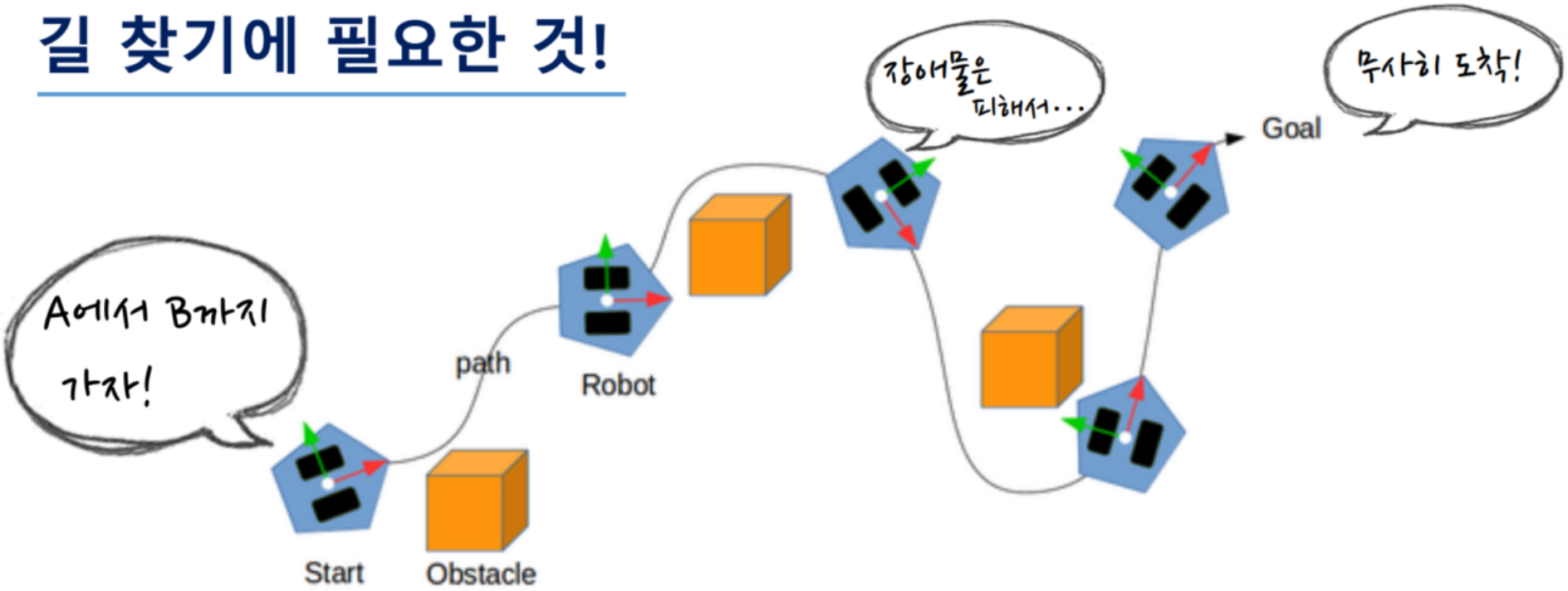
Turtlebot

길 찾기

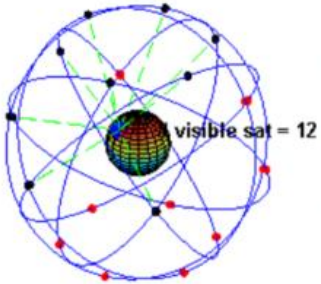
길 찾기에 필요한 것!



① **위치**: 로봇의 위치 계측/추정하는 기능
② **센싱**: 벽, 물체 등의 장애물의 계측하는 기능
③ **지도**: 길과 장애물 정보가 담긴 지도
④ **경로**: 목적지까지 최적 경로를 계산하고 주행하는 기능

① 위치: 로봇의 위치 계측/추정하는 기능

- GPS (Global Positioning System)



  - 오차
  - 날씨
  - 실외

- Indoor Positioning Sensor
  - Landmark (Color, IR Camera)
  - Indoor GPS
  - WiFi SLAM
  - Beacon

Estimote (Beacon)

StarGazer

Vicon MX

② 센싱: 벽, 물체 등의 장애물의 계측하는 기능

- 거리센서
  - LRF, 초음파센서, 적외선 거리센서(PSD)

- 비전센서
  - 스테레오 카메라, 모노 카메라, 전 방향 옴니 카메라
- Depth camera
  - SwissRanger, Kinect-2
  - RealSense, Kinect, Xtion, Carmine(PrimeSense), Astra

③ 지도: 길과 장애물 정보가 담긴 지도

- 로봇은 길을 찾아가기 위해 지도가 필요하다!
- 지도
  - 도로와 같은 기반 시설의 경우 디지털 지도 OK!
  - 병원, 카페, 회사, 가정집의 지도?
  - 탐사, 붕괴된 위험지역의 지도?
- **지도? 없으면 만들자!**
- **SLAM**

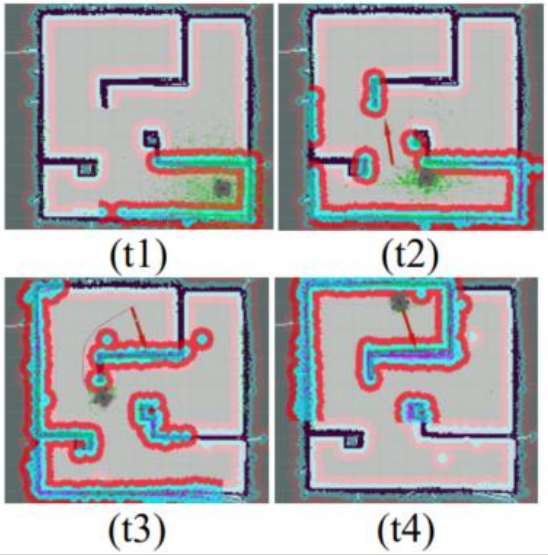
  (Simultaneous Localization And Mapping)

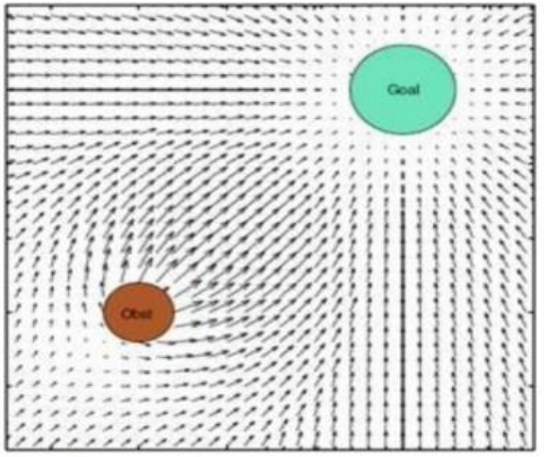
  같이 여긴 어디? 지도 만들자

④ 경로: 목적지까지 최적 경로를 계산하고 주행하는 기능

- 내비게이션(Navigation)
- 위치 추정 (Localization / Pose estimation)
- 경로 탐색/계획 (Path search and planning)

- Dynamic Window Approach (DWA)
- A* 알고리즘 (A Star)
- 포텐셜 장(Potential Field)
- 파티클 필터 (Particle Filter)
- 그래프 (Graph)





https://students.cs.byu.edu/~cs470ta, http://vimeo.com/3423169

① 위치

② 센싱

③ 지도

④ 경로

위치+센싱 → **지도**

SLAM

위치+센싱+지도 → **경로**

Navigation

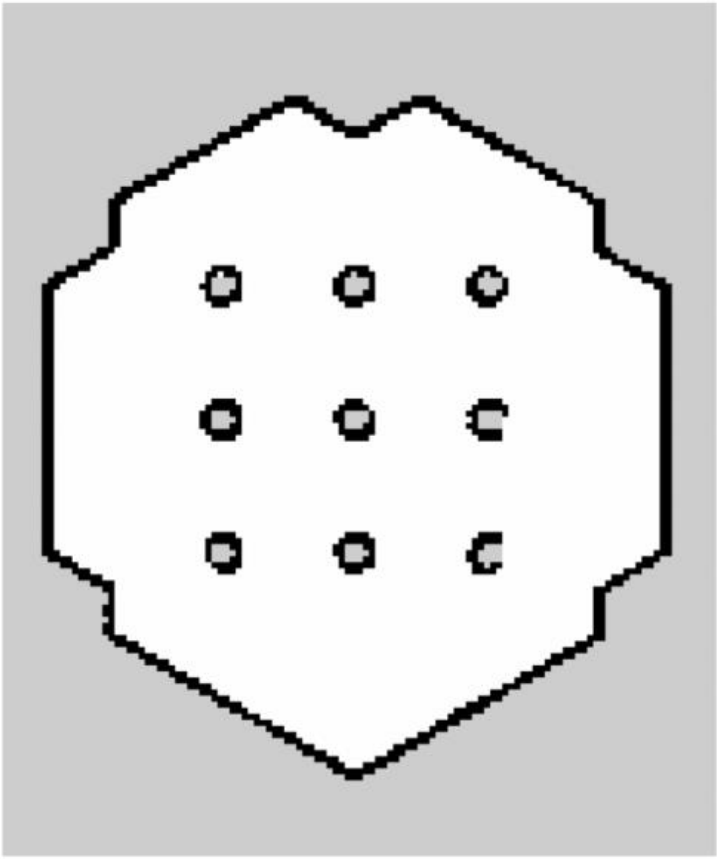
SLAM & Navigation

- SLAM(Simultaneous Localization And Mapping)
  - 동시적 위치 추정 및 지도 작성

- Navigation
  - 차량 자동 항법 장치

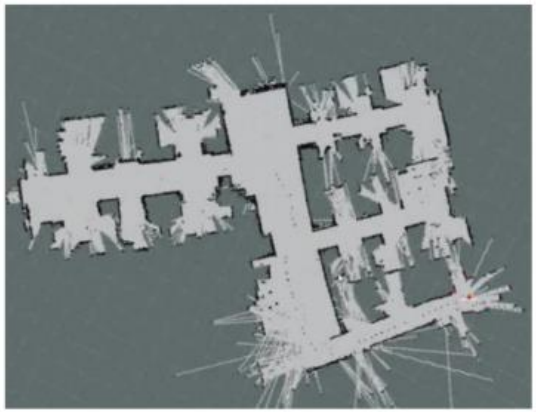
지도 작성

- 완성된 지도



2차원 **점유 격자 지도**
(OGM, Occupancy Grid Map)

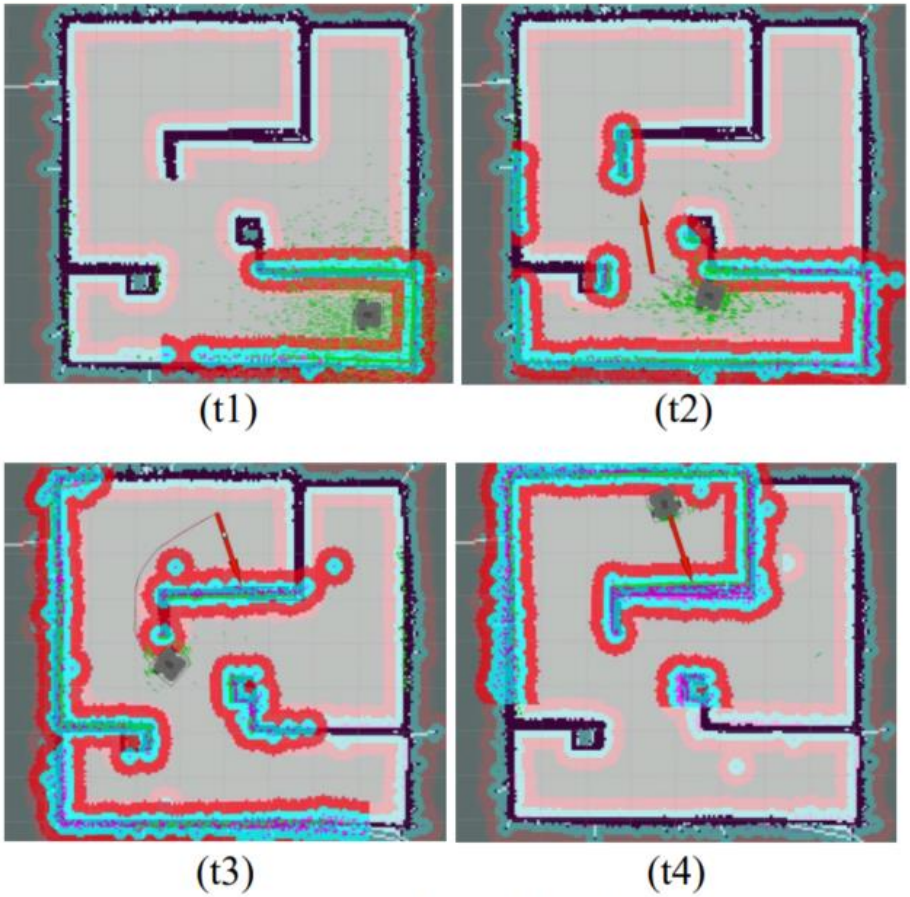
- 흰색 = 로봇이 이동 가능한 **자유 영역 (free area)**
- 흑색 = 로봇이 이동 불가능한 **점유 영역 (occupied area)**
- 회색 = 확인되지 않은 **미지 영역 (unknown area)**



위치 추정(localization) | Kalman filter, Particle filter, Graph, Bundle adjustment

- 파티클 필터(Particle Filter)
- 파티클 필터는 시행 착오(try-and-error)법을 기반으로한 시뮬레이션을 통하여 예측하는 기술으로 대상 시스템에 확률 분포로 임의로 생성된 추정값을 파티클(입자) 형태로 나타낸다.

1) 초기화(initialization)
2) 예측(prediction)
3) 보정(update)
4) 위치 추정(pose estimation)
5) 재추출(Resampling)



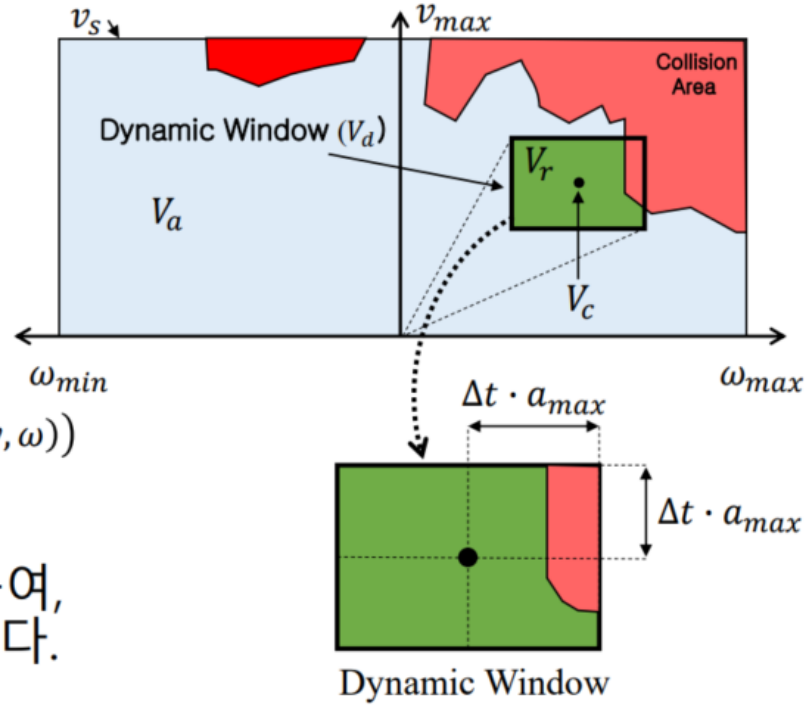
S. Thrun, W. Burgard, and D. Fox. Probabilistic Robotics. MIT Press, Cambridge, MA, 2005.

DWA(Dynamic Window Approach

- **Dynamic Window Approach** (local plan에서 주로 사용)
- 로봇의 속도 탐색 영역(velocity search space)에서 로봇과 충돌 가능한 장애물을 회피하면서 목표점까지 빠르게 다다를 수 있는 속도를 선택하는 방법

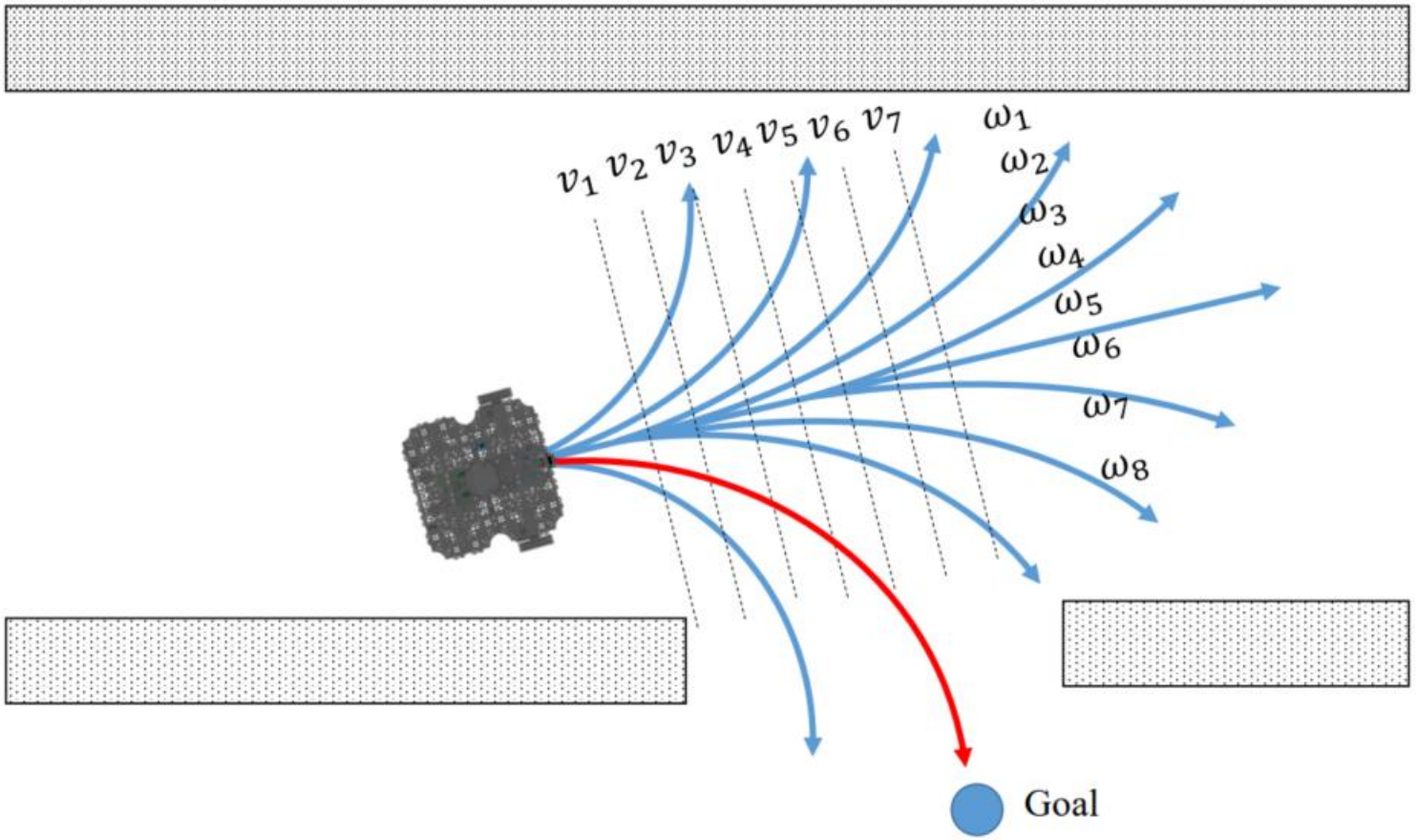
  - v (병진속도), ω (회전속도)
  - V_s: 가능 속도 영역
  - V_a: 허용 속도 영역
  - V_r: 다이나믹 윈도우 안의 속도 영역
  - $G_{(v,\omega)} = \sigma\big(\alpha \cdot heading(v,\omega) + \beta \cdot dist(v,\omega) + \gamma \cdot velocity(v,\omega)\big)$

- 목적함수 G는 로봇의 방향, 속도, 충돌을 고려하여, 목적함수가 최대가 되는 속도 v, ω 를 구하게 된다.



D. Fox, W. Burgard and S. Thrun, The dynamic window approach to collision avoidance, IEEE Robotics & Automation Magazine

DWA


v_1 v_2 v_3 v_4 v_5 v_6 v_7
ω_1
ω_2
ω_3
ω_4
ω_5
ω_6
ω_7
ω_8
Goal

Turtlebot
gazebo 실습

배치파일 확인

- 배치파일 확인

```
$ code ~/.bashrc
```

```
export ROS_DOMAIN_ID=30
export TURTLEBOT3_MODEL=waffle_pi
export LDS_MODEL=LDS-02
source /opt/ros/foxy/setup.bash
source ~/Workspaces/ros2_ws/install/setup.bash
```

- 배치파일 적용
  - 수정된 내용을 적용하려면 다음 명령어를 실행해야 함

```
$ source ~/.bashrc
```

Gazebo

- Gazebo : 3차원 시뮬레이터
  - 물리 엔진을 탑재, 로봇, 센서, 환경 모델 등을 지원
  - 실물 로봇 대신 시뮬레이션 환경을 구동해서 테스트 함

```
$ gazebo
```

- Turtlebot3 Gazebo 실행
  - 아래 코드를 처음 실행시 오래 걸림

```
$ ros2 launch turtlebot3_gazebo empty_world.launch.py
```

  - turtlebot3_gazebo 설치가 안되어 있을 경우 설치

```
$ sudo apt update
$ sudo apt upgrade
$ sudo apt install ros-foxy-turtlebot3-gazebo
```

Turtlebot3 움직이기

- Turtlebot3 Gazebo 실행

```
$ ros2 launch turtlebot3_gazebo empty_world.launch.py
```

- Turtlebot3 teleop 키보드 실행
  - 아래 명령어를 실행한 창에 커서를 두고,
  - w키를 누르면 앞으로 가고, s키를 누르면 멈춘다.

```
$ ros2 run turtlebot3_teleop teleop_keyboard
```

- Turtlebot3 worlds 실행하기
  - /opt/ros/foxy/share/turtlebot3_gazebo/worlds에 있는 world들

```
$ ros2 launch turtlebot3_gazebo empty_world.launch.py
$ ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
$ ros2 launch turtlebot3_gazebo turtlebot3_house.launch.py
```

google cartographer

- [창1] Turtlebot3 Gazebo 실행

```
$ ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
```

- [창2] Turtlebot3 cartographer 실행
  - Robot의 환경에 대한 각 Pixel의 Occupancy Status(0~100 정수)
    - 흰색은 자유 공간, 0: 완전한 Free Space
    - 검은색은 점유 공간, 100: 완전한 Occupied Space
    - 회색은 미지의 공간, -1: Unknown Space
  - 시뮬레이션인 경우 use_sim_time:=True를 작성함

```
$ ros2 launch turtlebot3_cartographer
cartographer.launch.py use_sim_time:=True
```

Turtlebot3 SLAM

- [창1] Turtlebot3 Gazebo 실행

```
$ ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
```

- [창2] Turtlebot3 cartographer 실행

```
$ ros2 launch turtlebot3_cartographer
cartographer.launch.py use_sim_time:=True
```

- [창3] Turtlebot3 teleop 키보드 실행

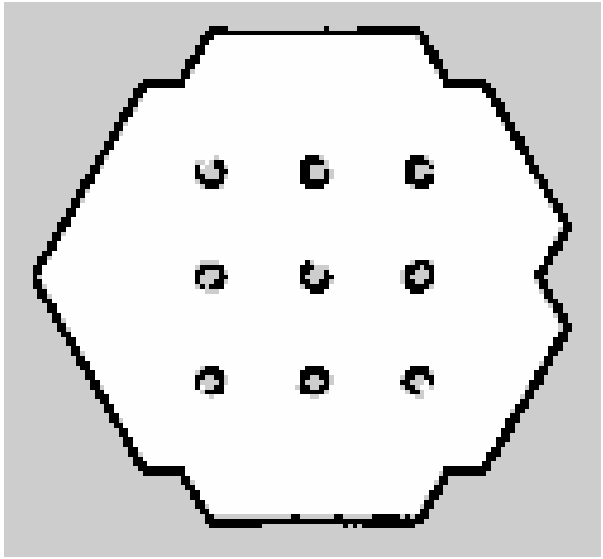
```
$ ros2 run turtlebot3_teleop teleop_keyboard
```

- [창4] 맵 저장

```
$ ros2 run nav2_map_server map_saver_cli -f ~/map
```

지도(Map)

- [파일1] ~/map.pgm
  - PGM(Portable Gray Map)
  - 흰색 - 자유 공간
  - 검정색 - 점유된 공간
  - 회색 - 미지의 공간



- [파일2] ~/map.yaml
  - image: 생성된 Map의 위치
  - mode: trinary 셀이 세 가지 상태 중 하나
  - resolution: Map 해상도(meter / pixel)
  - origin: Map의 좌측 하단 Pixel의 2D 좌표
  - negate: Map의 색 결정
  - occupied_thresh: 이 값보다 크면 Occupied
  - free_thresh: 이 값보다 크면 Free Space

```
image: map.pgm
mode: trinary
resolution: 0.05
origin: [-1.25, -2.41, 0]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.25
```

Turtlebot3 Navigation

- [창1] Turtlebot3 Gazebo 실행

```
$ ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
```

- [창2] Turtlebot3 navigation2 실행
  - 아래 코드 실행시 새로운 창인 Rviz(3차원 시각화 툴)가 열림
  - Rviz는 레이저, 카메라 등의 센서 데이터를 시각화 함
  - use_sim_time:=True 는 시뮬레이션 실행시 작성해야 함
  - map의 위치 작성시 '~' 대신 '$HOME'으로 작성해야 함

```
$ ros2 launch turtlebot3_navigation2 navigation2.
launch.py use_sim_time:=True map:=$HOME/map.yaml
```

- [창3] Turtlebot3 teleop 키보드 실행

```
$ ros2 run turtlebot3_teleop teleop_keyboard
```

Turtlebot3 Rviz

- 위치 추초기 Pose 추정
  - Rviz 메뉴에서 [2D Pose Estimate] 버튼 클릭
  - 로봇의 현재 위치(클릭) & 화살표 방향 설정(드래그 앤 드롭)
  - teleop 실행 후, 이동하며 확률이 수렴하는 것 확인
  - 위치 추정이 끝나면 teleop 종료

- 탐색 목표 설정
  - Rviz 메뉴에서 [Navigation2 Goal] 버튼 클릭
  - 지도를 클릭하여 로봇의 목적지를 설정
  - 녹색 화살표를 로봇이 향하는 방향으로 드래그

- 중간 목표 거쳐서 최종 목표로 이동
  - [Waypoint mode] 클릭 후, [Navigation2 Goal] 방향 설정 - 반복
  - [Start Waypoint Following]으로 설정한 길따라 움직임

Turtlebot
real robot 실습

Bringup

- Remote PC의 터미널(Ctrl+Alt+T) 열기

```
$ ssh ubuntu@192.168.1.30
```

- Bring up 실행
  - TurtleBot3의 모든 장치들을 구동함
  - 터미널에 'Run!'이 출력되면 Bringup 성공

```
$ ros2 launch turtlebot3_bringup robot.launch.py
```

Turtlebot3 SLAM

- [창1] Turtlebot3 Bringup 실행

```
$ ssh ubuntu@192.168.1.30
$ ros2 launch turtlebot3_bringup robot.launch.py
```

- [창2] Turtlebot3 cartographer 실행

```
$ ros2 launch turtlebot3_cartographer
cartographer.launch.py
```

- [창3] Turtlebot3 teleop 키보드 실행

```
$ ros2 run turtlebot3_teleop teleop_keyboard
```

- [창4] 맵 저장

```
$ ros2 run nav2_map_server map_saver_cli -f ~/map
```

Turtlebot3 Navigation

- [창1] Turtlebot3 Gazebo 실행

```
$ ssh ubuntu@192.168.1.30
$ ros2 launch turtlebot3_bringup robot.launch.py
```

- [창2] Turtlebot3 navigation2 실행
  - map의 위치 작성시 '~' 대신 '$HOME'으로 작성해야 함

```
$ ros2 launch turtlebot3_navigation2 navigation2.
launch.py map:=$HOME/map.yaml
```

- [창3] Turtlebot3 teleop 키보드 실행

```
$ ros2 run turtlebot3_teleop teleop_keyboard
```

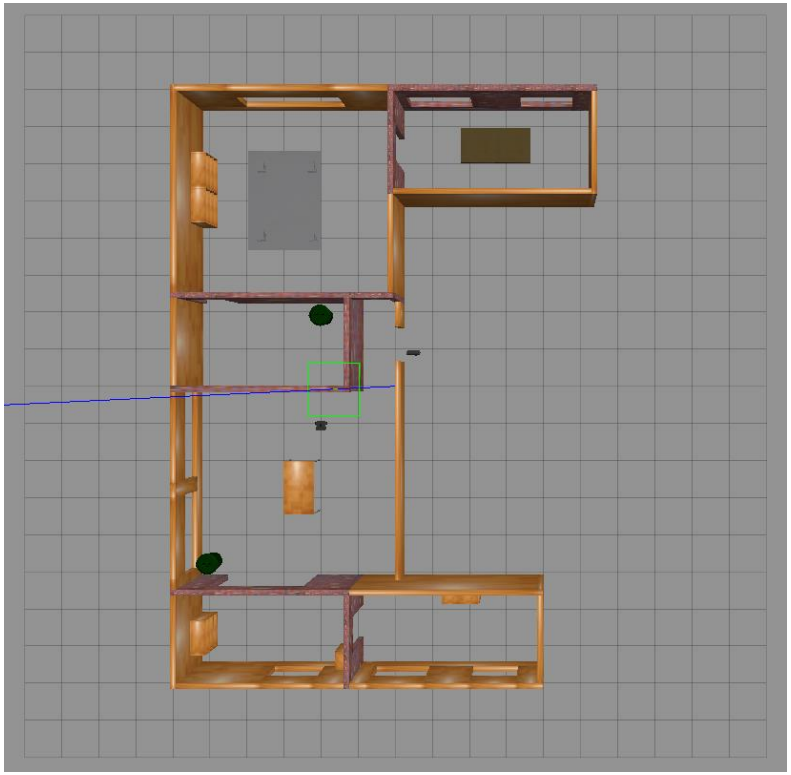
실습 과제

TurtleBot3 SLAM

과제1

TurtleBot3를 시뮬레이션환경에서 SLAM으로 지도를 만들기
turtlebot3_house의 지도를 만들고, 지도를 만드는 과정 캡쳐와
완성된 지도 파일을 제출하시오.

```
$ ros2 launch turtlebot3_gazebo turtlebot3_house.launch.py
```



TurtleBot3 Navigation

과제2

TurtleBot3를 시뮬레이션환경에서 Navigation 목표로 이동하기
turtlebot3_house의 지도를 기반으로 내비를 따라 목적지로 이동함
이동하는 장면을 캡쳐한 이미지를 제출하시오.

```
$ ros2 launch turtlebot3_gazebo turtlebot3_house.launch.py
```