2023.11.02.목.

2D-LiDAR를 이용한 장애물 회피

ROS-Simulation

Tae

2D-LiDAR를 이용한 장애물 회피
ROS-Simulation

차 례

1. 요약

1. 요약

실습 개요

로봇을 동작시키는 소프트웨어를 개발하기 위해 여러 가지 도구나 라이브러리들을 모아 놓은 Framework인 ROS(Robot Operating System) 환경에서 2D-LiDAR를 이용하여 로봇의 주변 환경 장애물을 피해서 주행하는 프로그램을 작성하고 가상 환경에서 Simulation 해본다. Simulation 결과에 따라 문제점을 찾고 개선해 본다.

실습 환경

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

2. LiDAR?

2.1. LiDAR란?

LiDAR란?

- Light Detection and Ranging 약자

- 고출력 Pulse Laser를 송신하여
  물체에 반사되어 돌아오는 시간을 분석
  → 물체 인지, 거리 측정

- 기본적 구성
  - Laser 송신부
  - Laser 검출부
  - 신호(data) 수집 및 처리부

- 회전형과 고정형이 있음

[고정형 LiDAR]

[회전형 LiDAR]

2.2. LiDAR 구분

1. 신호 수집 방법에 따라

- **2D-LiDAR(2차원 레이저 스캐너) - Simulation 실습 장비**
  - 단일 레이저 송신기와 단일 수신기로 구성
  - 회전 방식을 이용해 레이저의 진행 방향을 포함하는 특정 평면에서의 영상 정보 수집

- **3D-LiDAR(3차원 레이저 스캐너)**
  - 다수 레이저 송신기와 다수 수신기로 구성
  - 회전 방식을 이용해 레이저의 특정 방향의 시야각에 대해 동시 측정이 가능하도록 하여 3차원 영상을 수집
  - 3D Point Cloud 형태로 데이터 획득 및 표현

- **3차원 Flash LiDAR**
  - 단일 레이저 송신기와 다중 배열 수신기로 구성
  - 단일 레이저를 광시야각으로 확장해 조사해서 일반 카메라와 같이 실시간 영상 정보 수집 가능

2. 거리 측정 방식에 따라

- **ToF(Time of Flight) LiDAR**
  - Pulse Laser의 왕복 시간을 이용하여 거리 측정

- **Phase shift(위상 변이) LiDAR**
  - 빛의 위상 차이를 통해 거리 측정

- **FMCW(주파수 변조법) LiDAR**
  - Frequency Modulated Continuous Wave
  - 주파수에 변화를 준 후 반사되어 돌아왔을 때 주파수 차이를 통해 거리 측정

2.3. LiDAR 장·단점 및 활용

LiDAR 장·단점

· 장점

- 전자기파(빛)를 이용하는 Radar보다 높은 정밀도로 사물 표현이 가능
- 회전형일 경우 모든 각도에서 정보 수집 용이
- 직진성이 강한 레이저를 사용함으로써 먼 거리의 물체도 감지 가능

· 단점

- 현재까지는 제품 단가가 높음
- 레이저의 높은 에너지 때문에 비나 눈 등 날씨의 영향을 많이 받음

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

LiDAR 활용

· 객체 인식

- 동적인 환경을 인식하는데 활용
- 자율 주행에서 주로 보행자, 차량, 자전거, 도로 표면 등 실시간으로 바뀌는 환경을 인식하는 데 활용
- 카메라와 함께 사용하고 딥러닝 기술을 접목하여 더욱 정확한 객체 인식이 되도록 활용

· 위치 추정

- SLAM 기술을 사용하여 실내 또는 터널 등 GPS 신호를 받지 못하는 곳에서 자신의 위치를 추정하는데 활용
- GPS 신호를 받을 수 있는 실외라 하더라도 LiDAR와 랜드마크 등의 정보로 더 정확한 위치를 추정하는데 활용

▣ 차량, 로봇, 스마트공장, 드론산업 등 다양한 분야에서 활용됨

3. Simulation 준비

3.1. ROS?

3.2. TurtleBot?

3.3. ROS, Turtlebot3 Package 설치

3.1. 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, 스케쥴링, 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
2023년 5월 23일 Release

ROS2. LTS
Noetic Ninjemys
2022년 5월 23일 Release

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

ROS
Box Turtle
2010년 3월 2일 Release

Img 출처 : https://docs.ros.org

3.2. TurtleBot?

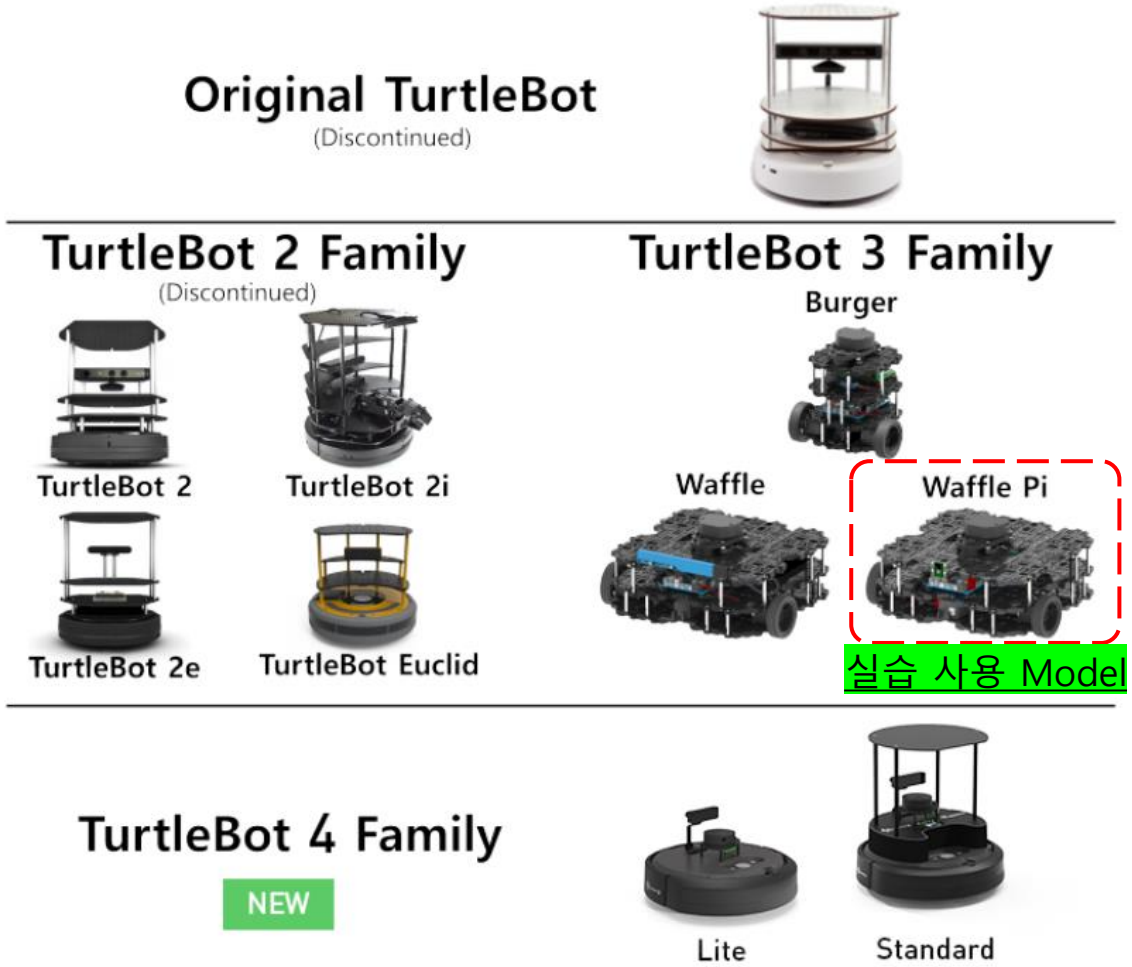
TurtleBot?

- 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.



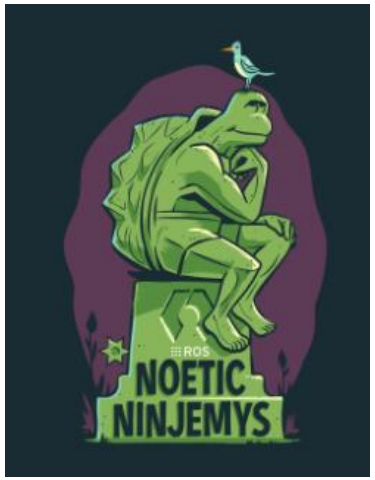
Img 출처 : https://www.turtlebot.com/about

3.3. 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

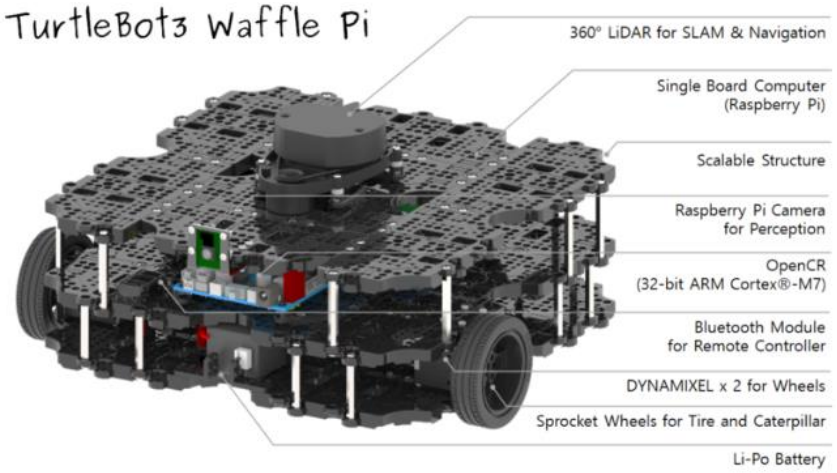
ROS Noetic Ninjemys is primarily targeted at the Ubuntu 20.04 (Focal) release,

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 과정부터 진행
- 실습은 waffle_pi 모델로 진행하므로 3.1.5 과정의 model 설정 시 waffle_pi를 지정하여 bashrc 파일에 기술



4. Code 실행 및 문제점 개선

4.1. data 전달 ROS Message Types

4.2. 실습 Code 내용(좌회전) 및 문제점

4.3. 개선 Code-1 내용(좌·우회전) 및 문제점

4.4. 개선 Code-2 내용(좌·우회전+회전시간) 및 문제점

4.5. 개선 Code-3 내용(좌·우회전+회전시간+후진)

4.1. data 전달 ROS Message Types

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

---

sensor_msgs::LaserScan

- 2D-LiDAR data를 주고 받을 때 사용
- ranges : Turtlebot3에 장착된 LiDAR는
  2D-LiDAR 360° 회전형이며 1°단위로 거리 측정. 이 때 측정된 거리 값은
  ranges[]에 배열형태로 저장됨. 단위 : m/s
- range_min, range_max : LiDAR는 센서마다 측정 가능한 범위가 있음.
  측정 가능 최소거리와 최대거리 값을 가짐. 단위 : m/s

4.2. 실습 Code 내용(좌회전) 및 문제점

TurtleBot3

- 'LaserScan' Type으로 'scan' Topic 발행

Local

- 'scan' Topic 구독
- 거리 정보 바탕으로 while-if문 data를 'Twist' Type으로 'cmd_vel' Topic 발행

TurtleBot3

- 'cmd_vel' Topic 구독
- Twist type인 'cmd_vel'의 값으로 TurtleBot 속도 제어

Call-back 함수 내용

- 콜백 함수 : Topic을 받았을 때 실행되는 함수
- 전방 0° 기준으로 좌우 각각 15° 씩 총 30° 범위의 거리 값을 1° 단위로 받고, 가장 작은 값을 range_ahead 변수에 담아 main함수로 전달

Main 함수 내용

- range_ahead(거리) 값이 0.8 미만이 되면 전방 속도 0m/s, 회전 속도 +4rad/s 값을 'cmd_vel' Topic으로 TurtleBot3로 전달
- range_ahead(거리) 값이 0.8 이상이면 전방속도 +0.6m/s, 회전 속도 0rad/s 값을 'cmd_vel' Topic으로 TurtleBot3로 전달

```
void scan_cb(const sensor_msgs::LaserScan::ConstPtr& msg){
    // range_ahead = msg -> ranges[0];
    range_ahead = 3.0;      // 최소값을 찾기 전 대략 큰 값으로 초기화.
                            // 0.8m 이하 거리가 될 때 회전하도록 프로그램 하였으므로
                            // 0.8보다는 큰 값으로 초기화.

    for(int i=0; i<=15; i++){
        // if(msg->ranges[i] >= msg->range_min){
        // LiDAR가 측정 가능 거리 범위를 벗어난 값 버리기
        if((msg->ranges[i] >= msg->range_min) && (msg->ranges[i] <= msg->range_max)){
            if(msg->ranges[i] < range_ahead){
                range_ahead = msg->ranges[i];
            }
        }
    }

    for(int i=359; i>=345; i--){
        // if(msg->ranges[i] >= msg->range_min){
        if((msg->ranges[i] >= msg->range_min) && (msg->ranges[i] <= msg->range_max)){
            if(msg->ranges[i] < range_ahead){
                range_ahead = msg->ranges[i];
            }
        }
    }
```

```
int main(int argc, char **argv){
    ros::init(argc, argv, "go_scan");
    ros::NodeHandle n;
    ros::Publisher cmd_pup = n.advertise<geometry_msgs::Twist>("cmd_vel", 1);
    ros::Subscriber scan_sub = n.subscribe<sensor_msgs::LaserScan>("scan", 1, scan_cb);

    ros::Rate loop_rate(20);
    geometry_msgs::Twist cmd;

    while(ros::ok()){
        if(range_ahead < 0.8){      // 거리가 0.8m 미만이면
            cmd.linear.x = 0;
            cmd.angular.z = 0.4;    // 0.2rad/s 속도로 제자리 회전하라
        }
        else{                       // 0.8m 이상이면
            cmd.linear.x = 0.6;      // 0.3m/s로 직진하라.
            cmd.angular.z = 0;
        }
        cmd_pup.publish(cmd);
        ros::spinOnce();
        loop_rate.sleep();
    }
}
```

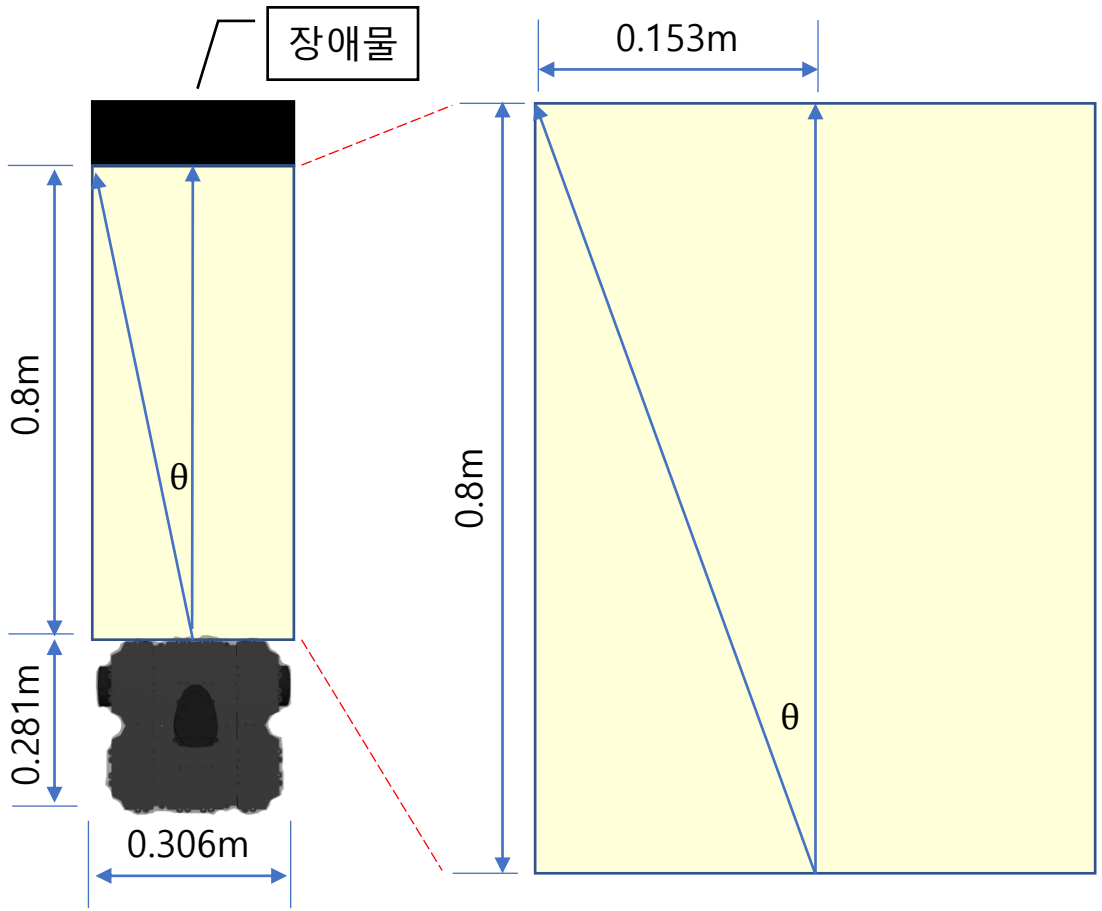
4.2. 실습 Code 내용(좌회전) 및 문제점 - Call-back 함수 추가 내용

왜 전방 기준으로 좌우 각각 15° 씩인가?

- TurtleBot3의 가로 size인 0.306m 범위 내에 장애물이 있으면 TurtleBot3와 충돌하게 됨
- Program Code에서 전방으로 0.8m 범위 내에 사물이 있다면 피하도록 되어 있으므로 0.8m 내에 장애물이 없도록 설계해야 함
- 오른쪽 그림처럼 장애물 탐지 최대 거리 0.8m로 잡고 TurtleBot3의 가로 길이 절반과 삼각함수를 이용하면 0.8m 앞의 TurtleBot3가 피해야할 장애물에 대한 각도(θ)를 구할 수 있음
- 이번 문제에서는 10.83° 이며, 여유분을 임의로 주어 전방 0° 기준 좌우 15° 로 설정함

▣ 각도가 너무 크면 피하지 않아도 되는 장애물도 인지하여 불필요한 로봇 제어를 하게 됨

▣ 각도가 계산한 값보다 작으면 장애물과 충돌하게 됨



waffle pi size(LxWxH)

281 x 306 x 141 mm

$\tan\theta = 0.153/0.8$

$\theta = \tan^{-1}0.153/0.8$

$= 10.83^{\circ}$

4.2. 실습 Code 내용(좌회전) 및 문제점

[영상. 실습 Code 문제점]

실습 Code 문제점

· **문제점**

- 장애물이 인지되면 좌회전만 하기 때문에 4각형의 방 안에서 어느 한 부분만 계속 맴돌게 됨

· **원인**

- 현재 Code는 0.8m 이내에 장애물이 인지되면 정지하여 좌회전만 하도록 프로그램 되어 있기 때문
- 만약 TurtleBot3가 로봇 청소기이거나 지도 작성을 목적으로 사용된다면 제대로 된 기능을 한다고 보기 어려움

실습 Code 개선 방안

· 15° 일 때 거리 값과 -15° 일 때 거리 값을 비교하여 **거리가 더 먼 방향으로 회전**하도록 개선

4.3. 개선 Code-1 내용(좌·우회전) 및 문제점

Call-back 함수 추가 내용

- 15˚, - 15˚ 일 때의 거리 값을 받기 위해 새로운 range_ahead_min, range_ahead_max 두 개의 변수 추가

Main 함수 수정 내용

- (range_ahead(거리) < 0.8) & (15˚거리 >= -15˚거리) 이면 전방 속도 0m/s, 회전 속도 +4rad/s(좌회전) 값을 'cmd_vel' Topic으로 TurtleBot3에 전달

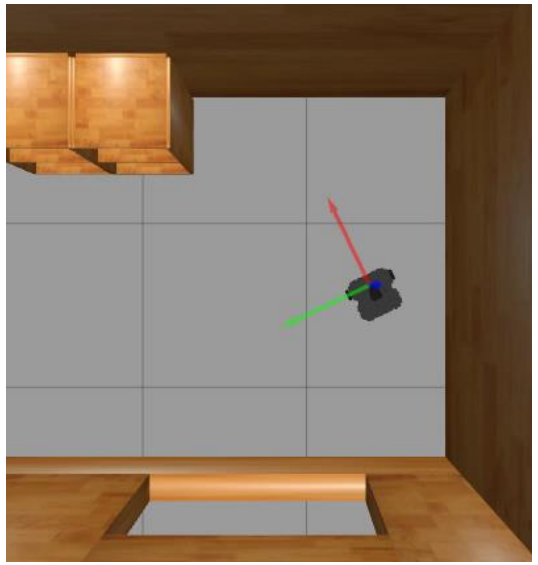
- (range_ahead(거리) < 0.8) & (15˚거리 < -15˚거리) 이면 전방 속도 0m/s, 회전 속도 -4rad/s(우회전) 값을 'cmd_vel' Topic으로 TurtleBot3에 전달

- 그 외 직진 속도 +0.6m/s 값을 'cmd_vel' Topic으로 TurtleBot3에 전달

```
float range_ahead;
float range_ahead_min;
float range_ahead_max;

void scan_cb(const sensor_msgs::LaserScan::ConstPtr& msg){
    range_ahead = 3.0;      // 최소값을 찾기 전 대략 큰 값으로 초기화.
                            // 0.8m 이하 거리가 될 때 회전하도록 프로그램 하였으므로
                            // 0.8보다는 큰 값으로 초기화.

    for(int i=0; i<=15; i++){
        // LiDAR가 측정 가능 거리 범위를 벗어난 값 버리기
        if((msg->ranges[i] >= msg->range_min) && (msg->ranges[i] <= msg->range_max)){
            if(msg->ranges[i] < range_ahead){
                range_ahead = msg->ranges[i];
            }
        }
        if (i==15){
            range_ahead_min = msg->ranges[i];
        }
    }

    for(int i=359; i>=345; i--){
        if((msg->ranges[i] >= msg->range_min) && (msg->ranges[i] <= msg->range_max)){
            if(msg->ranges[i] < range_ahead){
                range_ahead = msg->ranges[i];
            }
        }
        if (i==345){
            range_ahead_max = msg->ranges[i];
        }
    }

    printf("range_ahead: %f\n", range_ahead);
}
```

```
int main(int argc, char **argv){
    ros::init(argc, argv, "go_scan");
    ros::NodeHandle n;
    ros::Publisher cmd_pup = n.advertise<geometry_msgs::Twist>("cmd_vel", 1);
    ros::Subscriber scan_sub = n.subscribe<sensor_msgs::LaserScan>("scan", 1, scan_cb);

    ros::Rate loop_rate(20);
    geometry_msgs::Twist cmd;

    while(ros::ok()){
        if(range_ahead < 0.8){      // ranges 방향의 거리가 0.8m 미만이면
            if (range_ahead_min >= range_ahead_max){
                cmd.linear.x = 0;
                cmd.angular.z = 0.4;    // 0.4rad/s 속도로 제자리 좌회전하라
            } else {
                cmd.linear.x = 0;
                cmd.angular.z = -0.4;    // -0.4rad/s 속도로 제자리 우회전하라
            }
        }
        else{                       // 0.8m 이상이면
            cmd.linear.x = 0.6;      // 0.6m/s로 직진하라.
            cmd.angular.z = 0;
        }
        cmd_pup.publish(cmd);
        ros::spinOnce();
        loop_rate.sleep();
    }
}
```

4.3. 개선 Code-1 내용(좌·우회전) 및 문제점



[영상. 개선 Code-1 문제점]

개선 Code-1 문제점

· **문제점**

- 각진 코너에서 제자리 좌우 회전을 반복함

· **원인**

- 제자리 좌회전하면 우측인 -15° 거리 값이 멀어져 제자리 우회전을 하게 되고, 제자리 우회전을 하다 보면 좌측인 15° 거리 값이 멀어져 다시 제자리 좌회전을 하게 됨. 반복

개선 Code-1 해결 방안

· 15° 일 때 거리 값과 -15° 일 때 거리 값을 비교하여 거리가 더 먼 방향으로 회전 할 때 **회전 시간을 주어** LiDAR가 코너가 아닌 평면을 보도록 개선

4.4. 개선 Code-2 내용(좌·우회전+회전시간) 및 문제점

시간 기록을 위해 전역 변수 지정

- 회전 시간을 주기 위해 'ros::Time' 이라는 ROS Library 사용
- 단위 : s(초)

★ 주의) 단위가 초라서 숫자 형태이지만 자료형은 단순히 int나 float형이 아님. 숫자 자료형과 연산 시 오류 뜸

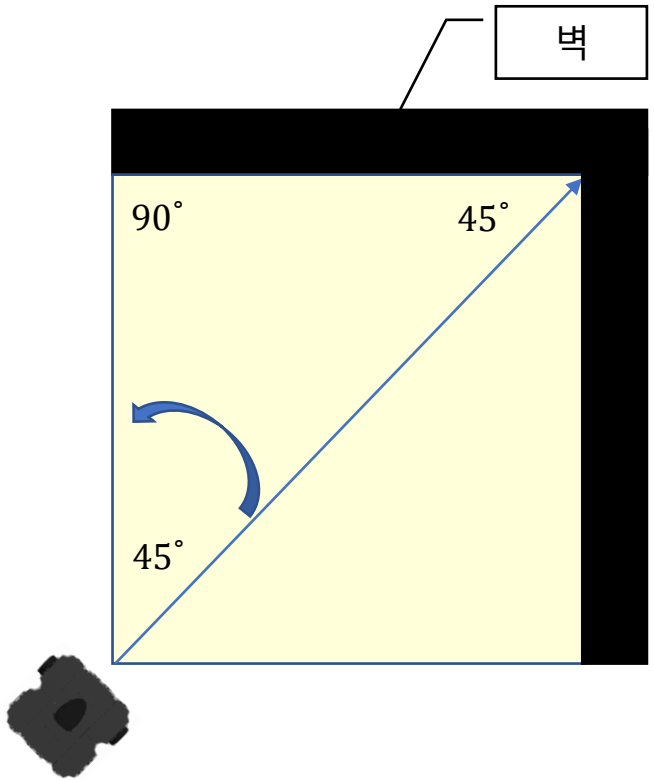
Main 함수 수정 내용

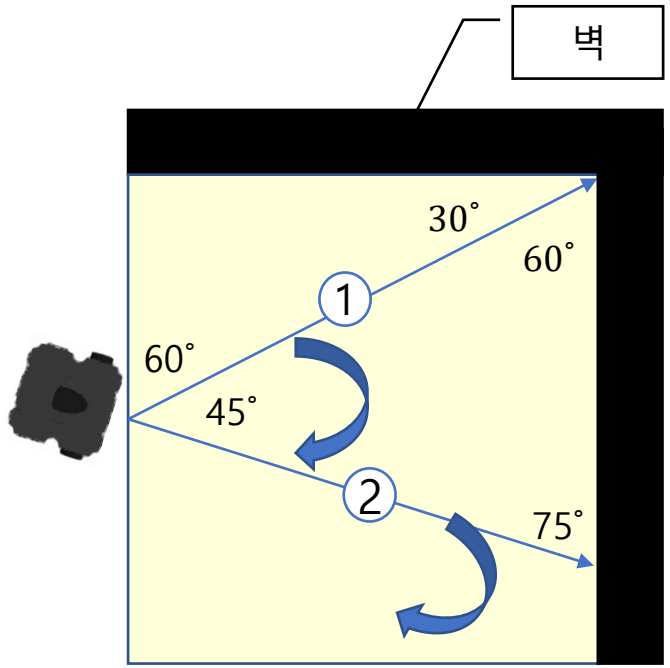
- 거리 값이 0.8m 미만이 되면 현재 시간을 turn_start_time 변수에 저장
- 시간이 지남에 따라 그 때의 현재시간에서 turn_start_time 값을 뺀 값이 2가 될 때까지, 즉, 2초 동안 회전

```
float range_ahead;
float range_ahead_min;
float range_ahead_max;
ros::Time turn_start_time;
```

```
int main(int argc, char **argv){
    ros::init(argc, argv, "go_scan");
    ros::NodeHandle n;
    ros::Publisher cmd_pup = n.advertise<geometry_msgs::Twist>("cmd_vel", 1);
    ros::Subscriber scan_sub = n.subscribe<sensor_msgs::LaserScan>("scan", 1, scan_cb);

    ros::Rate loop_rate(20);
    geometry_msgs::Twist cmd;

    while (ros::ok()) {
        if (range_ahead < 0.8) {
            if (range_ahead_min >= range_ahead_max) {
                turn_start_time = ros::Time::now();
                while ((ros::Time::now() - turn_start_time).toSec() <= 2.0) {
                    cmd.linear.x = 0;
                    cmd.angular.z = 0.4;

                    cmd_pup.publish(cmd);
                    loop_rate.sleep();
                }
            }

            if (range_ahead_min < range_ahead_max) {
                turn_start_time = ros::Time::now();
                while ((ros::Time::now() - turn_start_time).toSec() <= 2.0) {
                    cmd.linear.x = 0;
                    cmd.angular.z = -0.4;

                    cmd_pup.publish(cmd);
                    loop_rate.sleep();
                }
            }
        } else {
            cmd.linear.x = 0.6;
            cmd.angular.z = 0;
        }
        cmd_pup.publish(cmd);
        ros::spinOnce();
        loop_rate.sleep();
    }
```

4.4. 개선 Code-2 내용(좌·우회전+회전시간) 및 문제점 - 지속 시간 2초 기준



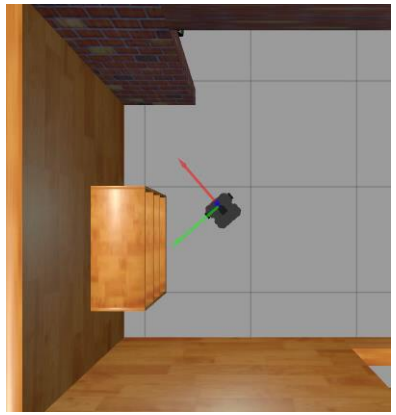


지속 시간 2초 기준

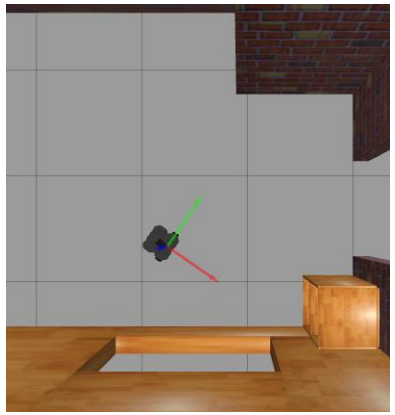
・45˚ = 45 x (π/180)
= 0.79 rad

・회전 속도 = 0.4 rad/s이므로
2초 뒤 0.8 rad 회전하여
평면을 봄으로써 그 구역을 벗어나게 함

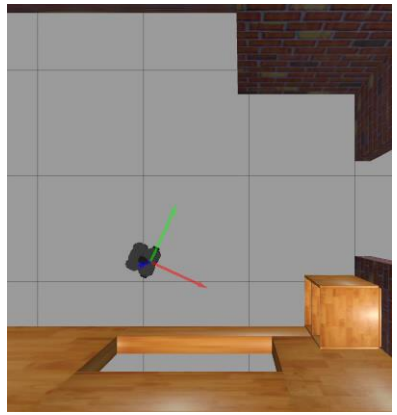
4.4. 개선 Code-2 내용(좌·우회전+회전시간) 및 문제점



[영상1. 개선 Code-2 문제점-1]



[영상2. 개선 Code-2 문제점]



[영상3. 개선 Code-2 지속시간 3초]

개선 Code-2 문제점

· **문제점**

- 대체로 코너 탈출을 잘 하지만 어떤 특정 각도로 코너 진입 시 제자리에서 좌우 회전을 반복(영상 1, 2)
  => 해결 방안 : 지속시간 늘리기
  => 탈출은 곧 잘 하지만 불안정(영상 3)

· **원인**

- 코너에서는 회전하면 회전 반대 방향의 거리가 더 멀어짐
- 지속 시간이 끝난 후 반대 방향 각도의 거리가 멀어져 반대 방향으로 회전. 그러면 다시 그 반대 방향의 거리가 더 멀어져 지속 시간 끝난 후 또 다시 회전 방향 전환. 반복

개선 Code-2 해결 방안

· **후진 추가**

- 회전과 후진을 함께 함으로써 동일 시간 동안 각도 변화를 더 크게 줄 수 있음
- 회전 지속 시간을 줄여 코너 탈출 시간을 줄일 수 있음

4.5. 개선 Code-3 내용(좌·우회전+회전시간+후진)

Main 함수 수정 내용

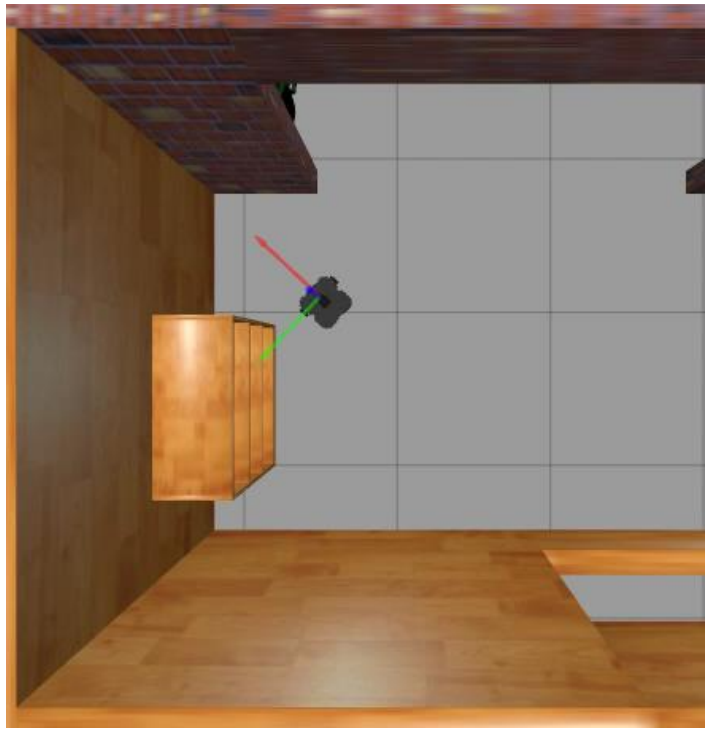
・2.5초 동안 회전과 함께 linear.x 값에 각각 -0.08m/s 값을 'cmd_vel' Topic으로 TurtleBot3에 전달

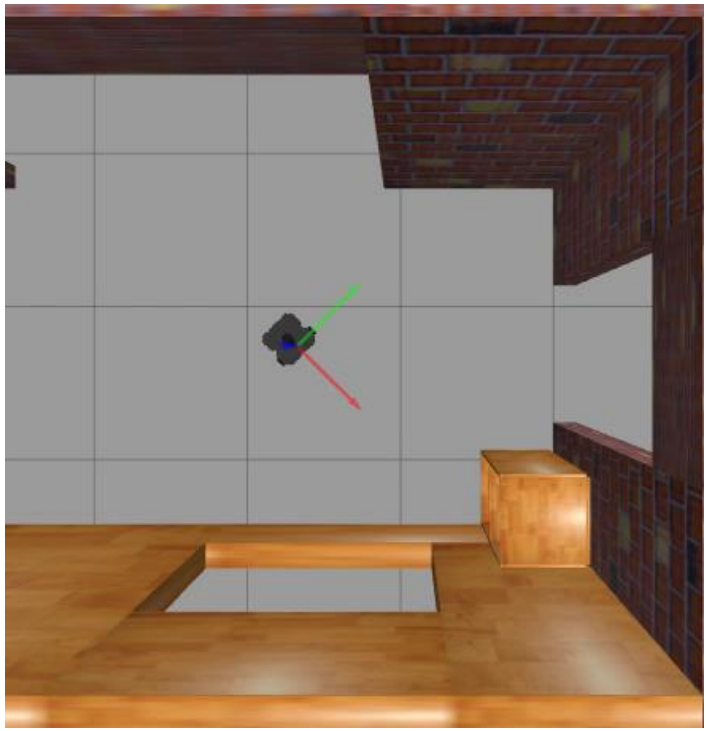
```
while (ros::ok()) {
    if (range_ahead < 0.8) {
        if (range_ahead_min >= range_ahead_max) {
            turn_start_time = ros::Time::now();
            while ((ros::Time::now() - turn_start_time).toSec() <= 2.5) {
                cmd.linear.x = -0.08;
                cmd.angular.z = 0.4;

                cmd_pup.publish(cmd);
                loop_rate.sleep();
            }
        }

        if (range_ahead_min < range_ahead_max) {
            turn_start_time = ros::Time::now();
            while ((ros::Time::now() - turn_start_time).toSec() <= 2.5) {
                cmd.linear.x = -0.08;
                cmd.angular.z = -0.4;

                cmd_pup.publish(cmd);
                loop_rate.sleep();
            }
        }
    } else {
        cmd.linear.x = 0.6;
        cmd.angular.z = 0;
    }
    cmd_pup.publish(cmd);
    ros::spinOnce();
    loop_rate.sleep();
}
```

4.5. 개선 Code-3 내용(좌·우회전+회전시간+후진)

개선 Code-3 결과

· 문제가 되었던 코너에서 쉽게 탈출함





[영상. 개선 Code-3]

5. 결론 및 고찰

5. 결론 및 고찰

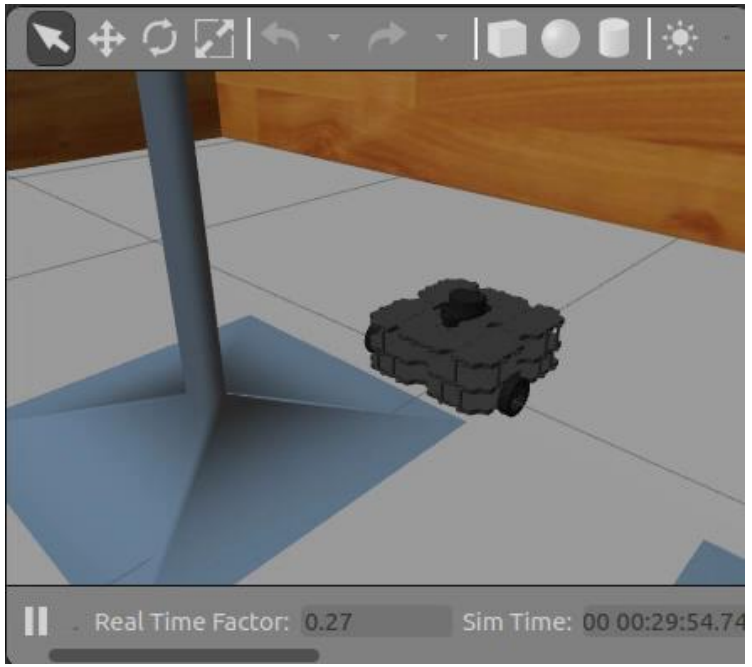
· 다양한 환경 - 환경에 따라 센서 정보를 실시간으로 받아 처리하는 방법은 무조건 옳다는 생각은 잘못된 생각.

처음 실습 Code(장애물 등장 시 좌측으로 회피)를 작성 시에는 모든 장애물을 만나면 피할 수 있다고 생각하였으나, 구현 시 단순 사각형의 닫힌 공간에서는 같은 루트로만 주행하였다. 만약 로봇이 서비스 로봇인 청소 로봇이거나 지도 작성용 로봇이라면 로봇의 역할을 제대로 하였다고 할 수 없다. 이를 개선하기 위해 장애물 탐지 각, 최대 최소 각에서의 거리 값을 바탕으로 좌우방향으로 회전하게 하였지만 이 역시 코너에서는 센서 정보를 실시간으로 받음으로써 로봇은 좌우 회전을 제자리에서 반복할 뿐이었다. 이 문제는 센서 정보를 특정 시간 동안 받지 않고 로봇 제어 명령을 특정 시간동안 계속 수행하게 함으로써 어느 정도 해결할 수 있었다.

가상 환경에서 생각지도 못했던 상황이 발생하여 개선하였지만, 실세계에서는 더욱 다양한 환경이 존재할 것이다. 프로그램이 완벽하다고 생각할지라도 로봇이 쓰이는 현장에서의 점검은 꼭 필요할 것이라 느꼈다.

· 로봇의 위치 정보의 중요성 - 2D-LiDAR의 한계점. 3D-LiDAR나 SLAM 기술로 극복 가능 할 것.

코너 같은 장애물을 회피할 수 있어도 LiDAR에 잡히지 않는 장애물 앞에서는 로봇이 제자리에서 바퀴만 맴돌았다.(영상1) 2D-LiDAR는 설치 위치(높이)에서 평면으로 Laser를 쏘기 때문이다. 이는 수직 방향으로도 측정가능한 3D_LiDAR를 통해 극복할 수 있겠으나 가격이 문제가 될 것이다. 아니면 지도를 이용하여 로봇 자신의 위치를 알 수 있는 SLAM 기술과 함께 사용한다면, 자신의 위치가 계속 제자리일 때 후진하며 회전함으로써 로봇의 주변 환경을 바꿔 센서에 잡히지 않는 장애물도 회피할 수 있을 것이다.



[영상. 2D-LiDAR 한계]

감사합니다.