1-1. 라이다

1) 라이다의 감지방식

- LiDAR(Light Detection and Ranging)의 약자로 거리측정을 위해 빛과 “비행시간=TOF”의 개념을 사용
- 라이다는 채널 수 만큼 레이저 펄스를 쏘는데(1초당 몇십만개이상 > 그래서 포인트 클라우드가 많음) 이 레이저 펄스를 이용하여 센서 주변의 구조를 인식하며, 이후 라이다는 포인트 클라우드를 만들어 3D맵을 만듦

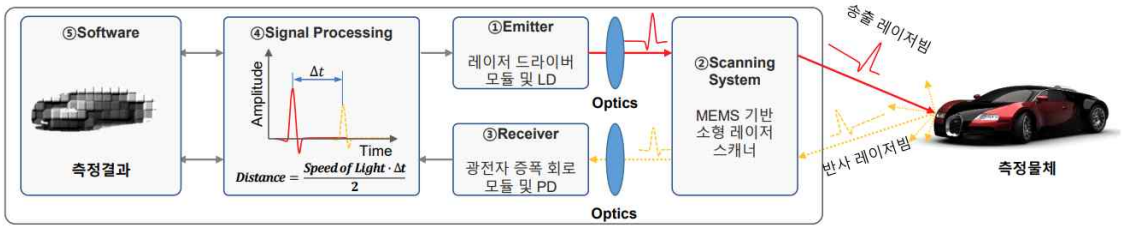
2) 라이다 구성 용어

- FOV(Field of View) : 시야각으로 기계식 라이다는 넓은 FOV를 가짐 (ex 기계식인 벨로다인 수평 FOV는 360도, 수직 FOV는 30도)
- Resolution(분해능) : 서로 떨어져 있는 두 물체를 서로 구별할 수 있는 능력으로 라이다는 각도 분해능의 성능이 중요함
- 각도 분해능은 먼 거리에서 농구공과 같은 물체를 감지·분류할 수 있는지, 아니면 단지 물체의 존재 여부만 감지할 수 있는지를 결정
- 물론 장거리의 물체를 탐지 할수록 공간 분해능의 성능이 떨어지는 것은 당연
- 파장 대역 905nm vs 1550nm
  - 905nm : 양산 쉽고, 가격 낮고, 소모 전력 낮음 / 태양광에 의한 노이즈
  - 1550nm : 탐지 거리가 김(905는 200m 얘는 500m) / 공기 중 수분 영향

3) 구동방식에 따른 라이다 종류

- 기계식 스캐닝 : 모터로 물리적으로 센서를 회전시킴
  - 수평 FOV넓음(센서 회전하니까) / 크고, 비싸고, 내구성 약함
- 고정식 : MEMS(기계식 스캐닝을 전자기계식으로 대체), 플래시(사진), OPA(Optical phase array)(광학-렌즈사용)

4) 라이다의 프로세서



① Emitter : 레이저를 송출하는 역할
② Scanning System : 송출한 레이저를 주변 환경에 맞게 조사하는 역할
③ Receiver : 반사되어 들어오는 빛을 다시 측정하는 역할
④ Signal Processing : Emitter ~ Receiver 까지 걸린 시간을 이용하여 각 포인트 마다의 거리를 계산하는 역할
⑤ Software : Signal Processing을 통해 얻은 각 Point 정보를 이용하여 주변 물체에 대한 측정 결과를 제공
<1-1. LiDAR의 GIF 보여드리기>

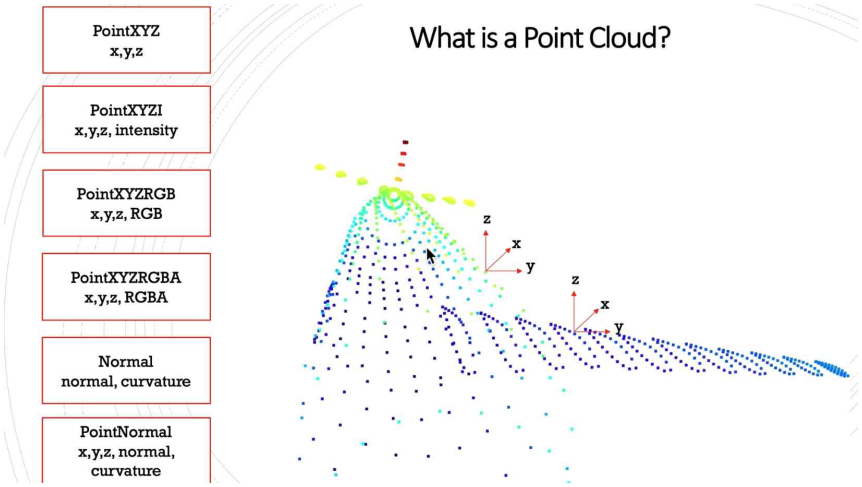
5) 카메라 vs Radar vs LiDAR

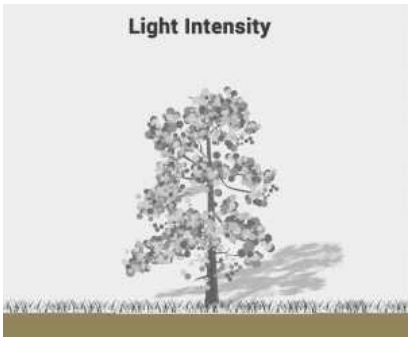
	카메라	레이더	라이다
원리	영상을 통해 시각적인 정보를 인지한다.	전파를 이용해 물체와의 거리를 측정한다.	빛(Light)을 이용해 물체와의 거리를 측정한다.
장점	물체를 구분할 수 있다. 비용이 효율적이다. 색상을 인지할 수 있다.	장거리에 있는 물체와의 거리를 측정할 수 있다. 날씨에 영향을 많이 받지 않는다. 가려져 있는 물체를 인지할 수 있다 (투과 가능).	레이더에 비해 작은 물체도 감지할 수 있다. 정확한 단색 3D 이미지를 제공할 수 있다. 형태 인식이 가능하고 정밀도가 높다.
단점	물체와의 정확한 거리를 알기 어렵다. 날씨에 영향을 많이 받는다.	작은 물체 식별이 어렵다. 정밀한 이미지를 제공하지 못한다. 물체의 종류를 판독할 수 없다. 최대 측정 거리에 반비례하여 측정범위가 줄어든다.	가격이 아직 비싸다. 레이더와 비교 시 탐지 거리가 비교적 짧고 날씨 등의 기상상황에 민감하다. 가려져 있는 물체는 감지할 수 없다 (투과 불가능).

1-2. Point Cloud

1) 포인트 클라우드

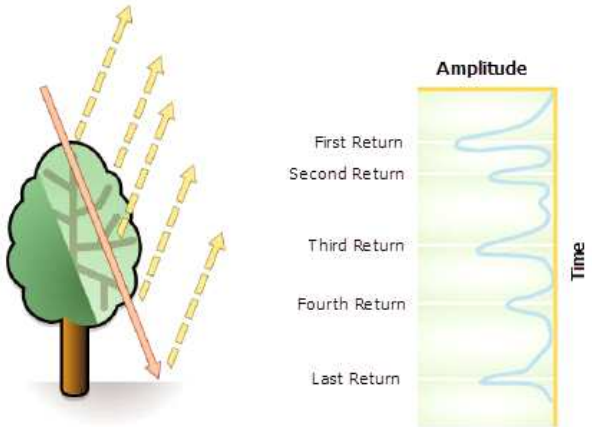
- 포인트 클라우드는 3차원 공간상에 퍼져 있는 여러 포인트(Point)의 집합(set cloud)를 의미

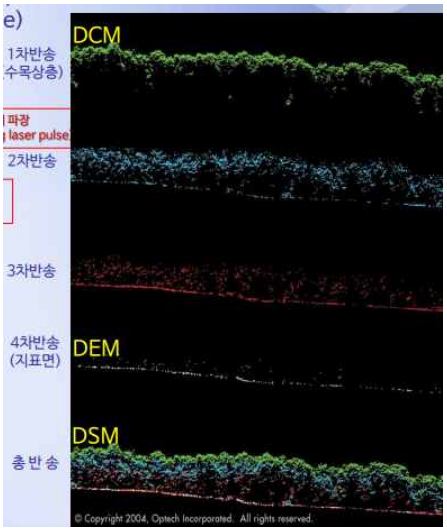




- 포인트 클라우드는 대표적으로 위 그림과 같이 6개의 포맷으로 저장하며 x,y,z와 intensity 정보를 가장 많이 사용
- 색은 Height에 따라 다름 (고-red, 저-blue)
- intensity는 반사율을 의미하며 반사를 하는 표면 물체의 구성에 따라 다름
- 레이저 펄스가 통과하는 매질, 물체의 표면과 펄스가 만나는 각도, 대상 물체 표면의 반사율에 따라서 돌아오는 신호의 세기가 결정

2) LiDAR Return





- LiDAR에서 방출되는 레이저 펄스는 식물, 건물, 다리 등에서 반사됨
- 방출된 하나의 레이저 펄스는 하나 또는 여러 개의 return으로 return됨
- 방출된 레이저 펄스는 반사 표면이 있는 만큼 많은 반사로 분할됨
- 처음으로 return된 레이저 펄스는 가장 중요한 return이며 나무 꼭대기나 건물 꼭대기 같은 풍경 등 가장 높은 feature와 연관
- 만약 첫 번째 return이 지표면을 나타내면, 이 경우 LiDAR는 하나의 return만 감지
- 다중 return은 레이저 펄스의 풋프린터 내에서 여러 물체의 고도를 감지
  - foot print ? 레이저 샘플링 영역의 크기를 설명하는 중요한 매개변수인 라이다 풋프린트는 스캐닝 지오메트리와 로컬 지형에 따라 다름
- 일반적으로 중간 return은 식물 구조에 사용되며 마지막 return은 bare-earth 지역 모델에 사용
- 여기서 Bare earth LiDAR is digital elevation data of the terrain surface consisting of irregularly spaced points, each with x/y location coordinates and z-value.

3) 전처리, 후처리를 통해 연석, 도로 검출등 여러 어플리케이션이 가능

1-3. Velodyne vs Ouster

- 공통점 : Dual return > 더 정확, 받는 포인트 수 多 : 정확한 3D 이미지 데이터를 제공할 수 있음
- 차이 (*Range와 FOV는 반비례 관계) / Ouster는 여러개 설치해야됨 Solid State LiDAR라서 FOV작음

	Velodyne	Ouster
Channel	16	최소 32
구동방식	Mechanical Scanning	Solid State
Range	100m	종류에 따라 50~240m
Vertical FOV	360도	종류에 따라 22.5~90도
Vertical Angular Resolution	2도	종류에 따라 0.18~0.7도

1-4. LiDAR on SLAM

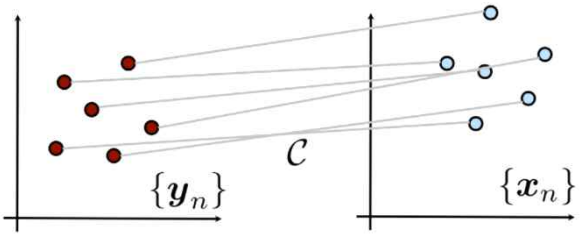
1) ICP(Iterative Closest Point)

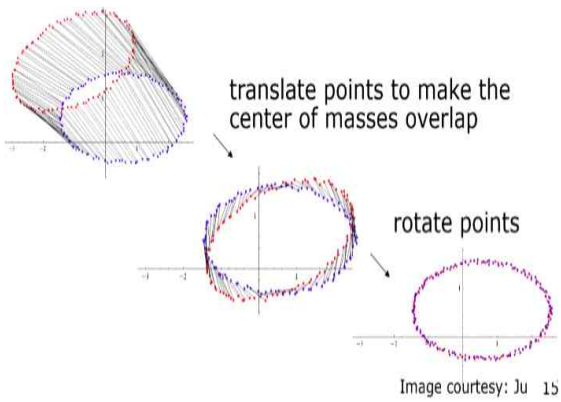
1) LiDAR 센서 데이터의 정합(registration)을 이용하여 움직이는 물체의 궤적을 결정하는 알고리즘을 통상적으로 LiDAR Odometry라고 하며 가장 기본적인 정합 방법은 ICP(Iterative Closest Point)와 NDT(Normal Distributions Transform)가 있음 > Loop closure에는 ICP, 가까운 sequence의 registration은 NDT추천

- 물론 ICP(Point to Point기법)뿐 아니라 업그레이드 G-ICP(Point to Plane)와 NDP(분포 기반)의 업그레이드 NDT-omp(멀티 쓰레드로 속도 2배 빠름)이 있음

2) ICP : 한 대상물에 대해 다른 지점에서 스캔된 두 개의 포인트 클라우드가 있을 경우, 이 두 개의 데이터를 퍼즐처럼 합쳐, 정합(registration)하는 알고리즘으로 실내외 지도를 만드는 데 핵심적으로 활용

3) ICP 방법 : 일반적인 점 기반 방법은 선택된 가장 가까운 점에 따라 두 점군 사이의 변환 적용 후, 거리 함수를 최소화





- 포인트 클라우드 point 기준 정합 방법은 수학적으로, 두 Point Cloud의 대응관계를 알 때와 모를 때로 나눠짐

* 대응 관계 알 때 : Translation값을 구하기 위해 두 포.클의 질량 중심을 일치시키고 이동량 계산 이후 Rotation값을 구하기 위해 SVD(Singular Value Decomposition, 특잇값 분해)을 수행

* 근데 ICP는 대응관계 모르기에 초기값을 대략적인 대응관계로 설정하고 품
  - 보통 대응관계를 구할 때는 x1n에서 각각의 point에 대해, 가장 가까운 거리에 있는 y1n의 하나의 점과 매칭을 통해 대응관계를 만듦

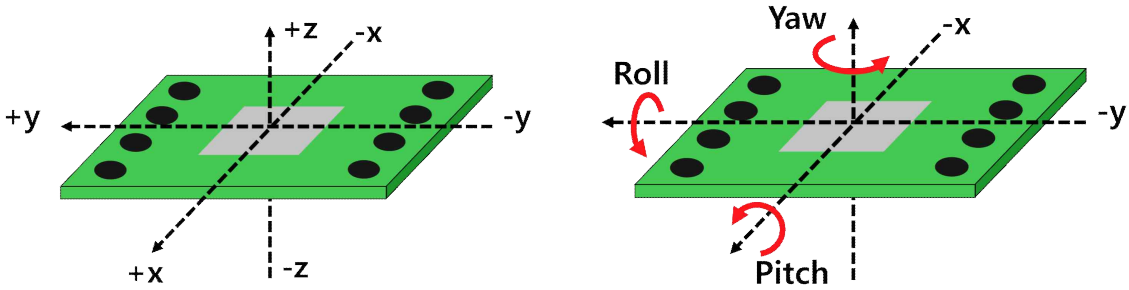
- 구해진 Rotation R과 Translation T를 활용해 x2n=R*x1n+t와 y1n을 맞추어 비교
- 이 때 x2n와 y1n의 차이를 Error로 정의하고 Error값이 원하는 Threshold값보다 적어질 때 까지 진행
- But, 계산 많고 초기 실수시 엄청난 오차 > 부분 Point Cloud 활용 등 다양한 Performance에 초점이 맞추어져 있음 (Speed, Stability, Oulier, Noise 등)

2-1. IMU

1) MEMS(Micro-Electro-Mechanical Systems)

- 기계식 센서를 반도체 안에 구현한 초소형 센서들을 말하며 예시 IMU(가속도s+자이로s+지자계s)
- MEMS센서들은 주변 전자장비, 철 등에 영향을 받으므로 주의 > 그 때 출장

2) IMU(Inertial Measurement Unit) : 관성 측정 장치



① 가속도 센서(Acceleration Sensor)

- 움직임의 변화에 따른 가속도의 변화(단위: m/s^2)감지
- 초기값을 계산할때 중력 가속도를 분해하여 얼마나 기울어졌는지를 측정 > 이동시 이동가속도붙어서 정지시만 사용

② 자이로스코프(Gyroscope)

- 중력이 가해질 때 진동 속도가 변하는 것을 각속도로 계산해, 질량과 진동 속도를 통해 값을 측정하여 검출
- 이 각속도(rad/s)에서 시간당 몇도(degree)를 회전했는지가 필요할 때 사용 > 이때 적분사용해야됨 > 누적오차
- 온도에 따라 측정값이 변화하므로 온도센서 추가 내장해서 오차 보상

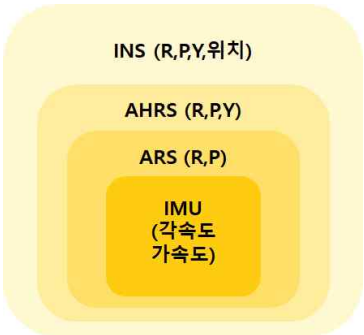
③ 지자기 센서(Magnetometer)

- 지자기(magnet)를 측정하며 자북을 기준으로 자기선속의 세기를 측정하여 자북을 기준으로 얼마나 틀어졌는지를 측정
- 자이로스코프 오차 보정용

3) IMU로 Roll, Pitch, Yaw 구하기 > AHS, ARHS

- Roll, Pitch는 가속도계로 OK / Yaw 알아볼려면 자이로 적분해야됨 > 누적오차 발생 > 지자계로 오차 보상하고 상호보완필터나 칼만필터등 필터를 사용 > AHRS

2-2. ARS, AHRS, INS



1) ARS모듈 (R,P and initial Y)

- 가속도계와 자이로(각속도)계를 이용해 필터를 통한 Roll, Pitch 출력 (보정안된 자이로 Yaw도 출력)

2) AHRS모듈 (R,P,Y)

- IMU의 가속도계, 각속도계, 지자기계에서 측정한 값을 바탕으로 MCU에서 Roll, Pitch, Yaw를 계산하고 이를 칼만 필터 등을 이용해 처리한 뒤 출력

3) INS (R,P,Y,Position) : 관성항법장치

- INS는 AHRS에 추가적으로 정밀한 가속도계가 있고, 초기 위치로부터 변화한 위치를 가속도계 측정치에 중력을 보상하고 두 번 적분하여 계산하는(가속도 두 번 적분하면 위치이므로) 등의 정밀한 항법 알고리즘을 내장
- 관성항법은 초기에 출발점 좌표를 기준으로 계산하므로 정확히 아는 것이 매우 중요
- INS도 결국 오차가 누적되므로 중간에 오차를 보정해줄 여러가지 방법이 고안 > GPS와 같이 사용 (상호보완)

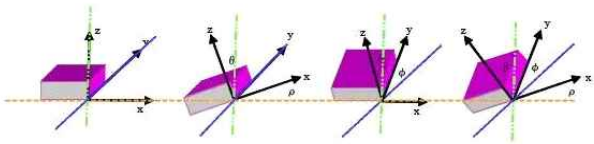
2-3. IMU on SLAM

1) Why IMU on SLAM

- IMU sensor는 주변 환경의 상태와 관련없이 로봇 자기 자신의 state값을 읽을 수 있는 센서이기 때문

2) Calculation IMU

- 오일러로 Roll, Pitch, Yaw 구하기



위의 그림으로부터 변화각 Roll, Pitch값을 구하는 공식은 다음과 같다.

$$\rho = \arctan\left(\frac{A_X}{\sqrt{A_Y^2 + A_Z^2}}\right)$$

$$\phi = \arctan\left(\frac{A_Y}{\sqrt{A_X^2 + A_Z^2}}\right)$$

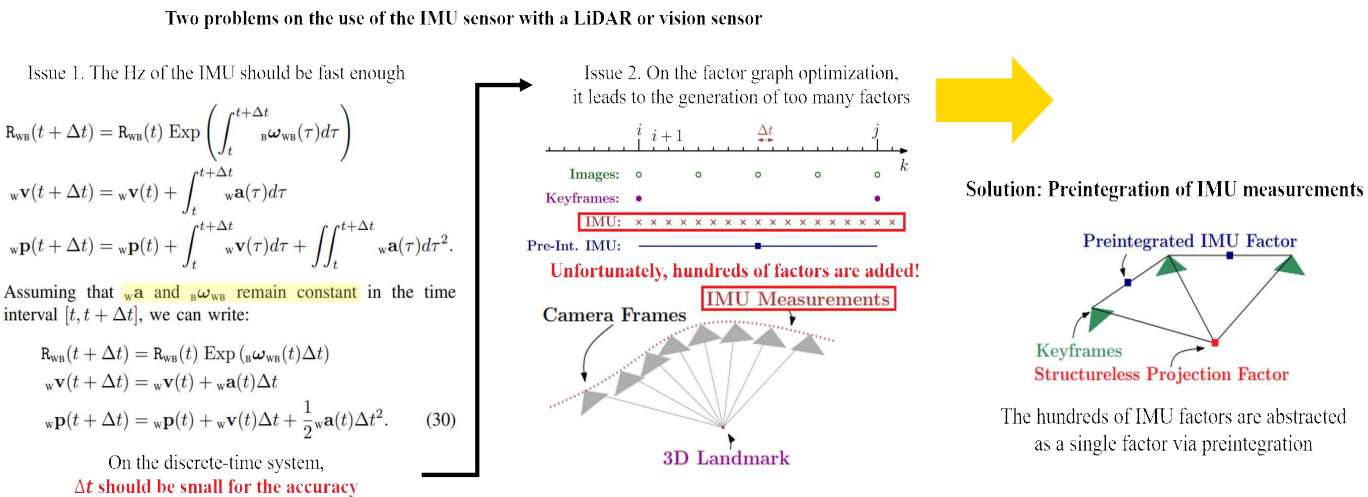
$$\alpha = \arctan\left(\frac{Y}{X}\right)$$

Eq. 1) Heading (azimuth α) in degrees.

- 짐벌락현생 때문에 roll, pitch각이 90도 이상으로 변화할시 오일러각이 이상한 값으로 변함
- 짐벌락? 상위 기준 축에 의해 하의 기준 축이 씹히는 현상으로 세 축이 종속적
- 때문에, 오일러각을 쿼터니언으로 변환하여 사용
- IN, Lio-SLAM imuPreintegration.cpp에서는 x,y,z를 받아 쿼터니안으로 tf해주고 이후 get rpy를 통해 rpy를 얻음

```
Eigen::Affine3f odom2affine(nav_msgs::Odometry odom)
{
    double x, y, z, roll, pitch, yaw;
    x = odom.pose.pose.position.x;
    y = odom.pose.pose.position.y;
    z = odom.pose.pose.position.z;
    tf::Quaternion orientation;
    tf::quaternionMsgToTF(odom.pose.pose.orientation, orientation);
    tf::Matrix3x3(orientation).getRPY(roll, pitch, yaw);
    return pcl::getTransformation(x, y, z, roll, pitch, yaw);
}
```

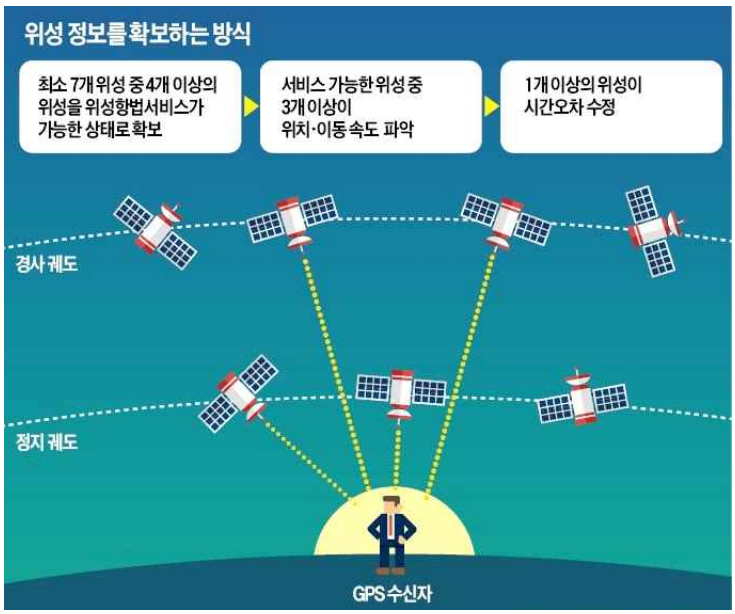
3) IMU Preintegration in Lio-SAM 해석



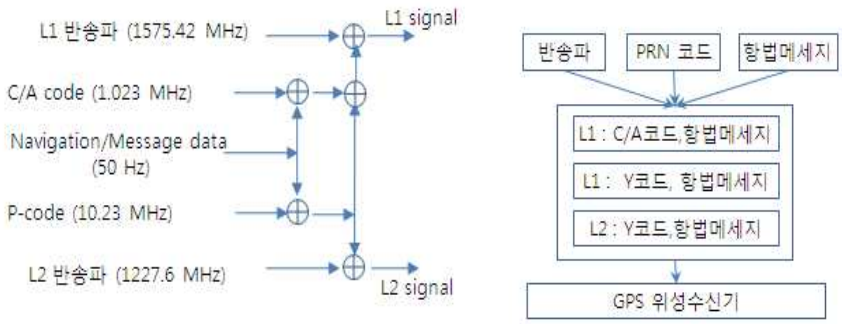
3. GPS / DGPS / RTK

1) GNSS(Global Navigation Satellite System)

- GNSS는 위성측위시스템을 말하며 미국, 러시아, 유럽, 그리고 중국 총 4개국에서 운영하는 위성측위시스템
- 이때, 미국의 GNSS가 GPS를 뜻함
- 러시아(GLONASS), 유럽(GALILEO), 중국(BEIDOU-2)



- 의사거리 : 위성과 지구에 존재하는 GPS 수신기 사이의 대략적인 거리를 의미하며 위성 3개로 삼변측량을 통해 위치를 특정할 수 있음 하지만, 이 거리에는 주요한 오차인자로 사용자 시계오차에 의한 거리오차가 포함되어 있음
- 위와 같이 위치, 이동 속도는 최소 3개의 위성이 필요하지만 실제로는 시간 오차를 보정하기 위해 4개 사용
- 위성에서 송출되는 신호는 세가지 성분으로 구성



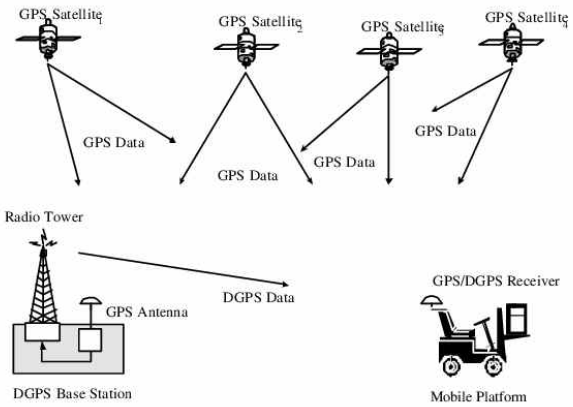
① 반송파 : L1(1575.42MHz), L2(1227.60MHz, 군사용) 두 가지 주파수 사용

② PRN : 위성의 식별과 의사거리 측정을 위해 사용 > 반송파에 섞여지는 의사랜덤 코드

- C/A CODE(=SPS)로 민간용, P CODE(=PPS)로 군용

③ 항법신호 : 위성시계 계수 및 위성배치 및 위성의 건강상태 등의 정보 > 반송파에 실려 송출되는 메시지

2) DGPS(Differential Global Positioning System)



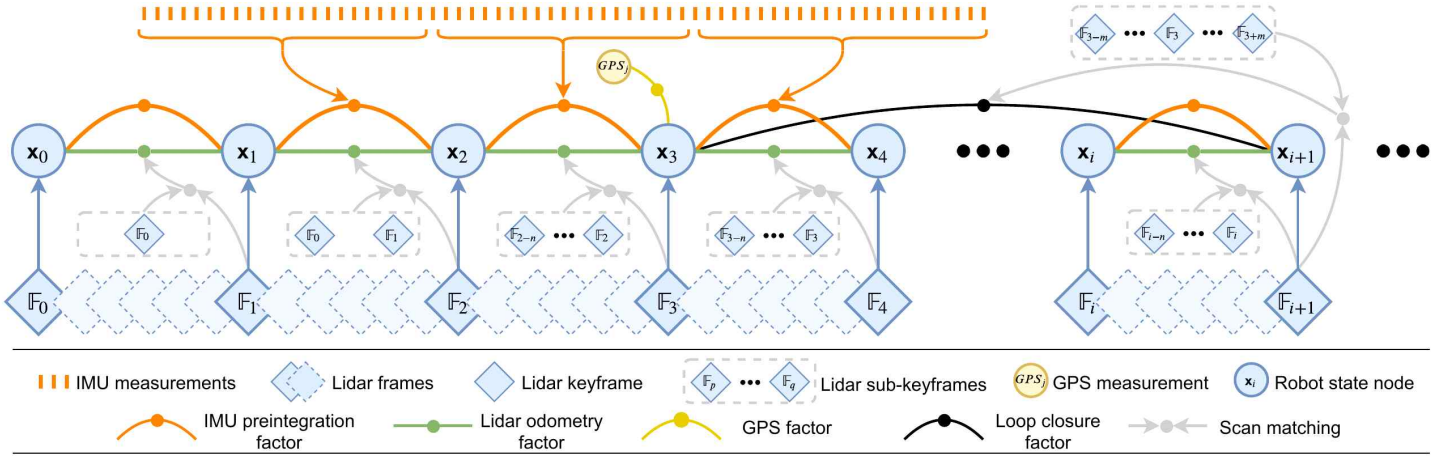
- GPS에 의한 상대 측위 방식의 일종으로 DGPS는 GPS의 오차를 줄이기 위해 서로 가까운 거리에 위치한 두 수신기를 이용해 보정하는 항법시스템
- 이미 알고 있는 기준점을 정하여 이곳에서 정확한 위치 값과 GPS에서 측정한 위치값을 비교하여 GPS에서 발생한 오차값을 보정한 후 그 보정값을 무선통신망(중파 283.5-325KhZ)을 이용하여 이용자에게 실시간으로 알려주는 시스템
- GPS오차는 대략적으로 30M 내외이며 DGPS오차는 1M내외

3) RTK(Real Time Kinematic, GPS)

- DGPS는 위성에서 오는 신호의 시간차를 계산하여 오차를 계산하지만 RTK는 위성에서 오는 신호의 파장 차이를 계산하여 오차를 계산
- 그리고 DGPS는 x,y 보정값이지만(z에 대한 오차가 너무 큼) RTK는 x,y,z 보정값이며 오차도 cm단위

4) LIO-Sam gps 적용

- IMU는 drift 현상이 발생하기 때문에 절대 측정값을 제공하여 드리프트를 제거하는 역할을 함 > GPS
- GPS raw값을 수신하면 로컬 데카르트 좌표계로 변환
- LIDAR프레임의 타임스탬프를 기반으로 GPS측정을 선형으로 보간



- 이렇게 가끔 한 번 IMU를 보정해주는 역할(보조)로 GPS가 사용됨 > 절대 좌표

4-1. Bayes Filter

1) 정의

- Posterior(p(x|z):z가 주어졌을 때의 x)를 Prior(p(x))를 통해서 update 하는 것

1: **Algorithm Bayes_filter**$(bel(x_{t-1}), u_t, z_t)$**:**
2: *for all* x_t *do*
3: $\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1})\ bel(x_{t-1})\ dx_{t-1}$
4: $bel(x_t) = \eta\ p(z_t \mid x_t)\ \overline{bel}(x_t)$
5: *endfor*
6: *return* $bel(x_t)$

- Sensor로 부터 관측되는 sensor observation data z와, 로봇에 전달되는 제어명령 control data u가 있을 때, 로봇의 현재 상태 x(주로 위치, 속도)를 Bayes' Theorem(베이즈정리)에 따라 추정하는 것으로 여기서 t는 현재 time step을 의미
- 3번째 줄 : Predict(물리 모델을 통한 예측 부분에 해당하는 알고리즘)
- 4번째 줄 : Correction(센서값을 통한 예측을 이용하여 현재 상태 보정에 해당하는 알고리즘)

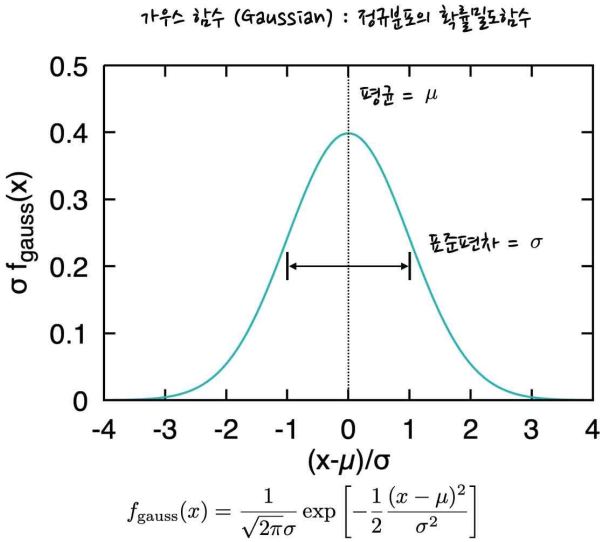
2) 한계

- 3번째 줄의 적분 > 연산이 복잡 or 적분이 불가능한 경우도 존재

3) 해결

- Particle Filter : 랜덤 샘플링 방식으로 적분이 되지 않는 식을 근사화
- Kalman Filter : 가우시안 정규 분포를 통해 적분이 안되는 식을 근사화하여 사용할 수 있음

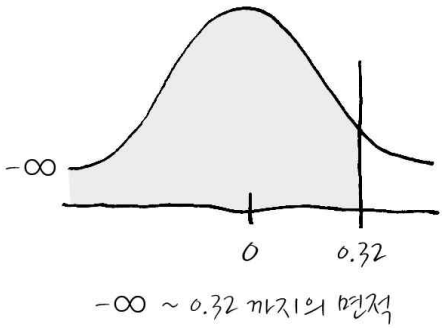
4) 가우시안(정규) 분포



$$f_{\text{gauss}}(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right]$$

- 정규분포는 Mean(평균)과 Variance(분산)만 알면 확률을 특정할 수 있음

가우시안 분포는 아래 그림과 같이 이미 적분이 다 되어 있어서 어느 영역이든지 적분이 가능



Z 값 0.32

Z	0.00	0.01	0.02	0.03
0.0	0.500000	0.503989	0.507978	0.511966
0.1	0.539828	0.543795	0.547758	0.551717
0.2	0.579260	0.583166	0.587064	0.590954
0.3	0.617911	0.621720	0.625516	0.629300
0.4	0.655422	0.659097	0.662757	0.666402
0.5	0.691462	0.694974	0.698468	0.701944
0.6	0.725747	0.729069	0.732371	0.735653
0.7	0.758036	0.761148	0.764238	0.767305
0.8	0.788145	0.791030	0.793892	0.796731
0.9	0.815940	0.818589	0.821214	0.823814

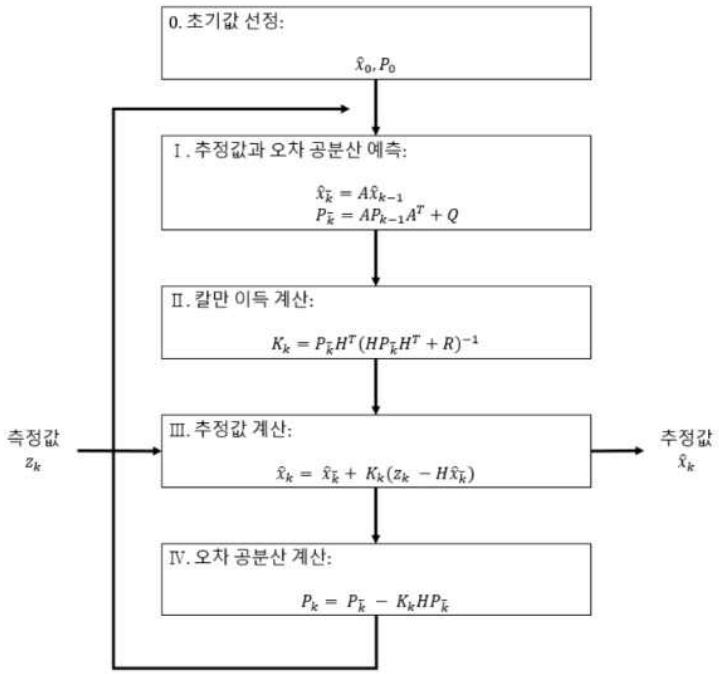
4-2. Kalman Filter

- 칼만 필터(KF, Kalman Filter)는 선형 가우시안 시스템에서 필터링 및 예측을 위한 기술로 시스템의 확률 분포가 Gaussian

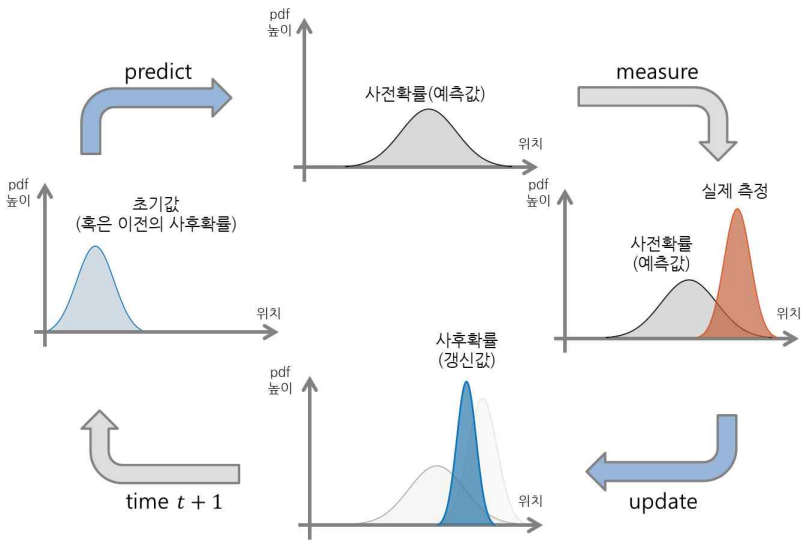
- 칼만 필터는 흔히 노이즈 제거에 사용되며 두 가지 가정을 만족

(1) 관찰값이 상태(state)의 현재 함수

(2) 다음 상태가 이전 상태의 선형 함수



출처 : 칼만필터의 이해



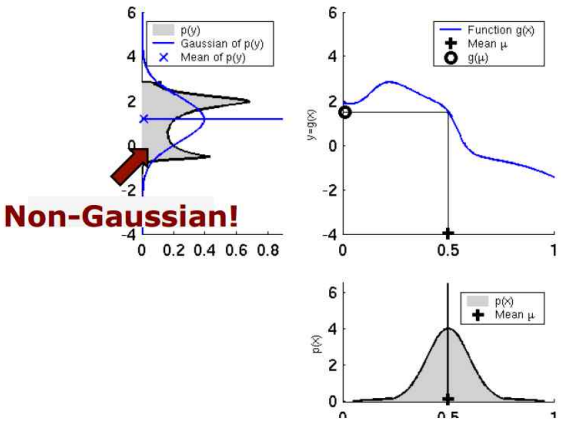
4-3. Extended Kalman Filter

1) 이론

- KF는 선형 Gaussian 모델의 경우이며, EKF는 '비선형' Gaussian 모델
- EKF는 아래와 같이 기존 KF의 선형 모델을 비선형 함수인 g와 h로 바꿈으로써 비선형으로 확장한 모델

$$x_t = g(u_t, x_{t-1}) + \epsilon_t \quad \leftarrow x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$
$$z_t = h(x_t) + \delta_t \quad \leftarrow z_t = C_t x_t + \delta_t$$

- 비선형일 때 추정값 가우시안 안됨



> 입력이 가우시안이지만 함수 g가 비선형이라 결과 가우시안X

2) 비선형 > 선형

① Taylor 근사법

- 무한히 미분되는 초월함수의 경우,(ex/ a^x, cosx, sinx, logx 등) 특정한 x값 이외에는 함숫값을 찾기 어려움
- 이럴 때 미분을 이용하여 찾아낼 수 있는 원래의 함수와 매우 근사한 다항함수를 테일러 급수
- '접선'을 통해 함수를 근사하는 선형 근사(linear approximation)를 일반화한 다항함수 형태라고 생각하면 이해하기 쉬움

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^k(a)}{k!}(x-a)^k$$
$$= \sum_{n=0}^{k} \frac{f^n(a)\,(x-a)^n}{n!}$$

- **Prediction:**

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + \underbrace{\frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}}}_{=: G_t} (x_{t-1} - \mu_{t-1})$$

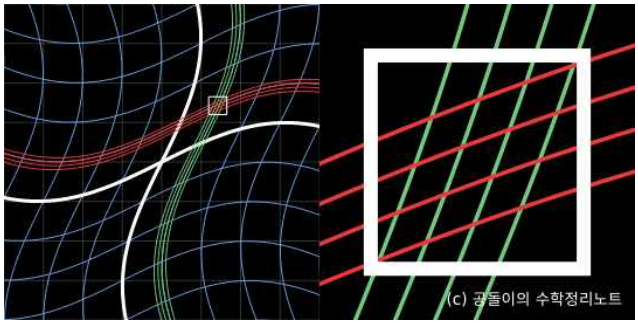
- **Correction:**

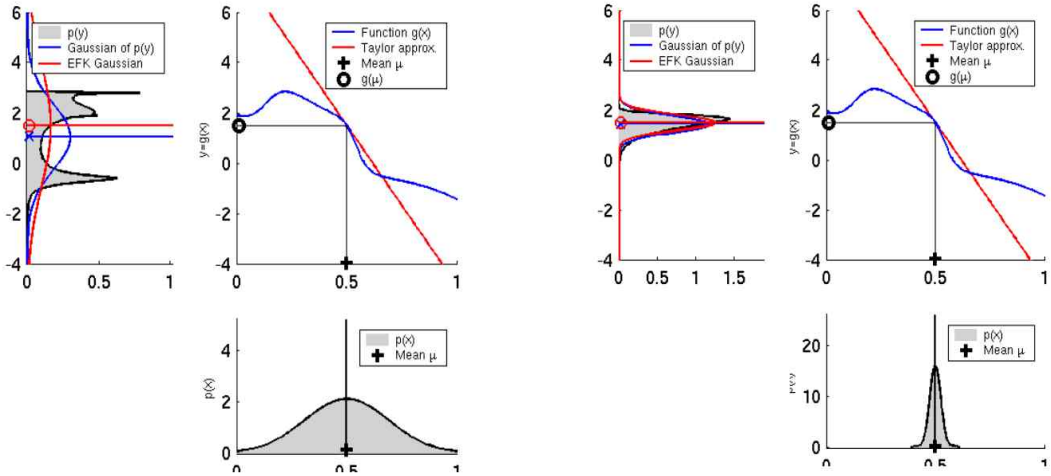
$$h(x_t) \approx h(\bar{\mu}_t) + \underbrace{\frac{\partial h(\bar{\mu}_t)}{\partial x_t}}_{=: H_t} (x_t - \bar{\mu}_t)$$

Jacobian matrices

② Jacobian

- Taylor 근사법을 이용해, 선형 근사화된 모델은 위와 같고, 이 때 비선형 함수를 xt로 편미분하여 matrix를 생성하는데 이 행렬을 Jacobian이라고 함
- 위 식 자체는 테일러를 이용해 선형근사화 시킨거고, 이 때 비선형 함수를 편미분(선형 근사화)하여 matrix가 생성되는데 이 행렬을 자코비안 행렬이라고 정의된거임
- Gt와 Ht는 nxn인 행렬로 여기서 n은 상태의 차원을 의미
- 자코비안 행렬은 모든 벡터들의 1차 편미분값으로된 행렬로 각 행렬의 값은 다변수 함수일 때의 미분값
- 체인룰을 통해 계산 됨





- 왼쪽 그림은 입력의 분산(다변수에서는 공분산)이 클 때 평균이 실제값과 크게 차이나는 결과를 볼 수 있음
- 오른쪽 그림은 입력의 분산이 작을 때 평균이 실제값과 차이가 작은 것을 볼 수 있음

3) KF vs EKF

	Kalman filter	EKF
state prediction (line 2)	$A_t\ \mu_{t-1} + B_t\ u_t$	$g(u_t, \mu_{t-1})$
measurement prediction (line 5)	$C_t\ \bar{\mu}_t$	$h(\bar{\mu}_t)$