

2023.11.08.수.

Camera를 이용해 특정 line 따라 주행 ROS-Simulation

Kim.TH

Camera를 이용해 특정 line 따라 주행 ROS-Simulation

1. 요약

2. Digital Camera

2.1. Digital Camera

2.2. Calibration

2.3. Camera 장·단점 및 활용

3. 특정 Line 인식 과정

3.1. 영상 처리(특정 색 추출)

3.2. 이미지 중심과 무게 중심

4. Simulation 준비

4.1. ROS?

4.2. Turtlebot3?

4.3. ROS, Turtlebot3 Package 설치

4.4. OpenCV 설치

5. Code 실행 및 결과 고찰

5.1. data 전달 ROS Message Types

5.2. 실습 Code 내용 및 결과 고찰

1. 요약

실습 개요

물류창고와 같이 물품을 어떤 기준으로 분류하여 어느 특정 위치에 이동시키는 작업 환경에서는 분류된 물품에 따라 이동해야 할 곳은 정해져 있다. 이러한 위치 이동이 이미 정해져 있다면 고가의 장비인 LiDAR 같은 센서는 불필요하다고 생각한다. 차라리 Camera를 이용해 Line을 인식하여 Line 따라 목적지까지 이동시키는 방법이 더욱 효율적일 것이다. 이번 실습에서는 특정 색의 Line을 Camera로 data를 받아 그 선을 따라서 주행하는 프로그램을 작성하고, 가상 환경에서 Simulation한 후 결과에 대해 고찰해 본다.

실습 환경

- OS : Ubuntu 20.04
- ROS : Noetic Ninjemys
- Simulator : Gazebo
- Robot : TurtleBot3 - Waffle Pi
- Program Language : C++
- 영상처리 라이브러리 : OpenCV



Ubuntu 20.04



ROS-Noetic



C++



OpenCV

2. Digital Camera

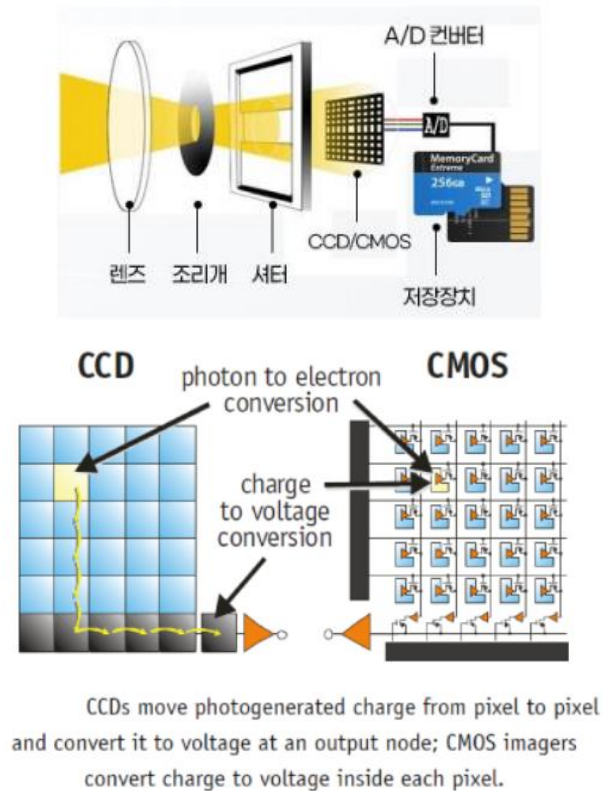
2.1. Digital Camera

2.2. Calibration

2.3. Camera 장·단점 및 활용

2.1. Digital Camera

■ **Digital Camera** : 빛의 밝기 신호를 이미지 센서(CCD or CMOS)를 통해 2차원 배열 형태의 전기적인 신호로 변환하여 영상을 만드는 장치



■ 이미지 센서

- CCD(Charge Coupled Device)

: 각 pixel에서 빛의 세기에 따라 생성된 전하를 shift register를 통해 이동시킨 후, 최종 출력단에서 전기적 신호로 전환되어 출력

: 장점

- 전하 이동이 비교적 손실이 적어 선명한 화질, 색상 구현
- 따라서 노이즈가 적음
- chip size가 작음

: 단점

- 부가 회로가 필요하여 가격 높음
- 전력 소비가 많음
- 최종 출력단에서 전하를 축적하여 전기적 신호로 변환되므로 속도 한계 존재. 영상 처리 속도가 느림

- CMOS(Complementary Metal-Oxide Semiconductor)

: 빛의 세기에 따라 생성된 전하를 각 pixel에서 바로 전기적 신호로 바뀌어 이동

: 장점

- 전력 소비가 작아 발열이 적음
- 각 pixel에서 바로 전기 신호로 바뀌므로 병렬 연산이 가능
=> 고속 영상 처리 가능
- 대량 생산 가능 => 낮은 단가

: 단점

- 전압으로 바로 바뀌어 전달 되므로 전하 손실 존재
=> 이미지 품질 비교적 낮음
- 노이즈 존재

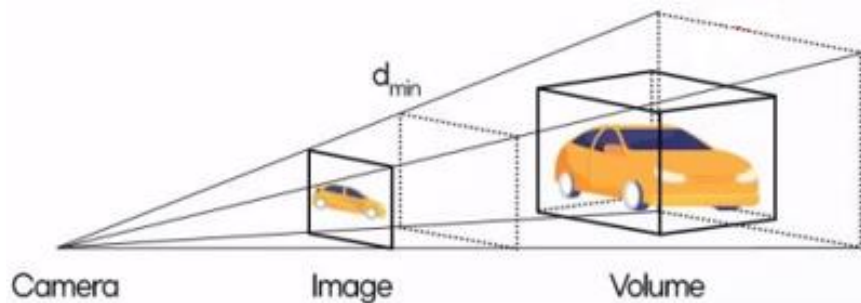
2.2. Calibration

1. Camera Calibration?

: Camera의 내부 파라미터(초점거리, 주점, 비대칭 계수)와 외부 파라미터(Camera 설치 높이, pan, tilt)를 추정하는 과정
=> 이미지(2D)와 실제(3D) 사이의 변환을 가능하게 함

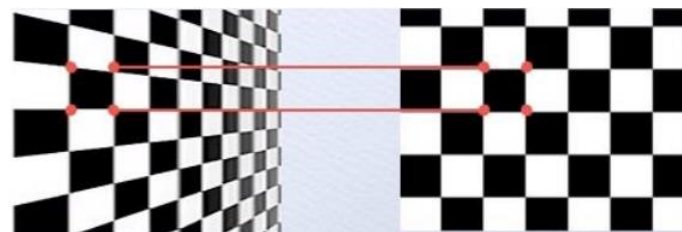
2. 필요성

- Camera의 이미지는 실제 세계(3D)와는 다른 2D 영상임
- 이미지에서 사물의 크기나 거리 등의 정보를 알아내기 위해
서 Camera 내부 파라미터와 외부 파라미터 값을 알아야 함



3. Calibration 방법

- 패턴 이용(ex. 체크 보드)



: 체크 보드의 모든 코너 점들은 균등한 간격이라는 특징을 활용

- 기하학적 단서 이용(ex. 현실 공간 특징)



- 딥러닝 이용

: 거리나 크기 등, 정답을 알고 있는 여러 dataset을 이용하여
학습시켜 Camera 파라미터 값들을 추정

2.3. Camera 장·단점 및 활용

Camera 장·단점

- 장점

- 주변 환경에 대한 색상, 형태 등을 사람의 시야와 가장 유사하게 제공
- 라이다, 레이더에 비해 상대적으로 가격이 저렴
- 데이터 획득 속도가 비교적 빠름
- 카메라의 렌즈를 변경하거나 보정하여 화각 조정 가능

- 단점

- 날씨나 밝기 변화에 민감 => 야간에 성능 저하
- 2D이므로 정확한 거리, 속도 정보 제공 어려움
- 해상도가 높을수록 영상 처리 속도 느려짐

▣ 카메라, 레이더, 라이다, GPS 센서 등 각 센서마다 장·단점이 있기 때문에 단점을 보완할 수 있는 센서들을 조합해서 사용함

자율 주행에서의 Camera 활용

· 신호등 변경 감지

· 차선 인식

ex) LDWS(Lane Departure Warning System, 차선이탈경고),
LKAS(Lane Keeping Assist System, 차선유지보조)

· 객체 인식

ex) FCWS(Forward Collision Warning System, 충돌경고),
ACC(Adaptive Cruise Control),
AEB(Autonomous Emergency Braking system, 긴급제동)

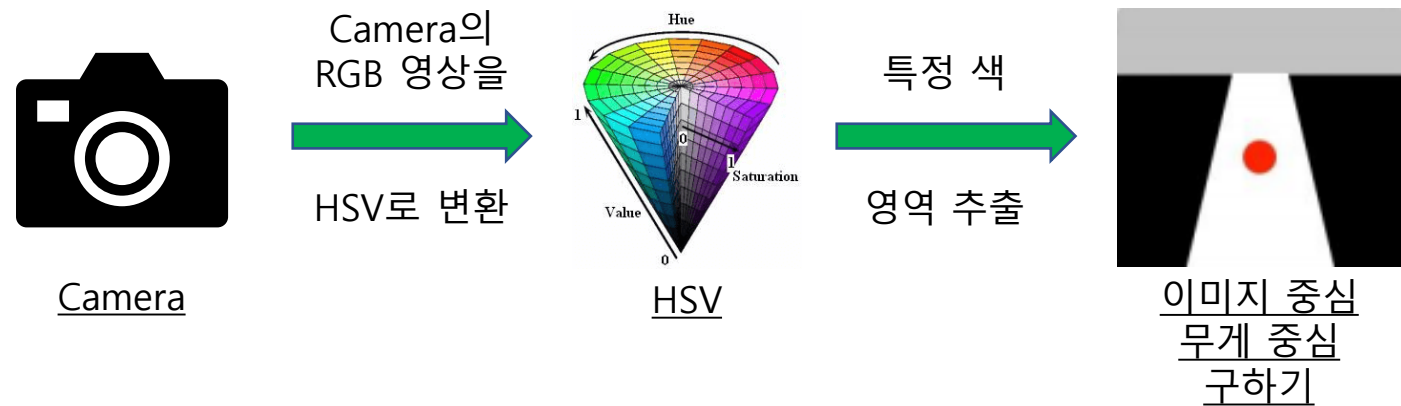
· AVM(Around View Monitor)

: 앞,뒤,좌,우에 장착되어 차량을 Top View로 볼 수 있음
그 외 차선 인식, 주차선 인식, 물체 인식에도 사용

3. 특정 Line 인식 과정

3.1. 영상 처리(특정 색 추출)

3.2. 이미지 중심과 무게 중심



3.1. 영상 처리(특정 색 추출)

■ 영상 처리 : Camera에서 출력된 영상을 받아 목적에 맞게 영상을 가공하는 것(ex. 특정 색 추출)

■ 차량이 특정 Line만 따라 가려면 Camera로 통해 들어오는 이미지 data에서 특정 Line만 추출하고 나머지 pixel은 0값인 검정색으로 영상 처리 해야 함. OpenCV 라이브러리 사용

- ① Camera 센서로부터 cv_ptr 변수에 저장
- ② cvtColor() 함수를 이용하여 BGR 이미지를 HSV로 변환하여 hsv_img 변수에 저장
- ③ HSV 색 공간에서 H(노랑) 60° 근처. OpenCV에서 색상은 360° 표현이 아닌 180° 표현이므로 30° 근처가 됨
=> 노랑일 경우 : $(20 \leq H \leq 40) \&\& (S \geq 100) \&\& (V \geq 100)$
- ④ inRange() 함수를 이용하여 노란색은 흰색(255), 나머지 색은 검정색(0)으로 표현
- ⑤ Camera 이미지 상단은 자율 주행 시 거리가 먼 곳을 나타냄.
따라서 차량과 가까이 있는 곳을 인식하도록 이미지 범위 설정
- ⑥ 이미지 범위 밖의 pixel 값 0으로(검정색) 처리

```
void img_cb(const sensor_msgs::ImageConstPtr& msg){
    cv_bridge::CvImagePtr cv_ptr;
    ① cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::BGR8);
    int h = cv_ptr->image.rows; // 구독한 이미지 세로 크기 구하기
    int w = cv_ptr->image.cols; // 구독한 이미지 가로 크기 구하기
    cv::Mat hsv_img;
    ② cvtColor(cv_ptr->image, hsv_img, CV_BGR2HSV); // HSV로 변경

    // 노란색 추출(색상 60/2 근처, 채도 100 이상, 명도 100 이상)
    // 참고(openCV에서는 색상은 360도로 표현안하고 반인 180도로 표현 됨)
    cv::Mat mask; // OpenCV 라이브러리의 Mat 객체(행렬 데이터 구조)의 mask 변수 선언
    ③ cv::Scalar lower_yellow = cv::Scalar(20, 100, 100); // 각 범위의 낮은 값
    cv::Scalar upper_yellow = cv::Scalar(40, 255, 255); // 각 범위의 높은 값
    ④ inRange(hsv_img, lower_yellow, upper_yellow, mask);

    ⑤ int search_top = 3 * h / 4; // 이미지 높이 상단 기준 3/4지점부터
    int search_bot = 3 * h / 4 + 20; // 3/4지점에서 +20까지의 영역 지정

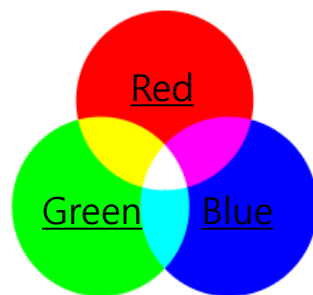
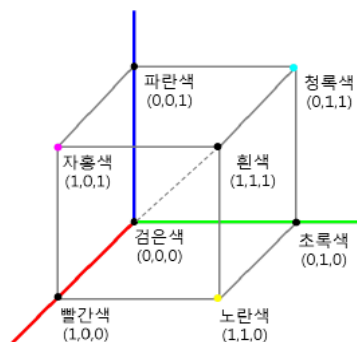
    // mask에 저장된 이미지에서 탐색 영역 밖의 값들은 모두 0으로 처리하여 검정색으로 바꿈
    ⑥ for (int y=0; y<h; y++){
        for (int x=0; x<w; x++){
            if(y<search_top){
                mask.at<uchar>(y,x) = 0; // uchar : 부호 없는 8비트 정수 데이터 타입 자료형
                // 8비트이므로 0~255까지 10진수 표현 가능
            }
            else if(y>=search_bot){
                mask.at<uchar>(y,x) = 0;
            }
        }
    }
}
```

3.1. 영상 처리(특정 색 추출) - HSV 색 공간으로 변환하는 이유

RGB인 빨강, 초록, 파랑 3가지 색상의 조합인 반면, HSV는 색상, 채도, 명도의 조합이므로
색상 값인 각도 범위만 잘 설정한다면 특정 색 추출이 RGB보다 용이함

■ RGB(Red, Green, Blue)

- 빨강, 초록, 파랑 3가지 색의 조합으로 색을 표현
- 영상에서 하나의 pixel은 RGB에 대해 8비트인 0~255 정수로 나타내어 색상을 표현
- RGB의 조합으로 색상을 표현함으로 한 가지 색상만 추출하기에는 부족한 부분이 존재

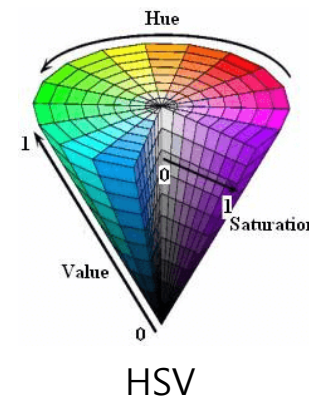


RGB

■ HSV(Hue:색상, Saturation:채도, Value:명도)

- 색상, 채도, 명도의 조합으로 색을 표현
 - Hue : 가시광선 스펙트럼을 고리모양으로 배치
: 스펙트럼의 파장이 제일 긴(빨강색 계열) 색 0°
 - Saturation : 무채색 0. 가장 진한 색 100.
 - Value : 색의 밝기. 흰색&빨간색 100. 검정색 0.

	색 공간 범위	OpenCV 범위
Hue	0°~360°	0~180
Saturation	0%~100%	0~255
Value	0%~100%	0~255



3.2. 이미지 중심과 무게 중심

■ 이미지 중심 : 이미지 크기에서 가로, 세로 1/2 한 지점. 영상 중심

■ 무게 중심 : 일반) 중력에 의한 알짜 회전력(토크)이 0인 지점

: 영상) 영상에서 총 pixel합에 대한 알짜 pixel의 위치

=> 검정색은 pixel값이 0 이므로 색상에 대한 중심을 의미

- OpenCV의 Moments() 함수를 이용하여 구할 수 있음

· m00 : 영상 전체의 pixel 값의 총 합

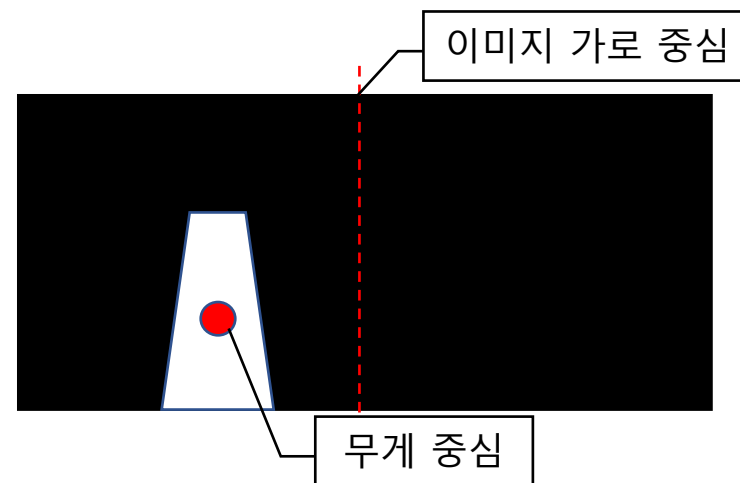
· m10 : 가로 방향 pixel 값의 총 합

· m01 : 세로 방향 pixel 값의 총 합

=> 무게 중심(x', y')

$$x' = m10/m00$$

$$y' = m01/m00$$



■ 차량 제어(이미지 중심과 무게 중심 활용)

- 이미지 가로 중심 기준으로 무게 중심이 왼쪽 위치

=> 차량을 왼쪽으로 제어. 좌회전

- 이미지 가로 중심 기준으로 무게 중심이 오른쪽 위치

=> 차량을 오른쪽으로 제어. 우회전

4. Simulation 준비

4.1. ROS?

4.2. TurtleBot?

4.3. ROS, Turtlebot3 Package 설치

4.4. OpenCV 설치

4.1. ROS?

ROS?

- Robot Operating System의 약자
- 흔히 알고 있는 OS(Linux, Windows, Android 등)의 프로세스 관리 시스템, 인터페이스, 프로그램 유틸 등을 사용하여 다수의 이기종 하드웨어 간의 데이터 송수신, 스케줄링, 에러 처리 등 로봇 소프트웨어 개발을 위한 라이브러리, 프레임워크를 제공하는 미들웨어 Open-source Software
- 현재 ROS1에서 ROS2로 전환되는 시기임
- Version은 알파벳 순으로 명명되고, Ubuntu 출시 해에 맞춰 LTS Version 출시



ROS2
Iron Irwini
Release : 2023년 5월 23일
EOL : 2024년 11월



ROS2
Ardent Apalone
Release : 2017년 12월 8일
EOL : 2018년 12월



Last ROS. LTS
Noetic Ninjemys
Release : 2020년 5월 23일
EOL : 2025년 5월

실습
Version



ROS
Box Turtle
Release : 2010년 3월 2일
EOL : --

4.2. TurtleBot?

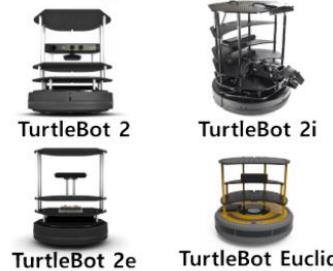
TurtleBot?

- 1967년 교육용 컴퓨터 프로그래밍 언어인 Logo를 바탕으로 만들어진 Turtle 로봇에서 시작
- 2010년 11월 ROS 기반 로봇으로 첫 TurtleBot 나옴
- 각종 sensor와 SBC(Single Board Computer) 등의 Hardware를 Robot에 장착하고 Open-source software를 이용하여 조작 가능함
- TurtleBot의 핵심 기술은 SLAM과 Navigation
TurtleBot에 장착된 각종 Sensor와 SLAM 알고리즘을 사용하여 TurtleBot 주변의 지도를 작성하고, 목적지를 입력하여 현재 위치에서 목적지까지 이동 명령을 내리는 등 로봇의 일반적인 제어를 경험할 수 있음. 또한 TurtleBot 상단에 로봇 팔을 장착 시켜 산업 로봇의 기능 역시 실습 가능함.

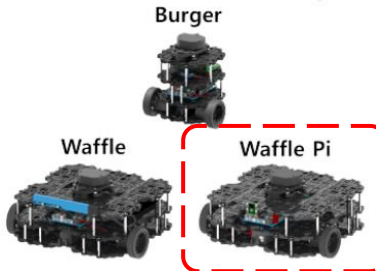
Original TurtleBot (Discontinued)



TurtleBot 2 Family (Discontinued)



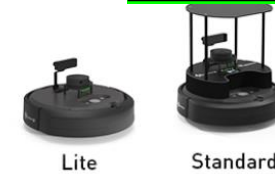
TurtleBot 3 Family



실습 사용 Model

TurtleBot 4 Family

NEW



Waffle Pi

2D-LiDAR (LDS-01, Hitachi-LG Data Storage)

Distance Range	120~3500mm
Scan Rate	300 rpm
Angular Range	360°
Angular Resolution	1°
<u>Camera (Raspberry Pi Camera Module v2.1)</u>	
Resolution	3280 x 2464, 8MP

4.3. ROS, Turtlebot3 Package 설치

ROS 설치

- ROS1은 2020년에 Release된 Noetic 버전이 마지막 이 버전은 Ubuntu 20.04에 맞춰져 있음
- wikiROS 사이트에 설치 과정 상세히 기술되어 있음.
ROS Noetic 버전 : <http://wiki.ros.org/noetic/Installation>

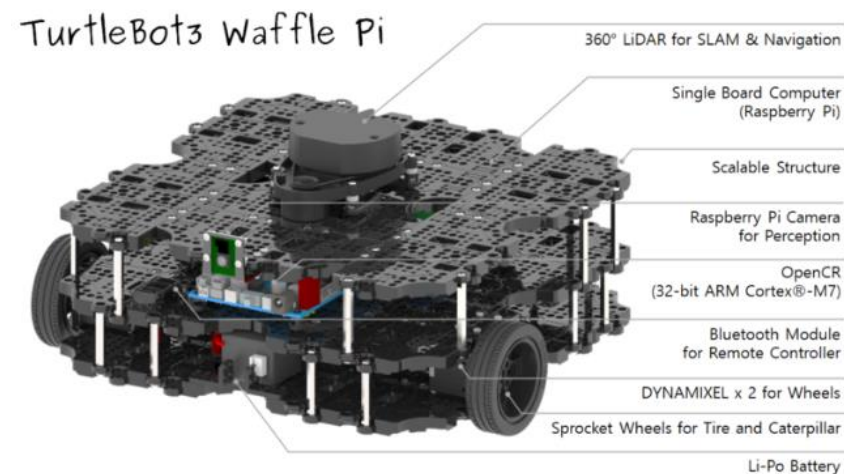


1. Platforms

ROS Noetic Ninjemys is primarily targeted at the Ubuntu 20.04 (Focal) release,

TurtleBot3 Package 설치

- Simulation을 하기 위해 가상 환경과 가상의 TurtleBot 필요 => TurtleBot3 Package가 제공
 - ROBOTIS e-manual 사이트에 설치 과정 기술
<https://emanual.robotis.com/docs/en/platform/turtlebot3/quick-start>
- 위의 사이트에서 ROS 버전을 클릭 후 3.1.3 과정보부터 진행
- 실습은 waffle_pi 모델로 진행하므로 3.1.5 과정의 model 설정 시 waffle_pi를 지정하여 bashrc 파일에 기술



4.4. OpenCV 설치

■ OpenCV 설치

- ① `sudo apt install libopencv-dev`
- ② `sudo apt install ros-noetic-opencv-apps`
- ③ `sudo apt install ros-noetic-cv-*`

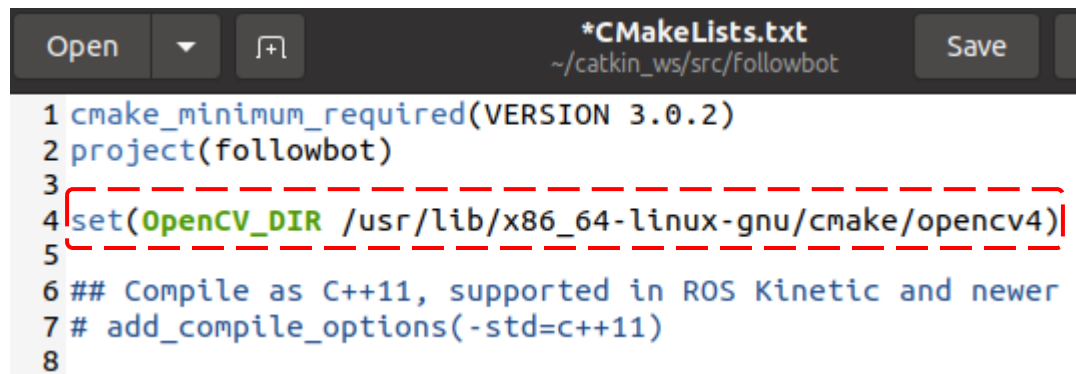
■ 패키지 생성

- ① `catkin_create_pkg` 패키지명 `rospy roscpp sensor_msgs geometry_msgs image_transport cv_bridge OpenCV`

■ 패키지 폴더의 CMakeLists.txt 수정

- ① `gedit CMakeLists.txt`
- ② OpenCV 사용한다는 내용 기입
`set(OpenCV_DIR OpenCVConfig.cmake 파일 있는 경로 기입)`
* 경로 찾기
`/usr` 폴더에서 : `find ./ -name OpenCVConfig.cmake`

```
@ubuntu:/usr$ find ./ -name OpenCVConfig.cmake
./lib/x86_64-linux-gnu/cmake/opencv4/OpenCVConfig.cmake
@ubuntu:/usr$
```



```
Open ▼ [+]
```

***CMakeLists.txt**
~/.catkin_ws/src/followbot

Save

```
1 cmake_minimum_required(VERSION 3.0.2)
2 project(followbot)
3
4 set(OpenCV_DIR /usr/lib/x86_64-linux-gnu/cmake/opencv4)
5
6 ## Compile as C++11, supported in ROS Kinetic and newer
7 # add_compile_options(-std=c++11)
8
```

- ③ 작업 공간에서 컴파일(`catkin_make`)

5. Code 실행 및 결과 고찰

5.1. data 전달 ROS Message Types

5.2. 실습 Code 내용 및 결과 고찰

5.1. 주요 data 전달 ROS Message Types

sensor_msgs

- Camera나 2D/3D-LiDAR의 data를 정의할 때 사용

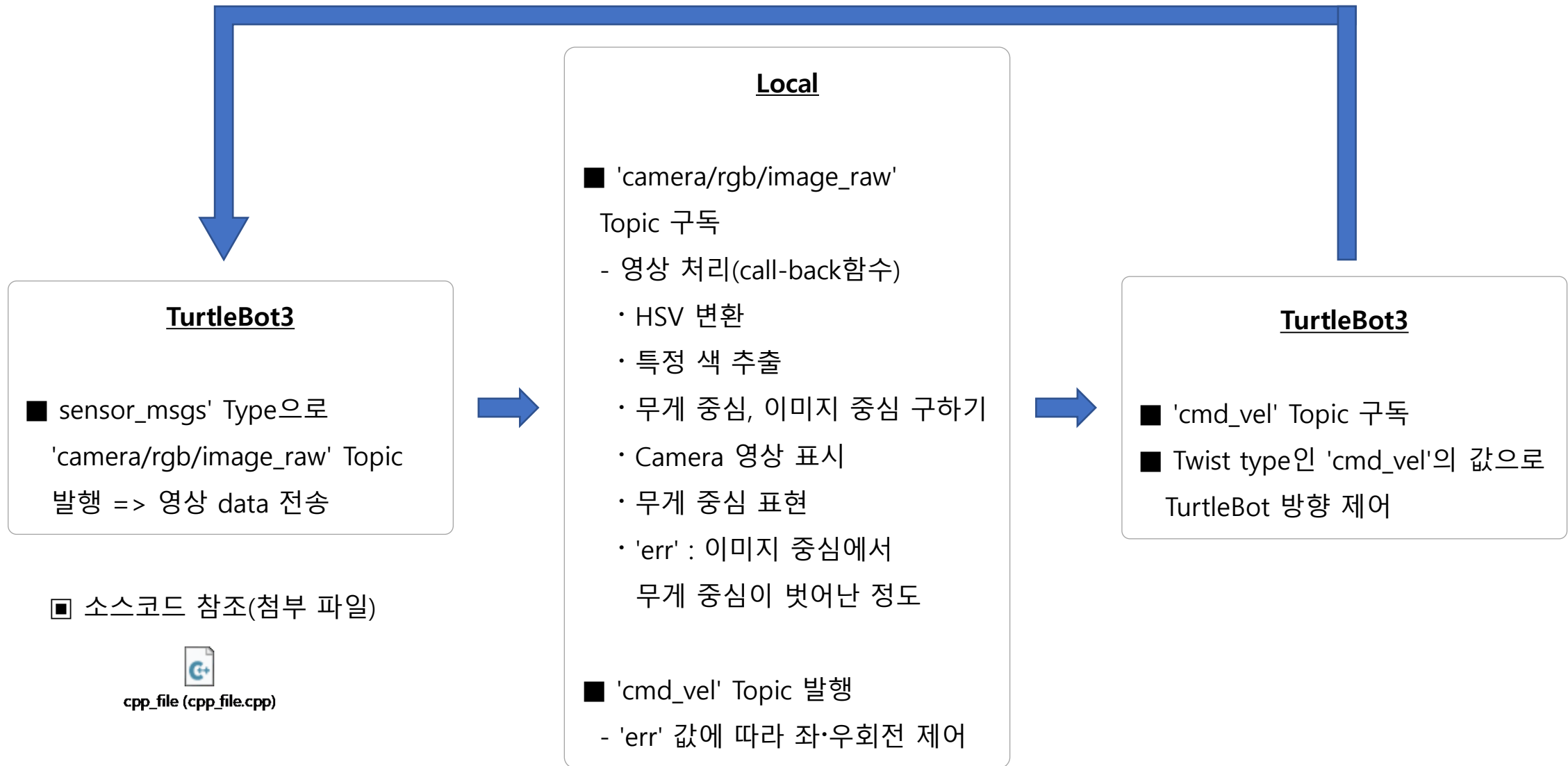
image_transport::ImageTransport

- Camera로부터 얻은 영상 data를 Topic으로 Publish 하거나 Subscribe 할 때 사용
- 영상 data format 형식
 - 이미지 : JPEG, PNG
 - video : Theora streaming

geometry_msgs::Twist

- 속도 관련 data를 주고 받을 때 사용
- linear(직진)와 angular(회전) 두 파트로 각각 3축으로 구성
- linear 파트는 x(+앞), y(+좌), z(+상)으로 구성. 단위 : m/s
- angular 파트는 x(Roll:+좌구르기), y(Pitch:+앞구르기), z(Yaw:+좌회전)으로 구성. 단위 : rad/s

5.2. 실습 Code 내용 및 결과 고찰 - data 처리 과정



제어 값

■ 직진 속도 = 0.6 m/s

■ 회전 속도 = -err/비례상수

* $err = cx(\text{가로 방향 무게 중심}) - w(\text{이미지 가로 pixel 수})/2$

```
int main(int argc, char **argv){
    ros::init(argc, argv, "followbot");
    ros::NodeHandle n;
    image_transport::ImageTransport it(n);
    ros::Publisher cmd_pub = n.advertise<geometry_msgs::Twist>("cmd_vel", 1);
    image_transport::Subscriber img_sub = it.subscribe("camera/rgb/image_raw", 1, img_cb);
    cv::namedWindow("road window");

    ros::Rate loop_rate(30);
    geometry_msgs::Twist cmd; // turtlebot에 명령을 내리기 위한 cmd 변수

    cmd.linear.x = 0.6; // 기본적으로 전진

    while(ros::ok){
        cmd.angular.z = -err/100.;
        cmd_pub.publish(cmd);
        ros::spinOnce();
        loop_rate.sleep();
    }
}
```

[main 함수]

5.2. 실습 Code 내용 및 결과 고찰 - 결과 및 고찰

결과

직진 속도(m/s)	회전 속도(rad/s)	결과
0.3	-err/2000	불안정 주행 불가
	-err/1000	불안정 주행
	-err/100	안정 주행
	-err/50	안정 주행
	-err/20	불안정 주행 불가
0.6	-err/500	불안정 주행
	-err/200	안정 주행
	-err/100	안정 주행
	-err/50	안정 주행
	-err/20	불안정 주행 불가
0.9	-err/200	불안정 주행 불가
	-err/100	불안정 주행 불가
	-err/85	불안정 주행 불가
	-err/75	불안정 주행 불가
	-err/50	불안정 주행 불가

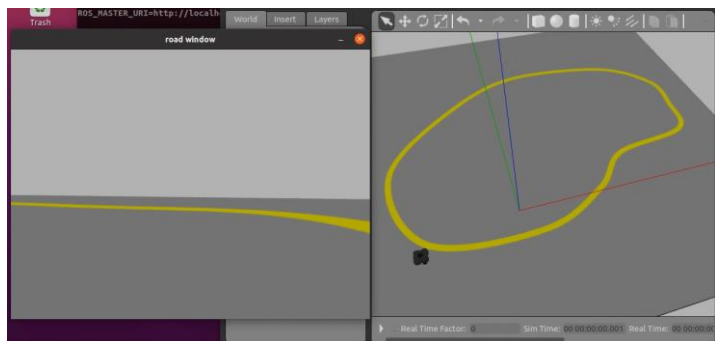
고찰

- 속도가 높을수록 차량 제어 어려움
- err값에서 나눠주는 비례상수 값이 작을수록 회전 속도가 높아져 차량이 휘청거림. 슬립현상 발생
- 비례상수 값이 클수록 회전 속도가 낮아져 커브 각이 클수록 차량이 따라가야 할 특정 line에서 벗어날 가능성이 큼

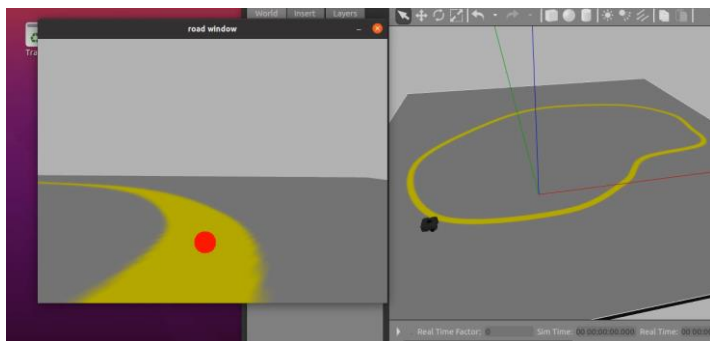
■ 결론

커브가 크지 않는 경로에서는 속도를 어느 정도 올려도 line을 잘 따라 가지만, 어느 정도 빠른 속도와 큰 커브일 경우 비례상수 값을 낮춰 회전 속도를 올리면 차량이 휘청거리거나 차량이 아예 돌아버리는 현상이 나타난다. 따라서 차량이 이동하는 경로에 따라 적당한 직진 속도와 비례상수 값을 찾는게 중요할 것이다.

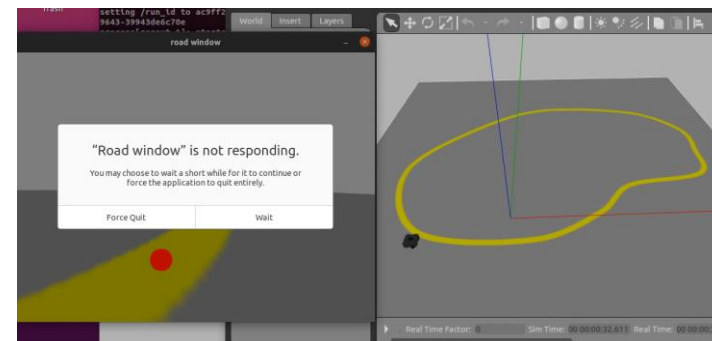
5.2. 실습 Code 내용 및 결과 고찰 - 결과 영상



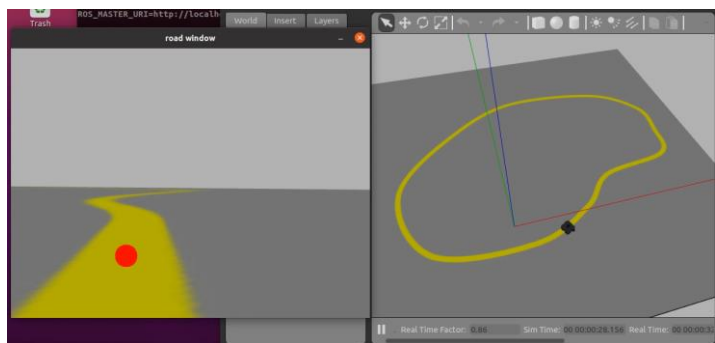
직진 속도 0.3m/s, 회전 비례상수 2000(주행 불가)



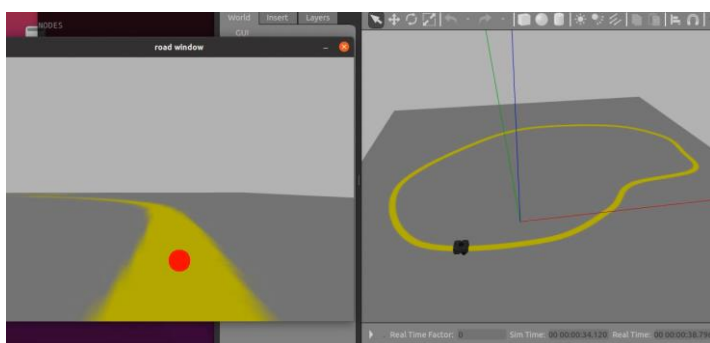
직진 속도 0.6m/s, 회전 비례상수 500(불안정 주행)



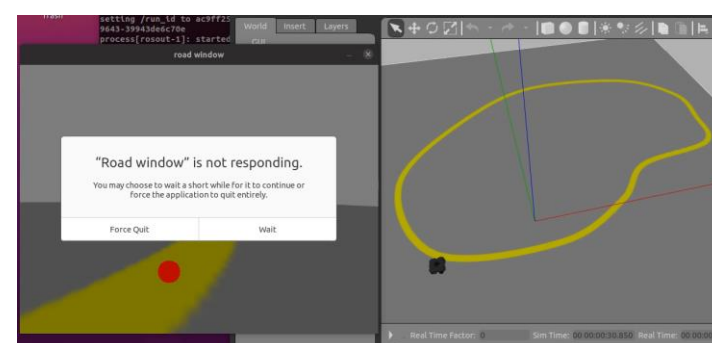
직진 속도 0.9m/s, 회전 비례상수 500(주행 불가)



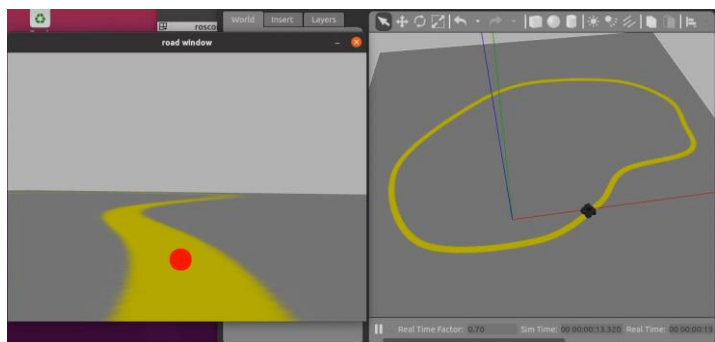
직진 속도 0.3m/s, 회전 비례상수 1000(불안정 주행)



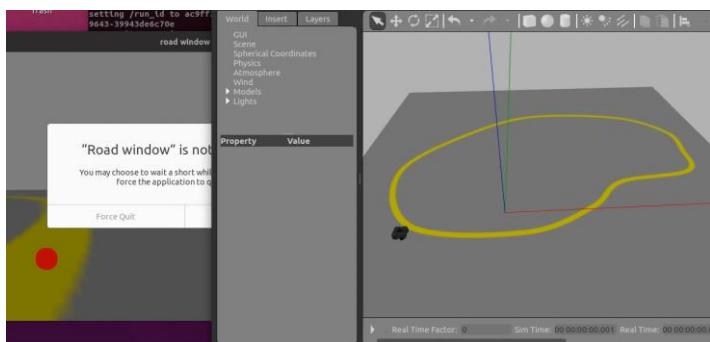
직진 속도 0.6m/s, 회전 비례상수 100(안정 주행)



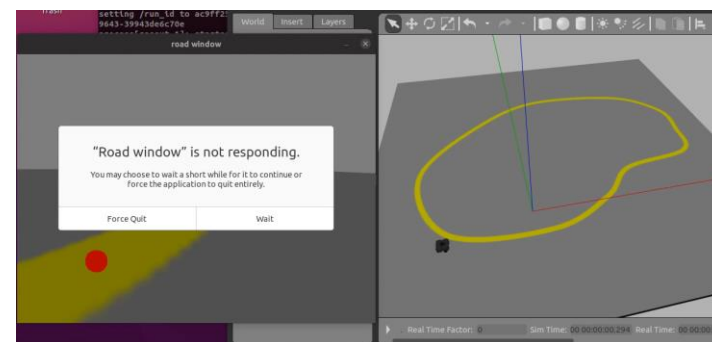
직진 속도 0.9m/s, 회전 비례상수 100(주행 불가)



직진 속도 0.3m/s, 회전 비례상수 100(주행 가능)



직진 속도 0.6m/s, 회전 비례상수 20(주행 불가)



직진 속도 0.9m/s, 회전 비례상수 50(주행 불가)

감사합니다.