

# ROS 를 활용한 SLAM과 내비게이션

2016. 09. 02

**ROBOTIS**

Open Source Team

Yoonseok Pyo

자~ 오늘의 주제는  
SLAM, Navigation  
입니다!

# Index

---

I. SLAM과 내비게이션

II. 모바일 로봇의 위치추정과 맵핑 (SLAM)

III. 모바일 로봇의 내비게이션 (Navigation)

# Index

---

## I. SLAM과 내비게이션

## II. 모바일 로봇의 위치추정과 맵핑 (SLAM)

## III. 모바일 로봇의 내비게이션 (Navigation)

# Simultaneous Localization And Mapping & Navigation

뭔 말이야?

———  
//

동시적 위치 추정 및 지도 작성  
&  
차량 자동 항법 장치

뭐야? OTL...  
더 어려워 보이잖아!

———  
//



좀~ 쉽게 갑시다!

길 찾기

어때요?

# 길...



## 길 「명사」

1. 사람이나 동물 또는 자동차 따위가 지나갈 수 있게 땅 위에 낸 일정한 너비의 공간.
2. 물 위나 공중에서 일정하게 다니는 곳.
3. 걸거나 탈것을 타고 어느 곳으로 가는 노정(路程).

-국립국어원 표준국어대사전-

# 길 찾기...



## 길 「명사」

1. 사람이나 동물 또는 자동차 따위가 지나갈 수 있게 땅 위에 낸 일정한 너비의 공간.
2. 물 위나 공중에서 일정하게 다니는 곳.
3. 걸거나 탈것을 타고 어느 곳으로 가는 노정(路程).

-국립국어원 표준국어대사전-

여행의 오랜 동반자

나침반과 지도

나침반도 없고  
지도도 없다면?



여긴 어디? 나는 누구?



# 나는 여기에 있다.

---

- 해, 달, 별의 위치만으로 떠나는 여행자
- 중국 4대 발명 나침반
- 나침반 진화
  - 자기 나침반
  - 전륜 나침반
  - GPS
- 지도



Big Dipper, by Magnus Manske, Public Domain



pixabay.com, CC0

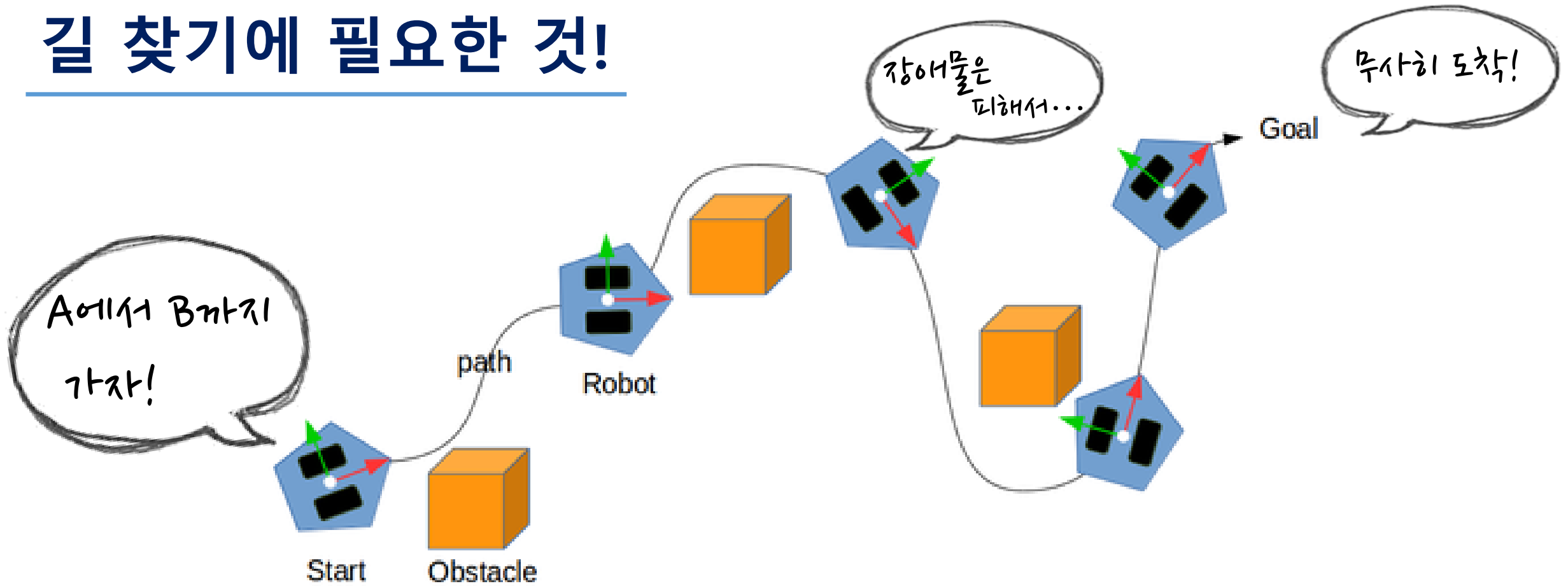
상상해 보세요!  
어둠 속 길 찾기

# 로봇의 길 찾기

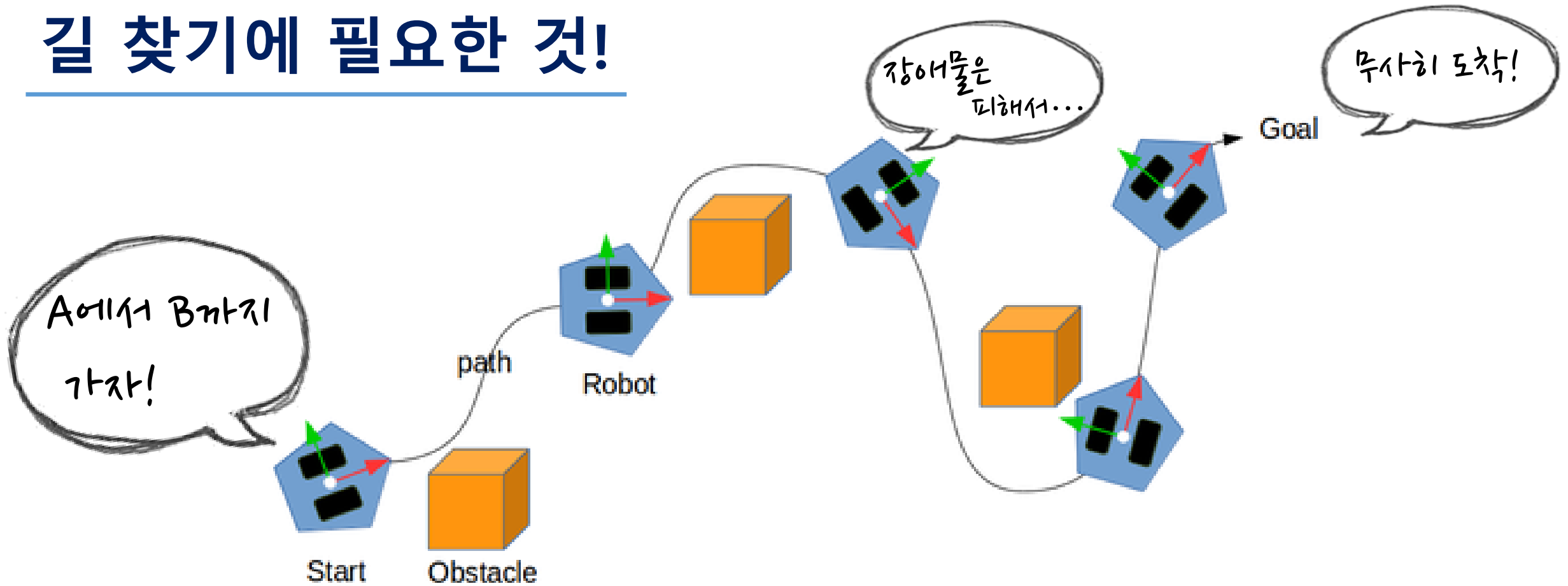
(이제부터는 열심히 풀어가 볼게요.)



# 길 찾기에 필요한 것!



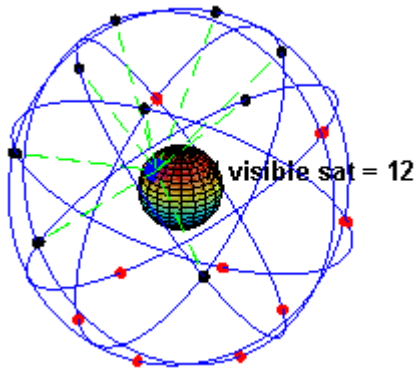
# 길 찾기에 필요한 것!



- 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

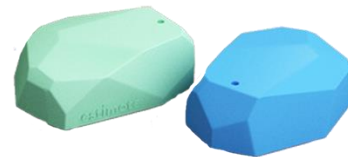
## ■ GPS (Global Positioning System)



- 오차
- 날씨
- 실외

## ■ Indoor Positioning Sensor

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



Estimote (Beacon)



StarGazer



Vicon MX

# ① 위치: 로봇의 위치 계측/추정하는 기능

## ■ 추측 방법(dead reckoning, 데드레커닝)

- 양 바퀴 축의 회전 값을 이용
- 이동 거리와 회전 값을 계산, 위치 측정
- 바닥 슬립, 기계적, 누적 오차 발생
- IMU 등의 관성 센서, 필터로 위치 보상
- 칼만필터 시리즈...



Turtlebot 1



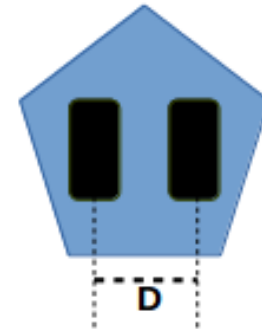
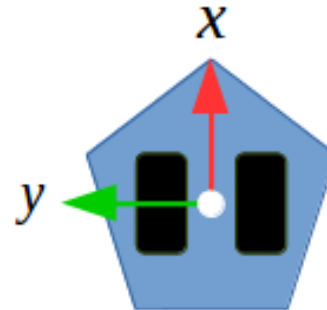
Turtlebot 2



Turtlebot 3

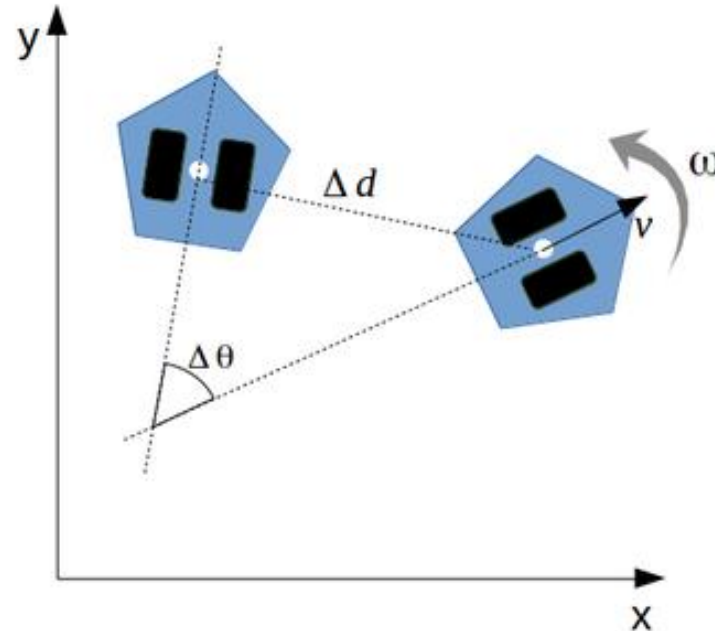
## ■ 필요한 정보

- 양 바퀴 축의 엔코더 값  $E$   
(모터 축인 경우 기어비로 재계산)
- 바퀴 간 거리  $D$
- 바퀴 반지름  $r$



# ① 위치: 로봇의 위치 계측/추정하는 기능

- 데드레커닝 계산
  - 선속도(linear velocity:  $v$ )
  - 각속도(angular velocity:  $w$ )
- Runge-Kutta 공식 이용
  - 이동한 위치의 근사 값  $x, y$
  - 회전 각도  $\theta$



$E_k, E_{lp}$ : 좌측 모터의 엔코더 값 (현재와 이전값)  
 $E_{rc}, E_{rp}$ : 우측 모터의 엔코더 값 (현재와 이전값)  
 $T_e$ : 경과시간  
 $v_l, v_r$ : 좌 / 우측 휠의 각속도  
 $r$ : 휠의 반지름  
 $V_l, V_r$ : 좌 / 우측 휠의 선속도  
 $v$ : 로봇의 선속도  
 $\omega$ : 로봇의 각속도

$$V_l = \frac{E_{lc} - E_{lp}}{T_e} \frac{\pi}{180} \quad (rad/s) \quad (1)$$

$$V_r = \frac{E_{rc} - E_{rp}}{T_e} \frac{\pi}{180} \quad (rad/s) \quad (2)$$

$$V_l = v_l r \quad (m/s) \quad (3)$$

$$V_r = v_r r \quad (m/s) \quad (4)$$

$$v = \frac{V_r + V_l}{2} \quad (m/s) \quad (5)$$

$$\omega = \frac{V_r - V_l}{D} \quad (rad/s) \quad (6)$$

$$x_{k+1} = x_k + T_e v_k \cos(\theta_k + \frac{T_e \omega_k}{2}) \quad (7)$$

$$y_{k+1} = y_k + T_e v_k \sin(\theta_k + \frac{T_e \omega_k}{2}) \quad (8)$$

$$\theta_{k+1} = \theta_k + \omega_k T_e \quad (9)$$

## 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
- Kinect, Xtion, Carmine(PrimeSense), Astra, Realsense





### ③ 지도: 길과 장애물 정보가 담긴 지도

---

- 로봇은 길을 찾아가기 위해 **지도**가 필요하다!



- 지도
  - 도로와 같은 기반 시설의 경우 디지털 지도 OK!
  - 병원, 카페, 회사, 가정집의 지도?
  - 탐사, 붕괴된 위험지역의 지도?

### ③ 지도: 길과 장애물 정보가 담긴 지도



- 로봇은 길을 찾아가기 위해 **지도**가 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

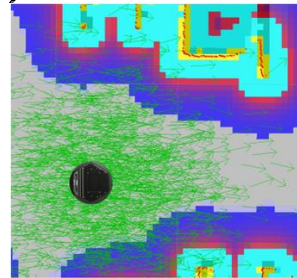
여긴 어디?

지도 만들자

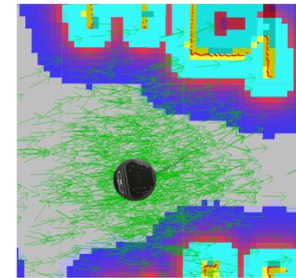


## ④ 경로: 목적지까지 최적 경로를 계산하고 주행하는 기능

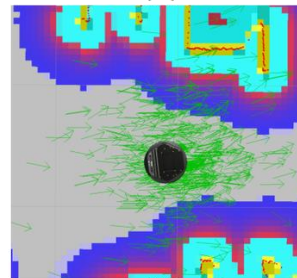
- 내비게이션(Navigation)
- 위치 추정 (Localization / Pose estimation)
- 경로 탐색/계획 (Path search and planning)
- A\* 알고리즘 (A Star)
- Dynamic Window Approach (DWA)
- 포텐셜 장(Potential Field)
- 파티클 필터 (Particle Filter)
- 그래프 (Graph)



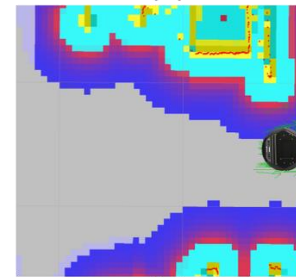
(t1)



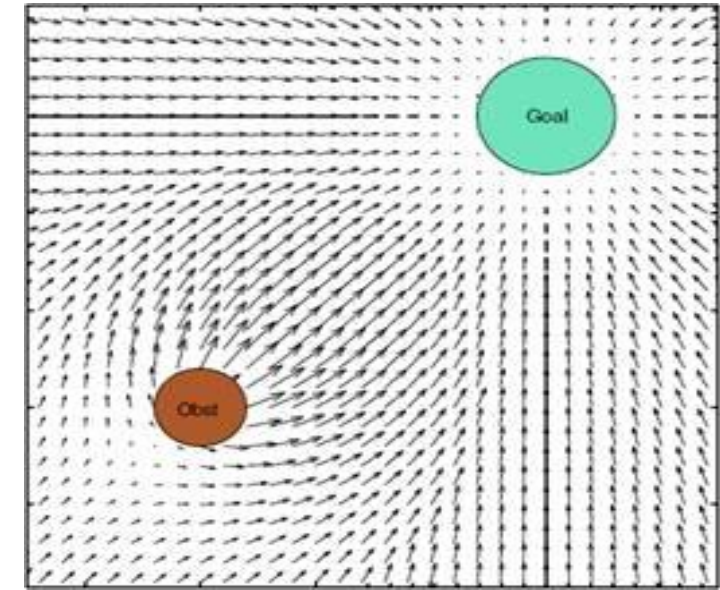
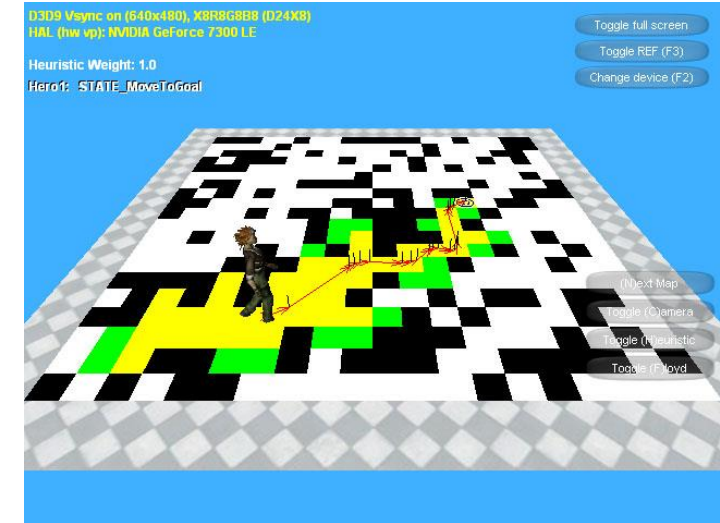
(t2)



(t3)



(t4)



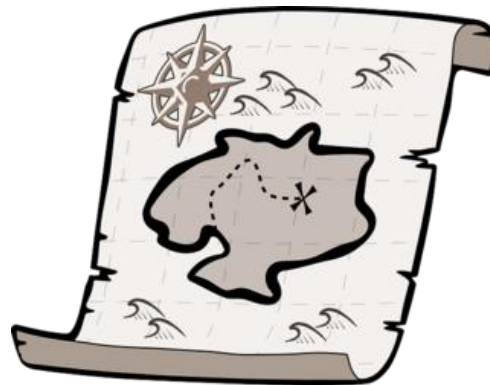
① 위치



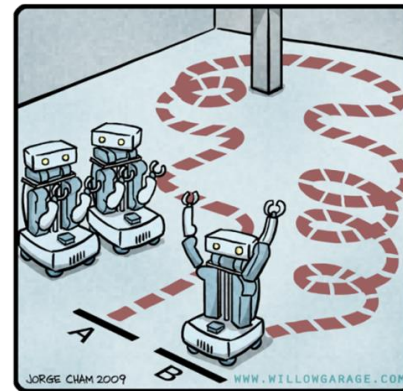
② 센싱



③ 지도



④ 경로



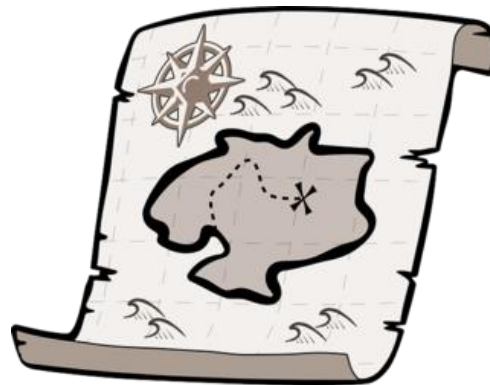
① 위치



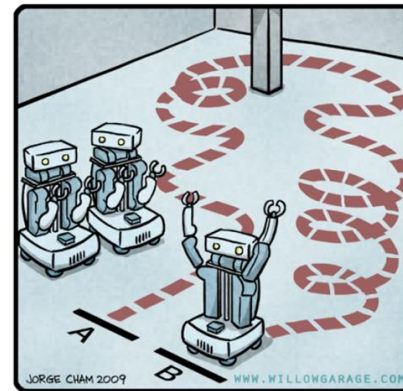
② 센싱



③ 지도



④ 경로



위치 + 센싱 → 지도

**SLAM**

위치 + 센싱 + 지도 → 경로

**Navigation**

# Index

---

I. SLAM과 내비게이션

II. 모바일 로봇의 위치추정과 맵핑 (SLAM)

III. 모바일 로봇의 내비게이션 (Navigation)

# Gmapping

---

- OpenSLAM에 공개된 SLAM 의 한 종류, ROS에서 패키지로 제공
- 저자: G. Grisetti, C. Stachniss, W. Burgard; CC BY-NC-SA 3.0
- 특징: Rao-Blackwellized 파티클 필터, 파티클 수 감소, 그리드 맵
- 하드웨어 제약 사항
  - **X, Y, Theta 속도 이동 명령**
    - 차동 구동형 모바일 로봇(differential drive mobile robot)
    - 전 방향 이동 로봇 (omni-wheel robot)
  - **주행기록계 (Odometry)**
  - **계측 센서: 2차 평면 계측 가능 센서( LRF, LiDAR, Kinect, Xtion 등)**
  - 직사각형 및 원형의 로봇

# Turtlebot

---

- ROS 공식 레퍼런스 모바일 로봇
- 전 세계 수 많은 연구소, 학교, DIY 에서 사용 중
- SLAM, Navigation, Gazebo, RViz 서포트!
  - <http://wiki.ros.org/Robots/TurtleBot>



**Turtlebot 1**



**Turtlebot 2**



**Turtlebot 3**



# 지도작성: Gmapping + Turtlebot3

---

- 소프트웨어 준비

```
$ sudo apt-get install ros-kinetic-gmapping  
$ sudo apt-get install ros-kinetic-navigation  
$ sudo apt-get install ros-kinetic-joy  
$ sudo apt-get install ros-kinetic-diagnostic*  
$ sudo apt-get install ros-kinetic-openni2-camera ros-kinetic-openni2-launch  
$ sudo apt-get install ros-kinetic-astra-camera ros-kinetic-astra-launch  
$ sudo apt-get install ros-kinetic-depthimage-to-laserscan
```

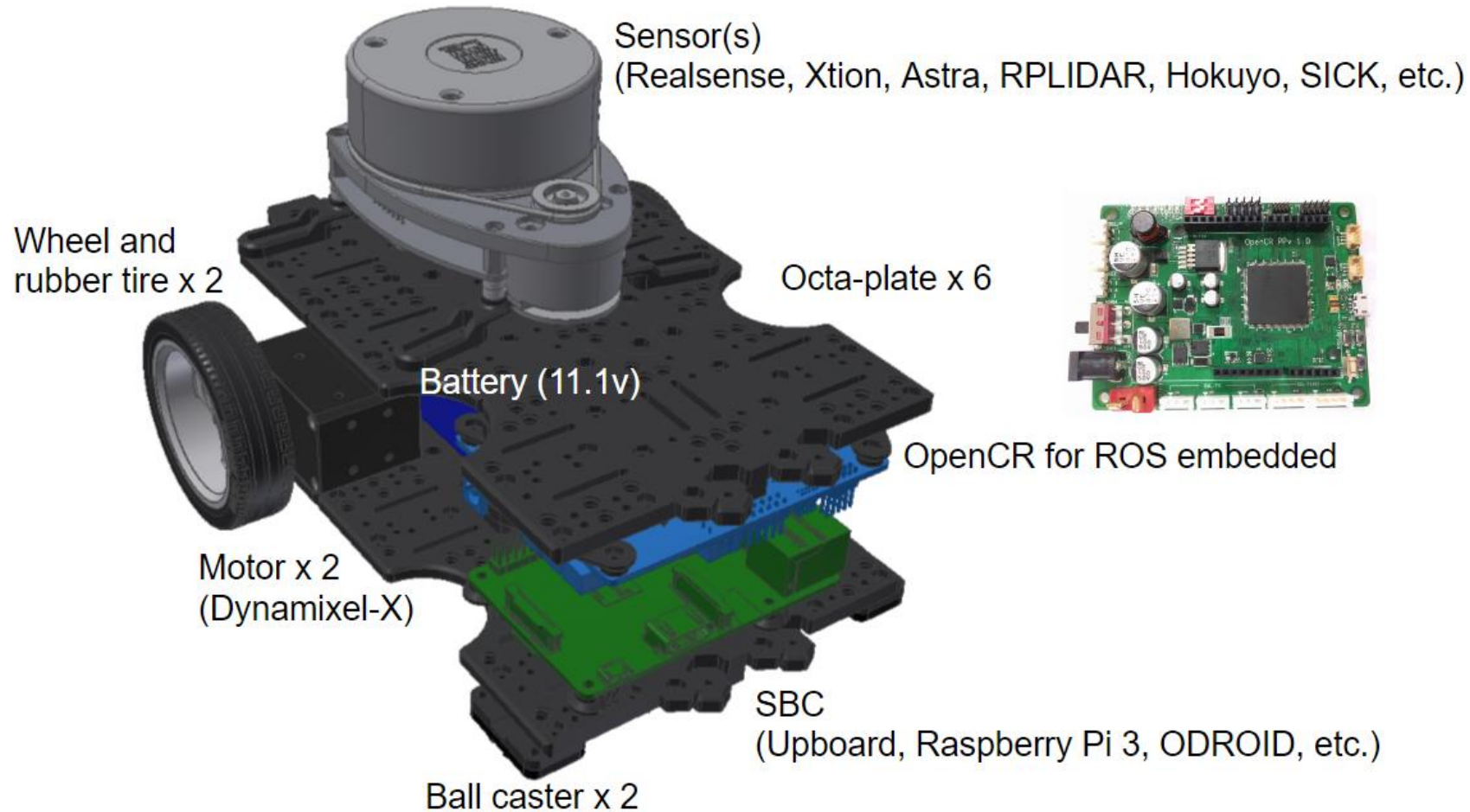
- Turtlebot3 패키지
  - ROSCon2016이후, 오픈 소스로 공개예정



# 지도작성: Gmapping + Turtlebot3

- 하드웨어 준비

\* Prototype 2.2





# 지도작성: Gmapping + Turtlebot3

---

- 마스터 실행 (데스크톱)

```
$ roscore
```

- 터틀봇 및 센서 구동 (랩톱)

```
$ roslaunch kobuki_node minimal.launch  
$ sudo chmod a+rw /dev/ttyACM0  
$ roslaunch kobuki_slam kobuki_slam.launch
```

- RViz, 터틀봇 원격 조종, 지도 작성 (데스크톱)

```
$ rosrn rviz rviz -d `rospack find kobuki_slam`/rviz/kobuki_slam.rviz  
$ roslaunch kobuki_keyop safe_keyop.launch  
$ rosrn map_server map_saver
```

# 지도작성: Gmapping + Turtlebot3

---

- 마스터 실행 (PC)

```
$ roscore
```

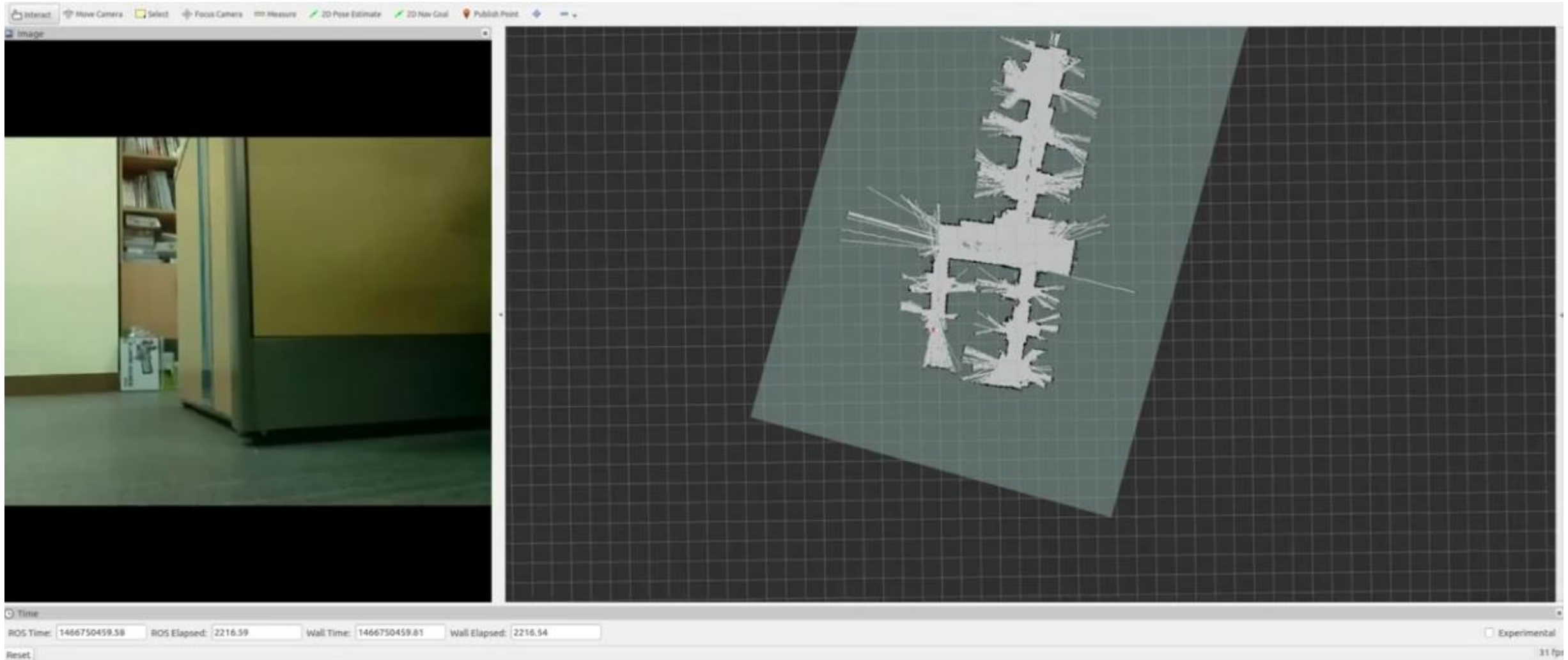
- 터틀봇 및 센서 구동 (MCU / Rplidar 의 경우)

```
$ sudo chmod a+rw /dev/ttyUSB0  
$ rosruntime turtlebot3_diff_driver turtlebot3_diff_driver_node  
$ sudo chmod a+rw /dev/ttyUSB1  
$ roslaunch rplidar_ros rplidar.launch
```

- RViz, 터틀봇 원격 조종, 지도 작성 (PC)

```
$ roslaunch turtlebot3_node turtlebot3_node.launch --screen  
$ roslaunch turtlebot3_slam turtlebot3_slam.launch  
$ rosruntime rviz rviz -d `rospack find turtlebot3_slam`/rviz/turtlebot3_slam.rviz  
$ roslaunch teleop_twist_joy teleop.launch  
$ rosruntime map_server map_saver
```

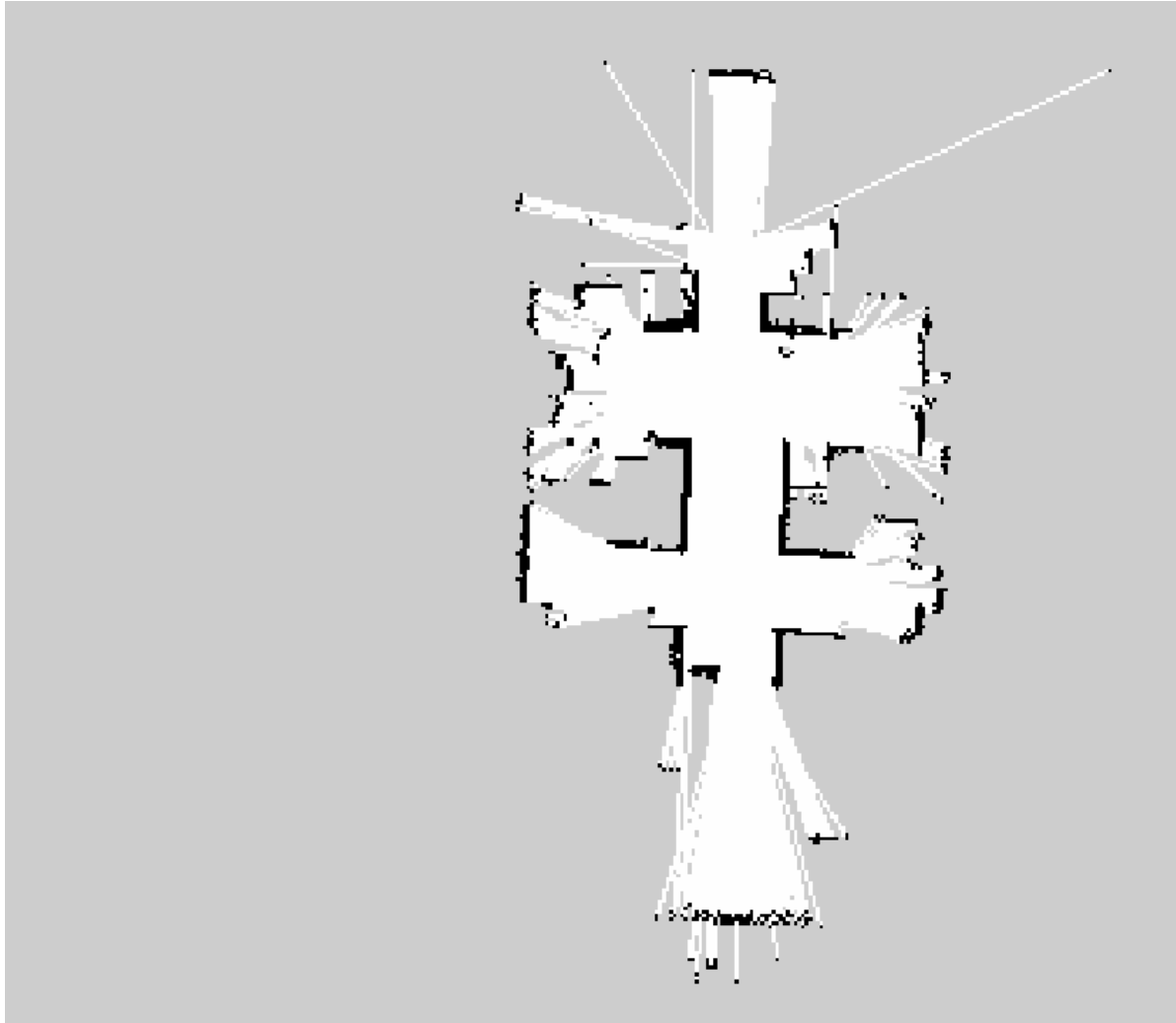
# 지도작성: Gmapping + Turtlebot3



<https://goo.gl/oLfX05>

# 지도작성: Gmapping + Turtlebot3

- 완성된 지도



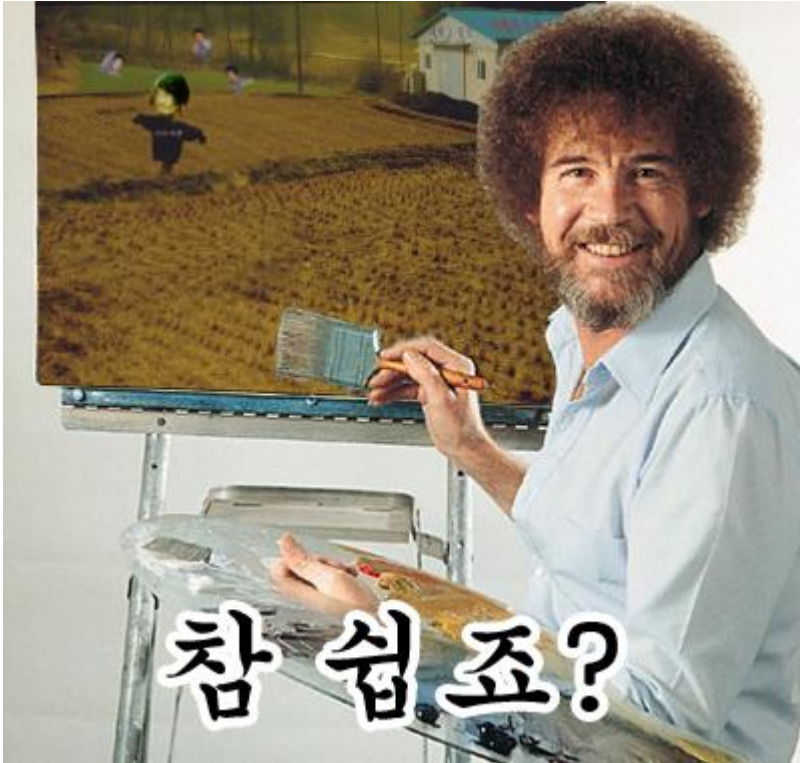
## 2차원 점유 격자 지도 (OGM, Occupancy Grid Map)

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

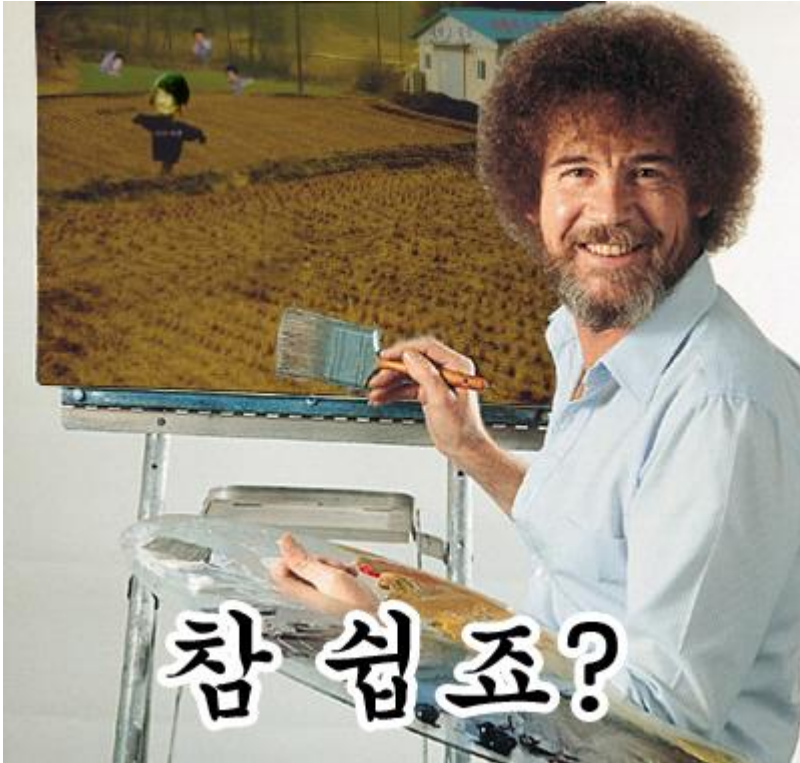


# 지도작성

---



# 지도작성



참 쉽죠?

SLAM, Navigation 은 기본 기능이고  
상위에 서비스 또는 모바일 로봇 자체를 하고 싶다고요?  
그렇다면 SLAM, Navigation 은 그대로 쓰시고  
좀 더 시간을 원하시는 부분에 투자하세요.  
세상에 없는 유니크한 당신만의 로봇을 기대해 봅니다.



# 지도작성

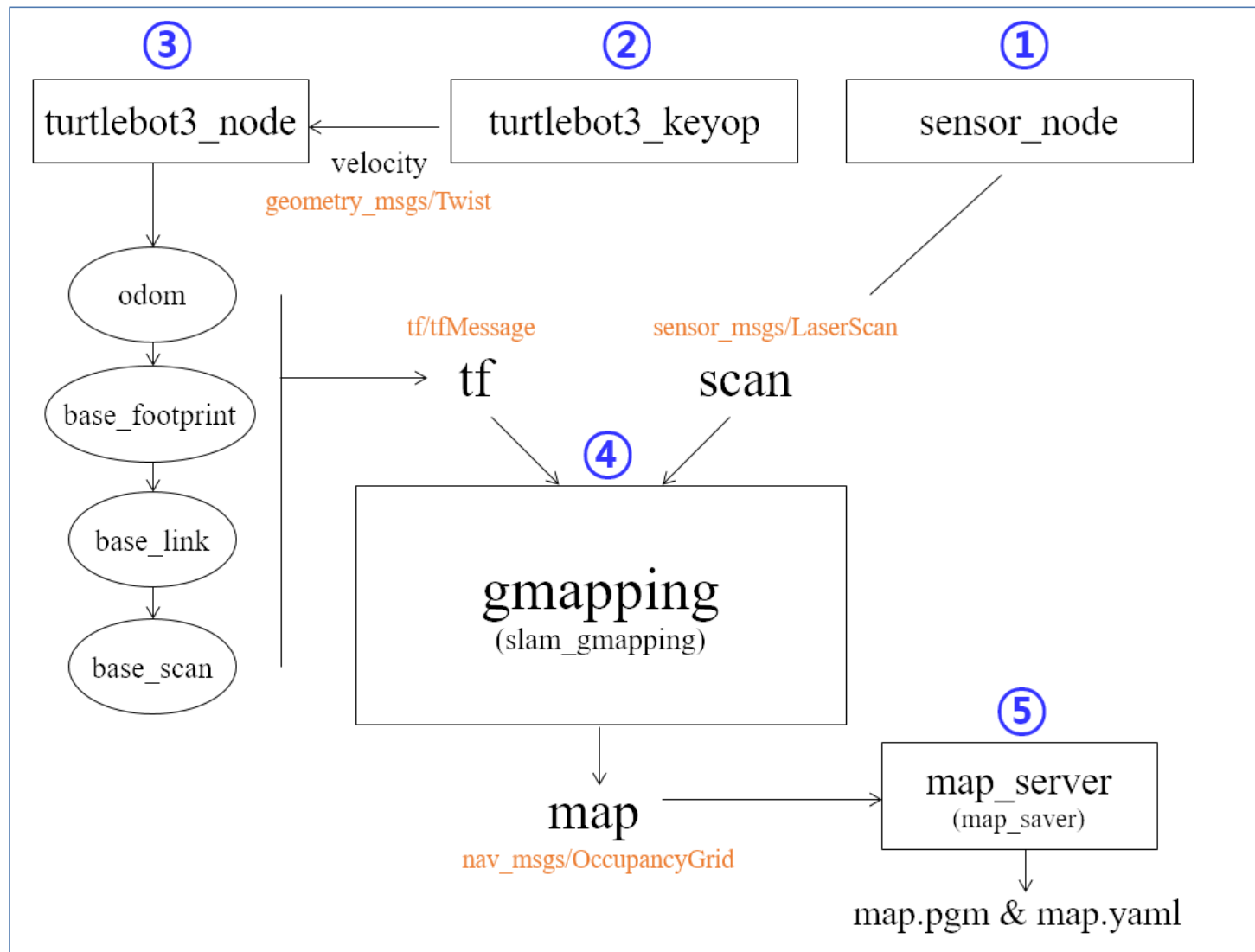


SLAM, Navigation 은 기본 기능이고  
상위에 서비스 또는 모바일 로봇 자체를 하고 싶다고요?  
그렇다면 SLAM, Navigation 은 그대로 쓰시고  
좀 더 시간을 원하시는 부분에 투자하세요.  
세상에 없는 유니크한 당신만의 로봇을 기대해 봅니다.

SLAM, Navigation 을 더 공부하고 싶다고요?  
모든 소프트웨어는 오픈 소스 입니다.  
마음껏 보고, 이해해 보고, 기능도 추가하며  
공부해 보세요. 이보다 더 좋은 교과서는 없습니다.

# SLAM 관련 노드들의 처리 과정

- ① sensor\_node
- ② turtlebot3\_keyop
- ③ turtlebot3\_node
- ④ slam\_gmapping
- ⑤ map\_server





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

## • 칼만 필터 (Kalman filter)

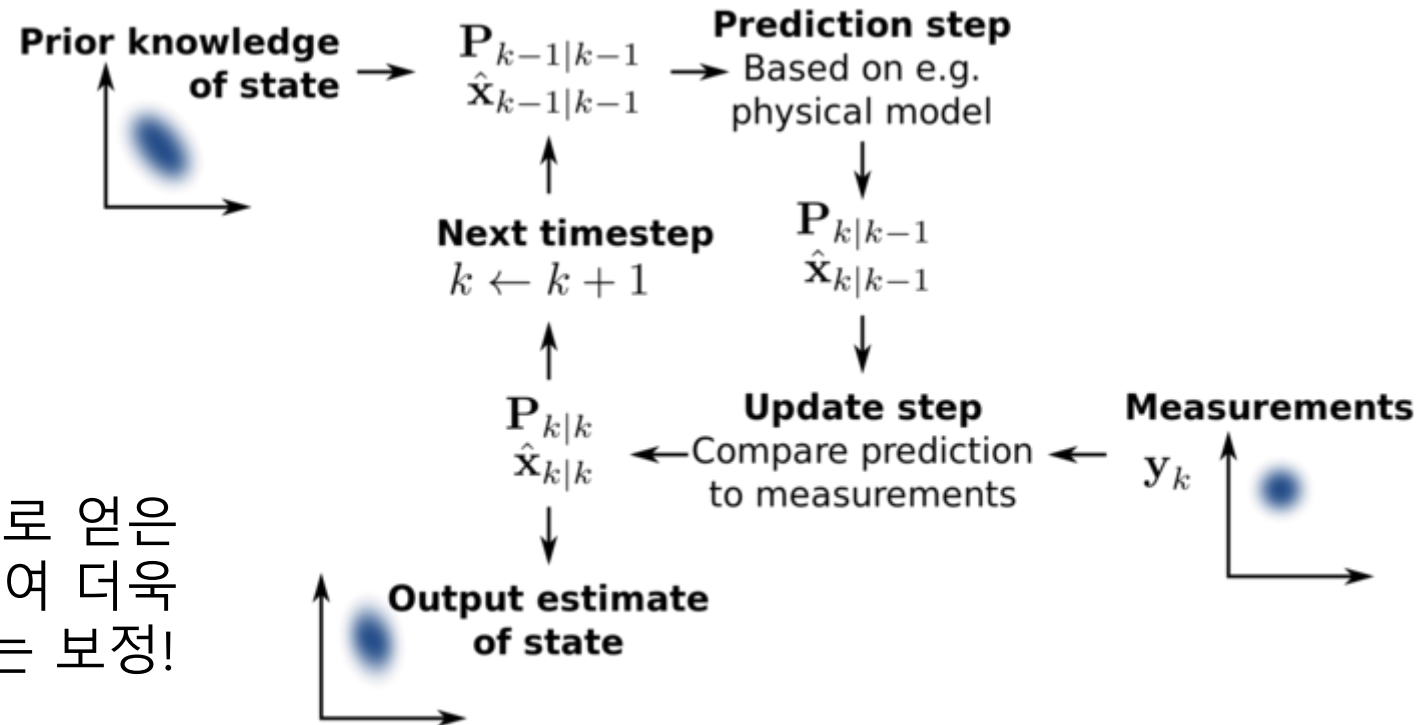
- 잡음이 포함되어 있는 선형 시스템에서 대상체의 상태를 추적하는 재귀 필터
- 베이지 확률 기반

### • 예측(Prediction)

- 모델을 상정하고 이 모델을 이용하여 이전 상태에서부터 현재 시점의 상태를 예측

### • 보정(update)

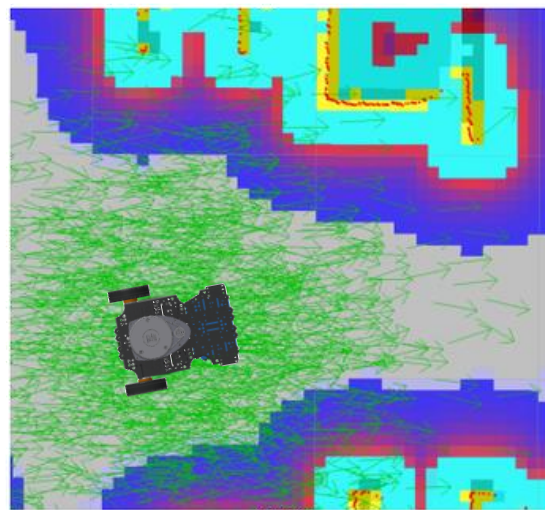
- 앞 단계의 예측 값과 외부 계측기로 얻은 실제 측정 값 간의 오차를 이용하여 더욱 정확한 상태의 상태 값을 추정하는 보정!



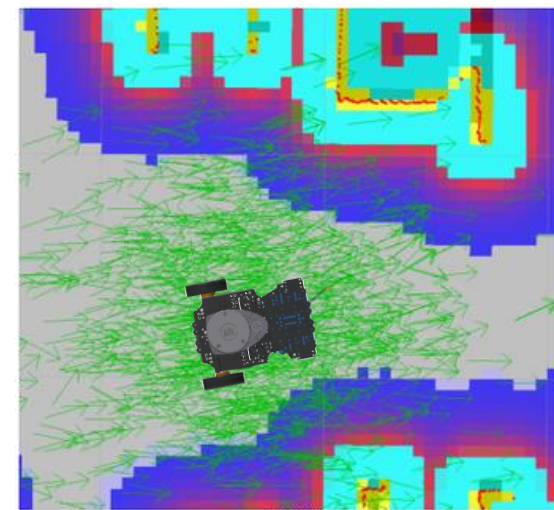
# 위치 추정(localization) | Kalman filter, Particle filter, Graph, Bundle adjustment

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

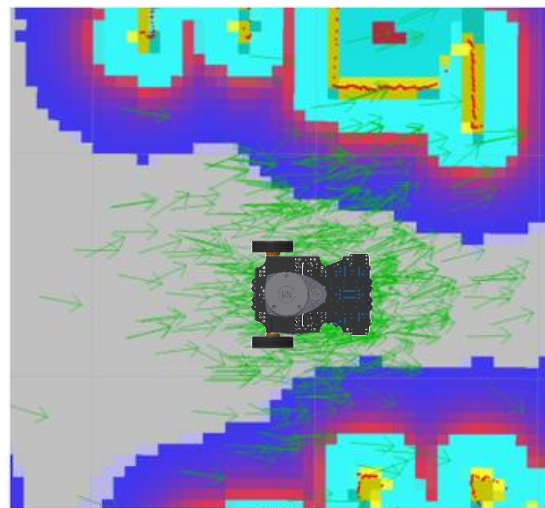
- 1) 초기화(initialization)
- 2) 예측(prediction)
- 3) 보정(update)
- 4) 위치 추정(pose estimation)
- 5) 재추출(Resampling)



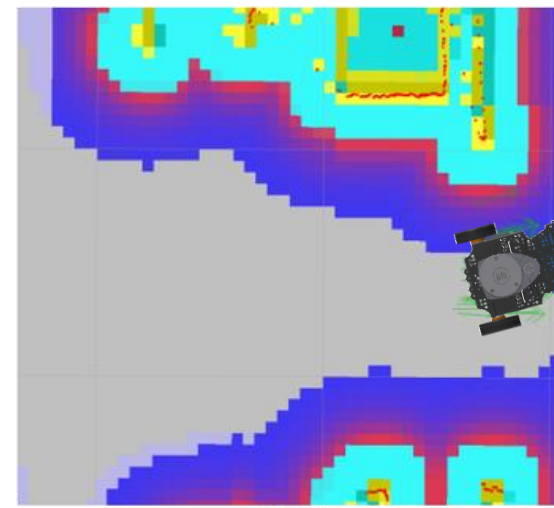
(t1)



(t2)



(t3)



(t4)

# Index

---

I. SLAM과 내비게이션

II. 모바일 로봇의 위치추정과 맵핑 (SLAM)

III. 모바일 로봇의 내비게이션 (Navigation)

# 내비게이션

---

- 마스터 실행 (PC)

```
$ roscore
```

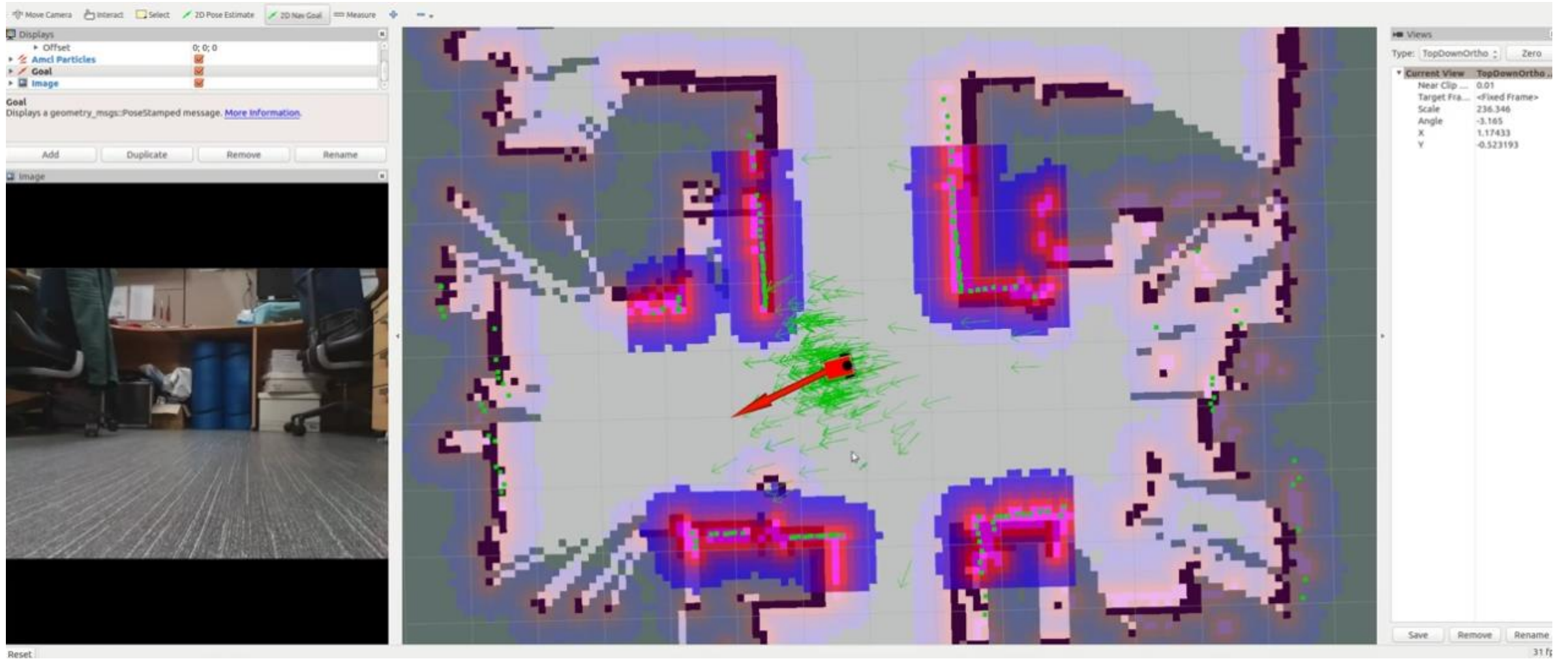
- 터틀봇 및 센서 구동 (MCU / Rplidar 의 경우)

```
$ sudo chmod a+rw /dev/ttyUSB0  
$ rosrn turtlebot3_diff_driver turtlebot3_diff_driver_node  
$ sudo chmod a+rw /dev/ttyUSB1  
$ roslaunch rplidar_ros rplidar.launch
```

- RViz, 터틀봇 원격 조종, 지도 작성 (PC)

```
$ roslaunch turtlebot3_node turtlebot3_node.launch --screen  
$ roslaunch turtlebot3_navigation turtlebot3_navigation.launch  
$ rosrn rviz rviz -d `rospack find turtlebot3_navigation`/rviz/turtlebot3_nav.rviz
```

# 내비게이션



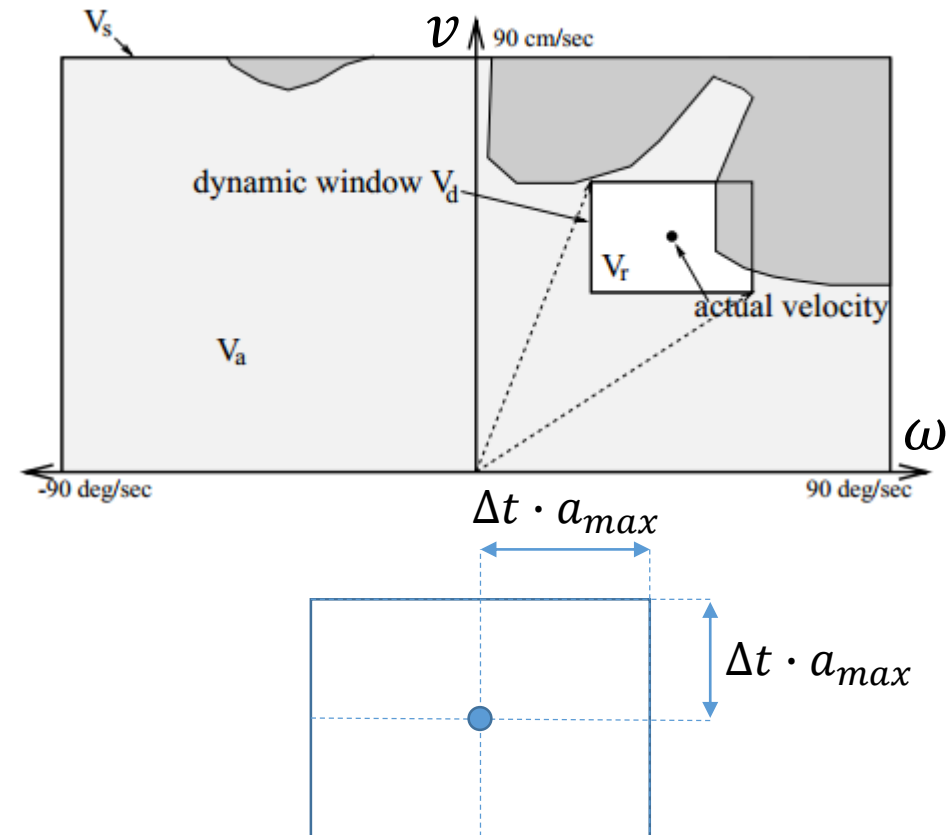
<https://goo.gl/TCI5Ge>

# 내비게이션

- **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),  $\omega$  (회전속도)
- $V_s$ : 가능 속도 영역
- $V_a$ : 허용 속도 영역
- $V_r$ : 다이내믹 윈도우 안의 속도 영역
- $G(v, \omega) = \sigma(\alpha \cdot \text{heading}(v, \omega) + \beta \cdot \text{dist}(v, \omega) + \gamma \cdot \text{velocity}(v, \omega))$

- 목적함수  $G$ 는 로봇의 방향, 속도, 충돌을 고려하여, 목적함수가 최대가 되는 속도  $v, \omega$  를 구하게 된다.

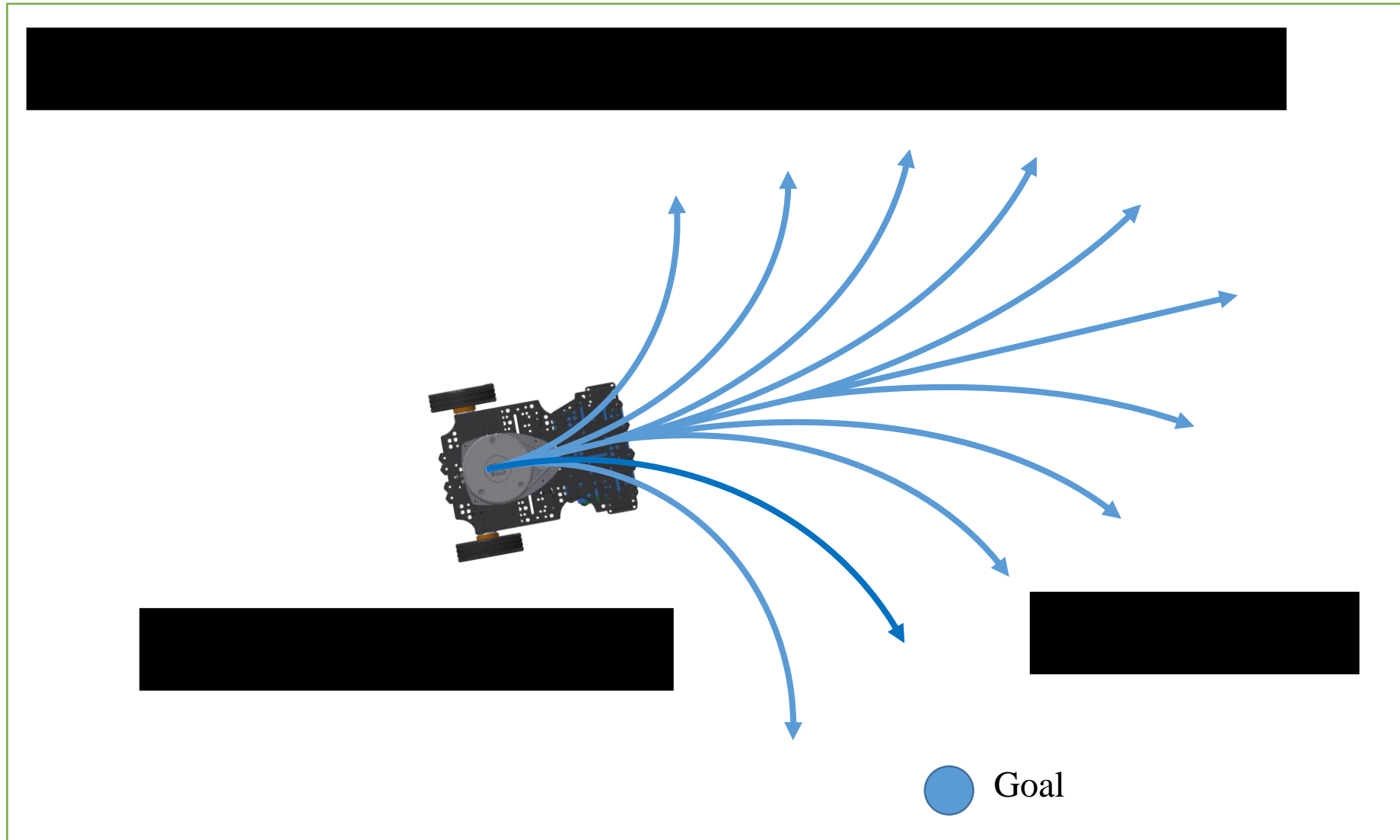


Dynamic Window

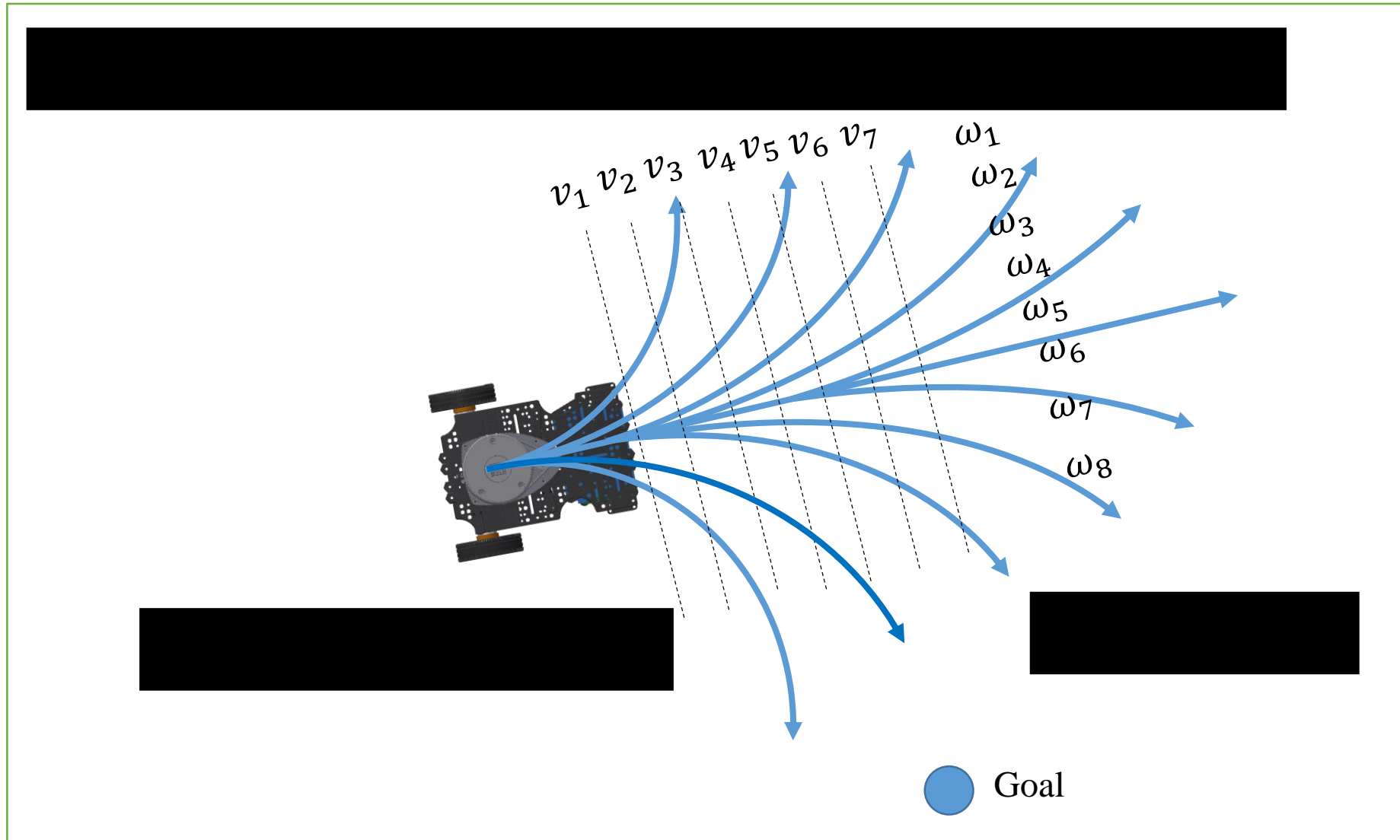


# Dynamic Window Approach

---

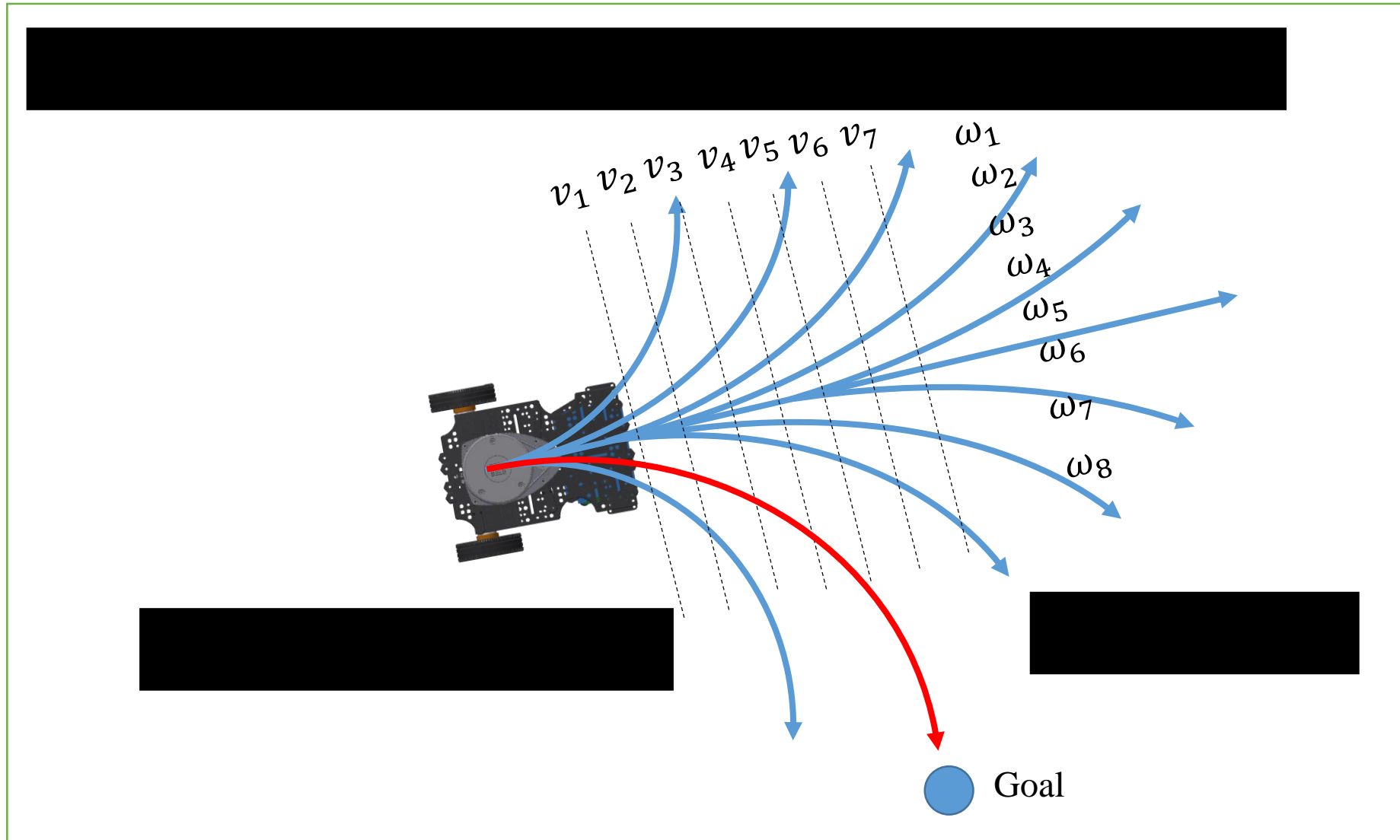


# Dynamic Window Approach



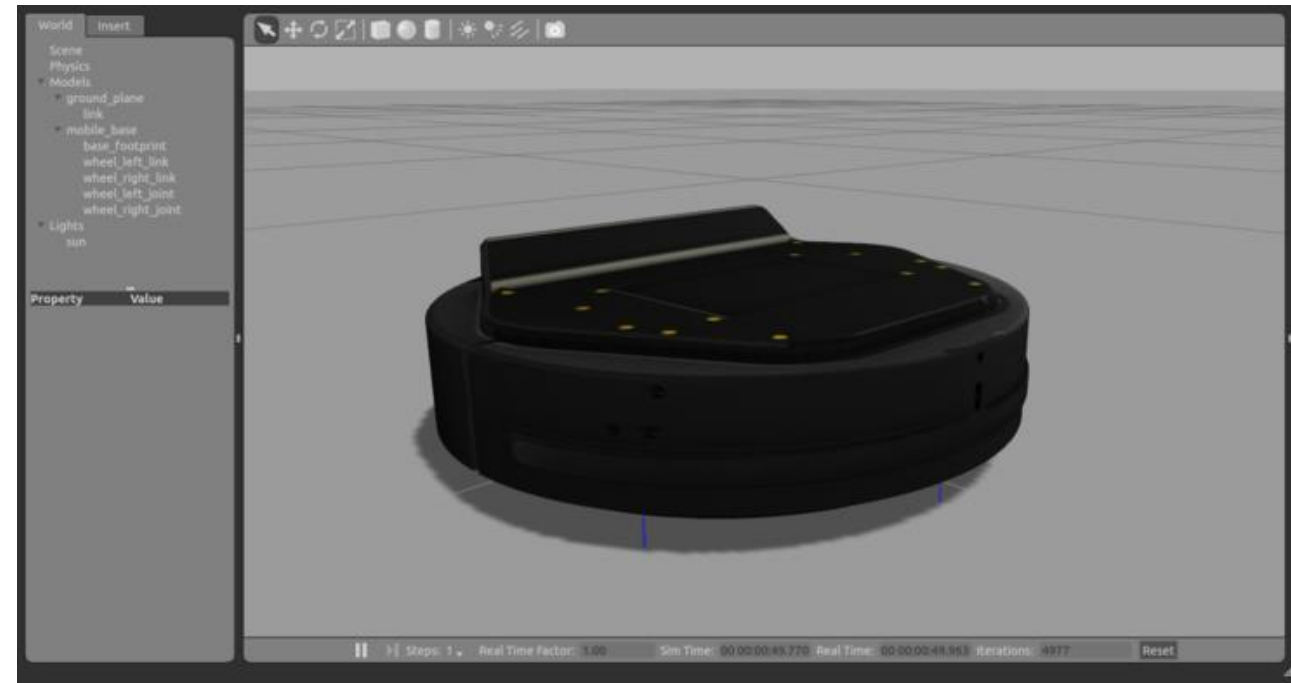
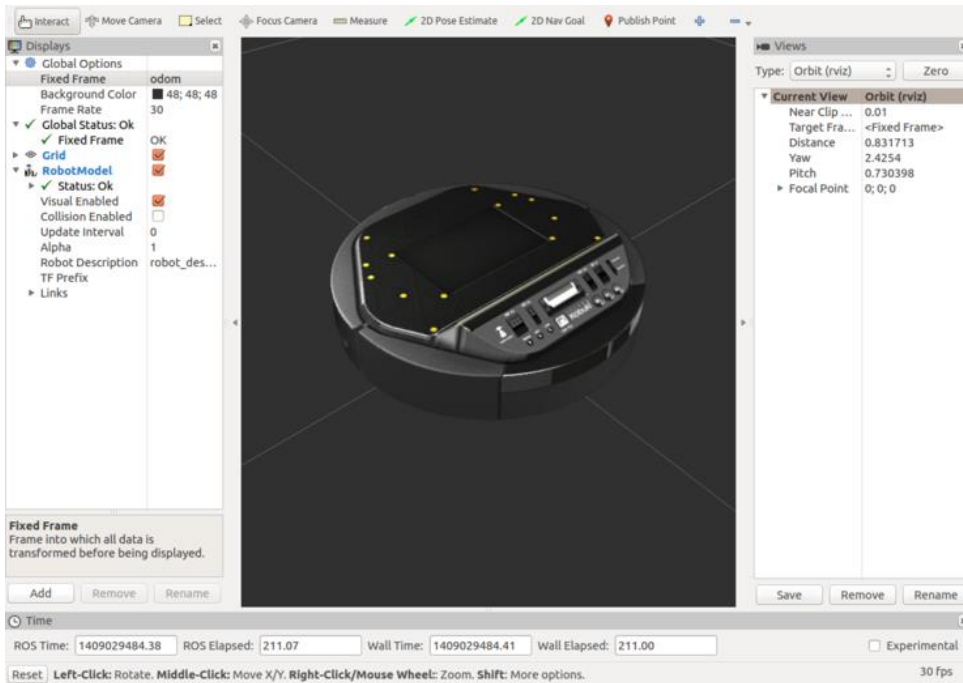


# Dynamic Window Approach



# 시뮬레이션

- 시뮬레이션을 위한 두 가지 방법
  - ROS의 3차원 시각화 도구인 RViz를 이용
  - 3차원 로봇 시뮬레이터 Gazebo를 이용



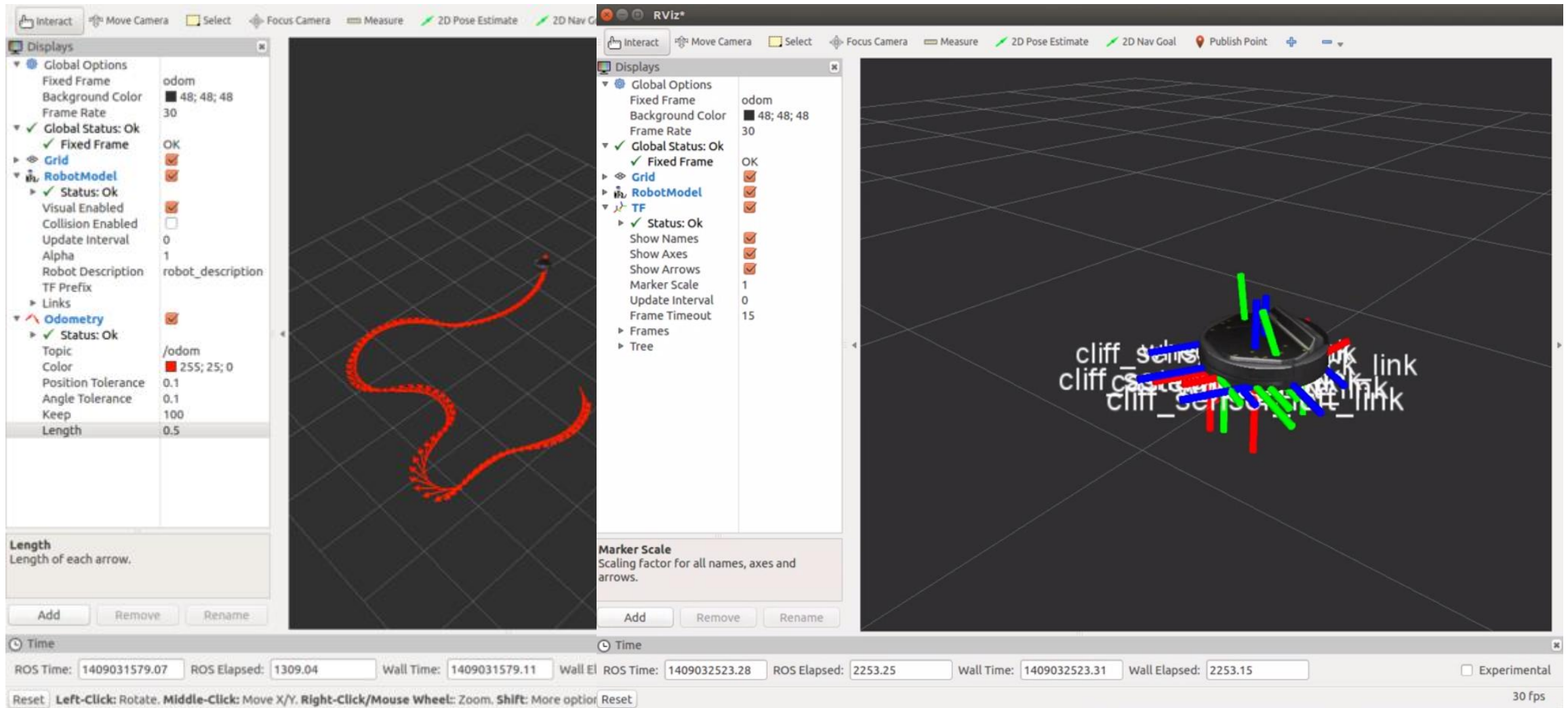
# 시뮬레이션 / RViz를 뷰어로 사용할 경우

```
$ sudo apt-get install ros-kinetic-kobuki-soft  
$ roslaunch kobuki_softnode full.launch --screen  
$ roslaunch kobuki_keyop keyop.launch
```

- 방향키 ↑: 전진(0.5씩, 단위=m/sec)
- 방향키 ↓: 후진(0.5씩, 단위=m/sec)
- 방향키 ←: 시계 반대방향으로 회전(0.33씩, 단위=rad/sec)
- 방향키 →: 시계방향으로 회전(0.33씩, 단위=rad/sec)
- 스페이스바: 리니어 속도 및 회전 속도 초기화
- d: 모터 비활성화(구동 불가능 상태)
- e: 모터 활성화(구동 가능 상태)
- q: 종료

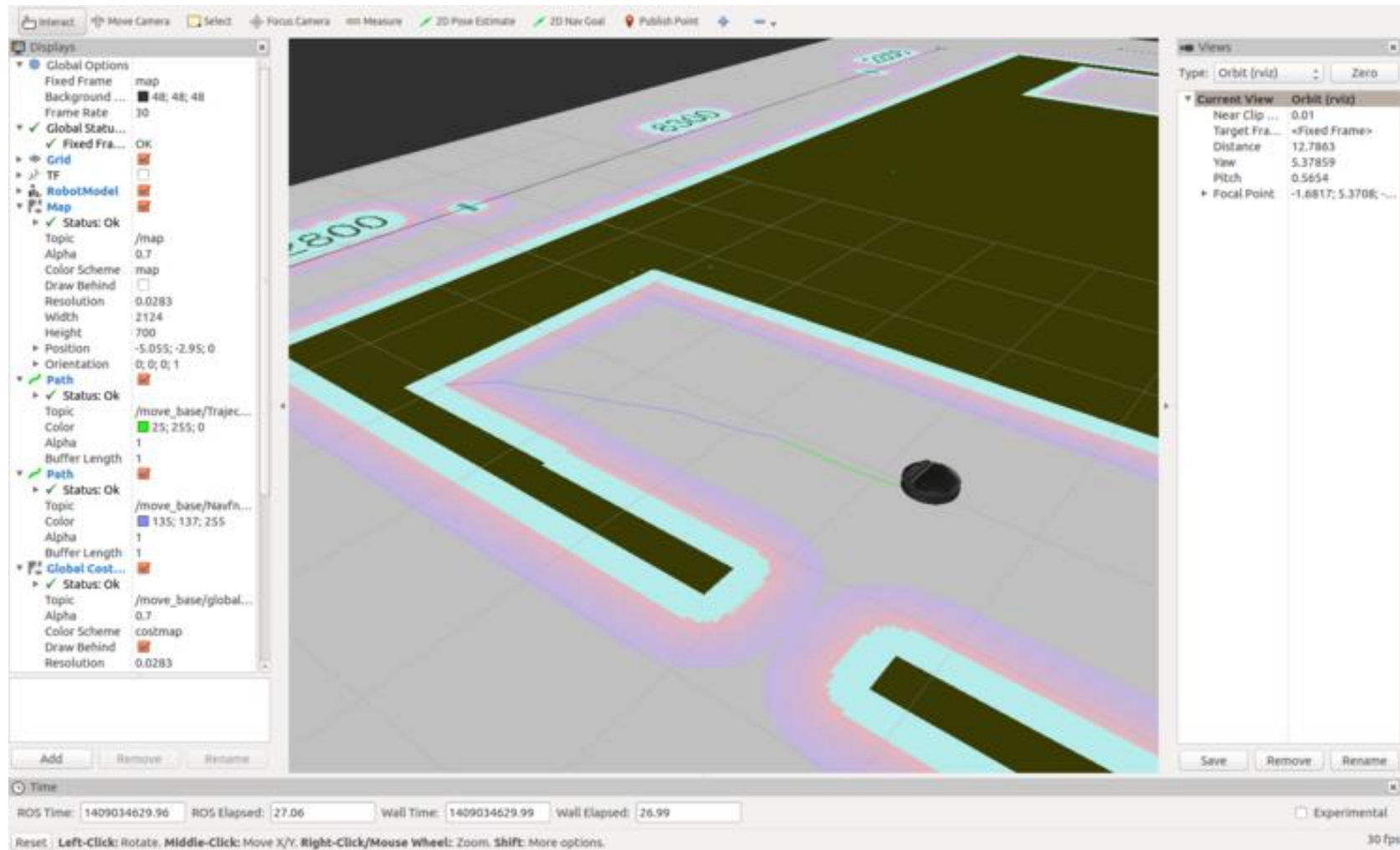
# 시뮬레이션 / RViz를 뷰어로 사용할 경우

- 로봇을 이동시켜 보면서 Odometry와 tf를 확인해 보자!



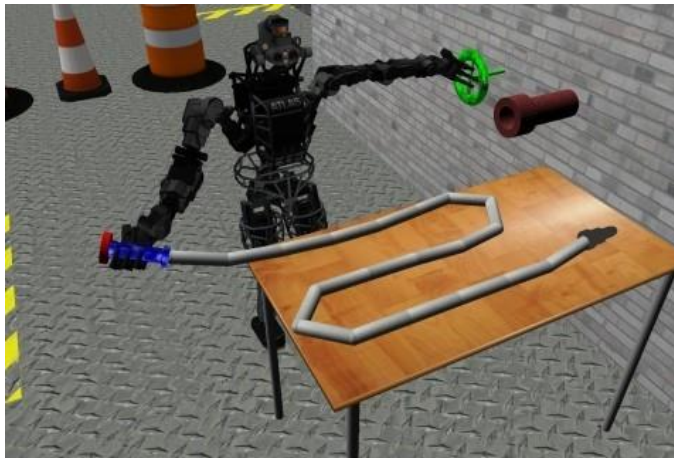
# 가상 내비게이션 in RViz









```
$ sudo apt-get install ros-kinetic-navigation ros-kinetic-yujin-maps  
$ roslaunch kobuki_softapps nav_demo.launch
```



# Gazebo

- Gazebo는 로봇 개발에 필요한 3차원 시뮬레이션을 위한 로봇, 센서, 환경 모델 등을 지원하고 **물리 엔진을 탑재**하여 실제와 근사한 결과를 얻을 수 있는 **3차원 시뮬레이터**이다.
- Gazebo는 최근에 나온 오픈 진영 시뮬레이터 중 가장 좋은 평가를 받고 있고, 미국 **DARPA Robotics Challenge**의 공식 시뮬레이터로 선정되어 개발에 더욱 박차를 가하고 있는 상황이다.
- ROS에서는 그 태생이 Player/Stage, Gazebo를 기본 시뮬레이터로 사용하고 있어서 **ROS와의 호완성도** 매우 좋다.



 <b>Dynamics Simulation</b> Access multiple high-performance physics engines including ODE, Bullet, Simbody, and DART.	 <b>Advanced 3D Graphics</b> Utilizing OGRE, Gazebo provides realistic rendering of environments including high-quality lighting, shadows, and textures.	 <b>Sensors and Noise</b> Generate sensor data, optionally with noise, from laser range finders, 2D/3D cameras, Kinect style sensors, contact sensors, force-torque, and more.	 <b>Plugins</b> Develop custom plugins for robot, sensor, and environmental control. Plugins provide direct access to Gazebo's API.
 <b>Robot Models</b> Many robots are provided including PR2, Pioneer2 DX, iRobot Create, and TurtleBot. Or build your own using SDF.	 <b>TCP/IP Transport</b> Run simulation on remote servers, and interface to Gazebo through socket-based message passing using Google Protobufs.	 <b>Cloud Simulation</b> Use CloudSim to run Gazebo on Amazon, Softlayer, or your own OpenStack instance.	 <b>Command Line Tools</b> Extensive command line tools facilitate simulation introspection and control.

# 가상 내비게이션 in Gazebo

---

- Gazebo 설치

```
$ sudo apt-get install gazebo7
```

- 관련패키지 설치 및 실행

```
$ sudo apt-get install ros-kinetic-gazebo-ros ros-kinetic-gazebo-plugins ros-kinetic-kobukidesktop
```

```
$ roslaunch kobuki_gazebo kobuki_empty_world.launch
```

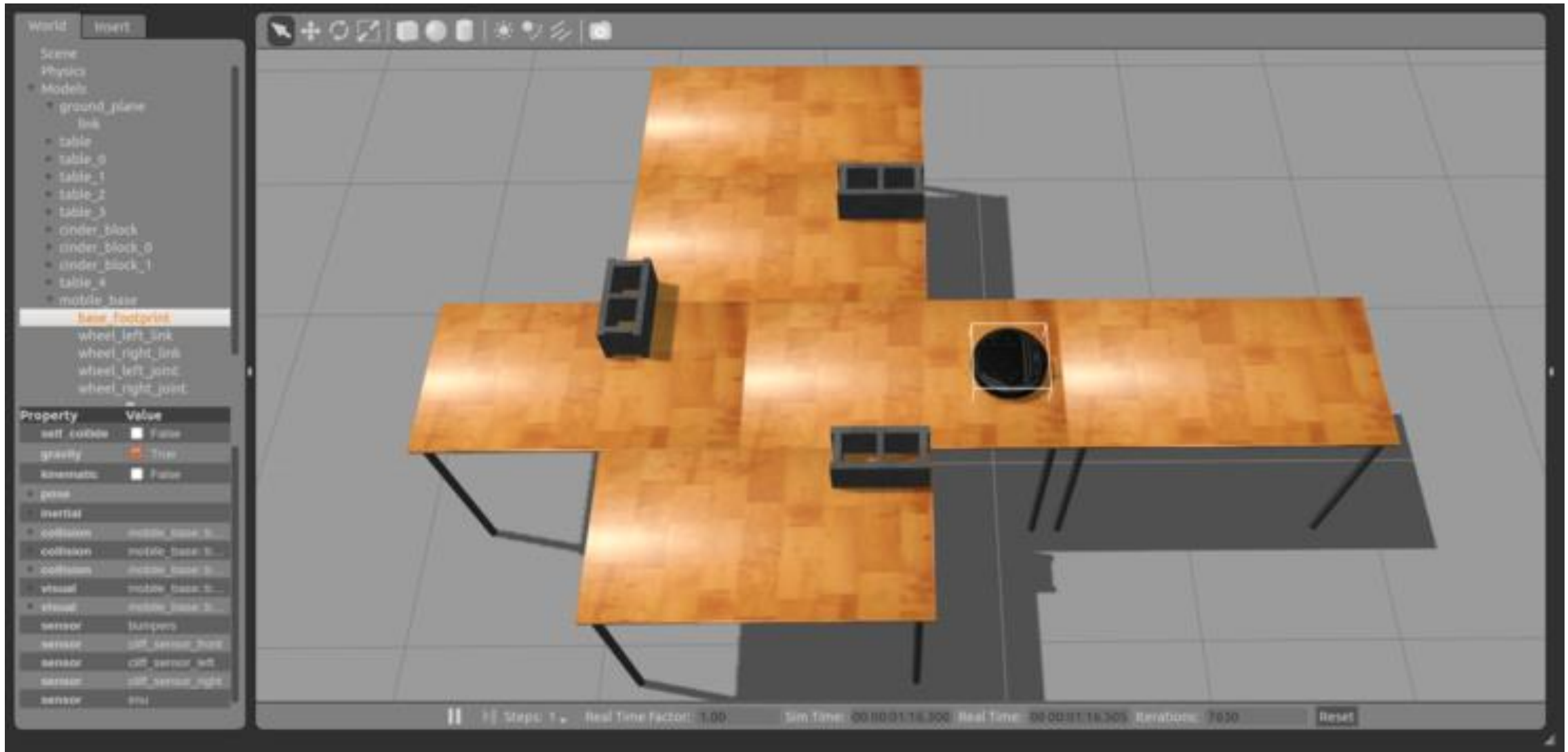
```
$ roslaunch kobuki_gazebo kobuki_playground.launch
```

- 시뮬레이션의 한가지 예 (벽 충돌시 방향 전환)

```
$ roslaunch kobuki_gazebo safe_random_walker_app.launch
```



# 가상 내비게이션 in Gazebo





# 마지막으로... 다시 한번!



참 쉽죠?

SLAM, Navigation 은 기본 기능이고  
상위에 서비스 또는 모바일 로봇 자체를 하고 싶다고요?  
그렇다면 SLAM, Navigation 은 그대로 쓰시고  
좀 더 시간을 원하시는 부분에 투자하세요.  
세상에 없는 유니크한 당신만의 로봇을 기대해 봅니다.

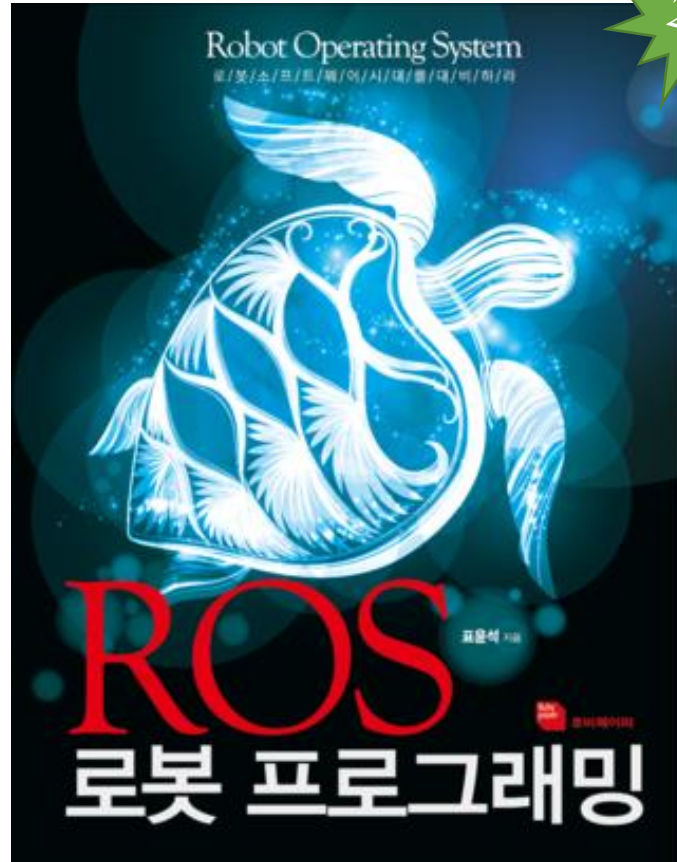
SLAM, Navigation 을 더 공부하고 싶다고요?  
모든 소프트웨어는 오픈 소스 입니다.  
마음껏 보고, 이해해 보고, 기능도 추가하며  
공부해 보세요. 이보다 더 좋은 교과서는 없습니다.

---

**질문  
대환영!**

---

여기서! 광고 하나 나가요~



국내 유일! 최초! ROS 책  
비 영어권 최고의 책  
인세 전액 기부

여기서! 광고 둘 나가요~



- 오로카
- [www.oroqa.org](http://www.oroqa.org)
- 오픈 로보틱스 지향
- 풀뿌리 로봇공학의 저변 활성화
- 공개 강좌, 세미나, 프로젝트 진행

- 로봇공학을 위한 열린 모임 (KOS-ROBOT)
- [www.facebook.com/groups/KoreanRobotics](https://www.facebook.com/groups/KoreanRobotics)
- 로봇공학 통합 커뮤니티 지향
- 일반인과 전문가가 어울러지는 한마당
- 로봇공학 소식 공유
- 연구자 간의 협력

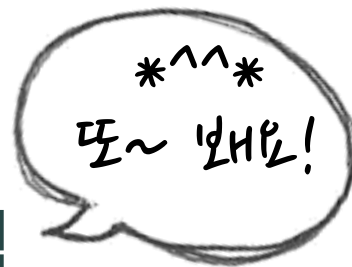
혼자 하기에 답답하시다고요?  
커뮤니티에서 함께 해요~

# 끝.

---

표윤석

Yoonseok Pyo  
pyo@robotis.com  
www.robotpilot.net



[www.facebook.com/yoonseok.pyo](https://www.facebook.com/yoonseok.pyo)

# Thanks for your attention!

표윤석

Yoonseok Pyo  
pyo@robotis.com  
www.robotpilot.net

[www.facebook.com/yoonseok.pyo](https://www.facebook.com/yoonseok.pyo)