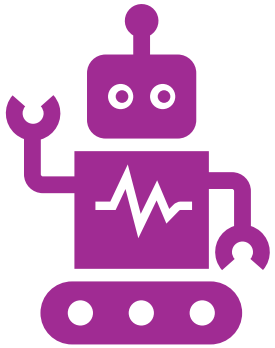
로봇 환경 AI 기초 실습

Artificial Intelligence for Robotics: Programming a Robotic Car

History



무인 시스템은 최근 몇 년 동안 무인 항공기(일명 UAV/드론)에서 무장 로봇 로버에 이르기까지 큰 진전을 이루었다.

군 전선의 "무인화"를 위한 지속적인 노력에서 미국군과 DARPA는 원격 조작자의 도움 없이 전투 지역과 위험 지역에서 작동할 수 있는 기계를 개발하기 위해 2000년 초반 부터 RnD 를 지속적으로 수행하였다.

History

- UAV 의 발전 - UAV(Unmanned Aerial Vehicles)
  - 자율 운행 시스템 개발이란 , 로봇 시스템이 결국 위험한 군사 작전(예: 보급 호송대)에서 인간 운전자를 대체할 수 있다.
  - 인간이 위험을 감소시키기 위한 구체성을 띤 연구 들이 이어져 왔는데,
  - 이를 위해 DARPA(DEFENSE ADVANCED RESEARCH PROJECTS AGENCY) 는 2004년에 Grand Challenge를 설립.
  - 최초의 완전 자율 운행 차량을 만드는 데 필요한 기술 개발을 촉진하도록 설계된 인센티브 프로그램.
  - 그 이후로 많은 도전이 수립됨, 그랜드 챌린지는 개발자를 유치하고 다양한 분야에서 협업을 장려하기 위해 상금 기반 경쟁을 활용하려는 DARPA의 첫 번째 주요 시도이다.
  - 2004년 3월 13일에 열린 첫 번째 챌린지는 15대의 자율 주행 지상 차량이 네바다주 프림의 사막을 가로지르는 228km(142마일) 코스를 주행하는 것이었습니다.

History

Grand Challange

- 일련의 자격 시험을 통과한 후 규정된 10시간 제한 시간 내에 테스트 코스를 완료한 첫 번째 팀은 100만 달러의 상금을 받았지만,안타깝게도 험난한 사막 코스와 관련된 기술적 장애물은 참여 팀에게 너무 힘들상황이었음.
- 아무도 코스를 완주하지 못했고 상금은 청구되지 않았습니다.

- 하지만, 여기서 첫 번째 경쟁은 공통의 목표를 위해 혁신가, 엔지니어, 프로그래머, 개발자로 구성된 커뮤니티를 만드는 데 가장 큰 의의가 있다.
- DARPA의 미 해병대 연락 담당자인 스콧 와들 중령은 그때의 감흥을 다음처럼 정리함

"그들이 가져온 신선한 사고방식은 그 이후 수년간 자율 로봇 지상 차량 기술 개발에 큰 진전을 가져온 불꽃이었습니다.“

- 두 번째 행사는 2005년 10월 8일 네바다 남부에서 열렸는데, 원래 195개 팀 중 5개 팀이 212km(132마일)를 완주하면서 상황이 조금 나아졌고.
- 스탠포드 대학의 참가팀인 " Stanley " (아래 표시)는 6시간 53분의 기록으로 1위를 차지했고, 200만 달러의 상금을 받게 됨.

History

- 2007년에 열린 세 번째 행사에서 DARPA는 모의 도시 환경에서 자율 주행을 포함하도록 초기 챌린지를 확대함.
- 이 행사는 DARPA Urban Challenge로 알려졌으며, 2007년 11월 3일 캘리포니아 주 빅터빌에 있는 전 조지 공군 기지에서 열리게 됨
- 여기에서 팀은 교통 체증 속에서 주행하고 합류, 추월, 주차 및 교차로 협상과 같은 복잡한 기동을 수행할 수 있는 자율 주행 차량을 만들어야 했다.
- 이전 챌린지는 참가자의 차량에 더 많은 신체적인 부담을 주었지만, 2007년 챌린지는 내비게이션 소프트웨어와 관련하여 수많은 추가 챌린지를 제시했다. 여기에서 설계자는 코스에서 다른 로봇을 감지하고 피하는 동안 모든 교통법을 준수할 수 있는 차량을 만들어야 했다. 이는 이전 챌린지에서는 거의 발생하지 않은 일이었다.
- 펜실베이니아 피츠버그에 있는 카네기 멜론 대학의 팀인 Tartan Racing은 Chevy Tahoe인 "Boss" 차량으로 200만 달러의 상금을 차지했다. Stanford University Racing Team은 2006년 폭스바겐 파사트인 "Junior" 차량으로 100만 달러의 2등 상금을 차지했다. 3등은 Virginia Tech의 VictorTango 팀으로 2005년 Ford Escape 하이브리드인 "Odin"으로 50만 달러의 상금을 차지했다.

History

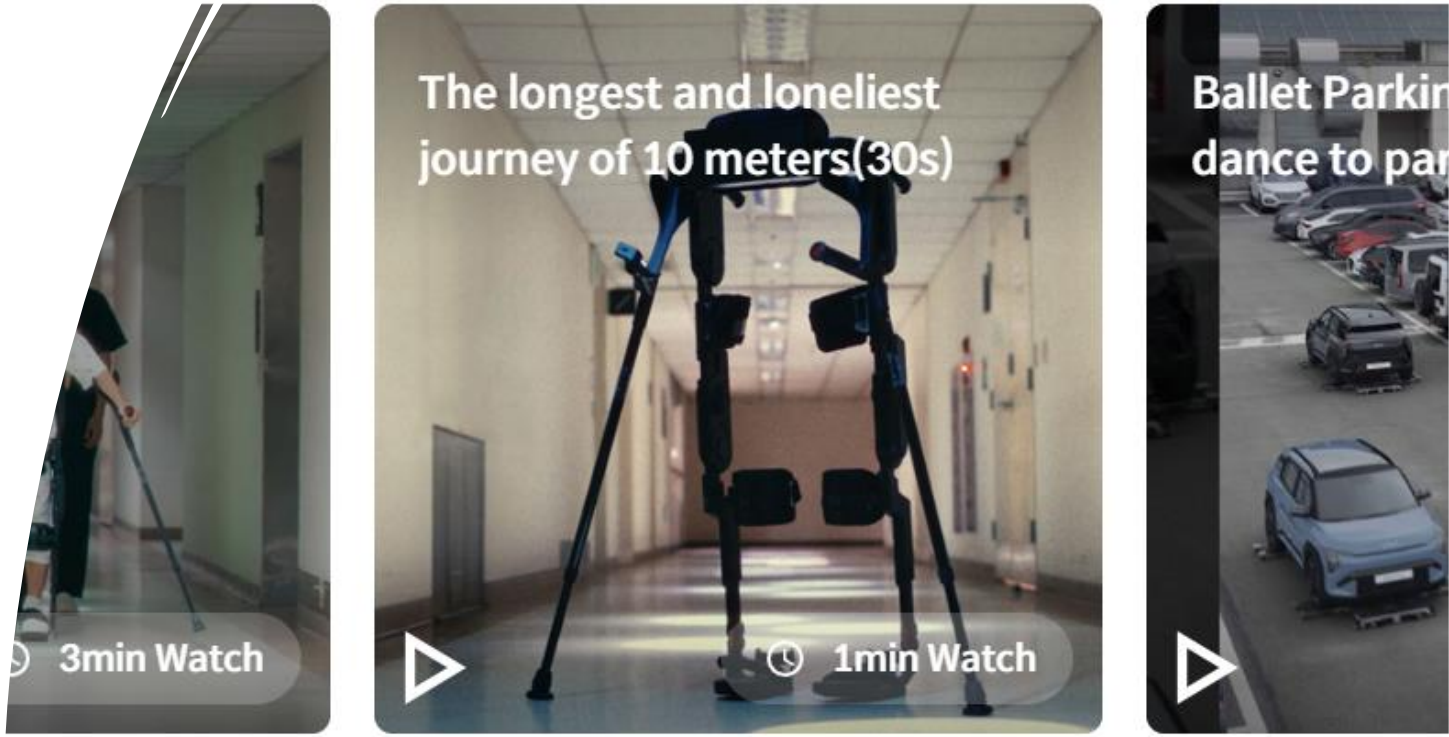
- 그 이후로 DARPA는 Grand Challenge의 상금 기반 경쟁 모델을 기반으로 하는 세 가지 다른 챌린지를 만들었다. 여기에는 Spectrum Challenge, DARPA Robotics Challenge (DRC), Cyber Grand Challenge(CGC)가 있다. 이 프로그램은 무선 통신, 비상 대응 로봇, 자동화된 네트워크 방어 시스템의 개발을 촉진하는 것을 목표로 한다.

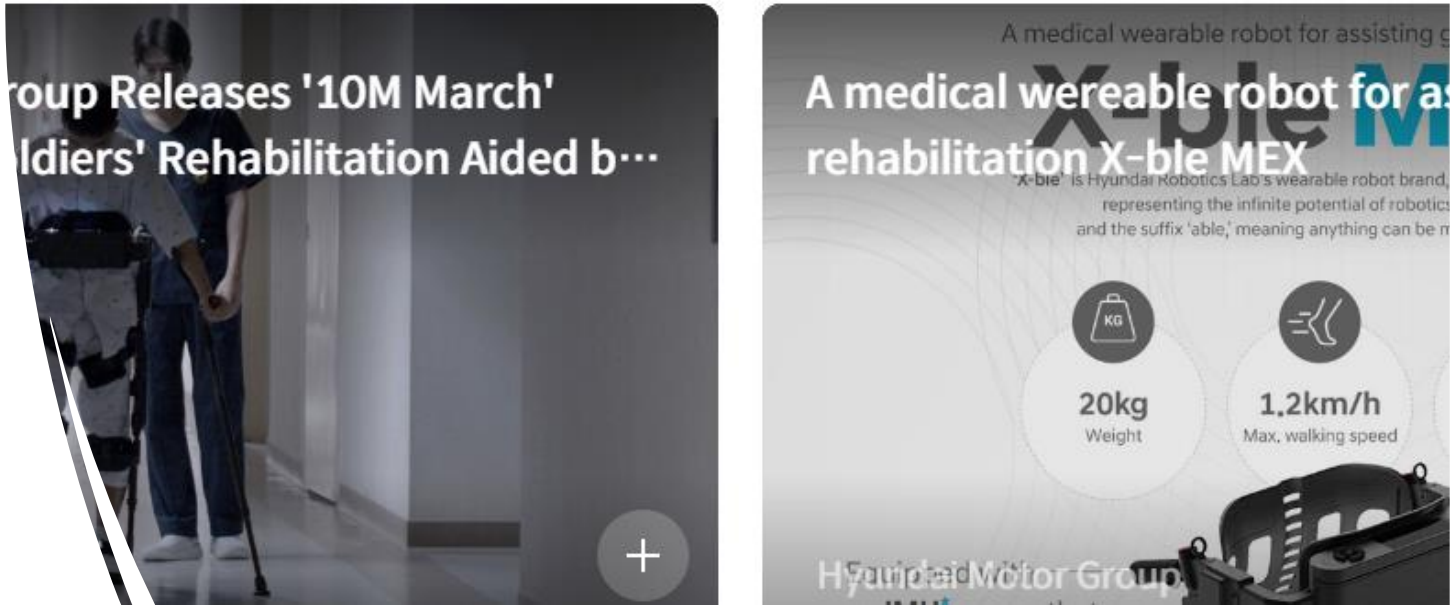
- 2020년, DARPA는 챌린지의 성공을 나타내는 지표로 방위 및 상업적 응용 프로그램의 확산을 시도한다. 방위 응용 프로그램의 예로는 해병대를 위해 개발된 Oshkosh Defense TerraMax 무인 지상 차량이 있다.

- 또한 Urban Challenge의 6개 완주자 중 하나인 TORC Robotics는 해병대 차량의 자율 기능을 계속 개발하고 있습니다.
- 상업 분야에서 자율 주행 시스템은 더욱 두드러지게 발전해 왔으며, 구글의 자율주행차 프로그램, 현대자동차그룹의 미래 자동차 기술 경진대회 , 제너럴 모터스의 전기 네트워크 차량 (EN-V, 아래 표시) 등이 그 예.

- 현재의 개발 속도를 감안할 때, 도로 위의 차량 상당수가 로봇 자율 주행 시스템으로 구동되기까지는 몇 년밖에 걸리지 않을 것입니다.

https://www.hyundaimotorgroup.com/tag/1021



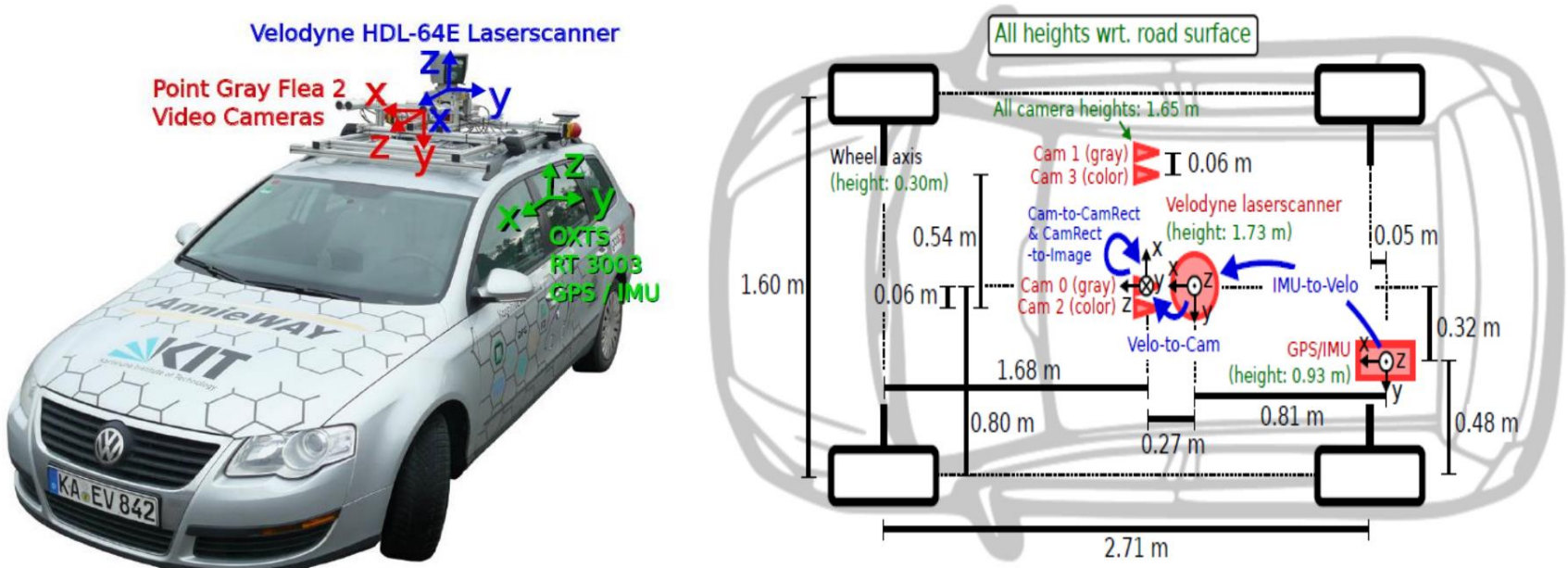



MY RIDE

Localization

로컬라이제이션 기술

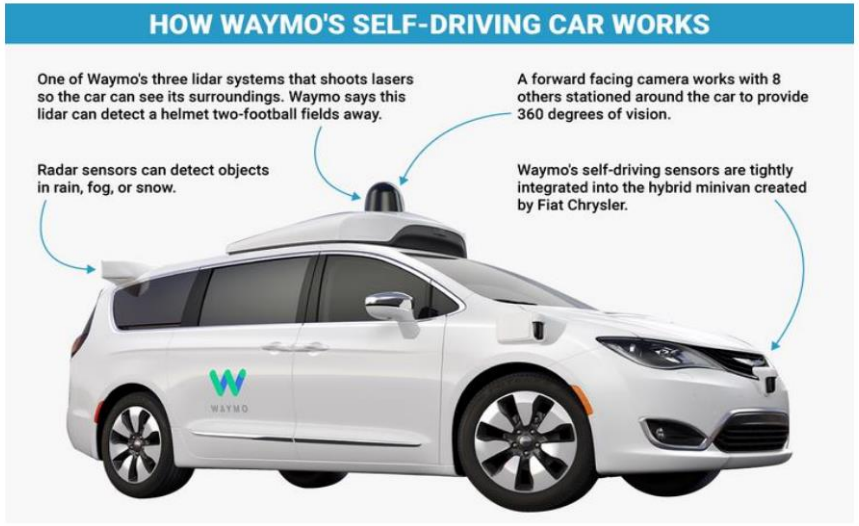
- 자동차의 위치를 실시간으로 정확하게 알아내는 기술
- 자율 주행 자동차에서 가장 핵심적인 태스크
- 정확도를 높이기 위해 다양한 센서 조합과 기법을 활용
- GNSS, LiDAR, HD Map, Visual Odometry, Dead Reckoning

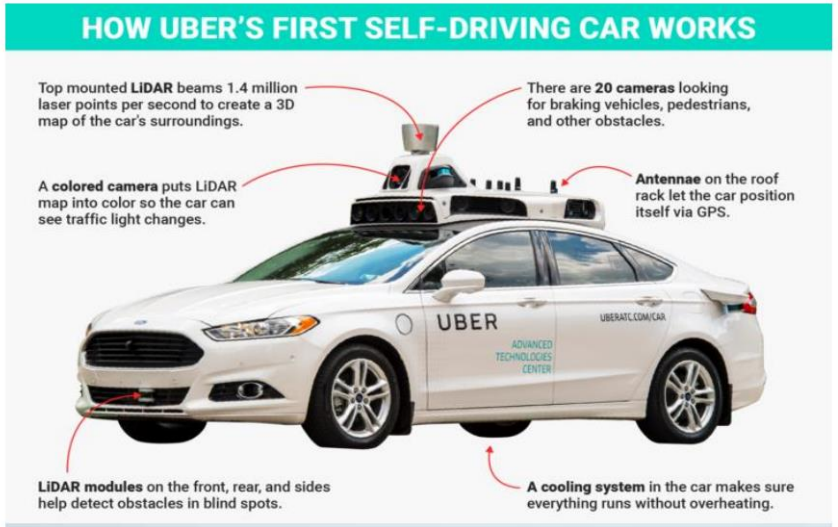


다양한 센서 조합의 예: KITTI dataset [Geiger2013]

로컬라이제이션 기술

상용 자율 주행 자동차들은 대부분 라이다와 HD 맵을 통해 로컬라이제이션을 수행





라이다를 장착한 자율주행 차량의 예 (Waymo, Uber, Baidu, KT)

LiDAR

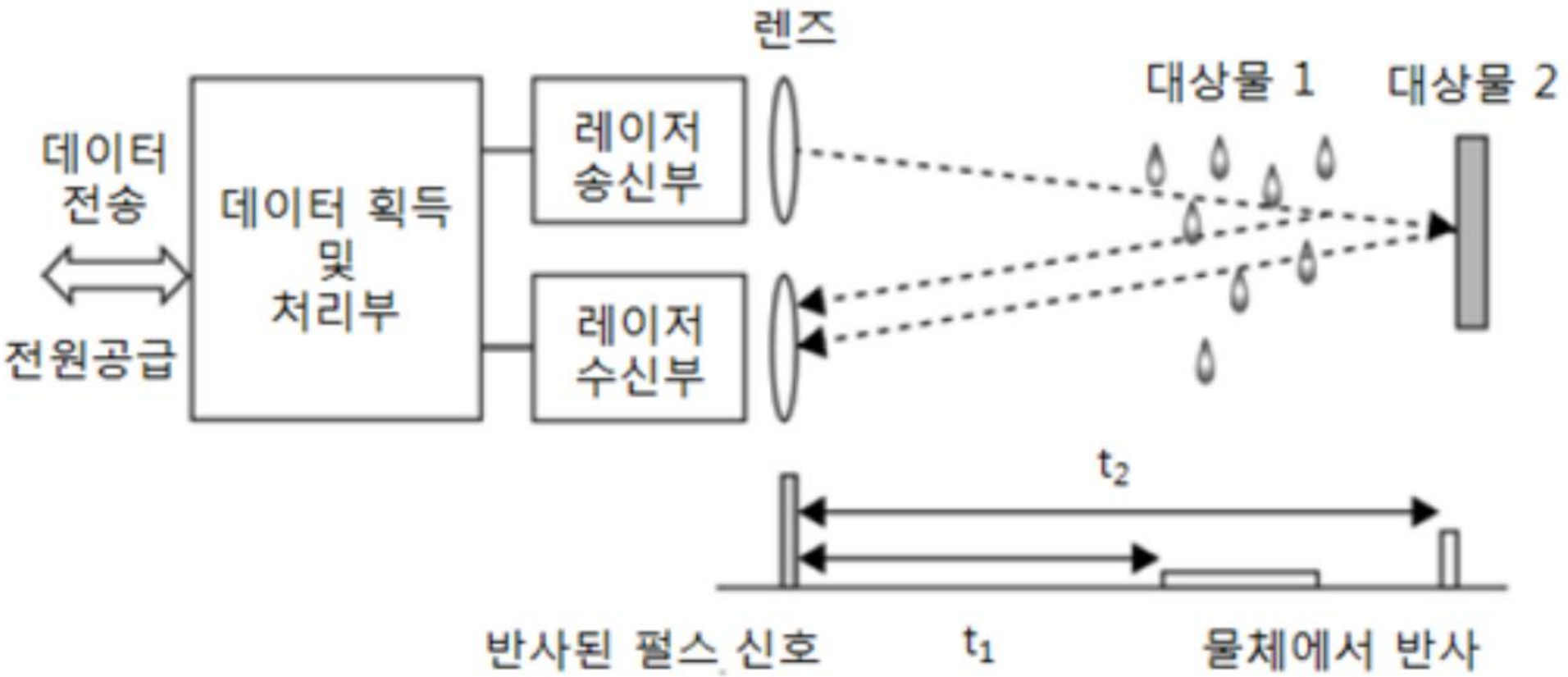
LiDAR (Light Detection And Ranging)

https://www.youtube.com/watch?v=y3Q7v5a0lnI

LiDAR (Light Detection And Ranging)

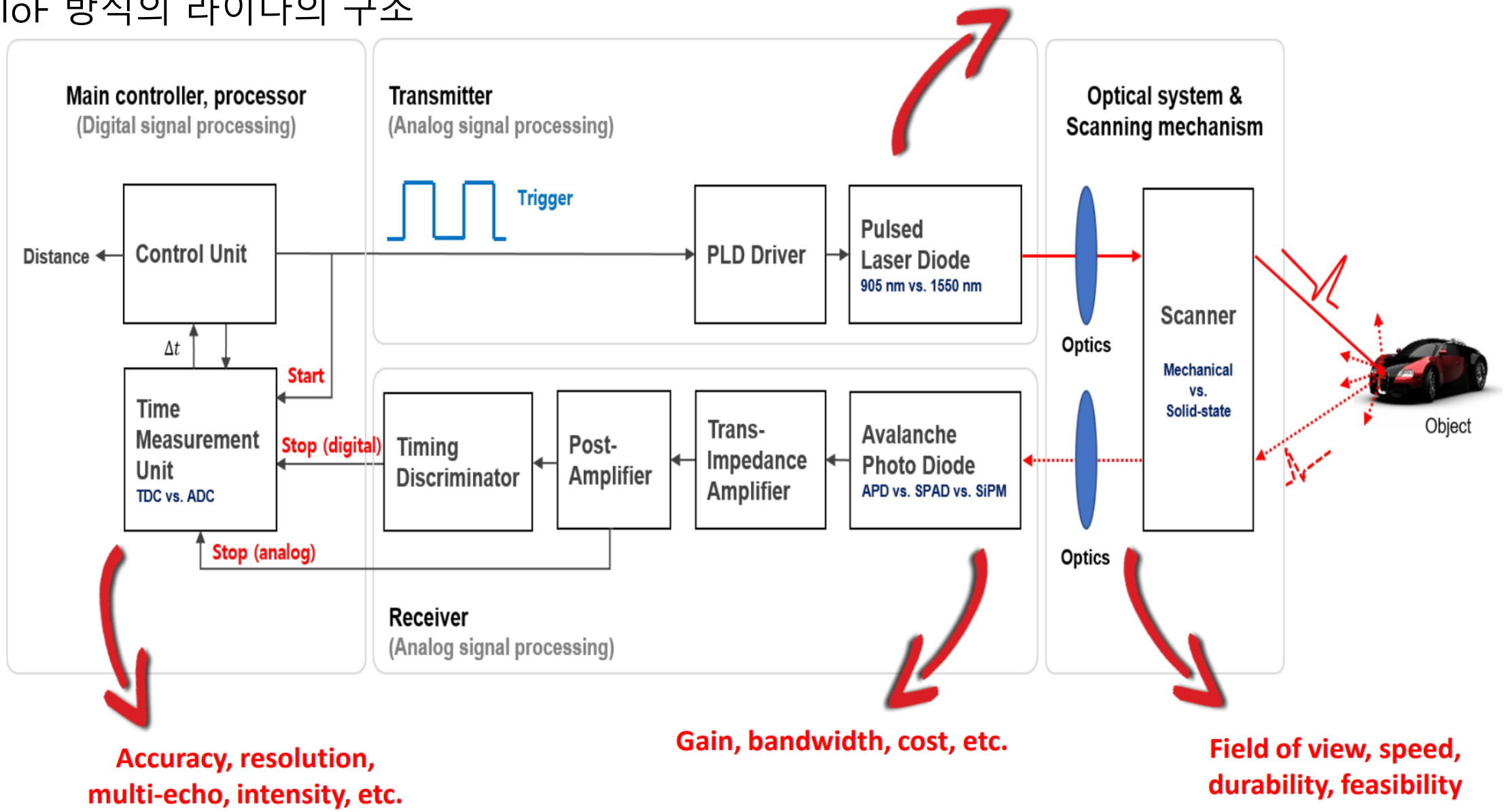
- 목표물에 펄스 레이저를 쏘아 반사된 펄스를 측정하는 방식으로 거리 계산
  - **TOF(Time-of-Flight):** 레이저 펄스 신호가 측정 범위 내의 물체에서 반사되어 수신기에도착하는 시간을 측정
  - **PS(Phase Shift):** 특정 주파수를 가지고 연속적으로 변조되는 레이저 빔을 방출하고, 물체로부터 반사되어 오는 레이저 신호의 위상 변화량을 측정
- 높은 정밀도의 3차원 공간 정보를 고속으로 획득 할 수 있는 기술
- 응용 분야: 항공 지도 제작, 대기/기상 측정, 대형 대상물의 역설계, 자율주행

LiDAR (Light Detection And Ranging)



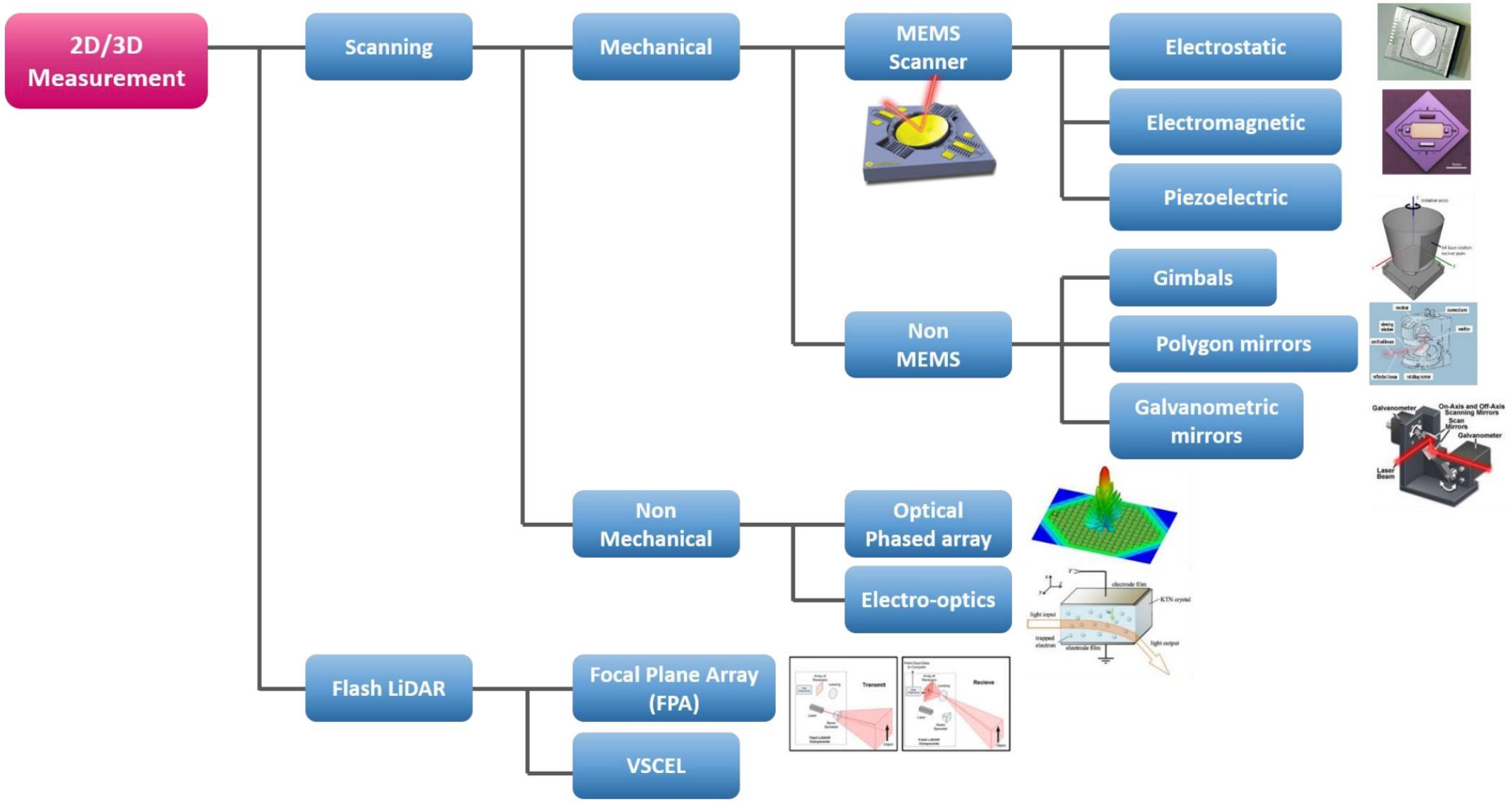
LiDAR (Light Detection And Ranging)

Pulsed ToF 방식의 라이다의 구조



스캐너 구조 기술

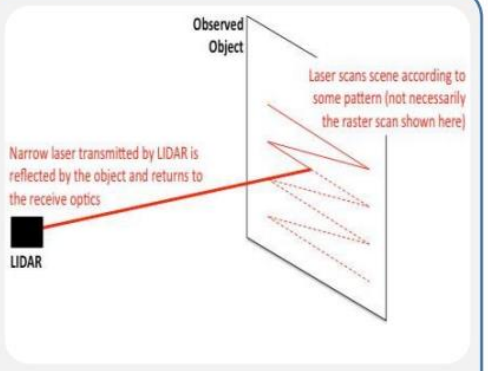
라이다의 빔 조향 방식 (Mechanical vs. Solid-state type)



스캐너 구조 기술

대표적인 라이다 스캐닝 방식

모터 기반의 스캐닝 방식



https://insights.globalspec.com/article/2013/lidar-gives-sight-to-autonomous-vehicles

3D scanning LIDAR_V사

http://velodynelidar.com/products.html

@CES2019

MEMS 기반의 스캐닝 방식

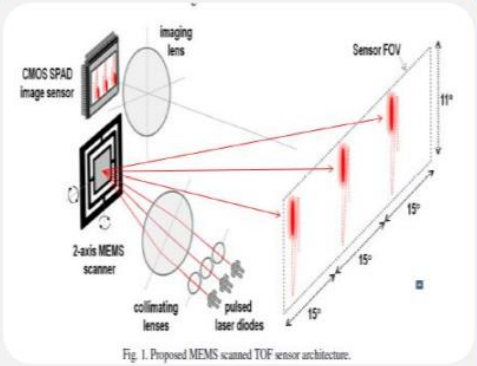


Fig. 1. Proposed MEMS scanned TOF sensor architecture.

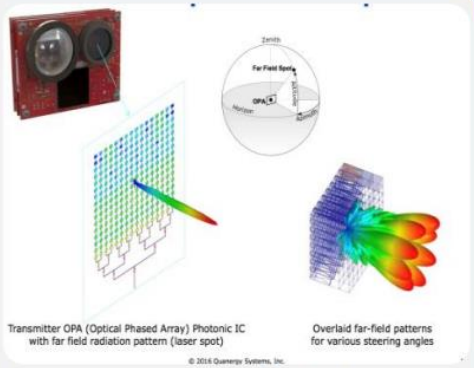
image-sensors-world.blogspot.com/2013/04/toyota-tof-imager-at-is-2013.html

3D MEMS LIDAR_I사

https://innoviz.tech/tech/

@CES2019

OPA 기반의 스캐닝 방식



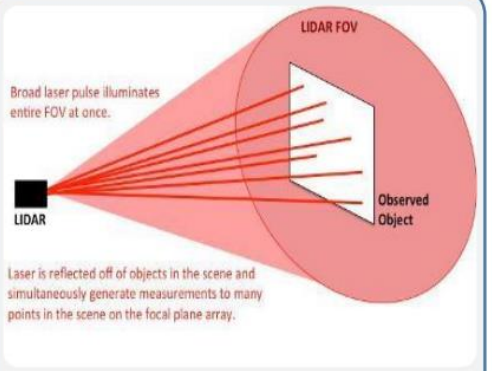
http://www.greencarcongress.com/2016/08/20160823-quanergy.html

3D solid-state LIDAR_Q사

https://www.spar3d.com/news/lidar/quanergys-mini-solid-state-lidar/

@CES2019

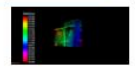
확산빔 기반의 플래시 방식



https://insights.globalspec.com/article/2013/lidar-gives-sight-to-autonomous-vehicles

3D Flash LIDAR_A사

http://safecarnews.com/continental-buys-asc-for-3d-lidar-tech-ma773/



라이다 기술 특징

◆거리: 수m 부터 200m 이상까지 떨어져 있는 물체 감지

◆공간 분해능: 짧은 파장대(905~1550nm)의 레이저 광을 조준할 수 있는 스캐닝 기술을 사용함으로써 최대 0.1도 단위까지 공간 분해능을 제공

◆FOV(Field of View): 회전형 기계식 라이다의 경우 360도의 수평 방위각으로 ADAS 기술 중 가장 넓은 FOV를 제공하며, 수직 FOV도 타 센서에 비해 뛰어남.

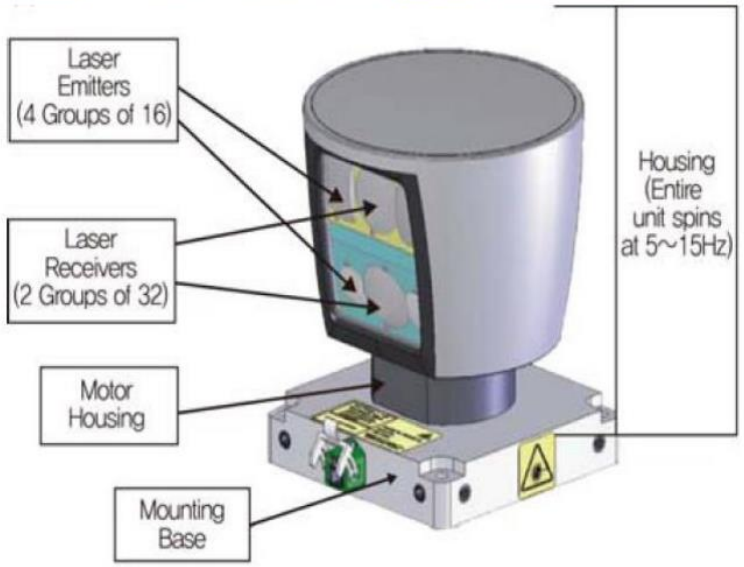
◆날씨: 비, 안개, 눈 등의 기후에서 성능저하가 발생할 수 있으나, 1550nm의 적외선 파장을 이용하면 악천후에서의 성능 개선 가능

◆주변광: 주변광 조건에 영향을 받음. 그러나 카메라에 비해 영향이 적은 편이며, 특히 야간의 경우에는 오히려 매우 높은 성능을 제공

◆가격: 현재 샘플 제품들의 경우 1만 달러(약 1,000만원) 내외

라이다 기술 특징

벨로다인 HDL-64 라이다



Specifications	Velodyne HDL-64E
Laser wavelength	905 [nm]
Operating range	**120 [m] (R~0.8)**
Rrange(@R=0.1)	50 [m]
Horizontal FOV	**360 [degree]**
Angular resolution (H)	0.09 [degree]
Vertical FOV	**+2 to -24.8 [degree]**
Angular resolution (V)	0.4 [degree]
Spin rate	**5~15 [Hz]**
Points per second	> 1.333 M
Operating voltage	15±1.5 [V DC] (@4A)
Power consumption	< 60 [W]
Weight	13 [kg]
Dimension	254(h) x 203.2(r) [mm]
Operating temp.	-10~50 [℃]

HD MAP

(High Definition)

HD 맵

- 자율 주행 자동차를 위한 정밀 지도

- 정밀 지도의 필요성
  - GPS, 카메라, 레이더, 라이다 등의 다중 센서 구성에도 불구하고 인지 오차 발생
  - 정확한 인지를 통한 신뢰성, 안전성 강화를 위해 가외성(redundancy)이 요구됨
  - 정밀 지도를 통해 자율주행 차량의 기존 센서를 보완하고 오차 범위를 축소
  - 자율주행자동차에 필요한 주행 공간에 대한 정보를 상세하게 전달

- 정밀 지도를 통한 기대 효과
  - 측위 센서와 정밀지도 정보의 매칭을 통해 현재 위치를 정밀하게 측정함으로써 자율 주행 차량의 측위 오차 감소
  - 도로상의 정보를 디지털화 시키면 자율주행 차량의 학습 능력(AI) 향상 가능
  - 주행 경로에 대한 상세한 정보를 사전에 제공함으로써 자율주행 S/W의 실시간 처리량 감소
  - 지형에 맞는 최적 운행, 관성 주행 안내, 배터리 충방전 예측 관리 등을 통해 친 환경차의 연비 및 배터리 효율성 향상
  - 정적인 정보(지형지물 위치, 도로 위치 등) 뿐만 아니라 V2X 기술과의 결합을 통해 실시간으로 도로상황, 주변 차량의 위치 및 주행 정보와 같은 동적 정보 제공 가능

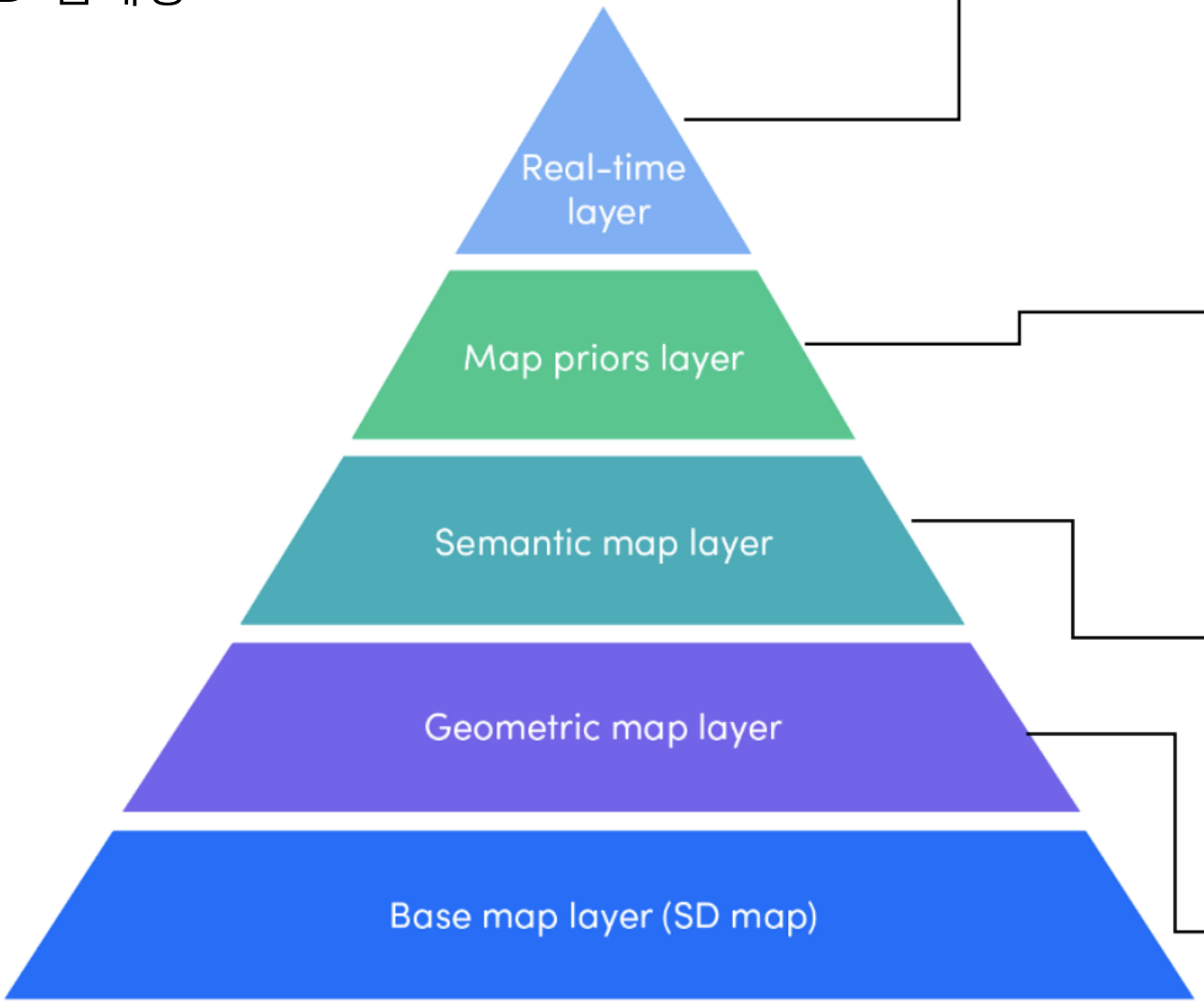
HD 맵

- HD 맵의 요구 조건
  1) 센티미터 수준의 정밀도를 갖춘 정밀 3D 입체 지도
  2) 도로의 변화를 반영하도록 주기적인 지도 업데이트
  3) 자율 주행 시스템과의 유기적인 연동
- 기존 지도 vs. HD 맵

구분	기존 수치 지형도	정밀도로지도
방법	항공사진 측량	Mobile Mapping System (MMS)
용도	• 항법 지도(Navigation)은 도로 단위의 정보, 경로 탐색/안내 정보 제공 • 국토, 도시관리, 건설, 토목, 행정, 인터넷 지도, 내비지도 등	• 자율 주행차 연구/개발 및 상용화, 도로관리, 정밀 내비지도 개발 등 • 자율 주행을 위해 차선(Lane) 단위 정보 제공
특징	2차원 전자지도 • 정적인 정보 제공 • 고정된 지형지물의 위치 등	3차원 전자지도 • 정적+동적인 정보 제공 • 도로환경 및 교통상황 (도로 고저, 차선 너비 등)
정확도	(1/5천) 평면: 3.5m / 수직: 1.67m	평면: 0.25m / 수직: 0.25m
정보	도로 경계, 도로 중심선, 교통 표지 (도심지, 위치정보)	차선, 차로 중심선, 규제선, 도로 경계, 도로 중심선, 교통 표지(위치+속성), 노면 표지(위치+속성)

HD 맵

HD 맵계층



Only layer in the map designed to be updated while the map is in use by the AV serving a ride
: real-time traffic information such as observed speeds, congestion, newly discovered construction zones, etc.

Information about dynamic elements and also human driving behavior
: order in which traffic lights at an intersection cycle through their various states

Semantic & static objects used for driving safely
: various traffic 2D and 3D objects such as lane boundaries, intersections, crosswalks, parking spots, stop signs, traffic lights, etc.

Very high detail 3D information of the world
: 3D point cloud with color/intensity information from raw sensor data

HD 맵 제작

- Mobile Mapping System (MMS)을 이용
  - 10~20cm 정확도의 3D 지형, 도로 정보 수집
  - 도로 및 표지판 등 차량 주행을 위한 기본 정보
  - 자율 주행에 필수적인 차선 경계면, 도로 경계면 등의 세밀한 정보
  - 대표 기업: RIEGL, Optech, HERE, TomTom, Mitsubishi, Naver labs

- Camera, Laser Scanner, GPS, INS, DMI 등 다양한 센서 장착
  - MMS 차량의 경우 대당 15억원 수준

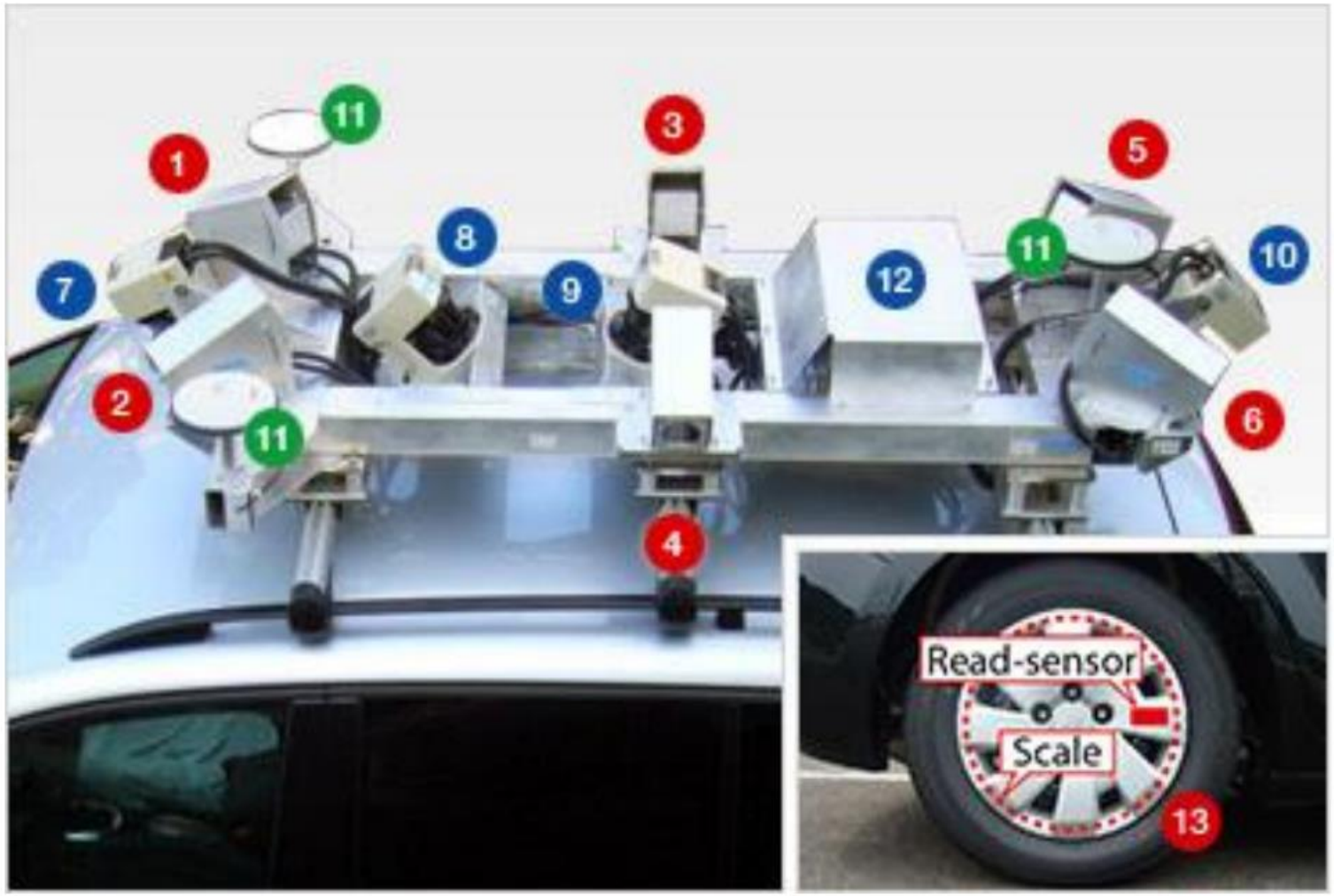
- 획득한 방대한 자료(data)를 정보(information)으로 가공하는 후 작업 필요
  - 상당한 비용과 시간 소모
  - 많은 메모리와 저장 공간 차지
  - 실시간 정보 송수신과 업데이트를 위한 V2X 기술 발전도 필수

HD 맵 제작

MMS의 구성

- GPS(Global Positioning System): 인공위성을 이용한 수신 지점의 위치 추정
- INS(Inertial Navigation System): GPS 음영지역에서 위치 추정이 가능한 관성 항법 장치
- LIDAR: 레이저 빛을 이용해 물체와의 거리 측정
- DMI(Distance Measuring Instrument): 바퀴 회전 수 측정을 통해 정확한 주행 거리 측정

HD 맵 제작



1 Camera(front;right)
2 Camera(front;left)
3 Camera(side;right)
4 Camera(side;left)
5 Camera(rear;right)
6 Camera(rear;left)
7 Laser scanner(front;downward)
8 Laser scanner(rear;upward)
9 Laser scanner(front;upward)
10 Laser scanner(rear;downward)
11 GPS antenna
12 IMU
13 In-wheel odometer

Mitsubishi Electric의 MMS 구성 예

HD 맵 제작

https://www.youtube.com/watch?v=kl6rcGj6uBY

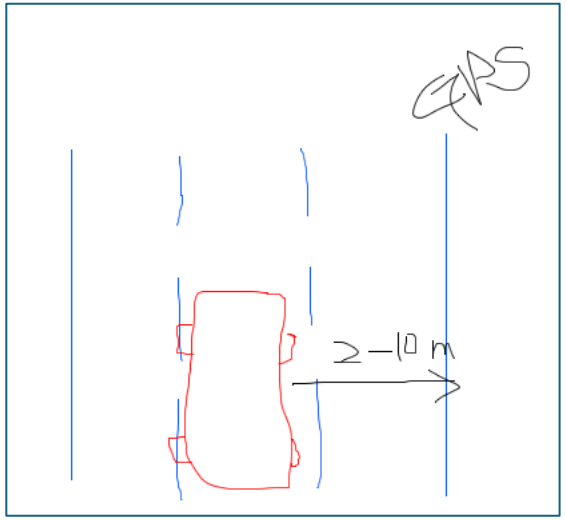
Localization (이론)

Localization

- Localization 은 자율 주행 시스템이 Vihicle(Device/Robot) 의 위치를 정확히 파악하고, 주변환경과의 상호작용을 이해 하는 과정이다.
- 즉 시스템은 정확한 Localization 정보 없이는 Safe Driving 과 Correct decision 을 구명 할 수가 없다.
- 일반적으로 GPS 는 자동차의 위치를 판단하는데, 쉽게 사용될수는 있지만, 성능에 한계가 존재

Localization

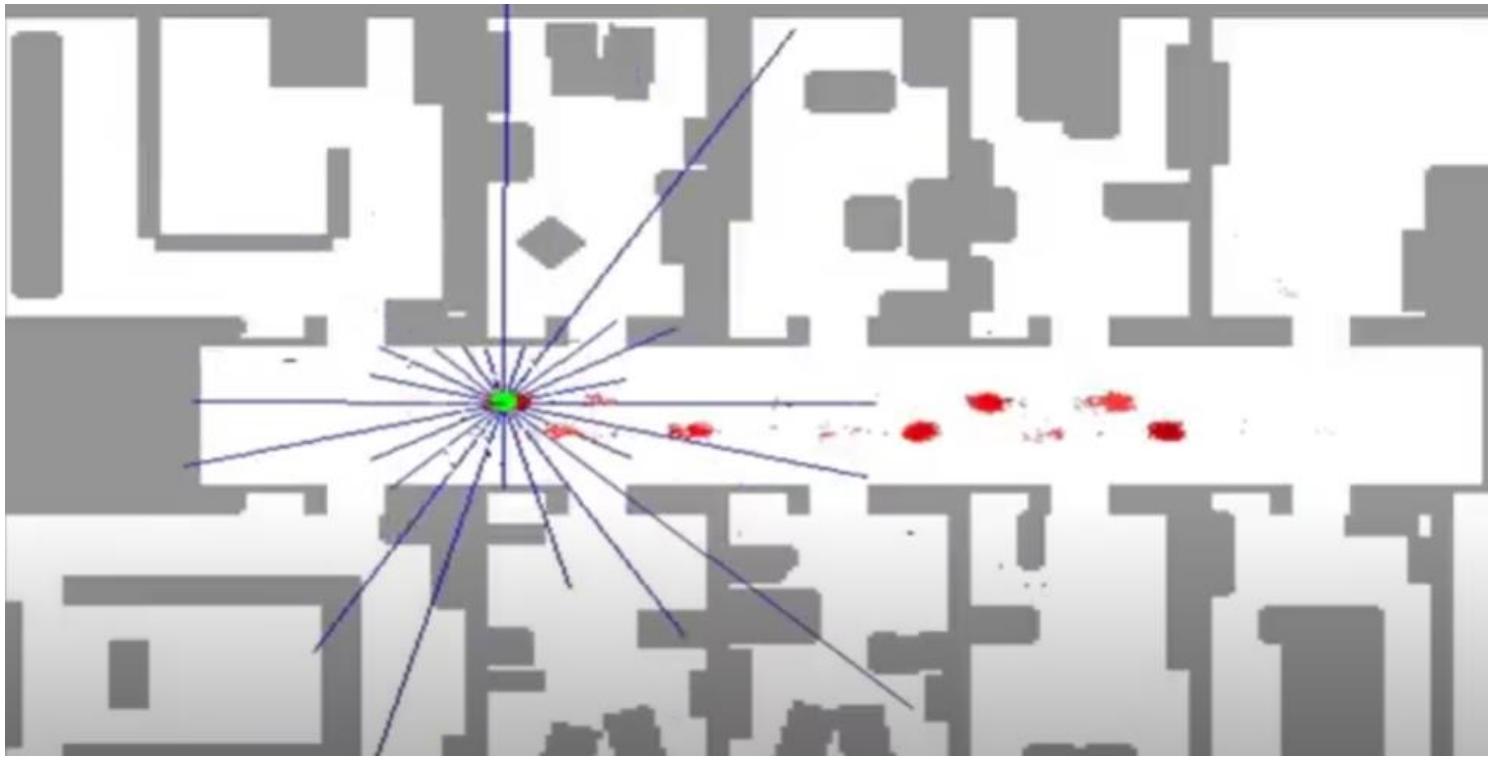
- 내 위치에 대한 인식?
- GPS (global POSITION SYSTEM)
- GPS 기반 로봇 or 자율 주행차 운행시
  - 2M ~ 10M error rate



- 안정적인 운행을 위한 최소 요구 정확도는?
  - 10cm
  - 무엇으로 이것을 유지할것인가?

Localization

- Google self driving car Simulation 에서는 Localization 을 적용하여 라인의 간격을 형상화 하고 안정성을 확보 하는 방법을 통해 자율 주행 기술을 확인하는 과정을 나가게 되었다.



Total Probability

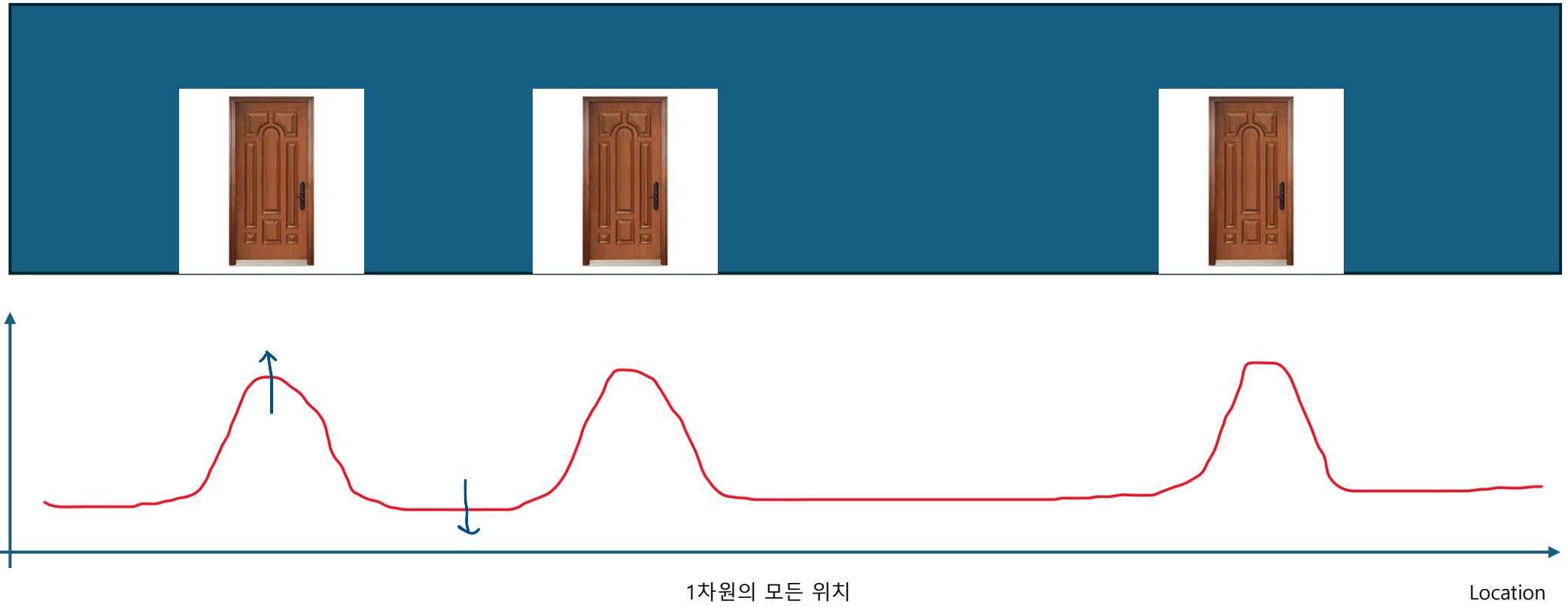
로봇은 자신의 위치를 스스로 결정 할수가 없다
이 상태에서 자신의 위치를 찾기 위해 세상의 모든 가능한 위치에 동일한 가충치를 부여하여 위치를 판단한다면, 그 방식으로 더큰 혼란을 갖게 된다.
State of maximum confusion.

Total Probability

문이 있고, 로봇은 문과 벽을 구별한다고 가정하자, 스스로 위치를 추정하기위한 문 앞에서 있을때
→ 이 정보는 로봇이 문 옆에 있을 가능성이 높다는 것을 의미 한다.
→ 문의 측정은 , 가능한 위치라고 정의된 신뢰도 함수를 만드는 것을 의미

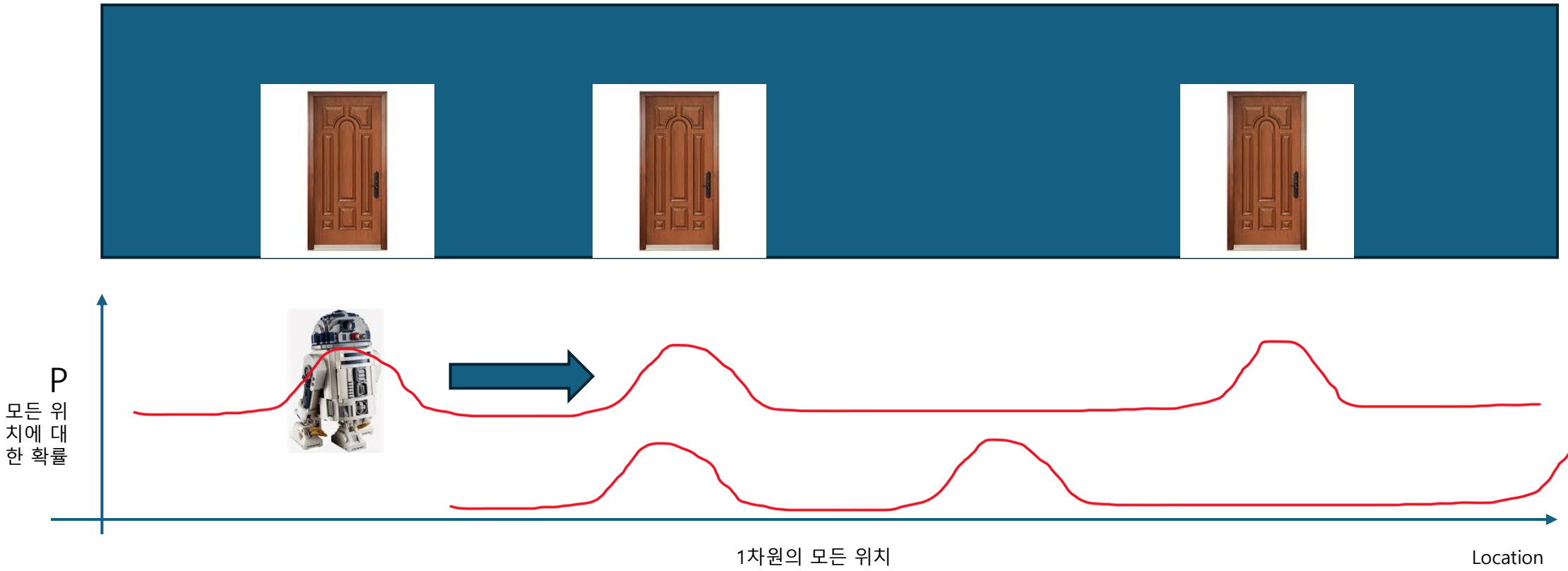
Total Probability

문이 옆에 있을때, 로봇은 문을 찾을 수 있는 확률이 올라간다
하지만. 여전히 가능성에 기반한 판단 사후확률 (Posterior Probability)이다.
→ 로봇의 센서가 오류로 없는 문을 보았을 수 있는 확률은 존재



Total Probability

로봇이 이동한다면, 이동에 따라 신뢰도는 변경된다.



Uniform Probability

x1	x2	x3	x4	x5
				

- 5개의 위치 중, 로봇이 처음에 어떤 위치에 있을 확률이 있다고 가정 할때 , 균일한 확률 벡터 p를 를 찾는 프로그램을 만들어보자
- Size n=5 사용
- $P(X_i) = ?$

Uniform Probability

```
p=[]
n=5

value = 1/n
for i in range(n):
    p.append(value)

print (p)
```

Compute sum

X1 0.2	X2 0.2	X3 0.2	X4 0.2	X5 0.2

로봇이 빨간 영역만을 감지 한다고 가정한다면, x2,x3 의 신뢰도는 상승하고, 그 외 나머지 영역의 신뢰도는 감소 할 것이다.

이 러한 측정된 결과를 통해, 신뢰도 기반 규칙을 만든다.

x 0.6

x 0.2

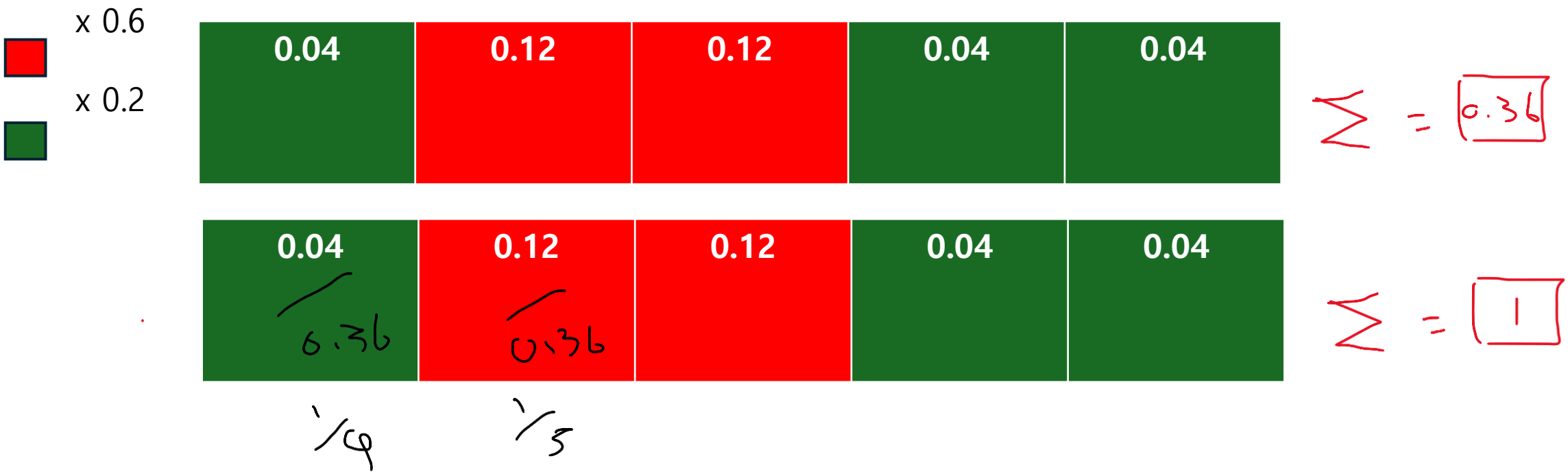
?	?	?	?	?

$\neq 1$

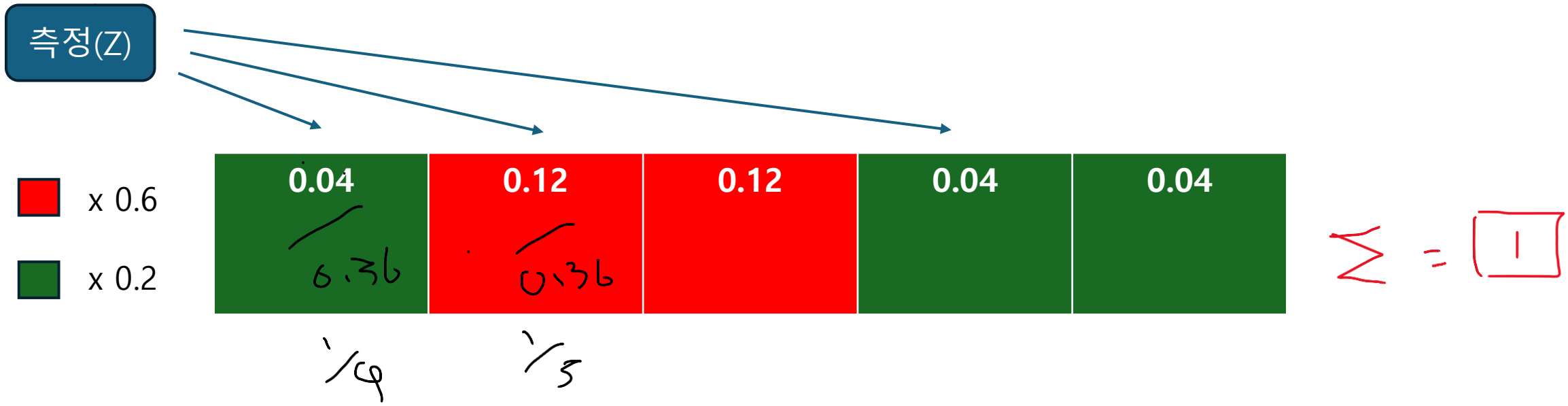
$\sum = \square$

Nomalize Distribution

합이 0.36 인 구성을, 정규 분포로 만들자



Nomalize Distribution



측정 Z를 본후 각 셀 i(1~5) 의 확률, Z(측정)➔ Posterior Distribution, dl 확률은 측정 z가 있을때, xi 위치의 '사후 분포'를 의미

p(Xi | Z)

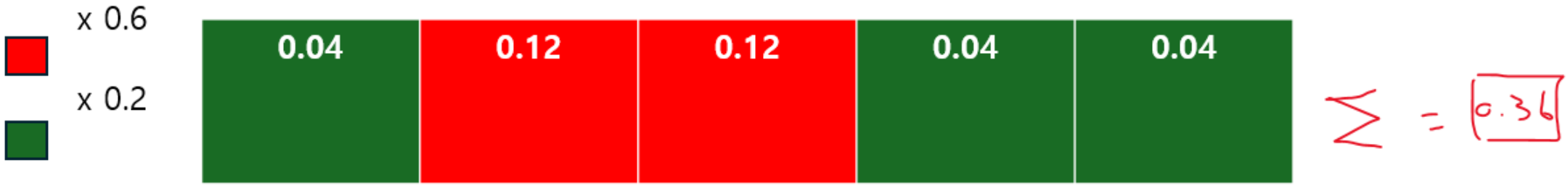
pHit,pMiss

균일한 P 각 셀의 항목을 곱한 후 , 그 리스트 p를 출력하는 코드를 작성 해보자

관측 지점은 pHit 반대는 pMiss로 정의

빨간색 셀 1과 2는 히트이고 다른 녹색 미스로 표기

그리고 그 합을 구하자



pHit,pMiss

```
#Modify the program to find and print the sum of all
#the entries in the list p.

p=[0.2,0.2,0.2,0.2,0.2]
pHit = 0.6
pMiss = 0.2
sum = 0

#Enter code here
for i in range(len(p)):
    if i == 1 or i == 2:
        p[i] *= pHit
    else:
        p[i] *= pMiss
    sum += p[i]
print(p)
print(sum)
```

Sense Function

p와 Z를 입력으로 사용하고, 정규화되지 않은 결과를 출력합니다.

확률 분포, p 항목을 곱한 후 p by pHit 또는 pMiss 색상에 따라 각 셀을 리스트로 구성해 보자

```
 8  p=[0.2, 0.2, 0.2, 0.2, 0.2]
 9  world=['green', 'red', 'red', 'green', 'green']
10  Z = 'red'
11  pHit = 0.6
12  pMiss = 0.2
13
14  def sense(p, Z):
15      #
16      #ADD YOUR CODE HERE
17      #
18      return q
19
20  print sense(p,Z)
```

Sense Function

```
p=[0.2, 0.2, 0.2, 0.2, 0.2]
world=['green', 'red', 'red', 'green', 'green']
Z = 'red'
pHit = 0.6
pMiss = 0.2

def sense(p, Z):
    #Enter code here
    for i in range(len(p)):
        if world[i] == Z:
            p[i] *= pHit
        else:
            p[i] *= pMiss
    return p

print (sense(p,Z))
```

Normalized Sense Function

출력을 정규화하도록 코드를 수정해보자

- sense() 함수의 q의 항목을 구성하자
- 합계가 1이 되도록 (정규화 로직을 적용 하자

Normalized Sense Function

출력을 정규화하도록 코드를 수정
sense 함수의 q의 항목이 ,
합계가 1이 되어야(정규화) 합니다.

```
p=[0.2, 0.2, 0.2, 0.2, 0.2]
world=['green', 'red', 'red', 'green', 'green']
Z = 'red'
pHit = 0.6
pMiss = 0.2

def sense(p, Z):
    q=[]
    sum = 0
    for i in range(len(p)):
        hit = (Z == world[i])
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
        sum += q[i]
    q = [x/sum for x in q]
    return q
print (sense(p,Z))
```

로직 검증하기

- Z 측정 시점을 녹색 적용
- 이때 결과 확률 분포가 올바른지 확인해보자

```
[0.2727272727272727, 0.09090909090909093, 0.09090909090909093, 0.2727272727272727, 0.2727272727272727]
```

측정 확률 로직 검증하기

```
p=[0.2, 0.2, 0.2, 0.2, 0.2]
world=['green', 'red', 'red', 'green', 'green']
Z = 'green'
pHit = 0.6
pMiss = 0.2

def sense(p, Z):
    q=[]
    sum = 0
    for i in range(len(p)):
        hit = (Z == world[i])
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
        sum += q[i]
    for i in range(len(p)):
        q[i]=q[i]/sum
    return q
print (sense(p, Z))
```

Multiple Measurements

확률을 두 번 업데이트하도록 코드를 수정해보자.

그리고 둘 다 후에, 사후 분포를 제공하자.

```
p=[0.2, 0.2, 0.2, 0.2, 0.2]
world=['green', 'red', 'red', 'green', 'green']
measurements = ['red', 'green']
pHit = 0.6
pMiss = 0.2

def sense(p, Z):
    q=[]
    for i in range(len(p)):
        hit = (Z == world[i])
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
    s = sum(q)
    for i in range(len(q)):
        q[i] = q[i] / s
    return q
#
#ADD YOUR CODE HERE
#
print p
```

```
[0.20000000000000004, 0.19999999999999996, 0.19999999999999996, 0.20000000000000004, 0.20000000000000004]
```

Multiple Measurements

```
p=[0.2, 0.2, 0.2, 0.2, 0.2]
world=['green', 'red', 'red', 'green', 'green']
measurements = ['red', 'green']
pHit = 0.6
pMiss = 0.2

def sense(p, Z):
    q=[]
    sum = 0
    for i in range(len(p)):
        hit = (Z == world[i])
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
        sum += q[i]
    for i in range(len(q)):
        q[i] = q[i] / sum
    return q
for measurement in measurements:
    p = sense(p, measurement)
print (p)
```

여기까지, 로봇의 환경 내에서 자신의 위치를 파악하기 위해 확률과 센서 측정을 결합하는 방법을 살펴보았습니다.

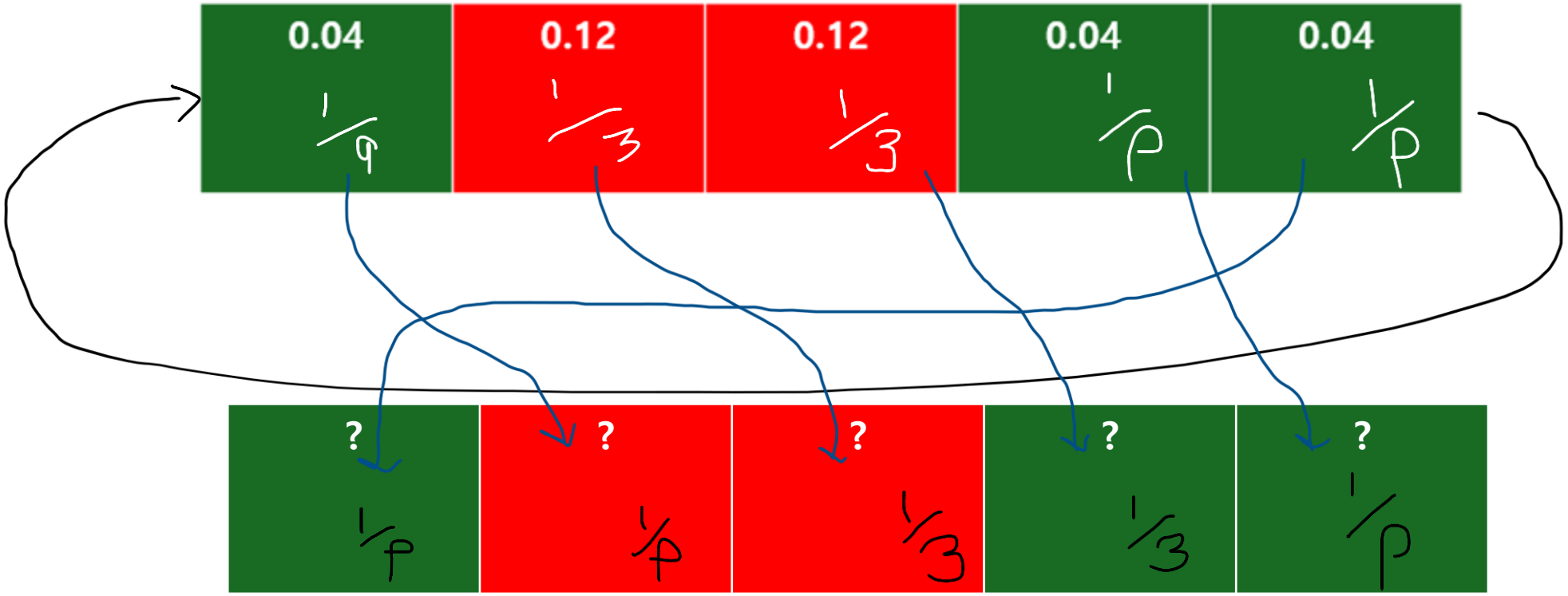
Robot Motion(Exact)

로봇이 왼쪽에서 오른쪽으로 1그리드 만큼씩 이동한다 가정하자
맨 끝에 도달시 다시 맨 왼쪽으로 이동하여 다시 실행되는 순환식 구조

만약 이러한 이동이 일어난후 사후 확률은 각 그리드 별로 어떻게 적용이 될것인가?

Robot Motion(Exact)

이동은 모두가 오른쪽으로 옴겨 진다는 것을 의미한다.
맨 마지막은 처음으로 연결된다.
→ 즉 로봇이 이동하는 만큼 확률을 변경하면 된다. (뒤로가기는 제외함)



Robot Motion(Exact)

새 분포를 반환하는 Move 함수를 구성한다.

- q, U 단위로 오른쪽으로 이동.
- U=0이면 q는 다음과 같다.
  (p와 같을 것).

```
p=[0, 1, 0, 0, 0]
world=['green', 'red', 'red', 'green', 'green']
measurements = ['red', 'green']
pHit = 0.6
pMiss = 0.2

def sense(p, Z):
    q=[]
    for i in range(len(p)):
        hit = (Z == world[i])
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
    s = sum(q)
    for i in range(len(q)):
        q[i] = q[i] / s
    return q

def move(p, U):
    #
    #ADD CODE HERE
    #
    return q

print move(p, 1)
```

Move Function

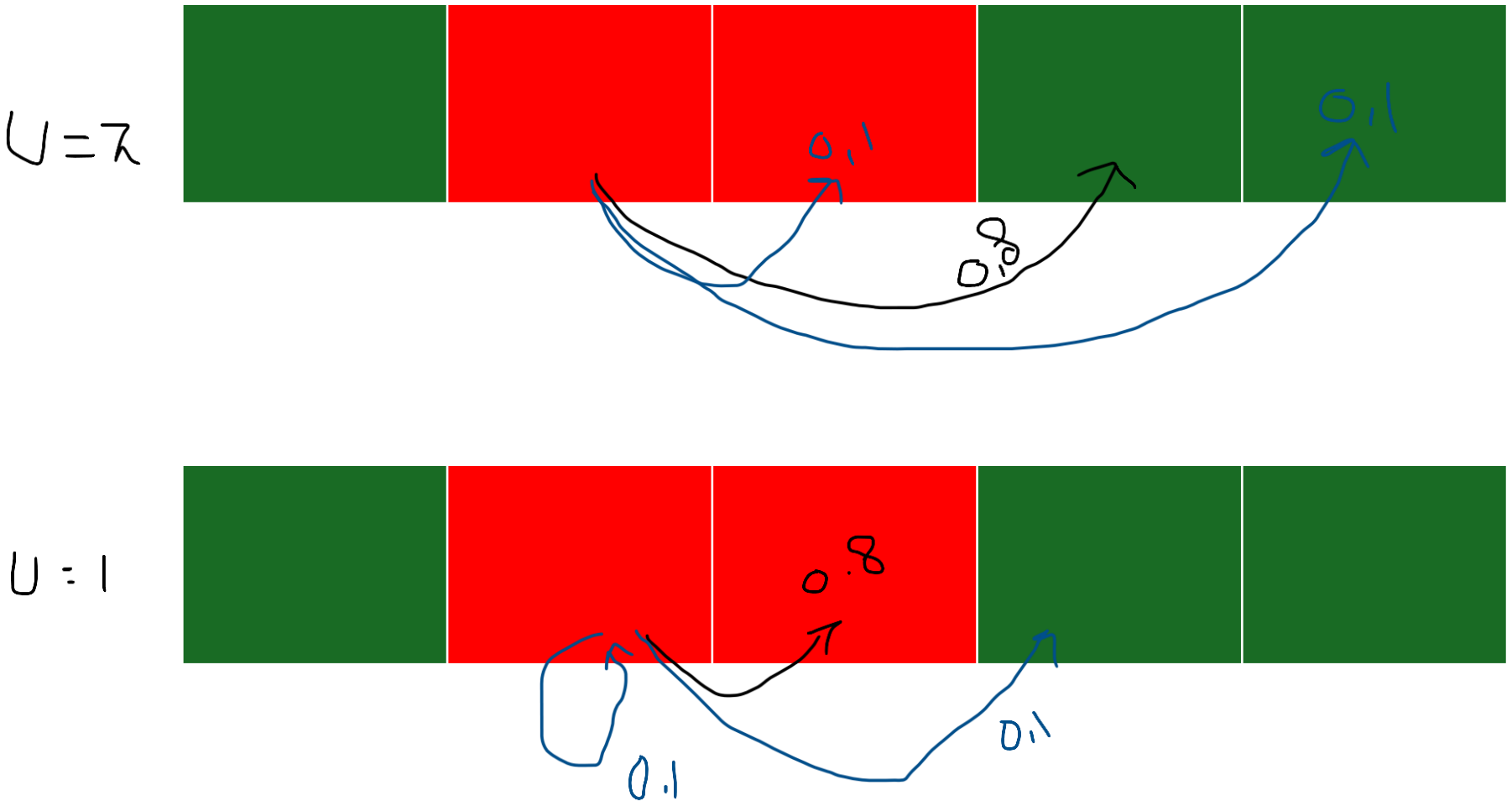
```
p=[0, 1, 0, 0, 0]
world=['green', 'red', 'red', 'green', 'green']
measurements = ['red', 'green']
pHit = 0.6
pMiss = 0.2

def sense(p, Z):
    q=[]
    for i in range(len(p)):
        hit = (Z == world[i])
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
    s = sum(q)
    for i in range(len(q)):
        q[i] = q[i] / s
    return q

def move(p, U):
    U = U%len(p)
    q = p[-U:] + p[:-U]
    return q

print (move(p, 1))
```

Inaccurate Robot Motion

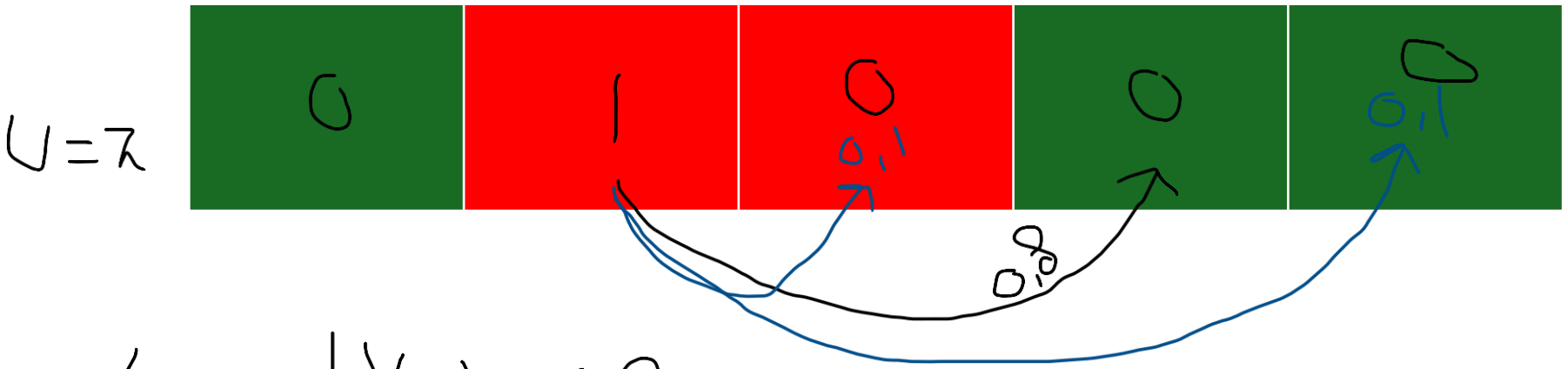


위치 추정이 어려운이유: 로봇의 이동이 부정확 하다

이따금 목표보다, 덜가거나, 더가는 현상이 나올 수 있다.

Inaccurate Robot Motion

수학적인 확률적 관점에서 살펴보기



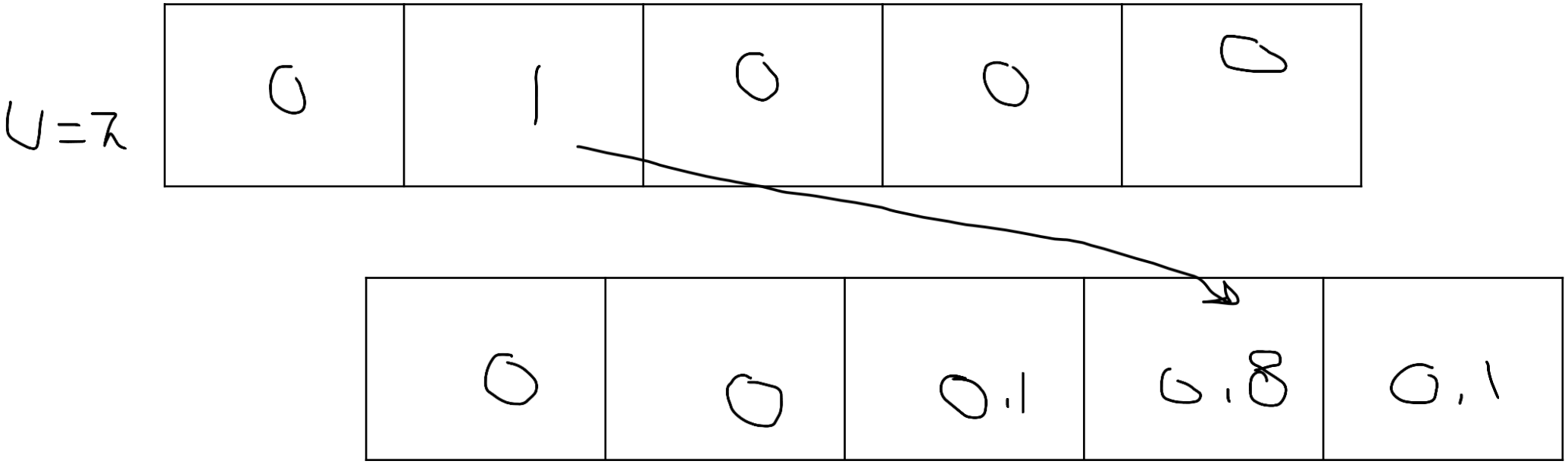
$P(X_{i+2} \mid X_i) = 0.8$

$P(X_{i+1} \mid X_i) = 0.1$ Under shoot

$P(X_{i+3} \mid X_i) = 0.1$ Over Shoot

Inaccurate Robot Motion

수학적인 확률적 관점에서 살펴보기 : Case1- 높은확률(0.8) 이동시 부정확한 이동이 나오는 확률(0.1)



Inaccurate Robot Motion

수학적인 확률적 관점에서 살펴보기 : Case2 – 0;.5 확률 이동 2개가 존재할 때

U=1

0	0.5	0	0.5	
0.4	0.05	0.05	0.4	0.05 0.05 0.1

overS + UnderS

$P(X_{i+2} \mid X_i) = 0.8$

$P(X_{i+1} \mid X_i) = 0.1$ Under shoot

$P(X_{i+3} \mid X_i) = 0.1$ Over Shoot

Inaccurate Robot Motion

수학적인 확률적 관점에서 살펴보기 : Case3 – 균일한 분포 이동 ➔ 정답 모든 곳 0.2

U=7

0.2	0.2	0.2	0.2	0.2
			0.7	

$P(X_{i+2} \mid X_i) = 0.8$

$P(X_{i+1} \mid X_i) = 0.1$ Under shoot

$P(X_{i+3} \mid X_i) = 0.1$ Over Shoot

= 1

0.2 X 0.8

0.2 X 0.1 – 0.5

0.2 X 0.1 – 0.5

Robot Motion(Inexact)

```
p=[0, 1, 0, 0, 0]
world=['green', 'red', 'red', 'green', 'green']
measurements = ['red', 'green']
pHit = 0.6
pMiss = 0.2
pExact = 0.8
pOvershoot = 0.1
pUndershoot = 0.1

def sense(p, Z):
    q=[]
    for i in range(len(p)):
        hit = (Z == world[i])
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
    s = sum(q)
    for i in range(len(q)):
        q[i] = q[i] / s
    return q

def move(p, U):
    q = []
    for i in range(len(p)):
        q.append(p[(i-U)%len(p)])
    return q

print(move(p, 1))
```

#Modify the move function to accommodate the added
#probabilities of overshooting or undershooting
#the intended destination.

```
[0, 0, 1, 0, 0]
```

Robot Motion(Inexact)

```
p=[0, 1, 0, 0, 0]
world=['green', 'red', 'red', 'green', 'green']
measurements = ['red', 'green']
pHit = 0.6
pMiss = 0.2
pExact = 0.8
pOvershoot = 0.1
pUndershoot = 0.1

def sense(p, Z):
    q=[]
    for i in range(len(p)):
        hit = (Z == world[i])
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
    s = sum(q)
    for i in range(len(q)):
        q[i] = q[i] / s
    return q
```

```
def move(p, U):
    q = []
    for i in range(len(p)):
        exact = p[(i-U)%len(p)] * pExact
        under = p[(i-U-1)%len(p)] * pUndershoot
        over = p[(i-U+1)%len(p)] * pOvershoot
        q.append(exact+under+over)
    return q


print (move(p, 1)) # Move Once
```

```
[0.0, 0.1, 0.8, 0.1, 0.0]
```

Limit Distribution

U=1, 초기 분포 1, 나머지는 0으로 가정 할때
→ 이는 1을 오른쪽으로 옮기는 각 동작마다 0.8의 확률을 의미
이때 매우 많은 이동 단계를 로봇이 이동하다면 어떻게 될까?

U = 1

1	**0**	**0**	**0**	**0**

U = 1

U = 1

U = 1

…

Limit Distribution

초기 분포에서는 위치를 정확히 알고 있지만, 이동함에 따라 확률은 조금씩 낮아질수 있다..
0.8 … 0.64 ….?
절대적인 최소 정보는 → 균일한 분포로 수렴된다 (수많은 이동의 결과)→ valance property

U = 1

1	**0**	**0**	**0**	**0**

U = 1

U = 1

U = 1

…

0.2	**0.2**	**0.2**	**0.2**	**0.2**

Robot Motion(Inexact)

```
p=[0, 1, 0, 0, 0]
world=['green', 'red', 'red', 'green', 'green']
measurements = ['red', 'green']
pHit = 0.6
pMiss = 0.2
pExact = 0.8
pOvershoot = 0.1
pUndershoot = 0.1

def sense(p, Z):
    q=[]
    for i in range(len(p)):
        hit = (Z == world[i])
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
    s = sum(q)
    for i in range(len(q)):
        q[i] = q[i] / s
    return q
```

```
def move(p, U):
    q = []
    for i in range(len(p)):
        exact = p[(i-U)%len(p)] * pExact
        under = p[(i-U-1)%len(p)] * pUndershoot
        over = p[(i-U+1)%len(p)] * pOvershoot
        q.append(exact+under+over)
    return q


print (move(p, 1)) # Move Once
print (move((move(p, 1)), 1)) # Move Twice
```

```
[0.0, 0.1, 0.8, 0.1, 0.0]
[0.010000000000000002, 0.010000000000000002, 0.16000000000000003, 0.6600000000000001, 0.16000000000000003]
```

Robot Motion(Inexact)

```
p=[0, 1, 0, 0, 0]
world=['green', 'red', 'red', 'green', 'green']
measurements = ['red', 'green']
pHit = 0.6
pMiss = 0.2
pExact = 0.8
pOvershoot = 0.1
pUndershoot = 0.1

def sense(p, Z):
    q=[]
    for i in range(len(p)):
        hit = (Z == world[i])
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
    s = sum(q)
    for i in range(len(q)):
        q[i] = q[i] / s
    return q
```

```
def move(p, U):
    q = []
    for i in range(len(p)):
        exact = p[(i-U)%len(p)] * pExact
        under = p[(i-U-1)%len(p)] * pUndershoot
        over = p[(i-U+1)%len(p)] * pOvershoot
        q.append(exact+under+over)
    return q


print (move(p, 1)) # Move Once
print (move((move(p, 1)), 1)) # Move Twice

for _ in range(1000):
    p = move(p, 1)

print(p) # moved a thousand times
```

```
[0.0, 0.1, 0.8, 0.1, 0.0]
[0.010000000000000002, 0.010000000000000002, 0.16000000000000003, 0.6600000000000001, 0.16000000000000003]
[0.20000000000000365, 0.20000000000000373, 0.20000000000000365, 0.20000000000000035, 0.20000000000000035]
```

Entropy

https://www.youtube.com/watch?v=4G7DwrltLnY

Entropy

로봇의 모션 업데이트 단계 후에는, 엔트로피가 감소하고 측정 단계 후에 엔트로피가 증가한다고 언급한다.

의미하는 바는 측정 업데이트(감지) 단계 후에 엔트로피가 감소하고 이동 단계(이동) 후에 엔트로피가 증가한다는 것!!

일반적으로 엔트로피는 시스템의 불확실성 정도를 나타냅니다. 측정 업데이트 단계는 불확실성을 감소시키기 때문에 엔트로피는 감소합니다. 이동 단계는 불확실성을 증가시키므로 이 단계 이후에는 엔트로피가 증가한다.

로봇이 다섯 가지 다른 위치 중 하나에 있을 수 있는 현재 예를 살펴보겠습니다.

"최대 불확실성은 모든 위치가 동일한 확률을 가질 때 발생한다."

$$[0.2, 0.2, 0.2, 0.2, 0.2]$$

Entropy

$$[0.2, 0.2, 0.2, 0.2, 0.2]$$

$$Entropy = \Sigma(-p \times log(p)), \text{ we get}$$

$$-5 \times (.2) \times log(0.2) = 0.699.$$

측정을 수행하면 불확실성과 엔트로피가 감소합니다. 측정한 후 확률은 다음과 같습니다.
[0.05,0.05,0.05,0.8,0.05]

이제 우리는 로봇이 어디에 있는지 더 확실하게 추측할 수 있게 되었고 엔트로피는 0.338로 감소했습니다.

Robot Sense and Move

주어진 목록의 Motions=[1,1]은 로봇을 의미합니다.

오른쪽으로 이동한 다음 다시 오른쪽으로 이동하여 후방을 계산합니다.

분포 로봇이 먼저 빨간색을 감지한 다음 이동하면

오른쪽 하나, 녹색을 감지한 다음 다시 오른쪽으로 이동합니다.

균일한 사전 분포로 시작합니다.

```
p=[0.2, 0.2, 0.2, 0.2, 0.2]
world=['green', 'red', 'red', 'green', 'green']
measurements = ['red', 'green']
motions = [1,1]
pHit = 0.6
pMiss = 0.2
pExact = 0.8
pOvershoot = 0.1
pUndershoot = 0.1

def sense(p, Z):
    q=[]
    for i in range(len(p)):
        hit = (Z == world[i])
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
    s = sum(q)
    for i in range(len(q)):
        q[i] = q[i] / s
    return q

def move(p, U):
    q = []
    for i in range(len(p)):
        s = pExact * p[(i-U) % len(p)]
        s = s + pOvershoot * p[(i-U-1) % len(p)]
        s = s + pUndershoot * p[(i-U+1) % len(p)]
        q.append(s)
    return q
#
# ADD CODE HERE
#
print p
```

Robot Sense and Move

```
p=[0.2, 0.2, 0.2, 0.2, 0.2]
world=['green', 'red', 'red', 'green', 'green']
measurements = ['red', 'green']
motions = [1,1]
pHit = 0.6
pMiss = 0.2
pExact = 0.8
pOvershoot = 0.1
pUndershoot = 0.1

def sense(p, Z):
    q=[]
    for i in range(len(p)):
        hit = (Z == world[i])
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
    s = sum(q)
    for i in range(len(q)):
        q[i] = q[i] / s
    return q
```

로봇이 일정 거리를 이동한 후 감지.
그리고 각각 이후 사후 분포 신념으로 종료.

```
def move(p, U):
    q = []
    for i in range(len(p)):
        s = pExact * p[(i-U) % len(p)]
        s = s + pOvershoot * p[(i-U-1) % len(p)]
        s = s + pUndershoot * p[(i-U+1) % len(p)]
        q.append(s)
    return q

for i in range(len(measurements)):
    p = sense(p, measurements[i])
    p = move(p, motions[i])
print (p)
```

```
[0.21157894736842106, 0.1515789473684211, 0.08105263157894738, 0.16842105263157894, 0.38736842105263164]
```

Robot Sense and Move

```
p=[0.2, 0.2, 0.2, 0.2, 0.2]
world=['green', 'red', 'red', 'green', 'green']
measurements = ['red', 'green']
motions = [1,1]
pHit = 0.6
pMiss = 0.2
pExact = 0.8
pOvershoot = 0.1
pUndershoot = 0.1

def sense(p, Z):
    q=[]
    for i in range(len(p)):
        hit = (Z == world[i])
        #print((hit * pHit + (1-hit) * pMiss))
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
    s = sum(q)
    for i in range(len(q)):
        q[i] = q[i] / s
    return q
```

로봇이 빨간색을 두 번 감지하도록
이전 코드를 수정

```
def move(p, U):
    q = []
    for i in range(len(p)):
        s = pExact * p[(i-U) % len(p)]
        s = s + pOvershoot * p[(i-U-1) % len(p)]
        s = s + pUndershoot * p[(i-U+1) % len(p)]
        q.append(s)
    return q

for k in range(len(measurements)):
    p = sense(p, 'red')
    p = move(p, motions[k])

print (p)
```

```
[0.07882352941176471, 0.07529411764705883, 0.22470588235294112, 0.4329411764705882, 0.18823529411764706]
PS C:\work\python_work>
```

Basis for autonomous driving

- BELIEF = PROBABILITY
  - 도로가 있을수 있는 모든 위치에서 기능을 유지할수 있다.이러한 위치에서 각 셀은 확률값과 연결이 된다.

- SENSE = PRODUCT(by Nomalization)
  - 측정업데이트 함수 sense 는 정확한 측정치에 따라 확률값을 높이거나, 낮추는 곱이 된다.
  - 이 곱은 확률을 모두 더하면, 1이 된다는 원칙에 따라 곱 다음에 정규화를 진행한다.

- MOVE = CONVOLUTION
  - 이동 후 가능한 각 위치에 상황을 추측하고, 상응하는 확률을 수집한다.