스마트 모빌리티 프로그래밍

Ch 19. 실내외 측위 (Localization), Mapping, SLAM

김 영 탁

영남대학교 기계IT대학 정보통신공학과

(Tel : +82-53-810-3940; E-mail : yse09@ynu.ac.kr)

Outline

- ◆ 실외 측위(Outdoor Localization)
  - GPS
- ◆ 실내 측위(Indoor Localization)
  - WiFi Finger Printing
  - SLAM
- ◆ LiDAR (Light Detection and Ranging)
- ◆ Localization
- ◆ Map 구성 (Mapping)
- ◆ SLAM (Simultaneous Localization and Mapping)
- ◆ LiDAR 기반 SLAM 구성 및 활용

실외 측위 (Outdoor Localization)

측위 (Localization)

◆ 측위 (Localization) 이란 ?

- 다양한 센서들을 사용하여 위치를 추정/측정

◆ 실외 측위 방법/기술

- GNSS (Global Navigation Satellite System) – GPS

◆ 실내 측위 방법/기술

- WiFi fingerprinting
- SLAM (simultaneous Localization and Mapping)

GNSS (Global Navigation Satellite System)

◆ Global Positioning System (GPS) – source : Wikipedia

- The Global Positioning System (GPS), also known as Navstar, is a global navigation satellite system (GNSS) that provides location and time information in all weather conditions, anywhere on or near the Earth where there is an unobstructed line of sight to four or more GPS satellites. The GPS system provides critical positioning capabilities to military, civil, and commercial users around the world.
- The GPS project was launched in the United States in 1973 , Advances in technology and new demands on the existing system have now led to efforts to modernize the GPS and implement the next generation of GPS Block IIIA satellites and Next Generation Operational Control System (OCX).
- In addition to GPS, other systems are in use or under development.
- The Russian Global Navigation Satellite System (GLONASS) was developed contemporaneously with GPS, but suffered from incomplete coverage of the globe until the mid-2000s.
- There are also the planned European Union Galileo positioning system, China's BeiDou Navigation Satellite System, the Japanese Quasi-Zenith Satellite System, and India's Indian Regional Navigation Satellite System.

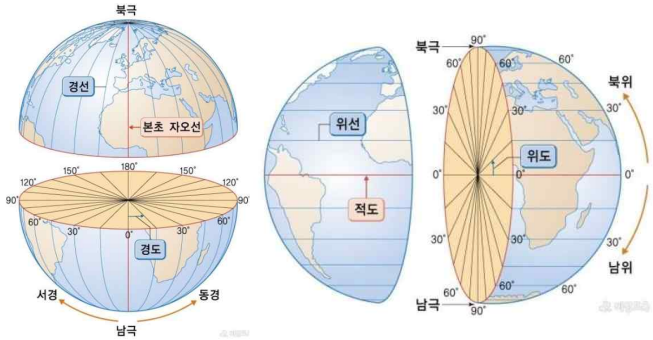
GNSS (Global Navigation Satellite System)

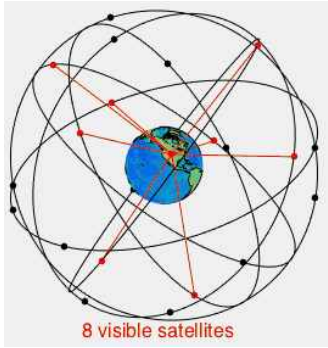
◆ GNSS positioning with Satellites

- GNSS provides location and time information in all weather conditions, anywhere on or near the Earth where there is an unobstructed line of sight to four or more GPS satellites.

◆ GNSS 좌표

- 10진법으로 소수점까지 표함된 표현을 사용
- 일반적으로 사용되는 도분초 표시를 위하여 추가 변환 필요

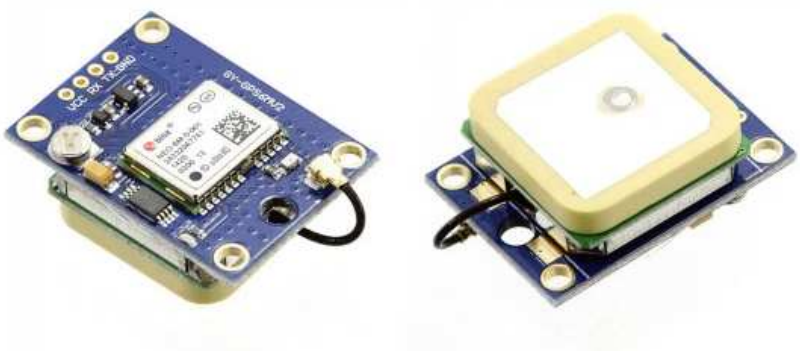




GNSS Module - NEO-8M

◆ NEO-8M

- NEO-M8 module family which provides concurrent reception of up to 3 GNSS (GPS, Galileo, GLONASS, BeiDou).
- NEO-M8 is backward compatible with NEO-7, NEO-6 and NEO-5 families.
- Data sheet : https://www.u-blox.com/sites/default/files/NEO-M8-FW3_DataSheet_%28UBX-15031086%29.pdf
- https://www.u-blox.com/sites/default/files/products/documents/ NEO-7_DataSheet_%28UBX-13003830%29.pdf



GNSS Application

◆ Block diagram

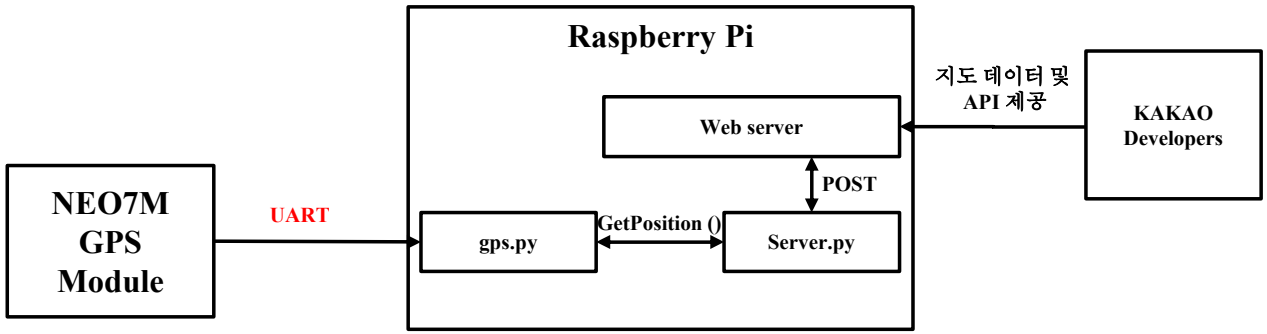


Figure 2: Pin Assignment

Figure 2: Pin assignment

GNSS Application

◆ NMEA (National Marine Electronics Association) Message format

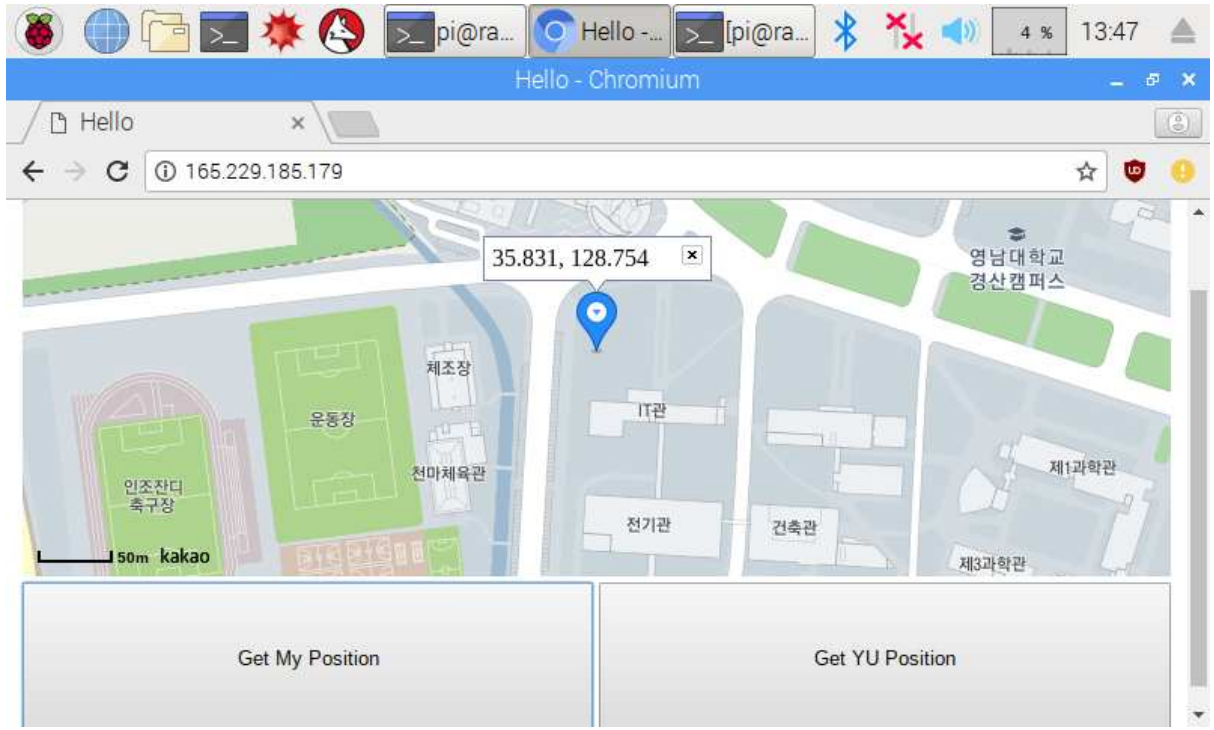
- Messages have a maximum length of 82 characters
- Message starts with $ or !, and ends with <LF>
- The first five characters following the start character ($ or !) identify the talker (two characters) and the type of message (three characters).
- All data fields that follow are comma-delimited.
- Provided data:
  $GPGGA, UTC Time, Latitude, N(north)/S(south), Longitude, W(west) / E(east), ….
- Example
  $GPGGA,092750.000,5321.6802,N,00630.3372,W,1,8,1.03,61.7,M,55.2,M,,*76

  UTC Time　　위도(latitude)　　경도(longitude)

GNSS Application

◆ GNSS를 이용해 현재 위치를 지도에 표시

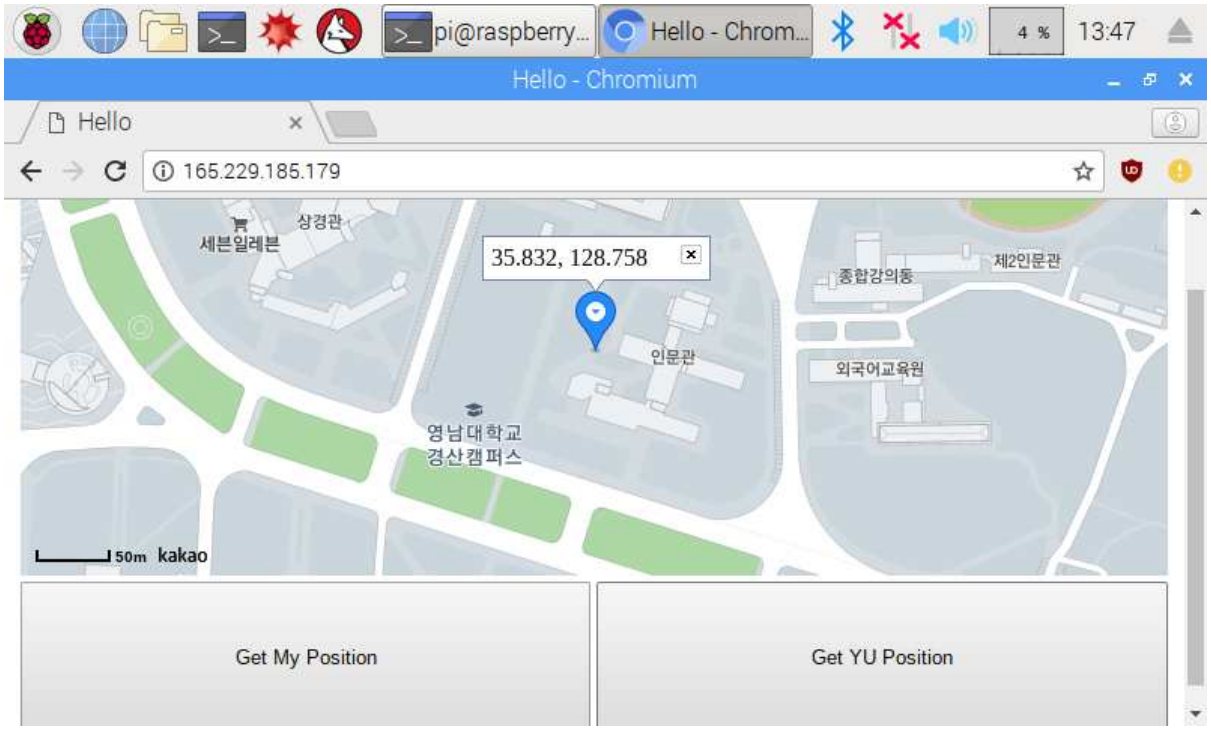
- Using KAKAO Map
  - Get My Position 클릭하면 현재 위치 표시



GNSS Application

◆ GPS를 이용해 주어진 좌표 위치를 지도에 표시

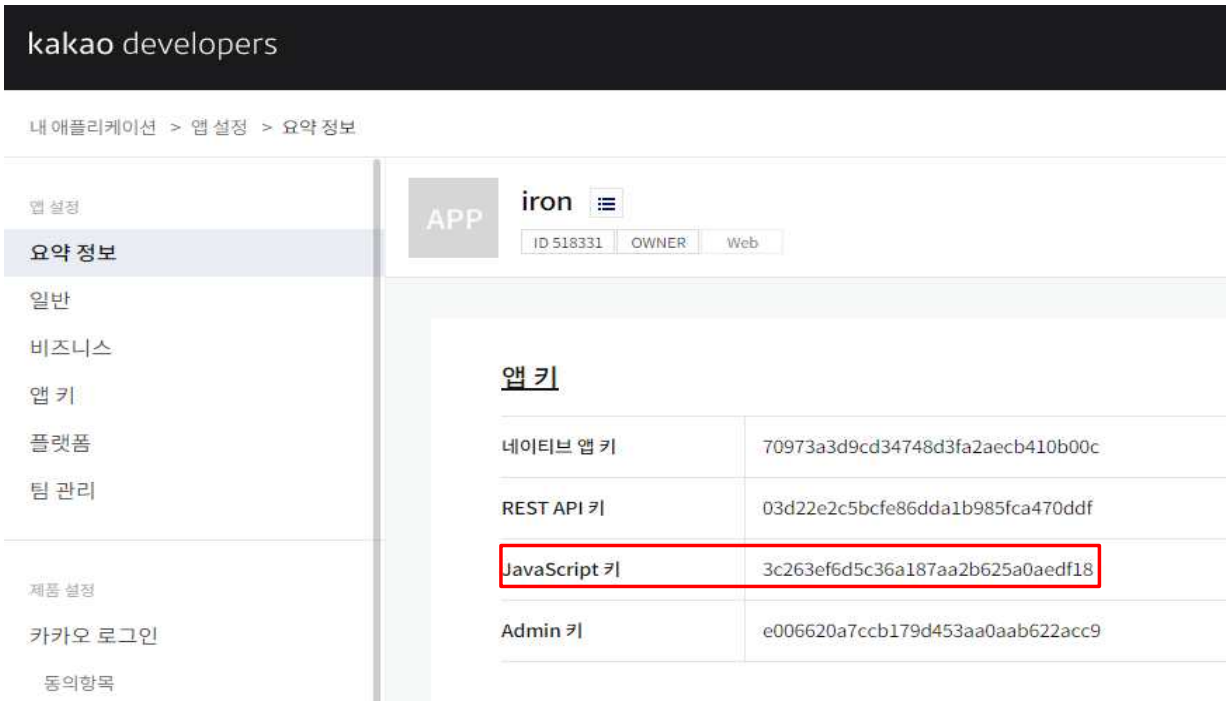
- 영남대학교의 좌표 YU Position (35.832, 128.785)의 위치를 지도상에 표시



GNSS Application

◆ 카카오 개발자 포털에서 API 키 값 받기

- https://developers.kakao.com/docs/latest/ko/getting-started/app



GNSS Application

```
# gps.py
import serial
import time
import threading

def GPS_Init ():
    global running
    running = True
    threading.Thread (target = GPS_Thread, daemon = True).start ()

def GPS_Thread ():
    global gps_serial, running, latitude, longitude
    gps_serial = serial.Serial ("/dev/ttyUSB0", 9600)
    while running:
        gps_msg = gps_serial.readline ()
         #example gps_msg: $GPGGA,092750.000,5321.6802,N,00630.3372,W,
         #                          1,8,1.03,61.7,M,55.2,M,,*76
         # ($GPGGA, UTC Time, Latitude, N(north)/S(south), Longitude, W(west) / E(east), ....)
        if (gps_msg[:6] == b"$GPGGA"):
            rmc_msg = gps_msg[6:].decode ().replace ("\r\n", "").split (",")
        if (rmc_msg[2] != ""):
                latitude = float (rmc_msg[2])
            if (rmc_msg[4] != ""):
                longitude = float (rmc_msg[4])
            #latitude = 3549.932
            #longitude = 12845.456
        time.sleep (0.1)
```

GNSS Application

```
def GPS_GetPosition ():
    global latitude, longitude
    tmp_lat = int (latitude / 100) # 위도의 도, 분, 초 단위 표현에서 도
    minute = (latitude - tmp_lat * 100) / 60 #위도의 도, 분, 초 단위 표현에서 분
    lat = tmp_lat + minute
    tmp_lng = int (longitude / 100) # 경도의 도, 분, 초 단위 표현에서 도
    minute = (longitude - tmp_lng * 100) / 60 # 경도의 도, 분, 초 단위 표현에서 분
    lng = tmp_lng + minute
    return_msg = "{:.3f},{:.3f}".format (lat, lng)
    print (return_msg)
    return return_msg

def GPS_GetPosition_YU (): # position of Yeungnam University
    return_msg = "{:.3f},{:.3f}".format (35.832, 128.758)
    print (return_msg)
    return return_msg

def GPS_Close ():
    global gps_serial, running
    running = False
    gps_serial.close ()
```

GNSS Application

```
# server.py
from bottle import route, run, get, post, response, static_file, request
import gps

gps.GPS_Init ()

@route ("/")
def index ():
    return static_file ("index.html", root = ".")

@route ("/req", method = "POST")
def req ():
    cmd = request.forms.get ("req")
    if (cmd == "my_pos"):
        print ("request: my_pos")
        pos = gps.GPS_GetPosition ()
        return pos
    elif (cmd == "yu_pos"):
        print ("request: yu_pos")
        pos = gps.GPS_GetPosition_YU ()
        return pos
try:
    run (host = "165.229.185.179", port = 80, server = "paste")
finally:
    gps.GPS_Close ()
```

```
<!--index.html-->
<!DOCTYPE html>
<html>
   <head>
      <title>
         GNSS Application with Kakao Map
      </title>
      <meta charset = "utf-8"/>
      <script type = "text/javascript"
          src = "//dapi.kakao.com/v2/maps/sdk.js?appkey=developer's app key"></script>
   </head>
   <body style = "text-align:center;">
      <table style = "width:100%;">
         <tr>
            <td colspan = "2">
               <div id = "map" style = "width:100%;height:300px"></div>
            </td>
         </tr>
         <tr>
            <td>
               <input style = "width:100%;height:100px;"
                   type = "button" value = "Get My Position" onClick = "MarkPosition_Current ()">
            </td>
            <td>
               <input style = "width:100%;height:100px;"
                    type = "button" value = "Get YU Position" onClick = "MarkPosition_YU ()">
            </td>
         </tr>
      </table>
      <script type = "text/javascript">
var container = document.getElementById ("map");
var options = {
   center: new kakao.maps.LatLng (33.450701, 126.570667),
   level: 3
};
```

```
var map = new kakao.maps.Map (container, options);
var markerPosition = new kakao.maps.LatLng (33.450701, 126.570667);
var marker = new kakao.maps.Marker ({
   position: markerPosition
});

marker.setMap (map);
var infowindow = new kakao.maps.InfoWindow ({
   removable: true
});

infowindow.open (map, marker);

function MarkPosition_Current ()
{
   var request = new XMLHttpRequest ();
   var cmd = "req=my_pos";
   request.open ("POST", "/req", false);
   request.setRequestHeader ("Content-Type", "application/x-www-form-urlencoded");
   request.send (cmd); # send command (requesting my_pos) to web server, and get response
   var response = request.responseText;
   var latlng = response.split (",");
   var lat = parseFloat (latlng[0]);
   var lng = parseFloat (latlng[1]);
   var new_latlng = new kakao.maps.LatLng (lat, lng);
   map.setCenter (new_latlng);
   marker.setPosition (new_latlng);
   var iwContent = "<div style=\"padding:5px;text-align:center;\">" + lat + ", " + lng + "</div>";
   var iwPosition = new_latlng;
   infowindow.setContent (iwContent);
   infowindow.setPosition (iwPosition);
}
```

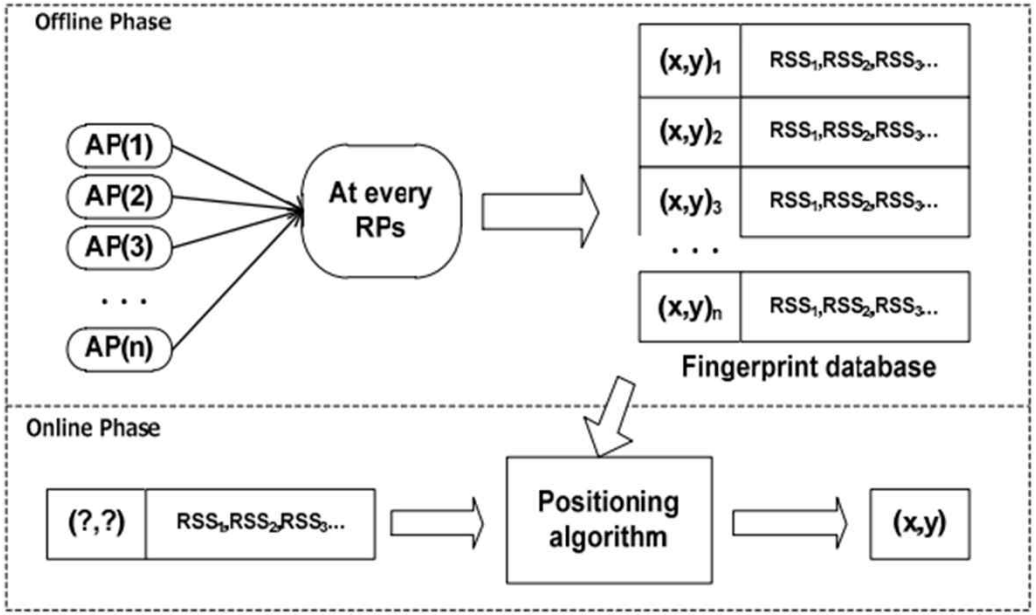
```
function MarkPosition_YU ()
{
   var request = new XMLHttpRequest ();
   var cmd = "req=yu_pos";
   request.open ("POST", "/req", false);
   request.setRequestHeader ("Content-Type", "application/x-www-form-urlencoded");
   request.send (cmd);
   var response = request.responseText;
   var lat_lng = response.split (",");
   var lat = parseFloat (lat_lng[0]);
   var lng = parseFloat (lat_lng[1]);
   var new_lat_lng = new kakao.maps.LatLng (lat, lng);
   map.setCenter (new_lat_lng);
   marker.setPosition (new_lat_lng);
   var iwContent = "<div style=\"padding:5px;text-align:center;\">" + lat + ", " + lng + "</div>";
   var iwPosition = new_lat_lng;
   infowindow.setContent (iwContent);
   infowindow.setPosition (iwPosition);
}
      </script>
   </body>
</html>
```

실내 측위 (Indoor Localization)

WiFi Fingerprinting Indoor Localization

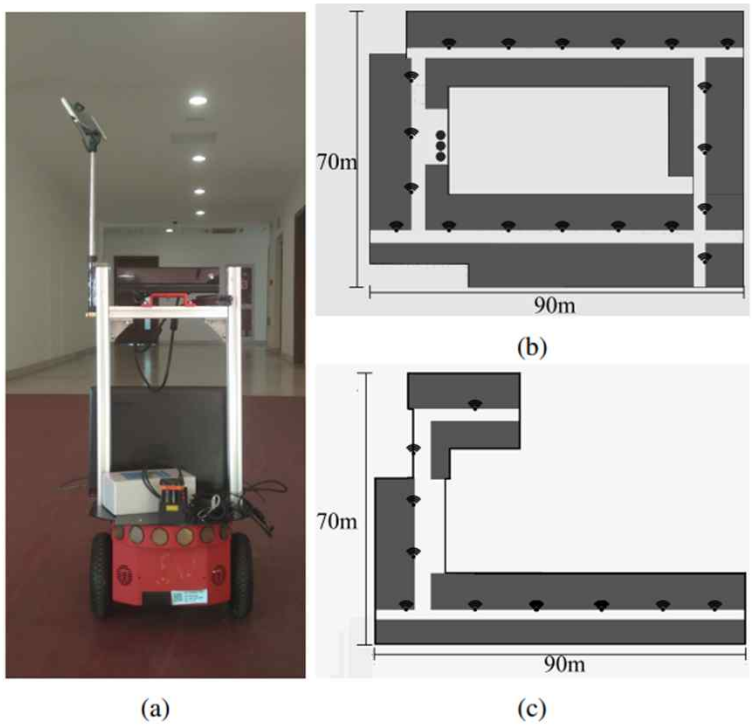
◆ WiFi Fingerprinting 기반 실내 측위

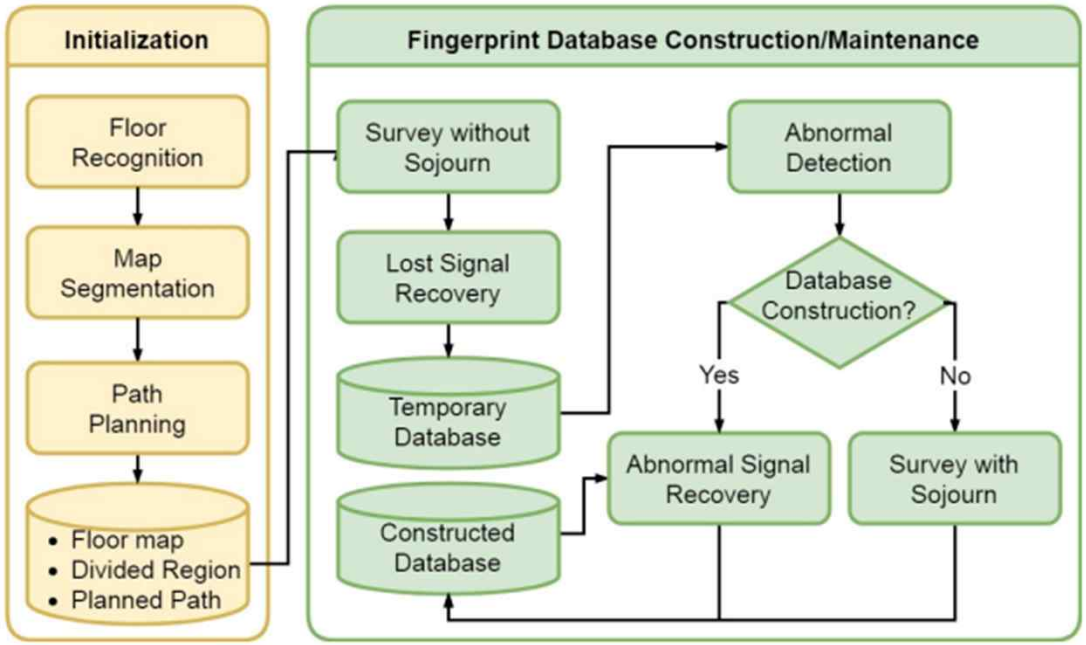
- 건물 내에 설치된 각 AP의 정확한 위치는 사전에 파악
- 사전에 건물내의 다양한 지점에서 각 AP로 부터의 신호 세기를 실제 측정하고, 측정된 값을 기반으로 map 구성
- 다수의 AP로 부터의 신호 세기를 사전에 작성된 map과 비교하여 현재의 실내 위치를 추정



WiFi Fingerprinting Indoor Localization

◆ WiFi Fingerprint Map 생성

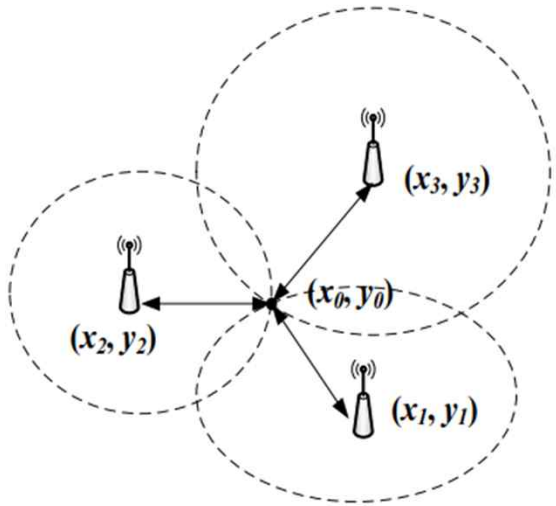




WiFi Fingerprinting Indoor Localization 기술 분류

◆ RSS (Received Signal Strength) 기반

- trilateration (삼각측량)
- approximate perception
- scene analysis



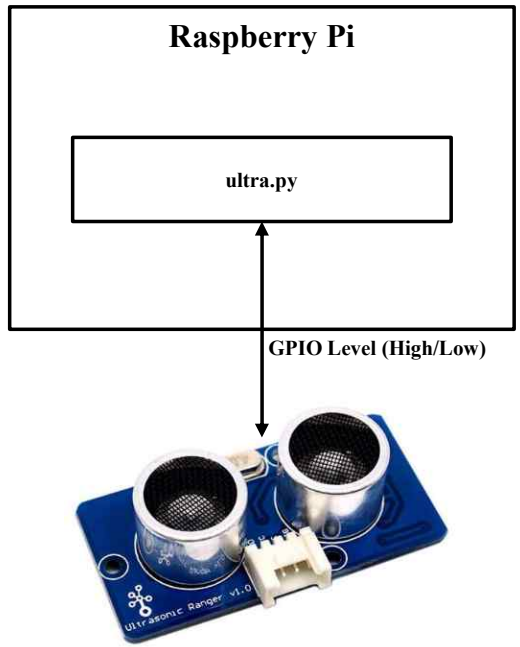
◆ TSARS (Time and Space Attribute of Received Signal) 기반

- AOA (Angle of Arrival)
- TOA (Time of Arrival)
- TDOA (time difference of arrival)

Ultrasonic Ranger

◆ Ultrasonic Ranger

- Ultrasonic ranger is a non-contact distance measurement module which works at 40KHz.
- When we provide a pulse trigger signal with more than 10uS through signal pin, the Grove_Ultrasonic _Ranger will issue 8 cycles of 40kHz cycle level and detect the echo.
- The pulse width of the echo signal is proportional to the measured distance.
- formula:
  Distance = echo signal high time * Sound speed (340M/S)/2.
- Measuring range: 2 ~ 350cm



Ultrasonic Ranger

◆ 초음파 센서로 거리 측정

```
7.26cm
6.76cm
6.60cm
7.08cm
7.12cm
6.56cm
6.94cm
5.61cm
6.44cm
7.08cm
7.19cm
6.97cm
6.11cm
7.56cm
7.59cm
7.61cm
7.22cm
6.82cm
6.52cm
7.05cm
6.90cm
7.24cm
7.33cm
7.17cm
```

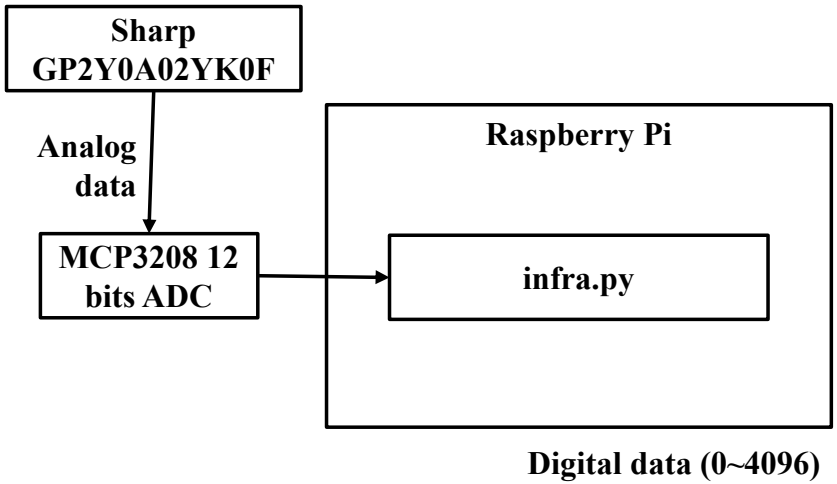
Ultrasonic Ranger

◆ Source code for ultrasonic ranger

```
# ultra.py
import RPi.GPIO as GPIO
import time

TRIG = 8
ECHO = 10

GPIO.setmode (GPIO.BOARD)
GPIO.setup (TRIG, GPIO.OUT)
GPIO.setup (ECHO, GPIO.IN)

start, end = 0, 0

def echo_action (channel):
    global start, end
    if (GPIO.input (ECHO) == GPIO.HIGH):
        start = time.time ()
    else:
        end = time.time ()

GPIO.add_event_detect (ECHO, GPIO.BOTH, callback = echo_action)

while True:
    GPIO.output (TRIG, GPIO.HIGH)
    GPIO.output (TRIG, GPIO.LOW)

    elapsed = end - start
    if (elapsed < 0.1):
        dist = elapsed * 340 * 100 / 2 # round-trip -> uni-directional distance in cm unit
        print ("{:.2f}cm".format (dist))

    time.sleep (0.1)
```

Infrared Proximity Ranger

◆ Block diagram

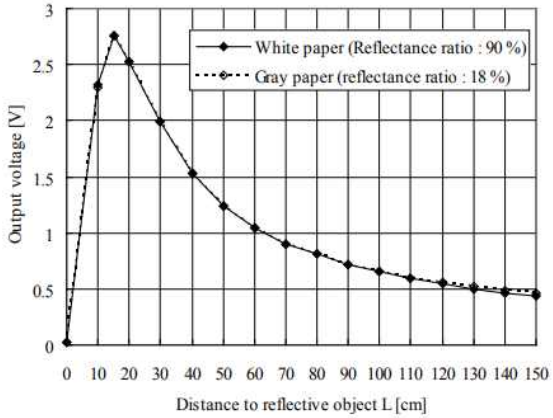


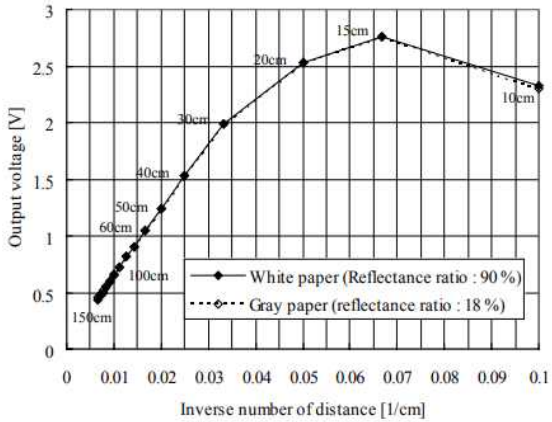
Infrared Proximity Sensor

◆ 적외선 센서

- 짧은 거리 탐지 가능

Distance Measuring Sensor Unit
Measuring distance: 20 to 150 cm
Analog output type

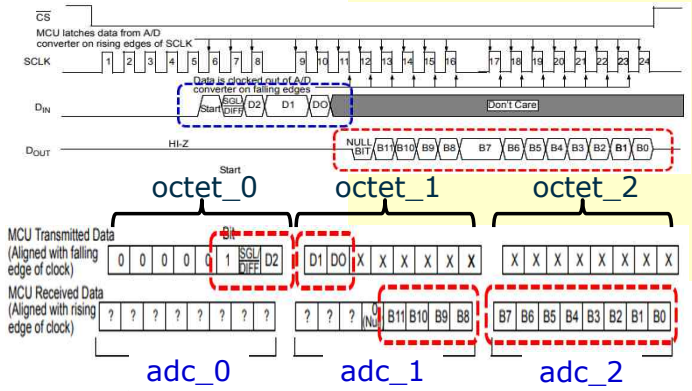




Infrared Proximity Sensor

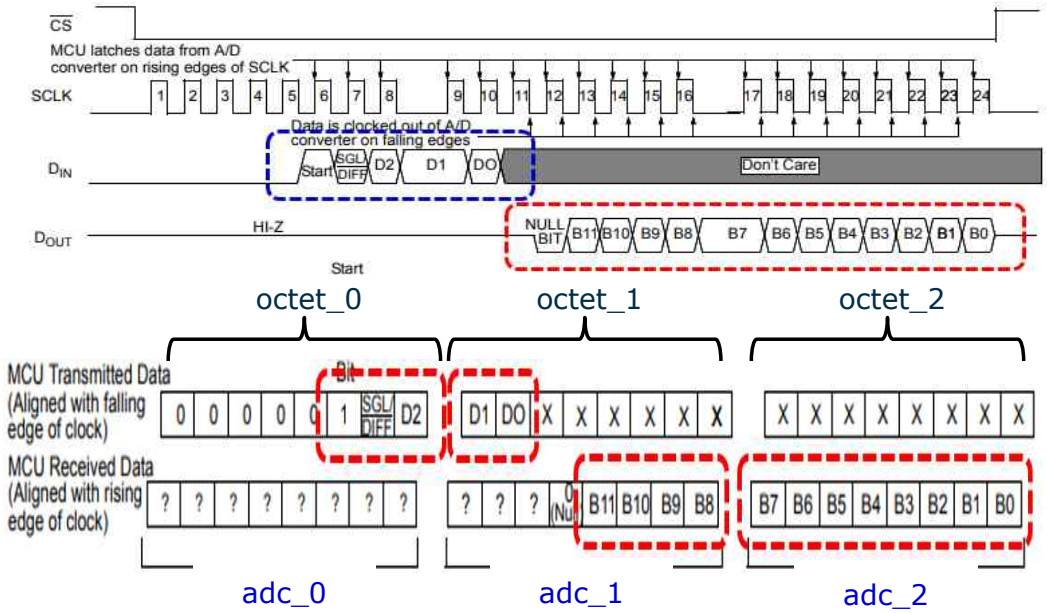
```
# infrared.py (1)
import spidev
import time
import RPi.GPIO as GPIO

spi = spidev.SpiDev ()
spi.open (0, 0)
spi.max_speed_hz = 500000

MCP3208_CS_PIN = 13

GPIO.setmode (GPIO.BOARD)
GPIO.setup (MCP3208_CS_PIN, GPIO.OUT)

def MCP3208_read (channel):
    chD2 = (channel & 0x04) >> 2
    chD1 = (channel & 0x02) >> 1
    chD0 = (channel & 0x01)

    octet_0  = 0x01 << 2 # MCP3208_SB
    octet_0 |= 0x01 << 1 # MCP3208_SD
    octet_0 |= chD2 << 0 # MCP3208_D2
```

```
# infrared.py (2)

    octet_1  = chD1 << 7 # MCP3208_D1
    octet_1 |= chD0 << 6 # MCP3208_D0

    octet_2 = 0x00;
    octets = [octet_0, octet_1, octet_2]

    GPIO.output (MCP3208_CS_PIN, GPIO.LOW)
    adc_0, adc_1, adc_2 = spi.xfer (octets)
    GPIO.output (MCP3208_CS_PIN, GPIO.HIGH)

    adc  = adc_2;
    adc |= (adc_1 & 0x0F) << 0x08

    return adc
```



◆ Source code

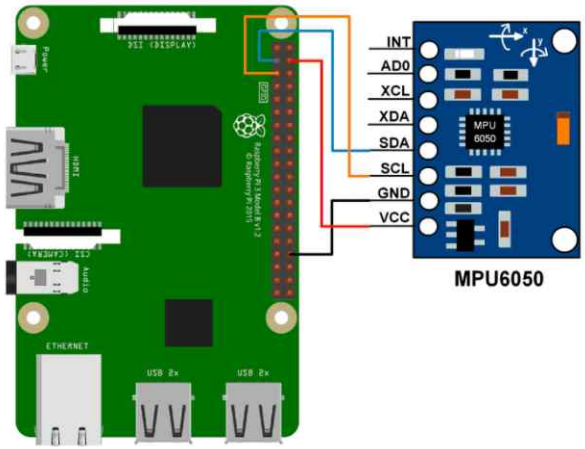
```
# infrared.py (3)

def get_distance (value):
    if (value > 2.6):
        return value * 3.95
    else:
        return value * 4.35


while True:
    value = (MCP3208_read (0) * 5) / 4096
    print ("{:.2f}V, {:.2f}cm"\
        .format (value, get_distance (value)))
    time.sleep (0.1)
```



6-Axis Gyro Accelerator Sensor Interface

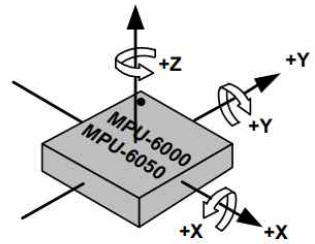
◆ MPU6050 (Accelerometer + Gyroscope)

- MPU6050 sensor module is an integrated 6-axis Motion tracking device.
- It has a **3-axis Gyroscope**, **3-axis Accelerometer**, Digital Motion Processor and a Temperature sensor, all in a single IC.
- It can accept inputs from other sensors like 3-axis magnetometer or pressure sensor using its Auxiliary I2C bus.
- If external 3-axis magnetometer is connected, it can provide complete 9-axis Motion Fusion output.
- A microcontroller can communicate with this module using **I2C communication protocol**. Various parameters can be found by reading values from addresses of certain registers using I2C communication.
- Gyroscope and accelerometer reading along X, Y and Z axes are available in 2's complement form.
- Gyroscope readings are in degrees per second (dps) unit; Accelerometer readings are in g unit.



MPU6050

MPU6050 Interfacing with Raspberry Pi



Orientation of Axes of Sensitivity and Polarity of Rotation

MPU6050 (Accelerometer+Gyroscope) Interfacing with Raspberry Pi

◆ MPU6050 (Accelerometer+Gyroscope) Interfacing with Raspberry Pi

- Source: https://www.electronicwings.com/raspberry-pi/mpu6050-accelerometergyroscope-interfacing-with-raspberry-pi

```
# Read Gyro and Accelerometer by Interfacing Raspberry Pi with MPU6050 using Python (1)

import smbus  #import SMBus module of I2C
from time import sleep #import

#MPU6050 Registers and their Address
PWR_MGMT_1 = 0x6B
SMPLRT_DIV = 0x19
CONFIG = 0x1A
GYRO_CONFIG = 0x1B
INT_ENABLE = 0x38
ACCEL_XOUT_H = 0x3B
ACCEL_YOUT_H = 0x3D
ACCEL_ZOUT_H = 0x3F
GYRO_XOUT_H = 0x43
GYRO_YOUT_H = 0x45
GYRO_ZOUT_H = 0x47
```

```
# Read Gyro and Accelerometer by Interfacing Raspberry Pi with MPU6050 using Python (2)

def MPU_Init():
    # write to sample rate register
    bus.write_byte_data(Device_Address, SMPLRT_DIV, 7)
    # Write to power management register
    bus.write_byte_data(Device_Address, PWR_MGMT_1, 1)
    #Write to Configuration register
    bus.write_byte_data(Device_Address, CONFIG, 0)
    #Write to Gyro configuration register
    bus.write_byte_data(Device_Address, GYRO_CONFIG, 24)
    #Write to interrupt enable register
    bus.write_byte_data(Device_Address, INT_ENABLE, 1)

def read_raw_data(addr):
    #Accelero and Gyro value are 16-bit
    high = bus.read_byte_data(Device_Address, addr)
    low = bus.read_byte_data(Device_Address, addr+1)

    #concatenate higher and lower value
    value = ((high << 8) | low)

    #to get signed value from mpu6050
    if(value > 32768):
        value = value - 65536
    return value
```

Read Gyro and Accelerometer by Interfacing Raspberry Pi with MPU6050 using Python (3)

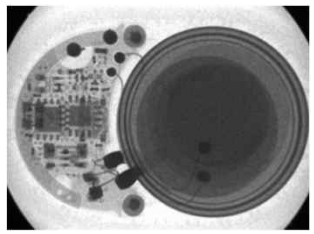
```
MPU_Init()

print (" Reading Data of Gyroscope and Accelerometer")

while True:

    #Read Accelerometer raw value
    acc_x = read_raw_data(ACCEL_XOUT_H)
    acc_y = read_raw_data(ACCEL_YOUT_H)
    acc_z = read_raw_data(ACCEL_ZOUT_H)

    #Read Gyroscope raw value
    gyro_x = read_raw_data(GYRO_XOUT_H)
    gyro_y = read_raw_data(GYRO_YOUT_H)
    gyro_z = read_raw_data(GYRO_ZOUT_H)

    #Full scale range +/- 250 degree/C as per sensitivity scale factor
    Ax = acc_x/16384.0
    Ay = acc_y/16384.0
    Az = acc_z/16384.0

    Gx = gyro_x/131.0
    Gy = gyro_y/131.0
    Gz = gyro_z/131.0


    print ("Gx=%.2f"%Gx, u'\u00b0'+ "/s", "\tGy=%.2f" %Gy, u'\u00b0'+ "/s", "\tGz=%.2f" %Gz,\
           u'\u00b0'+ "/s", "\tAx=%.2f g" %Ax, "\tAy=%.2f g" %Ay, "\tAz=%.2f g" %Az)
    sleep(1)
```

```
Gx=-0.305 °/s  Gy=0.099 °/s   Gz=-0.038 °/s  Ax=-0.059 g   Ay=-0.026 g   Az=0.945 g
Gx=-0.328 °/s  Gy=0.130 °/s   Gz=-0.015 °/s  Ax=-0.060 g   Ay=-0.032 g   Az=0.947 g
Gx=-0.359 °/s  Gy=0.153 °/s   Gz=-0.015 °/s  Ax=-0.074 g   Ay=-0.018 g   Az=0.951 g
Gx=-0.344 °/s  Gy=0.092 °/s   Gz=-0.015 °/s  Ax=-0.057 g   Ay=-0.030 g   Az=0.946 g
Gx=-0.214 °/s  Gy=-0.053 °/s  Gz=0.000 °/s   Ax=-0.061 g   Ay=-0.027 g   Az=0.965 g
Gx=-0.305 °/s  Gy=0.084 °/s   Gz=-0.107 °/s  Ax=-0.048 g   Ay=-0.024 g   Az=0.939 g
Gx=-0.290 °/s  Gy=0.092 °/s   Gz=-0.069 °/s  Ax=-0.061 g   Ay=-0.041 g   Az=0.938 g
Gx=-0.298 °/s  Gy=0.107 °/s   Gz=-0.053 °/s  Ax=-0.070 g   Ay=-0.031 g   Az=0.947 g
Gx=-0.305 °/s  Gy=0.107 °/s   Gz=-0.084 °/s  Ax=-0.053 g   Ay=-0.016 g   Az=0.951 g
Gx=-0.321 °/s  Gy=0.099 °/s   Gz=-0.053 °/s  Ax=-0.070 g   Ay=-0.027 g   Az=0.931 g
Gx=-0.305 °/s  Gy=0.107 °/s   Gz=-0.053 °/s  Ax=-0.063 g   Ay=-0.041 g   Az=0.948 g
Gx=-0.244 °/s  Gy=0.076 °/s   Gz=-0.076 °/s  Ax=-0.058 g   Ay=-0.037 g   Az=0.969 g
Gx=-0.290 °/s  Gy=0.107 °/s   Gz=-0.053 °/s  Ax=-0.056 g   Ay=-0.027 g   Az=0.946 g
Gx=-0.305 °/s  Gy=0.099 °/s   Gz=-0.053 °/s  Ax=-0.064 g   Ay=-0.030 g   Az=0.952 g
Gx=-0.305 °/s  Gy=0.099 °/s   Gz=-0.038 °/s  Ax=-0.060 g   Ay=-0.028 g   Az=0.939 g
Gx=-0.313 °/s  Gy=0.115 °/s   Gz=-0.046 °/s  Ax=-0.065 g   Ay=-0.032 g   Az=0.950 g
```

가속도 측정을 사용하여 이동 거리 환산

◆ 가속도 센서를 이용한 이동거리 측정

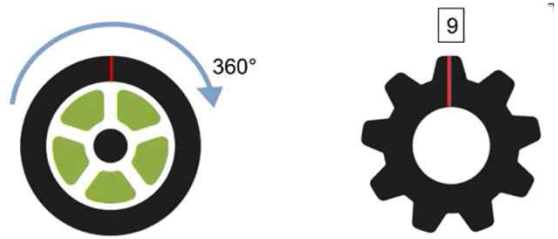
- 가속도 적분 -> 속도
- 속도 x 시간 -> 이동 거리
  - GPS 신호가 도달하지 않는 터널 구간에서의 이동 거리 계산
- 가속도 센서 기반 운동량 측정 시스템
  - Nike+ Shoe



거리 측정을 위한 Encoder, Odometer

◆ Encoder

- 차량 바퀴의 회전 수 및 회전 각도를 측정할 수 있도록 tick수를 제공
- 발생된 ticks를 사용하여 차량 바퀴 회전 수 계산

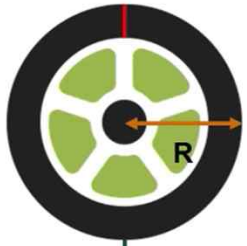


$$1\ rotation = 9\ ticks$$

$$Number\ of\ Wheel\ Rotations = \frac{Total\ Encoder\ Ticks}{Tick\ count\ per\ rotation}$$

◆ Odometer (주행거리계)

- 차량 바퀴의 반지름과 ticks를 기반으로 이동 거리를 계산



$$1\ rotation = 1\ circumference\ distance = 2 * \pi * R$$

$$Distance\ Traveled = Number\ of\ Wheel\ Rotations * 2 * \pi * R$$

$$Distance\ Traveled = \frac{Total\ Encoder\ Ticks}{Tick\ count\ per\ rotation} * 2 * \pi * R$$

LiDAR (Light Detection and Ranging)

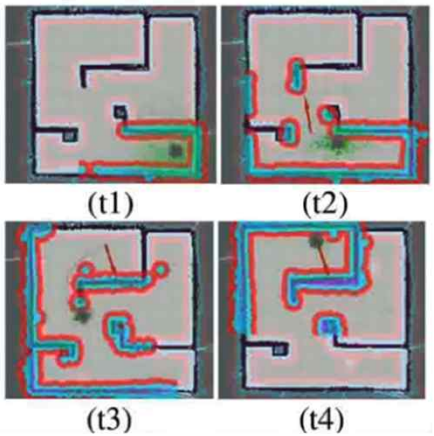
자율 주행에 필요한 핵심 기술 요소

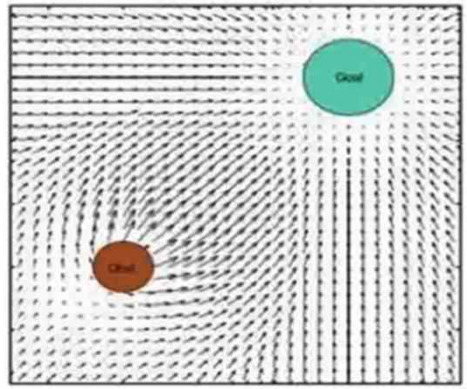
◆ 자율 주행에 필요한 핵심 기술 요소

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

◆ 이동 경로 탐색/계획 알고리즘

- Dynamic Window Approach (DWA)
- A* 알고리즘
- Potential Field
- Particle Filter
- Graph





Sensing, Localization, Mapping, Navigation

- **센서를 이용한 측정 (Sensing)**
  - 2D/3D 거리 센서를 사용한 거리 측정
- **위치 파악 (Localization)**
  - 현재 위치를 파악
- **지도 제작 (Mapping)**
  - 측정된 거리 정보를 사용하여 지도 제작
- **경로 탐색/계획(Navigation)**
  - 작성된 지도를 기반으로 출발지에서 목적지까지의 최적 경로를 탐색/계획

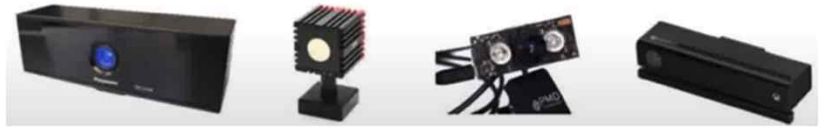
2차원 및 3차원 거리 측정 센서

◆ 거리 센서

- LRF(Laser Range Finder)
- 초음파 센서
- 적외선 거리 센서(PSD, position sensitive detector)

◆ 비전 센서

- stereo camera, mono camera, omni-directional camera



◆ Depth Camera

- SwissRanger, Kinect-2
- Kinect, Xtion, Carmine(PrimeSense), Astra
- Intel® RealSense™ Depth Camera D435i

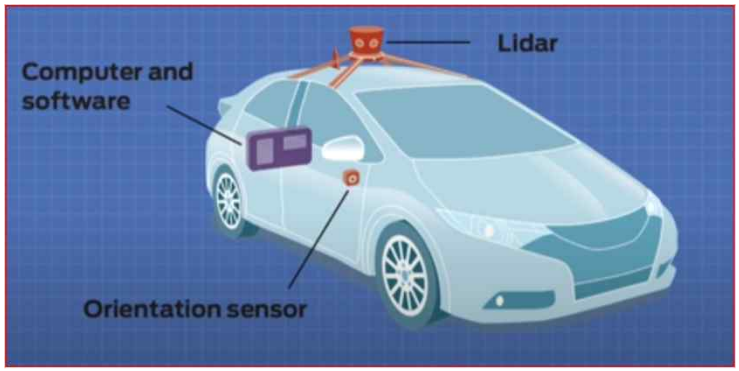
차량에 설치되는 LiDAR와 RADAR

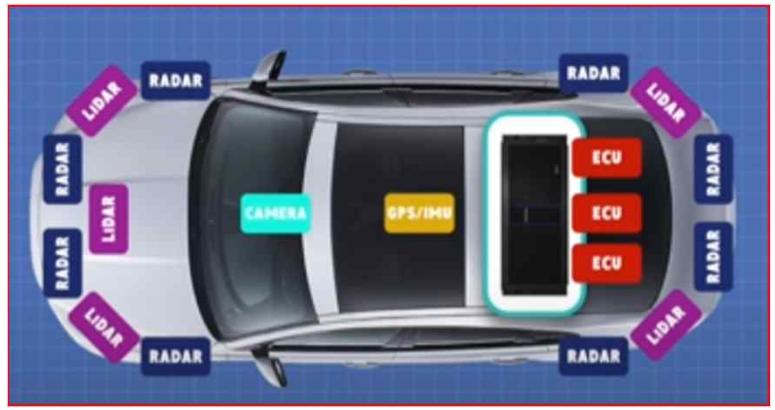
◆ LiDAR (Light Detection and Ranging)

- 중, 단거리 2D/3D 탐색 및 거리 측정
- 주로 200미터 이내의 중, 단거리 차량과 장애물 위치 탐지

◆ RADAR(Radio Detection and Ranging)

- 77GHz 주파수 방식을 사용하여 200 meter 이상의 원거리 차량과 장애물 위치 탐지





TF Mini-LiDAR

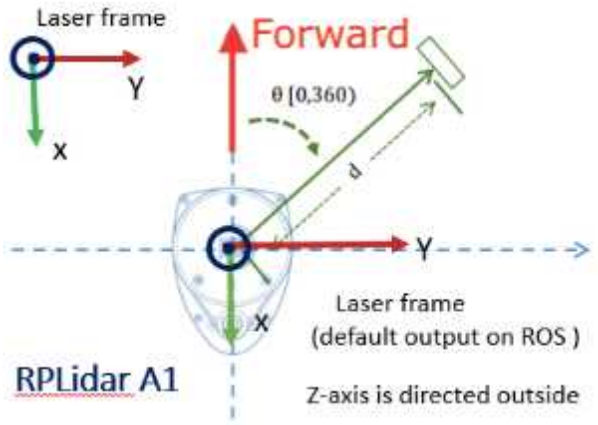
◆ TF Mini-LiDAR

- low cost stationary LiDAR device
- uses focused IR LED instead of laser
- range from 30 cm ~ 12meter
- internal processor with serial input/output
- maximum sample rates of 100 Hz
- accuracy of 5mm

2D LiDAR Sensor

◆ 2D LiDAR

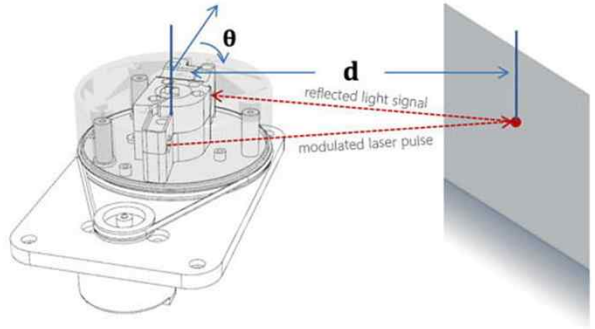
- Light Detection and Ranging
- also, Light Imaging, Detection and Ranging
- uses focused light and sensor to detect range and reflectivity
- used in survey applications to create high-resolution maps
- used in self-driving vehicles and robotics

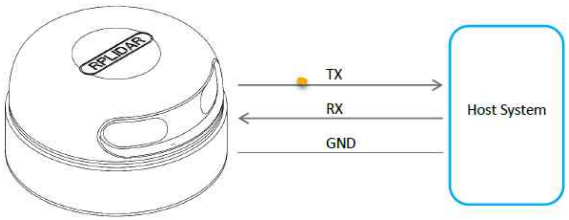


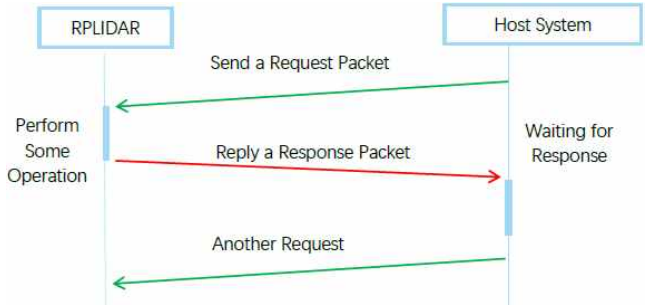
Robopeak RP LiDAR

◆ Robopeak RP LiDAR (http://wiki.ros.org/rplidar)

- 1차원 레이져 센서를 회전시키면서 회전각도별 거리를 스캔하여 2차원 데이터 생성
- 회전 속도는 모터 속도로 조정 (PWM control)
- 샘플링 속도를 소프트웨어로 조정
- 샘플링 속도는 스캔 속도와 정확도가 달라짐
- microUSB adapter를 사용하여 컴퓨터 접속
- 참고자료: https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=thumbdown&logNo=220385363246



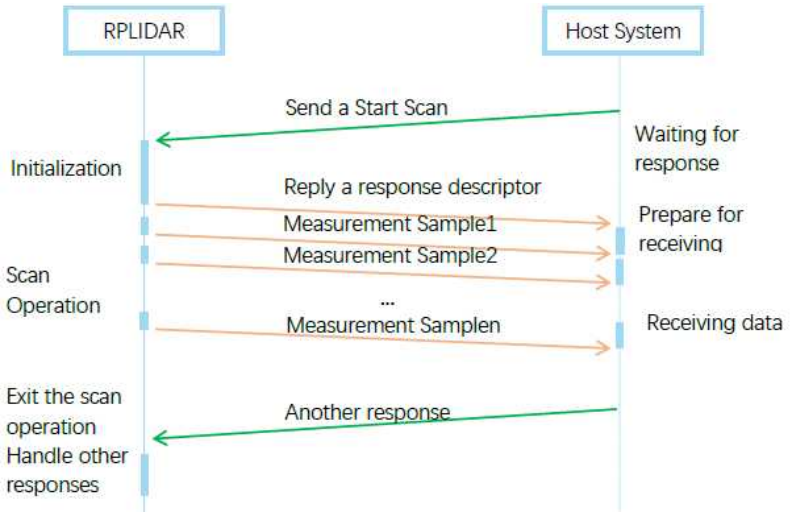




RP Lidar의 통신 프로토콜

◆ 단일 요청 / 다중 응답

- 스캔 작업 수행 요청 시에 사용
- RPLidar는 연속적으로 거리 스캔 측정 수행
- 결과 데이터 (거리, 각도)가 개별 응답 패킷으로 전송



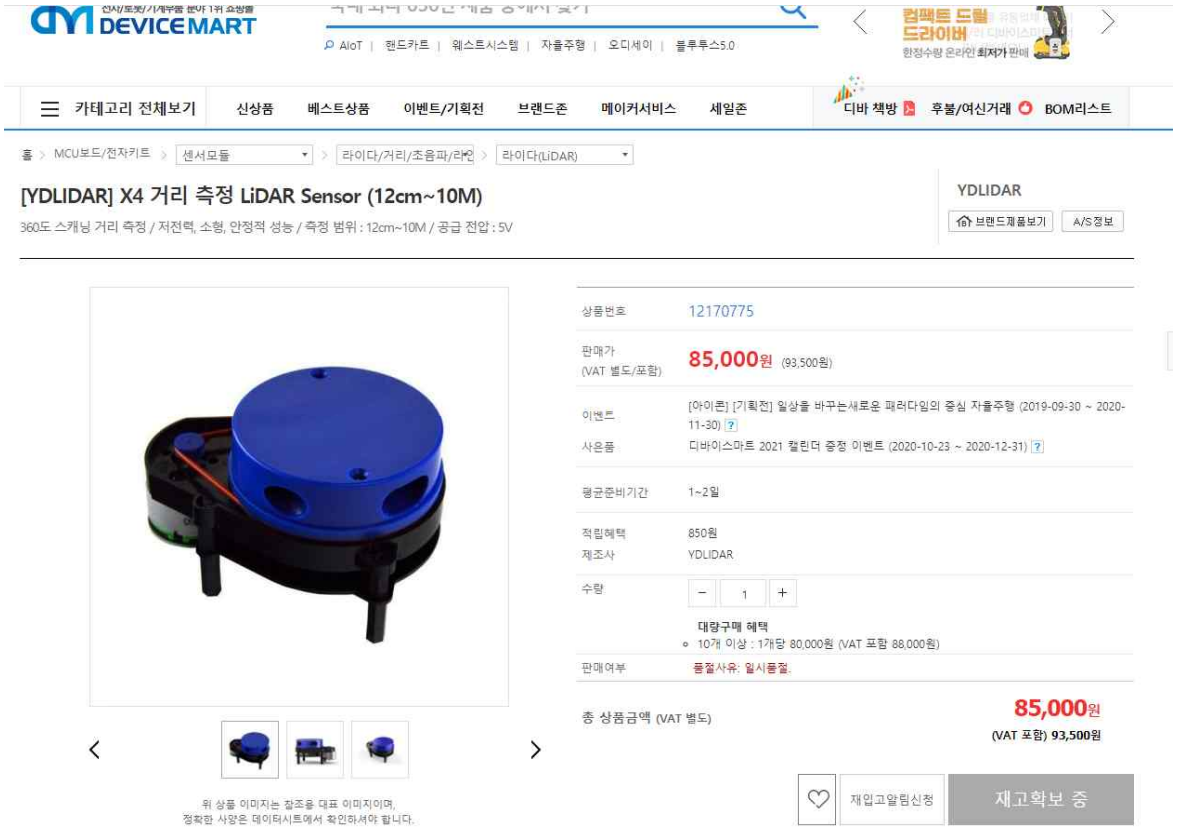
3D LiDAR

◆ 3D LiDAR

- Ouster
- Velodyne Velabit, USD 100 (KRW 116,000)
- SICK MRS1000 3D-LiDAR
- Speeed Studio RPLiDAR S1 Portable ToF Laser Scanner Kit 40M range, KRW 703,225

YD LiDAR

◆ YD LiDAR, Shenzhen EAI Technology



YD LiDAR 기능 시험 – PyLidar3

◆ PyLidar3

- 참고자료:
  - https://package.wiki/PyLidar3
  - PyLidar3, https://github.com/lakshmanmallidi/PyLidar3
  - Interfacing LiDAR using Python, https://www.youtube.com/watch?v=dR2XIwRIseY

◆ YDLiDAR X4

- YDLiDAR X4 Development Manual, https://www.ydlidar.com/Public/upload/files/2021-09-08/YDLIDAR%20X4%20development%20manual%20V1.6.pdf

◆ YDLiDAR G4

- low power mode 기능 추가
- YDLiDAR G4 Development Manual, https://www.ydlidar.com/Public/upload/files/2021-08-24/YDLIDAR%20G4%20development%20manual%20V2.0.pdf

YDLiDAR X4 (G4) System Command

System Command		Description	Mode Switching	Answer Mode
0xA5 (Start)	0x60	start scanning, output point cloud data	scan mode	sustained response
	0x65	stop, stop scanning	stop mode	no answer
	0x90	get device information (model, firmware, hardware version)	No	single response
	0x91 0x92	get device health status (X4) get device health status (G4)	No	single response
	0x09	increase the current scan frequency of 0.1Hz	No	single response
	0x0A	reduce the current scan frequency of 0.1Hz	No	single response
	0x0B	increase the current scan frequency of 1Hz	No	single response
	0x0C	reduce the current scan frequency of 1Hz	No	single response
	0x0D	get the currently scan frequency	No	single response
	0xD1	get the currently set ranging frequency	No	single response
	0xD9	power-down protection mode switch (off by default)	No	single response
	0x80 0x40	soft restart (X4) soft restart (G4)	/	no answer

YD LiDAR System Message Data Protocol

◆ YD LiDAR System Message Data Protocol

Start sign	Response length	Response mode	Type code	Content
16bits	30bits	2bits	8bits	/

Bite offset:

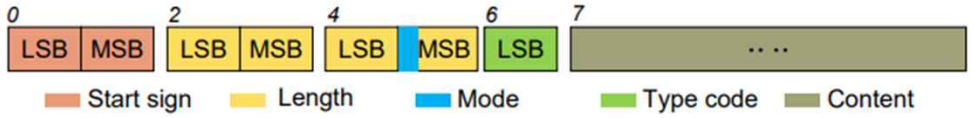


FIG 3 YDLIDAR G4 SYSTEM MESSAGE DATA PROTOCOL

◆ Fixed Code

- start sign : 0xA55A

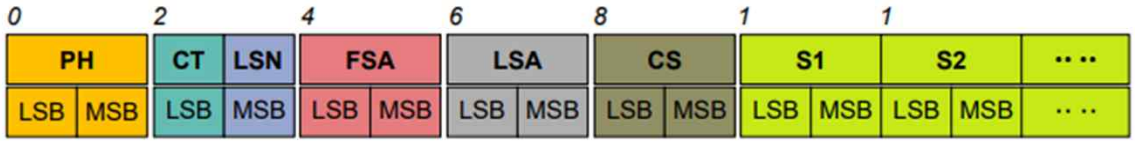
◆ Response Value and Mode

Mode	0x0	0x1	0x2	0x3
Response mode	Single response	continuous response	Undefined	

YD LiDAR Data Protocol

◆ Reply Message of Scan Command

◆ Scan Command Response Content Data Structure



- PH (packet header)
- CT (packet/content type)
- LSN (number of sampling points)
- FSA (first sampling angle)
- LSA (last sampling angle)
- CS (check code)
- S_i (i-th sample); distance = S_i / 4

YD LiDAR Scan Command Response Content Data Structure

Content	Name	Description
PH(2B)	Packet Header	2 bytes in length, fixed at 0x55AA, little endian (low in front, high in back)
CT(1B)	Packet Type	indicates the type of the current packet - 0x00 : point cloud data packet - 0x01 : beginning of data packet
LSN(1B)	Sample Quantity	indicates the number of sampling points contained in the current packet there is only one zero point of data in the zero packet (value = 1)
FSA(2B)	Start Angle	the angle data corresponding to the first sample point in the sampled data
LSA(2B)	End Angle	the angle data corresponding to the last sample point in the sampled data
CS(2B)	Check Code	the check code of the current data packet uses a two-byte exclusive OR to check the current data packet
S_i(2B)	Sample Data	the sampling data of the system test is the distance data of the sampling point distance = S_i / 4

```
# PyLidar3::__init__.py (1)
from serial import Serial
from time import sleep
from math import atan, pi, floor
from enum import Enum

name = "PyLidar3"

class FrequencyStep(Enum):
    oneTenthHertz=1
    oneHertz=2

class YdLidarX4:
    """Deals with X4 version of Ydlidar from http://www.ydlidar.com/"""
    def __init__(self, port, chunk_size=6000, no_value=0):
        """Initialize the connection and set port and baudrate."""
        self._port = port
        self._baudrate = 128000
        self._is_scanning = False
        self._is_connected = False
        self.chunk_size=chunk_size
        self._no_value = no_value

    def Connect(self):
        """Begin serial connection with Lidar by opening serial port.
           Return success status True/False."""
        try:
            if(not self._is_connected):
                self._s=Serial(self._port, self._baudrate)
                self._is_connected = True
                sleep(3)
                self._s.reset_input_buffer()
                if("YdLidarX4" in str(type(self))):
                    self._Stop_motor()
                if(self.GetHealthStatus()):
                    return True
                else:
                    raise Exception('Device status error. Try reconnecting device')
            else:
                raise Exception("Already connected")
        except Exception as e:
            print(e)
            return False
```

```
# PyLidar3::__init__.py (2)

    def _Start_motor(self):
        self._s.setDTR(1)
        sleep(0.5)

    def _Stop_motor(self):
        self._s.setDTR(0)
        sleep(0.5)

    @classmethod
    def _AngleCorr(cls, dist): # angle correction
        if dist == 0:
            return 0
        else:
            return (atan(21.8*((155.3 - dist)/(155.3 * dist)))*(180/pi))

    @classmethod
    def _HexArrToDec(cls, data):
        littleEndianVal = 0
        for i in range(0, len(data)):
            littleEndianVal = littleEndianVal + (data[i] * (256**i))
        return littleEndianVal
```

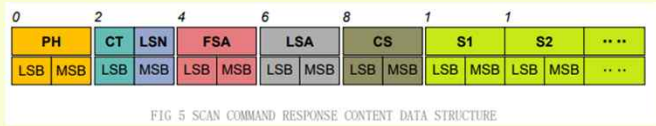
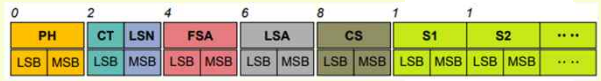


FIG 5 SCAN COMMAND RESPONSE CONTENT DATA STRUCTURE

```
    @classmethod
    def _Calculate(cls, d):
        ddict=[]
        LSN=d[1]
        Angle_fsa = ((YdLidarX4._HexArrToDec((d[2], d[3]))>>1)/64.0) #first sample ang
            #+YdLidarX4._AngleCorr(YdLidarX4._HexArrToDec((d[8], d[9])))/4)
        Angle_lsa = ((YdLidarX4._HexArrToDec((d[4], d[5]))>>1)/64.0) # last sample ang
            #+YdLidarX4._AngleCorr(YdLidarX4._HexArrToDec((d[LSN+6], d[LSN+7])))/4)
        if Angle_fsa < Angle_lsa:
            Angle_diff = Angle_lsa - Angle_fsa
        else:
            Angle_diff = 360 + Angle_lsa - Angle_fsa
```

PyLidar3::__init__.py (3)

```
    for i in range(0, 2*LSN, 2):
        # Distance calculation
        dist_i = YdLidarX4._HexArrToDec((d[8+i],d[8+i+1]))/4
        # Ignore zero values, they result in massive noise when
        #    computing mean of distances for each angle.
        if dist_i == 0:
            continue
        # Intermediate angle solution
        Angle_i_tmp = ((Angle_diff/float(LSN-1))*(i/2)) + Angle_fsa
        # Angle correction
        Angle_i_tmp += YdLidarX4._AngleCorr(dist_i)
        if Angle_i_tmp > 360 :
            Angle_i = Angle_i_tmp - 360
        elif Angle_i_tmp < 0 :
            Angle_i = Angle_i_tmp + 360
        else:
            Angle_i = Angle_i_tmp
        ddict.append((dist_i,Angle_i))
    return ddict

@classmethod
def _CheckSum(cls,data):
    try:
        ocs = YdLidarX4._HexArrToDec((data[6],data[7]))
        LSN = data[1]
        cs = 0x55AA^YdLidarX4._HexArrToDec((data[0],data[1]))\
             ^YdLidarX4._HexArrToDec((data[2],data[3]))
             ^YdLidarX4._HexArrToDec((data[4],data[5]))
        for i in range(0, 2*LSN, 2):
            cs = cs^YdLidarX4._HexArrToDec((data[8+i], data[8+i+1]))
        if(cs == ocs):
            return True
        else:
            return False
    except Exception as e:
        return False

@classmethod
def _Mean(cls,data):
    if(len(data)>0):
        return int(sum(data)/len(data))
    return 0
```



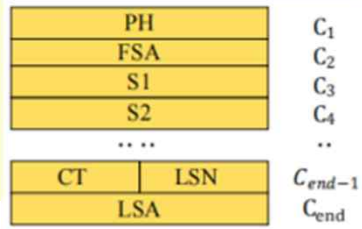


FIG 7 CS XOR SEQUENCE

PyLidar3::__init__.py (4)

```
    def StartScanning(self):
        """Begin the lidar and returns a generator which returns
            a dictionary consisting angle(degrees) and distance(meters).
            \nReturn Format : {angle(1):distance, angle(2):distance,
             .....,angle(360):distance}."""
        if(self._is_connected):
            if(not self._is_scanning):
                self._is_scanning = True
                self._s.reset_input_buffer()
                if("YdLidarX4" in str(type(self))):
                    self._Start_motor()
                self._s.write(b"\xA5\x60") # Start Scanning CMD 0xA560
                sleep(0.5)
                self._s.read(7)
                distdict = {}
                countdict = {}
                lastChunk = None
                while self._is_scanning == True:
                    for i in range(0,360):
                        distdict.update({i:[]})
                    data = self._s.read(self.chunk_size).split(b"\xaa\x55")
                    if lastChunk is not None:
                        data[0] = lastChunk + data[0]
                    lastChunk = data.pop()
                    for e in data:
                        try:
                            if(e[0] == 0):
                                if(YdLidarX4._CheckSum(e)):
                                    d = YdLidarX4._Calculate(e)
                                    for ele in d:
                                        angle = floor(ele[1])
                                        if(angle>=0 and angle<360):
                                            distdict[angle].append(ele[0])
                        except Exception as e:
                            pass
                    for i in distdict.keys():
                        if len(distdict[i]) > 0:
                            distdict[i]=self._Mean(distdict[i])
                        else:
                            distdict[i]=self._no_value
                    yield distdict
            else:
                raise Exception("Device is currently in scanning mode.")
        else:
            raise Exception("Device is not connected")
```

PyLidar3::__init__.py (5)

```
    def StopScanning(self):
        """Stops scanning but keeps serial connection alive."""
        if(self._is_connected):
            if(self._is_scanning):
                self._is_scanning = False
                self._s.write(b"\xA5\x65") # Stop command: 0xA565
                sleep(1)
                self._s.reset_input_buffer()
                if("YdLidarX4" in str(type(self))):
                    self._Stop_motor()
            else:
                raise Exception("Device is not set to scanning mode")
        else:
            raise Exception("Device is not connected")


    def GetHealthStatus(self):
        """Returns Health status of lidar\nTrue: good\nFalse: Not good"""
        if(self._is_connected):
            if self._is_scanning == True:
                self.StopScanning()
            self._s.reset_input_buffer()
            sleep(0.5)
            self._s.write(b"\xA5\x91") # Get HealthStatus Cmd: 0xA591
            sleep(0.5)
            data = self._s.read(10)
            if data[9]==0 and data[8]==0 and (data[7]==0 or data[7]==1):
                return True
            else:
                return False
        else:
            raise Exception("Device is not connected")
```

PyLidar3::__init__.py (6)

```
    def GetDeviceInfo(self):
        """Return device information of lidar in form of dictonary\n
           {"model_number":model_number, "firmware_version":firmware_version,
            "hardware_version":hardware_version,"serial_number":serial_number}"""
        if(self._is_connected):
            if self._is_scanning == True:
                self.StopScanning()
            self._s.reset_input_buffer()
            sleep(0.5)
            self._s.write(b"\xA5\x90") # GetDeviceInfo CMD : 0xA590
            sleep(0.5)
            data = self._s.read(27)
            model_number = str(data[7])
            firmware_version = str(data[9])+"."+str(data[8])
            hardware_version = str(data[10])
            serial_number = ""
            for i in range(11,20):
                serial_number = serial_number+str(data[i])
            return {"model_number":model_number,\
                    "firmware_version":firmware_version,\
                    "hardware_version":hardware_version,\
                    "serial_number":serial_number}
        else:
            raise Exception("Device is not connected")

    def Reset(self):
        """Reboots the Lidar."""
        if(self._is_connected):
            self._s.write(b"\xA5\x80") # X4 Soft Reset CMD: 0xA580
            sleep(0.5)
            self.Disconnect()
            self.Connect()
        else:
            raise Exception("Device is not connected")

    def Disconnect(self):
        """Stop scanning and close serial communication with Lidar."""
        if(self._is_connected):
            if(self._is_scanning == True):
                self.StopScanning()
            self._s.close()
            self._is_connected=False
        else:
            raise Exception("Device is not connected")
```

PyLidar3::__init__.py (7)

```
class YdLidarG4(YdLidarX4):
    """Deals with G4 version of Ydlidar from http://www.ydlidar.com/"""
    def __init__(self,port,chunk_size=6000):
        """Initialize the connection and set port and baudrate."""
        YdLidarX4.__init__(self, port, chunk_size)
        self._baudrate= 230400

    def EnableLowPowerMode(self):
        """Enable Low Power Consumption Mode(Turn motor and
           distance-measuring unit off in StopScanning)\n"""
        if(self._is_connected):
            if self._is_scanning == True:
                self.StopScanning()
            self._s.write(b"\xA5\x01") # G4 EnableLowPowerMode
            while(self._s.inWaiting()==0):
                sleep(0.5)
            self._s.reset_input_buffer()
        else:
            raise Exception("Device is not connected")

    def DisableLowPowerMode(self):
        """Disable Low Power Consumption Mode(Turn motor and
          distance-measuring unit on StopScanning)\n"""
        if(self._is_connected):
            if self._is_scanning == True:
                self.StopScanning()
            self._s.write(b"\xA5\x02") # G4 DisableLowPowerMode
            while(self._s.inWaiting()==0):
                sleep(0.5)
            self._s.reset_input_buffer()
        else:
            raise Exception("Device is not connected")
```

PyLidar3::__init__.py (8)

```
    def GetLowPowerModeStatus(self):
        """Return True if Low Power Consumption Mode is
            Enable else return False"""
        if(self._is_connected):
            if self._is_scanning == True:
                self.StopScanning()
            self._s.reset_input_buffer()
            sleep(0.5)
            self._s.write(b"\xA5\x05") # G4 GetLowPowerModeStatus
            sleep(1)
            data = self._s.read(8)
            if(data[7]!=1):
                return True
            return False
        else:
            raise Exception("Device is not connected")

    def IncreaseCurrentFrequency(self,frequencyStep):
        """Increase current frequency by oneTenth or
            one depends on enum FrequencyStep"""
        if(self._is_connected):
            if self._is_scanning == True:
                self.StopScanning()
            if(frequencyStep.value==1):
                self._s.write(b"\xA5\x09") #G4
            elif(frequencyStep.value==2):
                self._s.write(b"\xA5\x0B") #G4
            while(self._s.inWaiting()==0):
                sleep(0.5)
            self._s.reset_input_buffer()
        else:
            raise Exception("Device is not connected")
```

PyLidar3::__init__.py (9)

```
    def DecreaseCurrentFrequency(self,frequencyStep):
        """Decrease current frequency by oneTenth or
           one depends on enum FrequencyStep"""
        if(self._is_connected):
            if self._is_scanning == True:
                self.StopScanning()
            if(frequencyStep.value==1):
                self._s.write(b"\xA5\x0A")
            elif(frequencyStep.value==2):
                self._s.write(b"\xA5\x0C")
            while(self._s.inWaiting()==0):
                sleep(0.5)
            self._s.reset_input_buffer()
        else:
            raise Exception("Device is not connected")

    def GetCurrentFrequency(self):
        """Returns current frequency in hertz"""
        if(self._is_connected):
            if self._is_scanning == True:
                self.StopScanning()
            self._s.reset_input_buffer()
            sleep(0.5)
            self._s.write(b"\xA5\x0D")
            sleep(0.5)
            data = self._s.read(11)
            if(data[0]==165):
                return (self._HexArrToDec(data[-4:]))/100.0
            else:
                return (self._HexArrToDec(data[:4]))/100.0
        else:
            raise Exception("Device is not connected")
```

PyLidar3::__init__.py (10)

```
    def EnableConstantFrequency(self):
        """Enables constant frequency \n default:Enable"""
        if(self._is_connected):
            if self._is_scanning == True:
                self.StopScanning()
            self._s.write(b"\xA5\x0E")
            while(self._s.inWaiting()==0):
                sleep(0.5)
            self._s.reset_input_buffer()
        else:
            raise Exception("Device is not connected")

    def DisableConstantFrequency(self):
        """Disable constant frequency \n default:Enable"""
        if(self._is_connected):
            if self._is_scanning == True:
                self.StopScanning()
            self._s.write(b"\xA5\x0F")
            while(self._s.inWaiting()==0):
                sleep(0.5)
            self._s.reset_input_buffer()
        else:
            raise Exception("Device is not connected")

    def SwitchRangingFrequency(self):
        """Switch between ranging frequencies 4khz,
            8khz and 9khz.\ndefault:9khz"""
        if(self._is_connected):
            if self._is_scanning == True:
                self.StopScanning()
            self._s.write(b"\xA5\xD0")
            while(self._s.inWaiting()==0):
                sleep(0.5)
            self._s.reset_input_buffer()
        else:
            raise Exception("Device is not connected")
```

```
# PyLidar3::__init__.py (11)

    def GetCurrentRangingFrequency(self):
        """Returns current Ranging Frequency in khz"""
        if(self._is_connected):
            if self._is_scanning == True:
                self.StopScanning()
            self._s.reset_input_buffer()
            sleep(0.5)
            self._s.write(b"\xA5\xD1") #G4
            sleep(1)
            data = self._s.read(8)
            if(data[-1]==0):
                return 4
            elif(data[-1]==1):
                return 8
            elif(data[-1]==2):
                return 9
        else:
            raise Exception("Device is not connected")

    def Disconnect(self):
        """Stop scanning and close serial communication with Lidar."""
        if(self._is_connected):
            if(self._is_scanning == True):
                self.StopScanning()
            if(self.GetLowPowerModeStatus()==False):
                self.EnableLowPowerMode()
                sleep(2)
            self._s.close()
            self._is_connected=False
        else:
            raise Exception("Device is not connected")
```

LidarTestPlot.py

```
# LidarTestPlot.py (1)
import threading
import PyLidar3
import matplotlib.pyplot as plt
import math
import time

def draw():
    global is_plot
    while is_plot:
        plt.figure(1)
        plt.cla()
        plt.ylim(-9000,9000)
        plt.xlim(-9000,9000)
        plt.scatter(x,y,c='r',s=8)
        plt.pause(0.001)
    plt.close("all")

# ----------------------------------------
is_plot = True
x=[]
y=[]
for _ in range(360):
    x.append(0)
    y.append(0)
```

```
# LidarTestPlot.py (2)

port =  input("Enter port name which lidar is connected:") #windows
Obj = PyLidar3.YdLidarX4(port)  #PyLidar3.your_version_of_lidar(port, chunk_size)
threading.Thread(target=draw).start()
if(Obj.Connect()):
    print(Obj.GetDeviceInfo())
    gen = Obj.StartScanning()
    t = time.time() # start time
    while (time.time() - t) < 30: #scan for 30 seconds
        data = next(gen)
        for angle in range(0, 360):
            if(data[angle]>1000):
                x[angle] = data[angle] * math.cos(math.radians(angle))
                y[angle] = data[angle] * math.sin(math.radians(angle))
    is_plot = False
    Obj.StopScanning()
    Obj.Disconnect()
else:
    print("Error connecting to device")
```

Map 작성 (Mapping)

◆ Map 작성 (Mapping)

- 수집된 거리 정보를 기반으로 2차원 또는 3차원 지도 (map) 작성
- 거리 측정에서 발생되는 오류들을 처리하기 위한 filtering, smoothing 기능 사용
- SLAM (Simultaneous Localization and Mapping)에서는 위치 추정과 지도 작성을 함께 수행

LiDAR 기반 Map 작성의 시뮬레이션 예제

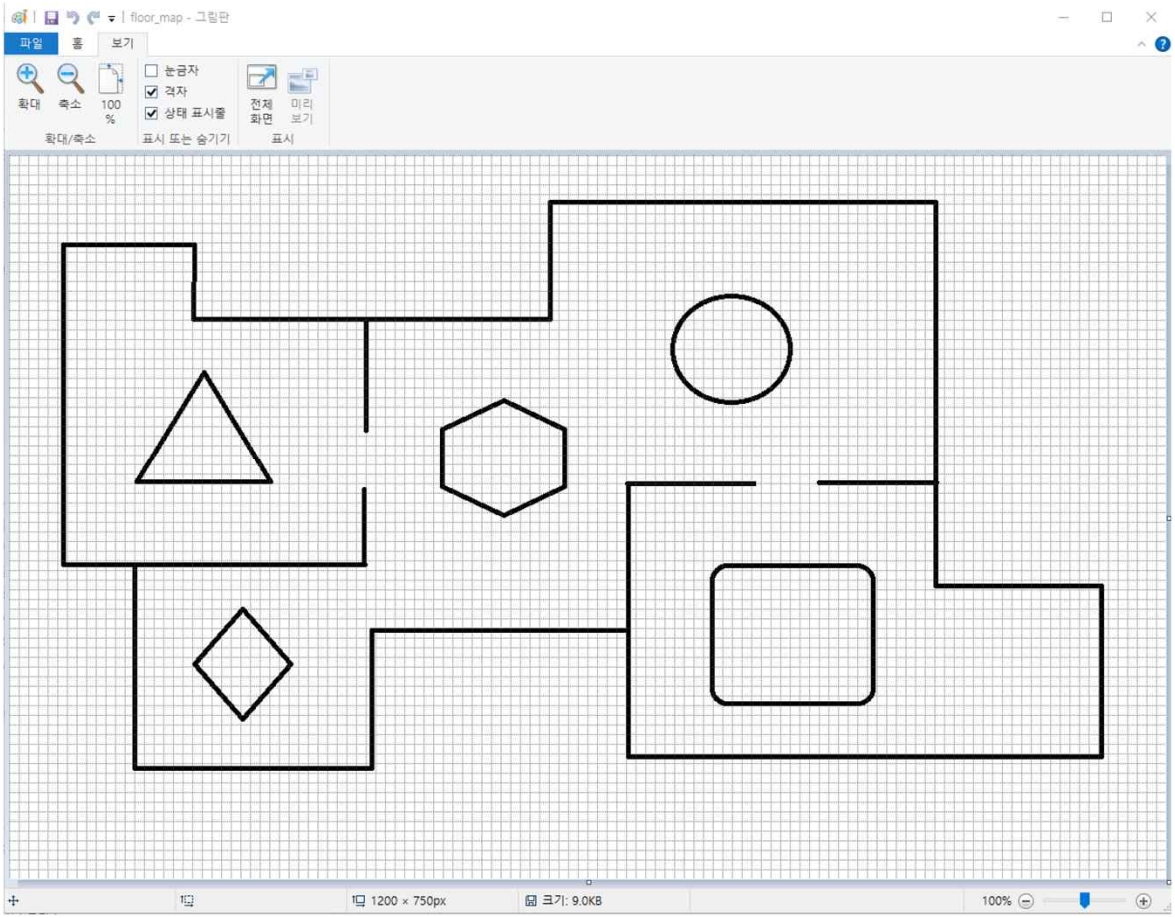
LiDAR 기반 Map 작성 시뮬레이션

◆ LiDAR 기반 Map 작성 시뮬레이션 기능 개요

- map 작성 대상인 floor_map을 MS-Paint (그림판)로 미리 준비
- pygame 모듈의 image.load() 메소드를 사용하여 floor_map을 loading
- 마우스 왼쪽 버튼을 누른 상태의 마우스 위치를 LiDAR의 위치로 사용
- pygame 모듈의 mouse.get_pos() 메소드를 사용하여 LiDAR 위치를 파악한 후, 주변 360°의 LiDAR_Range 이내에 있는 물체를 인식하고, 인식된 물체를 표시

Floor_Map 준비

◆ MS-Paint (그림판)를 사용하여 floor_map.png (1200 x 750) 준비



Colors.py

```
# Colors.py

import random

Color_Black = (0, 0, 0)
Color_Grey = (70, 70, 70)
Color_Blue = (0, 0, 255)
Color_Green = (0, 255, 0)
Color_Red = (255, 0, 0)
Color_White = (255, 255, 255)

def random_color():
    levels = range(32, 256, 32)
    return tuple(random.choice(levels) for _ in range(3))
```

SLAM_Sim_Env.py

```
# SLAM_Sim_Env.py (1)

import math
import numpy as np
import pygame
import Colors

neighbor_range = 20
class Env_Point():
    def __init__(self):
        self.x = 0
        self.y = 0
        self.cluster_id = -1 # initially not belong to any cluster
        self.value = 0 # value of this point

class SLAM_Sim_Environment():
    def __init__(self, MapDimensions, floor_map_fname):
        pygame.init()
        self.obj_points = []
        #self.clusters = []
        #self.num_clustered_points = 0
        self.floor_map = pygame.image.load(floor_map_fname)
        self.mapW, self.mapH = MapDimensions
        self.MapWindowTitle = "SLAM_Sim"
```

```
# SLAM_Sim_Env.py (2)

        pygame.display.set_caption(self.MapWindowTitle)
        self.map = pygame.display.set_mode((self.mapW, self.mapH))

        self.obj_map = pygame.display.set_mode((self.mapW, self.mapH))
        self.obj_map.fill(Colors.Color_White)
        # initialize env_points 2-dimensional list
        self.env_points = [] # used as two-dimentional array of env_point
        self.num_sensed_points = 0
        for y in range(self.mapH):
            row = []
            for x in range(self.mapW):
                env_point = Env_Point()
                env_point.x = x
                env_point.y = y
                env_point.cluster_id = -1
                env_point.value = 0
                row.append(env_point)
            self.env_points.append(row)
```

SLAM_Sim_Env.py

```
# SLAM_Sim_Env.py (3)

    def update_obj_map(self, points, lidar):
        #print("SLAM_Sim_Env::update_obj_map() - lidar.num_objects\
        #  = {}".format(lidar.num_objects))
        if len(points) <= 0:
            print("Trying to store_data with empty data")
            return
        for point in points:
            # obj : (distance, angle, (lidar_pos x, y), obj_pos(x, y))
            pos_x, pos_y = point
            if isinstance(pos_x, int) and isinstance(pos_y, int):
                if self.env_points[pos_y][pos_x].value != 255:
                    self.env_points[pos_y][pos_x].value = 255
                      # set as occupied by object
                    self.num_sensed_points += 1
                    pygame.draw.circle(self.obj_map, Colors.Color_Blue, point, 2, 0)
            else:
                print("update_obj_map():: Trying to update_obj_map with\
                   non-int coordinates({}, {})".format(pos_x, pos_y))
```

SLAM_Sim_LiDAR.py

```
# SLAM_Sim_LiDAR.py (1)

import pygame, math
import numpy as np
import Colors

class LiDAR:
    def __init__(self, laser_range, environment, uncertainty):
        self.lidar_range = laser_range
        self.speed = 4 # rounds per second
        self.sigma = np.array([uncertainty[0], uncertainty[1]])
        self.lidar_pos = (0, 0)
        self.floor_map = environment.floor_map
        self.winW, self.winH = pygame.display.get_surface().get_size()
        self.sensed_objects = []
        self.sensed_points = []
        self.num_objects = 0

    def add_uncertainty(self, distance, angle, sigma):
        mean = np.array([distance, angle])
        cov = np.diag(sigma ** 2)
        dist, angle = np.random.multivariate_normal(mean, cov)
        distance = max(dist, 0)
        angle = max(angle, 0)
        return distance, angle

    def distance(self, obj_pos):
        sq_dx = (obj_pos[0] - self.lidar_pos[0])**2
        sq_dy = (obj_pos[1] - self.lidar_pos[1])**2
        dist = math.sqrt(sq_dx + sq_dy)
        return dist
```

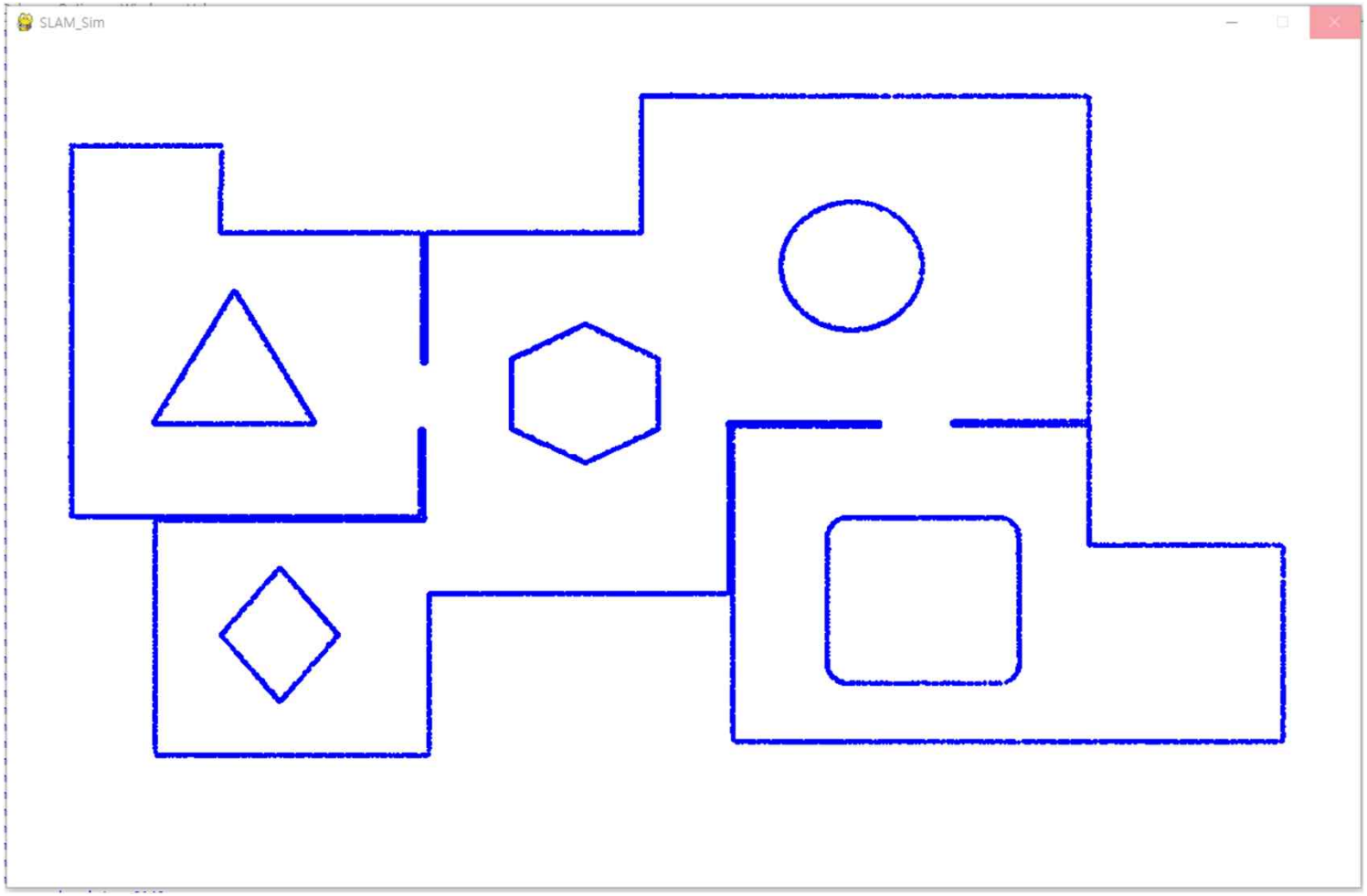
```
# SLAM_Sim_LiDAR.py (2)

    def sense_obstacles(self):
        sensed_points = []
        x1, y1 = self.lidar_pos[0], self.lidar_pos[1]
        for angle in np.linspace(0, 2*math.pi, 60, False):
            x2, y2 = (x1 + self.lidar_range*math.cos(angle),\
                y1 - self.lidar_range*math.sin(angle))
            for i in range(0, 100):
                u = i / 100
                x = int(x2*u + x1*(1-u))
                y = int(y2*u + y1*(1-u))
                if not (0<x<self.winW and 0<y<self.winH):
                    continue
                color = self.floor_map.get_at((x, y))
                if color == Colors.Color_Black:
                    distance = self.distance((x, y))
                    dist, angle = self.add_uncertainty(distance, angle, self.sigma)
                    sensed_obj = (dist, angle, self.lidar_pos, (x, y))
                    # sensed_obj : (distance, angle, (lidar_pos x, y), obj_pos(x, y))
                    sensed_points.append(sensed_obj[3])
                    self.sensed_points.append(sensed_obj[3])
                    self.sensed_objects.append(sensed_obj)
                    self.num_objects += 1
                    break
        return sensed_points
```

SLAM_Sim_Main.py

```
# SLAM_Sim main.py (1)

import SLAM_Sim_Env, SLAM_Sim_LiDAR, Colors
import pygame, math, time

environment = SLAM_Sim_Env.\
    SLAM_Sim_Environment((1200, 750), "floor_map.png")
lidar = SLAM_Sim_LiDAR.LiDAR(200, environment,\
    uncertainty=(0.5, 0.01))
pygame.mouse.set_cursor(*pygame.cursors.diamond)

MAX_ROUND = 350
running = True
round = 0
```

```
# SLAM_Sim main.py (2)

while round < MAX_ROUND:
    print("Round {:5d}, num_sensed_points {:5d}“\
      .format(round, environment.num_sensed_points))
    sensor_on = False
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
            break
        if pygame.mouse.get_focused() and\
           pygame.mouse.get_pressed() == (1, 0, 0):
            sensor_on = True
            mouse_pos = pygame.mouse.get_pos()
            lidar.lidar_pos = mouse_pos # set the position of lidar
            sensed_points = lidar.sense_obstacles()
            environment.update_obj_map(sensed_points, lidar)
        elif not pygame.mouse.get_focused() and\
            pygame.mouse.get_pressed() != (1, 0, 0):
            sensor_on = False
    environment.map.blit(environment.obj_map, (0, 0))
    pygame.display.update()
    if running == False:
        break
    time.sleep(0.1)
    round += 1

input("\nInut any key to Quit SLAM simulation")
pygame.quit()
```

실행 결과

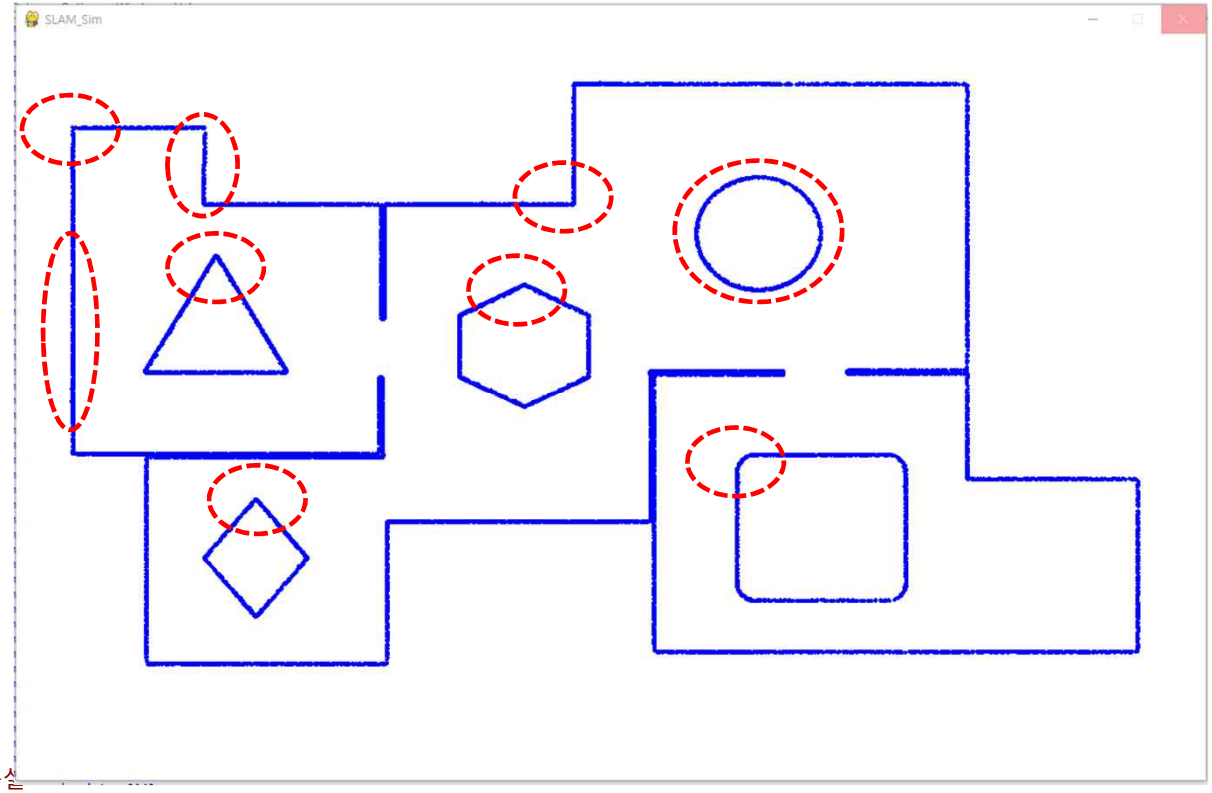


LiDAR Sensor의 Raw Data로 부터 특징 추출 (Feature Extraction)

LiDAR 센서의 raw data들의 클러스터 구성 (Clustering)

◆ Raw Data Point들의 Clustering

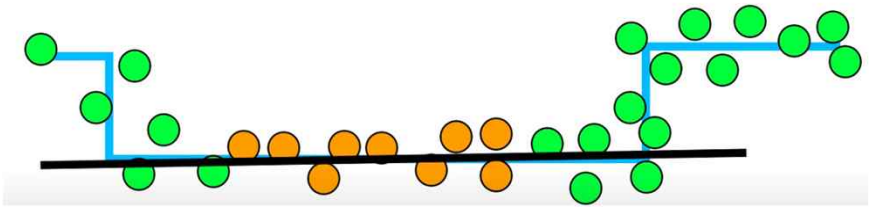
- LiDAR 센서 데이터로 부터 전달받은 장애물 위치 (거리, 방향) 정보들은 점으로 표현
- 다수의 인접된 점들을 cluster로 구성

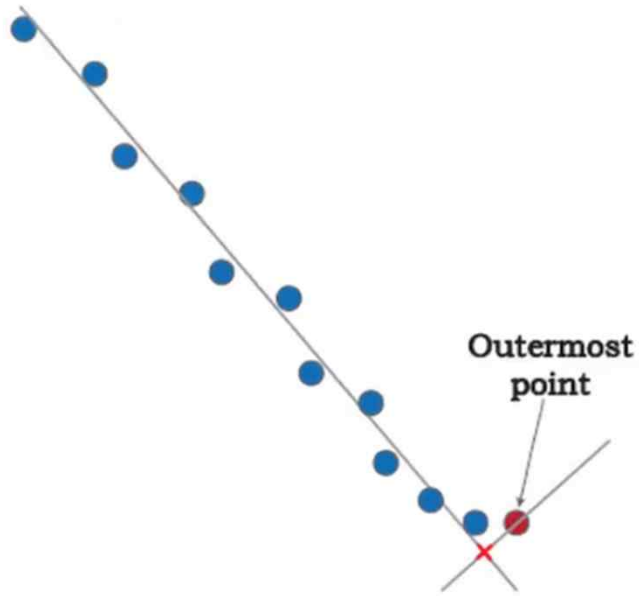


LiDAR 센서의 raw data로 부터 특징 추출 (Feature Extraction)

◆ 특징 추출

- LiDAR 센서 데이터로 부터 코너, 직선 구간 (line segment)등의 특징을 추출
- seed region growing
- length threshold
- finding outmost point





SLAM_Sim_FeatureExtraction.py

◆ Feature Extraction

- ref. Haiming Gao et. al, "A line segment extraction algorithm using laser data based on seeded region growing," International Journal of Advanced Robotic Systems, 2018.
- distance of two points
- distance of a point and line
- extract point from line
- slop-intercept
- intersect

- detection of seed segment
- growing seed segment

SLAM_Sim_FeatureExtraction.py

```
# SLAM_Sim_FeatureExtraction.py (1)

import math
import numpy as np
from fractions import Fraction
from scipy.linalg import lstsq

class FeatureExtraction():
    def __init__(self, env, lidar, max_cluster):
        self.EPSILON = 10
        self.DELTA = 10
        self.SNUM = 6
        self.PMIN = 5
        self.GMAX = 20
        self.LMIN = 5 # minimum length of a line segment
        self.seed_segments = []
        self.line_segments = []
        self.env = env
        self.lidar = lidar
        self.line_params = None
        self.num_points = 0 # the number of LiDAR points contained in the line segment

    def dist_p2p(self, p1, p2):
        if (isinstance(p1[0], int) or isinstance(p1[0], float)) and (isinstance(p1[1], int) or isinstance(p1[1], float)) and\
            (isinstance(p2[0], int) or isinstance(p2[0], float)) and (isinstance(p2[1], int) or isinstance(p2[1], float)):
            sq_dx = (p1[0] - p2[0]) ** 2
            sq_dy = (p1[1] - p2[1]) ** 2
            return math.sqrt(sq_dx + sq_dy)
        else:
            print("Warning in dist_p2p() :: p1 = {}, p2 = {}".format(p1, p2))
            return 0.0
```

SLAM_Sim_FeatureExtraction.py (2)

```
    def dist_p2line(self, line_eq, p):
        #print("dist_p2line:: line_eq = {}, p = {}".format(line_eq, p))
        A, B, C = line_eq
        if (isinstance(p[0], int) or isinstance(p[0], float))\
            and (isinstance(p[1], int) or isinstance(p[1], float)) :
            dist = abs(A*p[0] + B*p[1] + C) / math.sqrt(A**2+B**2)
            return dist
        else:
            print("Error in dist_p2line():: p[0] = {}, p[1] = {}".format(p[0], p[1]))
            return None

    def line_2points(self, m, b):
        x = 5
        y = m * x + b
        x2 = 2000
        y2 = m * x2 + b
        return [(x, y), (x2, y2)]

    def lineForm_G2SI(self, A, B, C): # general form of slope-intercept
        m = -A / B
        b = -C / B
        return m, b

    def lineForm_SI2G(self, m, b):
        A, B, C = -m, 1, -b
        if A < 0:
            A, B, C = -A, -B, -C
        den_a = Fraction(A).limit_denominator(1000).as_integer_ratio()[1]
        den_c = Fraction(C).limit_denominator(1000).as_integer_ratio()[1]
        gcd = np.gcd(den_a, den_c)
        lcm = den_a * den_c / gcd

        A = A * lcm
        B = B * lcm
        C = C * lcm
        return A, B, C
```

SLAM_Sim_FeatureExtraction.py (3)

```
    def line_intersect_general(self, line_eq1, line_eq2):
        a1, b1, c1 = line_eq1
        a2, b2, c2 = line_eq2
        x = (c1*b2 - b1*c2) / (b1*a2 - a1*b2)
        y = (a1*c2 - a2*c1) / (b1*a2 - a1*b2)
        return x, y

    def points_2line(self, p1, p2):
        m, b = 0, 0
        if p2[0] == p1[0]:
            pass
        elif isinstance(p1[0], int) and isinstance(p1[1], int) and\
             isinstance(p2[0], int) and isinstance(p2[1], int):
            m = (p2[1] - p1[1]) / (p2[0] - p1[0])
            b = p2[1] - m*p2[0]
        else:
            print("Warning in points_2line() :: p1 = {}, p2 = {}".format(p1, p2))
        return m, b

    def projection_p2line(self, p, m, b):
        x, y = p
        m2 = -1 / m
        c2 = y - m2 * x
        intersection_x = -(b - c2) / (m - m2)
        intersection_y = m2 * intersection_x + c2
        return intersection_x, intersection_y
```

```
# SLAM_Sim_FeatureExtraction.py (4)

    def lstsq_fit(self, points):
        #print("lstsq_fit({})..".format(len(points)), end='') # for debugging only
        x_data = []
        y_data = []
        for p in points:
            if (isinstance(p[0], int) or isinstance(p[0], float))\
               and (isinstance(p[1], int) or isinstance(p[1], float)):
                x_data.append(p[0])
                y_data.append(p[1])
        x = np.array(x_data)
        X = np.vstack([x, np.ones(len(x))]).T
        y = np.array(y_data)
        p, resiuals, rank, s = lstsq(X, y)
        #print("lstsq(X, y, rcond=None) => p={}, residuals={}, rank={}, s={}".format(p, resiuals, rank, s))
        m, b = p[0], p[1]
        #print("np_lstsq_fit() returns m = {}, b = {}".format(m, b))

        return m, b

    def predictPoint(self, line_params, sensed_point, lidar_pos=(0, 0)):
        m, b = self.points_2line(lidar_pos, sensed_point)
        line_eq1 = self.lineForm_SI2G(m, b)
        pred_x, pred_y = self.line_intersect_general(line_eq1, line_params)
        return (pred_x, pred_y)
```

SLAM_Sim_FeatureExtraction.py (5)

```
  def seed_segment_detection_from_clusters(self, cluster_points):
      #print("SSDC..", end='') # for debugging only
      flag = True
      num_points = len(cluster_points)
      if num_points < self.SNUM:
          return None # cluster segment is too short
      seed_segments = []
      for i in range(num_points):
          predicted_points_to_draw = []
          j = min(i + self.SNUM, len(cluster_points))
          m, b = self.lstsq_fit(cluster_points[i:j])
          params = self.lineForm_SI2G(m, b)
          for k in range(i, j):
              predicted_point = self.predictPoint(params, cluster_points[k])
              if not (isinstance(predicted_point[0], int) or isinstance(predicted_point[0], float)) and\
                 not (isinstance(predicted_point[1], int) or isinstance(predicted_point[1], float)):
                  print("Warning in seed_segment_detection:: predicted_point({}, {})".format(predicted_point[0], predicted_point[1]))
              predicted_points_to_draw.append(predicted_point)
              pk = cluster_points[k]
              if not isinstance(pk[0], int) or not isinstance(pk[1], int):
                  print("Warning in seed_segment_detection:: pk({}, {})".format(pk[0], pk[1]))
              d1 = self.dist_p2p(predicted_point,pk)

              if d1 > self.DELTA:
                  flag = False
                  break
              d2 = self.dist_p2line(params, predicted_point)
              if d2 > self.EPSILON:
                  flag = False
                  break
          if flag:
              self.line_params = params
              result = [cluster_points[i:j], predicted_points_to_draw, (i, j)]
              return result
      return None
```

```
# SLAM_Sim_FeatureExtraction.py (6)

    def seed_segment_growing_from_clusters(self, cluster_points, index_range):
        #print("SSG..", end='') # for debugging only
        num_points = len(cluster_points)
        if num_points < self.SNUM:
            return None # cluster segment is too short

        line_eq = self.line_params
        pb, pf = index_range
        pf = min(pf, len(cluster_points)-1)
        while True:
            pf_point = cluster_points[pf]
            if not isinstance(pf_point[0], int) or not isinstance(pf_point[1], int):
                print("Warning in seed_segment_growing:: calling dist_p2line with pf_point({}, {})".format(pf_point[0], pf_point[1]))
            if self.dist_p2line(line_eq, pf_point) >= self.EPSILON:
                break
            if pf >= num_points - 1:
                break
            else:
                #m, b = self.odr_fit(self.lidar.sensed_points[pb:pf])
                #m, b = self.linear_regression_fit(self.lidar.sensed_points[pb:pf])
                m, b = self.lstsq_fit(self.lidar.sensed_points[pb:pf])
                line_eq = self.lineForm_SI2G(m, b)

                point = cluster_points[pf]
            pf = pf + 1
            next_point = cluster_points[pf]
            if not isinstance(next_point[0], int) or not isinstance(next_point[1], int):
                print("Warning in seed_segment_growing:: next_point({}, {})".format(next_point[0], next_point[1]))
            if self.dist_p2p(point, next_point) > self.GMAX:
                break
        pf = pf - 1
```

```
# SLAM_Sim_FeatureExtraction.py (6)

        while True:
            pb_point = cluster_points[pb]
            if not isinstance(pb_point[0], int) or not isinstance(pb_point[1], int):
                print("Warning in seed_segment_growing():: calling dist_p2line with pb_point({}, {})".format(pb_point[0], pb_point[1]))
            if self.dist_p2line(line_eq, pb_point) >= self.EPSILON:
                break
            if pb <= 0:
                break
            else:
                #m, b = self.odr_fit(self.lidar.sensed_points[pb:pf])
                #m, b = self.linear_regression_fit(self.lidar.sensed_points[pb:pf])
                m, b = self.lstsq_fit(self.lidar.sensed_points[pb:pf])
                line_eq = self.lineForm_SI2G(m, b)
                point = self.lidar.sensed_points[pb]
            pb = pb - 1
            next_point = self.lidar.sensed_points[pb]
            if not isinstance(next_point[0], int) or not isinstance(next_point[1], int):
                print("Warning in seed_segment_growing:: calling dist_p2line with next_point({}, {})".format(next_point[0], next_point[1]))
            if self.dist_p2p(point, next_point) > self.GMAX:
                break
        pb = pb + 1

        dist_line_seg = self.dist_p2p(cluster_points[pb], cluster_points[pf]) # distance of line segment
        pts_line_seg = len(cluster_points[pb:pf]) # number of points in line segment
        if (dist_line_seg >= self.LMIN) and (pts_line_seg >= self.PMIN):
            self.line_params = line_eq
            m, b = self.lineForm_G2SI(line_eq[0], line_eq[1], line_eq[2])
            self.two_points = self.line_2points(m, b)
            self.line_segments.append((cluster_points[pb+1], cluster_points[pf-1]))
            result = [cluster_points[pb:pf], self.two_points,\
                    (cluster_points[pb], cluster_points[pf]), pf, line_eq, (m, b)]
            return result
        else:
            return None
```

SLAM_Sim_Env.py (확장본)

```
# SLAM_Sim_Env.py (1)

import math, pygame
import numpy as np
import Colors

neighbor_range = 20
class Env_Point():
    def __init__(self):
        self.x = 0
        self.y = 0
        self.cluster_id = -1 # initially not belong to any cluster
        self.value = 0 # value of this point

class Cluster():
    def __init__(self):
        self.id = -1
        self.points = [] # points (x, y) included in this cluster
        self.shape = "not_defined"
    def add_point(self, point):
        self.points.append(point)
```

```
# SLAM_Sim_Env.py (2)

class SLAM_Sim_Environment():
    def __init__(self, MapDimensions, floor_map_fname):
        pygame.init()
        self.obj_points = []
        self.clusters = []
        self.num_clustered_points = 0
        self.floor_map = pygame.image.load(floor_map_fname)
        self.mapW, self.mapH = MapDimensions
        self.MapWindowTitle = "SLAM_Sim"
        pygame.display.set_caption(self.MapWindowTitle)
        self.map = pygame.display.set_mode((self.mapW, self.mapH))
        self.obj_map = pygame.display.set_mode((self.mapW, self.mapH))
        self.obj_map.fill(Colors.Color_White)
        #self.map.blit(self.obj_map, (0, 0))
        # initialize env_points 2-dimensional list
        self.env_points = [] # used as two-dimentional array of env_point
        self.num_sensed_points = 0
        for y in range(self.mapH):
            row = []
            for x in range(self.mapW):
                env_point = Env_Point()
                env_point.x = x
                env_point.y = y
                env_point.cluster_id = -1
                env_point.value = 0
                row.append(env_point)
            self.env_points.append(row)
```

```
# SLAM_Sim_Env.py (3)

    def update_obj_map(self, points, lidar):
        #print("SLAM_Sim_Env::update_obj_map() - lidar.num_objects = {}".format(lidar.num_objects))
        if len(points) <= 0:
            print("Trying to store_data with empty data")
            return
        for point in points:
            # obj : (distance, angle, (lidar_pos x, y), obj_pos(x, y))
            pos_x, pos_y = point
            if isinstance(pos_x, int) and isinstance(pos_y, int):
                #obj_pos = (pos_x, pos_y)# for test only
                #self.obj_map.set_at(point, Colors.Color_Blue) #Colors.Color_Blue
                if self.env_points[pos_y][pos_x].value != 255:
                    self.env_points[pos_y][pos_x].value = 255 # set as occupied by object
                    self.num_sensed_points += 1
                    pygame.draw.circle(self.obj_map, Colors.Color_Blue, point, 2, 0)
            else:
                print("update_obj_map():: Trying to update_obj_map with non-int coordinates({}, {})".format(pos_x, pos_y))
```

SLAM_Sim_Env.py (2)

```
    def update_clusters(self):
        y_max = len(self.env_points)
        x_max = len(self.env_points[0])
        self.num_clustered_points = 0
        for y in range(y_max):
            for x in range(x_max):
                if self.env_points[y][x].value == 0:
                    continue
                if self.env_points[y][x].cluster_id == -1:
                    neighbors = []
                    x_low = 0 if (x - neighbor_range) < 0 else x - neighbor_range
                    x_high = x_max if (x + neighbor_range) > x_max else x + neighbor_range
                    y_low = 0 if (y - neighbor_range) < 0 else y - neighbor_range
                    y_high = y_max if (y + neighbor_range) > y_max else y + neighbor_range
                    for y_n in range(y_low, y_high):
                        for x_n in range(x_low, x_high):
                            if y == y_n and x == x_n:
                                continue
                            if self.env_points[y_n][x_n].value == 0:
                                continue
                            neighbors.append((x_n, y_n))
                    if len(neighbors) != 0:
                        dist_min = neighbor_range * 1.5
                        neighbor_min_dist = None
                        cluster_id_min = -1
                        for nh in neighbors:
                            (x_n, y_n) = nh
                            dist = np.sqrt((x - x_n)**2 + (y - y_n)**2)
                            nh_cid = self.env_points[y_n][x_n].cluster_id
                            if (dist < dist_min) and  nh_cid != -1:
                                cluster_id_min = nh_cid
                                neighbor_min_dist = nh
```

SLAM_Sim_Env.py (2)

```
                    if neighbor_min_dist != None:
                        self.clusters[cluster_id_min].points.append((x, y))
                        self.env_points[y][x].cluster_id = cluster_id_min
                    else:
                        print("update_clusters :: configuring a new cluster by adding ({}, {})".format(x, y))
                        cluster = Cluster()
                        cluster.id = len(self.clusters)
                        cluster.points = [(x, y)]
                        cluster.shape = "non_defined"
                        self.clusters.append(cluster)
                        self.env_points[y][x].cluster_id = cluster.id
                else: # if len(neighbors) == 0: # no nearby neighbor(s)
                    print("update_clusters :: configuring a new cluster by adding ({}, {})".format(x, y))
                    cluster = Cluster()
                    cluster.id = len(self.clusters)
                    cluster.points = [(x, y)]
                    cluster.shape = "non_defined"
                    self.clusters.append(cluster)
                    self.env_points[y][x].cluster_id = cluster.id
                self.num_clustered_points += 1
```

```
# SLAM_Sim_Env.py (2)

        # expand cluster with its neighbor points
        num_clusters = len(self.clusters)
        self.num_clustered_points = 0
        for cluster in self.clusters:
            num_points_in_cluster = len(cluster.points)
            self.num_clustered_points += num_points_in_cluster
            for point in cluster.points:
                my_cid = cluster.id
                points = []
                (x, y) = point
                x_low = 0 if (x - neighbor_range) < 0 else x - neighbor_range
                x_high = x_max if (x + neighbor_range) > x_max else x + neighbor_range
                y_low = 0 if (y - neighbor_range) < 0 else y - neighbor_range
                y_high = y_max if (y + neighbor_range) > y_max else y + neighbor_range
                for y_n in range(y_low, y_high):
                    for x_n in range(x_low, x_high):
                        if y == y_n and x == x_n:
                            continue
                        if  self.env_points[y_n][x_n].cluster_id == -1:
                            points.append((x_n, y_n))
                            self.env_points[y_n][x_n].cluster_id = my_cid
                        elif self.env_points[y_n][x_n].value == my_cid:
                            continue
                        elif self.env_points[y_n][x_n].value == 0:
                            continue
                        else: # neighbor cluster can be merged
                            pass # to be updated
        # merge clusters
```

SLAM_Sim_Env.py (2)

```
    def draw_clusters(self, feature_extract):
        num_clusters = len(self.clusters)
        # draw lines from clusters
        for cluster in self.clusters:
            num_points_in_cluster = len(cluster.points)
            result_seg_dect = feature_extract.seed_segment_detection_from_clusters(cluster.points)
            if result_seg_dect == None:
                continue
            seed_segment_objs = result_seg_dect[0] # self.lidar.sensed_objects[i:j]
            predicted_points_to_draw = result_seg_dect[1] #predicted_points_to_draw
            index_range = result_seg_dect[2] # (i, j)
            result_seg_grow = feature_extract.seed_segment_growing_from_clusters(cluster.points, index_range)
            # results format : [self.lidar.sensed_objects[pb:pf], self.two_points, (self.lidar.sensed_objects[pb+1],
            #                   self.lidar.sensed_objects[pf-1]), pf, line_eq, (m, b)]

            if result_seg_grow == None:
                continue

            line_eq = result_seg_grow[4]
            m, c = result_seg_grow[5]
            line_segs = result_seg_grow[0]
            outer_most = result_seg_grow[2]
            break_point_idx = result_seg_grow[3]
            end_points_0 = feature_extract.projection_p2line(outer_most[0], m, c)
            end_points_1 = feature_extract.projection_p2line(outer_most[1], m, c)
            color = Colors.random_color()
            pygame.draw.line(self.obj_map, Colors.Color_Red, outer_most[0], outer_most[1], 4)
            self.map.blit(self.obj_map, (0, 0))
            pygame.display.update()
```

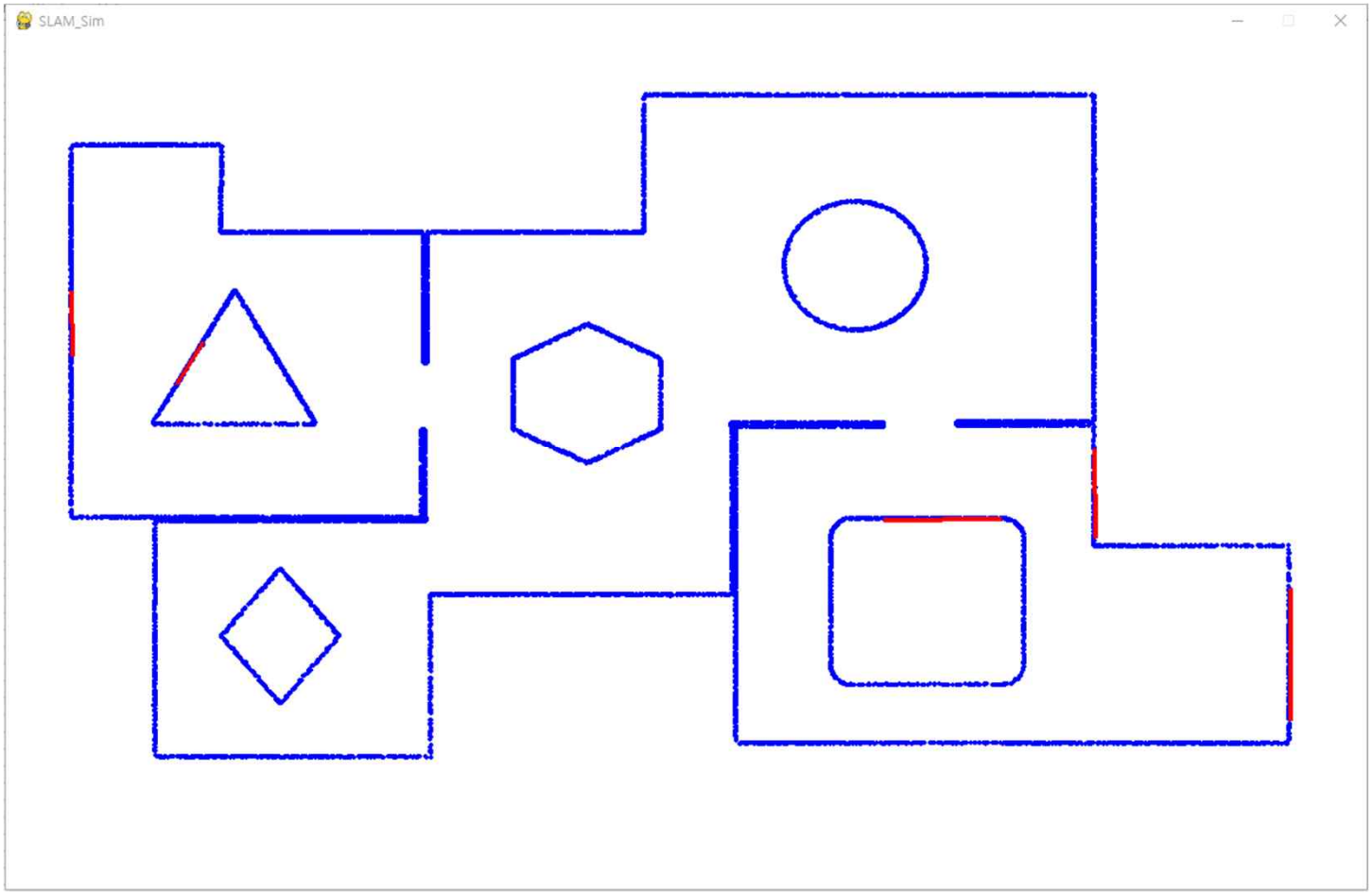
SLAM_Sim_Main.py (확장본)

```
# SLAM_Sim main.py (1)

import SLAM_Sim_Env, SLAM_Sim_LiDAR, SLAM_Sim_FeatureExtraction, Colors
import pygame, math, time

environment = SLAM_Sim_Env.SLAM_Sim_Environment((1200, 750), "floor_map.png")
lidar = SLAM_Sim_LiDAR.LiDAR(200, environment, uncertainty=(0.5, 0.01))
feature_extract = SLAM_Sim_FeatureExtraction.FeatureExtraction(environment, lidar, max_cluster=500)
pygame.mouse.set_cursor(*pygame.cursors.diamond)

MAX_ROUND = 350
running = True
round = 0
while round < MAX_ROUND:
    print("Round {:5d}, num_sensed_points {:5d}, num_clusters {:3d}, num_clustered_points {:4d}"\
          .format(round, environment.num_sensed_points, len(environment.clusters), environment.num_clustered_points))
    sensor_on = False
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
            break
        if pygame.mouse.get_focused() and pygame.mouse.get_pressed() == (1, 0, 0):
            sensor_on = True
            mouse_pos = pygame.mouse.get_pos()
            lidar.lidar_pos = mouse_pos # set the position of lidar
            sensed_points = lidar.sense_obstacles()
            environment.update_obj_map(sensed_points, lidar)
        elif not pygame.mouse.get_focused() and pygame.mouse.get_pressed() != (1, 0, 0):
            sensor_on = False
```

SLAM_Sim_Main.py (확장본)

```
# SLAM_Sim main.py (2)

    # Feature extraction from the collected/sensed data
    environment.update_clusters()
    environment.draw_clusters(feature_extract)

    if running == False:
        break
    environment.map.blit(environment.obj_map, (0, 0))
    pygame.display.update()
    if running == False:
        break

    #time.sleep(0.1)
    round += 1

input("\nInut any key to Quit SLAM simulation")
pygame.quit()
```

실행 결과



Localization (위치 추정)

위치 추정 (Localization)

◆ 위치 추정 (localization) 이란 ?

- 다양한 센서를 사용하여 데이터를 수집
- 융합된 센서 데이터를 기반으로 현재 위치를 측정/추정

◆ 위치 추정 (localization)의 주요 기능 및 알고리즘

- Landmark detection
- Sliding window object detection
  - convolutional implementation of sliding windows
- Selective search for ROI (range of interest)
- Region Proposal Network (RPN)
- Unified Detection

◆ 위치 추정에서 발생할 수 있는 센서 데이터 오류

- 센서 데이터에 포함된 잡음
- 자율주행 자동차 및 로봇의 odometry 데이터 (이동 거리 및 방향)에 포함된 오류

작성된 Map으로 부터 특징 추출 및 관리

◆ 특징 추출 (feature extraction)

- LiDAR 센서로 부터 수집된 데이터를 기반으로 cluster 구성
- 구성된 cluster로 부터 특징 (모서리, 벽면의 접속점, 특정 형태 등)을 추출
- 추출된 특징들의 좌표를 등록하여 추후 위치 추정 (localization)에서 사용

Uncertainty in Autonomous Vehicle

◆ Uncertainty in Sensors (LiDAR, IMU)

- LiDAR의 레이저 신호 반사 물체 재질 차이에 따른 반사 신호 세기 차이
- LiDAR의 수신 레이저 신호에 포함되는 잡음

◆ Uncertainty in Odometry, Slippery Tire

- 타이어의 미끄러짐에 따른 오차 발생 (vehicle orientation with error because of slippery tire)
- 타이어 공기압 차이에 따른 주행 거리 계산에 오차 발생

센서 잡음 및 오차 처리를 위한 필터링 및 스무딩

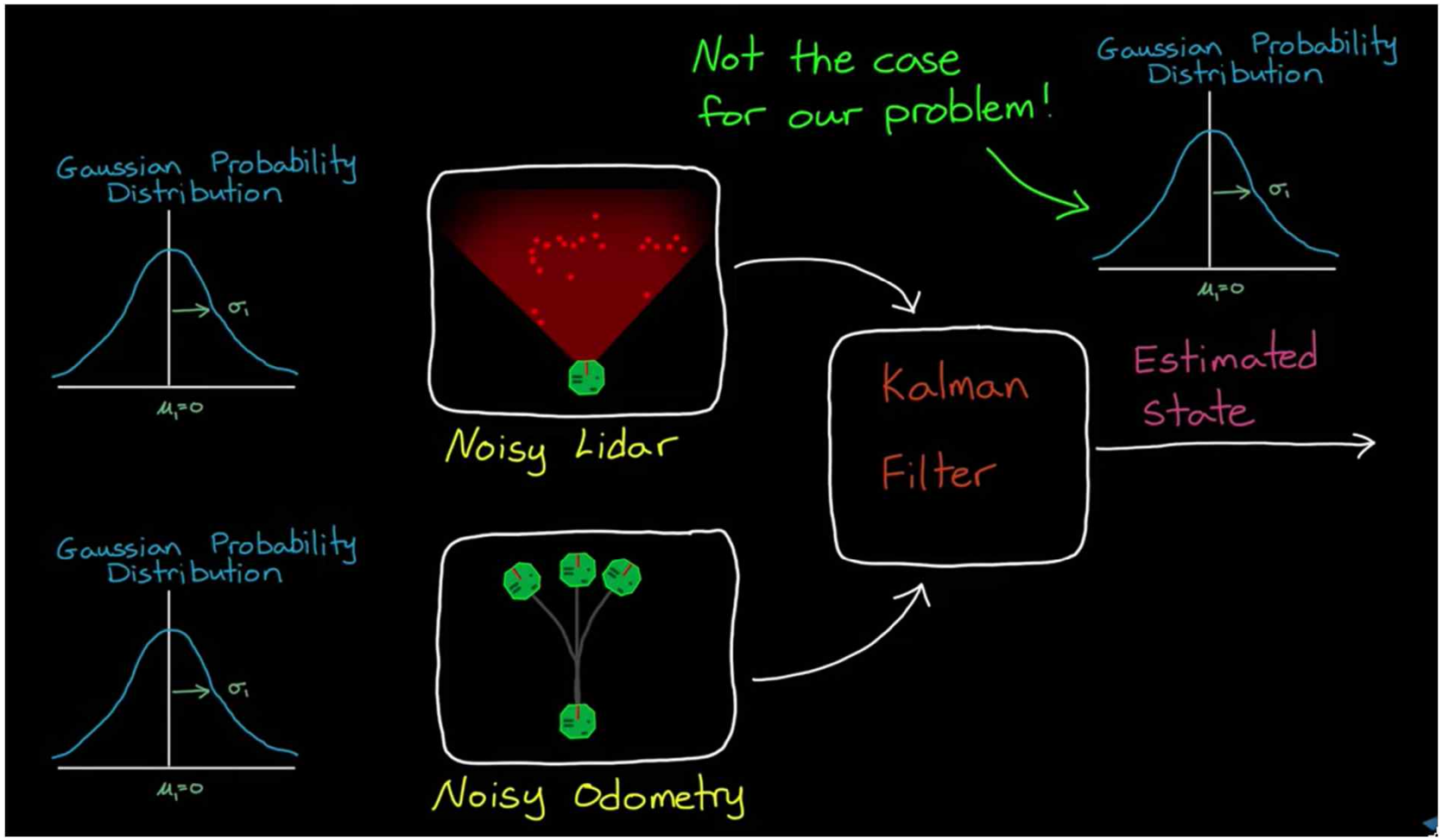
◆ Filtering

- 잡음이 포함된 센서 데이터의 통계적 분석
- 자율주행 자동차 및 로봇의 odometry 데이터 (이동 거리 및 방향)에 대한 보정

◆ Smoothing

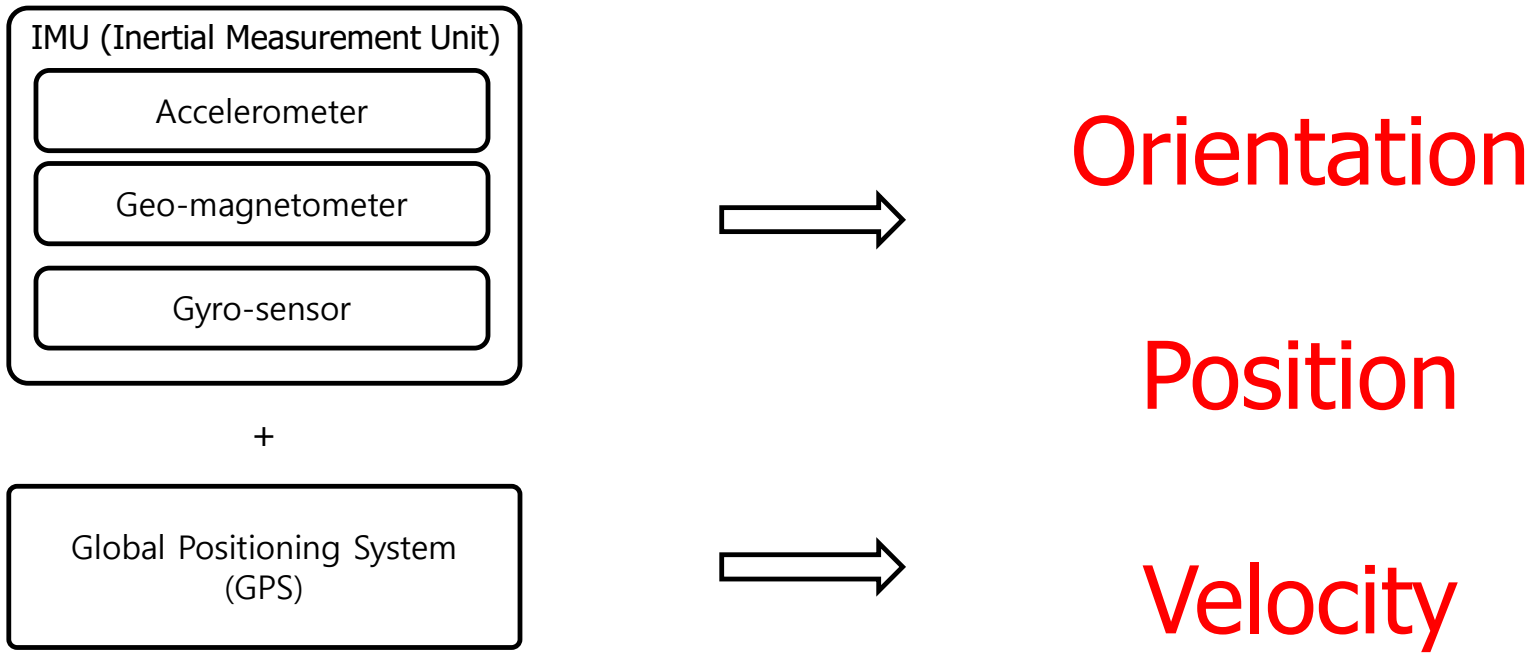
- 일부 오류가 포함된 데이터를 통계적으로 처리하여 전체 오차가 줄어들 수 있도록 보정

Filtering for Noisy Sensor Data



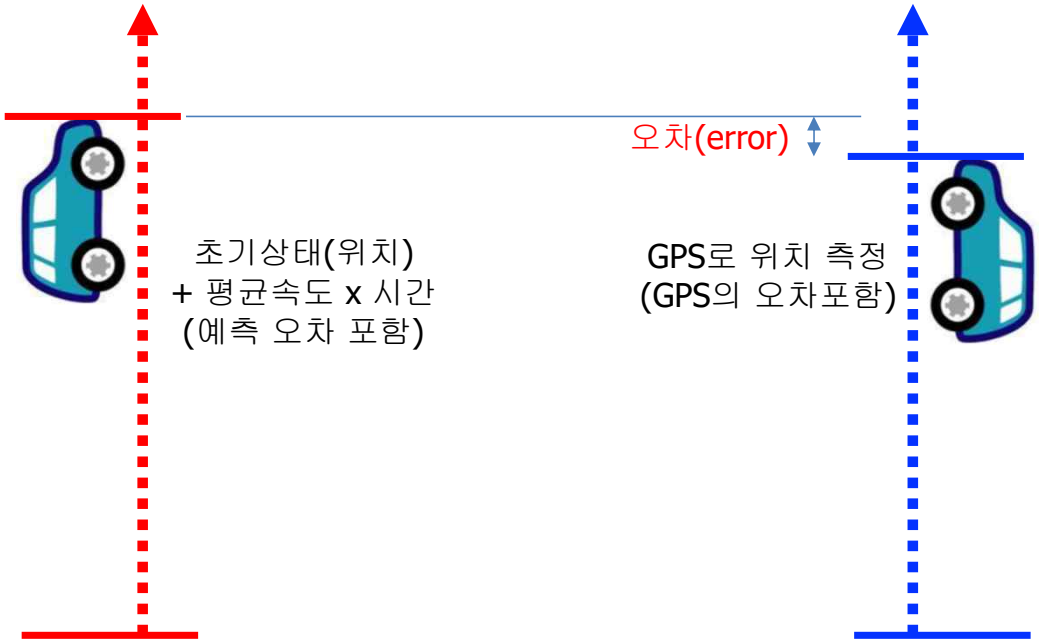
Sensor Fusion

◆ Sensor Fusion



예측 (Predict) 및 수정(Correct)

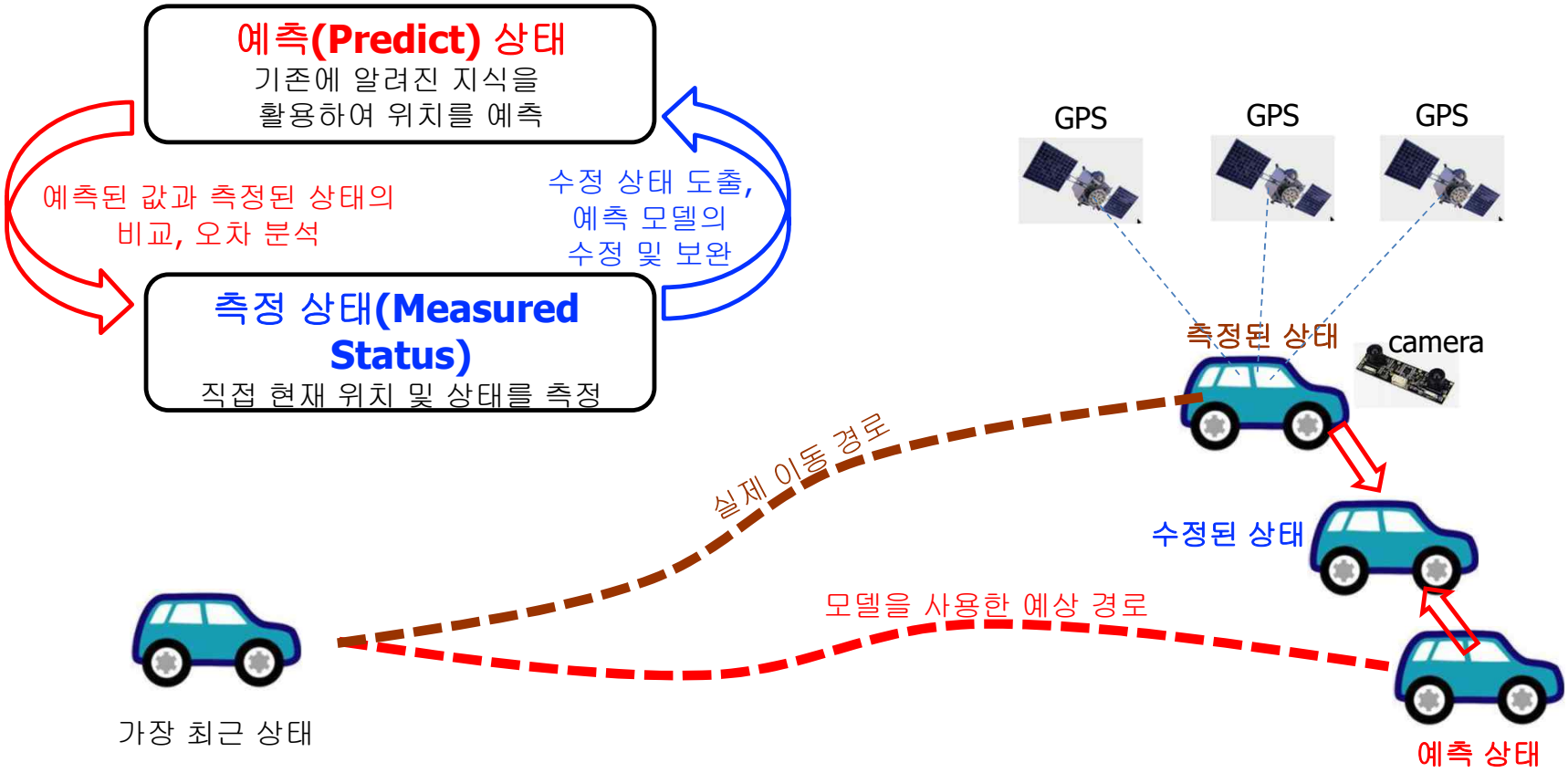
◆ 예측 (Predict) 및 수정(Correct)



예측(Predict)
기존에 알려진 지식을
활용하여 위치를 예측

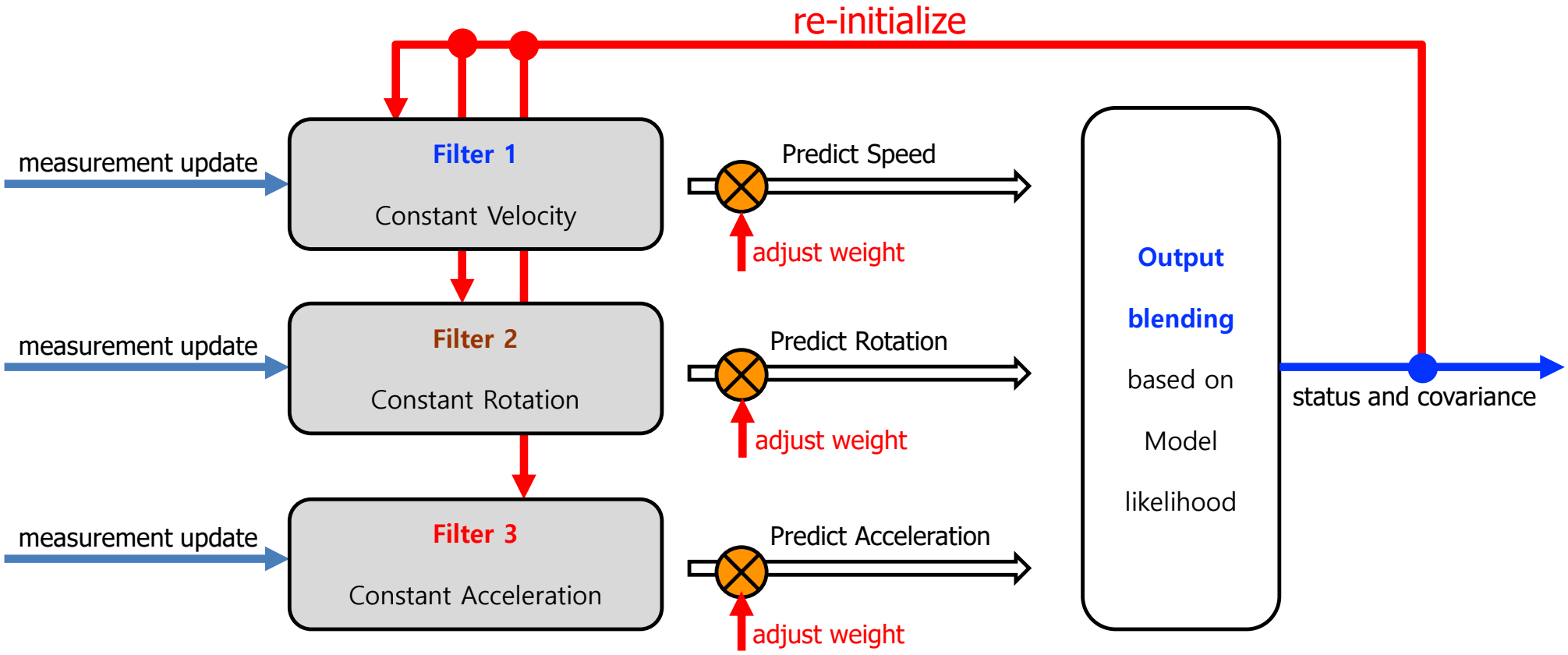
수정(Correct)
직접 현재 위치 및
상태를 측정

Estimation Filter for Prediction & Correction



Interacting Multiple Model Filter

◆ Interacting Multiple Model Filter



Filtering Algorithms

◆ Comparisons of Filtering Algorithms

Filter	Description
Bayesian Filter	▪ The Bayesian filter is a framework for recursive state estimation that utilizes the Bayes theorem, Markov assumption, probability theory, and Bayesian networks to do so. A Bayes filter allows you to estimate a probability density function of states over time using observations. ▪ Kalman filter is a **special case** of the Bayes filter where the dynamics and sensory model is linear Gaussian
Kalman Filter	▪ Recursively update an estimate of the state and find the innovations driving a stochastic process given a sequence of observations ▪ Kalman filter accomplishes this goal by linear projections ▪ Kalman filter can be used for state estimation for non-linear system
Extended Kalman Filter	▪ the **extended Kalman filter** (**EKF**) is the nonlinear version of the Kalman filter which linearizes about an estimate of the current mean and covariance. ▪ In the case of well defined transition models, the EKF has been considered the *de facto* standard in the theory of nonlinear state estimation, navigation systems and GPS.
Particle Filter	▪ Recursively update an estimate of the state and find the innovations driving a stochastic process given a sequence of observations ▪ Particle filter accomplishes this goal by a sequential Monte Carlo method ▪ Particle filter can perform better for a system with non-Gaussian noise ▪ depending on the number of particles, particle filters are more accurate although being computationally more expensive.

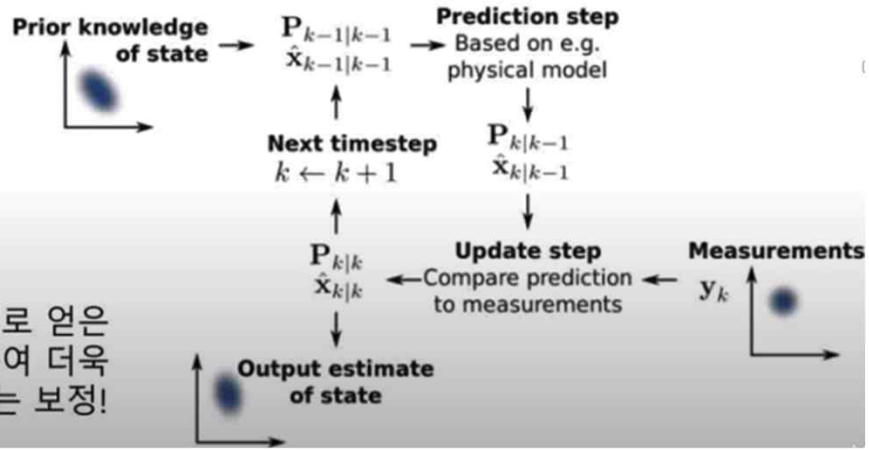
Kalman Filter 기반 위치 추정 (Localization)

◆ Kalman Filter 기반 위치 추정

- **칼만 필터 (Kalman filter)**
  - 잡음이 포함되어 있는 선형 시스템에서 대상체의 상태를 추적하는 재귀 필터
  - 베이즈 확률 기반

- 예측(Prediction)
  - 모델을 상정하고 이 모델을 이용하여 이전 상태로부터 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!



Kalman Filter의 관련 참고자료

◆ Kalman Filter

- https://www.youtube.com/watch?v=LioOvUZ1MiM
- https://www.youtube.com/watch?v=ul3u2yLPwU0
- https://www.youtube.com/watch?v=IKuV6fAvuoc
- C++ & Arduino Tutorial - Implement a Kalman Filter - For Beginners, https://www.youtube.com/watch?v=ruB917YmtgE
- Implementation of Kalman Filter in Python, https://www.youtube.com/watch?v=m5Bw1m8jJuY

◆ Extended Kalman Filter

- https://en.wikipedia.org/wiki/Extended_Kalman_filter

◆ OpenCV의 Kalman Filter

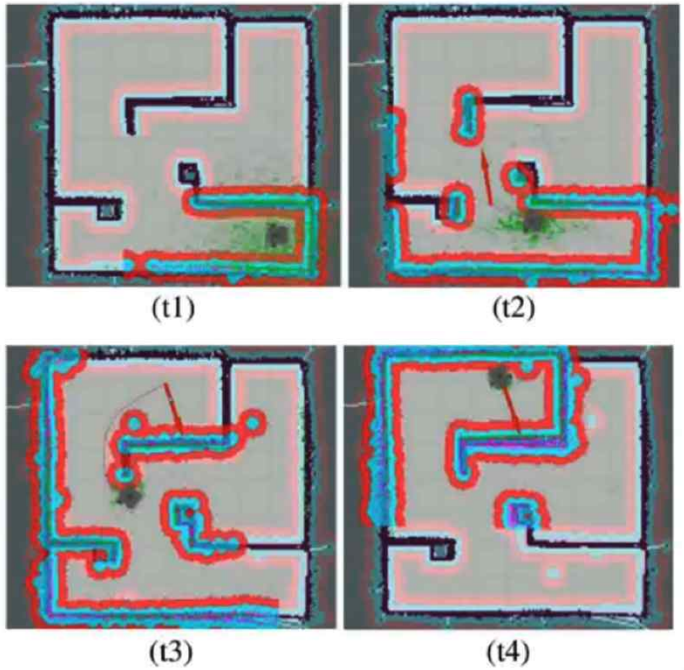
- cv2.KalmanFilter()
- https://www.youtube.com/watch?v=VJ-gB9izieU

Particle Filter 기반 위치 추정 (Localization)

◆ Particle Filter 기반 위치 추정

- 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)



SLAM (Simultaneous Localization and Mapping)

SLAM의 기능

◆ SLAM의 기능

- 참고: https://kr.mathworks.com/discovery/slam.html
- SLAM(동시적 위치추정 및 지도작성)은 자율주행 차량에 사용되어 주변 환경 지도를 작성하는 동시에 차량의 위치를 작성된 지도 안에서 추정
- SLAM 알고리즘을 통해 차량은 미지의 환경에 대한 지도를 작성할 수 있음
- SLAM으로 작성된 지도 정보를 사용하여 경로 계획 및 장애물 회피 등의 작업을 수행

SLAM (Simultaneous Localization and Mapping) 기능

◆ Localization

- 현재 위치를 측정/추정

◆ Mapping

- 주변 환경에 대한 map을 작성

◆ SLAM

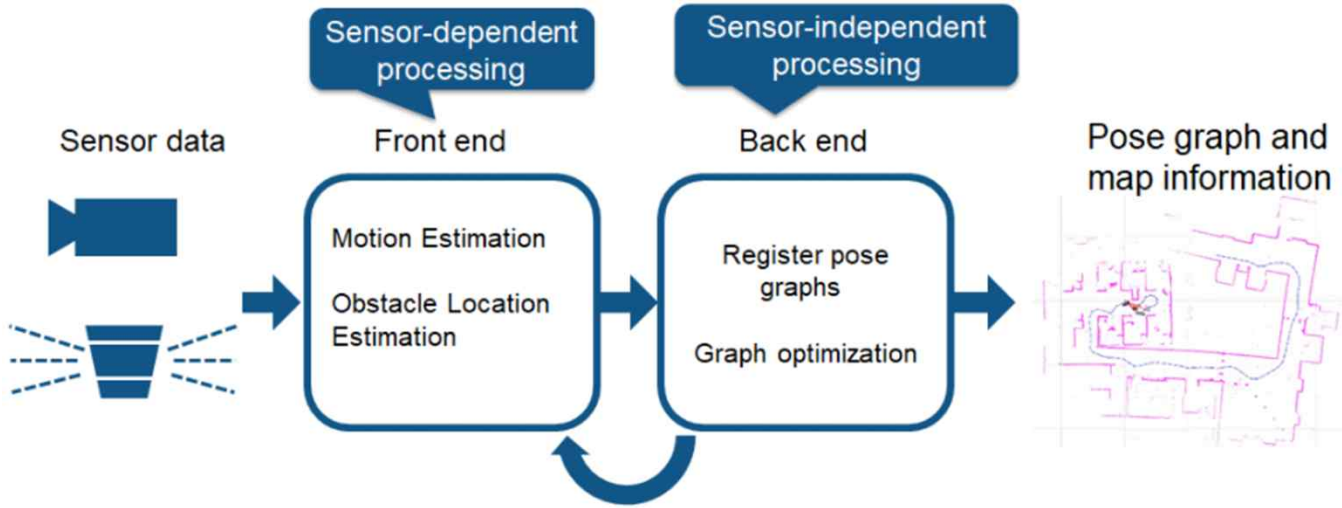
- Localization 과 Mapping을 동시에 실행

일반적인 Graph SLAM 기능 절차

◆ 일반적인 Graph SLAM 기능 절차

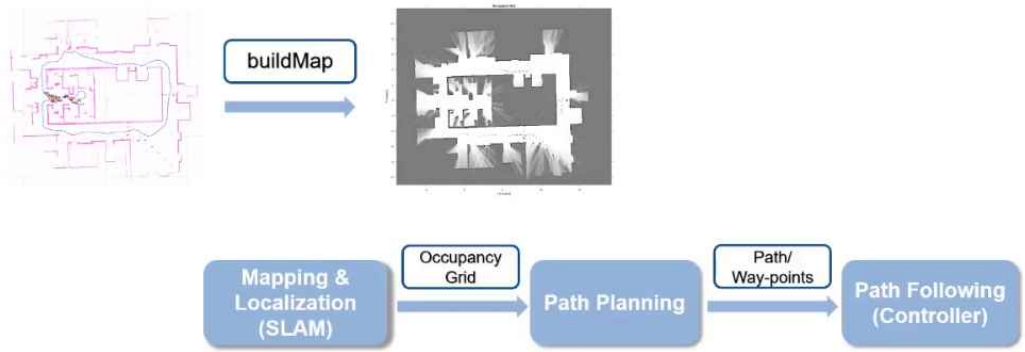
- Local SLAM front end
  - Observation -> local map, feature map
  - Motion -> pose trajectory

- Global SLAM, backend
  - Loop closure detection
  - Optimization



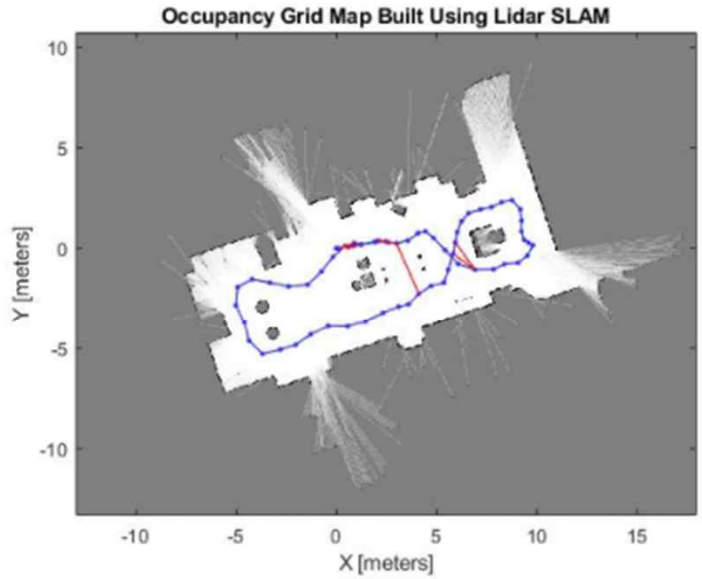
SLAM 관련자료

- **OpenSLAM**
  - platform for SLAM researchers providing information of each project/proposal
  - https://www.youtube.com/watch?v=Ro_s3Lbx5ms
- **How doses SLAM work?**
  - https://www.youtube.com/watch?v=IH_n9bfy-nM
- **SLAM with MatLab**
  - https://www.youtube.com/watch?v=XZxpmS0QuHI

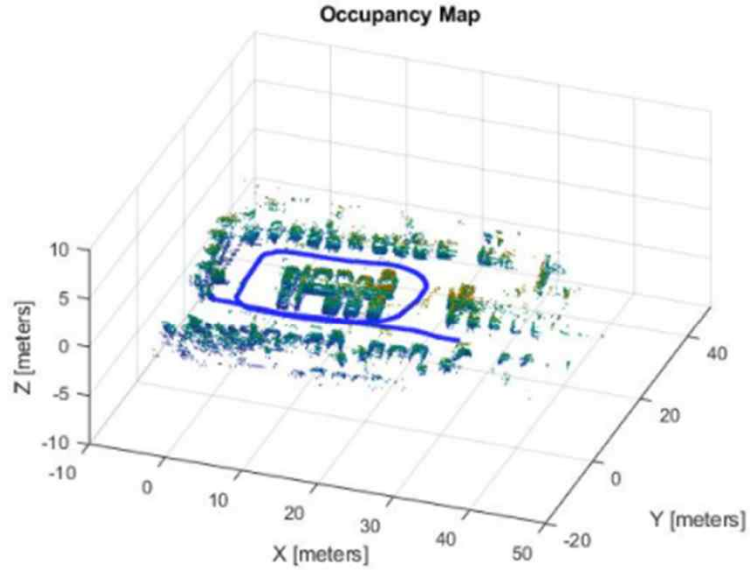


LiDAR 기반 실내 측위 (Indoor Localization)

◆ LiDAR 기반 SLAM



2차원 라이다를 사용한 SLAM



3차원 라이다를 사용한 SLAM

Map 작성 (Mapping) - Gmapping

◆ Gmapping

- http://wiki.ros.org/gmapping
- https://dabit-industries.github.io/turtlebot2-tutorials/06-Gmapping.html
- OpenSLAM에 공개된 SLAM의 한 종류: slam_gmapping ROS node
- 저자: G. Grisetti, C. Stachniss, w. Burgard
- ROS에 패키지로 제공
- 특징: Rao-Blackwellized Particle Filter 사용, Particle 수 감소, grid map 사용
- 하드웨어 제약 사항
  - 계측 센서: 2차 평면 계측 가능 센서 (LRF, LiDAR, Kinect, Xtion 등)
  - 주행 기록계 (odometry)
  - X, Y, theta 속도 이동 명령
  - 차동 구동형 모바일 로봇 (differential drive mobile robot)
  - 전방향 이동 로봇(omni-wheel robot)
  - 직사각형 및 원형의 로봇

SLAM Algorithms

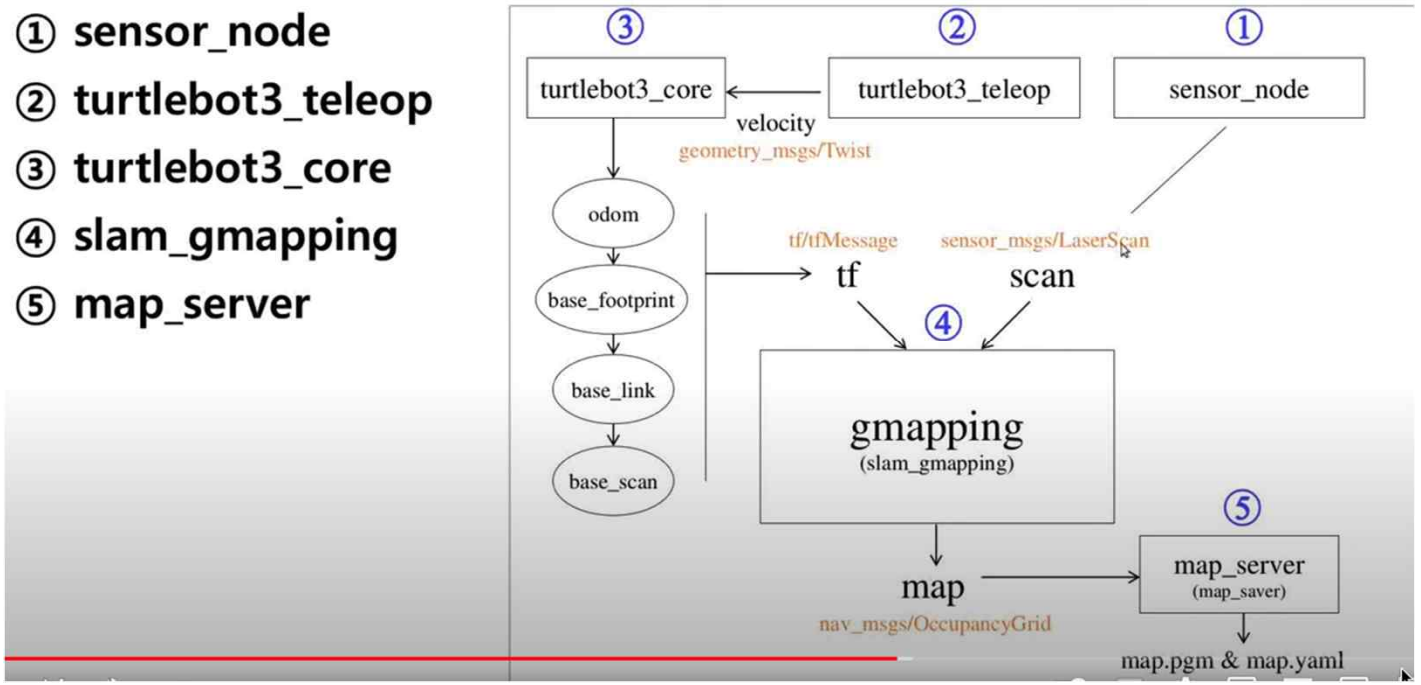
◆ SLAM Algorithms

SLAM Algorithm	Features
Gmapping	▪ Rao-Blackwellized Particle Filter, grid map 사용 ▪ slam_gmapping ROS node로 ROS에 포함 ▪ 2차 평면 계측 가능 센서 (LRF, LiDAR, Kinect, Xtion 등)
Hector-SLAM	▪ based on scan matching algorithm
Cartographer	▪ Cartographer is a scan matching algorithm with loop detection ▪ Cartographer_ros 모듈로 ROS에 포함

(Ref: Evaluation of Modern Laser Based Indoor SLAM Algorithms, https://fruct.org/publications/fruct22/files/Kri2.pdf
2D Lidar-Based SLAM and Path Planning for Indoor Rescue Using Mobile Robots,
https://www.hindawi.com/journals/jat/2020/8867937/)

Gmapping 동작

① sensor_node
② turtlebot3_teleop
③ turtlebot3_core
④ slam_gmapping
⑤ map_server



Google Cartographer

◆ Realtime Simultaneous Localization and Mapping in 2D and 3D

- https://opensource.google/projects/cartographer
- https://google-cartographer-ros.readthedocs.io/en/latest/
- https://github.com/cartographer-project/cartographer

- F1TENTH Autonomous Racing: Modern SLAM - Google Cartographer, https://www.youtube.com/watch?v=L51S2RVu-zc
- https://www.youtube.com/watch?v=GzZGl0kzGOM

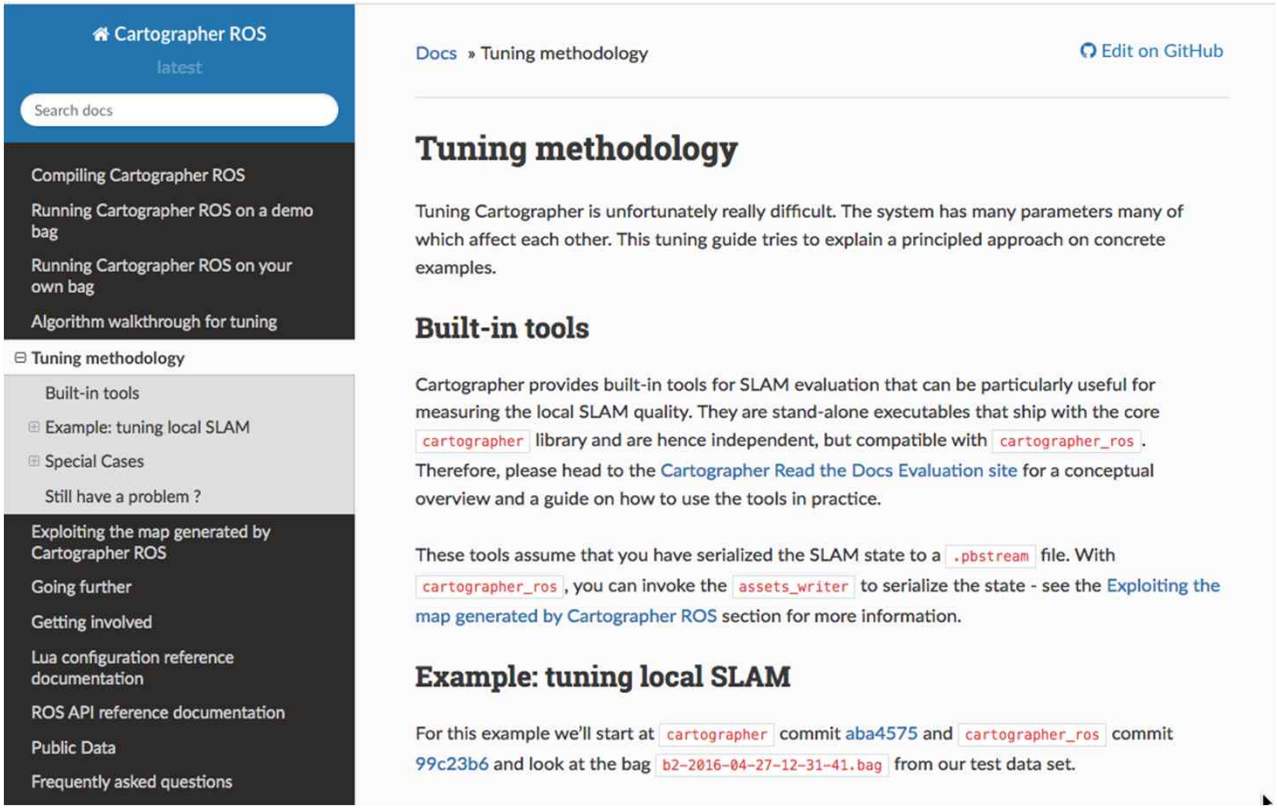
◆ SLAM 가이드 ROS Cartographer

- cython.org

Cartographer ROS

◆ Cartographer ROS

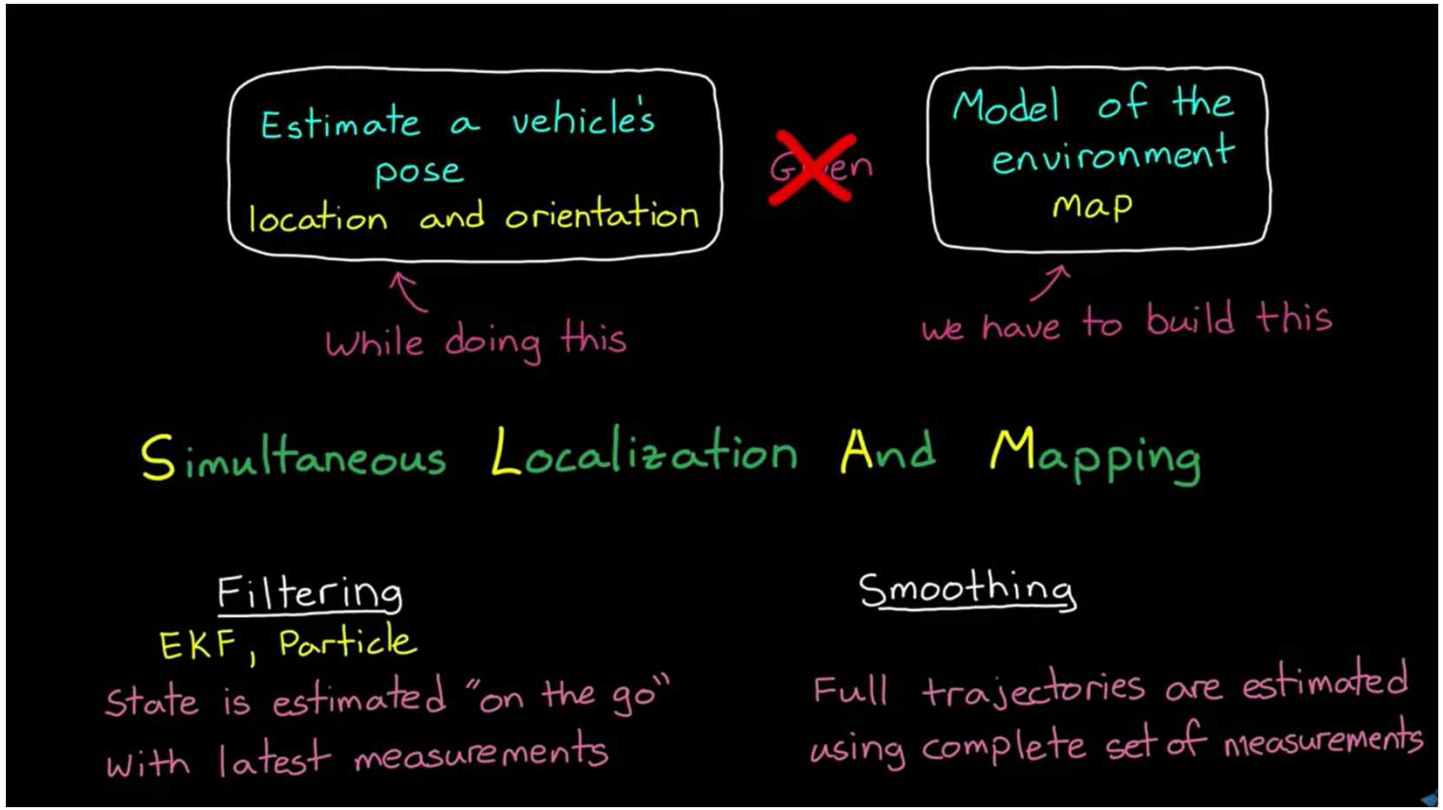
- https://google-cartographer-ros.readthedocs.io/en/latest/tuning.html



SLAM에서 흔히 생기는 문제

- 실제 값과의 상당한 편차를 초래하는 위치추정 오차 누적
- 위치추정 실패 및 지도상 위치 상실
- 영상 처리, 포인트 클라우드 처리 및 최적화에 소요되는 높은 계산 비용

SLAM with Filtering and Smoothing



WiFi SLAM

◆ WiFi SLAM

- https://www.youtube.com/watch?v=kaZ4nlh1KZo
- WiFi SLAM algorithms: an experimental comparison, https://www.cambridge.org/core/journals/robotica/article/abs/wifi-slam-algorithms-an-experimental-comparison/E7E86C5E42BE34E5B1FF0E002F1F7ADB

References

[1] Sebastian Thrun, Wolfram Burgard and Dieter Fox, Probabilistic Robotics, MIT press, August 2005.

[2] Mapping, Localization and Self-Driving Vehicles, MIT Lecture, https://www.youtube.com/watch?v=1kel8U86EVE.

[3] Kirill Krinkin et. al, Evaluation of Modern Laser based Indoor SLAM Algorithms, https://fruct.org/publications/fruct22/files/Kri2.pdf.

[4] Ines V. Stelzer et. al, Comparison of Particle Filter and Extended Kalman Filter Algorithms for Monitoring of Bioprocesses, https://www.sciencedirect.com/science/article/abs/pii/B978044463965350249X.

[5] Eric Ewing et. al, Bayesian and Kalman Filters, http://stefanosnikolaidis.net/course-files/CS545/Lecture6.pdf.

[6] Wi-Fi Fingerprint-Based Indoor Positioning: Recent Advances and Comparisons, IEEE Comm. Surveys & Tutorials, Vol. 18, No. 1, Q1 2016.

[7] simulating SLAM from scratch using python | introduction, https://www.youtube.com/watch?v=2GJuEIh4xGo.

[8] simulating a LIDAR sensor from scratch with python | SLAM SERIES, https://www.youtube.com/watch?v=JbUNsYPJK1U&t=0s.

[9] Feature Extraction from 2D LIDAR data using python 1/2 | SLAM SERIES, https://www.youtube.com/watch?v=6mivXP3rAfg&t=0s.

[10] Feature Extraction from 2D LIDAR data using python 2/2 | SLAM SERIES, https://www.youtube.com/watch?v=oux9LfdqFm4&t=0s.

[10] Data association for SLAM | coding slam from scratch, https://www.youtube.com/watch?v=ZxaXfahaP2s&t=0s.

[11] Real-time planning and re-planning with SLAM on a mobile robot, https://www.youtube.com/watch?v=O3RKzukyiFQ.

[12] Visual and LIDAR based SLAM with ROS using Bittle and Raspberry Pi, https://www.youtube.com/watch?v=uXpQUIF_Jyk.

[13] 2D Mapping using Google Cartographer and RPLidar with Raspberry Pi 3B+, https://www.youtube.com/watch?v=qNdcXUEF7KU.

[14] RPLidar and Hector SLAM for Beginners | ROS Tutorial #8, https://www.youtube.com/watch?v=Qrtz0a7HaQ4.

[15] Using ROS SLAM on the REAL ROBOT like ATCart, https://www.youtube.com/watch?v=Ng54fg8k8JE.

[16] Robot Wall Following Demo | ROS | Lidar | Raspberry Pi | Arduino, https://www.youtube.com/watch?v=oieffIypkoU.

References

[17] Robotics Weekends #2 - CbBot. Experimental ROS robot platform with SLAM and Gmapping, https://www.youtube.com/watch?v=MqSB-T0HsFo.

[18] ROS and Raspberry Pi for Beginners | Tutorial #0 - Topics Packages RosMaster, https://www.youtube.com/watch?v=iLiI_IRedhI.

[19] Probabilistic Algorithms in Robotics - Sebastian Thrun, http://robots.stanford.edu › thrun.probrob.pdf

[20] pygame documentation, https://www.pygame.org/docs/.

[21] YD LiDAR X4 User Manual, https://www.ydlidar.com/Public/upload/files/2021-08-20/YDLIDAR%20X4%20Lidar%20User%20Manual%20V1.3.pdf.

[22] YDLIDAR 레이저 거리 스캐너 사용 설명서, https://manuals.plus/ko/ydlidar/laser-range-scanner-manual.