1. Turtlebot3 Simulation (Ubuntu 20.04, ros noetic version)

참조: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

PC setup -> Gazebo, SLAM, Navigation simulation

이론적 배경

가. ROS

1) 노드(Node)

- ROS에서 실행되는 프로그램의 단위.
- rosrun 명령어로 실행 가능.

2) 토픽(Topic)과 메시지(Message)

- 토픽(Topic): 노드 간 데이터를 송수신하는 채널.
- 메시지(Message): 노드 간 주고받는 데이터 형식.

3) Publisher와 Subscriber

- Publisher: 특정 토픽으로 메시지를 보내는 노드.
- Subscriber: 특정 토픽에서 메시지를 받는 노드..

4) 서비스(Service)와 요청/응답(Request/Response)

- 클라이언트-서버 방식의 데이터 통신 구조.
- 요청(Request)을 보내면 응답(Response)을 받는 방식.

5) 액션(Action) 서버

- 서비스와 유사하지만, 장시간 작업이 필요한 경우 사용.

6) TF(Transform Frame)

- 로봇의 좌표계를 관리하는 프레임워크.

7) 파라미터 서버(Parameter Server)

- 노드 간 공유되는 설정값을 저장하는 서버.

나. 네비게이션

1) Map과 SLAM

로봇의 내비게이션을 위해서는 맵 정보가 필수적이며, ROS에서는 SLAM을 이용하여 자동으로 맵을 생성할 수 있음.

- ROS의 map_server 노드는 미리 저장된 맵을 불러오고 이를 ROS 서비스로 제공.

- 맵 파일은 .pgm 이미지 파일과 .yaml 설정 파일을 함께 사용하여 구성됨.

2) 로봇의 Pose

- ROS에서 로봇의 위치는 **(x, y, z) 좌표와 방향(Quaternion: x, y, z, w)**으로 표현.

- odom → base_link 변환을 통해 이동하는 동안 로봇의 위치를 계속 갱신.

3) 거리 측정 센서

- 로봇은 주행 중 장애물을 감지하고 경로를 수정하기 위해 LiDAR 또는 LDS를 사용.

- TurtleBot3에서는 sensor_msgs/LaserScan 메시지를 이용하여 /scan 토픽으로 데이터를 전달.

4) 경로 탐색 및 주행

- 로봇은 목적지까지 최적의 경로를 탐색하고 이동하기 위해 다양한 경로 탐색 알고리즘을 사용.

- Dijkstra 알고리즘과 Particle Filter를 조합하여 장애물을 회피하면서 최적 경로를 설정.

5) Odometry

- 로봇의 바퀴 회전 정보를 이용하여 주행 거리와 이동 방향을 계산하는 방식.

- odom → base_link 변환을 통해 로봇의 상대적 이동을 파악하며, 센서 오차를 보정하기 위해 IMU 센서를 추가적으로 활용.

6) Localization

- 로봇은 AMCL 노드를 사용하여 현재 위치를 추정.

- AMCL은 Particle Filter 기반으로 확률적 로봇 위치를 결정.

- /amcl_pose 토픽을 통해 현재 위치 데이터를 제공.

7) ROS 내비게이션 스택(Navigation Stack)

- ROS의 내비게이션 스택은 로봇이 목적지까지 안전하게 이동할 수 있도록 여러 노드로 구성.

<주요 노드 및 기능>

AMCL 노드: 로봇의 현재 위치를 지도 기반으로 추정.

Move Base 노드: 목적지까지의 최적 경로를 계산하고 속도 명령을 생성.

맵 서버(Map Server): 로봇이 사용할 환경 지도를 제공.

Global Planner & Local Planner:

Global Planner: 장애물을 고려하여 전체적인 최적 경로를 생성 (Dijkstra 알고리즘 활용).

Local Planner: 실시간 장애물 회피 및 속도 명령을 생성.

Recovery Behaviors: 경로 상에서 로봇이 장애물로 인해 멈출 경우 문제 해결을 시도.

2. Turtlebot3 조립 && rasberrypi4 ubuntu setting

참조: https://ubuntu.com/tutorials/how-to-install-ubuntu-on-your-raspberry-pi#1-overview

3. Download ROS Docker containers && Docker Compose

참조: https://github.com/zekeriyyaa/ROS-Communication-Via-Docker-Containers

이론적 배경

가. Docker 명령어

- $ Docker pull <이미지_이름>: 도커 허브에서 이미지를 가져오기
- $ Docker run –name some-nginx –d –p 8080:80 nginx

  : --name 옵션은 컨테이너의 이름 설정, -d 옵션은 이미지를 백그라운드에서 실행, -p 옵션은 호스트 머신의 8080 포트를 컨테이너의 80포트로 매핑하겠다는 설정

  ⇨ 컨테이너 실행 시 웹에서 http://localhost:8080/에 연결하여 'Welcome to nginx'가 뜨는 것을 확인할 수 있다.
- $ docker container ps : 현재 실행 중인 컨테이너를 확인 컨테이너의 이미지 정보 및 포트 정보를 알 수 있다.

  ⇨ $ docker container ps –a : 모든 컨테이너의 정보를 알 수 있다.
- $ docker logs <컨테이너 ID>: Docker 컨테이너의 로그 출력 확인
- $ docker container rm <컨테이너 ID>: 컨테이너 삭제
- $ docker image rm <이미지 이름:버전> : 이미지 삭제

나. Dockerfile

- Dockerfile은 도커이미지를 만들기위해 필요한 구성 파일이다. 이미지를 조립하기 위한 명령어들을 포함하고 있다.

  FROM some_base_image

  ⇨ 이미지의 기본 구성요소를 위한 기본 이미지가 필요하다.

  COPY executable_files

  ⇨ 호스트 머신의 실행 파일을 내부 디렉토리 경로로 복사한다.

  ENV app_env

  ⇨ 환경변수 설정 등을 통해 어플리케이션의 환경을 설정한다.

  RUN my_app

  ⇨ 명령어를 통해 어플리케이션을 실행한다.

- $ docker build –t myapp:1.0 **.** : 현재 디렉토리에 있는 Dockerfile을 통해 myapp:1.0 태그의 이미지를 생성

- $ docker –p 9000:8000 myapp:1.0 : myapp:1.0 이미지를 통해서 컨테이너를 만들고 호스트 머신의 localhost의 9000포트와 컨테이너의 8000포트를 매핑

다. Docker Networking

- 도커 네트워킹은 서로 다른 컨테이너가 통신하기 위한 방식이다.

  1) Bridge Network : 도커의 기본 네트워크이다. 같은 호스트 내의 컨테이너 간에는 통신이 가능하지만 서로 다른 호스트나 외부 네트워크는 통신이 불가능하다. 컨테이너를 실행하면 기본으로 bridge network가 선택되고 고유한 ip가 할당된다. 컨텐이너는 ip를 통해 같은 네트워크에 있는 다른 컨테이너와 통신할 수 있다.

     $ docker network ls : 현재 존재하는 도커 네트워크를 확인할 수 있다.

     $ docker network inspect <네트워크 이름> : 특정 네트워크에 연결된 컨테이너를 확인할 수 있다.

     $ docker exec –it <컨테이너 이름> bash : 실행 중인 컨테이너에 bash 셸 명령어로 터미널에 접근할 수 있다.

     $ apt update && apt install iputils-ping $ ping <다른 컨테이너의 IP>

     : ping 패키지를 이용해 같은 브릿지의 다른 컨테이너와 ip를 통해 통신한다.

  2) User Defined Bridge Network : 기본 제공되는 브릿지 네트워크와 구분되는 브릿

지 네트워크를 사용하고자 할 때 사용한다.

$ docker network create –driver bridge <브릿지 네트워크 이름> : 사용자 정의 브릿지 네트워크를 생성

3) Overlay Network : 서로 다른 호스트 머신에서 실행되는 컨테이너 간 통신을 가능하게 하는 드라이버이다. 기본적으로 Docker의 Bridge Network는 단일 호스트 내에서만 컨테이너 간 통신을 지원하지만 Overlay Network는 여러 Docker 호스트에 걸쳐 있는 컨테이너들이 서로 직접 통신 가능하다. 또한 Docker Swarm에서 서비스 간의 네트워크 통신을 위해 사용된다.

** Overlay Network 생성 및 사용 방법

1) Swarm 모드 활성화

Overlay Network는 Docker Swarm 모드에서만 동작한다. 먼저 Swarm 모드를 활성화해야한다.

$ docker swarm init

2) Overlay 네트워크 생성

$ docker network create --driver overlay my_overlay_network

- --driver overlay → Overlay Network 사용.
- my_overlay_network → 생성할 네트워크 이름.

3) 네트워크 확인

$ docker network ls

- 출력 예시 :

NETWORK ID NAME DRIVER SCOPE

df2e11fdfbe5 my_overlay_network overlay swarm

4) Overlay 네트워크에서 컨테이너 실행

$ docker service create --name web --network my_overlay_network -p 8080:80 nginx

- --network my_overlay_network → 해당 네트워크에서 실행.
- 여러 호스트에서 동일한 네트워크를 사용할 수 있음.

5) Overlay 네트워크에 컨테이너 추가

기존 컨테이너를 Overlay 네트워크에 연결하려면:

$ docker network connect my_overlay_network my_container

6) Overlay 네트워크 삭제

$ docker network rm my_overlay_network

- 주의: 사용 중인 네트워크는 삭제할 수 없음. 먼저 관련 컨테이너를 삭제해야 함.

라. Docker Compose

** Docker 공식 사이트에서 compose plugin 설치

docker docs -> Manuals -> Docker Compose/install && DockerCompose/QuickStart

: 도커 컴포즈는 단일 서버에서 여러 개의 컨테이너를 하나의 서비스로 정의하여 컨테이너를 묶음으로 관리할 수 있는 작업환경을 제공하는 도구이다. 여러 개의 컨테이너가 하나의 어플리케이션으로 동작할 때, 매번 run 명령어에 옵션을 설정해 CLI로 컨테이너를 생성하기 보다는 여러 개의 컨테이너를 하나의 서비스로 정의해 컨테이너 묶음으로 관리할 수 있다.

- docker-compose.yml

1) 'docker-compose.yml' 이름으로 파일 생성

2) 파일 내용 편집

- version : 파일 버전
- services : 만들고 싶은 컨테이너들을 들어오게 하는 지시어
- db : 컨테이너 이름
- -./db_data:/var/lib/mysql : 현재 디렉토리에 db_data와 /var/lib/mysql을 연결
- Environment: 환경변수 셋팅
- depends_on : db 컨테이너 먼저 생성 후 app 컨테이너 생성
- ports : 8080포트와 80포트 연결

3) $ docker compose up 실행

4) 실행 종료 시 $ docker compose down

- 예제: Docker compose를 이용해서 master, talker, listener 컨테이너를 만들고 컨테이너 간 ROS 네트워크 설정 및 토픽 교환을 한번에 실행

in pc

$ mkdir –p ~/docker_ws -> 새로운 디렉토리 생성

$ touch compose.yaml -> 도커 컴포즈를 위한 파일 compose.yaml 파일 생성

$ code . -> visual studio code 실행. code에서 docker extension 설치

in compose.yaml

services:

master: -> master 이름의 컨테이너

image: ros:noetic -> base 이미지를 ros:noetic 버전의 이미지로 설정

restart: always

environment:

- ROS_HOSTNAME=master -> ROS 환경변수 설정

- ROS_MASTER_URI-http://master:11311 -> ROS Master 환경변수 설정

command: roscore -> ROS Master 생성을 위해 roscore 명령어 실행

talker: -> talker 이름의 컨테이너

depends_on:

- master -> master 컨테이너 생성 후 talker 컨테이너 생성

image: ros:noetic

restart: always

environment:

- ROS_HOSTNAME=talker

- ROS_MASTER_URI-http://master:11311

command: rostopic pub /test std_msgs/String “hello world” –r 10

-> 토픽 publish를 위해 명령어 실행

listener: -> listener 이름의 컨테이너

depends_on:

- master -> master 컨테이너 생성 후 talker 컨테이너 생성

image: ros:noetic

restart: always

environment:

- ROS_HOSTNAME=listener

- ROS_MASTER_URI-http://master:11311

command: rostopic echo /test -> 토픽 subscribe를 위해 명령어 실행

in pc terminal 1

$ docker compose up

in pc terminal 2

$ docker compose down

4. TurtleBot3 Docker Image Creation

Base Docker Image: arm64v8/ros:noetic

USB Ports(check please using command in host PC $ sudo dmesg | grep tty):

- LDS-02: /dev/ttyUSB0

- OpenCR: /dev/ttyACM0

1) Container Run

: 도커 컨테이너 실행 (master 컨테이너)

$ docker run --name master --hostname master -p 11311:11311 ₩

--device=/dev/ttyACM0:/dev/ttyACM0 ₩

--device=/dev/ttyUSB0:/dev/ttyUSB0 ₩

-d -it arm64v8/ros:noetic bash

> --name: 컨테이너 이름

> --hostname: 컨테이너 내부의 호스트 이름

> -p: 컨테이너의 포트를 호스트 시스템에 매핑,

형식: -p <호스트 포트>:<컨테이너 포트>

이 옵션은 호스트의 11311 포트를 컨테이너의 11311 포트와 연결하여 외부 시스템이 이 포트를 통해 컨테이너 내부의 로스 마스터와 통신할 수 있도록 설정

> --device: 호스트의 장치 파일을 컨테이너 내부로 전달,

형식: --device=<호스트 경로>:<컨테이너 경로>

컨테이너 내부에서도 동일한 경로로 접근할 수 있도록 설정

$ docker container attach master

2) In master container

$ apt update && apt upgrade -y

$ apt install -y git wget libudev-dev ros-noetic-rosserial-python ros-noetic-tf

➔ 시스템 업데이트 및 필수 패키지 설치

$ cd /root

$ mkdir -p catkin_ws/src && cd catkin_ws/

$ catkin_make

➔ Catkin 작업 공간 생성

$ cd src/

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

$ git clone -b noetic https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git

$ git clone -b noetic https://github.com/ROBOTIS-GIT/DynamixelSDK.git

$ git clone -b noetic https://github.com/ROBOTIS-GIT/ld08_driver.git

➔ Turtlebot3 관련 이미지 다운

$ cd turtlebot3/

$ rm -rf turtlebot3_description/ turtlebot3_example/

$ cd ~/catkin_ws && catkin_make

$ export LDS_MODEL=LDS-02

$ export ROS_MASTER_URI=http://master:11311

$ export ROS_HOSTNAME=master

➔ 환경변수 설정

$ source /opt/ros/noetic/setup.bash

$ source ~/catkin_ws/devel/setup.bash

$ sudo dpkg --add-architecture armhf

➔ 시스템에 armhf 아키텍처 추가 -> ARM 기반 패키지 설치 가능해짐

$ sudo apt update

➔ 패키지 목록을 업데이트하여 새로 추가된 armhf 아키텍처와 관련된 패키지 정보를 가져옴

$ sudo apt install libc6:armhf

➔ Libc6 라이브러리의 ARM 버전 설치

$ export OPENCR_PORT=/dev/ttyACM0

$ export OPENCR_MODEL=burger_noetic

$ rm -rf ./opencr_update.tar.bz2

$ wget ₩

https://github.com/ROBOTIS-GIT/OpenCR-Binaries/raw/master/turtlebot3/ROS1/latest/opencr_update.tar.bz2

➔ OpenCR 업데이트 파일을 Github에서 다운로드

$ tar -xvf opencr_update.tar.bz2

➔ 다운로드한 tar.bz2 파일을 압축 해제 -> 'opencr_update' 디렉토리 생성

$ cd ./opencr_update

$ ./update.sh $OPENCR_PORT $OPENCR_MODEL.opencr

➔ 업데이트 스크립트를 실행하여 OpenCR 보드를 업데이트

$ export TURTLEBOT3_MODEL=burger

$ roslaunch turtlebot3_bringup turtlebot3_robot.launch

3) Make DockerFile

FROM arm64v8/ros:noetic

RUN apt update && apt upgrade -y && apt install -y git wget libudev-dev ros-noetic-rosserial-python ros-noetic-tf

RUN mkdir -p ~/catkin_ws/src/

WORKDIR /root/catkin_ws/src

RUN git clone -b noetic https://github.com/ROBOTIS-GIT/turtlebot3.git ₩

&& git clone -b noetic https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git ₩

&& git clone -b noetic https://github.com/ROBOTIS-GIT/DynamixelSDK.git ₩

```
&& git clone -b noetic https://github.com/ROBOTIS-GIT/ld08_driver.git

WORKDIR /root/catkin_ws/src/turtlebot3

RUN rm -rf turtlebot3_description/ turtlebot3_example/

WORKDIR /root

RUN dpkg --add-architecture armhf ₩

&& apt update ₩

&& apt install -y libc6:armhf

ENV OPENCR_PORT=/dev/ttyACM0

ENV OPENCR_MODEL=burger_noetic

RUN rm -rf ./opencr_update.tar.bz2 ₩

&& wget https://github.com/ROBOTIS-GIT/OpenCR-Binaries/raw/master/turtlebot3/

ROS1/latest/opencr_update.tar.bz2 ₩

&& tar -xvf opencr_update.tar.bz2


# in host PC terminal

$ sudo docker build -t turtlebot3:noetic .
```

➔ 이미지 빌드: 컨테이너를 실행하기 위한 이미지를 만드는 것

```
$ sudo docker run --name tester --hostname tester ₩

-p 11311:11311 ₩

--device=/dev/ttyACM0:/dev/ttyACM0 ₩

--device=/dev/ttyUSB0:/dev/ttyUSB0 ₩

-d -it turtlebot3:noetic bash

$ sudo docker attach tester

# in tester container


$ cd ~/opencr_update
```

$ ./update.sh $OPENCR_PORT $OPENCR_MODEL.opencr

$ cd ~/catkin_ws/ && catkin_make

$ export LDS_MODEL=LDS-02

$ export ROS_MASTER_URI=http://tester:11311

$ export ROS_HOSTNAME=tester

$ source /opt/ros/noetic/setup.bash

$ source ~/catkin_ws/devel/setup.bash

$ export TURTLEBOT3_MODEL=burger

$ roslaunch turtlebot3_bringup turtlebot3_robot.launch

5. TurtleBot3 자동 슬램

참조: https://github.com/winwinashwin/frontier_exploration

Using turtlebot3 (Burger)

<Shell#1 : Gazebo>

$ export TURTLEBOT3_MODEL=burger

$ roslaunch turtlebot3_gazebo turtlebot3_world.launch

<Shell#2 : SLAM + Rviz>

$ export TURTLEBOT3_MODEL=burger

$ roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping

<Shell#3 : move_base>

$ export TURTLEBOT3_MODEL=burger

$ roslaunch turtlebot3_navigation move_base.launch

<Shell#4 : Frontier exploration>

1.frontier exploration 파일을 저장할 워크스페이스 생성

$ mkdir -p frontier_ws/src

$ cd frontier_ws

$ catkin_make

2. src 디렉토리에서 자동 슬램 GitHub 다운로드

$ cd src/ (=>~/frontier_ws/src)

$ git clone https://github.com/winwinashwin/frontier_exploration.git

$ cd ..

$ catkin_make

3. 작업공간 환경 활성화 후 런치 파일 실행 (~/frontier_ws)

$ source devel/setup.bash

$ roslaunch frontier_exploration explore_costmap.launch

6. ROS with Docker Swarm

1. About Docker Swarm

- swarm: 스웜을 만든다는 것은 클러스터를 만든다는 것이다. 클러스터는 여러 개의 서버를 하나의 서버처럼 보는 것이다. 서버는 하나의 호스트 PC로 생각하면 된다.
- node: 노드는 스웜 클러스터에 속한 서버의 단위이다. 보통 하나의 서버에 하나의 도커 데몬만 실행하기 때문에 서버는 곧 노드라고 이해하면 된다.
- Manager node: 매니저노드는 스웜 클러스터 상태를 통합 관리하는 노드이다. 매니저 노드는 워커노드가 될 수 있고 스웜 명령어는 매니저 노드에서만 실행된다.
- Worker node: 워커노드는 매니저노드의 명령을 받아 컨테이너를 생성하고 상태를 체크하는 노드이다.
- Service: 서비스는 기본적인 배포단위이다. 하나의 이미지를 기반으로 생성되고 복제 컨테이너를 여러 개 실행할 수 있다.
- Stack: 스택은 서비스의 묶음 단위이다. 스택이라는 공간 내부에 웹서버 컨테이너, db 컨테이너 등을 구축한다. 하나의 스택으로 묶인 컨테이너들은 기본적으로 같은

overlay 네트워크에 속하게 된다

2. Our Docker Swarm for ROS

Manager node: Raspberry Pi 4

Worker node: Desktop

Services: ROS master, ROS talker, ROS listener

The name of stack: rosstack

"라즈베리파이를 매니저노드로 하고, PC를 워커노드로 한다. 라즈베리파이에서 스웜 클러스터를 생성하여 PC를 클러스터에 추가한다. 라즈베리파이에서 작성된 컴포즈 파일을 통해 스택을 만들어서 라즈베리파이와 PC에 서비스를 배포한다. 라즈베리파이와 PC에 컨테이너가 생성되면, 해당 컨테이너들은 같은 overlay 네트워크에 속한다. ROS listener 컨테이너에 attach하여 토픽을 주고받는지 확인하는 예제이다."

3. Write Compose File

```
# in Raspberry Pi Terminal
$ mkdir -p ~/docker_ws/comp_file/
$ cd ~/docker_ws/comp_file/
$ vim compose.yaml
# Compose file
services:
  master:
    image: ros:noetic
    networks:
      - rosnet
```

```
  environment:
    - ROS_HOSTNAME=master
    - ROS_MASTER_URI=http://master:11311
  command: roscore

talker:
  depends_on:
    - master
  image: ros:noetic
  networks:
    -  rosnet
  environment:
    - ROS_HOSTNAME=talker
    - ROS_MASTER_URI=http://master:11311
  command: rostopic pub /test std_msgs/String "hello world" -r 10

listener:
depends_on:
  - master
image: ros:noetic
networks:
  - rosnet
environment:
  - ROS_HOSTNAME=listener
  - ROS_MASTER_URI=http://master:11311
command: rostopic echo /test
```

networks:

rosnet:

driver: overlay

attachable: true

4. Start Swarm

in Raspberry Pi Terminal

$ docker swarm init

in PC Terminal

$ docker swarm join --token <manager node token> <manager-ip>:2377

"라즈베리파이가 매니저노드가 되었고, PC에서 해당 명령어를 입력하면 PC가 스웜 클러스터에 추가되면서 워커노드가 된다."

in Raspberry Pi Terminal

$ docker node ls

"라즈베리파이에서 해당 명령어를 통해서 현재 스웜 클러스터에 속한 노드들의 정보를 확인할 수 있다."

5. Make Stack

in Raspberry Pi Terminal

$ cd ~/docker_ws/comp_file/

$ docker stack deploy –compose-file compose.yaml rosstack

"라즈베리파이에서 컴포즈 파일이 저장되어 있는 디렉터리로 이동한다. 해당 디렉터리 안에서 docker stack deploy 명령어를 이용하여 서비스들이 작성된 컴포즈 파일을 통해서 스택과 서비스를 만든다."

$ docker stack ls: 현재 docker에 있는 모든 스택을 보여준다. 해당 스택에 몇 개의 서비스가 있는지, 오케스트레이션 툴은 어떤 것을 사용하고 있는지 나타난다.

$ docker service ls: 현재 가동중인 모든 서비스를 보여준다. 상태에 대한 정보도 확인할

수 있다.

$ docker network inspect rosstack-rosnet: 현재 라즈베리파이의 노드에 할당된 서비스가 오버레이 네트워크에 연결된 상태를 확인할 수 있다. 하단 정보를 통해 해당 오버레이 네트워크가 라즈베리파이와 PC에 걸쳐서 연결되어 두 장치의 IP를 가지고 있는 것을 알 수 있다.

in PC Terminal

$ docker network inspect rosstack-rosnet: 마찬가지로 현재 PC의 노드에 할당된 서비스가 오버레이 네트워크에 연결된 상태를 확인할 수 있다.

"만들어진 서비스를 확인하여, listener 서비스가 만들어진 장치에 접속한다. 해당 장치에서 listener 서비스의 컨테이너에 attach하여 토픽을 잘 구독하고 있는지 확인한다."

7. TurtleBot3 teleoperation using Overlay network

1. $ docker build –t pc:noetic
2. $ docker run --name hostpc --hostname hostpc –network rosstack-rosnet –d –it pc:noetic absh
3. catkin_make 실행 후 환경변수 설정
4. 텔레오프 런치 파일 실행
5. 터틀봇 배터리 연결 후 동작 확인