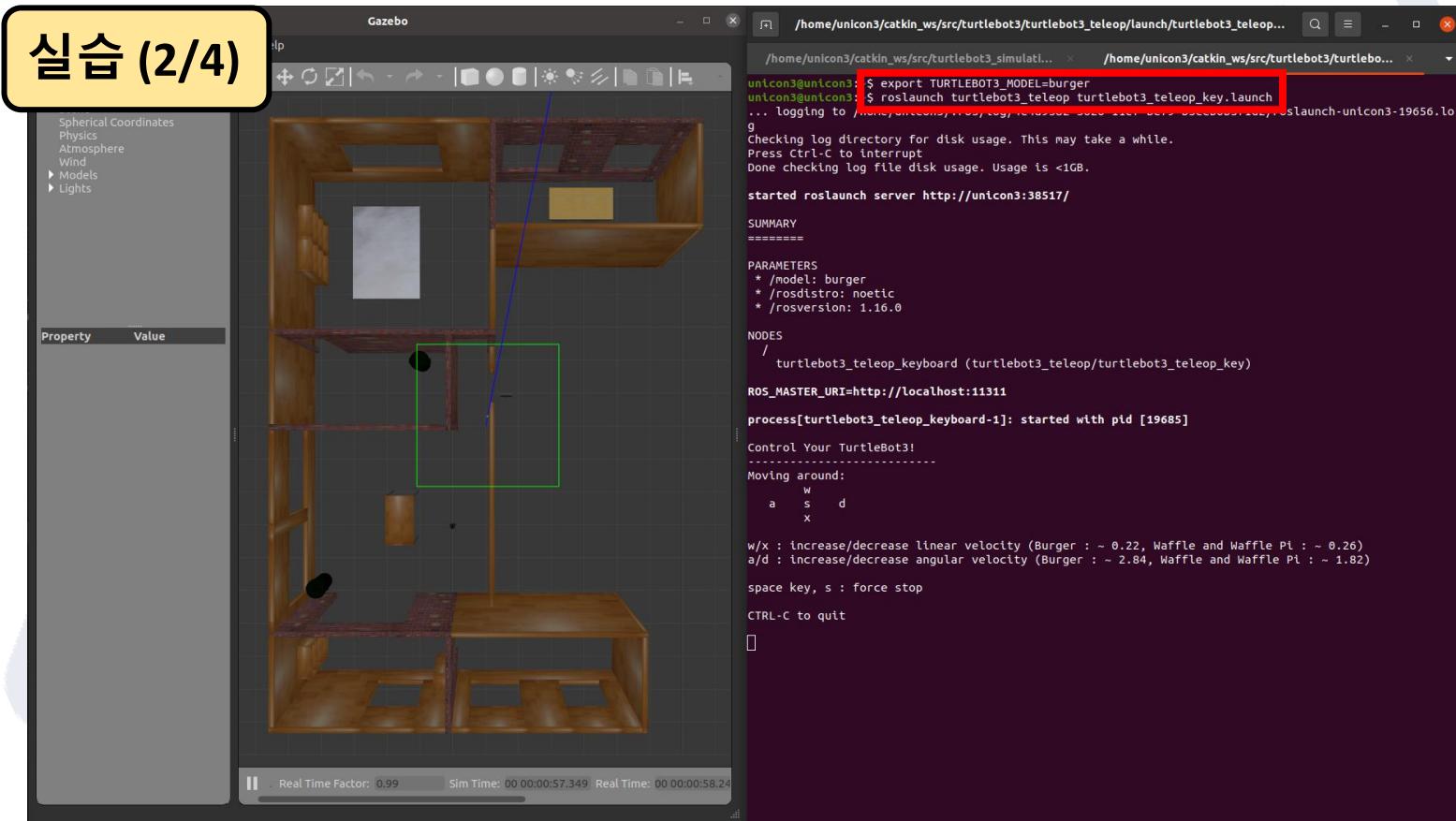
**\$ export TURTLEBOT3\_MODEL=burger (또는 \$ export TURTLEBOT3\_MODEL=waffle)**

(→ Turtlebot3 시뮬레이션 환경 실행 전, 사용할 Turtlebot3 모델 정의)

**\$ roslaunch turtlebot3\_gazebo turtlebot3\_house.launch** (→ 또 다른 Turtlebot3 환경 실행)

# 1. SLAM

## 4) ROS 기반 SLAM 실습 (16/19)



`$ export TURTLEBOT3_MODEL=burger` (또는 `$ export TURTLEBOT3_MODEL=waffle`)

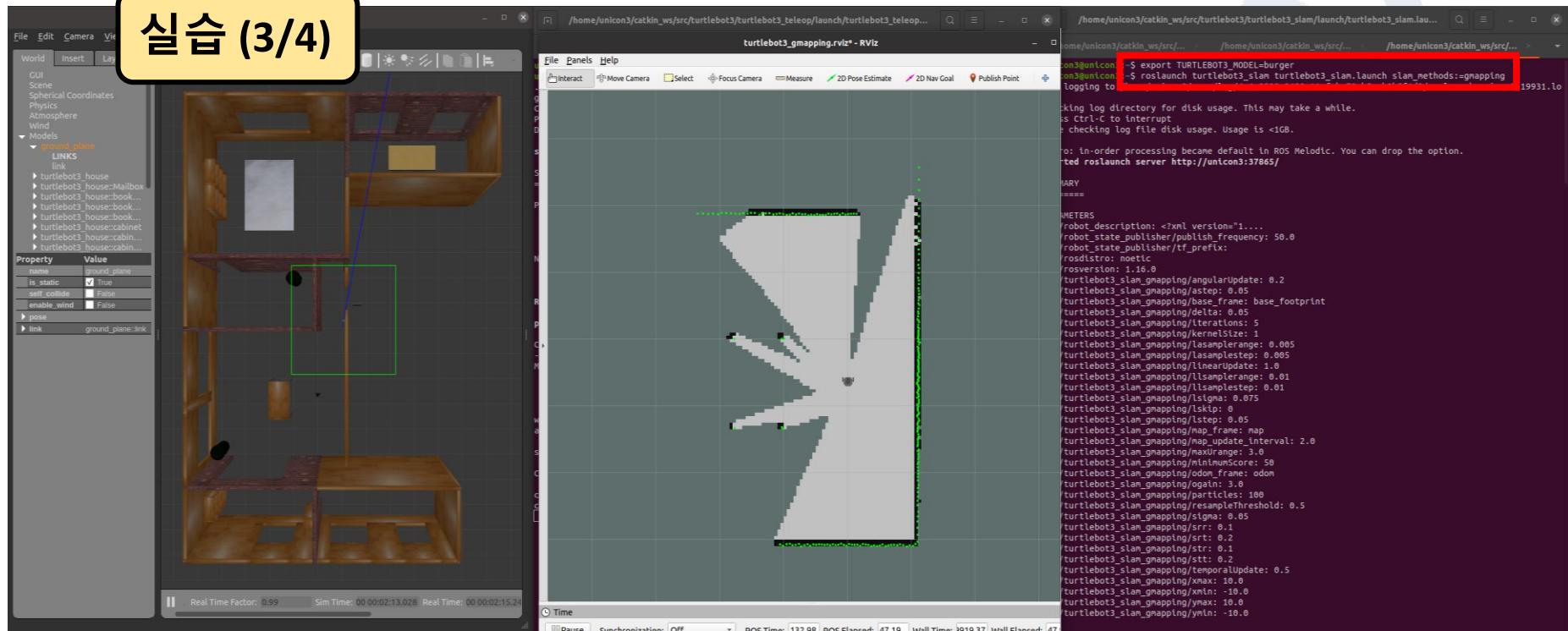
(→ Turtlebot3 시뮬레이션 환경에서 사용할 Turtlebot3 모델 정의)

`$ rosrun turtlebot3_teleop turtlebot3_teleop_key.launch` (→ Turtlebot3 키보드 제어)

# 1. SLAM

## 4) ROS 기반 SLAM 실습 (17/19)

실습 (3/4)



`$ export TURTLEBOT3_MODEL=burger` (또는 `$ export TURTLEBOT3_MODEL=waffle`)

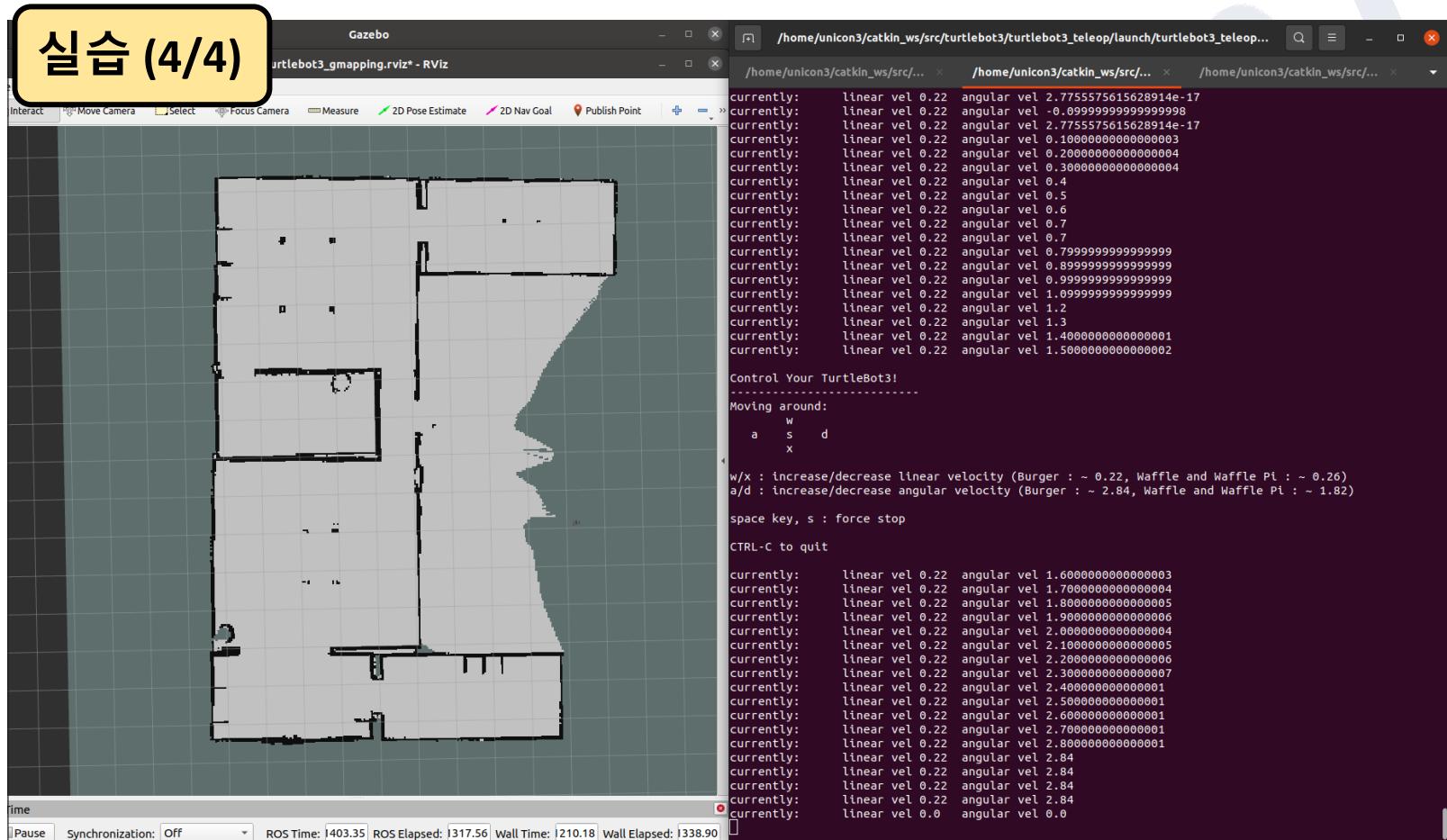
(→ 빌드 후, 환경 실행 전에 사용할 Turtlebot3 모델 정의)

`$ roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping`

(→ Turtlebot3 기반 Gmapping SLAM 실행)

# 1. SLAM

## 4) ROS 기반 SLAM 실습 (18/19)



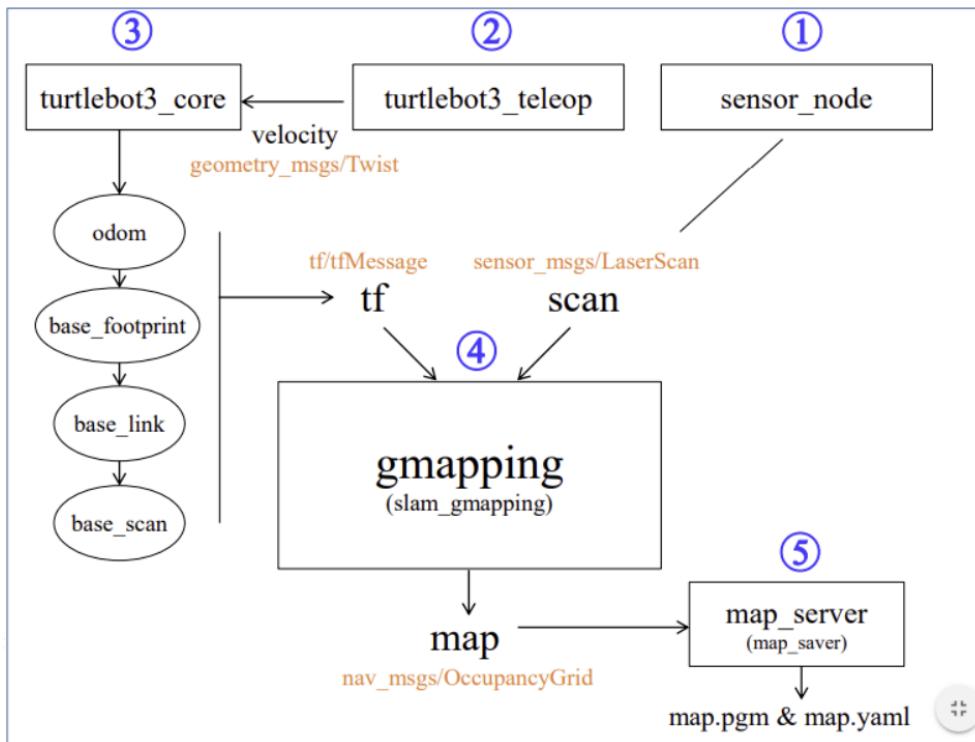
\$ rosrun map\_server map\_saver -f ~/map2

(→ Gmapping을 기반으로 생성한 Occupancy Grid Map을 “map2.pgm” 파일로 저장)

# 1. SLAM

## 4) ROS 기반 SLAM 실습 (19/19)

### Gmapping 구조



Gmapping	
개발사	OpenSLAM
라이선스	BSD-3-Clause
특징	2D 개발 지원 중단 Static한 환경에 유리
입력 데이터	Odometry 데이터 Laser scan 데이터 IMU 데이터
출력 데이터	Occupancy Grid Map 데이터

- Gmapping은 로봇이 얼마나 이동했는지를 추정하는 주행 데이터인 Odometry 데이터와 2D LiDAR를 통한 Laser scan 데이터, 그리고 Odometry의 추정 정확도를 향상시키기 위한 IMU 데이터를 입력으로 받으며, Occupancy Grid Map을 출력으로 도출함.

# 1. SLAM

## 5) GIMP 기반 Occupancy Grid Map 보정 (1/27)

GIMP란?



- GIMP(GNU Image Manipulation Program)는 GNU/Linux, macOS, Windows 등 여러 운영체제에서 사용할 수 있는 크로스 플랫폼 이미지 편집기.
- GIMP 편집기를 통해 이미지를 원하는 대로 편집 가능.
- 앞서 SLAM을 통해 도출한 Occupancy Grid Map을 GIMP 편집기를 사용하여 보정, 병합 등 다양하게 편집 가능

# 1. SLAM

## 5) GIMP 기반 Occupancy Grid Map 보정 (2/27)

### GIMP 편집기를 통한 보정 시 이점

- SLAM을 수행하였던 기존 환경이 변하는 경우, 재 Mapping을 할 필요 없이 간단한 편집 과정을 통해 Occupancy Grid Map을 수정 가능.
- SLAM을 수행하여 도출한 Occupancy Grid Map 중, 일부 지역으로 mobility가 이동하지 못하도록 편집함으로써 위험을 사전에 방지 가능.
- SLAM을 통해 도출한 Occupancy Grid Map이 완전히 Mapping 되지 않아도, 보정 작업을 통해 어느 정도 완전한 Map이 될 수 있도록 편집 가능

# 1. SLAM

## 5) GIMP 기반 Occupancy Grid Map 보정 (3/27)

### GIMP 편집기 설치 (in Ubuntu Env)

```
$ sudo apt-get update
```

```
$ sudo add-apt-repository ppa:otto-kesselgulasch/gimp-edge
```

```
$ sudo apt-get update
```

```
$ sudo apt install gimp
```

# 1. SLAM

## 5) GIMP 기반 Occupancy Grid Map 보정 (4/27)

### GIMP 편집기 설치 (in Ubuntu Env)

- ❖ 만약 설치 과정에서 아래와 같은 오류 메시지가 나온다면,

```
unicorn3@unicorn3:~$ sudo apt install gimp
Reading package lists... Done
Building dependency tree
Reading state information... Done
Some packages could not be installed. This may mean that you have
requested an impossible situation or if you are using the unstable
distribution that some required packages have not yet been created
or been moved out of Incoming.
The following information may help to resolve the situation:

The following packages have unmet dependencies:
 gimp : Depends: libgimp2.0 (>= 2.10.18) but it is not going to be installed
        Depends: libgimp2.0 (<= 2.10.18-2) but it is not going to be installed
        Depends: libgegl-0.4-0 (>= 0.4.22) but it is not going to be installed
E: Unable to correct problems, you have held broken packages.
```

\$ sudo apt install ppa-purge && sudo ppa-purge ppa:otto-kesselgulasch/gimp (수행 후)

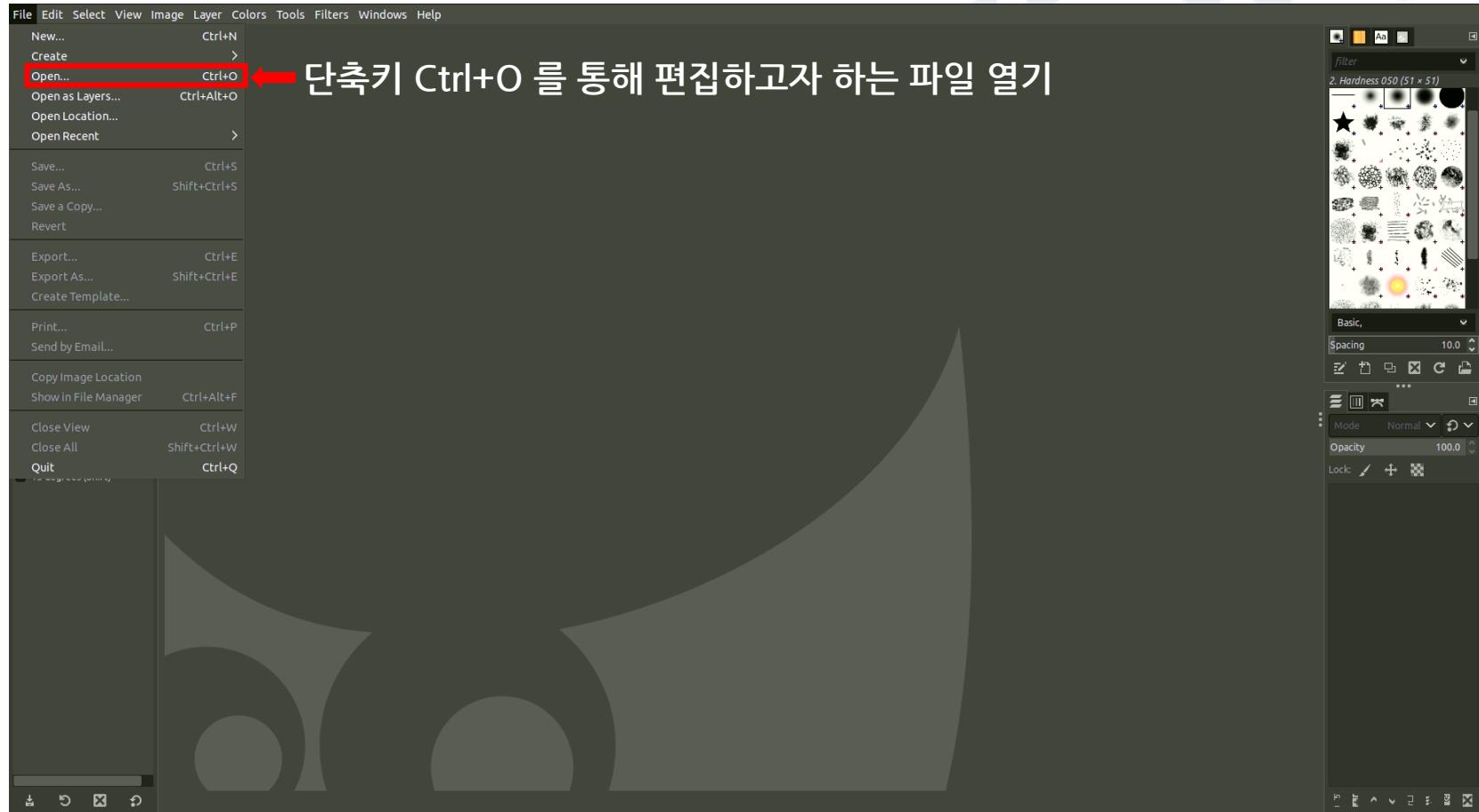
\$ sudo apt install gimp (재 수행)

\$ gimp (GIMP 실행)

# 1. SLAM

## 5) GIMP 기반 Occupancy Grid Map 보정 (5/27)

### GIMP 편집기 사용 방법 - Map 파일 불러오기

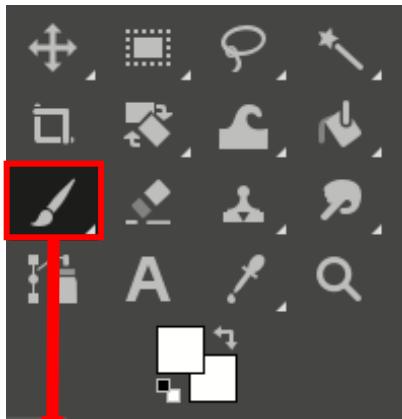


# 1. SLAM

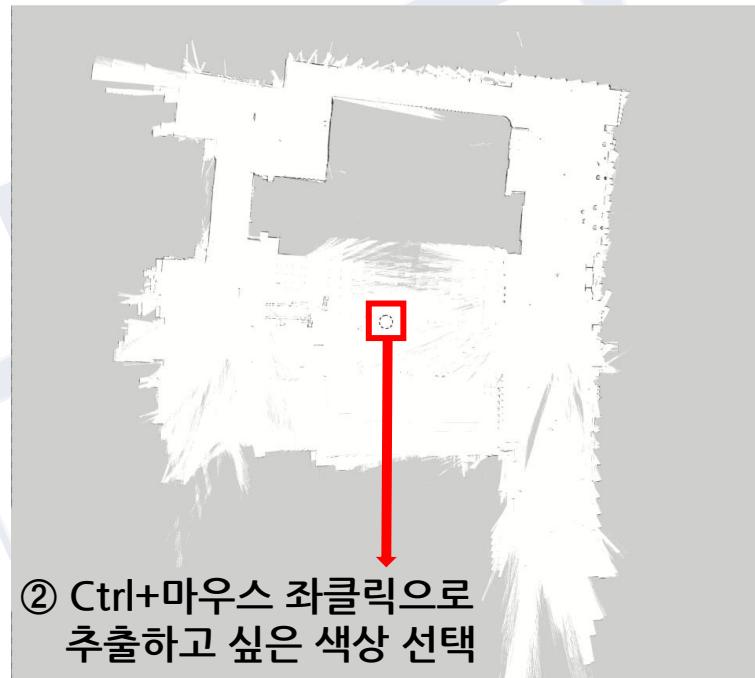
## 5) GIMP 기반 Occupancy Grid Map 보정 [6/27]

### GIMP 편집기 사용 방법 - Map 보정 (1개 이상의 Map 수정)

- 만약 앞선 과정들을 수행하였음에도 추가적인 Map 보정이 필요할 경우



① 단축키 P를 통해  
Paintbrush Tool을 선택  
**(Map 상의 원하는 색 추출)**



② Ctrl+마우스 좌클릭으로  
추출하고 싶은 색상 선택

③ 색상 추출 후 보정하고자 하는 부분을 마우스 좌 클릭을 누른 채 드래그 하거나,

마우스 좌클릭을 반복적으로 수행하여 보정 수행.

(만약, 직선 보정 수행을 위해서는 직선 시작 지점을 **마우스 좌 클릭** 후,

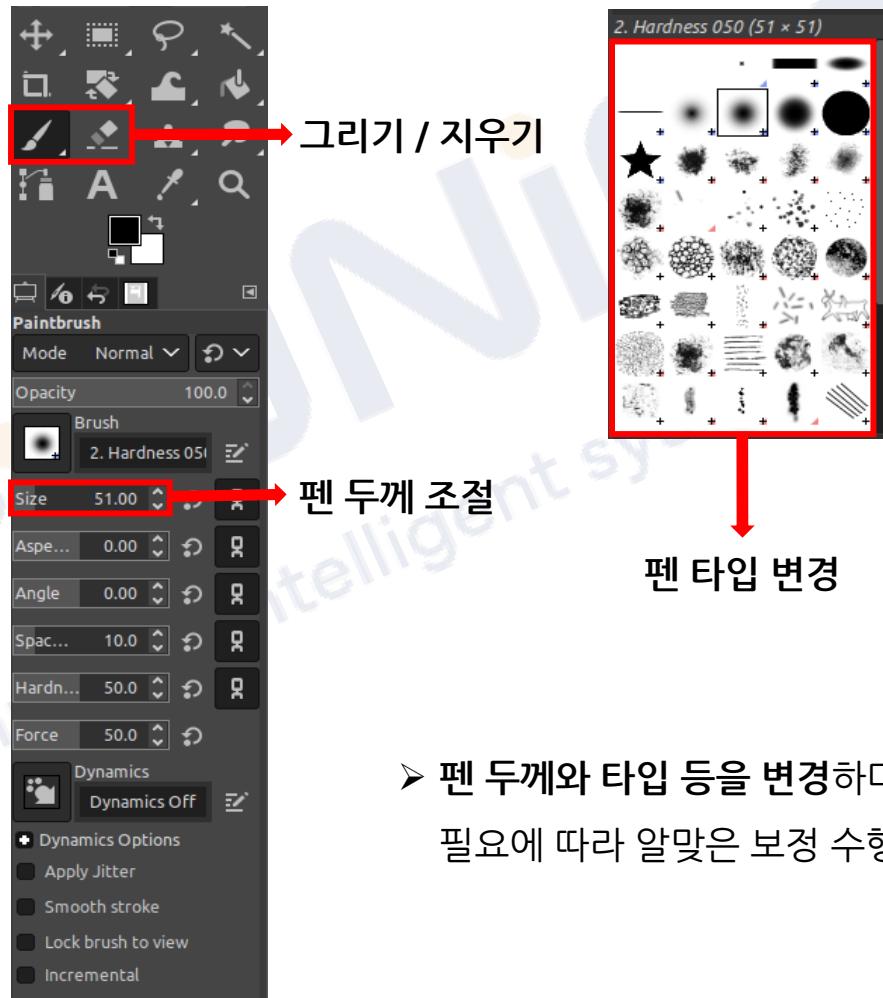
도착 지점에 **Shift+마우스 좌 클릭**을 수행하면 일직선의 보정을 수행)

41

# 1. SLAM

## 5) GIMP 기반 Occupancy Grid Map 보정 (7/27)

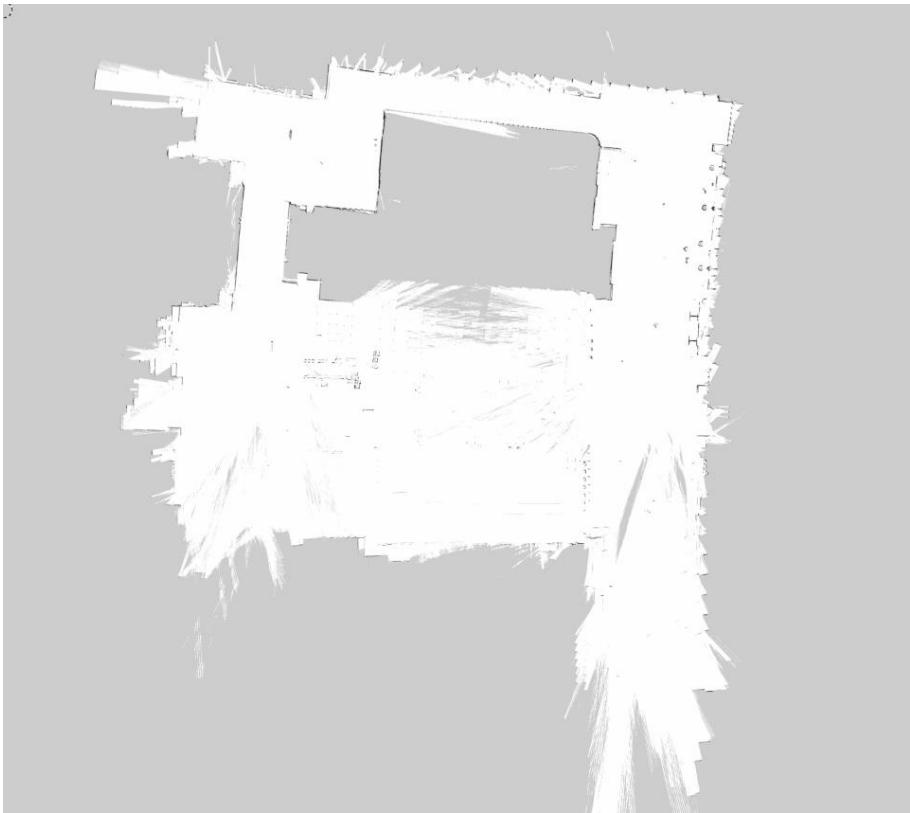
GIMP 편집기 사용 방법 - Map 보정 (1개 이상의 Map 수정)



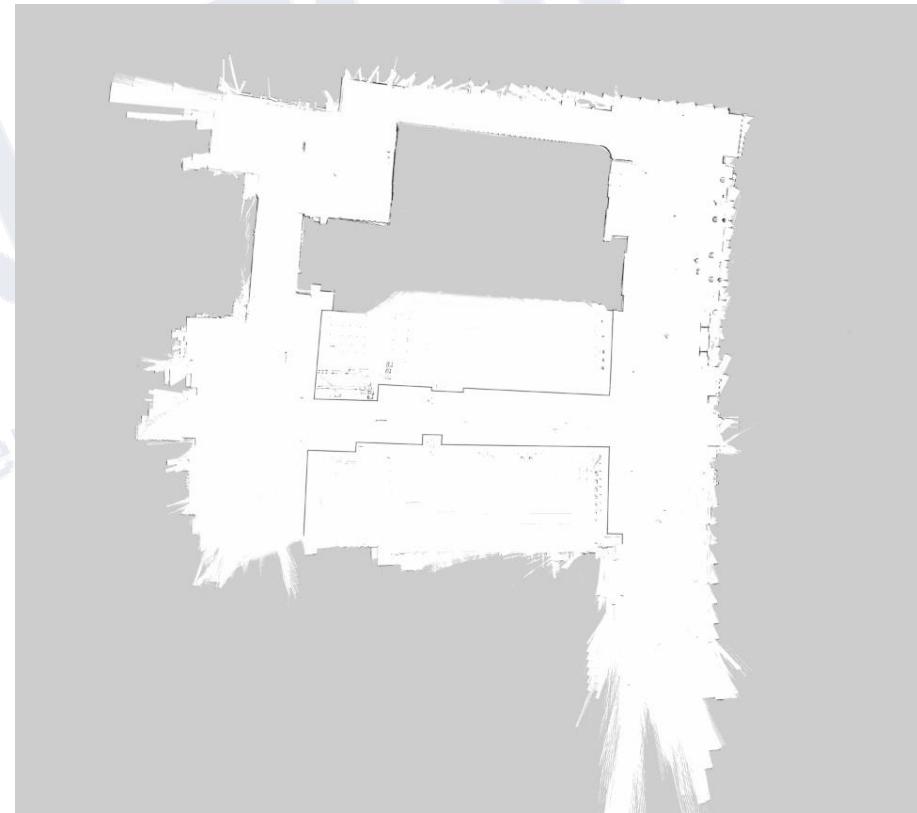
# 1. SLAM

## 5) GIMP 기반 Occupancy Grid Map 보정 [8/27]

GIMP 편집기 사용 방법 - Map 보정 (1개 이상의 Map 수정)



보정 전



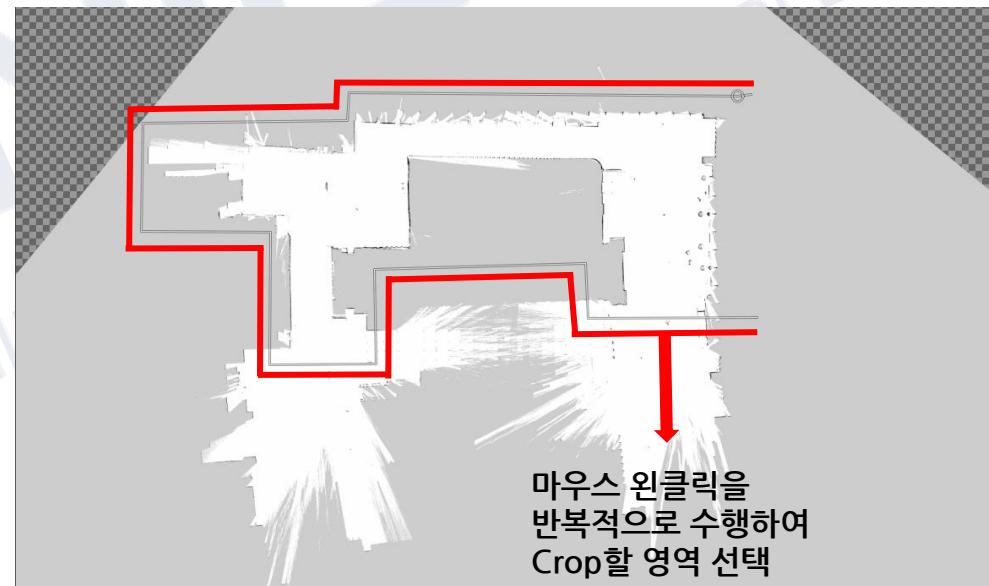
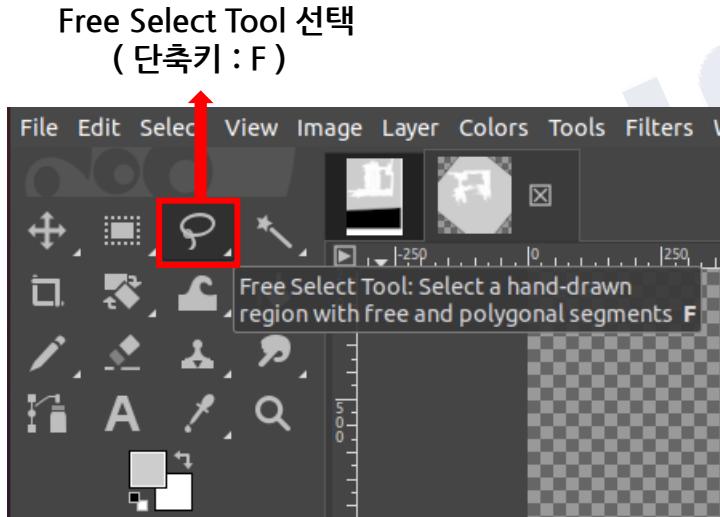
보정 후

# 1. SLAM

## 5) GIMP 기반 Occupancy Grid Map 보정 [9/27]

### GIMP 편집기 사용 방법 - Map 보정 (1개 이상의 Map 수정)

➤ 보다 수월한 Map 편집을 위해, 이미지를 Crop하여 복사 및 붙여 넣는 방법

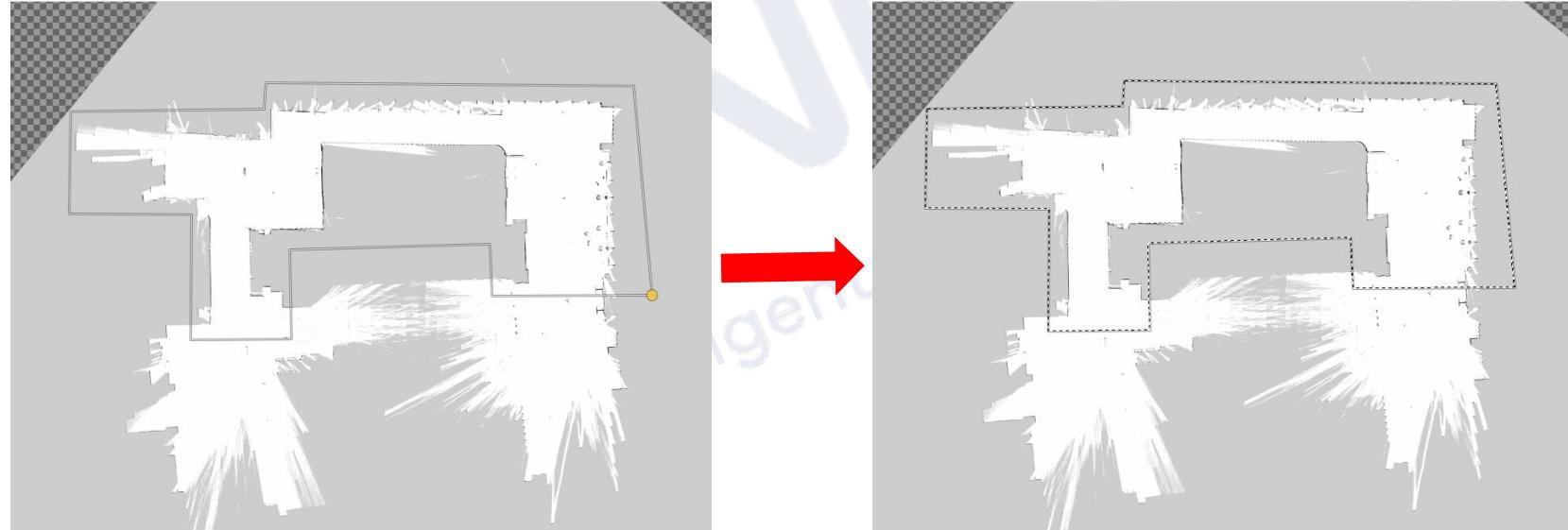


# 1. SLAM

## 5) GIMP 기반 Occupancy Grid Map 보정 (10/27)

### GIMP 편집기 사용 방법 - Map 보정 (1개 이상의 Map 수정)

- Crop할 영역이 완성되면, 오른쪽 그림과 같이 영역이 점선으로 나타나게 됨.

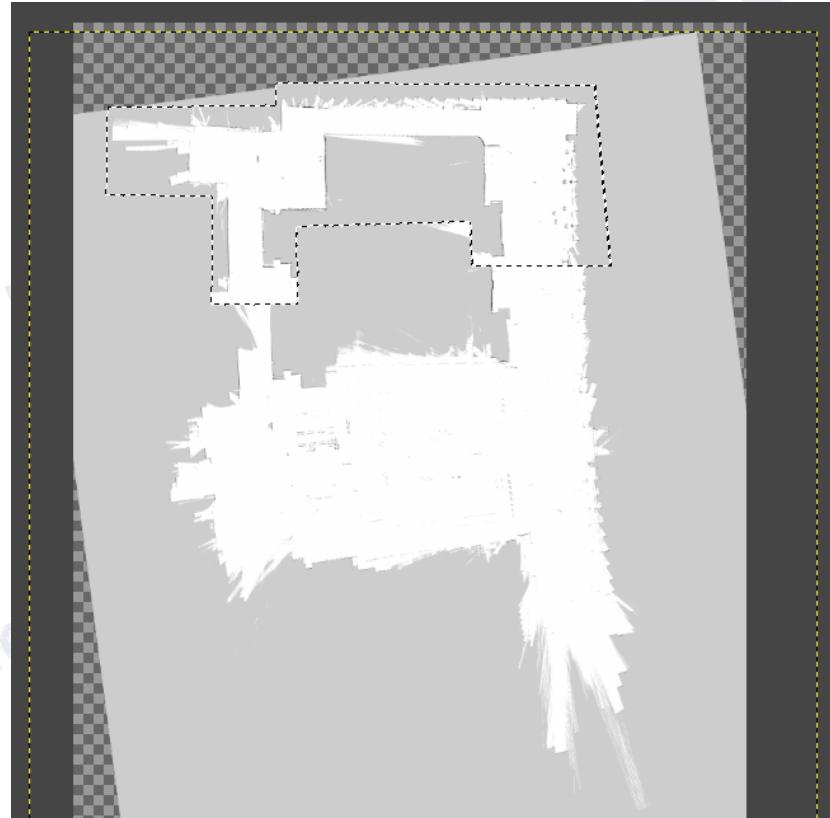


- 단축키 Ctrl + C를 통해 점선으로 둘러싸인 영역을 복사

# 1. SLAM

## 5) GIMP 기반 Occupancy Grid Map 보정 (11/27)

GIMP 편집기 사용 방법 - Map 보정 (1개 이상의 Map 수정)

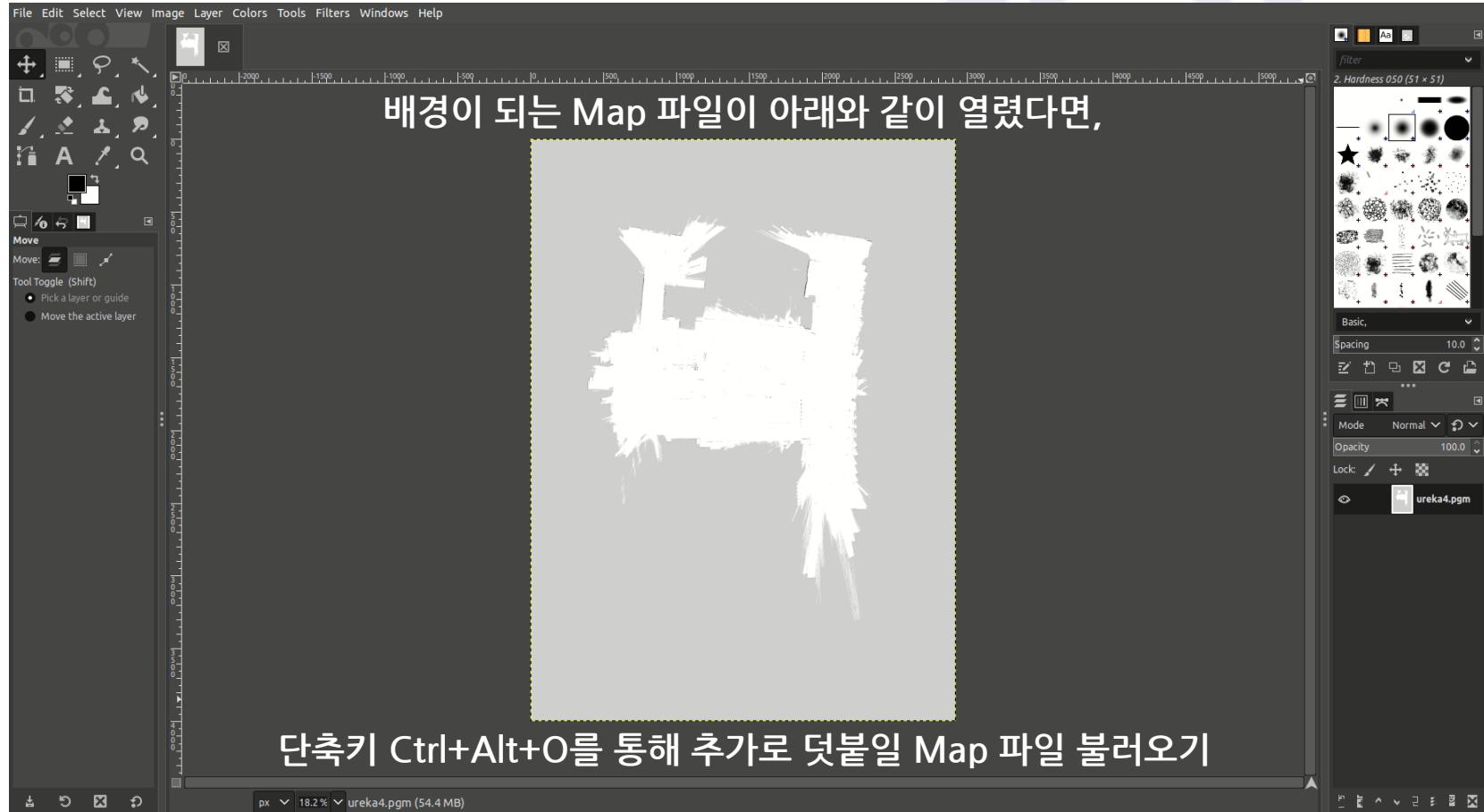


- Ctrl + V를 통해 복사한 영역을, 수정중인 다른 Map에 붙여넣기 수행.

# 1. SLAM

## 5) GIMP 기반 Occupancy Grid Map 보정 (12/27)

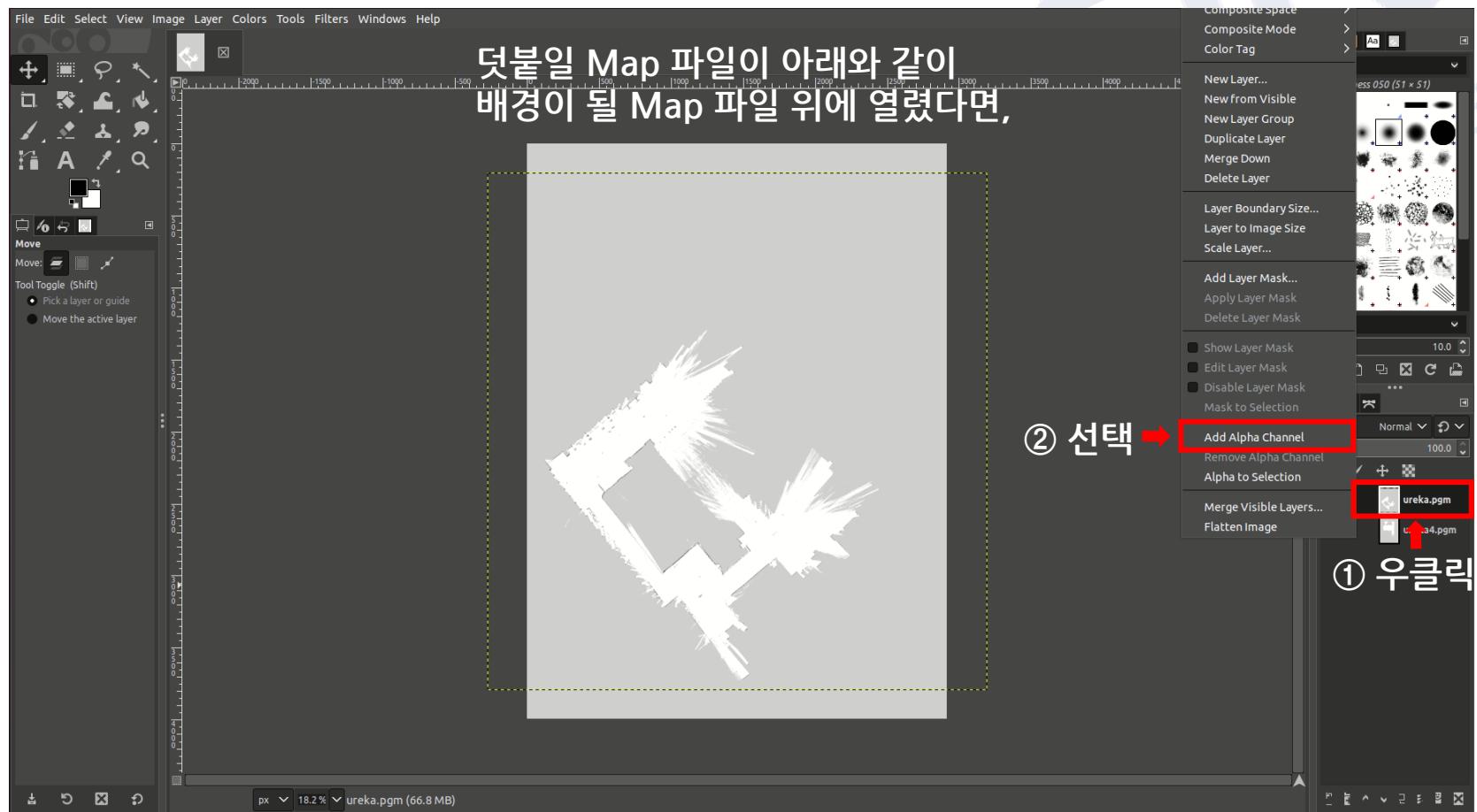
GIMP 편집기 사용 방법 - Map 병합 (2개 이상의 Map 병합)



# 1. SLAM

## 5) GIMP 기반 Occupancy Grid Map 보정 (13/27)

### GIMP 편집기 사용 방법 - Map 병합 (2개 이상의 Map 병합)

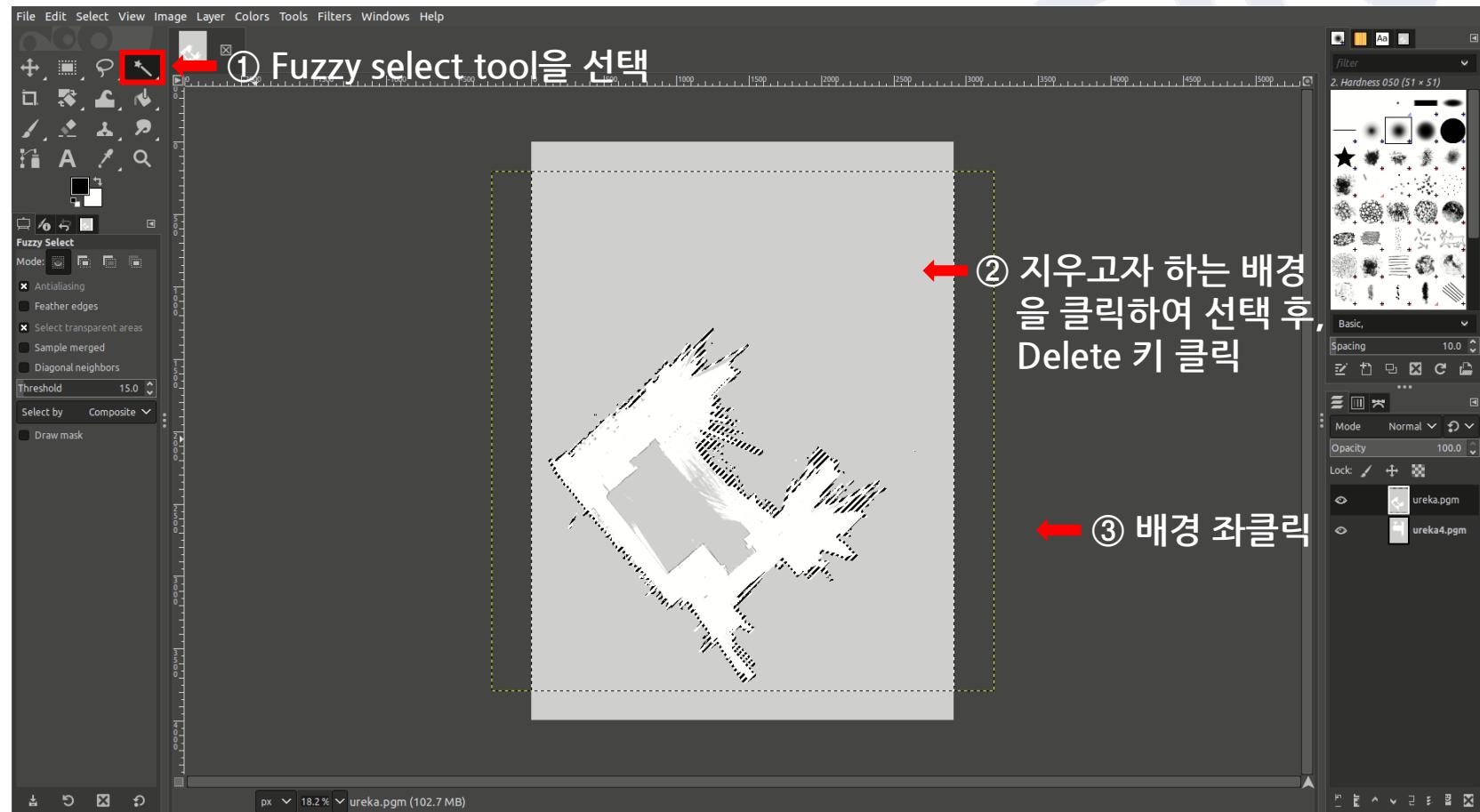


➤ ①~② 과정은 덧붙일 Map 파일의 배경을 제거하기 위한 전 처리 과정

# 1. SLAM

## 5) GIMP 기반 Occupancy Grid Map 보정 (14/27)

### GIMP 편집기 사용 방법 - Map 병합 (2개 이상의 Map 병합)

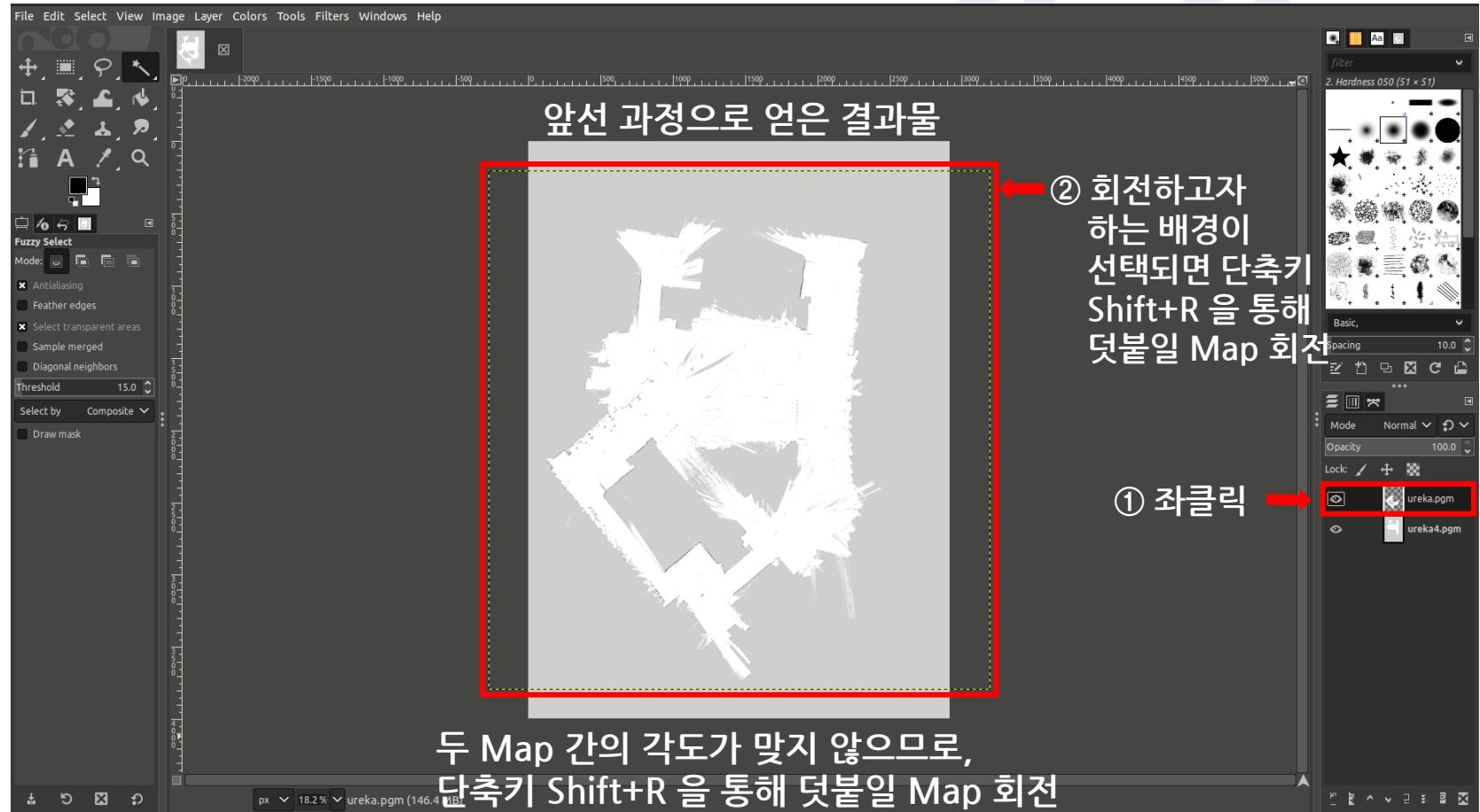


➤ ①~③ 과정을 통해 덧붙일 Map 파일의 배경 제거

# 1. SLAM

## 5) GIMP 기반 Occupancy Grid Map 보정 (15/27)

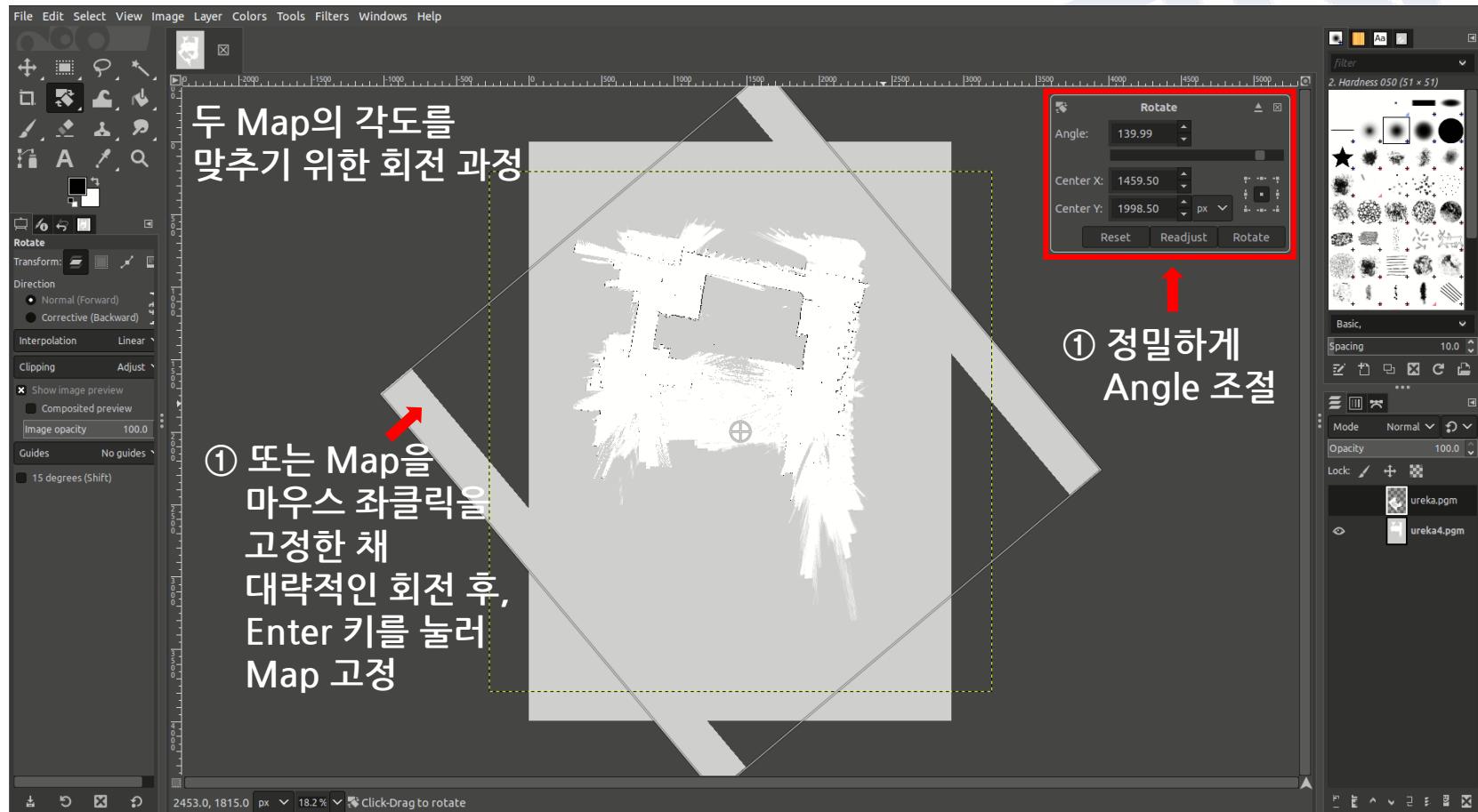
### GIMP 편집기 사용 방법 - Map 병합 (2개 이상의 Map 병합)



# 1. SLAM

## 5) GIMP 기반 Occupancy Grid Map 보정 (16/27)

### GIMP 편집기 사용 방법 - Map 병합 (2개 이상의 Map 병합)



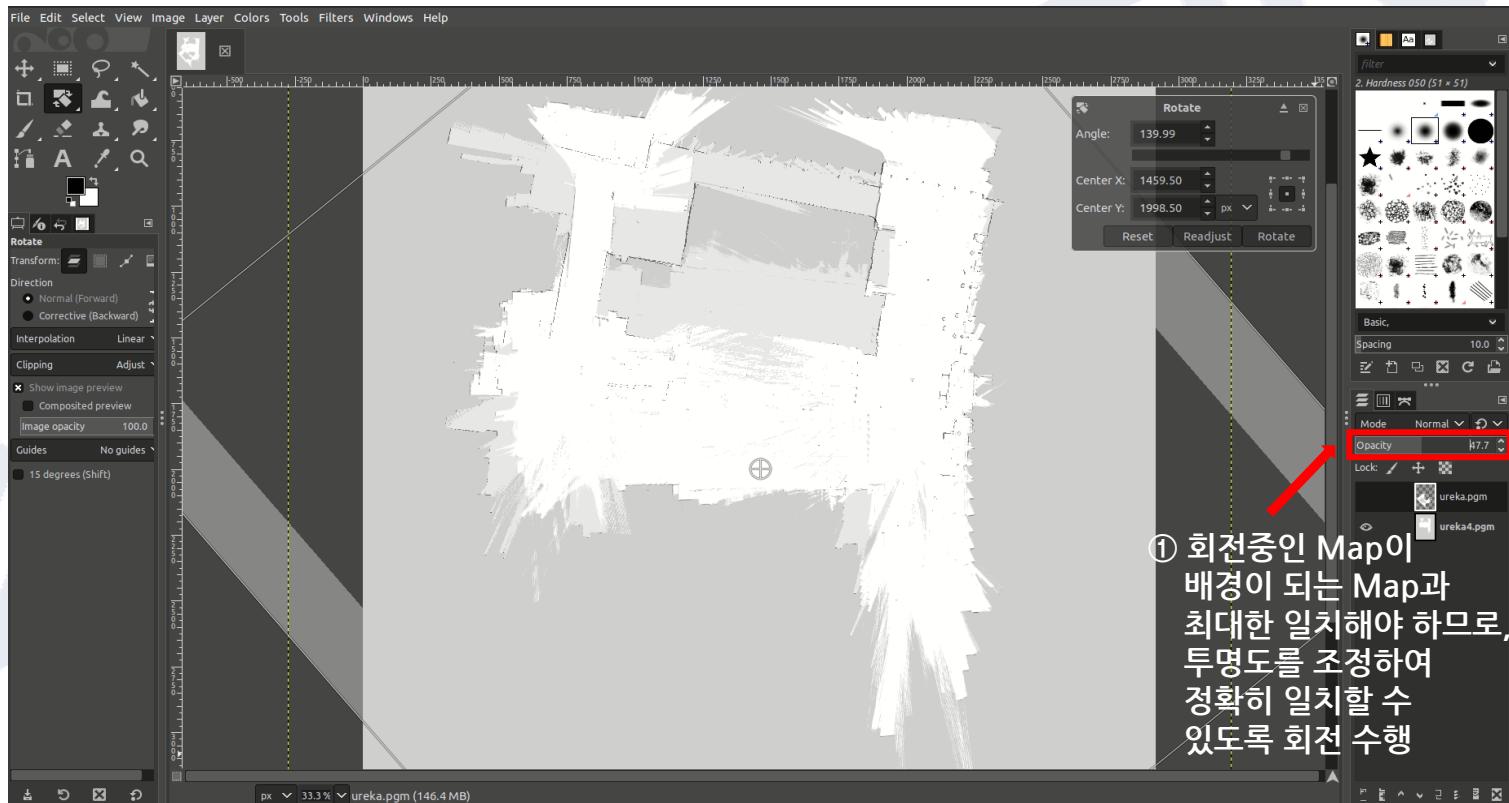
➤ 위 과정을 통해 각도를 맞춘 뒤, Enter 키로 Map을 고정하고

단축키 M을 눌러 마우스로 Map을 드래그하여 이동 가능.

# 1. SLAM

## 5) GIMP 기반 Occupancy Grid Map 보정 (17/27)

### GIMP 편집기 사용 방법 - Map 병합 (2개 이상의 Map 병합)

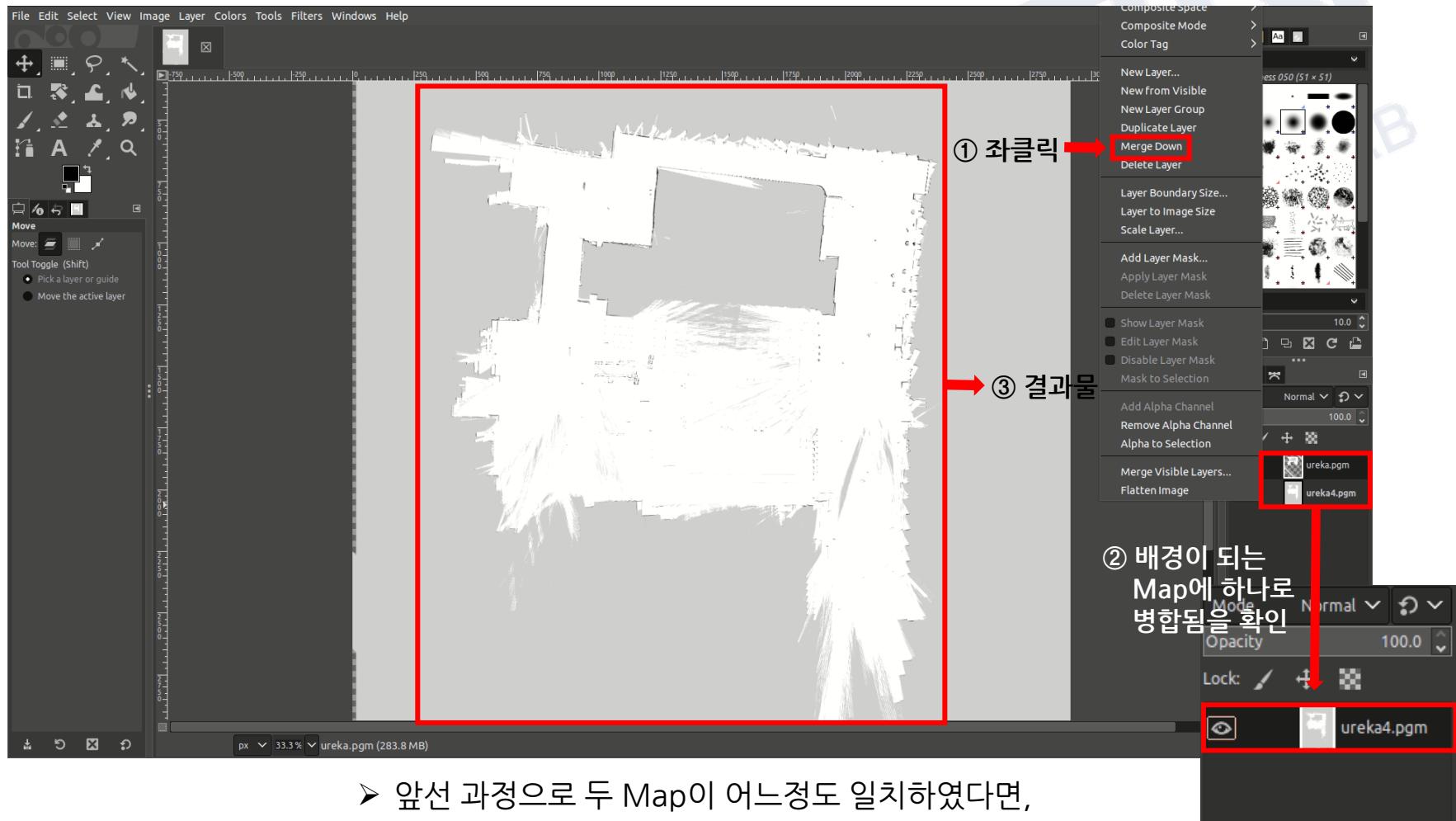


- Map을 결합하는 과정에서 보다 정확한 결합을 위해 덧붙이는 Map의 투명도를 조절함으로써 배경이 되는 Map과 일치하도록 보정 수행 가능.
- 두 Map을 일치시킨 후, 반드시 투명도(Opacity)를 다시 100으로 설정해야 함.

# 1. SLAM

## 5) GIMP 기반 Occupancy Grid Map 보정 (18/27)

### GIMP 편집기 사용 방법 - Map 병합 (2개 이상의 Map 병합)



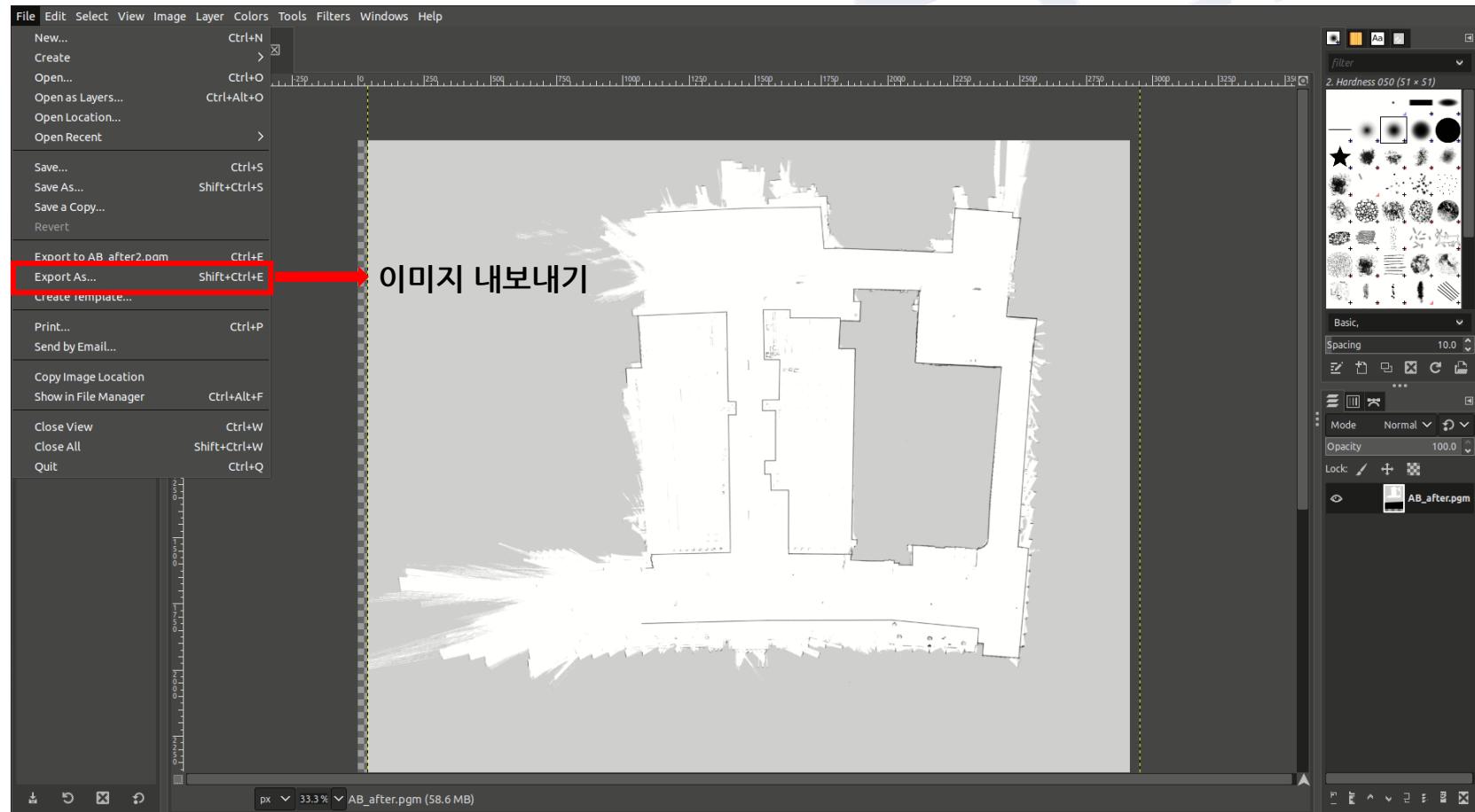
➤ 앞선 과정으로 두 Map이 어느정도 일치하였다면,  
 ①의 Merge Down을 통한 두 layer 병합 수행.

# 1. SLAM

## 5) GIMP 기반 Occupancy Grid Map 보정 (19/27)

### GIMP 편집기 사용 방법 - Map 저장

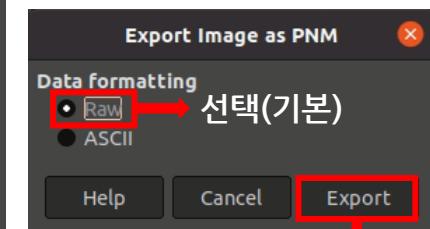
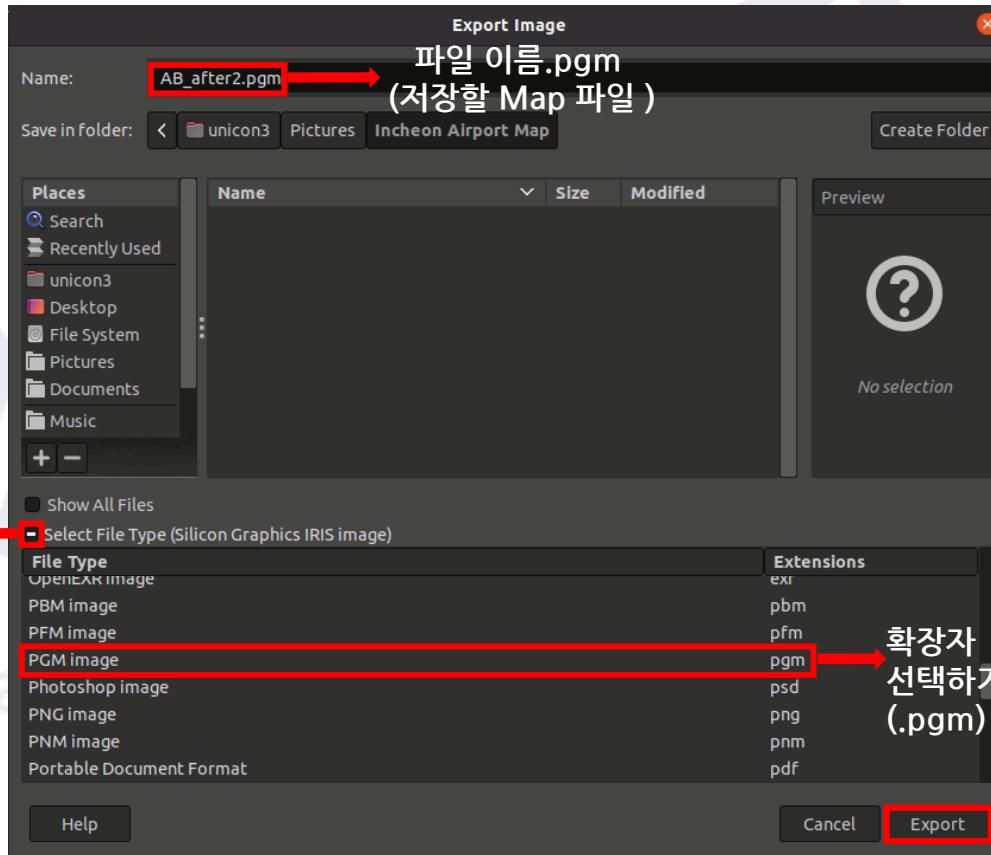
➤ Map에 대한 모든 보정 작업이 끝나고, 해당 Map 파일을 저장하기 위해서는,



# 1. SLAM

## 5) GIMP 기반 Occupancy Grid Map 보정 (20/27)

### GIMP 편집기 사용 방법 - Map 저장



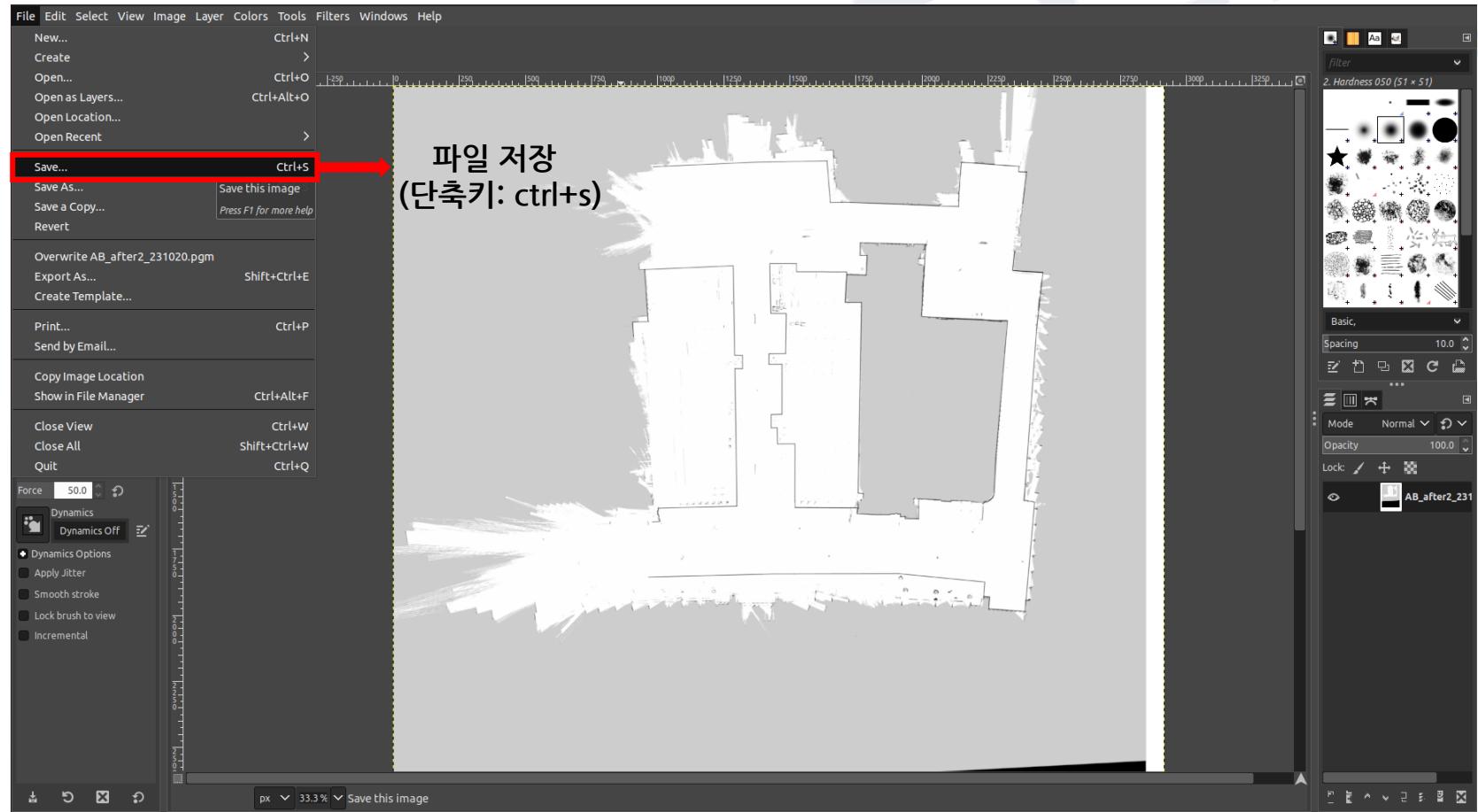
내보내기

# 1. SLAM

## 5) GIMP 기반 Occupancy Grid Map 보정 (21/27)

### GIMP 편집기 사용 방법 - Map 저장

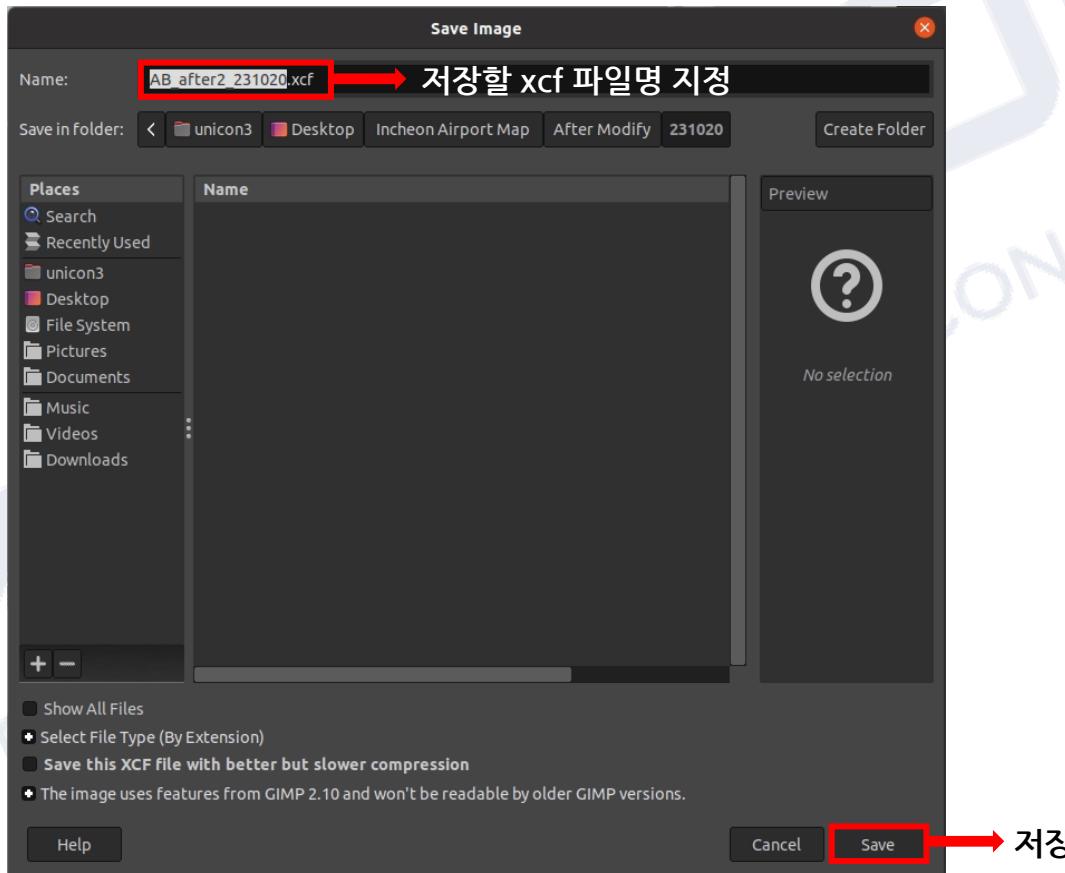
➤ 작업중인 Map 이미지 정보를 재 편집 시 사용하기 위한, xcf 파일로 저장하기 위해



# 1. SLAM

## 5) GIMP 기반 Occupancy Grid Map 보정 (22/27)

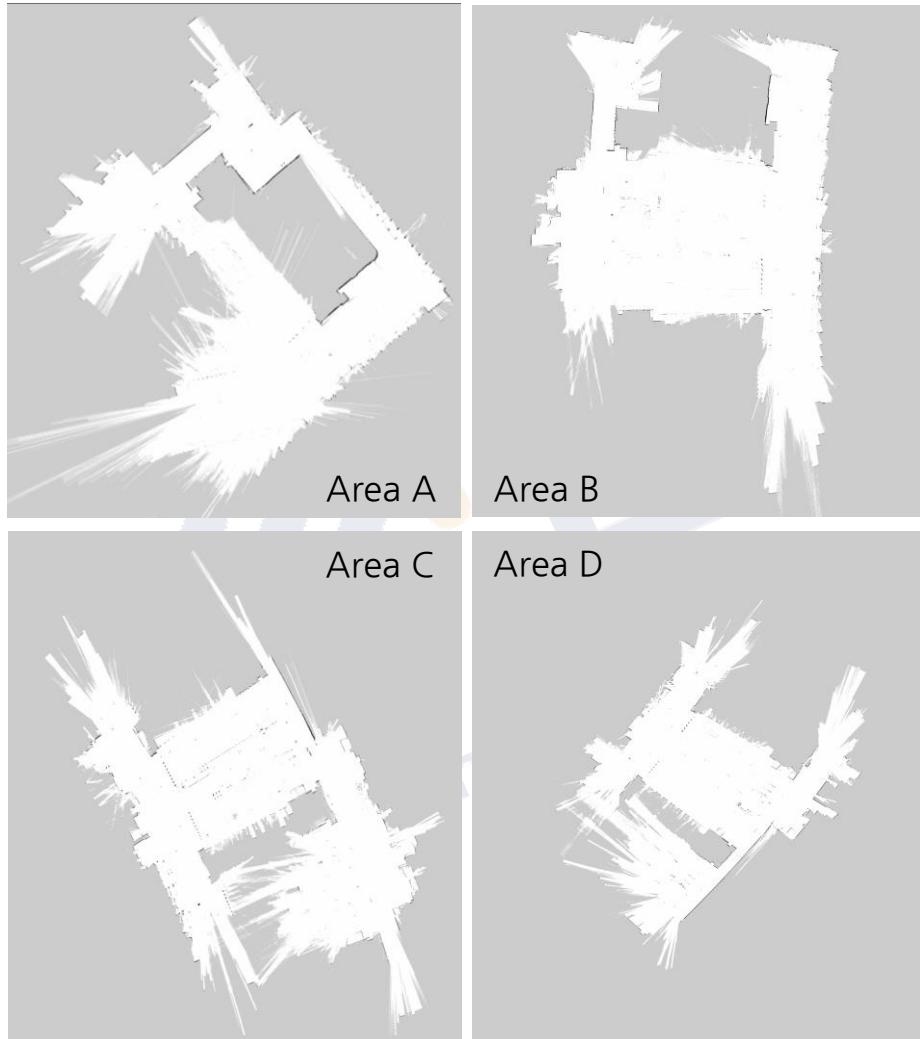
### GIMP 편집기 사용 방법 - Map 저장



➤ 이와 같이 xcf 파일을 저장하게 되면 이미지의 레이어, 텍스트, 효과 및 다른 추가적인 요소를 보존하게 되며 향후 이미지 재 편집 시 용이함.

# 1. SLAM

## 5) GIMP 기반 Occupancy Grid Map 보정 (23/27)



GIMP 편집기 사용 방법 - Map 보정 예시



Incheon Airport 3f A~D Area Map Calibration results

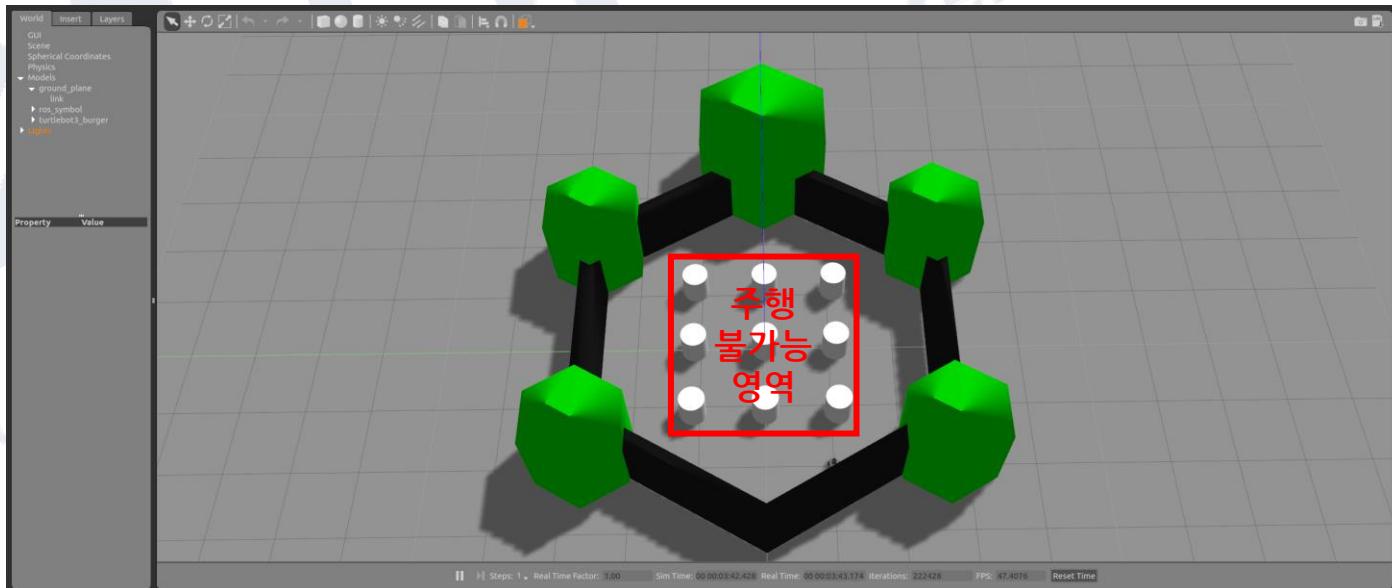
- ❖ 인천공항 3층 동편 A ~ D 구역 각각의 Mapping 결과와 A ~ D 구역의 Map을 결합 후 보정한 결과.

# 1. SLAM

## 5) GIMP 기반 Occupancy Grid Map 보정 (24/27)

### GIMP 편집기 사용 방법 - 실습

- 앞선 GIMP 편집기를 사용하여, 이전 Turtlebot3 시뮬레이션 환경에서 SLAM을 통해 얻었던 Occupancy Grid Map 보정 수행.
- 실습 1. 다음 그림과 같이, Turtlebot3 burger 모델이 주어진 환경의 내부로 주행하지 못하도록 GIMP 편집기를 통해 Occupancy Grid Map을 보정하시오.

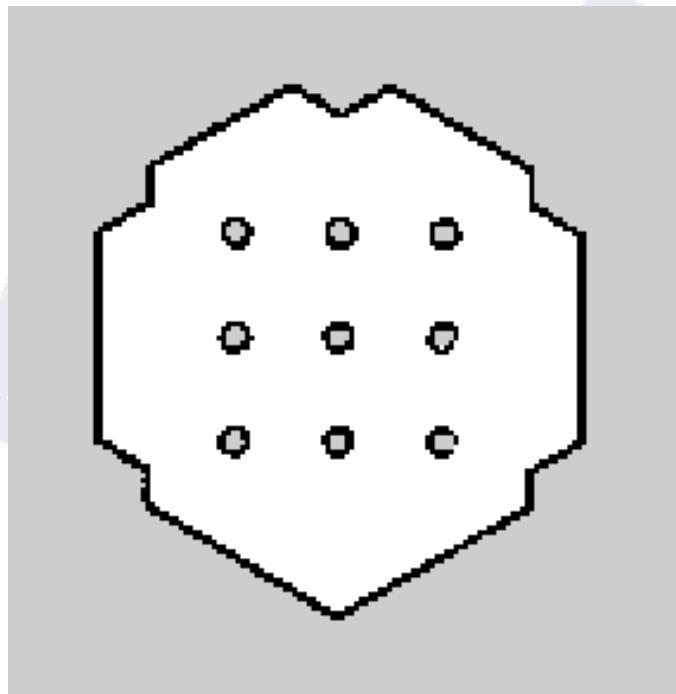


# 1. SLAM

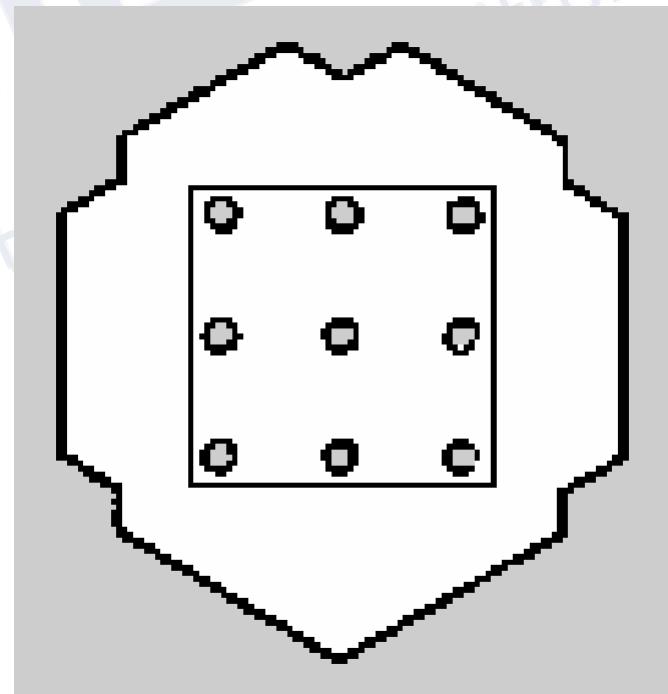
## 5) GIMP 기반 Occupancy Grid Map 보정 [25/27]

### GIMP 편집기 사용 방법 - 실습

- 실습 1. 다음 그림과 같이, Turtlebot3 burger 모델이 주어진 환경의 내부로 주행하지 못하도록 GIMP 편집기를 통해 Occupancy Grid Map을 보정하시오.



보정 전



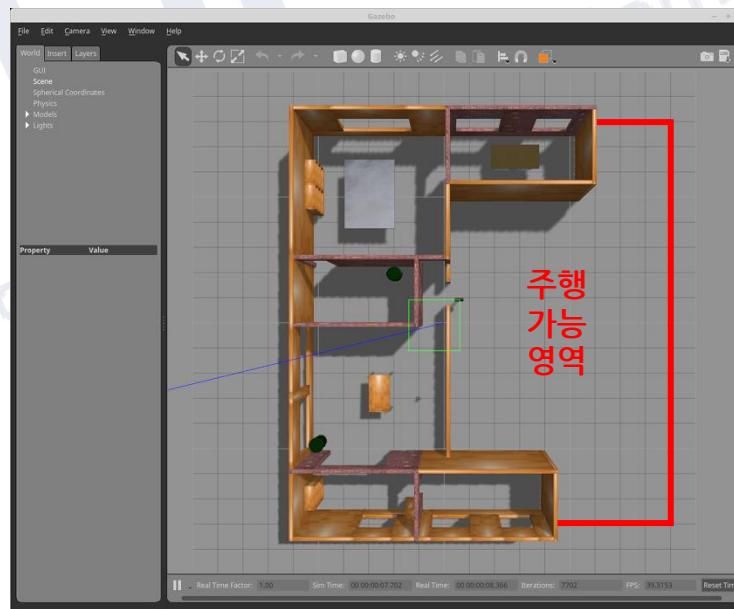
보정 후

# 1. SLAM

## 5) GIMP 기반 Occupancy Grid Map 보정 (26/27)

### GIMP 편집기 사용 방법 - 실습

- 앞선 GIMP 편집기를 사용하여, 이전 Turtlebot3 시뮬레이션 환경에서 SLAM을 통해 얻었던 Occupancy Grid Map 보정 수행.
- 실습 2. 다음 그림과 같이, Turtlebot3 burger 모델이 주어진 환경의 외부로 주행할 수 있도록 GIMP 편집기를 통해 Occupancy Grid Map을 보정하시오.

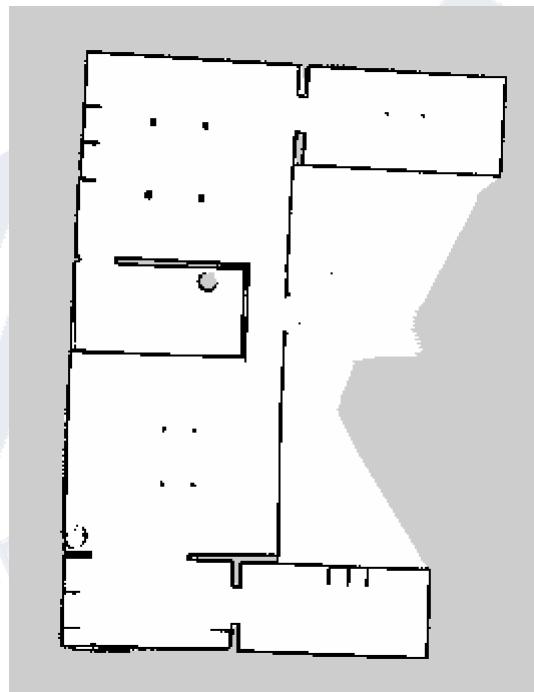


# 1. SLAM

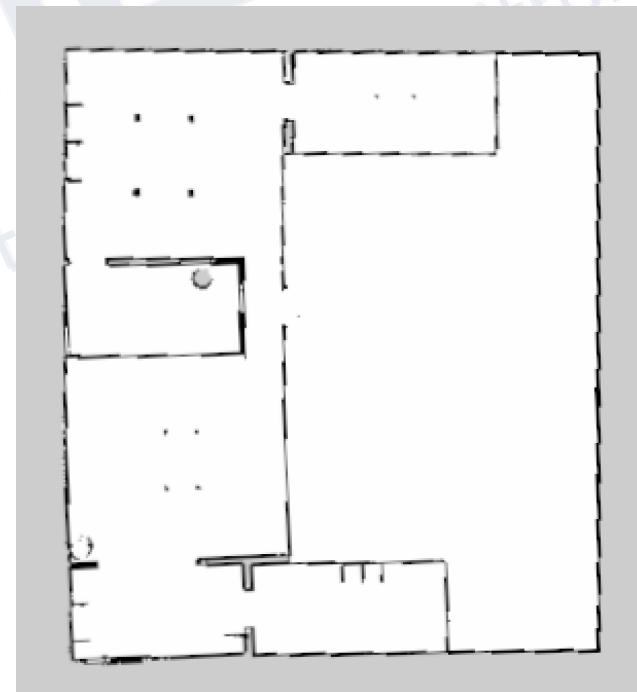
## 5) GIMP 기반 Occupancy Grid Map 보정 (27/27)

### GIMP 편집기 사용 방법 - 실습

- 실습 2. 다음 그림과 같이, Turtlebot3 burger 모델이 주어진 환경의 외부로 주행할 수 있도록 GIMP 편집기를 통해 Occupancy Grid Map을 보정하시오.



보정 전

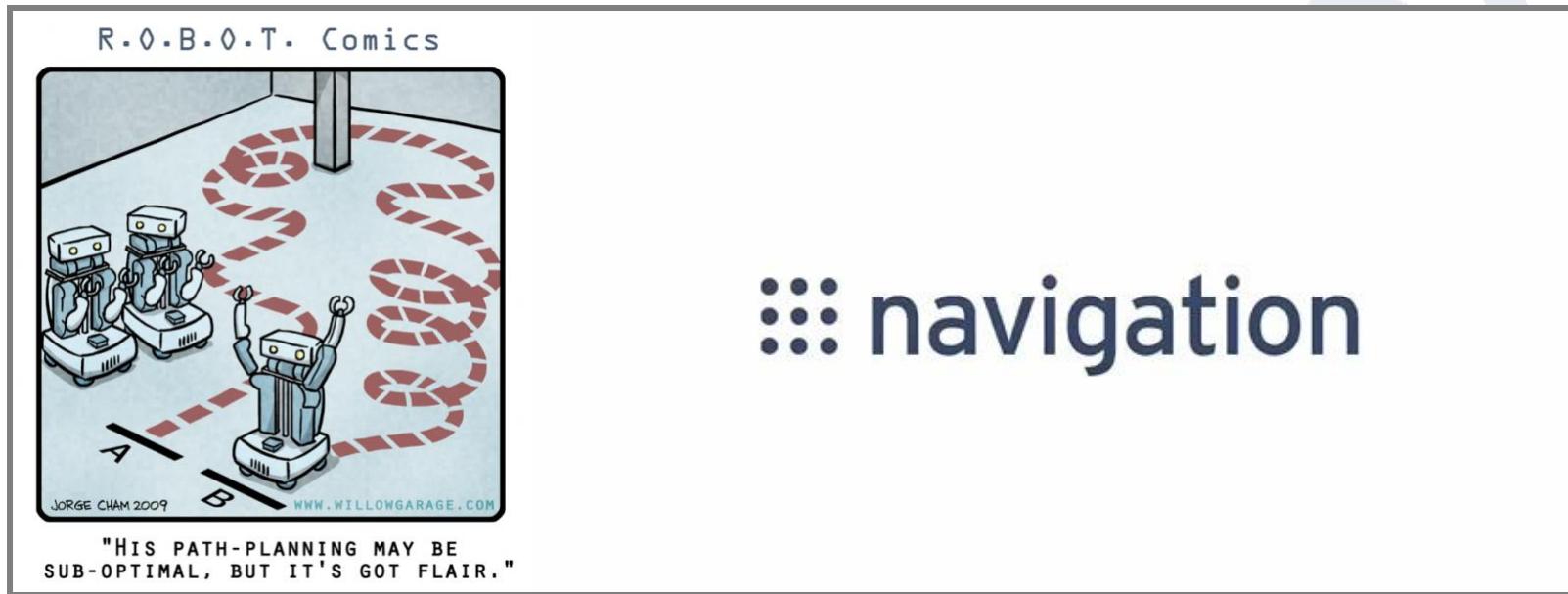


보정 후

## 2. Navigation

## 2. Navigation

### 1) ROS Navigation Stack이란?

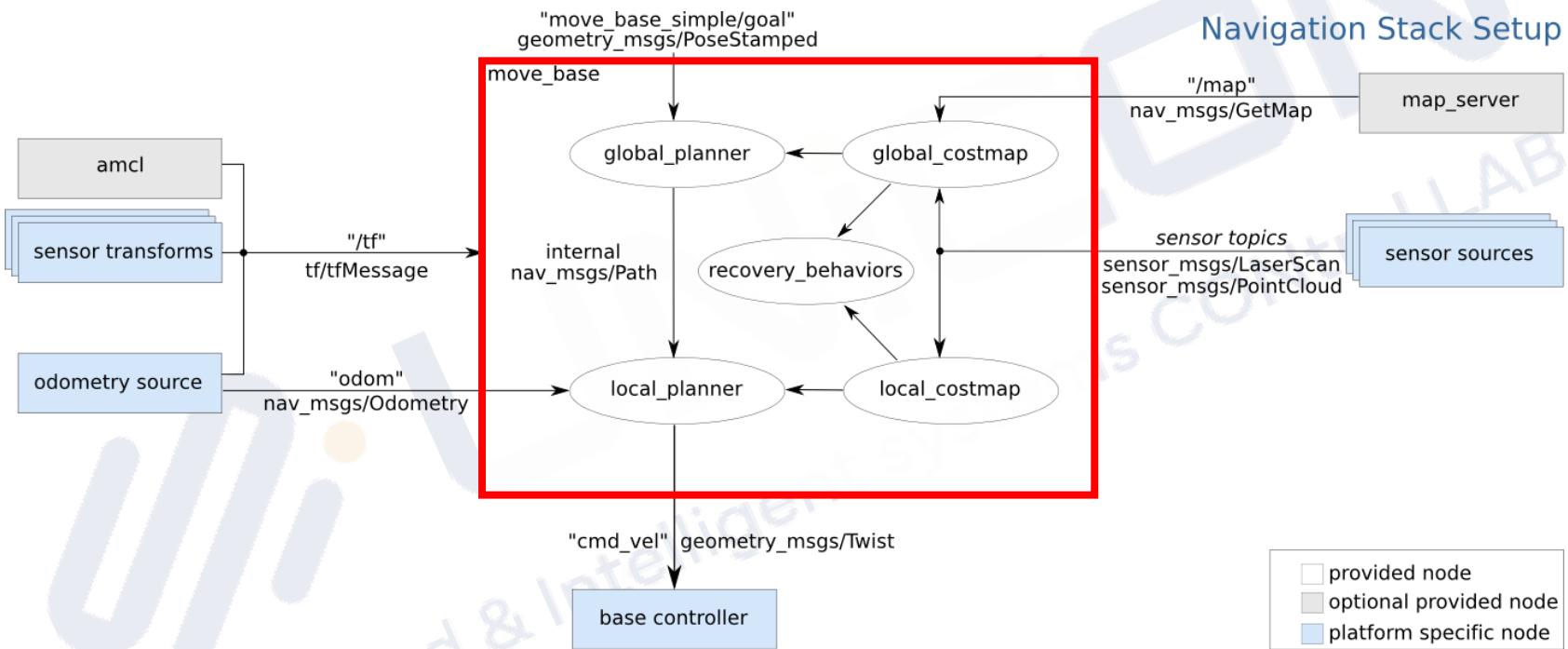


자료 출처. <https://wiki.ros.org/navigation>

- ROS Navigation Stack은 로봇이 주변 환경을 인식하고, 계획된 경로를 따라 자율적으로 이동할 수 있도록 하는 데 사용되는 통합 소프트웨어 프레임워크.
- ROS Navigation Stack은 센서 데이터의 통합, 위치 추정, 경로 계획, 장애물 회피, 그리고 실행까지의 전 과정을 관리하며, 특히 이동 로봇의 복잡한 탐색 문제를 해결하는 데 적합.

## 2. Navigation

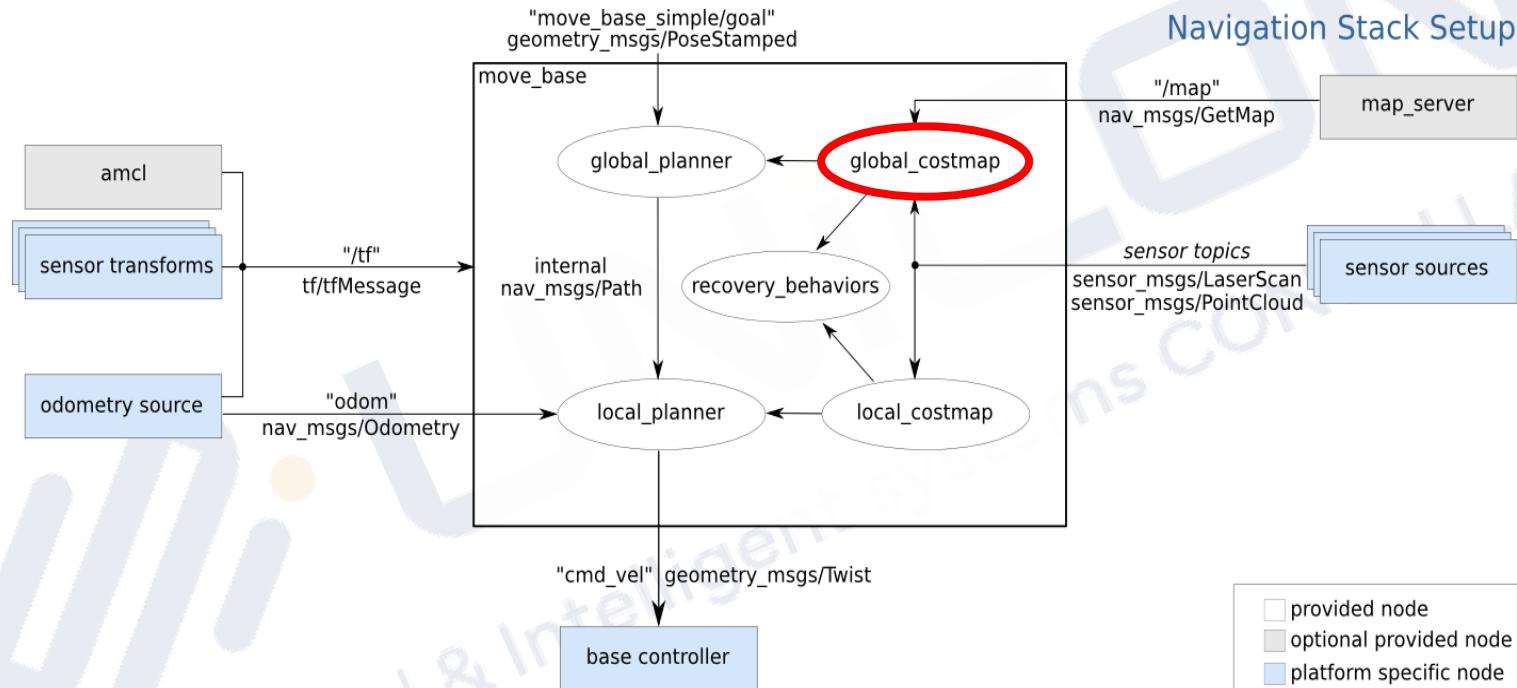
### 2) ROS Navigation Stack의 구조 및 원리



- ROS Navigation Stack은 **move\_base** 노드를 중심으로 구성되어 있으며, **move\_base** 노드를 통해 로봇이 이동할 경로를 계획하거나 정적·동적 장애물 회피를 위한 제어 입력을 생성.
- move\_base** 노드는 **global\_costmap**, **local\_costmap**, **global\_planner**, **local\_planner**로 구성.

## 2. Navigation

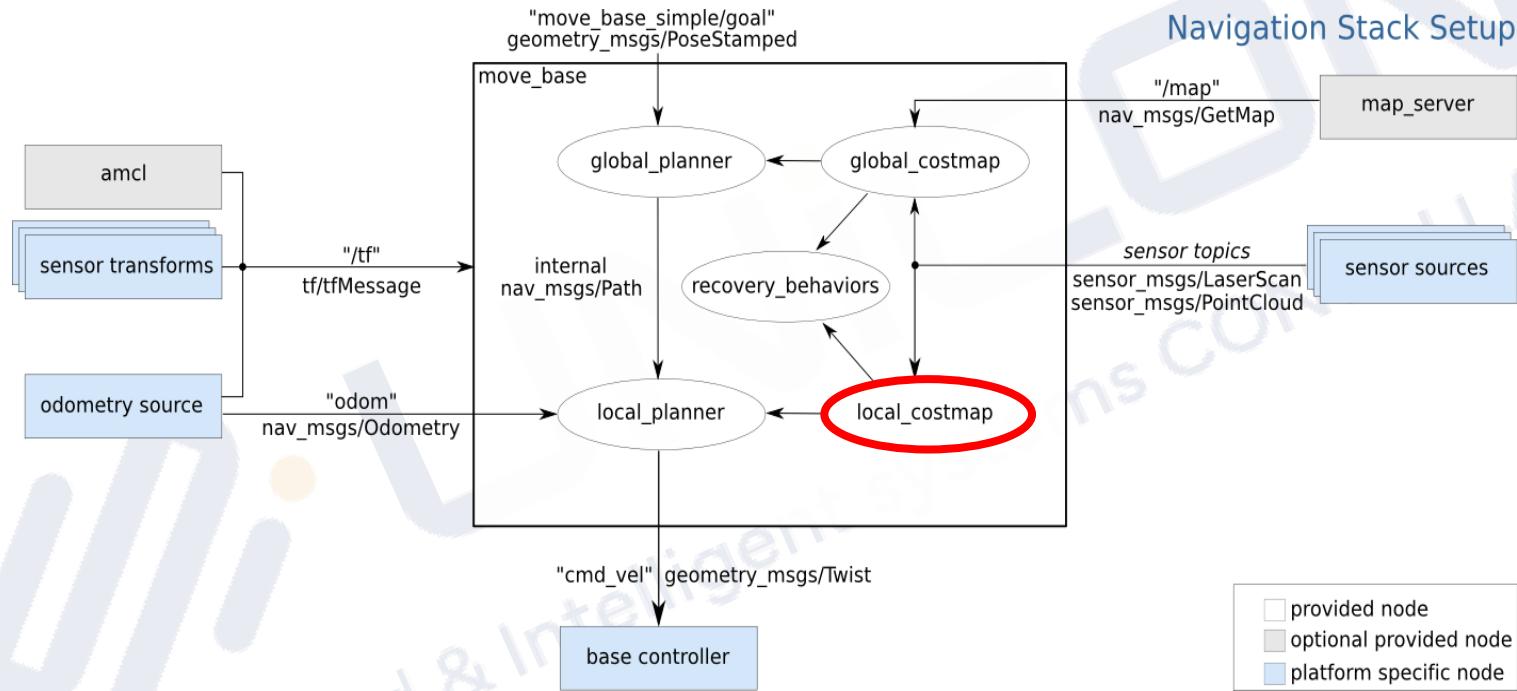
### 2) ROS Navigation Stack의 구조 및 원리 – global\_costmap



- Global cost map은 앞서 SLAM(Gmapping)을 통해 생성한 Occupancy Grid Map을 기반으로, 각 셀이 가지는 비용 값을 재 구성하여 Global planner에게 제공.

## 2. Navigation

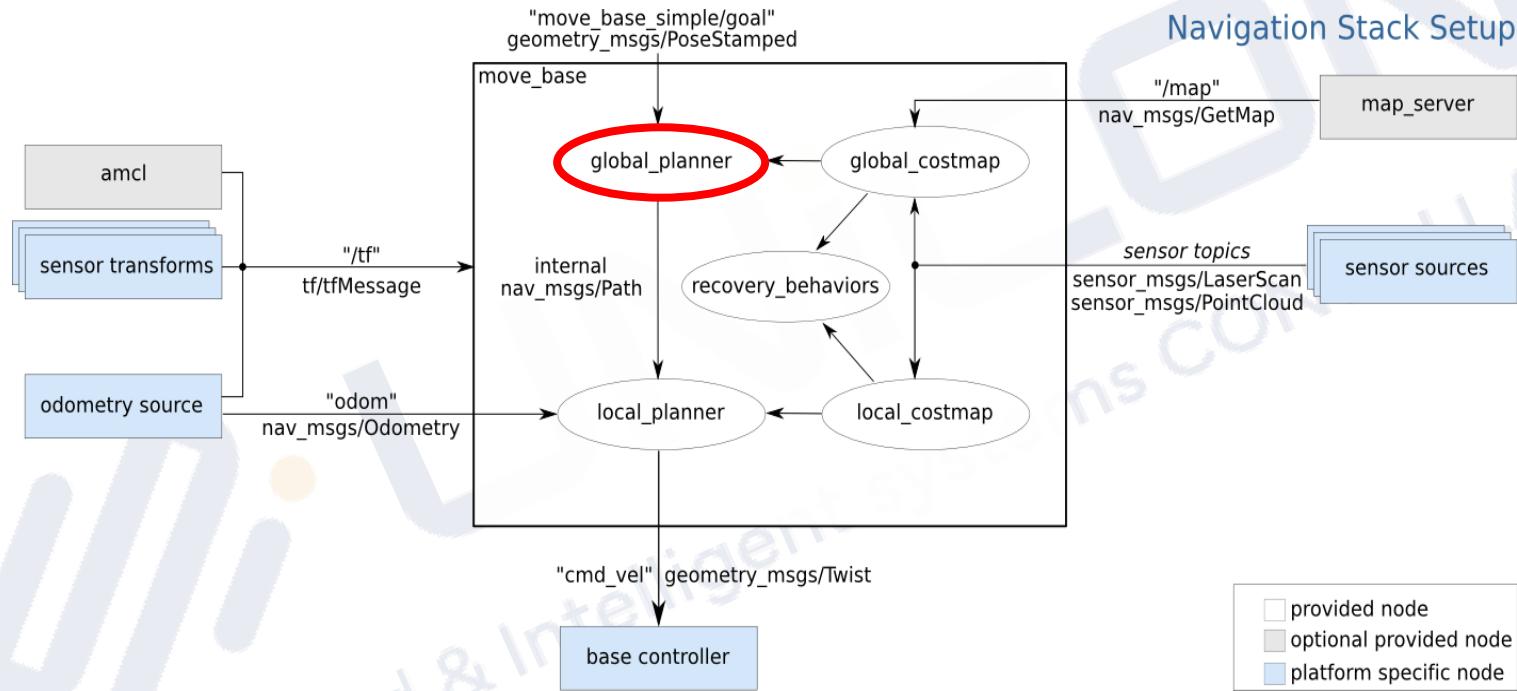
### 2) ROS Navigation Stack의 구조 및 원리 – local\_costmap



- Local cost map은 LiDAR 센서 등을 바탕으로 로봇 주변의 국소적인 비용 지도를 생성하는 기법.
- 생성된 Local cost map은 Local planner에 제공되며, 이를 통해 사람이나 다른 로봇과 같은 동적 장애물을 회피하기 위한 제어 입력 생성 가능.

## 2. Navigation

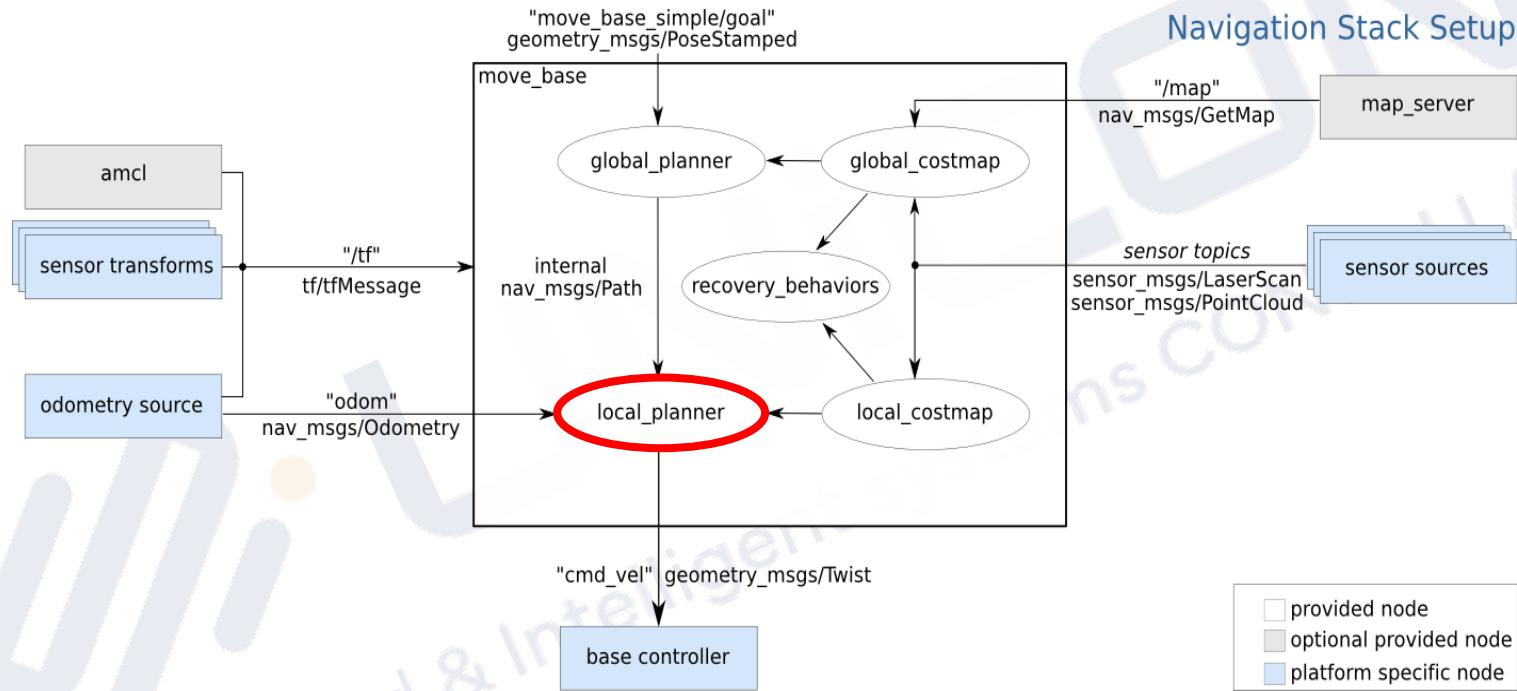
### 2) ROS Navigation Stack의 구조 및 원리 – global\_planner



- Global planner(Global Path Planning)는 앞서 생성된 Global cost map을 바탕으로 로봇이 목표 지점까지 이동하기 위한 Global path를 계산하여 도출.
- 생성된 Global path는 목표 위치가 변경되는 경우에만 다시 계산됨.

## 2. Navigation

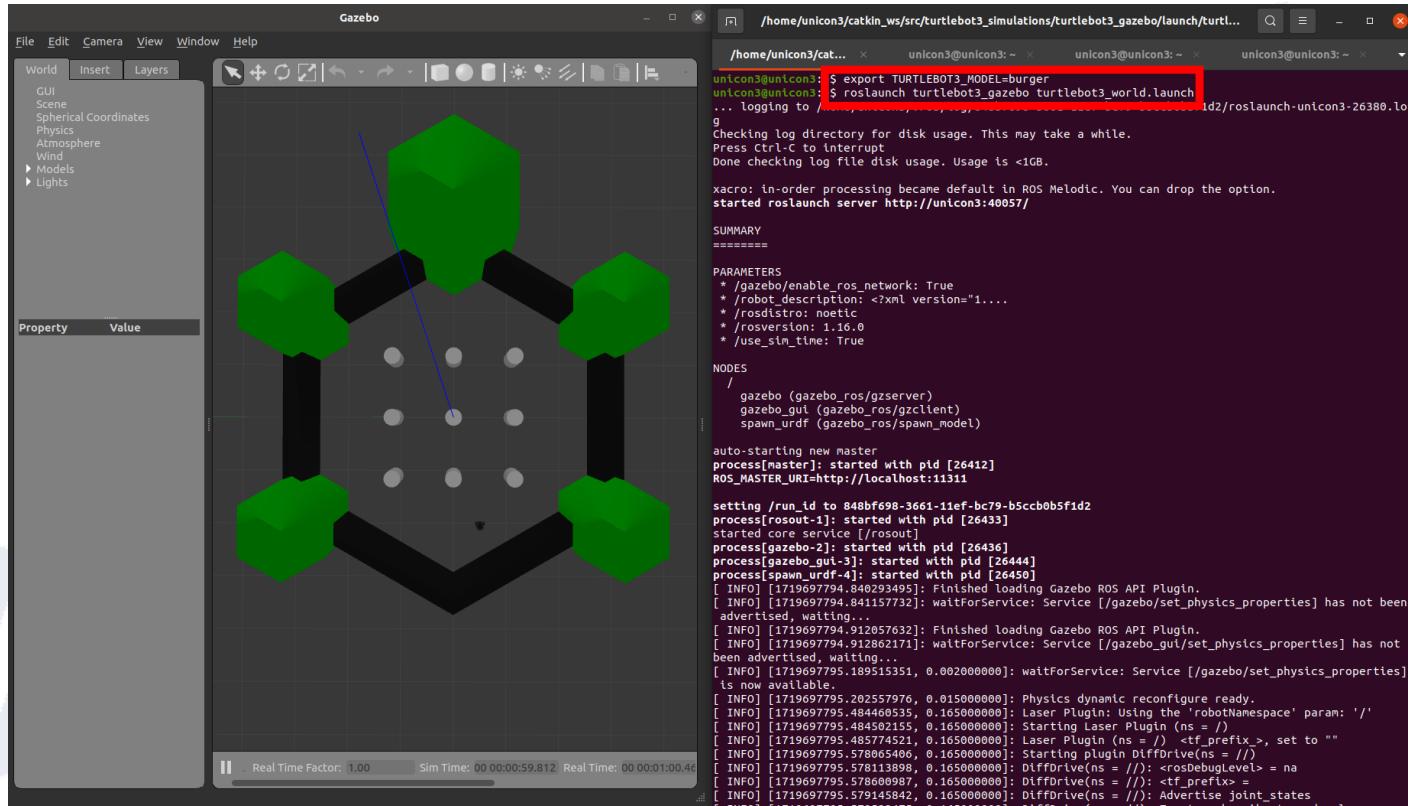
### 2) ROS Navigation Stack의 구조 및 원리 – local\_planner



- Local planner(Local Path Planner)는 Global planner를 통해 도출된 Global path를 추종함과 동시에, 센서 데이터를 통해 생성된 Local cost map을 바탕으로 정·동적 장애물을 회피하기 위한 제어 입력을 생성.

## 2. Navigation

### 3) ROS Navigation Stack 기반 자율주행 실습 (1/26)



**\$ export TURTLEBOT3\_MODEL=burger (또는 export TURTLEBOT3\_MODEL=waffle)**

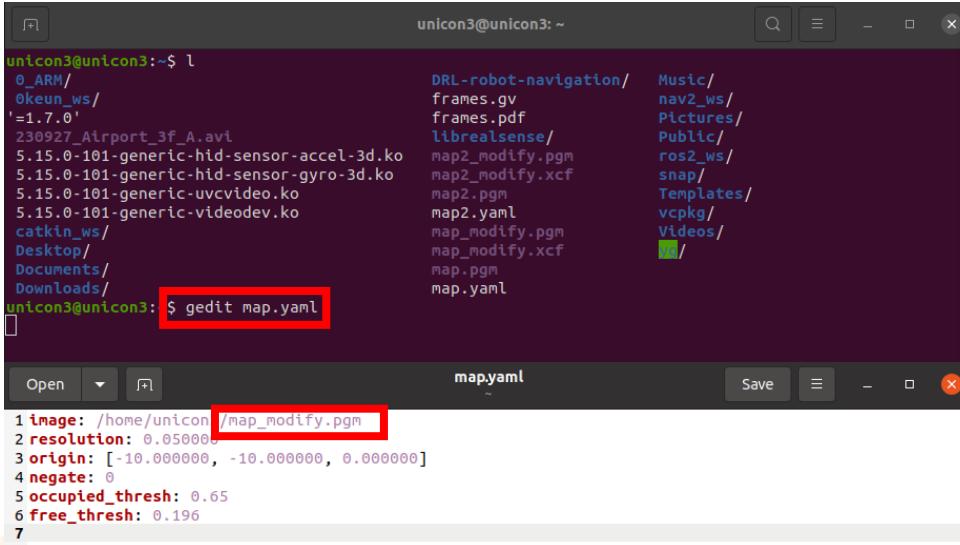
(→ 빌드 후, 환경 실행 전에 사용할 Turtlebot3 모델 정의)

**\$ roslaunch turtlebot3\_gazebo turtlebot3\_world.launch**

(→ Turtlebot3 시뮬레이션 환경 실행)

## 2. Navigation

### 3) ROS Navigation Stack 기반 자율주행 실습 [2/26]



The terminal window shows the command `ls` being run, listing various directories and files including `DRL-robot-navigation/`, `frames.gv`, `frames.pdf`, `librealsense/`, and `map2_modify.pgm`. The `map2_modify.pgm` file is highlighted with a red box.

The gedit editor window displays the `map.yaml` file content:

```

1 image: /home/unicon/map_modify.pgm
2 resolution: 0.050000
3 origin: [-10.000000, -10.000000, 0.000000]
4 negate: 0
5 occupied_thresh: 0.65
6 free_thresh: 0.196
7
  
```

`$ ls`

(→ ls(=list)를 통해 현재 경로에 위치한 폴더 및 파일 확인)

`$ gedit map.yaml`

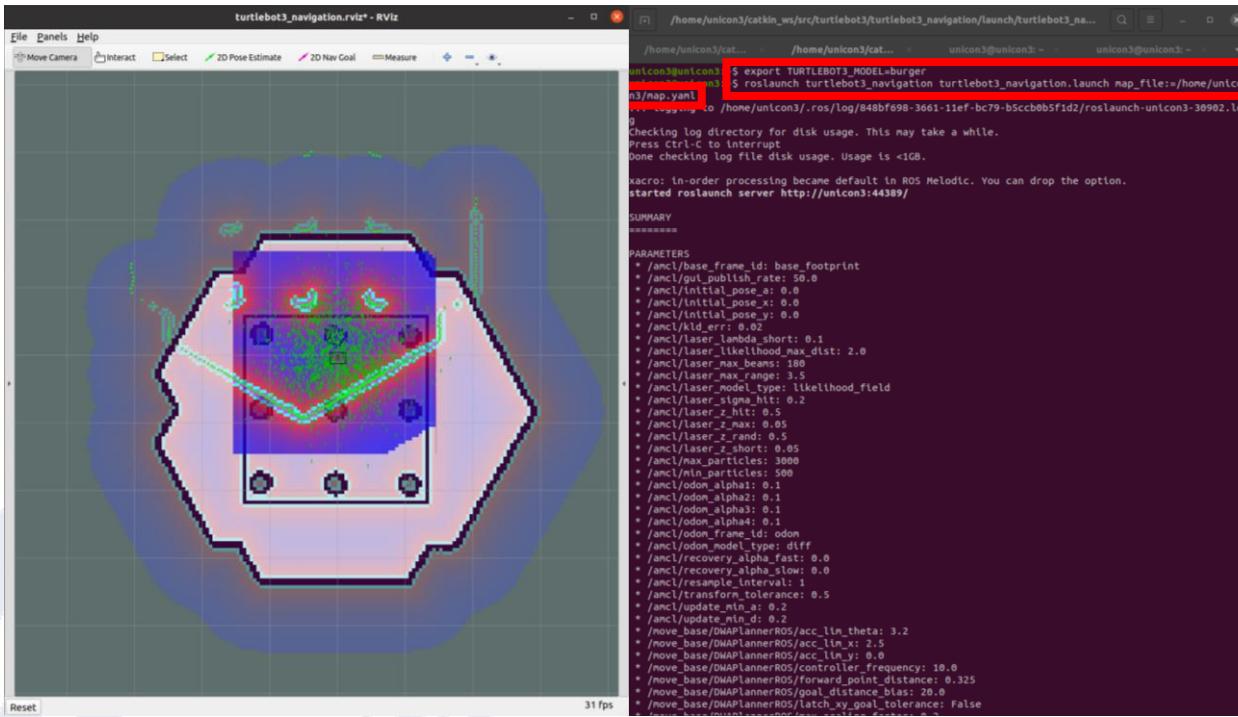
(→ gedit(=GNOME editor)을 통해 “map.yaml” 파일 수정)

(→ image란의 “map.pgm”을 “map\_modify.pgm”으로 수정 후, 단축어 “ctrl+s”를 통해 저장)

`$ 위 과정을 통해 보정된 Occupancy Grid Map을 불러올 수 있음.`

## 2. Navigation

### 3) ROS Navigation Stack 기반 자율주행 실습 [3/26]



\$ export TURTLEBOT3\_MODEL=burger (또는 export TURTLEBOT3\_MODEL=waffle)

(→ 빌드 후, 환경 실행 전에 사용할 Turtlebot3 모델 정의)

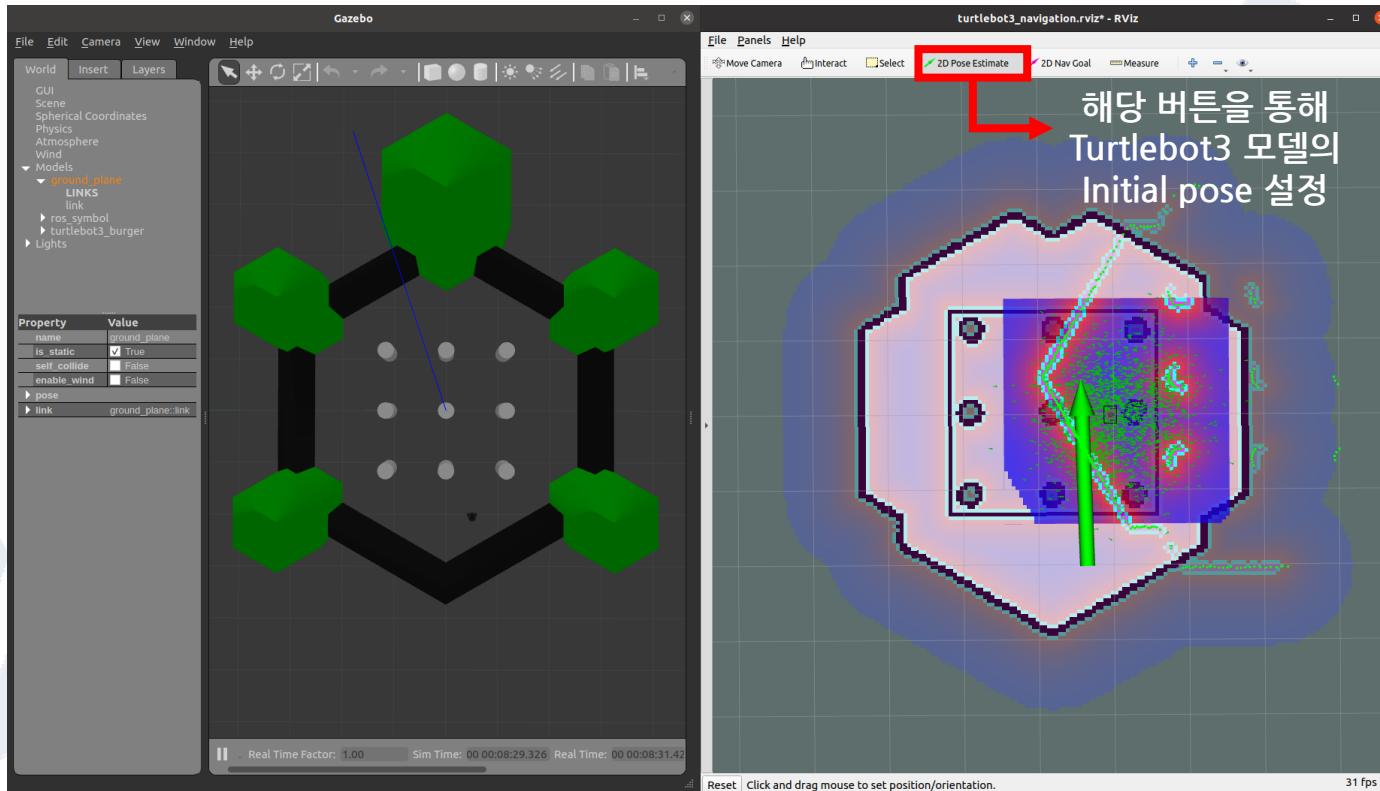
\$ rosrun turtlebot3\_navigation turtlebot3\_navigation.launch map\_file:=/home/unicon3/map.yaml

(→ 보정된 Occupancy Grid Map을 기반으로 Turtlebot3 모델의 자율주행을 위한 Navigation 노드 실행)

(→ 위 빨간색의 경로는 로컬 PC마다 모두 다르므로, Map이 저장된 경로에서 명령어 “pwd”를 통해 경로 확인)

## 2. Navigation

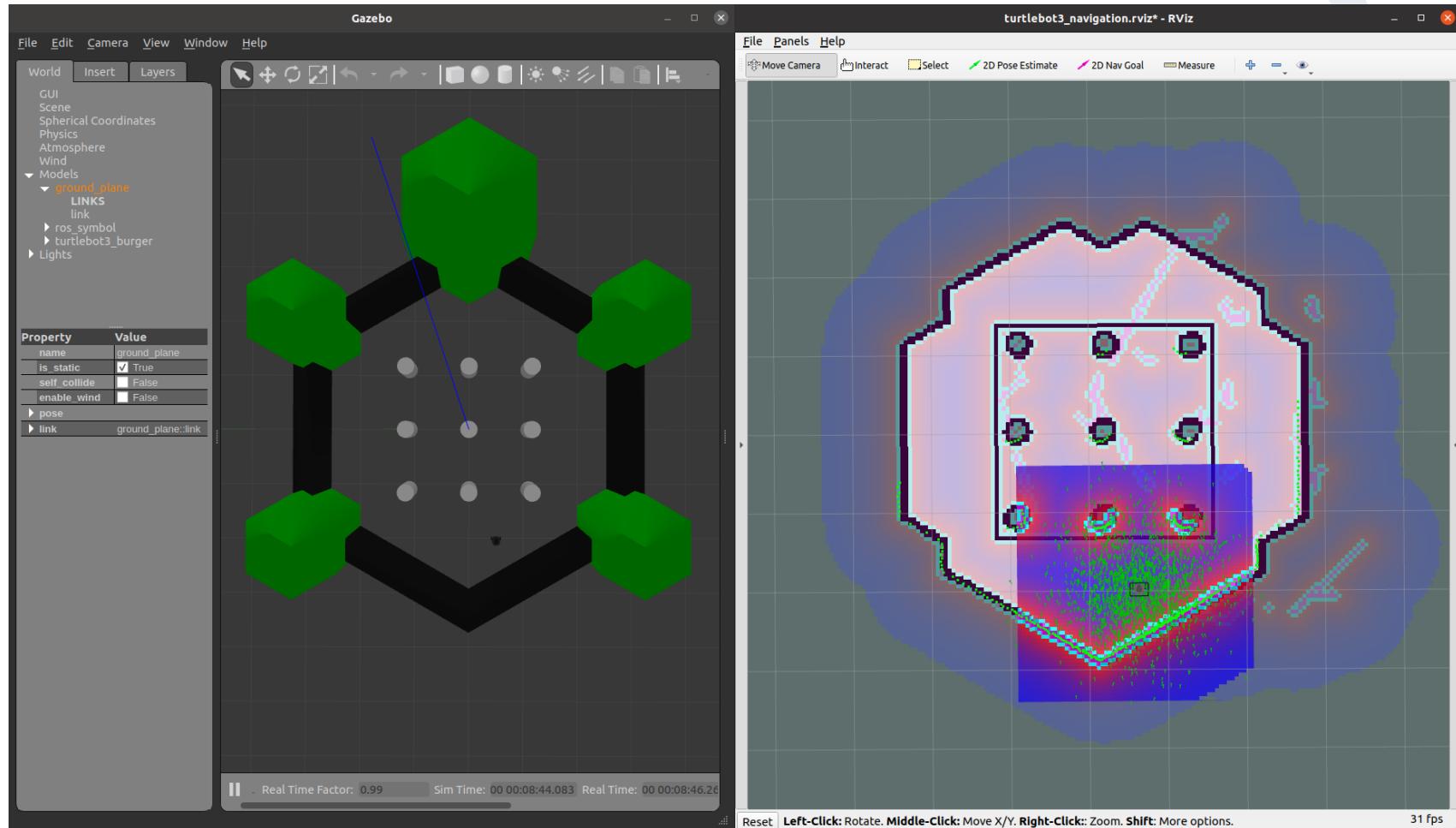
### 3) ROS Navigation Stack 기반 자율주행 실습 (4/26)



- 오른쪽 Rviz(Ros visualization)의 상단부 중앙에 위치한 “2D Pose Estimate”를 클릭한 후, 왼쪽 Gazebo 시뮬레이션 환경에 위치한 Turtlebot3 burger 모델이 위치한 곳과 동일한 Rviz 상의 위치에 마우스를 드래그하여 로봇의 초기 위치 설정

## 2. Navigation

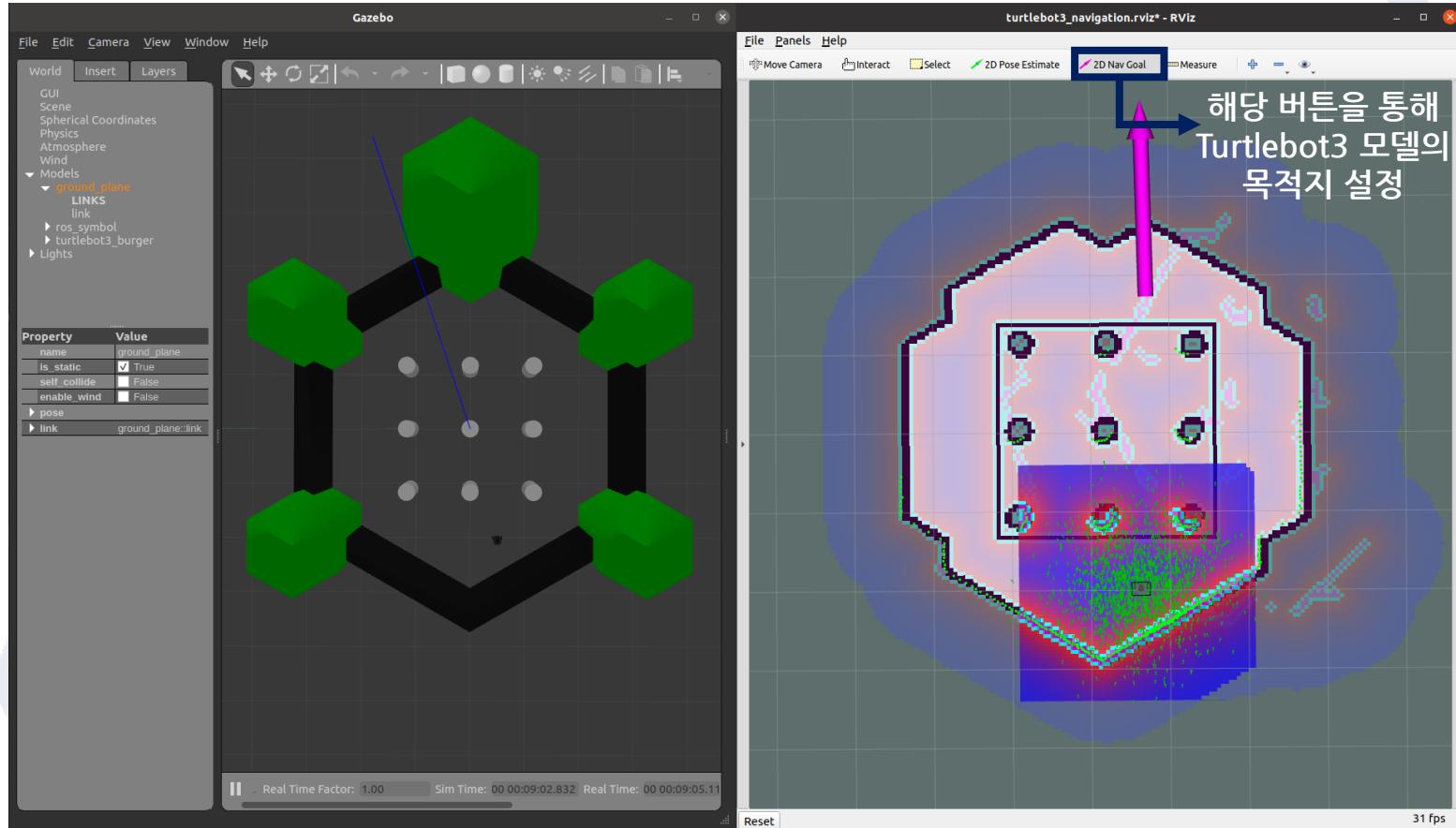
### 3) ROS Navigation Stack 기반 자율주행 실습 (5/26)



- 오른쪽 Rviz(Ros visualization)상의 Map과 같이, Turtlebot3 burger 모델의 초기 위치 설정

## 2. Navigation

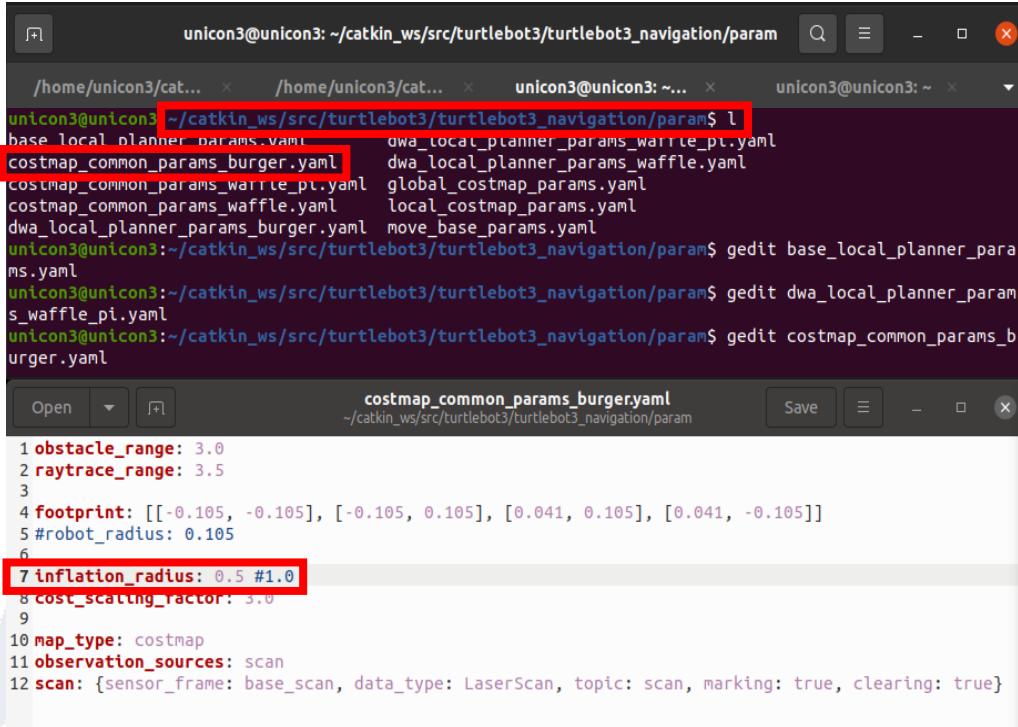
### 3) ROS Navigation Stack 기반 자율주행 실습 [6/26]



- 오른쪽 Rviz(Ros visualization)의 상단부에 위치한 “2D Nav Goal”를 클릭한 후, 앞서 설정한 “2D Pose Estimate” 과정과 동일하게 Rviz 상의 원하는 위치에 마우스를 드래그하여 로봇이 도달해야 하는 목표 위치와 설정

## 2. Navigation

### 3) ROS Navigation Stack 기반 자율주행 실습 (7/26)



```
unicon3@unicon3: ~/catkin_ws/src/turtlebot3/turtlebot3_navigation/param$ ls
base_local_planner_params.yaml          dwa_local_planner_params_waffle_pi.yaml
costmap_common_params_burger.yaml        dwa_local_planner_params_waffle.yaml
costmap_common_params_waffle_pi.yaml     global_costmap_params.yaml
costmap_common_params_waffle.yaml        local_costmap_params.yaml
dwa_local_planner_params_burger.yaml    move_base_params.yaml
unicon3@unicon3:~/catkin_ws/src/turtlebot3/turtlebot3_navigation/param$ gedit base_local_planner_params.yaml
unicon3@unicon3:~/catkin_ws/src/turtlebot3/turtlebot3_navigation/param$ gedit dwa_local_planner_params_waffle_pi.yaml
unicon3@unicon3:~/catkin_ws/src/turtlebot3/turtlebot3_navigation/param$ gedit costmap_common_params_burger.yaml
```

costmap\_common\_params\_burger.yaml

```
1 obstacle_range: 3.0
2 raytrace_range: 3.5
3
4 footprint: [[-0.105, -0.105], [-0.105, 0.105], [0.041, 0.105], [0.041, -0.105]]
5 #robot_radius: 0.105
6
7 inflation_radius: 0.5 #1.0
8 cost_scaling_factor: 3.0
9
10 map_type: costmap
11 observation_sources: scan
12 scan: {sensor_frame: base_scan, data_type: LaserScan, topic: scan, marking: true, clearing: true}
```

- 만약 GIMP 편집기를 통해 Occupancy Grid Map상에 설정한 장애물(벽)으로 인해 Turtlebot3 burger 모델이 설정한 목표 위치까지 도달하지 못한다면, 다음과 같이 파라미터를 수정.

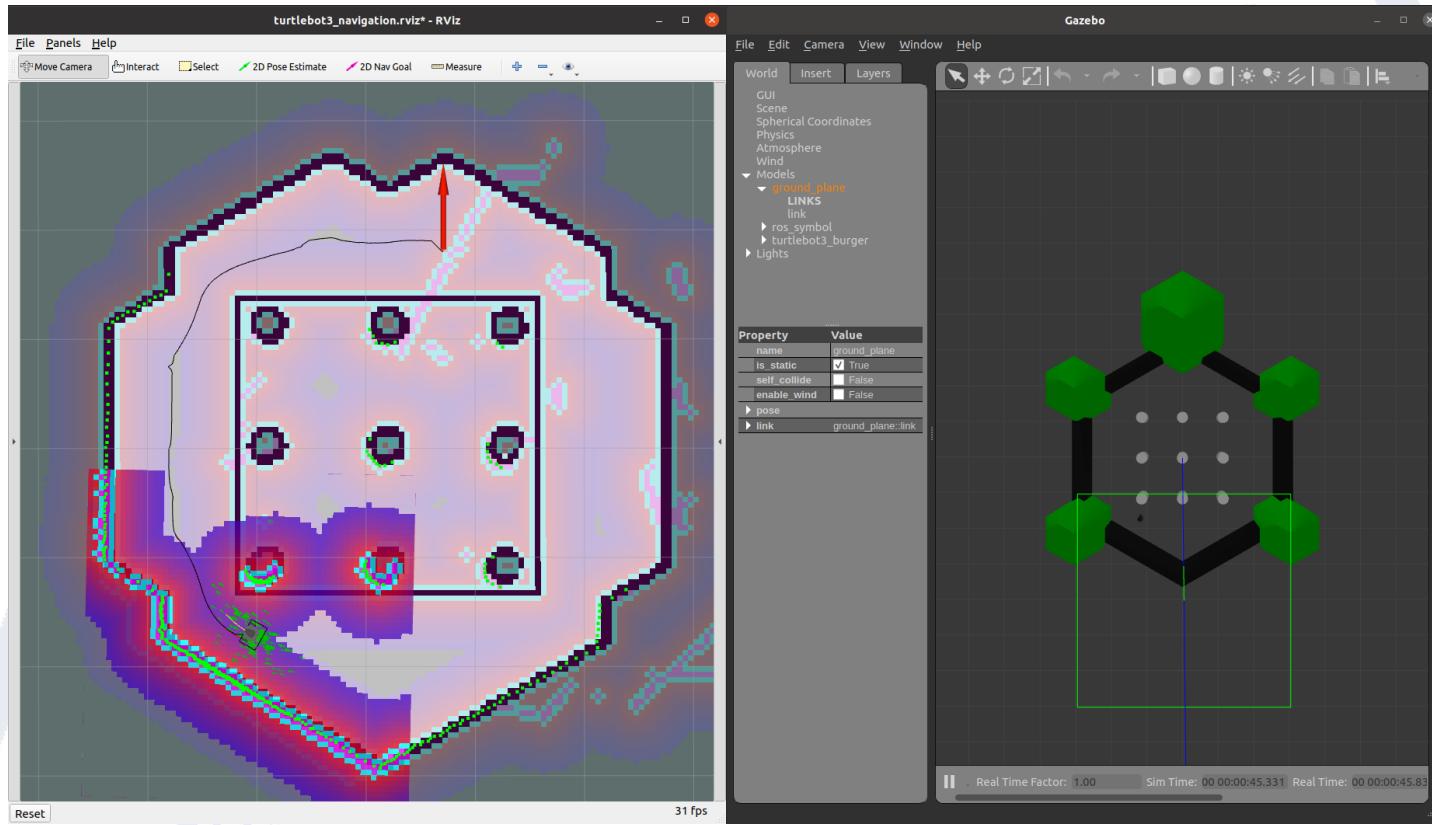
\$ cd ~/catkin\_ws/src/turtlebot3/turtlebot3\_navigation/param

(→ Turtlebot3 Navigation을 구성하는 parameter 폴더로 이동하여 위와 같이 수정 후 저장)

\$ gedit costmap\_common\_params\_burger.yaml

## 2. Navigation

### 3) ROS Navigation Stack 기반 자율주행 실습 [8/26]

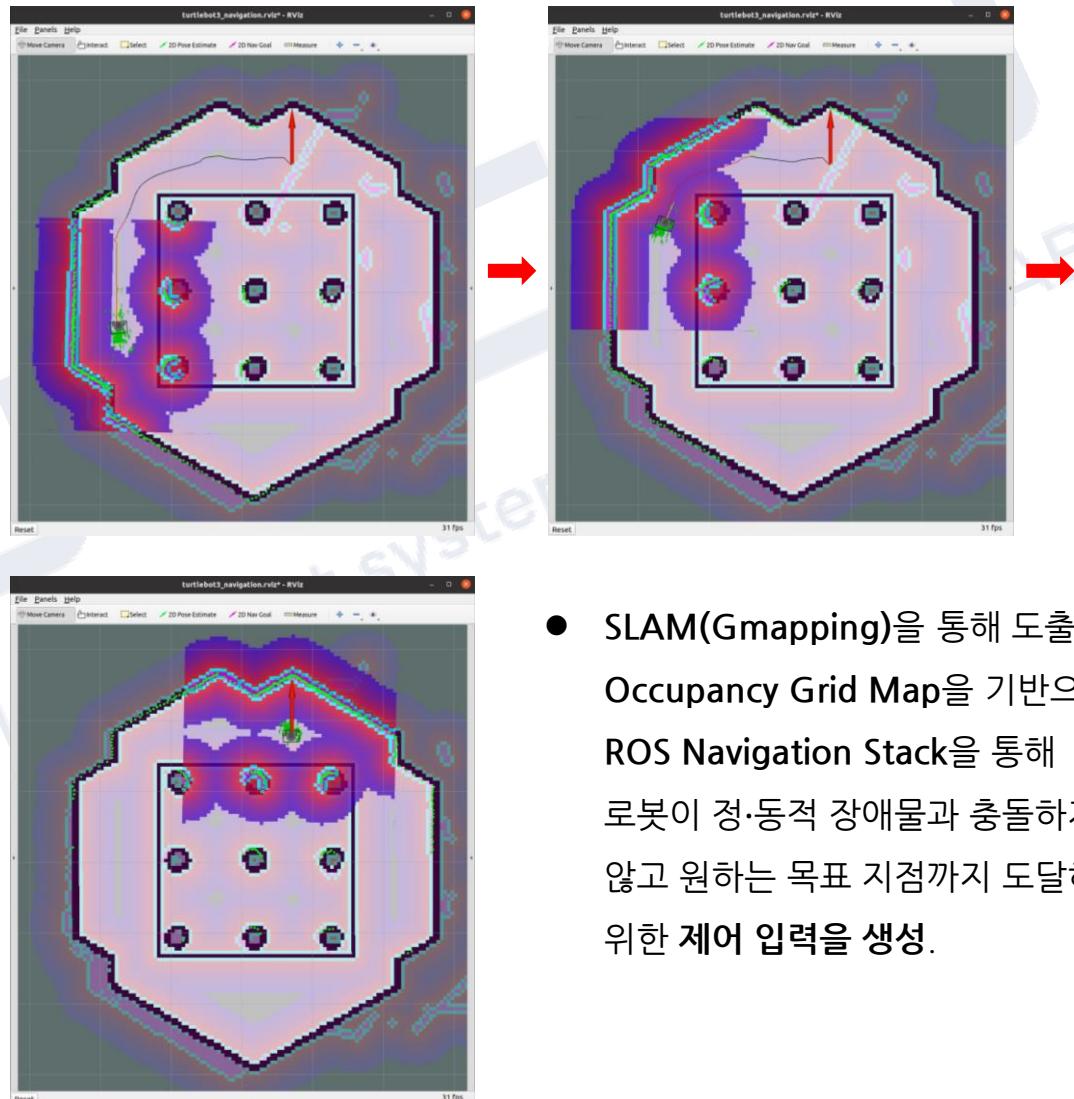


- 앞선 과정을 통해 장애물의 부피를 증가시키는 파라미터인 ‘inflation\_radius’ 값을 0.5로 낮추면, ‘global\_planner’의 경로 탐색을 통해 원하는 목적지까지 로봇이 주행해야 할 경로를 생성.
- 또한, 생성된 경로를 추종하기 위해 ‘local\_planner’를 통한 제어 입력을 생성함으로써 자율주행 수행.

## 2. Navigation

### 3) ROS Navigation Stack 기반 자율주행 실습 [9/26]

- SLAM(Gmapping)  
Occupancy Grid  
ROS Navigation  
로봇이 정·동적 장애물과 충돌하지 않고 원하는 목표 위치에 도달하기 위한 제어 입력을



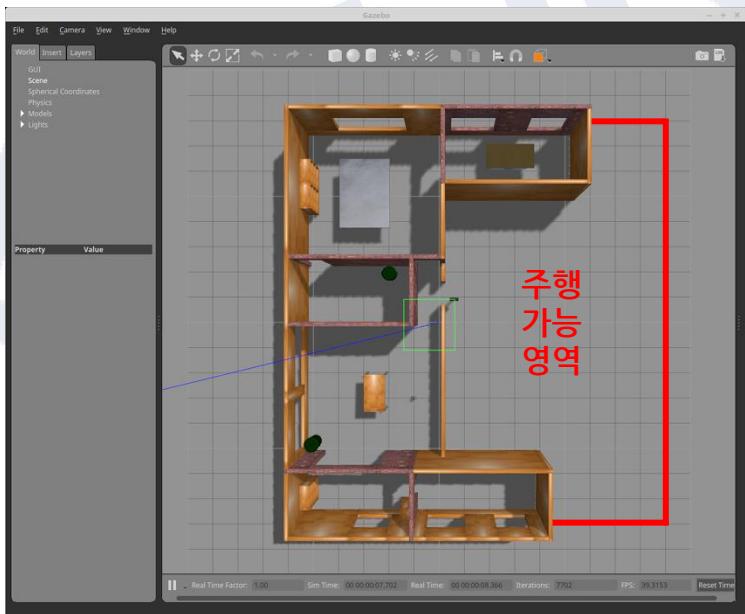
- SLAM(Gmapping)을 통해 도출한 Occupancy Grid Map을 기반으로, ROS Navigation Stack을 통해 로봇이 정·동적 장애물과 충돌하지 않고 원하는 목표 지점까지 도달하기 위한 제어 입력을 생성.

## 2. Navigation

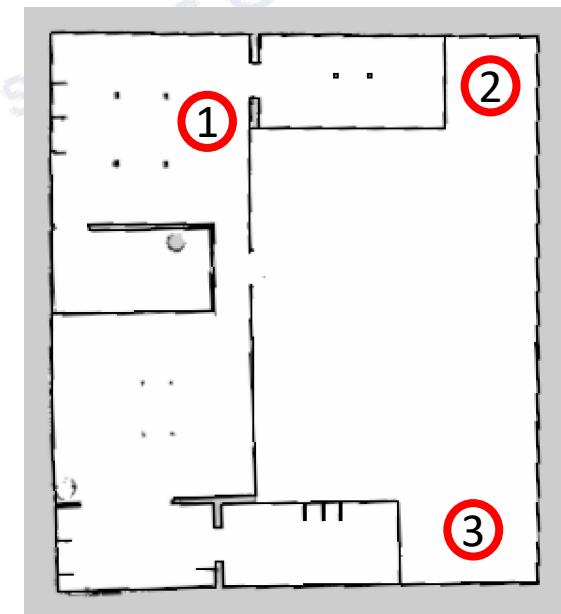
### 3) ROS Navigation Stack 기반 자율주행 실습 (10/26)

#### ROS 기반 Navigation Stack 실습

- 아래의 Turtlebot3 시뮬레이션 환경에서 SLAM(Gmapping)을 통해 도출한 Occupancy Grid Map을 기반으로 ROS Navigation Stack 수행하기.
- 실습 1. 아래의 그림과 같이, Turtlebot3 burger 모델이 주어진 Occupancy Grid Map에서 ROS Navigation Stack을 통해 **빨간색 원(1→2→3)**으로 이동 가능한지 검증하시오.



ROS Gazebo 시뮬레이션 환경

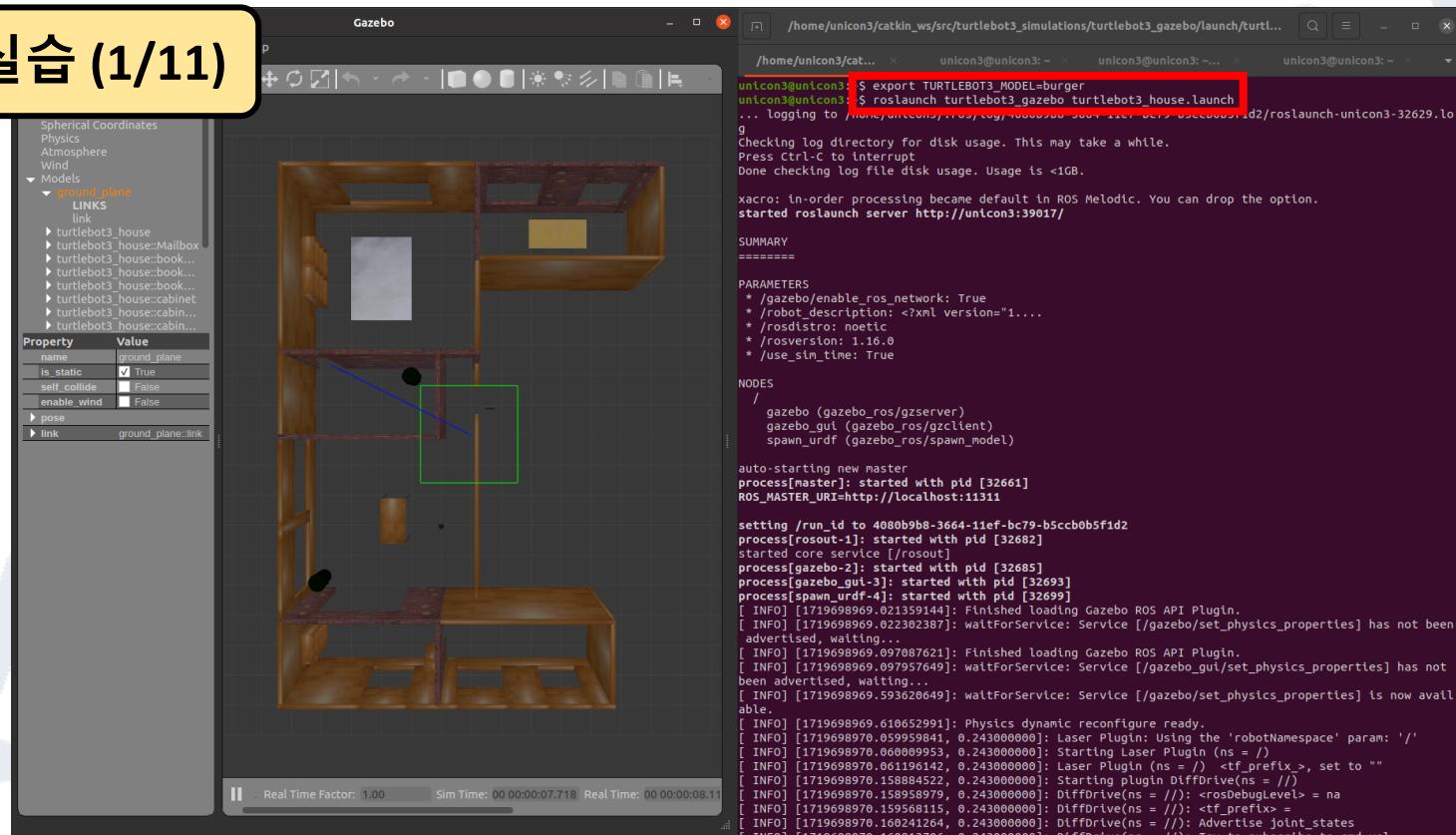


Occupancy Grid Map

## 2. Navigation

### 3) ROS Navigation Stack 기반 자율주행 실습 (11/26)

실습 (1/11)



**\$ export TURTLEBOT3\_MODEL=burger (또는 export TURTLEBOT3\_MODEL=waffle)**

(→ 빌드 후, 환경 실행 전에 사용할 Turtlebot3 모델 정의)

**\$ roslaunch turtlebot3\_gazebo turtlebot3\_world.launch**

(→ Turtlebot3 시뮬레이션 환경 실행)

## 2. Navigation

### 3) ROS Navigation Stack 기반 자율주행 실습 (12/26)

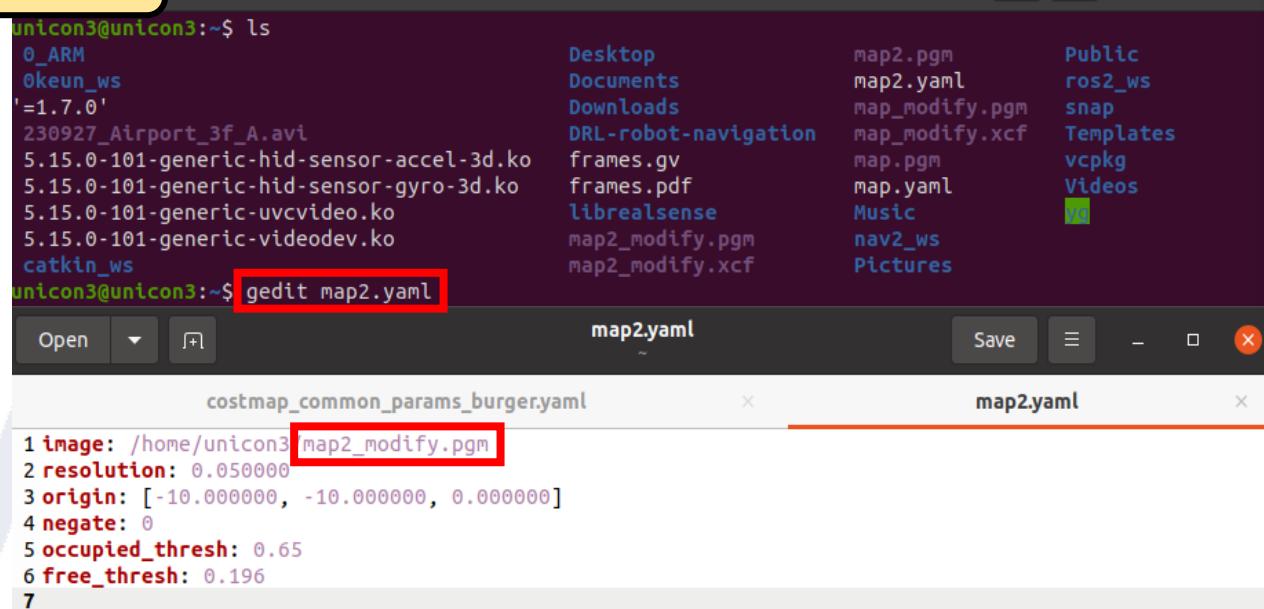
#### 실습 (2/11)

unicon3@unicon3:~\$ ls

```

 0_ARM          Desktop          map2.pgm        Public
 0keun_ws       Documents        map2.yaml      ros2_ws
 '=1.7.0'       Downloads        map_modify.pgm snap
 230927_Airport_3f_A.avi   DRL-robot-navigation frames.gv    Templates
 5.15.0-101-generic-hid-sensor-accel-3d.ko frames.pdf   map_modify.xcf vcpkg
 5.15.0-101-generic-hid-sensor-gyro-3d.ko librealsense map.pgm    Videos
 5.15.0-101-generic-uvccvideo.ko   map2_modify.pgm map.yaml  yg
 5.15.0-101-generic-videodev.ko   map2_modify.xcf Music     Pictures
 catkin_ws
unicon3@unicon3:~$ gedit map2.yaml
  
```

unicon3@unicon3:~\$ gedit map2.yaml



The terminal shows the command `ls` outputting a list of files and folders. It then shows the command `gedit map2.yaml`. Below the terminal is a screenshot of the GIMP editor window titled "map2.yaml" containing the YAML configuration for a costmap. A red box highlights the line `image: /home/unicon3/map2_modify.pgm`.

\$ ls

(→ ls(=list)를 통해 현재 경로에 위치한 폴더 및 파일 확인)

\$ gedit map2.yaml

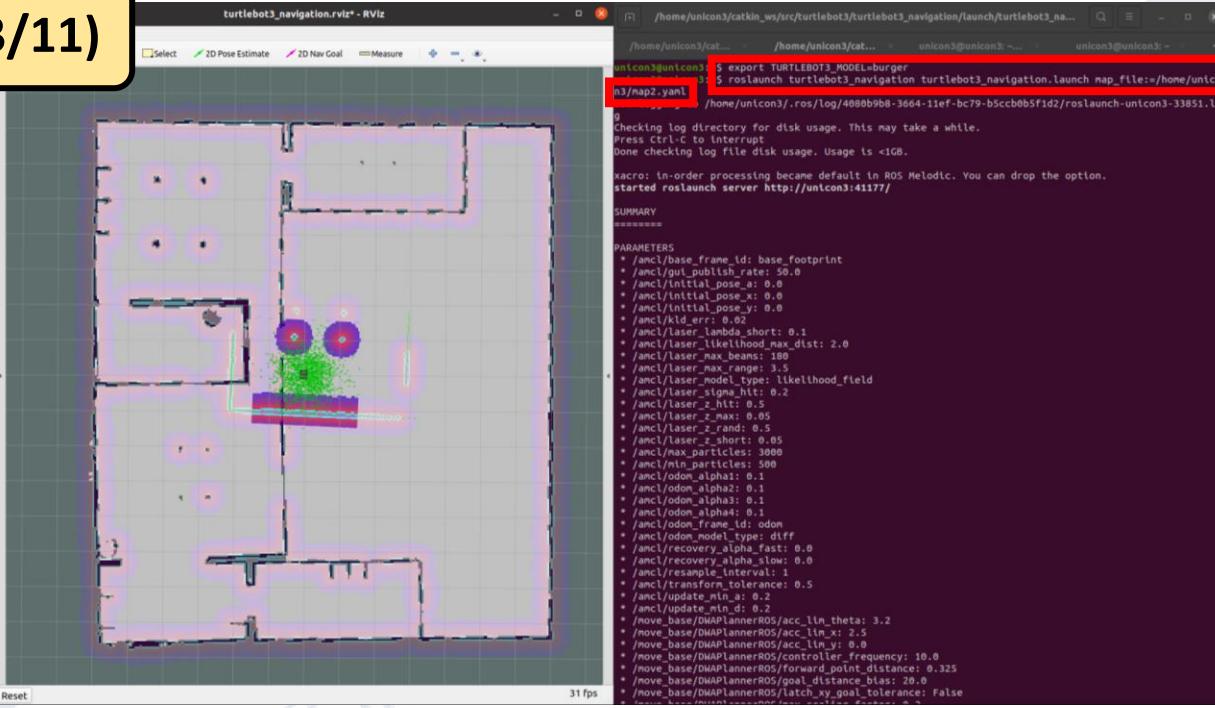
(→ gedit(=GNOME editor)을 통해 “map2.yaml” 파일 수정)

(→ image란의 “map2.pgm”을 “map2\_modify.pgm”으로 수정 후, 단축어 “ctrl+s”를 통해 저장)

## 2. Navigation

### 3) ROS Navigation Stack 기반 자율주행 실습 (13/26)

실습 (3/11)



**\$ export TURTLEBOT3\_MODEL=burger (또는 export TURTLEBOT3\_MODEL=waffle)**

(→ 빌드 후, 환경 실행 전에 사용할 Turtlebot3 모델 정의)

**\$ roslaunch turtlebot3\_navigation turtlebot3\_navigation.launch map\_file:=/home/unicon3/map2.yaml**

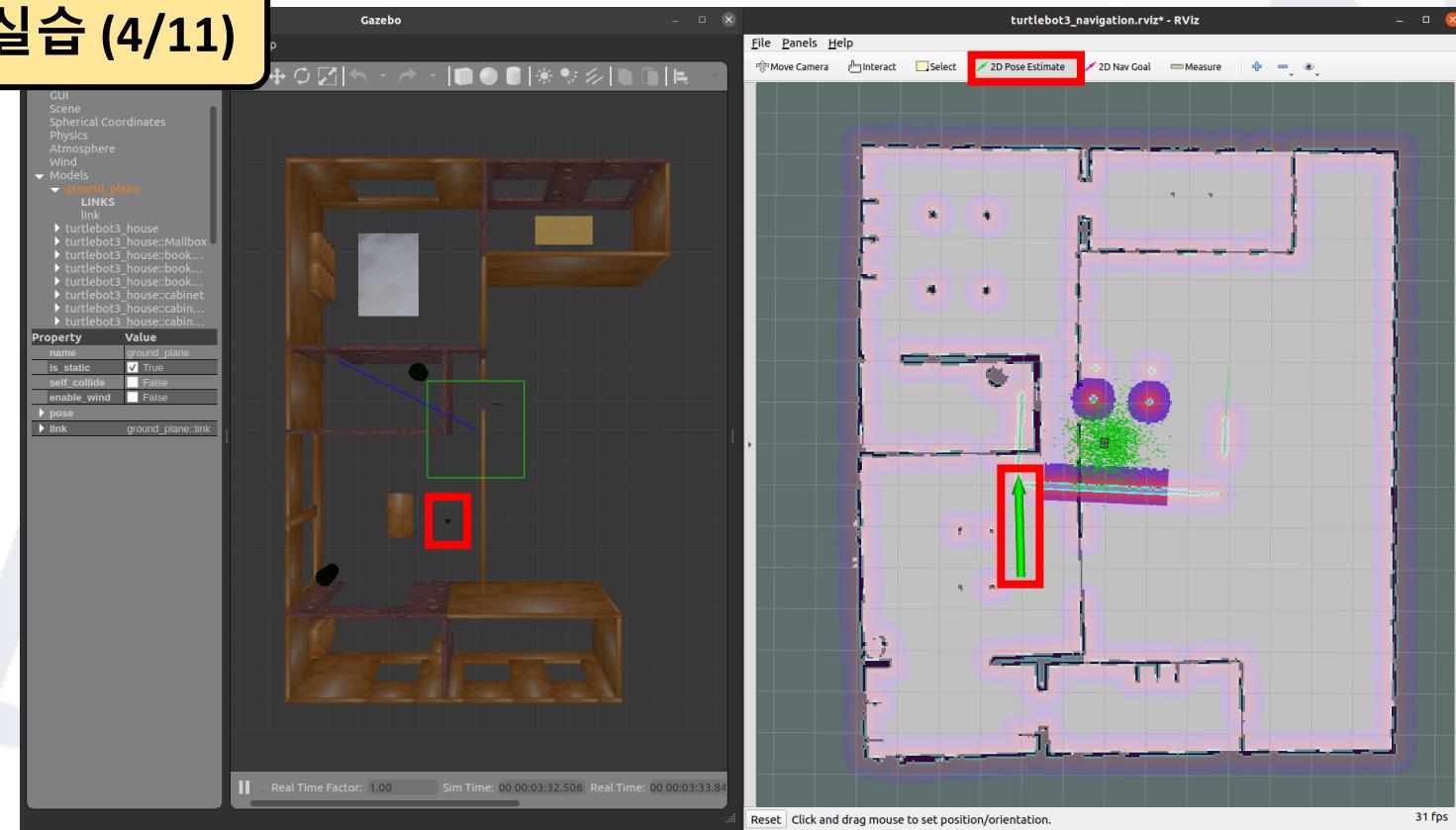
(→ 보정된 Occupancy Grid Map을 기반으로 Turtlebot3 모델의 자율주행을 위한 Navigation 노드 실행)

(→ 위 빨간색의 경로는 로컬 PC마다 모두 다르므로, Map이 저장된 경로에서 명령어 “pwd”를 통해 경로 확인)

## 2. Navigation

### 3) ROS Navigation Stack 기반 자율주행 실습 (14/26)

실습 (4/11)

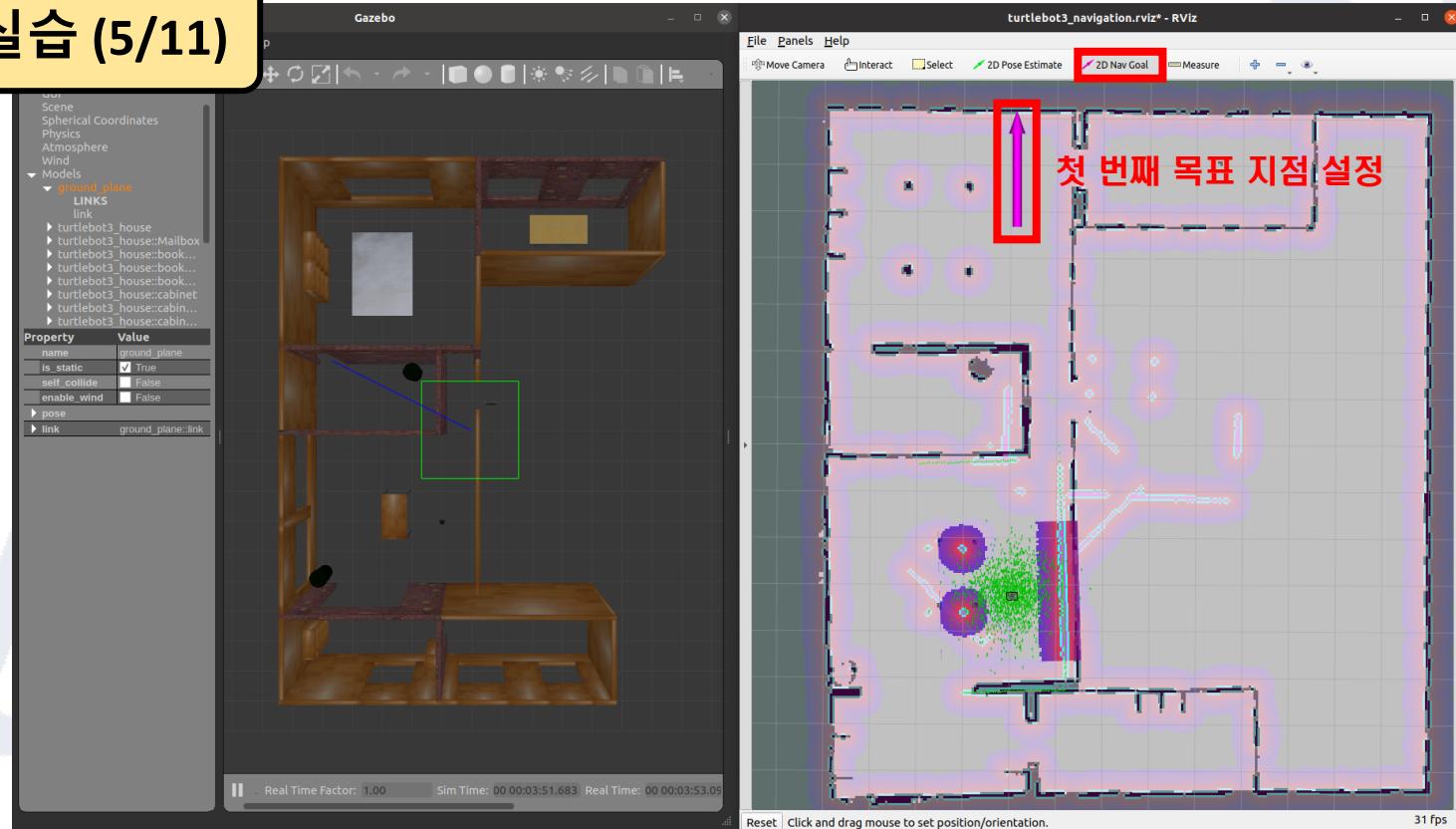


- 오른쪽 Rviz(Ros visualization)의 상단부 중앙에 위치한 “2D Pose Estimate”를 클릭한 후, 왼쪽 Gazebo 시뮬레이션 환경에 위치한 Turtlebot3 burger 모델이 위치한 곳과 동일한 Rviz 상의 위치에 마우스를 드래그하여 로봇의 초기 위치 설정

## 2. Navigation

### 3) ROS Navigation Stack 기반 자율주행 실습 (15/26)

실습 (5/11)

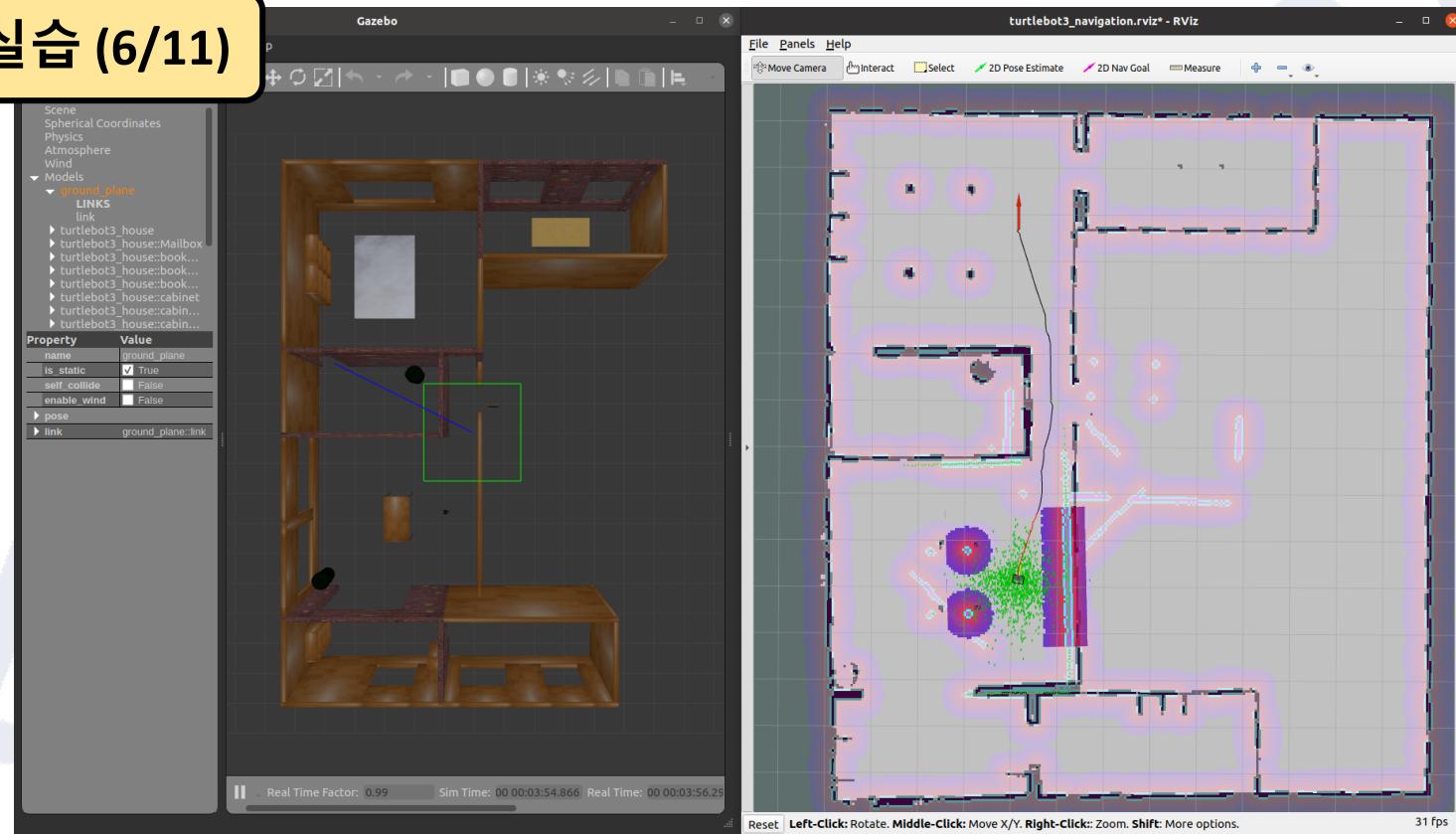


- 오른쪽 Rviz(Ros visualization)의 상단부에 위치한 “2D Nav Goal”를 클릭한 후, 앞서 설정한 “2D Pose Estimate” 과정과 동일하게 Rviz 상의 원하는 위치에 마우스를 드래그하여 로봇이 도달해야 하는 목표 위치와 설정

## 2. Navigation

### 3) ROS Navigation Stack 기반 자율주행 실습 (16/26)

실습 (6/11)

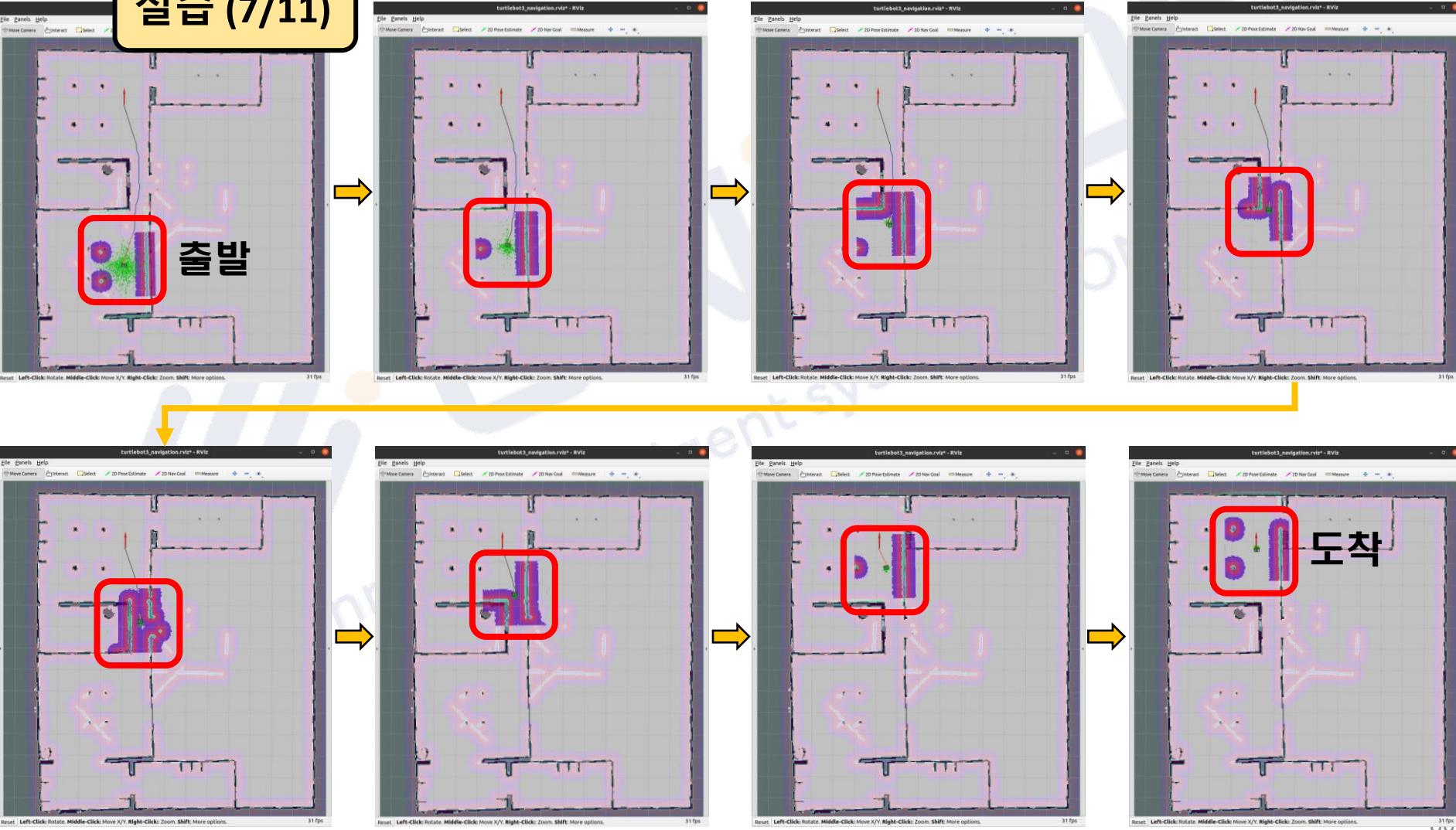


- 앞선 과정을 통해 ‘2D Nav Goal’ Pose를 설정하게 되면, ‘global\_planner’의 경로 탐색 과정을 통해 원하는 목적지까지 로봇이 주행해야 할 경로를 생성.
- 또한, 생성된 경로를 추종하기 위해 ‘local\_planner’를 통한 제어 입력을 생성함으로써 자율주행 수행.

## 2. Navigation

### 3) ROS Navigation Stack 기반 자율주행 실습 (17/26)

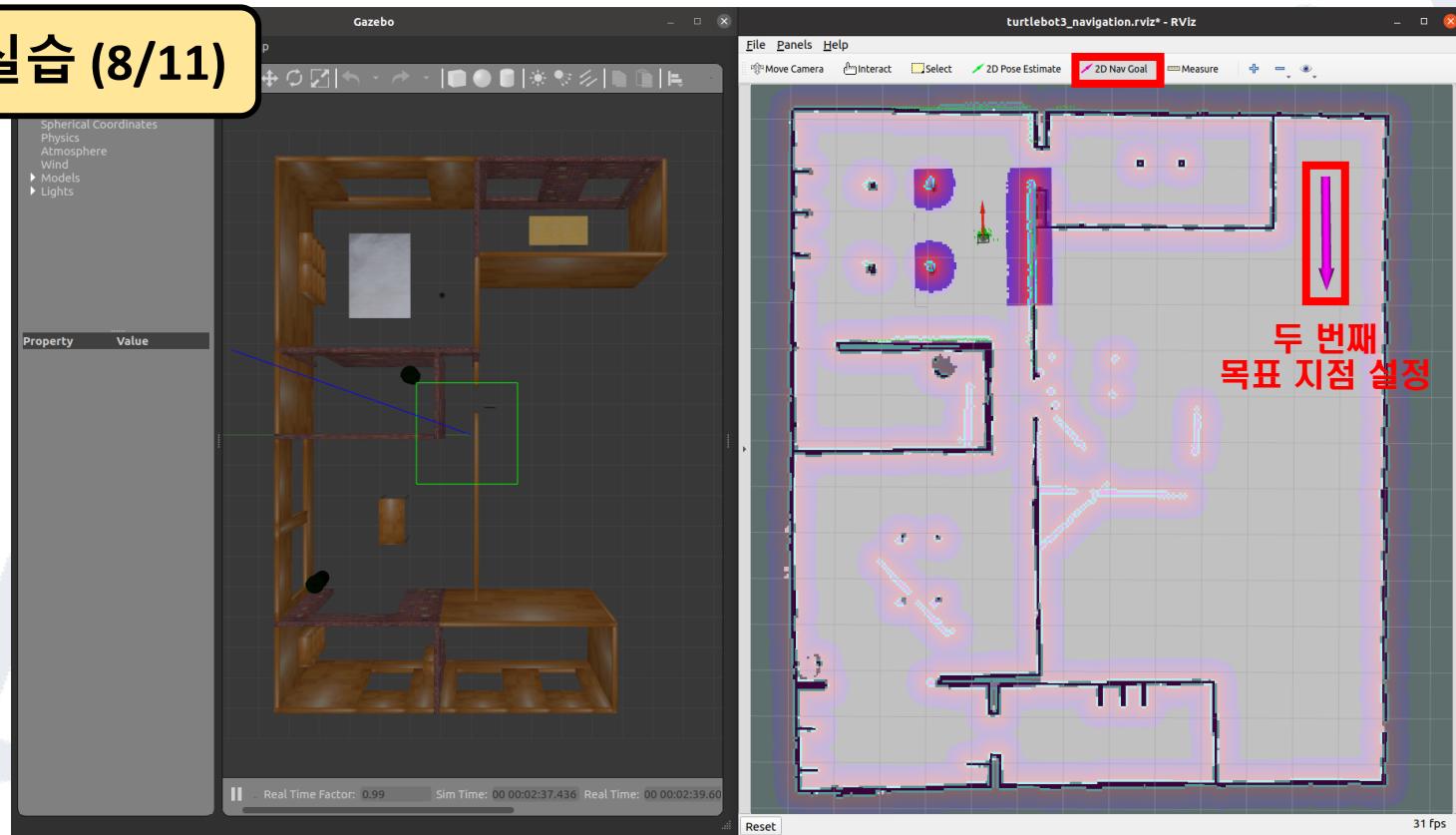
실습 (7/11)



## 2. Navigation

### 3) ROS Navigation Stack 기반 자율주행 실습 (18/26)

실습 (8/11)

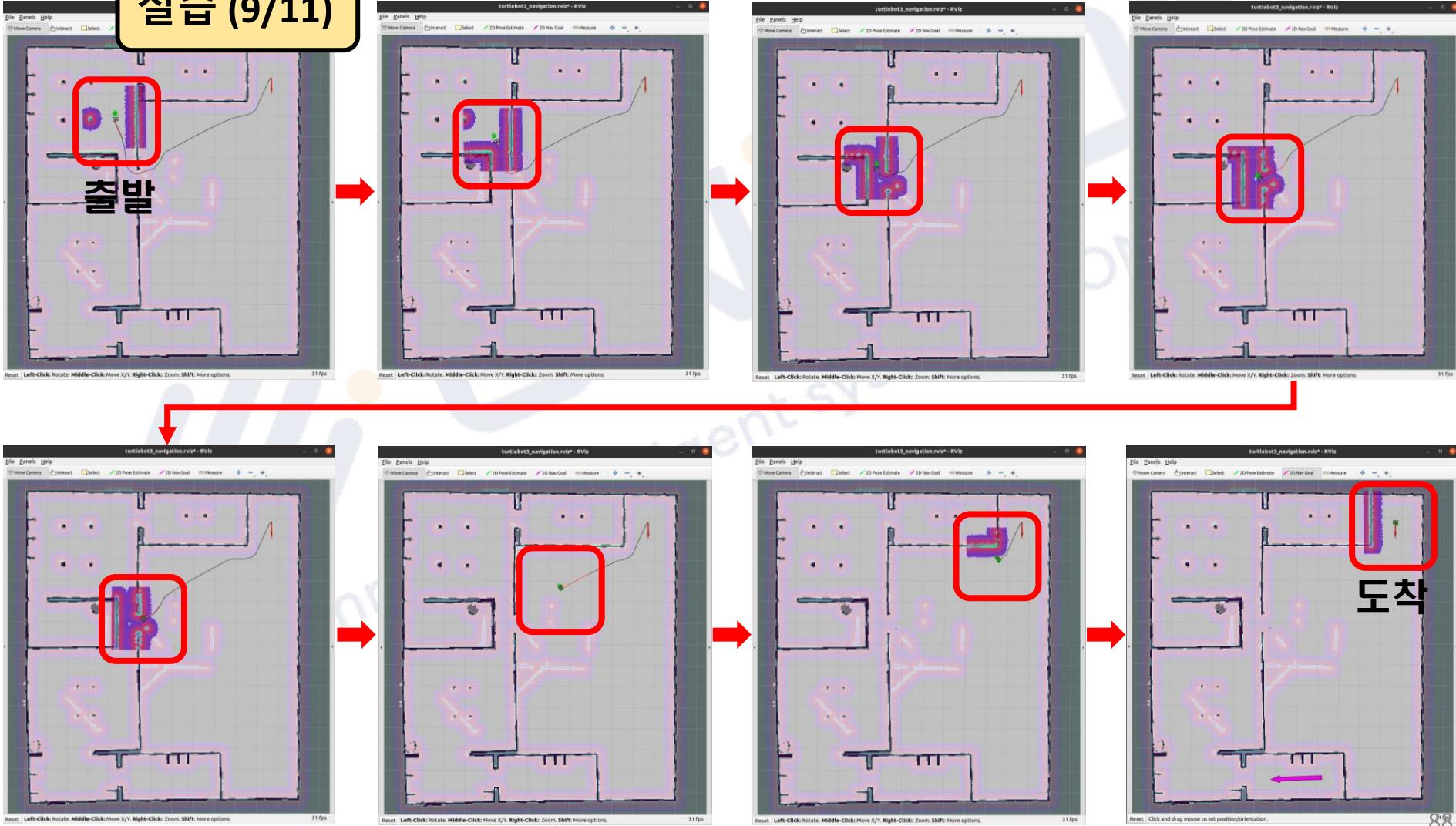


- 오른쪽 Rviz(Ros visualization)의 상단부에 위치한 “2D Nav Goal”를 클릭한 후, 앞서 설정한 “2D Pose Estimate” 과정과 동일하게 Rviz 상의 원하는 위치에 마우스를 드래그하여 로봇이 도달해야 하는 목표 위치와 설정

## 2. Navigation

### 3) ROS Navigation Stack 기반 자율주행 실습 (19/26)

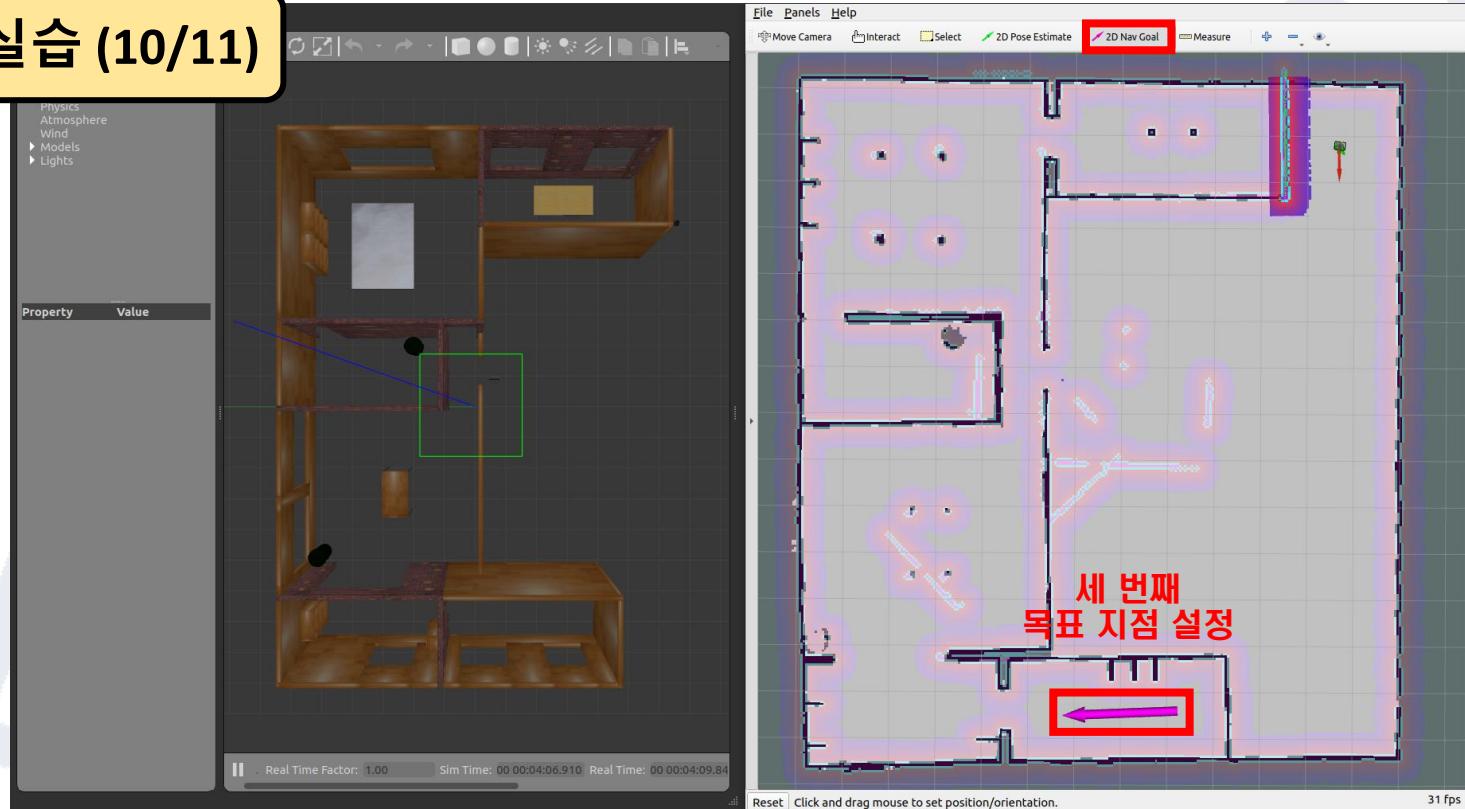
실습 (9/11)



## 2. Navigation

### 3) ROS Navigation Stack 기반 자율주행 실습 (20/26)

실습 (10/11)

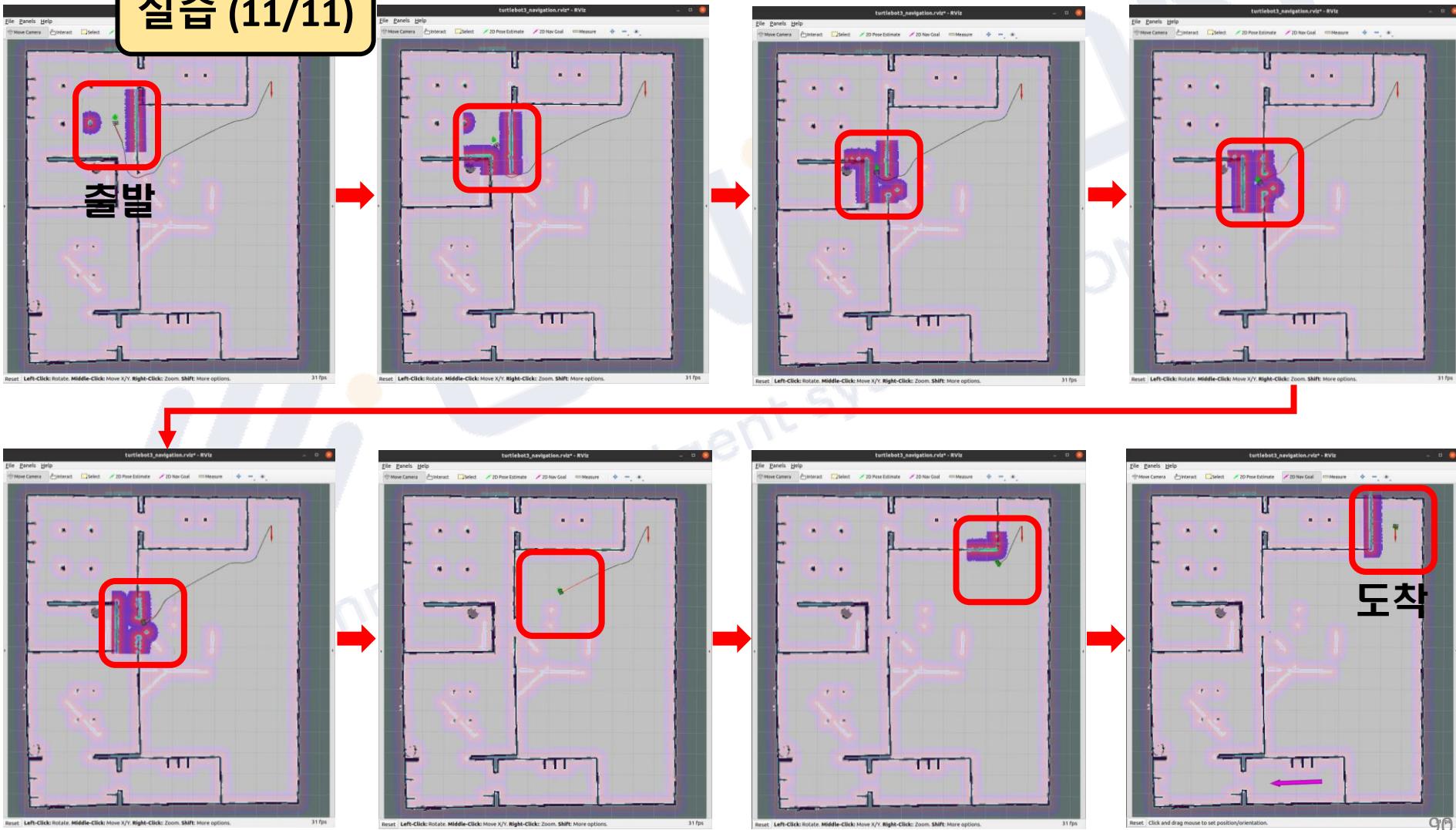


- 오른쪽 Rviz(Ros visualization)의 상단부에 위치한 “2D Nav Goal”를 클릭한 후, 앞서 설정한 “2D Pose Estimate” 과정과 동일하게 Rviz 상의 원하는 위치에 마우스를 드래그하여 로봇이 도달해야 하는 목표 위치와 설정

## 2. Navigation

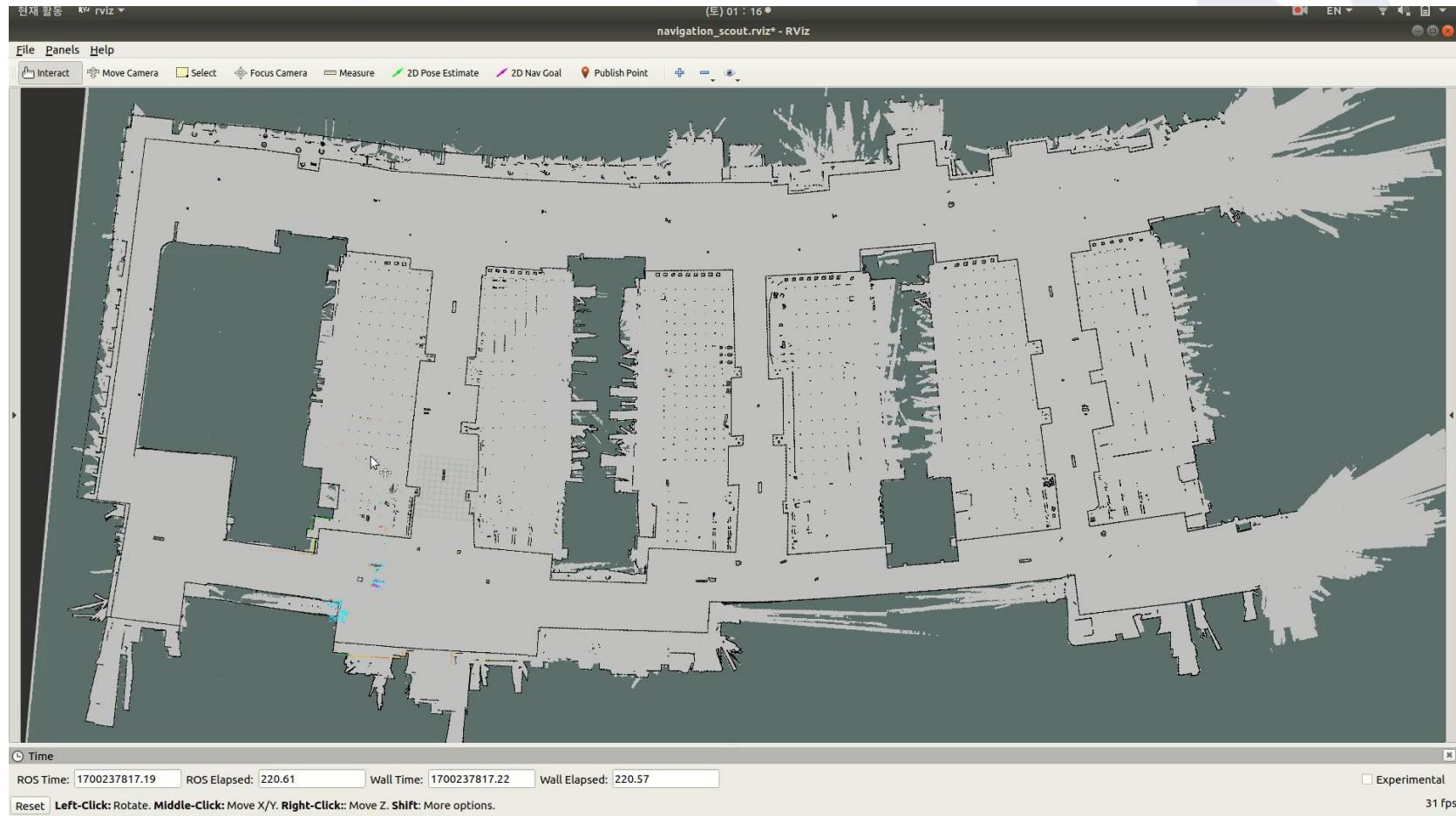
### 3) ROS Navigation Stack 기반 자율주행 실습 (21/26)

실습 (11/11)



## 2. Navigation

### 3) ROS Navigation Stack 기반 자율주행 실습 (22/26)

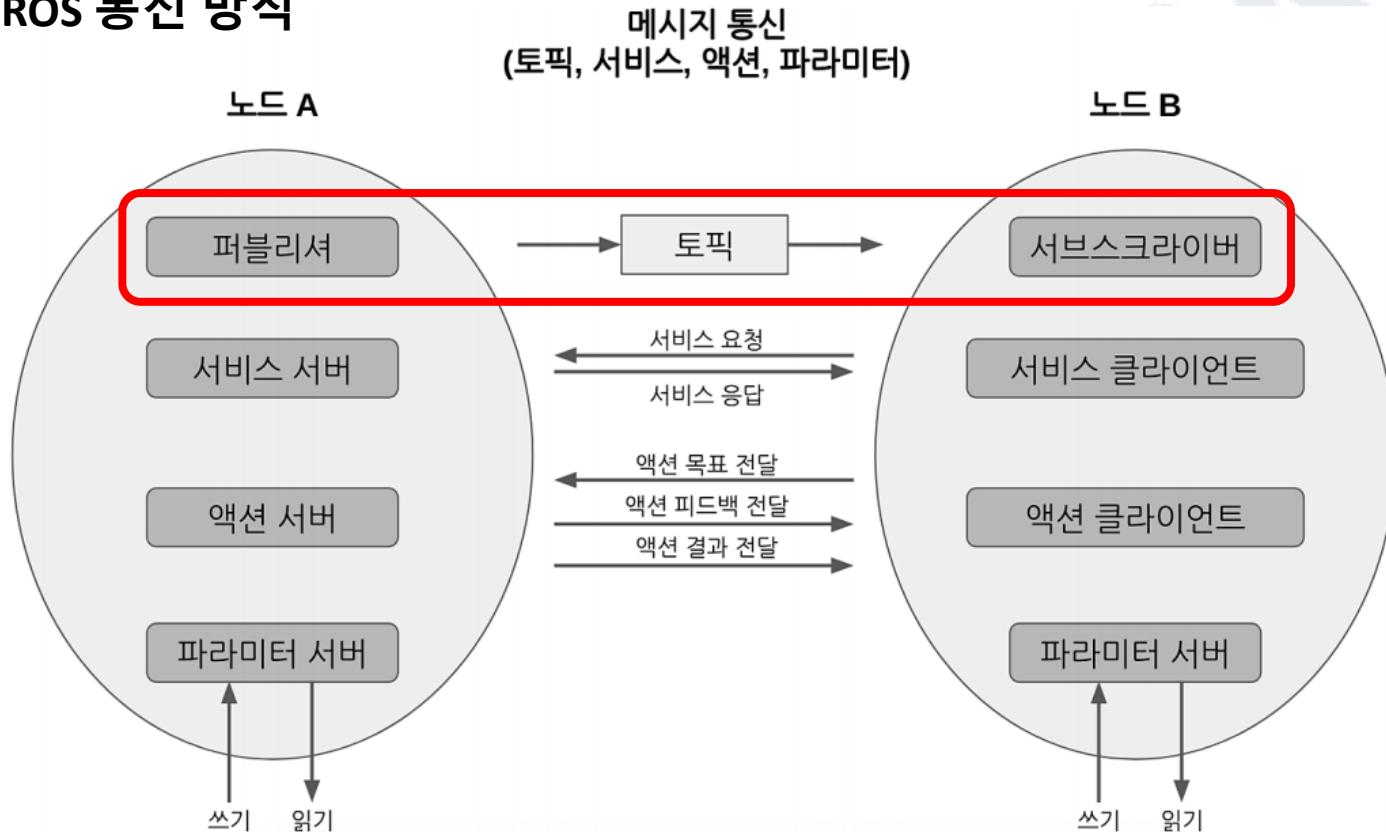


- SLAM을 통해 Mapping 된 각 구역의 Map을, GIMP 편집기를 통해 보정 및 병합된 Map(A~D 구역)을 기반으로 주행 테스트 수행.

## 2. Navigation

### 3) ROS Navigation Stack 기반 자율주행 실습 (23/26)

- ROS 통신 방식



- 각 노드(Node)는 메시지(토픽, Topic)을 통해 데이터를 송·수신

## 2. Navigation

### 3) ROS Navigation Stack 기반 자율주행 실습 (24/26)

- Topic (토픽)

- ROS에서 가장 많이 사용하는 메시지의 종류
- Publisher 노드는 자신의 토픽을 마스터에 등록한 후, 해당 토픽에 대한 메시지를 송신
- Subscriber 노드는 수신하고자 하는 메시지를 송신하는 Publisher에 대한 정보를 마스터에 요청하고, 이를 바탕으로 Publisher노드와 연결하여 토픽으로 메시지를 교환

- Publisher

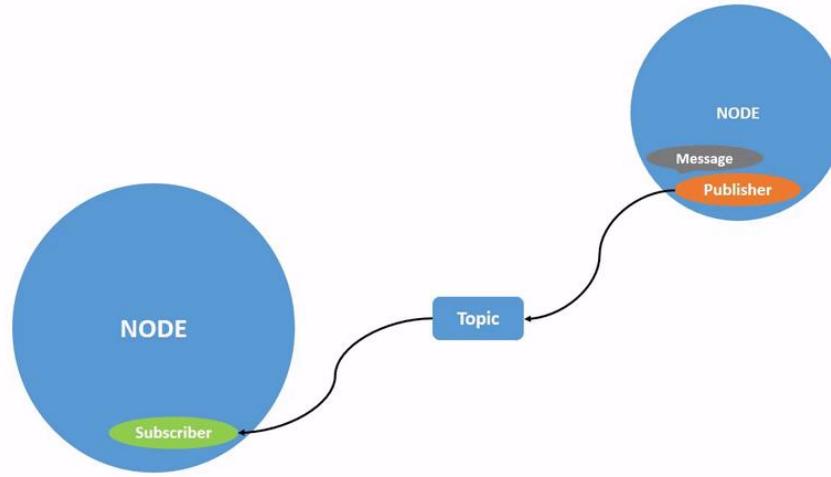
- 토픽 메시지를 송신하는 것을 publish라 하며, 그 역할을 Publisher가 수행
- 한 노드에서 여러 Publisher가 선언될 수 있음

- Subscriber

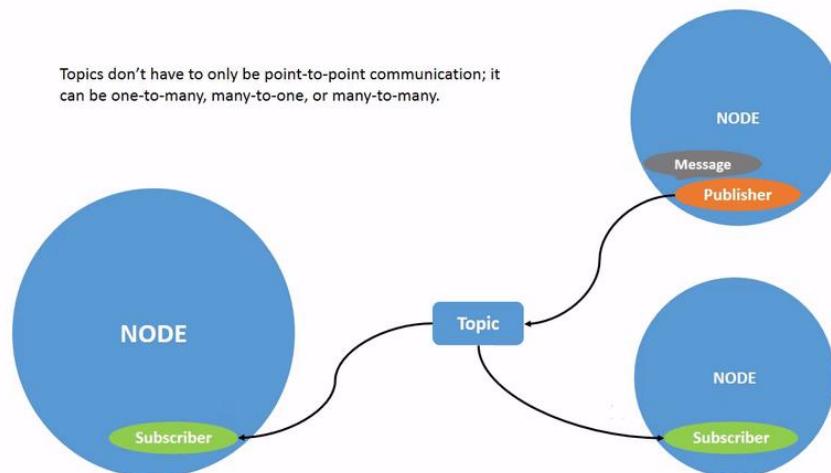
- 토픽 메시지를 수신하는 것을 subscribe라고 하며, 그 역할을 Subscriber가 수행
- 한 노드에서 여러 Subscriber가 선언될 수 있음

## 2. Navigation

### 3) ROS Navigation Stack 기반 자율주행 실습 (25/26)



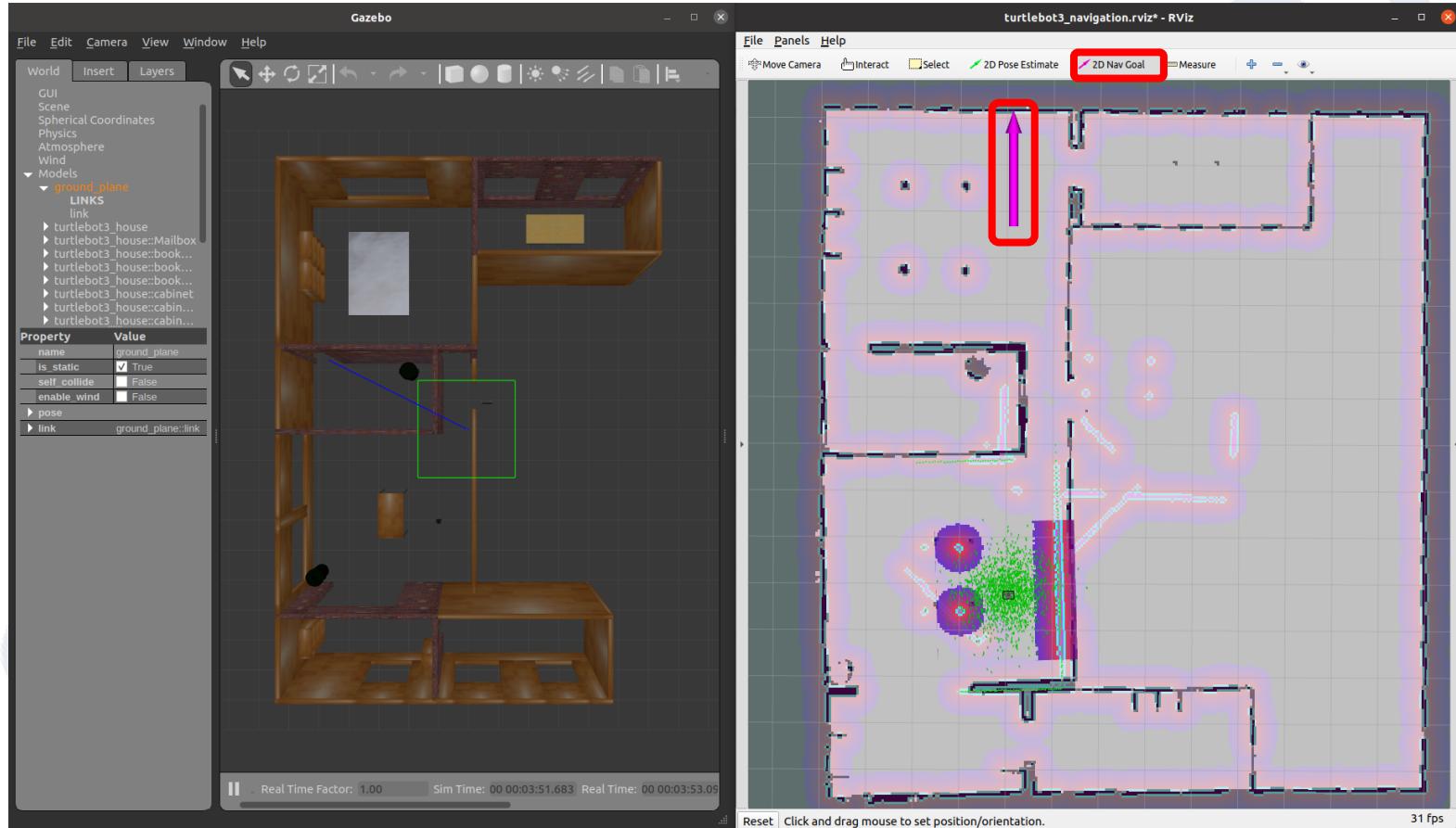
Topics don't have to only be point-to-point communication; it can be one-to-many, many-to-one, or many-to-many.



- 1 대 1 통신, 1 대 N 통신, N 대 1 통신, N 대 N 통신 모두 가능

## 2. Navigation

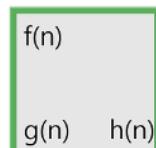
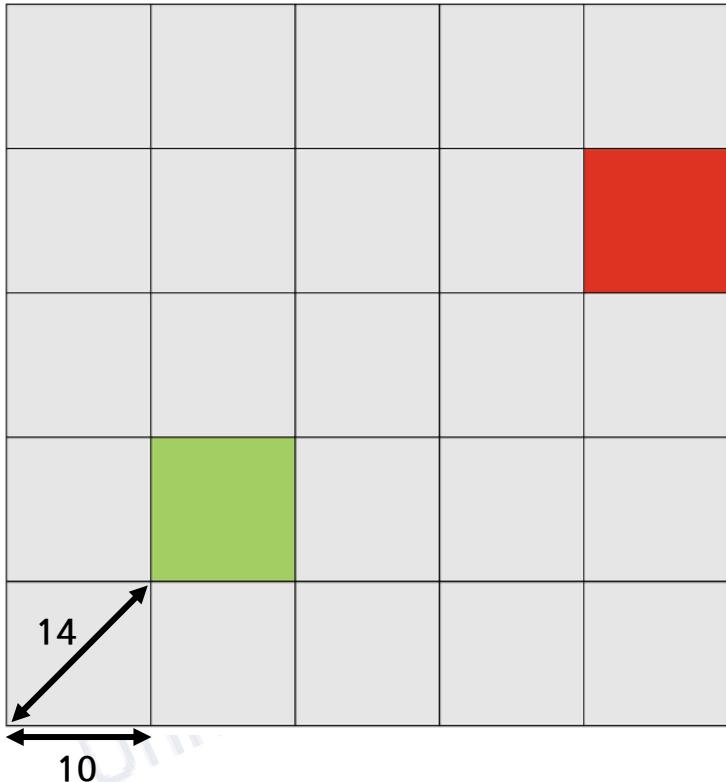
### 3) ROS Navigation Stack 기반 자율주행 실습 (26/26)



- Gazebo 시뮬레이션 환경에 위치한 Turtlebot3 burger 모델이 **도달해야 할 목적지를 설정하기 위한 “2D Nav Goal”은 Topic(토픽)의 한 예시**

## 2. Navigation

### Global planner – A\* algorithm



f(n) : total cost  
 g(n): cost from start  
 h(n): cost to end

 Start node

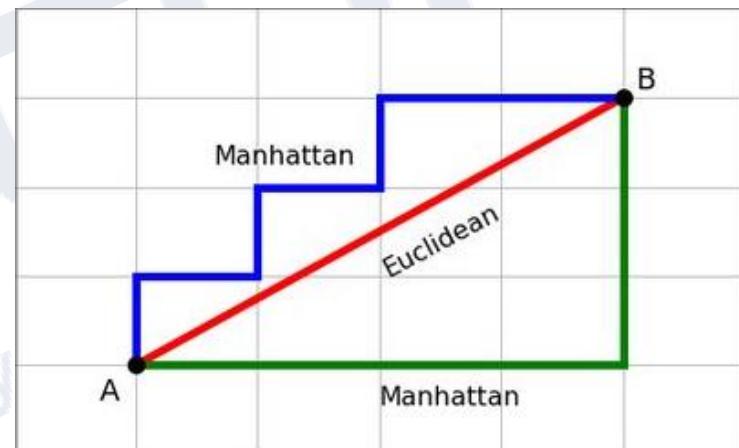
 End node

- g(n)
- h(n)
- f(n)
- Parent node
- Open list
- Closed list

## 2. Navigation

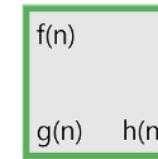
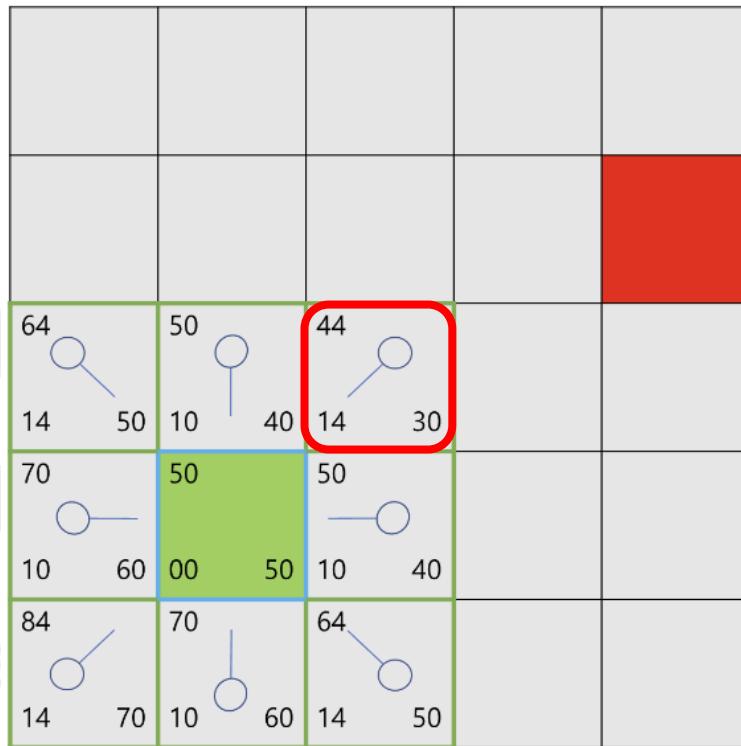
### Global planner – A\* algorithm

- $g(n)$   
시작 노드로부터 현재 위치한 노드까지의 거리 값
- $h(n)$   
현재 위치한 노드로부터 목표 노드까지의 거리 값
- $f(n)$   
 $g(n)$  값과  $h(n)$  값을 더한 값
- Parent node  
현재 노드를 탐색하기 전에 위치했던 노드
- Open list  
탐색 중인(또는 탐색했던) 노드들을 저장하는 리스트
- Closed list  
탐색한 노드 가운데, 가장  $f(n)$  값이 작은 노드들을 저장하는 리스트



## 2. Navigation

### Global planner – A\* algorithm



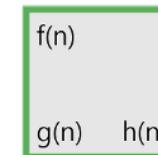
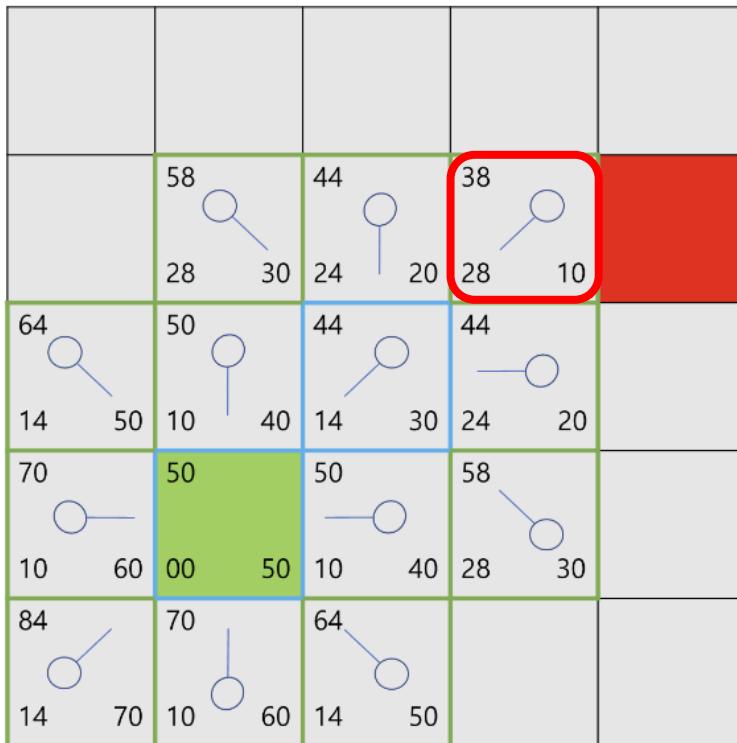
f(n) : total cost  
 g(n): cost from start  
 h(n): cost to end

Start node

End node

## 2. Navigation

### Global planner – A\* algorithm

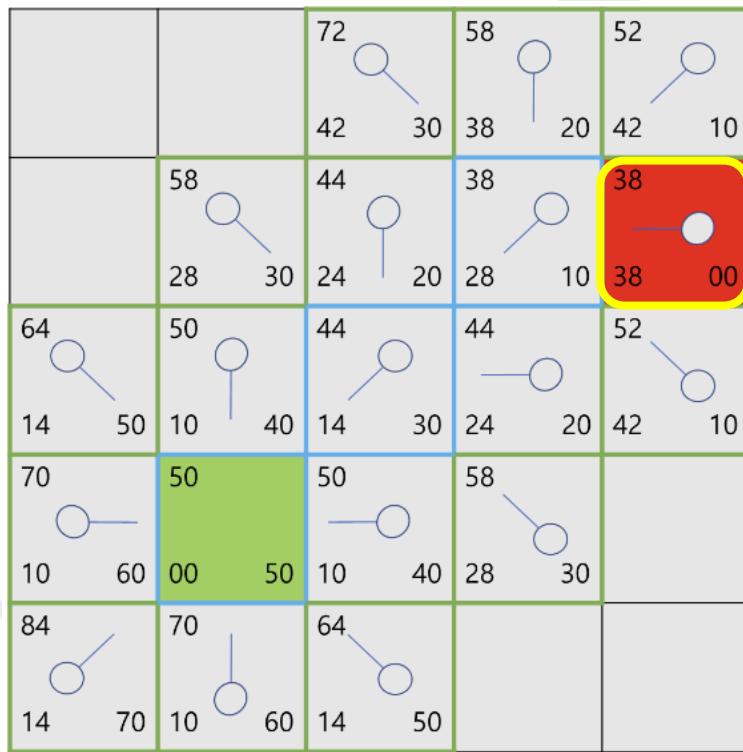


**Start node**

**End node**

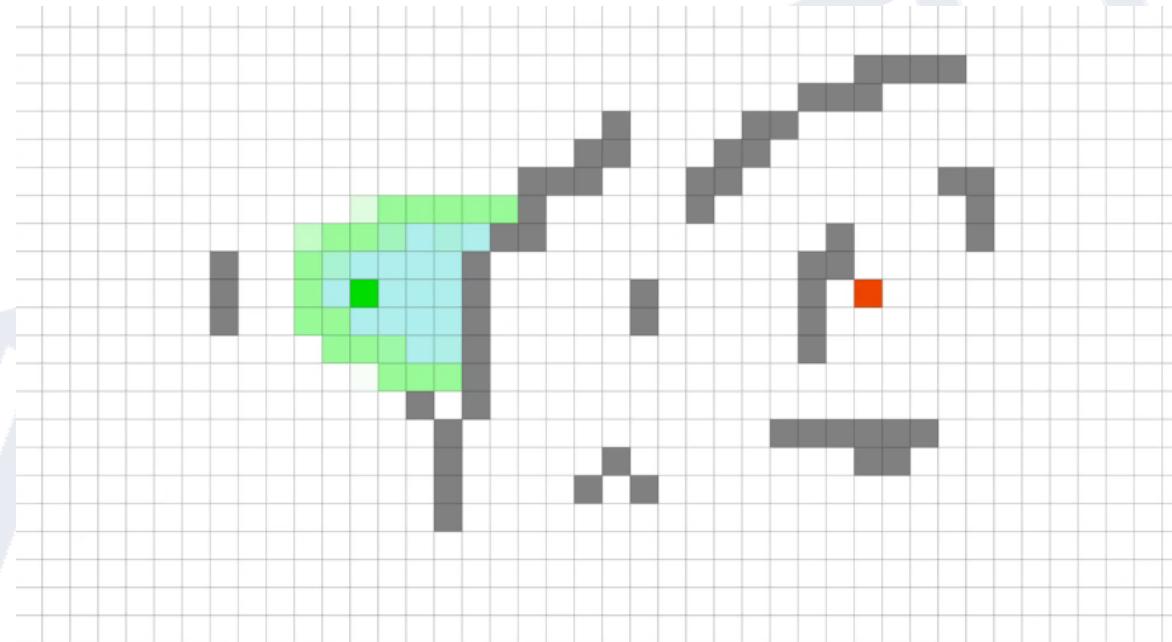
## 2. Navigation

### Global planner – A\* algorithm



## 2. Navigation

### Global planner – A\* algorithm



 : Start position

 : Goal position

 : Static obstacle

$$F(n) = G(n) + H(n)$$

$$H(n) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

# QnA

E-mail : [leehwasu9696@inu.ac.kr](mailto:leehwasu9696@inu.ac.kr)  
[mkbae97@gmail.com](mailto:mkbae97@gmail.com)

Mobile : 010-8864-5585  
010-6218-9259



청주대학교  
미래형자동차 인력양성사업단

자율주행 분야

# 단기집중 교육과정



6월 11일(화)  
오후 4시~8시

리눅스 설치,  
ROS 설치 및 ROS 개발환경 구축

융합관  
410호

\*개인 노트북 필수 지참(대여불가)

온라인  
사전교육

Python 교육(ROS 기반)



7월 2일~  
5일(화~금)  
오후 1시~6시

라이다 센서 교육

새천년종합정보관  
107A호

(경진대회 참여학생)

7월 8일~  
10일(월~수)  
오후 1시~6시

ROS 활용 교육(Python 기반)

융합관  
410호

\*개인 노트북 필수 지참(대여불가)

7월 15일~  
17일(월~수)  
오후 1시~6시

Python OpenCV 설치 및 응용

새천년종합정보관  
409호

\*개인 노트북 필수 지참(대여불가)



청주대학교  
CHEONGJU UNIVERSITY

**Thank you.**