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.

Fomal Definition of Probability

0 <= p(X) <= 1

확률함수 p(x) 는 하한과 상한이 0과 1 사이로 지정된 값이다.

- 그리드 셀이 <u>두개</u> 즉 x1 , x2 만 있다고 가정하자

$p(X_1) = 0.2$ 라면 , 이때 $p(X_2) =$ 값은 ?

Fomal Definition of Probability

0 <= p(X) <= 1

- $p(X_1) = 0.2$
- $p(X_2) = 0.8$

→ 확률 더해서 1 이 된다.

Fomal Definition of Probability

0 <= p(X) <= 1

확률함수 p(x) 는 하한과 상한이 0과 1 사이로 지정된 값이다.

- 그리드 셀이 두개 즉 x1 , x2 만 있다고 가정하자

$p(X_1) = 0$, 이때 $p(X_2)$ = 값은 ?

Fomal Definition of Probability

0 <= p(X) <= 1

- $p(X_1) = 0$
- $p(X_2) = 1$

→ 확률, 더해서 1 이 된다.

Fomal Definition of Probability

$$\mathbf{0 <= p(X) <= 1} \quad , \quad \sum \boldsymbol{p(Xi)} = \mathbf{1}$$

확률함수 p(x) 는 하한과 상한이 0과 1 사이로 지정된 값이다.

- 그리드 셀이 5개 있다고 가정하자.
- 이때 처음 4개의 셀이 0.1 임을 알고 있다. 가정하자

0.1	**0.1**	**0.1**	**0.1**	**?**

이때 $p(X_5)$ = 값은 ?

Fomal Definition of Probability

$$\mathbf{0 <= p(X) <= 1} \quad , \quad \sum \boldsymbol{p(Xi)} = \mathbf{1}$$

확률함수 p(x) 는 하한과 상한이 0과 1 사이로 지정된 값이다.

- 그리드 셀이 5개 있다고 가정하자.
- 이때 처음 4개의 셀이 0.1 임을 알고 있다. 가정하자

0.1	**0.1**	**0.1**	**0.1**	**?**

1 – (4 x 0.1) = 0.6

- https://www.youtube.com/watch?v=Y4ecU7NkiEI

Bayes Rule

X = grid cell , Z = Measurement 라 가정하자,

모든 가능한 위치에 대해, 빨간색 또는 녹색 그리드를 볼 수 있는 확률➔ 측정 확률

사전분포(사전확률)

$$p(X|Z) = \frac{p(Z|X)p(X)}{p(Z)}$$

$$\boldsymbol{p(z) = \sum_{i} p(Xi) = 1}$$

모든 I 에 대한 합이다.

Bayes Rule

$$\bar{p}(X_i|Z) \leftarrow p(Z|X)p(X)$$

사전분포*측정확률의 곱을, 비정규화 확률인 $\bar{p}$ 에 할당.

Bayes Rule

$$\alpha \leftarrow \sum \overline{p}(X_i|Z)$$

α는 모든 합을 의미한다.

$$p(X_i|Z) \leftarrow \frac{1}{\alpha}\ \overline{p}(X_i|Z)$$

그 다음 정규화를 진행,
결과 확률은 비정규화된 확률 x $1/\alpha$ 이다. (베이즈의 정리)

Localization 구현

- 로봇은 왼쪽, 오른쪽, 위, 아래로만 움직일 수 있다고 가정합니다. 대각선으로는 움직일 수 없습니다. 또한 이 임무에서 로봇은 목적지 광장을 결코 초과하지 않습니다. 움직이거나 정지 상태로 유지됩니다

```
# The function localize takes the following arguments:
#
# colors:
#       2D list, each entry either 'R' (for red cell) or 'G' (for green cell)
#
# measurements:
#       list of measurements taken by the robot, each entry either 'R' or 'G'
#
# motions:
#       list of actions taken by the robot, each entry of the form [dy,dx],
#       where dx refers to the change in the x-direction (positive meaning
#       movement to the right) and dy refers to the change in the y-direction
#       (positive meaning movement downward)
#       NOTE: the *first* coordinate is change in y; the *second* coordinate is
#             change in x
#
# sensor_right:
#       float between 0 and 1, giving the probability that any given
#       measurement is correct; the probability that the measurement is
#       incorrect is 1-sensor_right
#
# p_move:
#       float between 0 and 1, giving the probability that any given movement
#       command takes place; the probability that the movement command fails
#       (and the robot remains still) is 1-p_move; the robot will NOT overshoot
#       its destination in this exercise
#
# The function should RETURN (not just show or print) a 2D list (of the same
# dimensions as colors) that gives the probabilities that the robot occupies
# each cell in the world.
#
```

```
#
# Compute the probabilities by assuming the robot initially has a uniform
# probability of being in any cell.
#
# Also assume that at each step, the robot:
# 1) first makes a movement,
# 2) then takes a measurement.
#
# Motion:
#  [0,0] - stay
#  [0,1] - right
#  [0,-1] - left
#  [1,0] - down
#  [-1,0] - up

def sense(p, measurement, colors, sensor_right):

    a=[]
    for i in range(len(p)):
        q = []
        for j in range(len(p[0])):
            hit = (measurement == colors[i][j])
            q.append(p[i][j] * (hit * sensor_right + (1-hit) * (1-sensor_right)))
        a.append(q)
    si = sum(sum(a, []))
    for i in range(len(a)):  # Normalizing the distribution
        for j in range(len(a[0])):
            a[i][j] = a[i][j] / si
    return a
```

```
def move(p, motion, p_move):
    dy = motion[0]
    dx = motion[1]
    b = [[0.0 for row in range(len(p[0]))] for col in range(len(p))]
    for i in range(len(p)):
        for j in range(len(p[0])):
            s = p_move * (p[(i-dy) % len(p)][(j-dx) % len(p[i])]) # Movement takes place
            s += (1 - p_move) * p[i][j] # Movement doesn't take place
            b[i][j] = s
    return b


def localize(colors,measurements,motions,sensor_right,p_move):
    # initializes p to a uniform distribution over a grid of the same dimensions as colors
    pinit = 1.0 / float(len(colors)) / float(len(colors[0]))
    p = [[pinit for row in range(len(colors[0]))] for col in range(len(colors))]

    # >>> Insert your code here <<<



    # def sense(p, Z):
    for k in range(len(measurements)):
        p = move(p, motions[k], p_move)
        p = sense(p, measurements[k], colors, sensor_right)

    return p
```

```
def show(p):
    rows = ['[' + ','.join(map(lambda x: '{0:.5f}'.format(x),r)) + ']' for r in p]
    print ('[' + ',₩n '.join(rows) + ']')
```

Test Case

```
# test 1
colors = [['G', 'G', 'G'],
          ['G', 'R', 'G'],
          ['G', 'G', 'G']]
measurements = ['R']
motions = [[0,0]]
sensor_right = 1.0
p_move = 1.0
p =
localize(colors,measurements,motions,sensor_right,p_move)
show(p)

# correct_answer = (
#      [[0.0, 0.0, 0.0],
#       [0.0, 1.0, 0.0],
#       [0.0, 0.0, 0.0]])
```

```
# test 2
colors = [['G', 'G', 'G'],
          ['G', 'R', 'R'],
          ['G', 'G', 'G']]
measurements = ['R']
motions = [[0,0]]
sensor_right = 1.0
p_move = 1.0
p =
localize(colors,measurements,motions,sensor_right,p_move)
# correct_answer = (
#      [[0.0, 0.0, 0.0],
#       [0.0, 0.5, 0.5],
#       [0.0, 0.0, 0.0]])
```

Test Case

```
# test 3
colors = [['G', 'G', 'G'],
          ['G', 'R', 'R'],
          ['G', 'G', 'G']]
measurements = ['R']
motions = [[0,0]]
sensor_right = 0.8
p_move = 1.0
p =
localize(colors,measurements,motions,sensor_right,p_move)
# correct_answer = (
#      [[0.06666666666, 0.06666666666, 0.06666666666],
#       [0.06666666666, 0.26666666666, 0.26666666666],
#       [0.06666666666, 0.06666666666, 0.06666666666]])
```

```
# test 4
colors = [['G', 'G', 'G'],
          ['G', 'R', 'R'],
          ['G', 'G', 'G']]
measurements = ['R', 'R']
motions = [[0,0], [0,1]]
sensor_right = 0.8
p_move = 1.0
p =
localize(colors,measurements,motions,sensor_right,p_move)
# correct_answer = (
#      [[0.03333333333, 0.03333333333, 0.03333333333],
#       [0.13333333333, 0.13333333333, 0.53333333333],
#       [0.03333333333, 0.03333333333, 0.03333333333]])
```

Test Case

```
# test 5
colors = [['G', 'G', 'G'],
          ['G', 'R', 'R'],
          ['G', 'G', 'G']]
measurements = ['R', 'R']
motions = [[0,0], [0,1]]
sensor_right = 1.0
p_move = 1.0
p =
localize(colors,measurements,motions,sensor_right,p_move)
# correct_answer = (
#      [[0.0, 0.0, 0.0],
#       [0.0, 0.0, 1.0],
#       [0.0, 0.0, 0.0]])
```

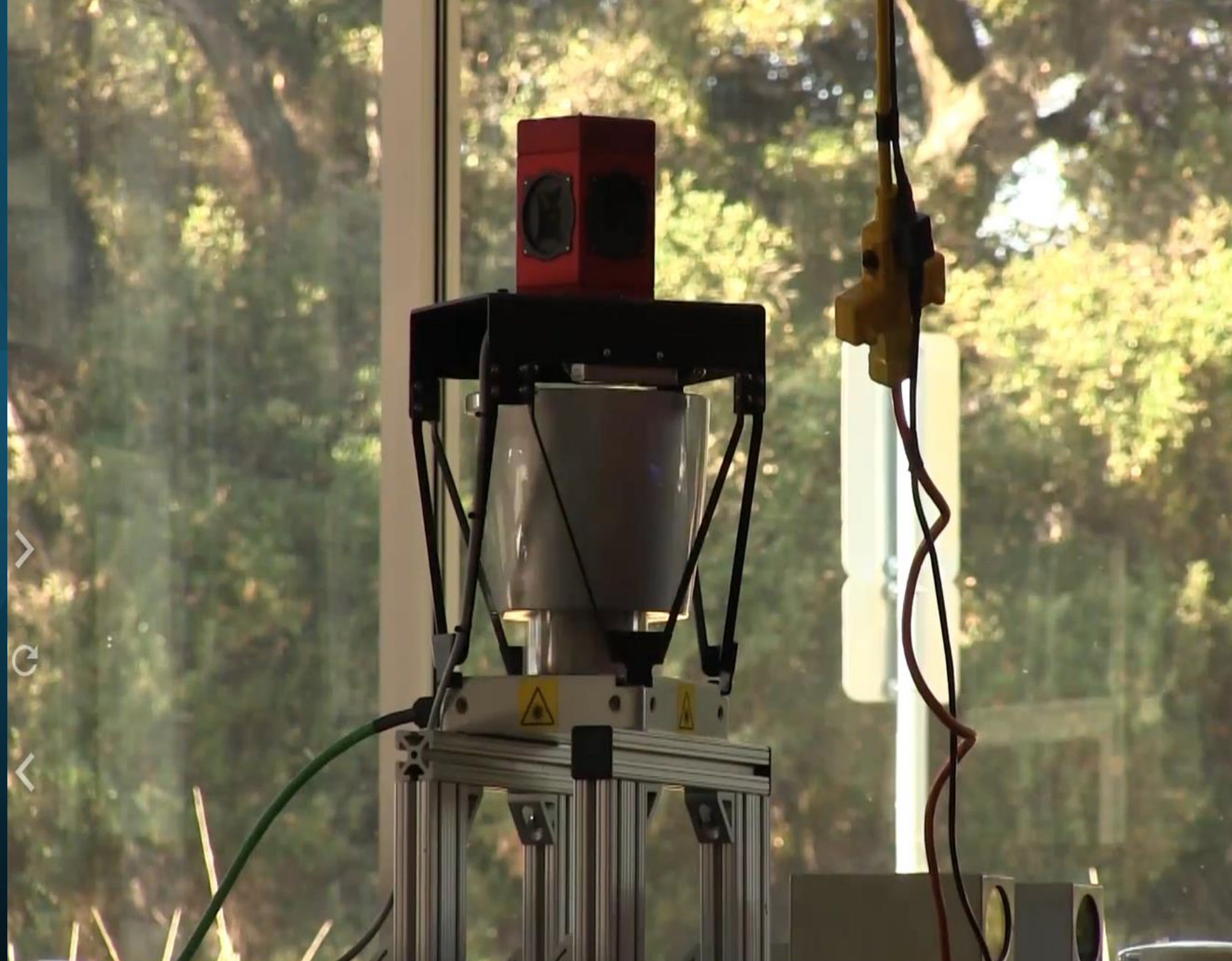
```
# test 6
colors = [['G', 'G', 'G'],
          ['G', 'R', 'R'],
          ['G', 'G', 'G']]
measurements = ['R', 'R']
motions = [[0,0], [0,1]]
sensor_right = 0.8
p_move = 0.5
p =
localize(colors,measurements,motions,sensor_right,p_move)
# correct_answer = (
#      [[0.0289855072, 0.0289855072, 0.0289855072],
#       [0.0724637681, 0.2898550724, 0.4637681159],
#       [0.0289855072, 0.0289855072, 0.0289855072]])
```

Test Case

```
# test 7
colors = [['G', 'G', 'G'],
          ['G', 'R', 'R'],
          ['G', 'G', 'G']]
measurements = ['R', 'R']
motions = [[0,0], [0,1]]
sensor_right = 1.0
p_move = 0.5
p =
localize(colors,measurements,motions,sensor_right,p_move)
# correct_answer = (
#     [[0.0, 0.0, 0.0],
#      [0.0, 0.33333333, 0.66666666],
#      [0.0, 0.0, 0.0]])
```

```
############################################## For the
following test case, your output should be
# [[0.01105, 0.02464, 0.06799, 0.04472, 0.02465],
#  [0.00715, 0.01017, 0.08696, 0.07988, 0.00935],
#  [0.00739, 0.00894, 0.11272, 0.35350, 0.04065],
#  [0.00910, 0.00715, 0.01434, 0.04313, 0.03642]]
# (within a tolerance of +/- 0.001 for each entry)

colors = [['R','G','G','R','R'],
          ['R','R','G','R','R'],
          ['R','R','G','G','R'],
          ['R','R','R','R','R']]
measurements = ['G','G','G','G','G']
motions = [[0,0],[0,1],[1,0],[1,0],[0,1]]
p = localize(colors,measurements,motions,sensor_right = 0.7, p_move = 0.8)
show(p) # displays your answer
```

Kalman Filters



- https://www.youtube.com/watch?v=mwn8xhgNpFY

필터(filter)란?

우선, 본격적인 설명에 들어가기에 앞서, '칼만 필터'의 이름에 들어있는 **'필터'라는 게 도대체 어떤 의미인지** 궁금해집니다. 주로, 제어이론이나 신호처리 등에서 '필터'는 측정값이나 신호에서 우리가 원하지 않는 불필요한 성분을 '제거(걸러내는)'하는 역할을 합니다. 예를 들어, 저주파 통과 필터(low-pass filter)는 어떤 신호에서 고주파 영역을 제외한 신호를 통과시키며, 고주파 통과 필터(high-pass filter)는 반대의 역할을 하겠죠. 일정 범위의 주파수만 통과시키는 band-pass filter, 일정 범위의 주파수만 차단하는 band-stop 필터 등도 있습니다.

칼만 필터는, 우리가 예측하고자 하는 **시스템의 동역학**(이론적 모델 및 현상 그 자체)이나 **측정값이 가진 불확실성**(노이즈, 잡음) 하에서, 이 두 데이터의 **적절한 균형(최적값)**을 찾기 위해 주로 쓰입니다. (아직, 무슨 말인지 와닿지 않습니다. 이에 대한 더 구체적인 상황 설명은 아래에서 다루겠습니다.) 이러한 관점에서, 칼만 필터는 신호처리에 쓰이는 '노이즈 제거 필터'와 비슷한 개념으로 볼 수 있습니다. '노이즈 제거 필터'의 가장 쉬운 예로는, 이미지 처리에 있어서의 미디언 필터(median filter)*나, 시계열 자료에도 적용가능한 이동평균 필터(moving average filter)** 등을 들 수 있습니다.

*이미지 주변 값들의 중간값으로 특정 픽셀값을 대체으로써, 주변에 비해 튀는 값을 가진 노이즈를 제거, 평준화 하는 필터(주변 값과 비슷하게 만듬

**예를 들어, 일단위 시계열 자료에서 시간적 변동이 너무 급격하거나, 일시적으로 튀는 값(노이즈)이 있는 경우, 현재로부터 일정 기간(예를 들어, 3일간)의 평균값을 구하여 현재의 값으로 쓰는 방법. 이를 통해, 노이즈가 제거되면서 전체적 트렌드는 유지하게 됨

즉, **칼만 필터**는 **시스템 동역학 모델과 측정값의 노이즈**를 최대한 **제거(필터링)**하면서, 불완전한 모델 예측값과 불완전한 측정값 사이의 어딘가에 있는, 최적의 시스템 상태를 **추정**(estimation)하는 알고리즘이라고 말할 수 있습니다.

칼만 필터는 어떤 상황에서 필요한지?

위에서 **추정**(estimation)이라는 말이 나왔습니다. 그렇다면, **우리가 다루려는 대상**을 **수학적 모델을 통해 '예측**(prediction)'하거나, 직접 **'측정**(measurement, observe)'하면 될 것 같은데, **도대체 왜 '추정'이라는 개념이 필요한 걸까요?** 이 부분은 아래의 [예시]를 통해 좀 더 자세히 설명해 보고자 합니다.

아래의 [예시]는 비교적 단순하지만, **칼만 필터**가 **1960년대 아폴로 계획**에서 **우주선을 달에 보내기 위해 쓰여졌다는 사실**을 아래 예제와 비교하여 떠올려 본다면, 좀 더 쉽게, 왜 칼만 필터가 필요한지 와닿을 듯 합니다. 당시의 제한된 지식과 측정장비만으로 우주인들을 달에 보냈다가 다시 지구로 데려오려면, 우주선의 현재 위치와 궤도를 정확히 알아내는 게 상당히 중요했을테니까요.

게다가 당시의 컴퓨터 성능을 고려한다면, 그것을 계산하는 방법이 이론적으로 명확하면서도, 상당히 간단하고 효율적이어야 했을 겁니다. 그걸, 칼만 필터가 가능하게 한거죠.

[예시] 자동차의 현재 위치를 알아내는 문제

▶ **[목적] 어떤 자동차의 정확한(가장 그럴법한) 현재 위치를 알아내는 것**

▶ **[방법]** 이를 위해, 우리는 자동차의 운동을 수학적으로 표현한 **'시스템 동역학 모델 (정보 1)'**과 여러가지 소스를 통해 **'실제 측정한 데이터 (정보 2)'**를 이용할 수 있습니다. 이 두가지 정보를 잘 조합함으로써, 우리는 최적의 자동차 위치를 알아내야 합니다. 하지만, 실제 상황에서는 이게 생각보다 쉽지 않은데요. **이 두가지 정보**가 가진 **문제점과 한계**에 대해 **가볍게 고찰**해 보고자 합니다.

▶ **[정보 1 -시스템 동역학]**에 대한 고찰

우리는 고등학교 때 배운 운동법칙을 적용하여, 자동차의 속도, 가속도, 위치의 관계를 수학적으로 표현할 수 있습니다. 이것이 우리가 가진 **수학적인 '시스템 동역학'**입니다. 하지만, 실제 상황에서는, 이 수학적 모델은 아주 이상적인 조건(노면이 완전한 평면, 노면 마찰력, 공기 저항, 타이어 압력, 자동차의 출력, 엔진의 성능 및 마모도, 완벽히 정확한 속도계 등)이 모두 충족되어야만 자동차의 위치를 정확히 예측할 수 있을 것입니다. 그리고 직관적으로 알 수 있듯이, 이 모든 것을 정확히 아는 것은 '거의 불가능'합니다.

즉, 우리가 다루는 시스템이 복잡해질수록, 그 시스템을 완벽하게 수학적으로 표현하는 것은 불가능하다고 할 수 있습니다. 여기서, 바로 우리가 가진 수학적 시스템 동역학 모델에 '프로세스 노이즈'라는 개념을 추가하게 됩니다. 즉, 모델 자체가 완벽하지 않고, **모델을 이용한 예측값 자체가 '우리가 가진 지식과 수학적 모델로는 표현이 불가능한 어떤 불확실성을 가진다'**고 보는 것입니다.

▶ **[정보 2 - 센서 또는 측정값]**에 대한 고찰

한편, 일반적으로 우리는 어떤 대상을 실제로 측정한 값이 참값(true value)라고 받아들이기 쉽습니다. 하지만 많은 경우, 이것은 사실이 아닙니다. 기본적으로 **우리의 측정값은 잡음(노이즈)을 가집니다**. 그 원인의 하나로는 우선, '측정 장비나 기술, 측정하고자 하는 대상 자체가 가진 근본적 불확실성 문제'가 있고, 또한, '직접 측정할 수 없거나 일부만 측정 가능한 대상'에 대한 문제가 있습니다.

- **측정과정이나 대상의 불확실성:** 실제 상황에서 이뤄지는 모든 측정값은 기본적으로 불확실성을 가집니다. 센서를 이용한 측정의 경우, 각종, 물리적, 화학적, 전기적 노이즈로 인해 오차를 가지며, 불균질한 어떤 대상의 특성(예를 들어, 실제 하천의 수질)을 측정하는 경우, 우리의 측정값이 시공간적으로 해당 대상의 특성을 대표할 수 있는지에 대한 문제가 생깁니다.

또한, 수질 측정을 예로 들면, 수질 시험실에서 습식 시험 또는 기기 분석을 통해 얻은 결과를 거의 참값처럼 여기는 경우가 있는데, 사실, 수질오염 공정시험법을 보면, 모든 수질분석 방법은 각 항목마다 시험법 자체의 정확성을 말해주는 '표준편차'를 제시하고 있습니다. 즉, 시험실에서 분석한 수질 농도값은 하나의 수치가 아니라, 사실은 어떤 '분포(distribution)'를 갖는 것으로 해석해야 한다는 말입니다. 따라서, 상대적으로 더 정확한 수질 농도를 얻기 위해서는, 여러번의 반복 실험(시료 샘플링시 시공간적으로 좀 더 넓은 범위를 택하면, 대표성이 높아지겠죠.)을 함으로써, 실제 '평균'에 가까운 값을 구해야 합니다.

- **직접 측정할 수 없거나 일부만 측정가능한 대상:** 또한, 우리는 제한적인 측정값 만으로 결론을 내야할 수 있습니다. 로켓 엔진의 내부 농도를 알기 위해 엔진 표면에 설치한 센서값을 이용해야 한다던지, 호소의 수질을 알기 위해 일부 대표 지점의 측정 자료만 활용해야 한다던지, 자동차의 위치를 알기 위해 GPS정보 없이 (터널 내부 처럼) 속도계와 가속도계에 의지하여 위치를 추정해야 한다던지 하는 경우입니다.

이런 경우, 우리는 측정값을 바탕으로, (비록 불확실하지만) 우리가 가진 시스템 동역학에 대한 수학적 표현과 이러한 제한된 측정 정보들을 통해, 시스템의 현재 상태를 추정해야 합니다. (자동차의 속도를 이용해 일정 시간 뒤의 다음 위치를 예측한다던지 하는 일이죠. 하지만, 이 경우, 시작 위치(초깃값)를 알아야 합니다.)

▶ 칼만 필터가 하는 일

위에서 다룬 내용들이, 우리가 다루는 문제에서 어떤 답(자동차의 위치)를 얻으려고 할 때, 오직 모델을 통해 '예측(prediction)'만 하거나, 반대로 단순히 '측정(measurement, observe)'만 하는 걸로는 부족한 이유입니다.

그리고 **칼만 필터**는 위와 같은 상황에서, **수학적인 [시스템 동역학 모델]**과 **노이즈가 섞이거나 제한적으로 측정된 [실측 데이터]**를 동시에 고려하여, **현재 자동차의 가장 그럴듯한**(추정 오차를 최소화하는) **위치**를 **'추정'하는 알고리즘**이라고 할 수 있습니다.

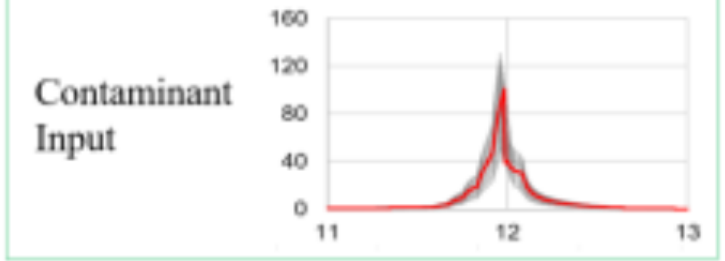
- **At every time step...**

Hydrologic and hydraulic model (pipedream)

Green Ampt and Saint-Venant Equations

Hydraulic model results (flow rates and depths)

Uncertain model input

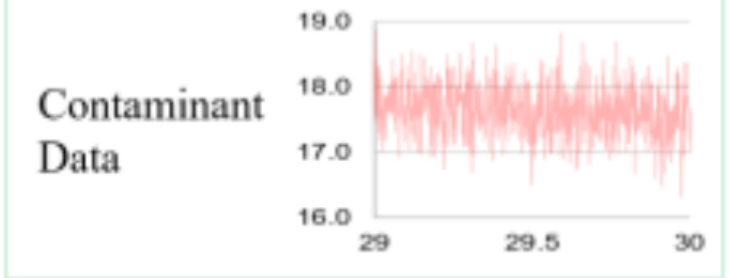


New water quality model (pipedream-WQ)

Advection-Reaction-Diffusion Equation

Water quality **model results** including the **input uncertainty**

Noisy sensor measurement



Kalman Filter

State estimation

$$\hat{\mathbf{x}}_{t+\Delta t} = A_t\hat{\mathbf{x}}_t + B_t\mathbf{u}_{t+\Delta t} + \mathbf{y}_t + L_{t+\Delta t}[\mathbf{z}_{t+\Delta t} - \hat{\mathbf{z}}_t]$$

$$\hat{\mathbf{z}}_t = H_{t+\Delta t}(A_t\hat{\mathbf{x}}_t + B_t\mathbf{u}_{t+\Delta t} + \mathbf{y}_t)$$

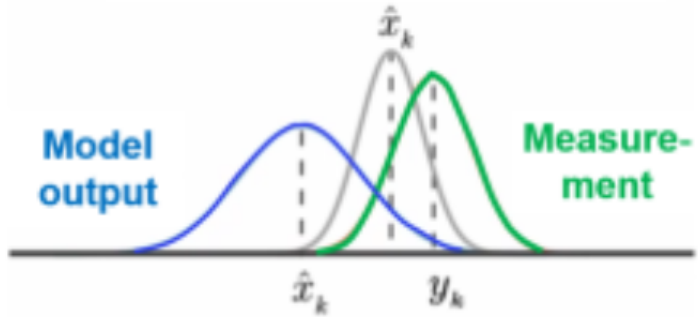
$$L_t = P_tH_t^T(H_tP_tH_t^T + V_t)^{-1}$$

$$P_{t+\Delta t} = A_t(P_t - P_tH_t^T(H_tP_tH_t^T + V_t)^{-1}H_tP_t)A_t^T + W_t$$

Data assimilation

Update state of physics-based model

Optimal estimation of states



수리&수질 모형에서, 불확실성을 가진 모델 입력+노이즈를 가진 센서 데이터를 바탕으로, 칼만 필터를 이용하여 시스템 상태를 추정(estimation)하고, 모델을 업데이트(data assimilation)하는 과정의 모식도,

칼만 필터는 어떤 가정을 바탕으로 하며, 어떤 특징을 갖는지?

알고리즘 유도 과정에서도 나중에 다루겠지만, 먼저, **칼만 필터가 갖는 특성**과 **전제 조건(가정)**에 대해 간략히 언급하고 넘어가겠습니다. 흔히, **칼만 필터**를 소개하는 이름 앞에는 아래와 같은 **다양한 수식어가 나열**됩니다. 여기에 등장하는 용어 중심으로 **칼만 필터의 특징과 가정 조건**들에 대해 살펴보겠습니다.

> *칼만필터는 재귀적(recursive) 최적 선형 제곱(optimal linear quadratic) 추정(estimation) 알고리즘*

[가정] 선형(linear)

가장 중요한 가정의 하나입니다. 기본적인 선형 칼만 필터(original Kalman filter)는 우리가 다루는 시스템 동역학이 선형성을 만족할 때에만 적용할 수 있습니다. 즉, 우리가 시스템에 대한 입력을 2배로주면, 2배의 결과가 나오고, 여러 개의 입력을 갖는 경우에도, 시스템의 출력(예측값)은 입력 사이의 선형 결합으로 표현될 수 있어야 한다는 의미입니다.

만약, 우리가 다루려는 시스템이 비선형적인 반응이나, 동역학을 포함하고 있다면, 선형 칼만 필터가 아닌, 확장 칼만 필터(Extended Kalman filter)나 앙상블 칼만 필터 등을 이용해야 합니다.

[가정] Gaussian 노이즈

위에서 개념적으로 설명드렸듯, 우리가 다루는 시스템 동역학은 불확실성(프로세스 노이즈)을 갖고, 측정값 역시 불확실성(관측 노이즈)을 갖습니다. 그리고 노이즈는 불규칙하게(랜덤하게) 발생하지만, 어떤 분포(distribution)를 따른다고 가정할 수도 있습니다.

그리고, **칼만 필터**는 **프로세스 노이즈와 관측 노이즈가 모두 평균이 0인 정규분포**(zero mean Gaussian distribution)**를 따른다고 가정**합니다. 이 둘의 분포를 정규분포로 가정하기 때문에, 우리는 해석적인(analytical) 수식 유도를 통해 칼만 필터 알고리즘을 얻을 수 있게 됩니다. 바로, 정규분포의 곱은 또다른 정규분포가 된다는 아주 편리한 성질을 이용할 수 있기 때문입니다.

만약, 시스템의 프로세스 노이즈나 측정값의 노이즈가 **zero mean Gaussian 분포를 따르지 않는다면**(사실, 실제 세상의 시스템은 대부분 비선형이죠.), 이를 극복하기 위해 **일반적인 칼만 필터가 아닌, Unscent Kalman filter, Particle filter 등의 다른 방법을 이용**하여야 하며, 이 경우, 정규분포를 가정했을 때 처럼, 해석적으로(analytical) 알고리즘을 유도할 수 없기 때문에, 특정 포인트들을 샘플링하고, 그것을 통해 실제 분포(정규분포가 아닌)를 나타낼 수 있는, Monte Carlo 법과 Bayesian filtering의 조합이 필요하게 됩니다.

[가정/특징] 재귀적(recursive)

재귀적이라는 것은, 우리가 칼만 필터를 적용할 때, 오직 **이전 시간(n)과 현재 시간(n+1)의 시스템 상태에 대한 관측값과 시스템 입력값들만 이용하여 계산할 수 있다는 의미**입니다. 다시 말해, 이전 시간(n) 에서의 시스템의 상태(state)가 'n-1시간~ 아주 오래된 과거'의 누적된 입력에 의한 모든 결과(현상)을 나타낸다고 가정하기 때문에, 우리는 칼만 필터 적용 과정에서 n-1시간 이전의 입력이나 시스템 상태를 고려할 필요가 없으며, 오직 한번에 하나의 time step(n에서 n+1사이)에 대해서만 칼만 필터를 적용하면 된다는 뜻입니다. 즉, 칼만 필터에는 1차 Markov process를 따른다는, 기본 가정이 포함됨을 의미합니다.

바로 이러한 성질 때문에, 우리는 과거(n-1시간 이전)에 대한, 모든 시간 단계마다의 시스템 상태(state)를 따로 기억하고 있을 필요가 없게 되며(컴퓨터 메모리에 저장할 필요가 없음), 이것이 칼만 필터의 적용을 효과적으로 만듭니다.(아폴로 계획 당시, 1960년대 컴퓨터로도 충분히 계산이 가능했던 이유죠.) 또한, 이러한 특성으로 인해 칼만 필터를 단순한 대상에 적용할 경우, 많은 계산량을 필요로 하지 않으므로, 실시간 시스템이나 임베디드 시스템에도 무리없이 탑재할 수 있게 됩니다.

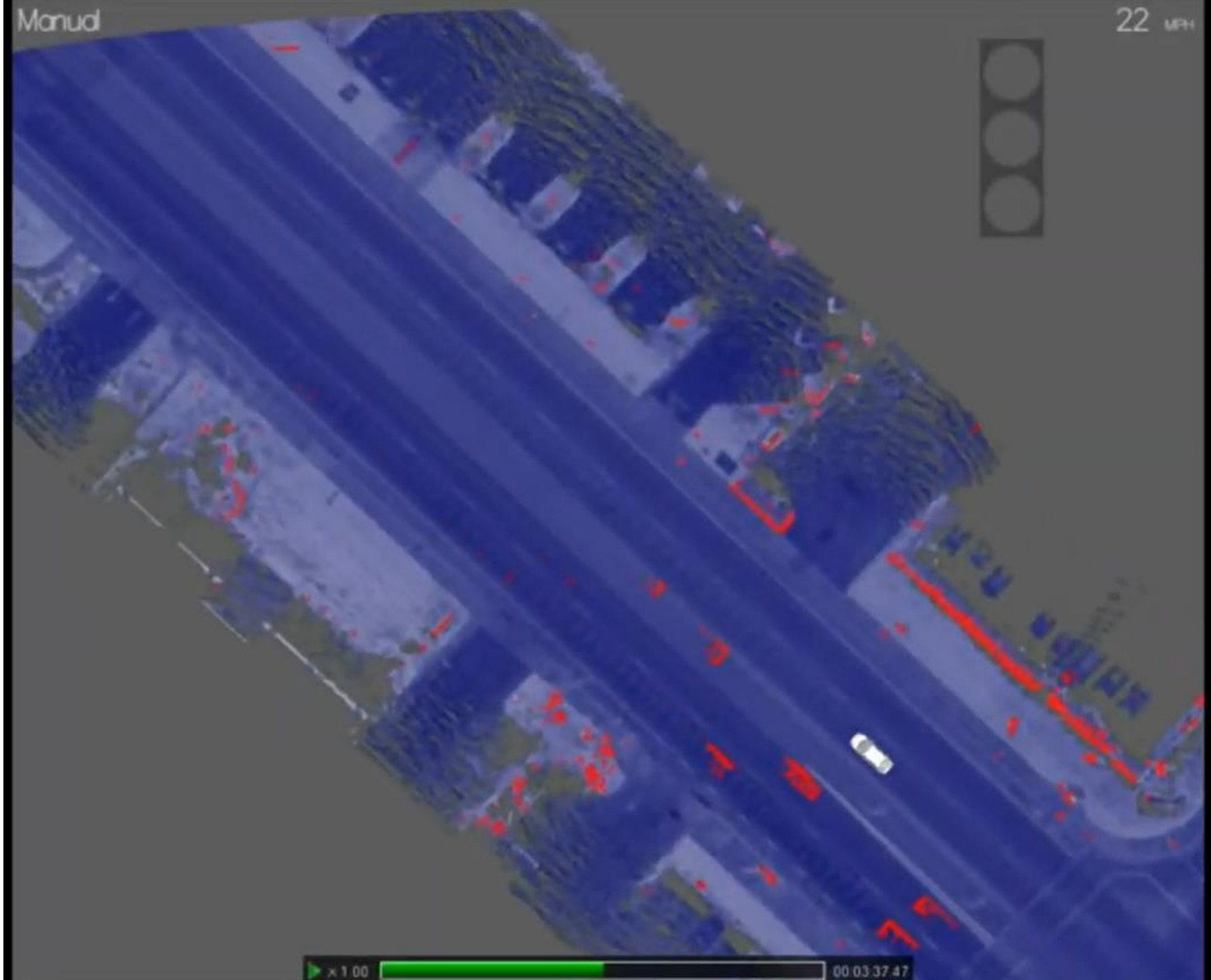
[특징] 최적(optimal)

칼만 필터는 위 아래에 나열된 가정과 조건 하에서는, 최적의 추정값을 제공합니다. 즉, 칼만 필터에 의해 추정된 시스템의 상태는 추정 오차(실제 참값과 우리가 추정한 값의 차이)를 최소화하는 결과값들로 이뤄집니다. 다시말해, **칼만 필터 알고리즘**은 **주어진 동역학 모형과 제약 조건, 가정 하**에서, **추정 오차를 최소화하는 최적화 문제를 풀어서 정리하여 얻어낸 계산식**이라고 할 수 있습니다. 그리고 이 과정(알고리즘)을, 최적의 칼만 게인(Kalman gain)을 얻도록 해줍니다. 왜, 여기서 게인(gain)이라는 말이 나오고 그게 무슨 뜻인지는 추후 상태 추정자(state observer)에 대해 이야기 할 때, 다시 살펴보도록 하겠습니다.

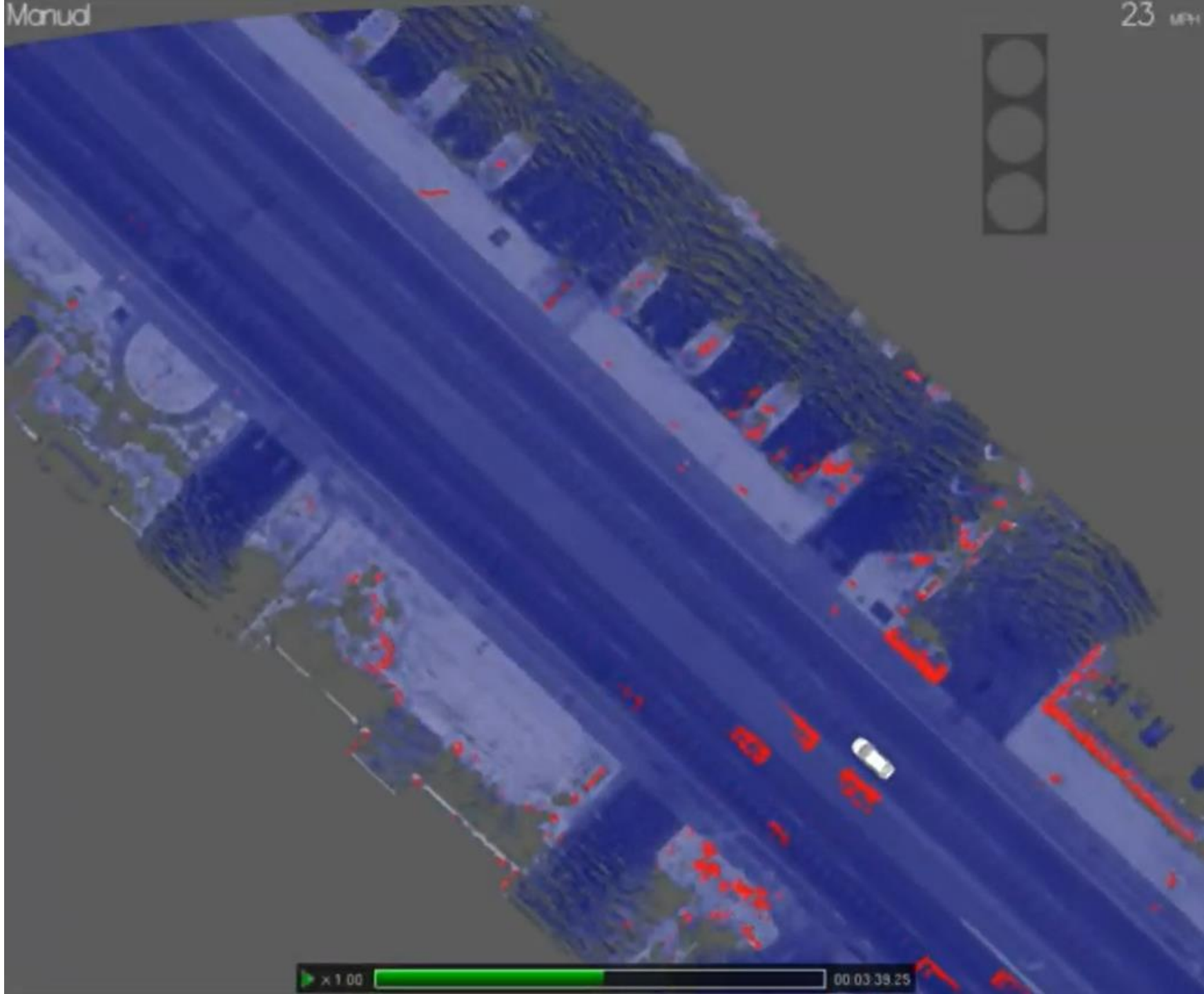
그 밖의 가정

칼만 필터는 **초기의 오차 공분산 행렬과, 프로세스 노이즈 공분산 행렬, 관측 노이즈 공분산 행렬**을 **알 수 있다(known)고 가정**합니다. 따라서, 칼만 필터를 적용하려면, 사용자가 이러한 행렬을 문제의 성격에 맞게 적절하게 정의해서 입력해줘야 하며, 이 과정에 대한 정확한 이론적 근거가 없는 경우에는, 여러가지 값들을 실제 문제에 적용해가면서 시행착오를 통해 경험적으로 가장 좋은 결과를 낼 수 있는 값들을 찾아가는 보정(calibration) 과정이 필요할 수 있습니다.

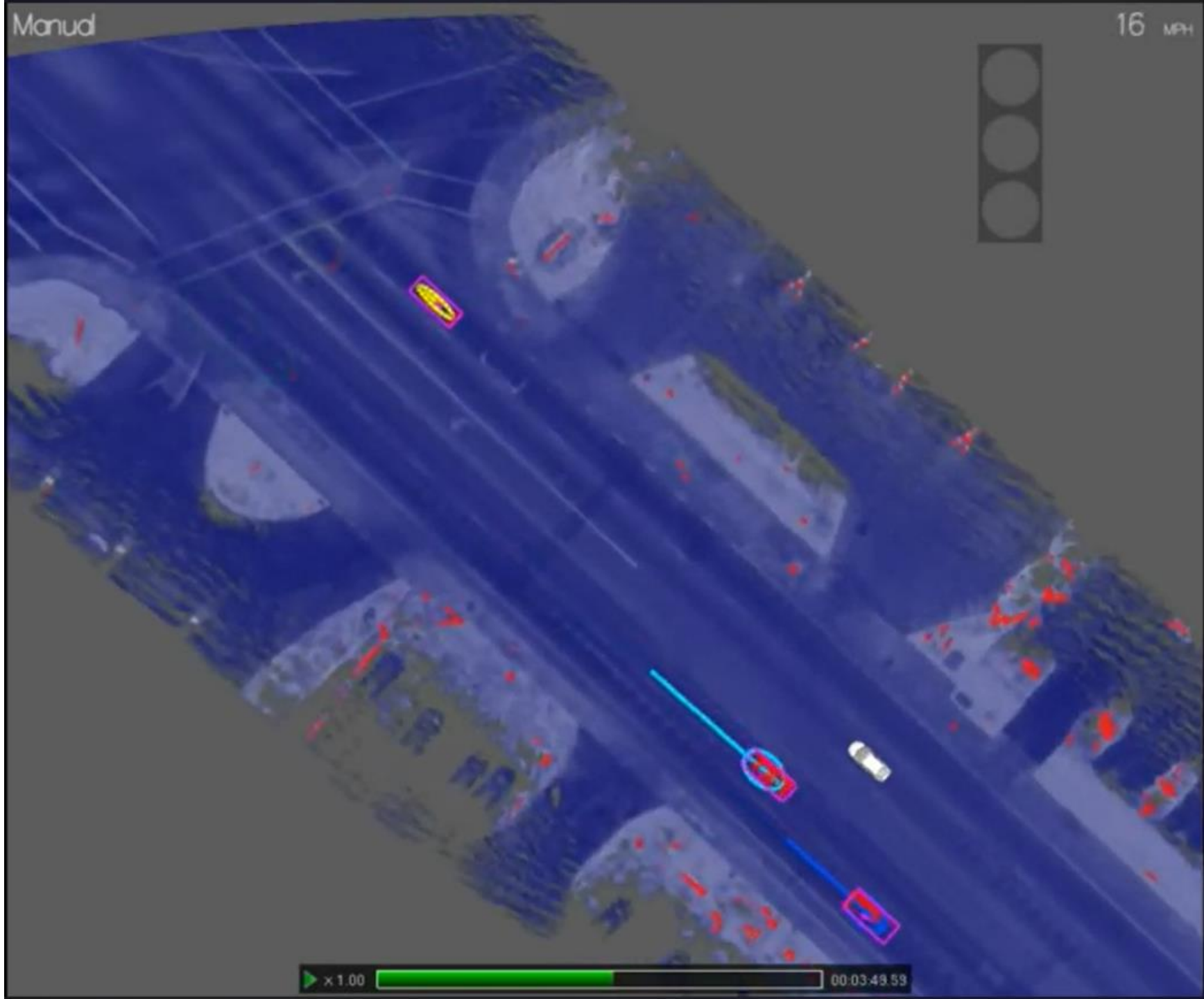
- Google self-driving car localizing itself using roadmap

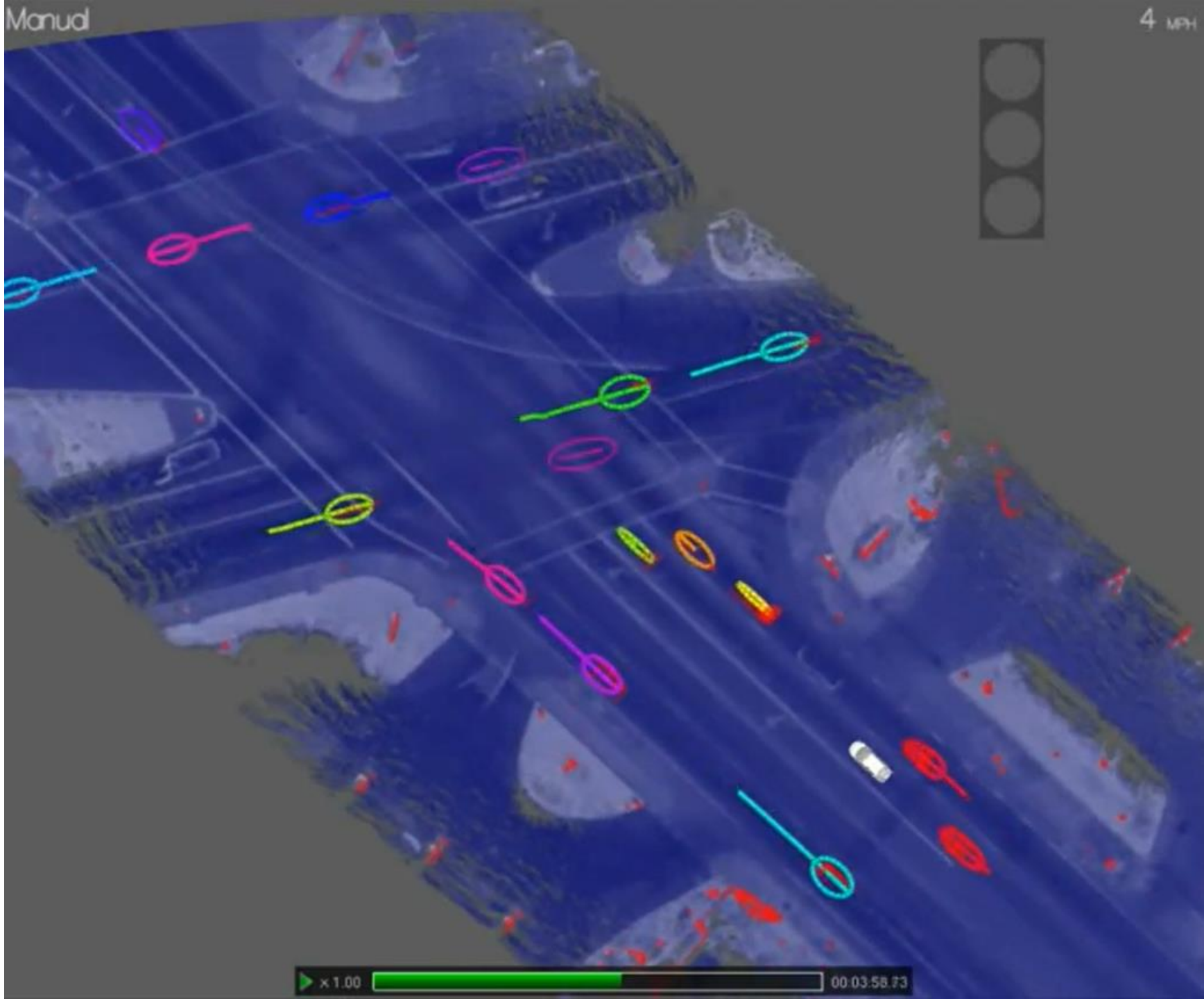


- 빨간 색 블록은 라이다로 추적한 상대방 차량이다.



- 상대방 차를 찾는 이유는 부딪치지 않기 위해서.



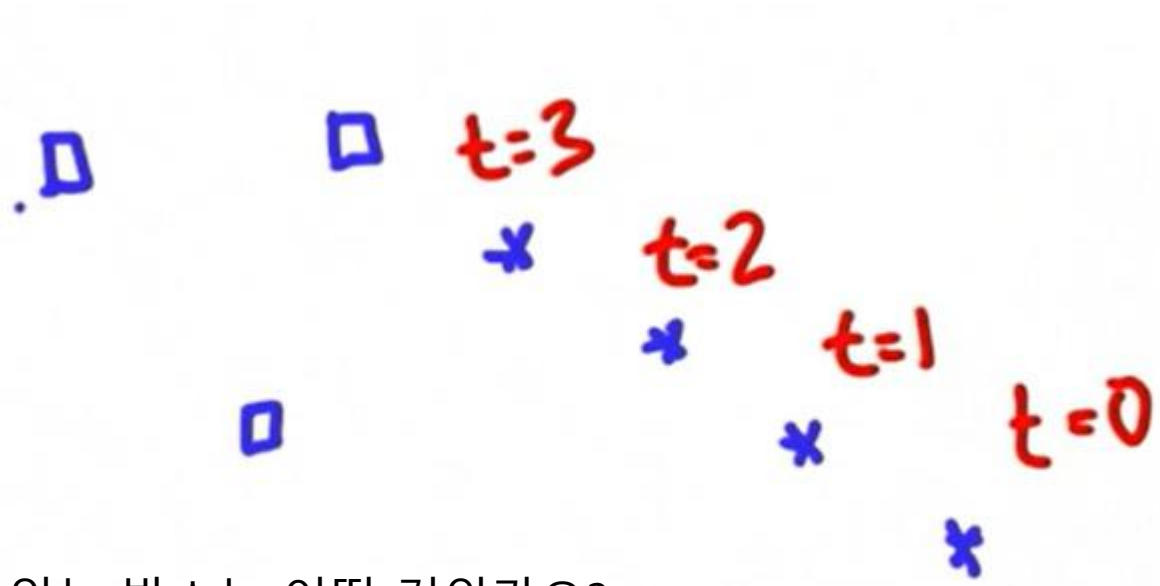

Manual
4 MPH
x 1.00
00:03:58.73

Tracking(로봇의 위치및 다른 차량 추적)

- KALMAN Filter
- Continuous world
- Unimodal distribution

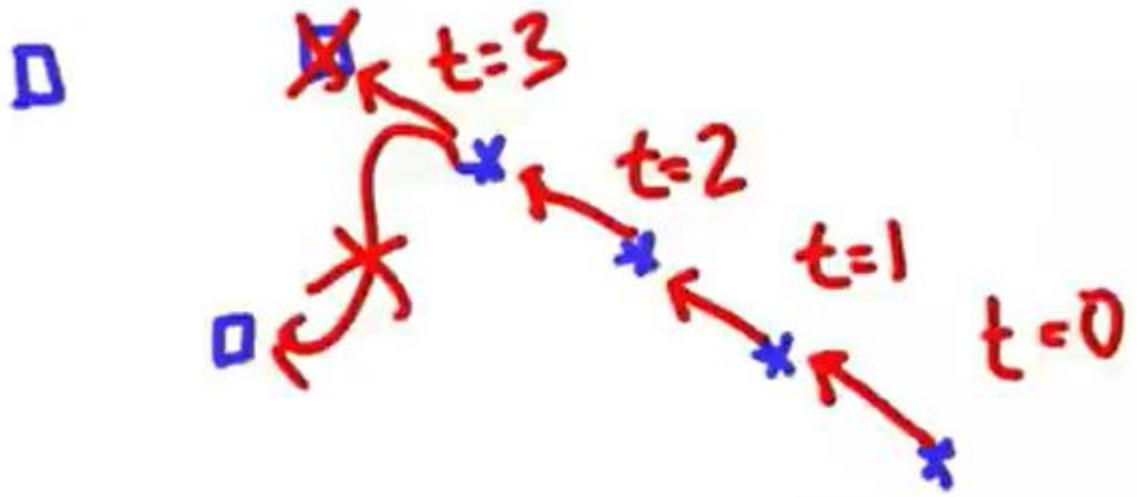
- Monte carlo Localization
- Discrete places
- Multimodal distribution

Kalman Filter



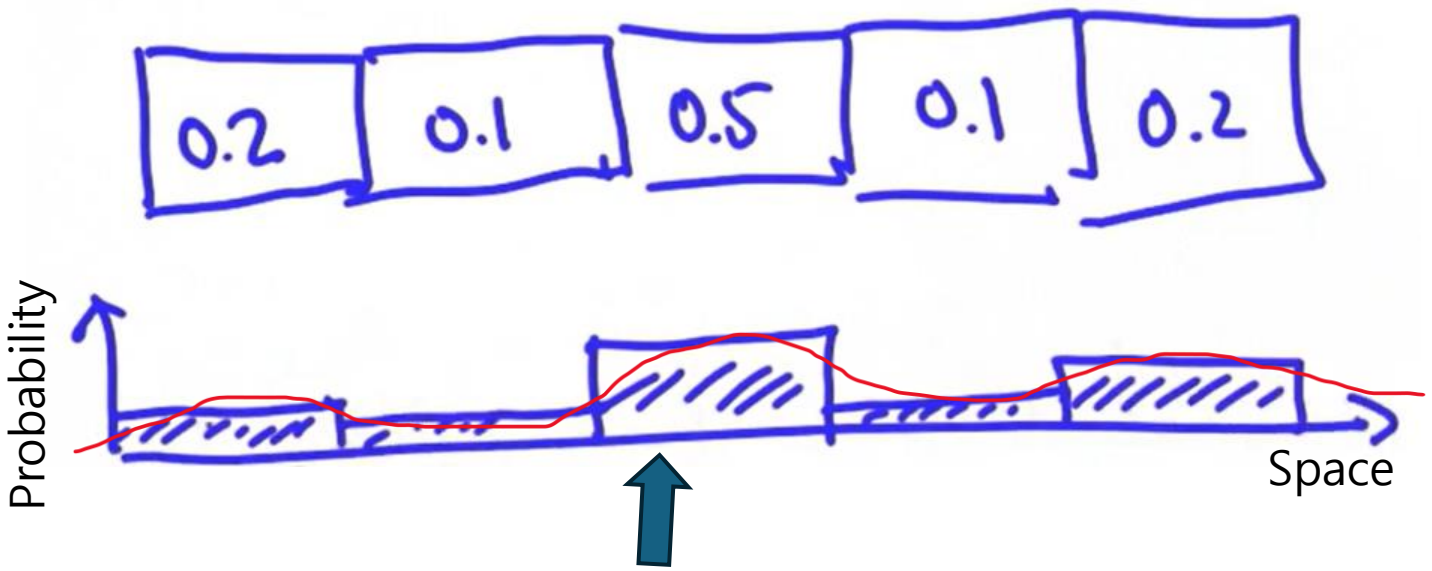
자동차에서 물체를 인식할때
사각형중 t=4 라고 인식할수 있는 박스는 어떤 것인가요?

Kalman Filter

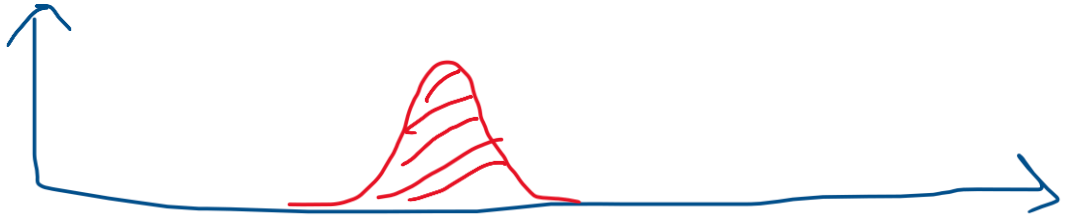


속도에 급격한 변화가 업다고 가정할때, 속도가 벡터의 방향을 가리킨다라 얘기할수 있다.
칼만 필터는 이런 관찰을 통해서, 데이터를 기준으로 미래의 위치와 속도를 추정할수 있다.
데이터(레이더 거리데이터) 를 근거로 다른 차량이 어디 있는 지 파악해볼수 있다.

Gaussian Intro

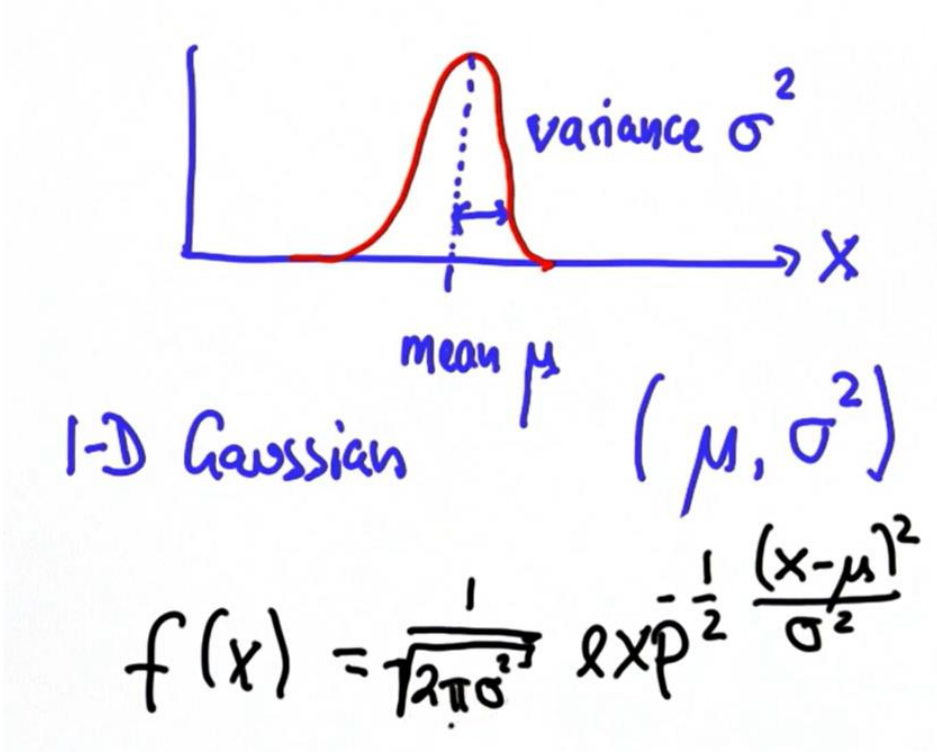


연속 공간을 유한한 그리드로 나누고 사후 분포값을 표현→히스토그램
히스토그램은 연속 분포에 대한 단순화 시킨 근사치 값이다.



칼만 필터에서 분포는 가우시안 함수에 의해서 구성됩니다.
가우시안은 위치공간에 연속적인 함수이다.아래 영역의 합은 1이다.

Gaussian Intro

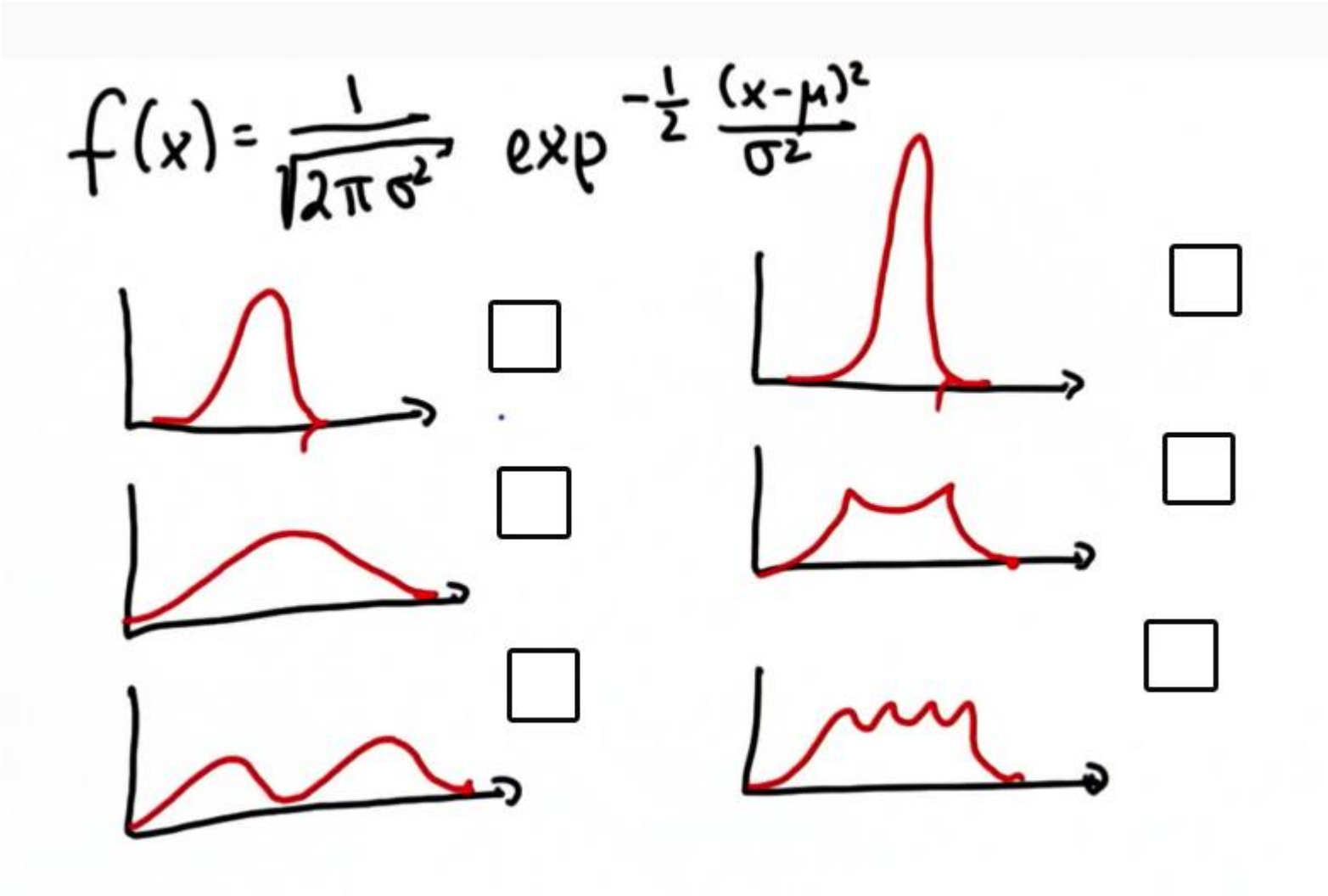


공간을 x , 가우시안은 두개의 변수 로 구성
평균: μ, width : $\sigma 2$

: 객체의 추정치를 얻기위한 파라미터 구성

: 평균 뮤에 대한 이차 차이를 얻기위한 공식을 구성

Gaussian Intro:가우시안 찾기

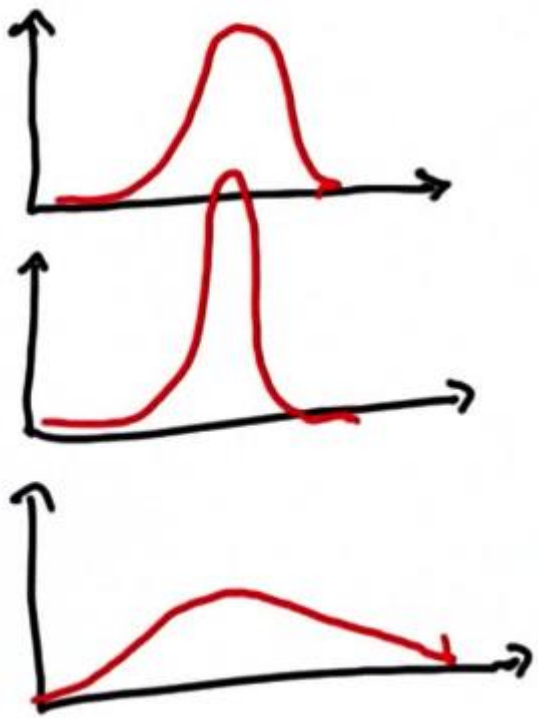


대칭인 양측에서 지수 Drop-off 효과 발생, 하나의 피크를 갖는다.

variance Comparison

각 그래프의 공분산의 크기를 선택해보세요.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp -\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}$$

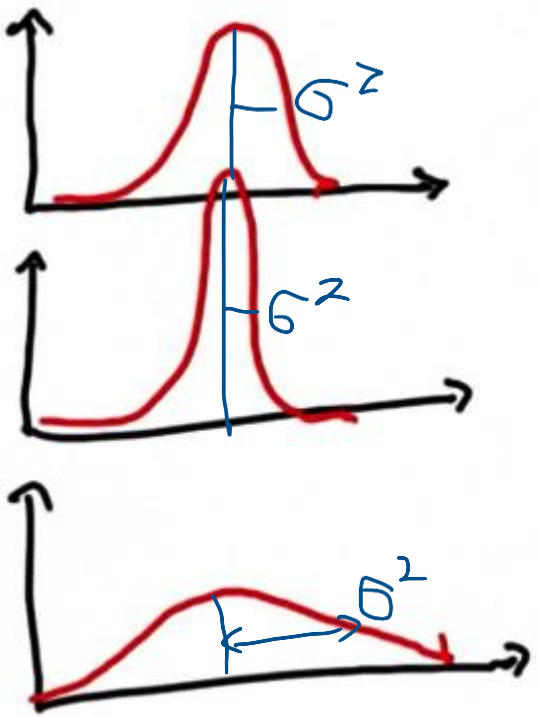


σ^2

	LARGE	MEDIUM	SMALL
	☐	☐	☐
	☐	☐	☐
	☐	☐	☐

variance Comparison

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}}$$



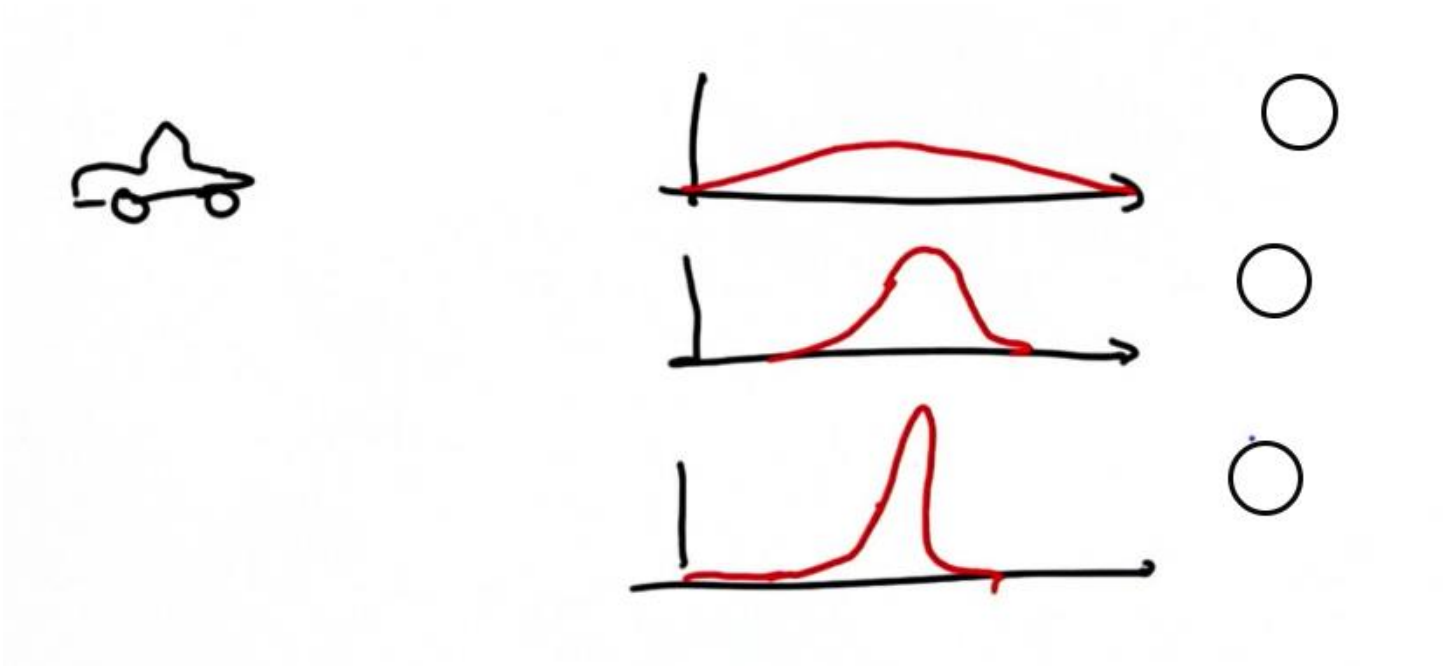
σ^2

LARGE	MEDIUM	SMALL
☐	☑	☐
☐	☐	☑
☑	☐	☐

X와Y 간의 차이가 공분산에 의해 정규화 된다.
넓은 분포를 가지는 함수가 최대 공분산, 작은 분포→ 최소 공분산
또 시그마 제곱 공분산은 → 불확실성 척도→클수록 실제상태가 더 불안해진다.

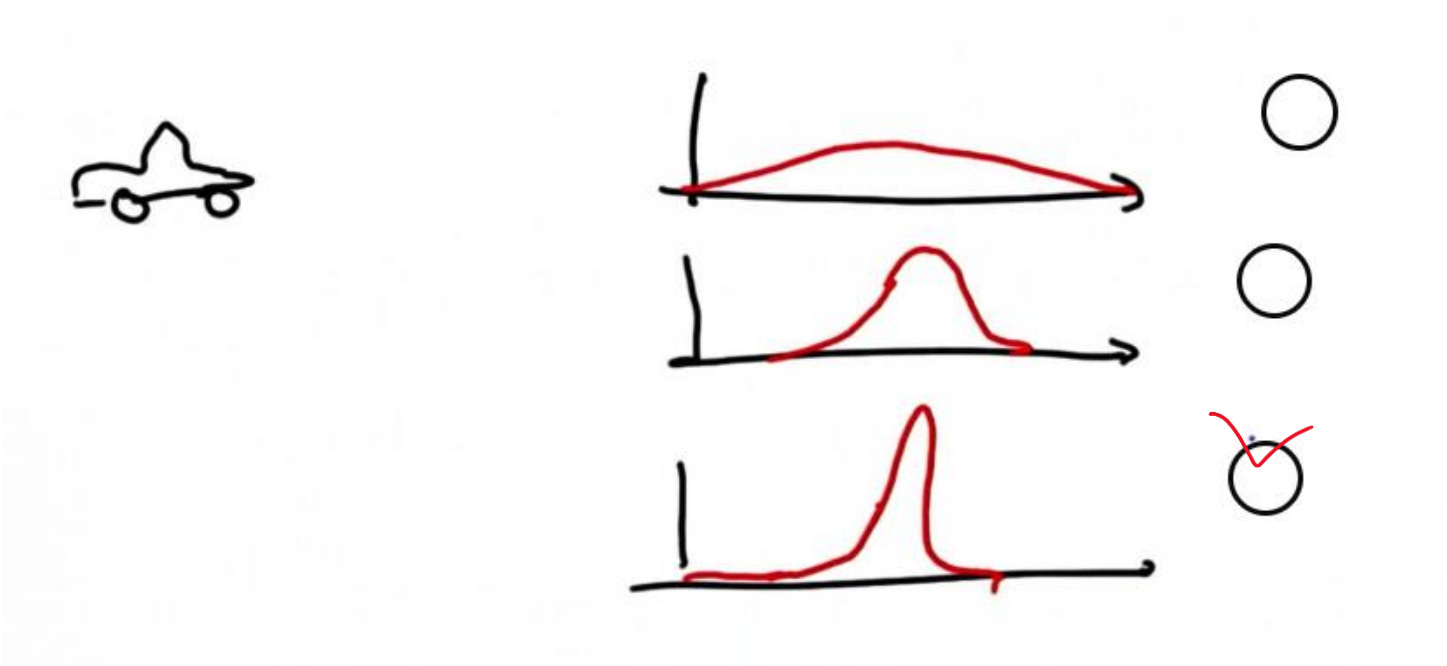
Preferred Gaussian

다른 차를 추적할때 선호되는 가우시안 그래프는?



Preferred Gaussian

3번: 다른것에 비해 분포가 확실하다



Evaluate Gaussian

다음 가우시안 함수 값을 구해보자

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}}$$

$\mu = 10$

$\sigma^2 = 4$

$x = 8$

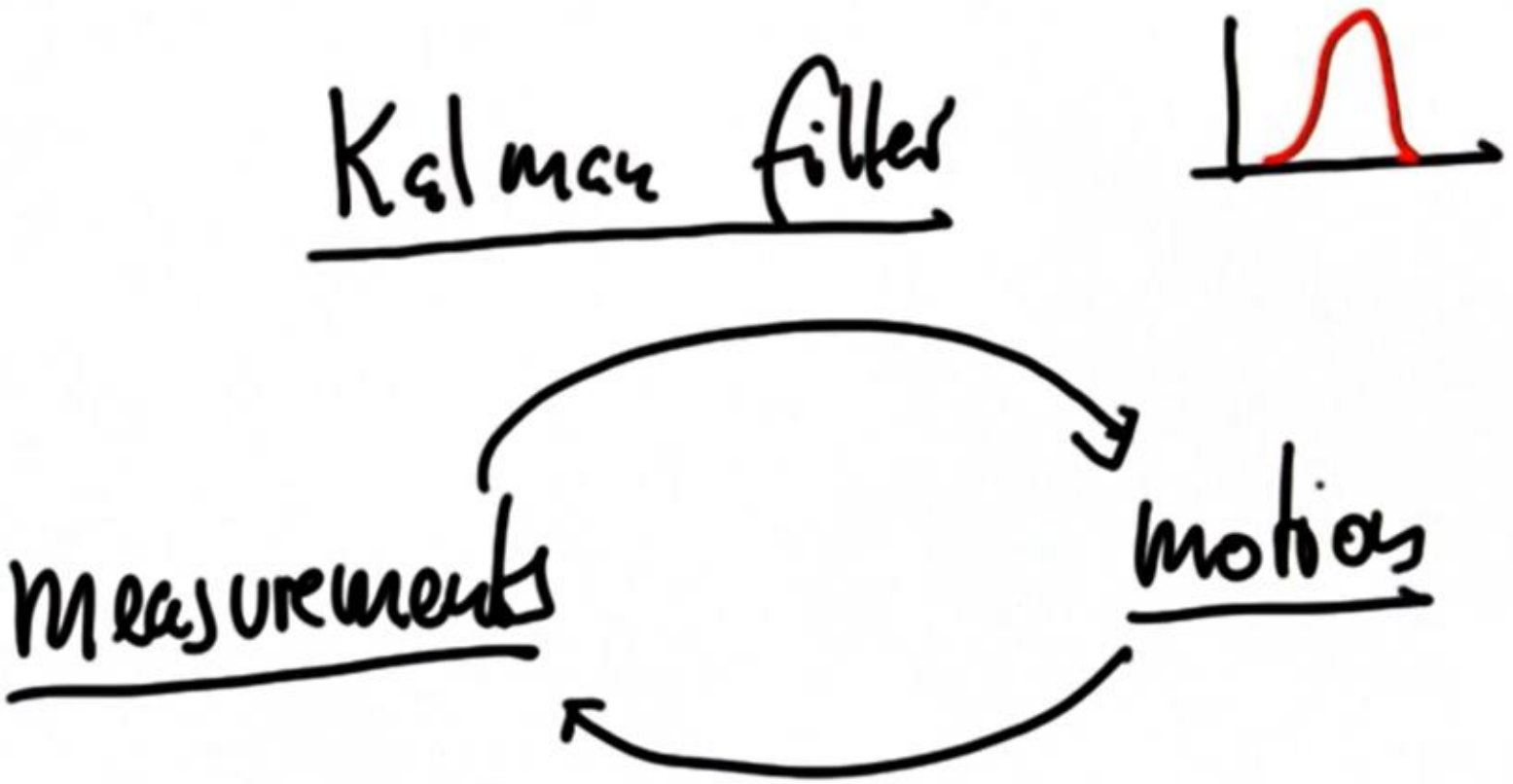
$f(x) =$ 

Evaluate Gaussian

```
#Instead, please just change the last argument
#in f() to maximize the output.

from math import *

def f(mu, sigma2, x):
    return 1/sqrt(2.*pi*sigma2) * exp(-.5*(x-mu)**2 / sigma2)

print(f(10.,4.,8.)) #Change the 8. to something else!
```

0.1209853622595716 8

Measurement and Motion 1

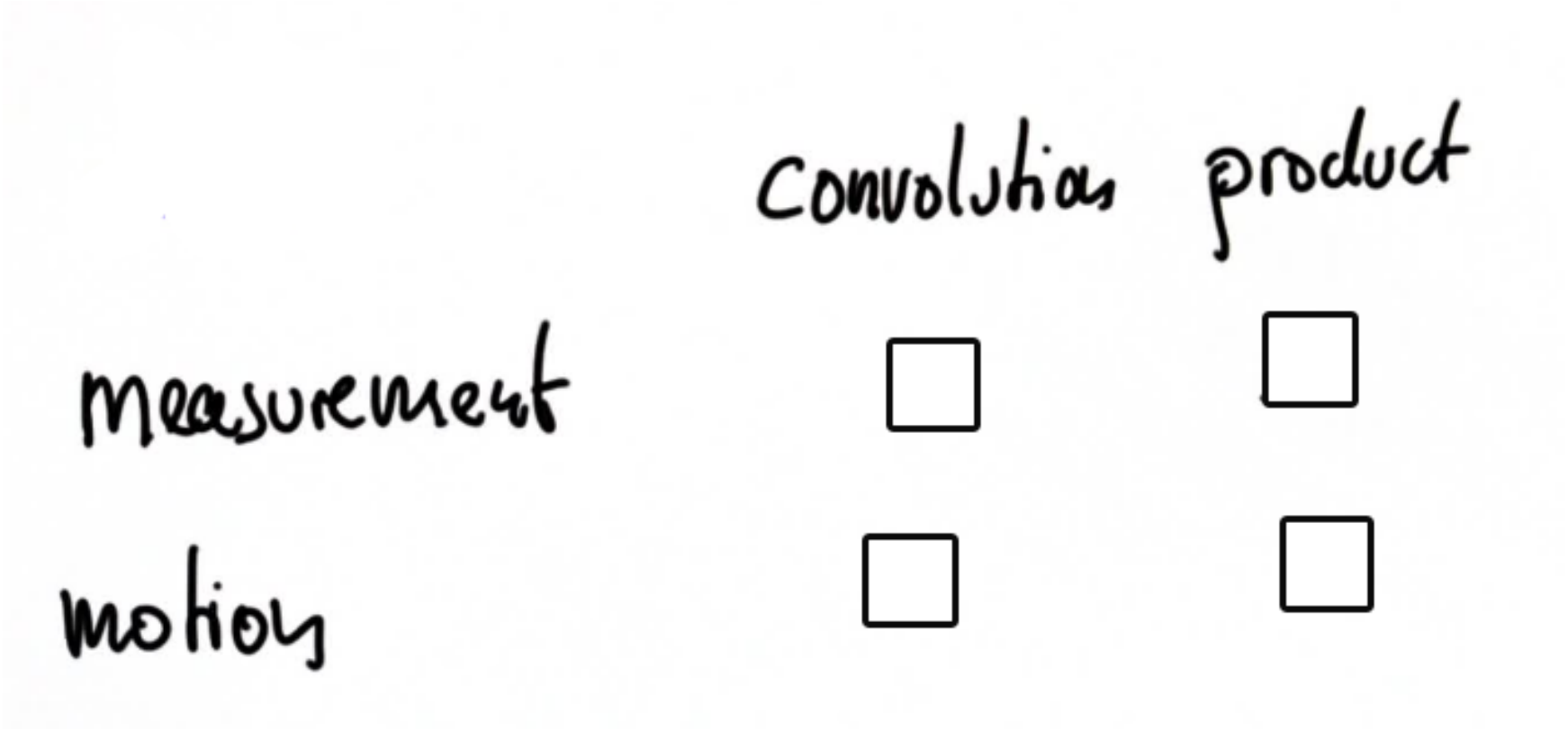


칼만 필터는 측정 업데이트와 동작 업데이트를 주기적으로 반복한다.

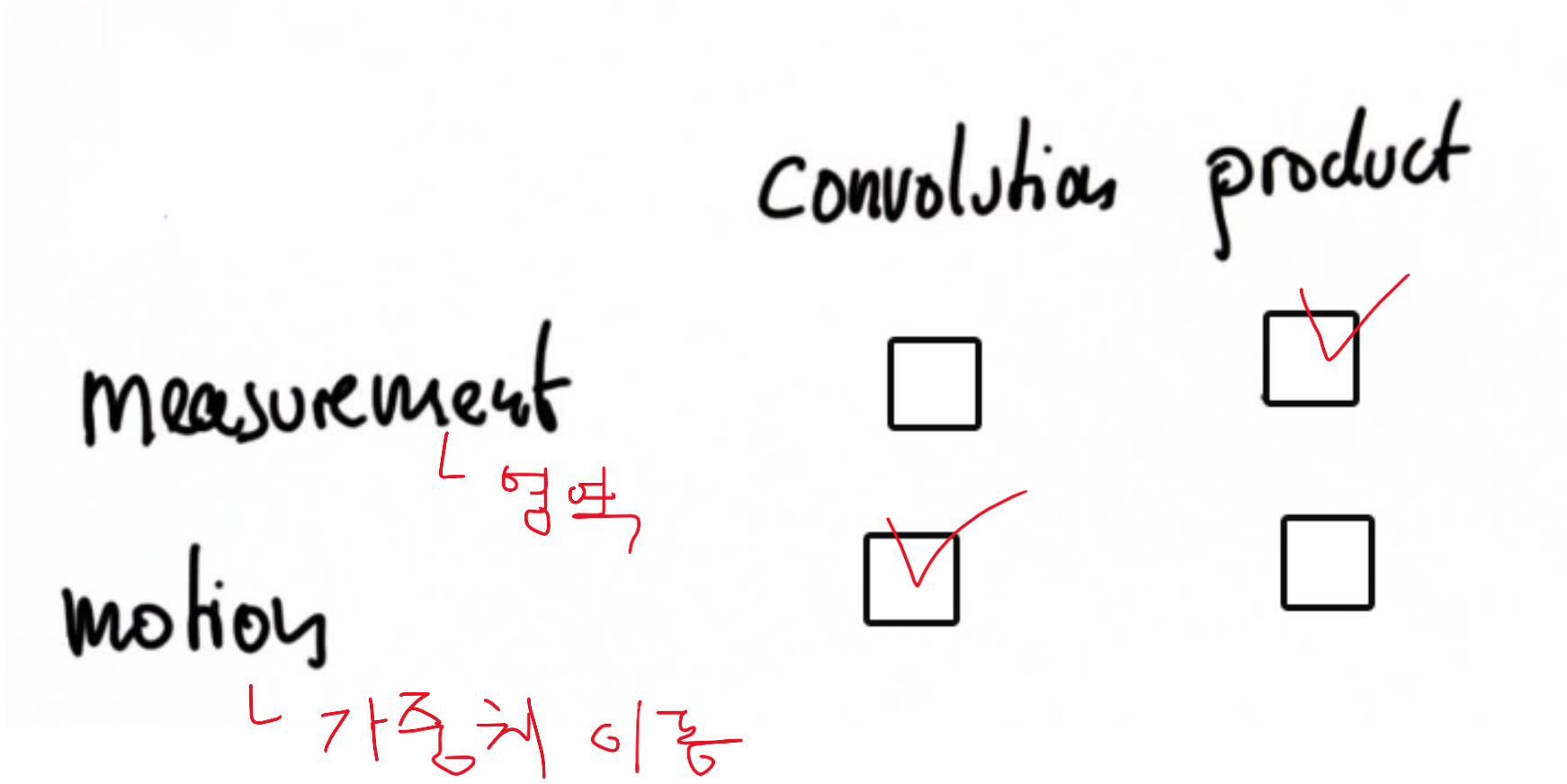
측정 후 동작을 취한 로컬라이제이션 이전 상태

Measurement and Motion 1

각 상태별 해당하는 부분은?

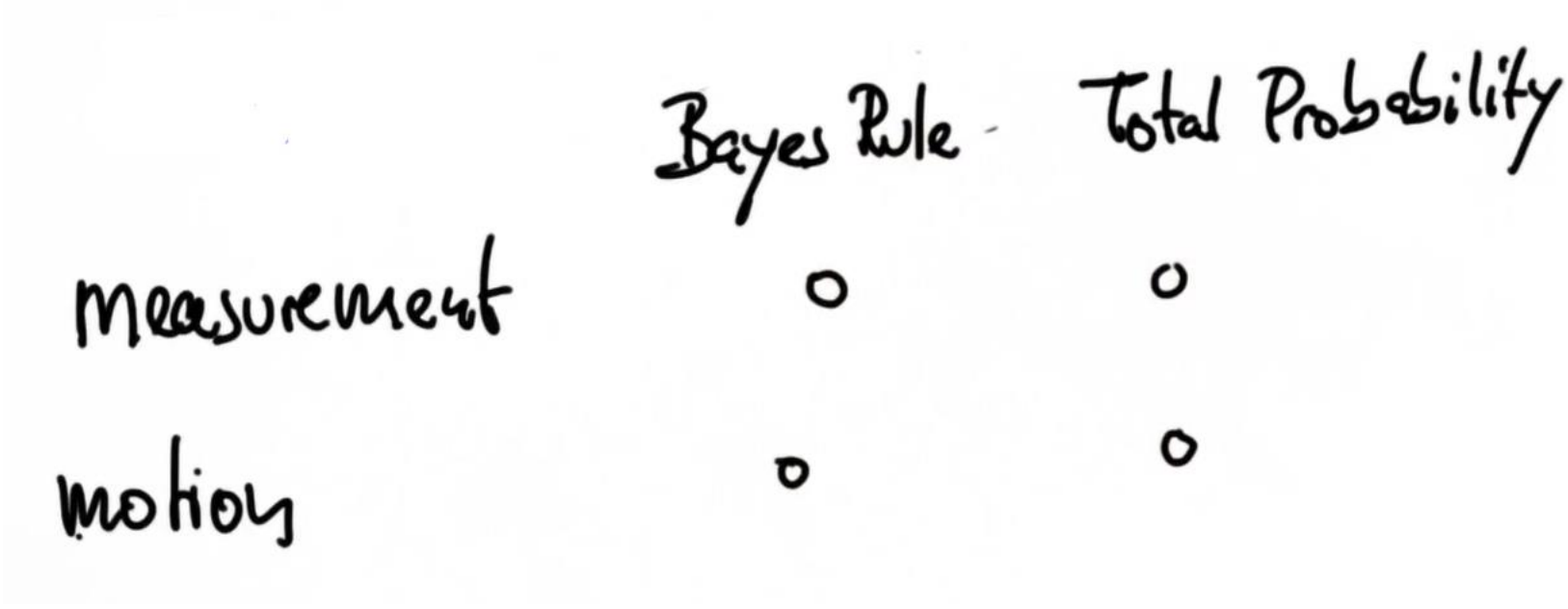


Measurement and Motion 1

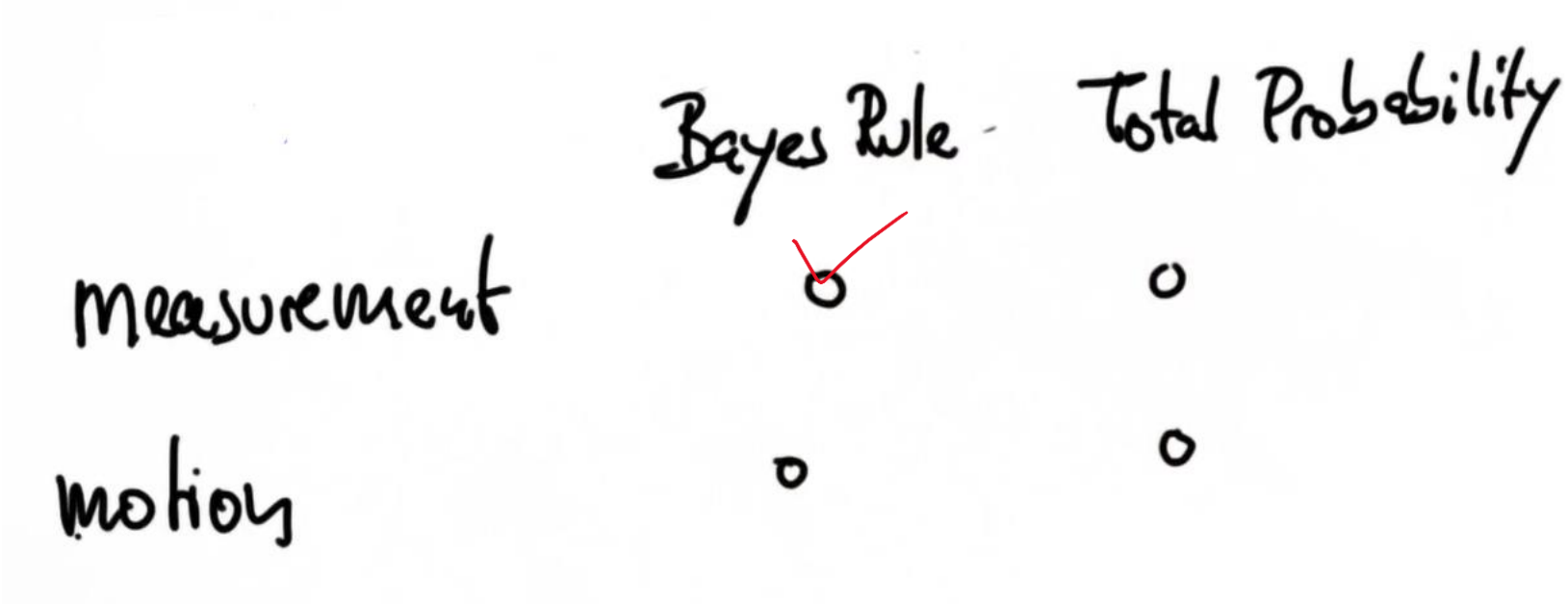


Measurement and Motion 2

베이즈 룰과 , 전체 확률이 측정과 모션과는 어떤 관계가 있을까?

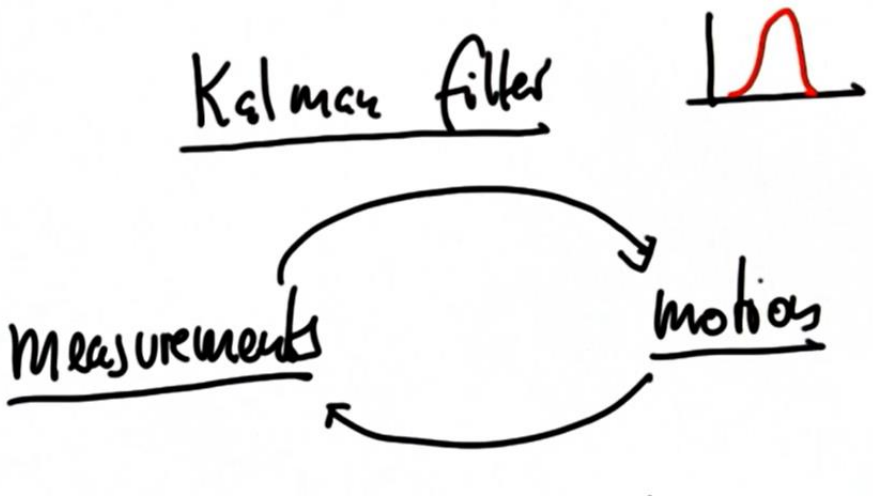


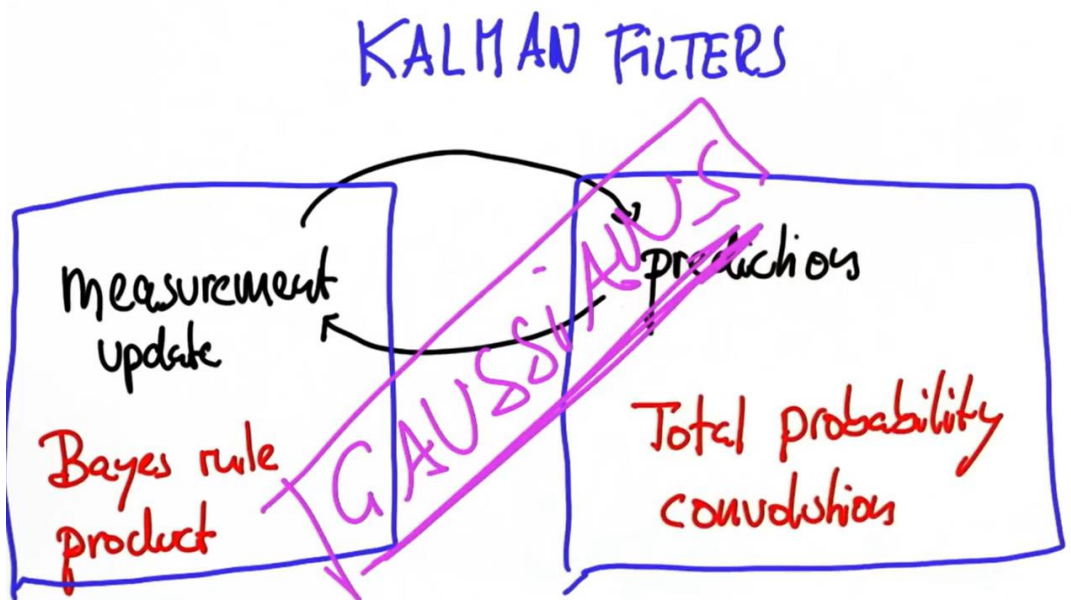
Measurement and Motion 2



측정값인 제품 → 베이즈룰 사용
동작 → 전체 확률을 사용

Shifting the Mean





칼만 필터에선,

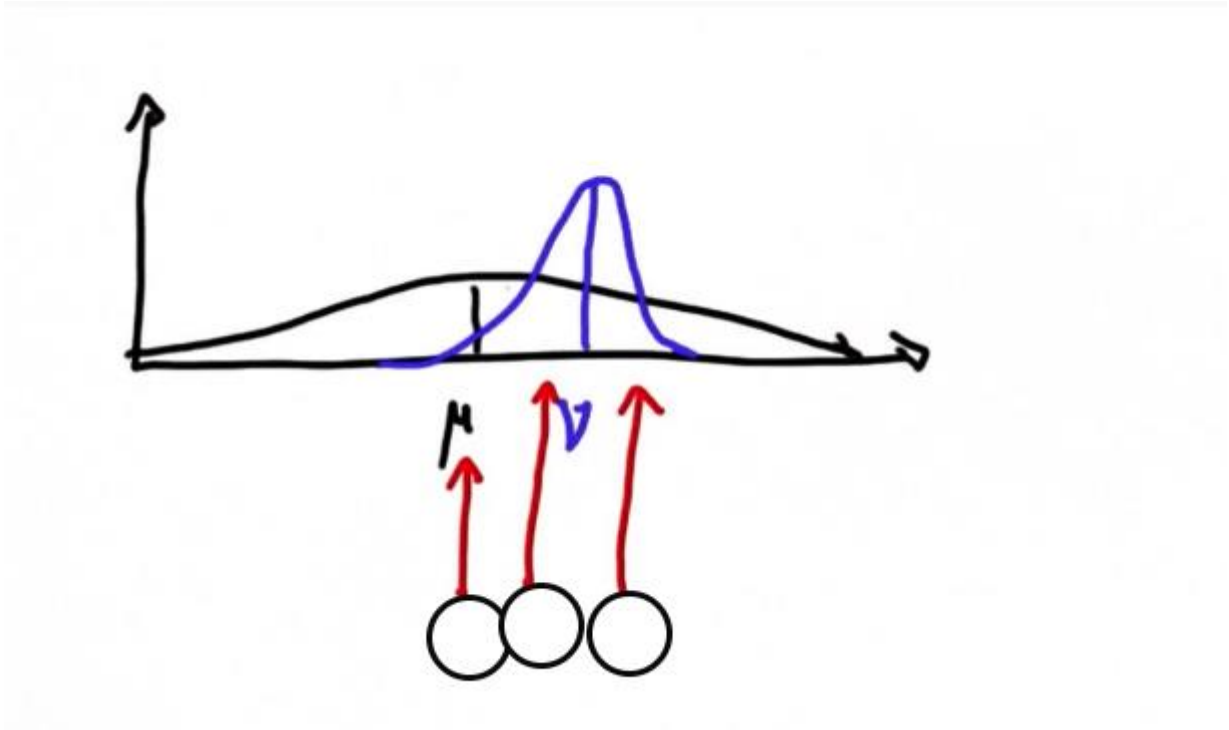
측정 동작을 반복

측정→ 업데이트 동작→ 예측으로 정의 가능

베이즈룰, 프로덕트 사용 전체확률과 합성곱을 사용

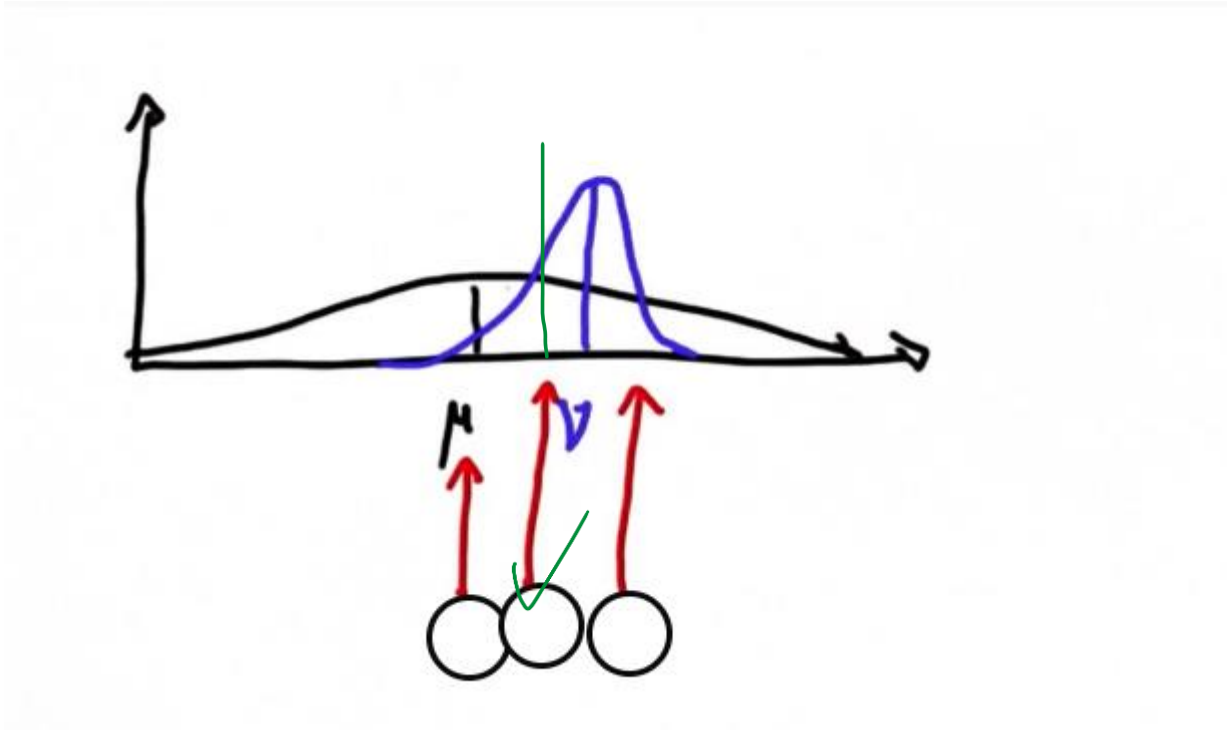
가우시안 활용

Shifting the Mean



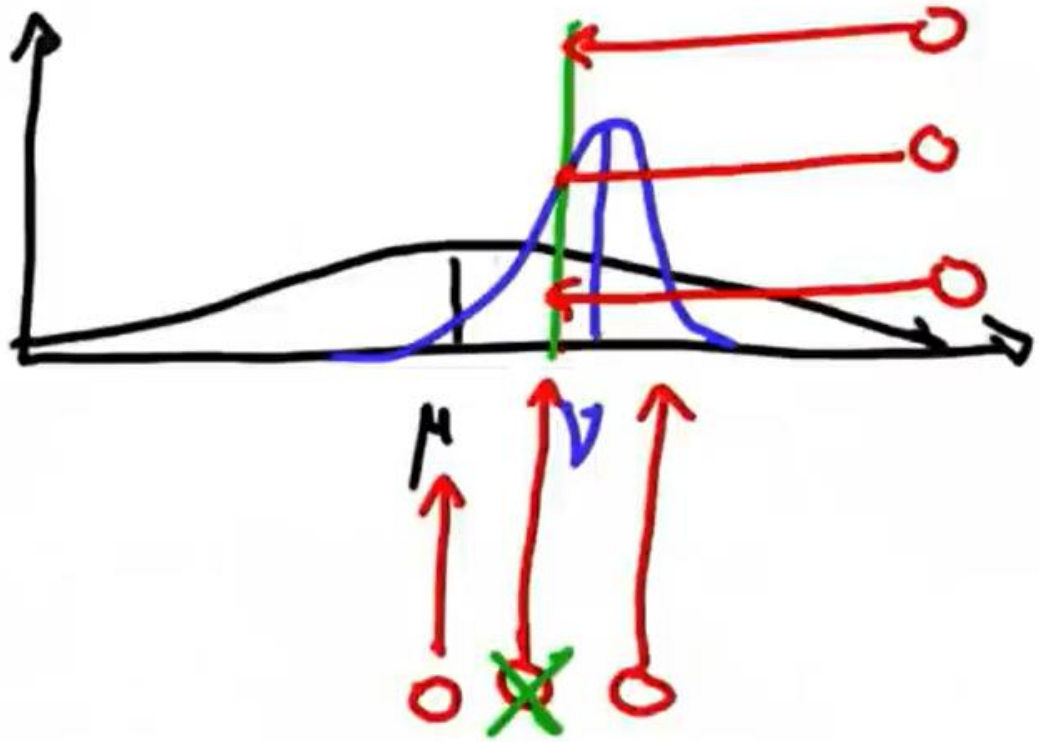
검정: 사전 분포 가우스
파랑: 차량의 로컬라이제이션 가우스
빨강: 후속 가우스의 평균의 위치는?

Shifting the Mean



- 사전 평균과 측정 평균의 가운데 있다.
- 특히 차량이 어디 있는지, 측정이 확실하기 때문에 측정쪽으로 치우치게 된다.
- 확실할수록 평균을 답의 방향으로 끌어당기는 특징이 있다

Predicting the Peak

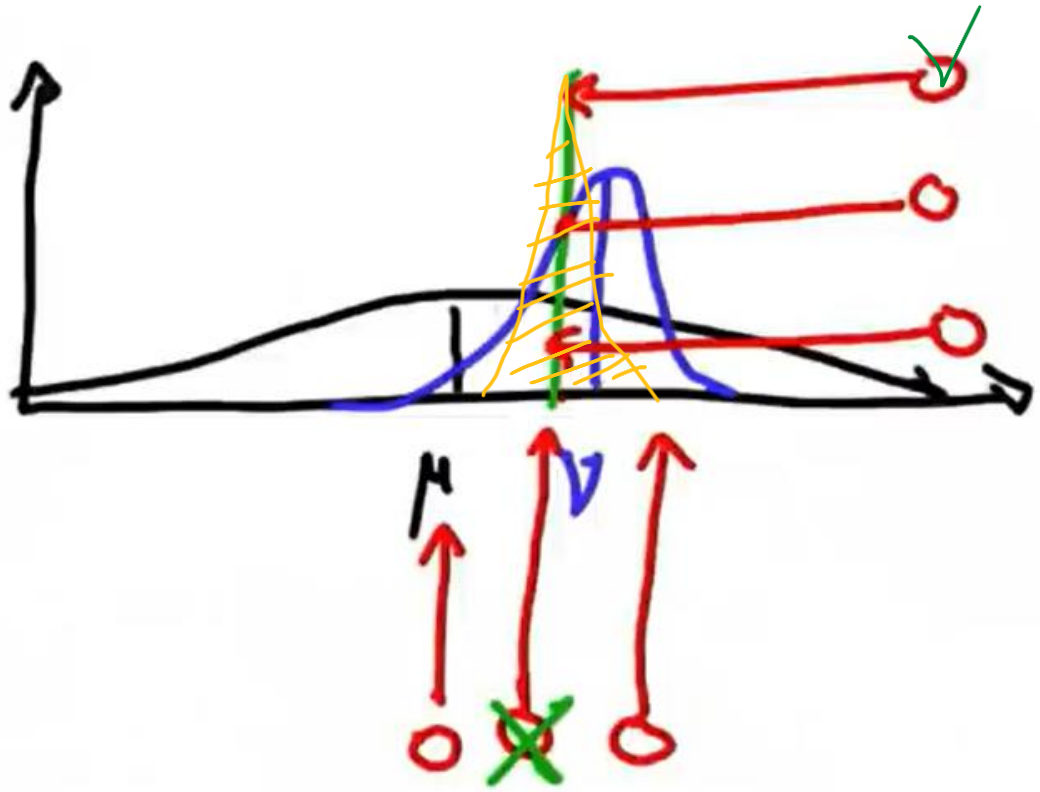


새로운 가우시안을 그리면 매우 뾰족한 형태로 그릴 수 있다. 새 가우스의 피크를 측정

- 매우좁고 가느다란 가우스
- 폭이 두 가우스 사이가 되는 가우스
- 두개보다 넓은 가우스

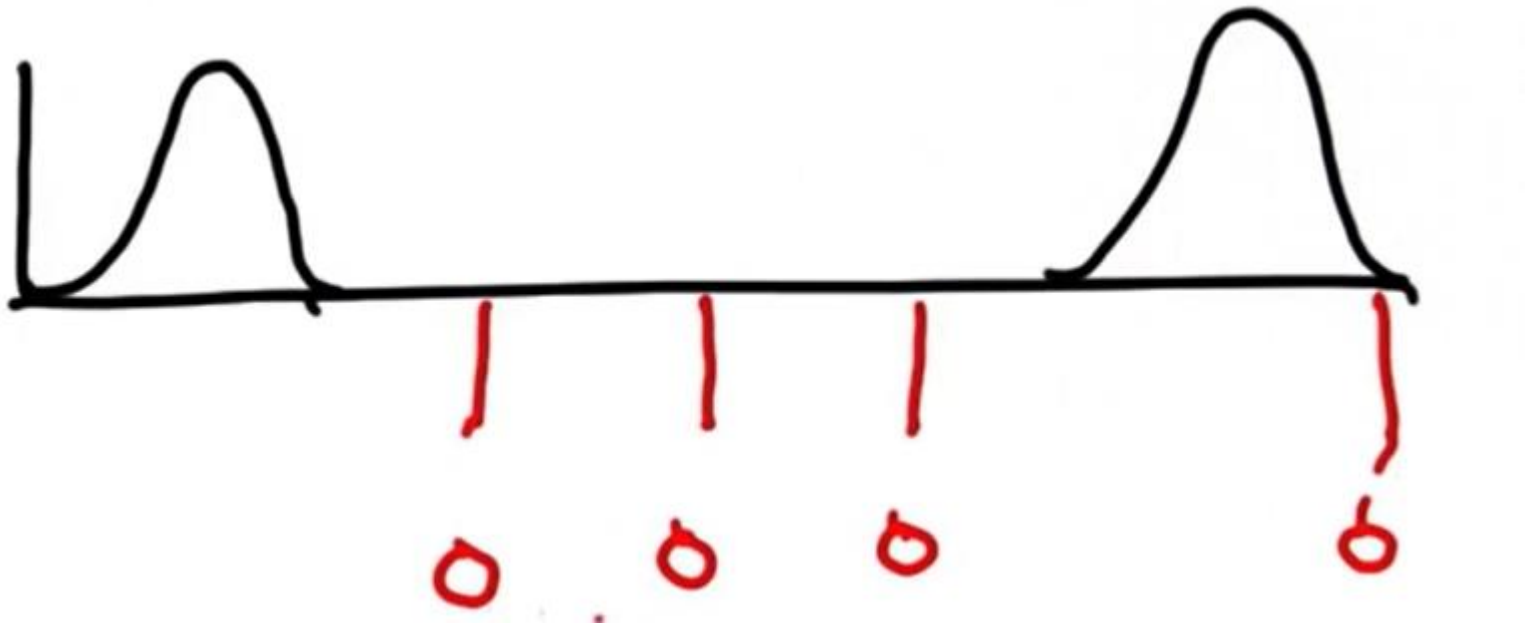
3개중 어떤것이 정확한 Posterior 가 될수 있는가?

Predicting the Peak



1번 가우스가 정확한 정보를 전달한다.
공분산이 가장 작다.

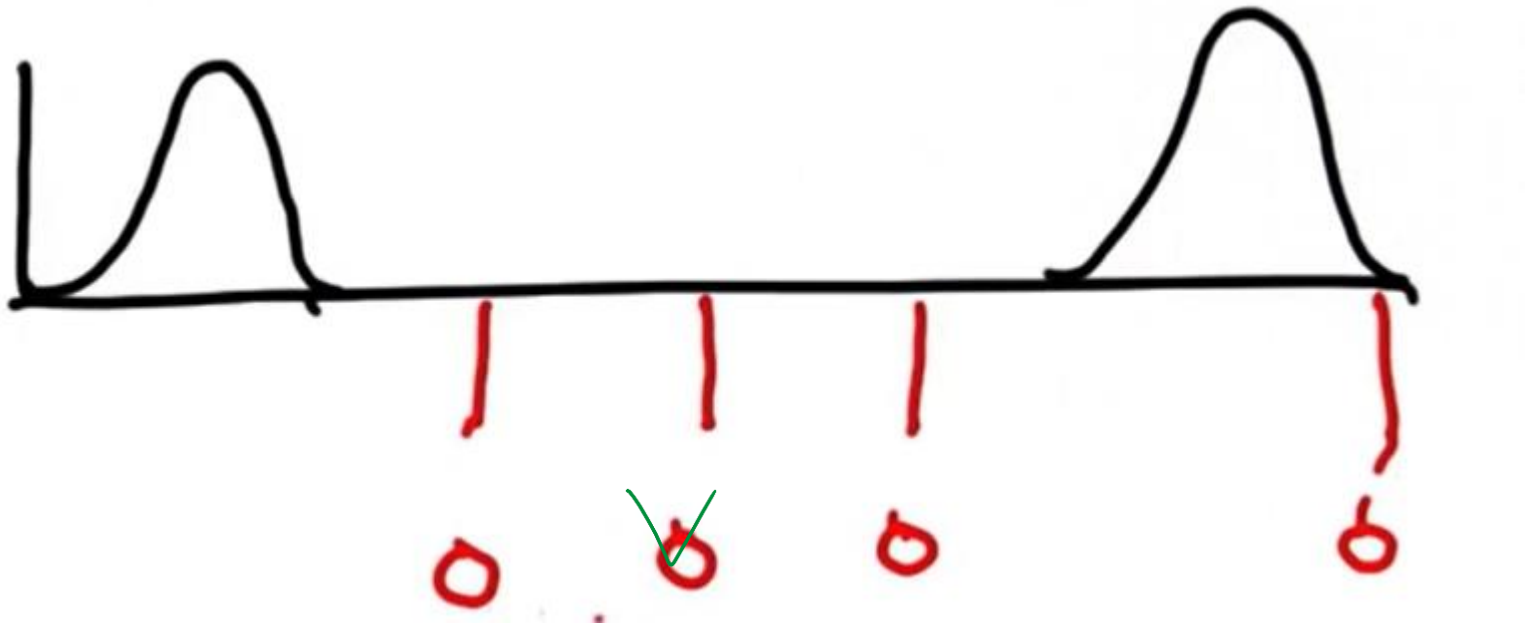
Separated Gaussians



둘 모두 동일한 가우시안을 가진다 가정하자

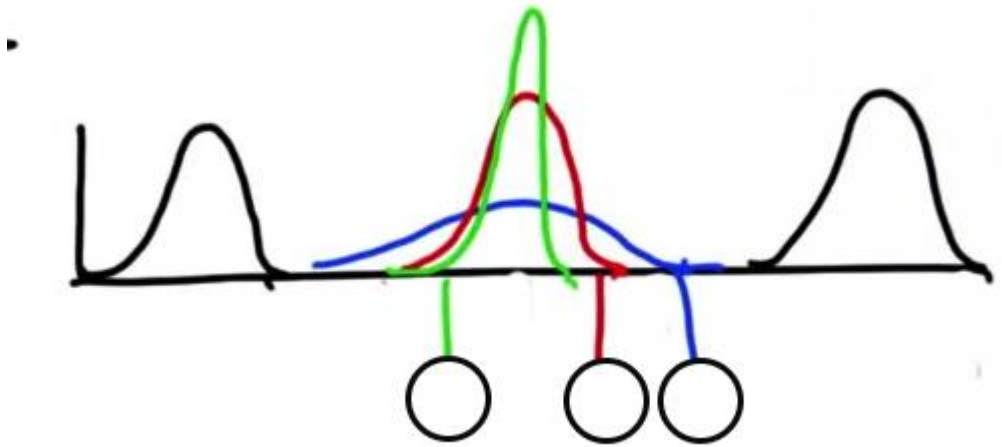
새 가우시안 평균의 위치는 ?

Separated Gaussians



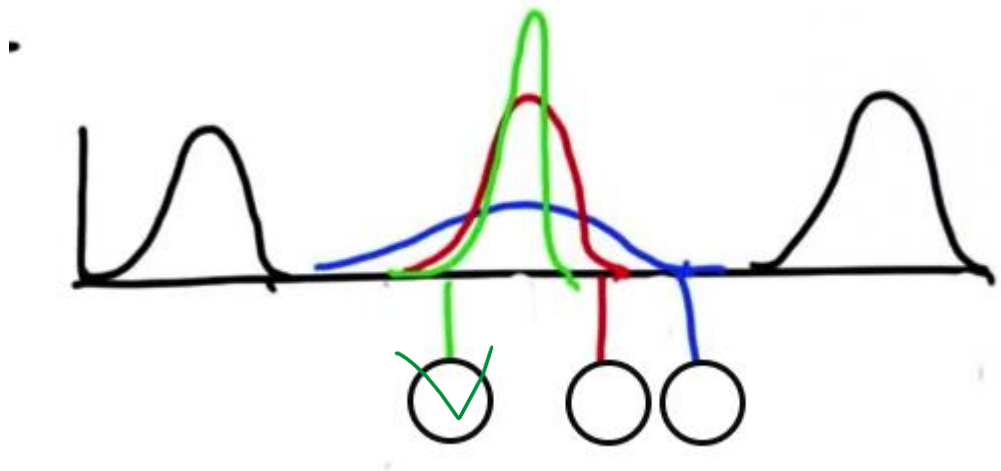
두 분산이 동일하기 때문에 정확히 중간에 존재

Separated Gaussians



분리된 가우스 사이 분산이 만들어 질때 어떤 가우스를 가지게 될까요?

Separated Gaussians



새 가우스는 분산의 제곱이 아닌 절반으로 형성이 된다.
고로 좁은 가우스를 만든다.

Separated Gaussians

평균과 분산을 업데이트하는 프로그램을 작성하세요

```
def update(mean1, var1, mean2, var2):
    new_mean =
    new_var =
    return [new_mean, new_var]

print update(10.,8.,13., 2.)
```

Separated Gaussians

```
def update(mean1, var1, mean2, var2):
    new_mean = ((var2 * mean1) + (var1 * mean2)) / (var2 + var1)
    new_var = 1/((1/var2) + (1/var1))
    return [new_mean, new_var]

print (update(10.,8.,13., 2.))
```

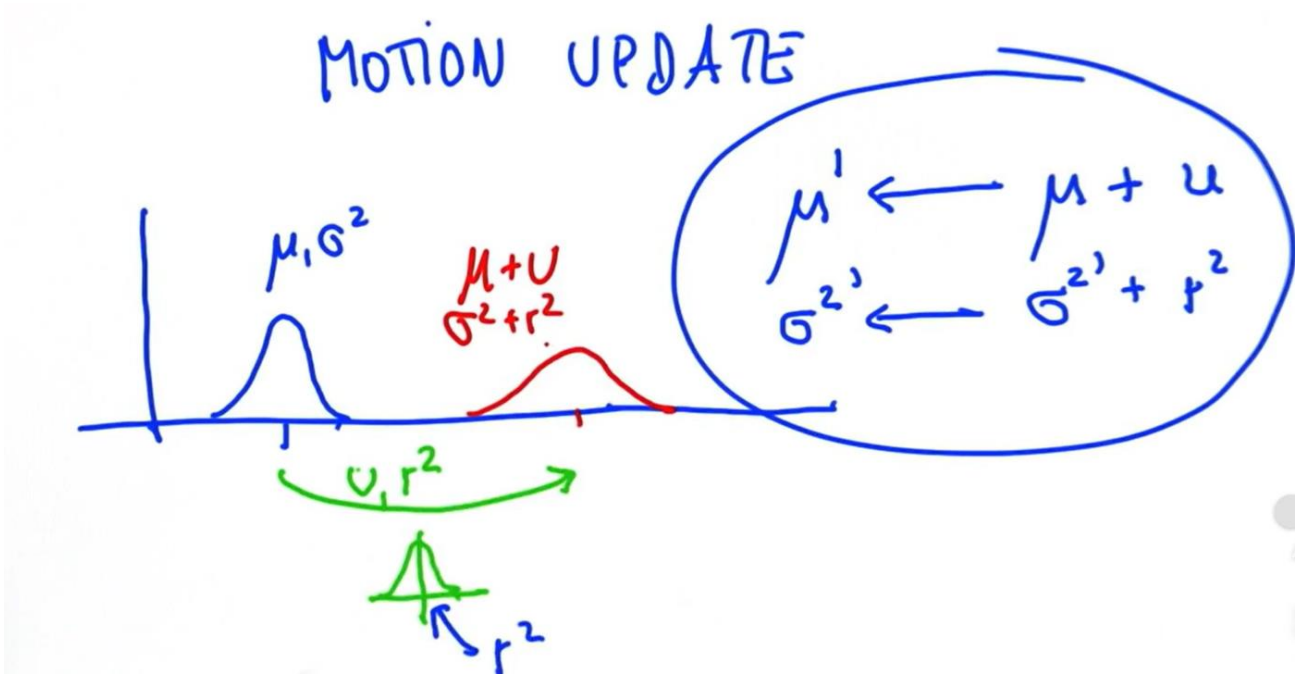
```
[12.4, 1.6]
```

Gaussian Motion

KALMAN FILTER

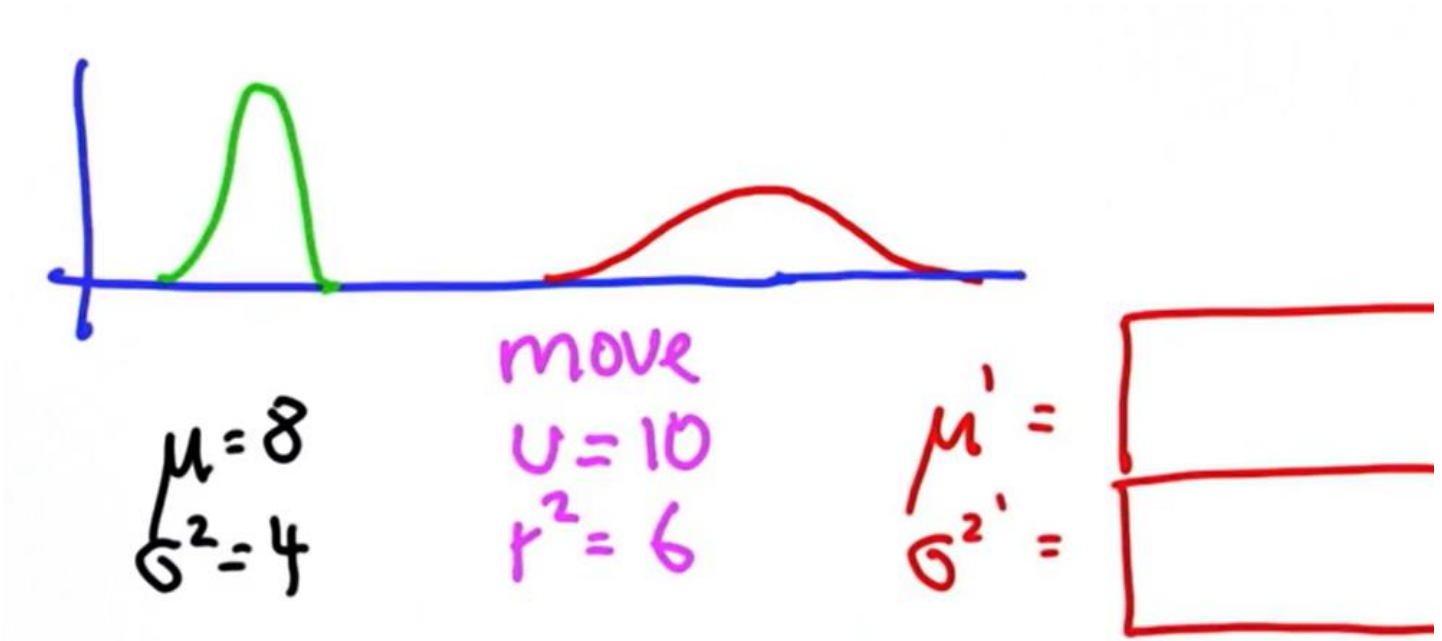
MEASUREMENT UPDATE
BAYES RULE
MULTIPLICATION

MOTION UPDATE
PREDICTION
TOTAL PROBABILITY
ADDITION



왼쪽 지점에서 오른쪽으로 이동, 동작은 불확실성을 갖는다.
명령을 더한 예측으로 불확실성이 증가한 지점(빨강) 을 갖는다←동작은 정보를 상실.
새평균 = 이전평균 + u(이동)
시그마제곱 = 이전 시그마제곱+가우스분산(r)
➔ 새로 만들어진 불확실성 가우스

Gaussian Motion

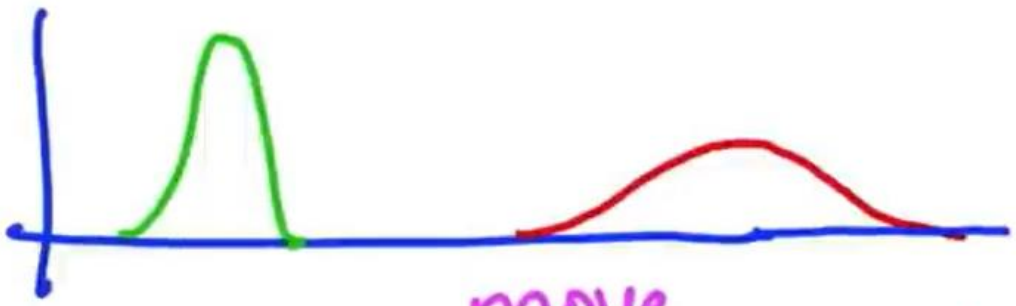


예측 단계전 가우스 값

6의 동작불확실성으로 오른쪽으로 10만큼 이동

이때 예측 단계 가우스의 뮤와 시그마 값은?

Gaussian Motion



move

$\mu = 8$

$\sigma^2 = 4$

$U = 10$

$r^2 = 6$

$\mu' =$ 18

$\sigma^{2'} =$ 10

Predict Function

새로운 평균을 예측하는 프로그램을 작성하세요

```
def update(mean1, var1, mean2, var2):
    new_mean = (var2 * mean1 + var1 * mean2) / (var1 + var2)
    new_var = 1/(1/var1 + 1/var2)
    return [new_mean, new_var]

def predict(mean1, var1, mean2, var2):
    new_mean =
    new_var =
    return [new_mean, new_var]

print predict(10., 4., 12., 4.)
```

Predict Function

```
def update(mean1, var1, mean2, var2):
    new_mean = ((var2 * mean1) + (var1 * mean2)) / (var2 + var1)
    new_var = 1/((1/var2) + (1/var1))
    return [new_mean, new_var]

print (update(10.,8.,13., 2.))
```

```
[22.0, 8.0]
```

Kalman Filter

반복적으로 업데이트하고 업데이트하는 프로그램을 작성합니다.
위치 측정을 기반으로 예측하는 프로그램 구성

```
def update(mean1, var1, mean2, var2):
    new_mean = float(var2 * mean1 + var1 * mean2) / (var1 + var2)
    new_var = 1./(1./var1 + 1./var2)
    return [new_mean, new_var]

def predict(mean1, var1, mean2, var2):
    new_mean = mean1 + mean2
    new_var = var1 + var2
    return [new_mean, new_var]

measurements = [5., 6., 7., 9., 10.]
motion = [1., 1., 2., 1., 1.]
measurement_sig = 4.
motion_sig = 2.
mu = 0.
sig = 10000.

#Please print out ONLY the final values of the mean
#and the variance in a list [mu, sig].

# Insert code here

print [mu, sig]
```

Kalman Filter

```
def update(mean1, var1, mean2, var2):
    new_mean = float(var2 * mean1 + var1 * mean2) / (var1 + var2)
    new_var = 1./(1./var1 + 1./var2)
    return [new_mean, new_var]

def predict(mean1, var1, mean2, var2):
    new_mean = mean1 + mean2
    new_var = var1 + var2
    return [new_mean, new_var]

measurements = [5., 6., 7., 9., 10.]  # means of measurement
motion = [1., 1., 2., 1., 1.]  # means of motion
measurement_sig = 4.  # uncertainties of both measurement(update) and
motion(predict) remain same
motion_sig = 2.
mu = 0.
sig = 10000.
```

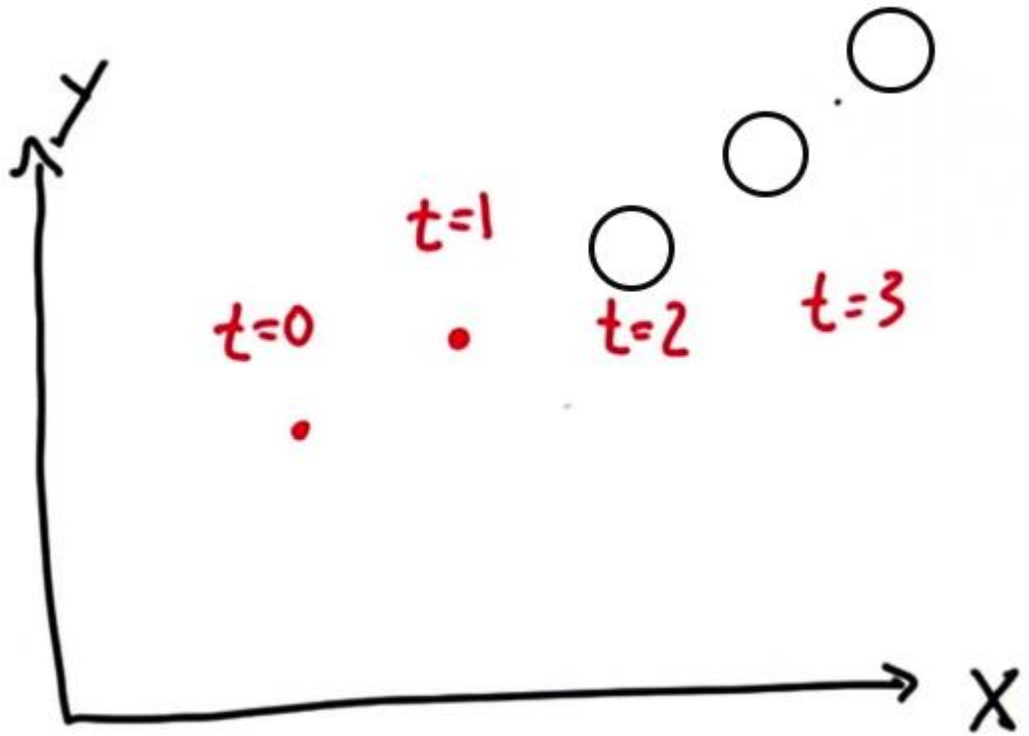
Kalman Filter

```
#Please print out ONLY the final values of the mean
#and the variance in a list [mu, sig].

# Insert code here
for k in range(len(measurements)):
    mu, sig = update(mu, sig, measurements[k], measurement_sig)
    mu, sig = predict(mu, sig, motion[k], motion_sig)


print ([mu, sig]) # mean and variance of the posterior belief (next position and its variance)
```

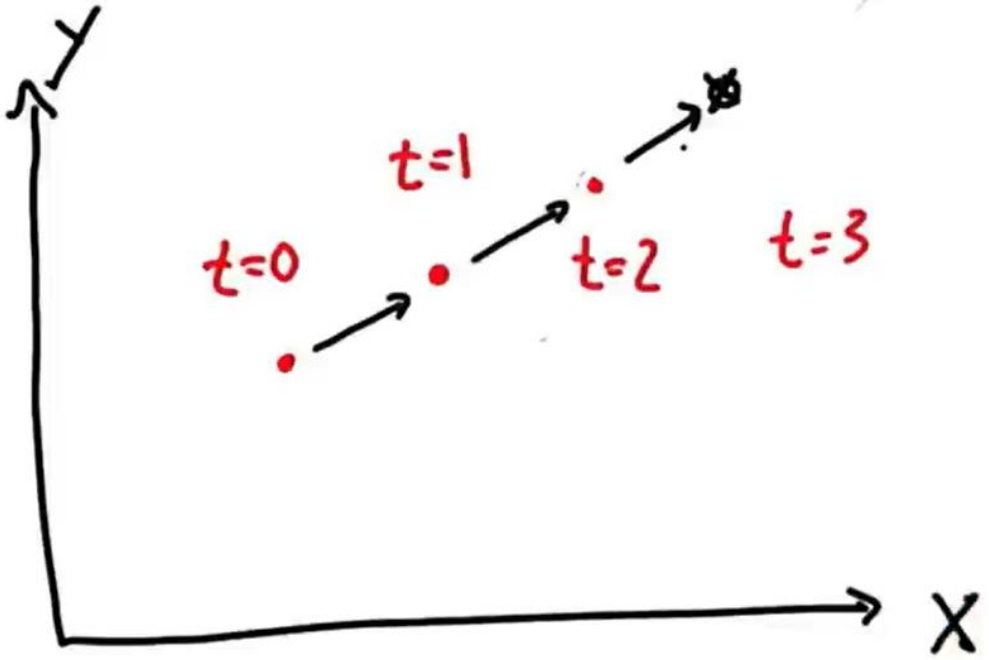
```
[10.999906177177365, 4.005861580844194]
```

Kalman Prediction

2차원 칼만 필터 구조에서, 자율 주행 차가 각 시간 단계별 진행시 보이는 t3 단계의 위치는 어디가 될까요?



Kalman Prediction



칼만 필터 물체의 속도가 얼마나 빠른지 암묵적으로 파악한 다음, 속도 추정치를 사용, 미래에 대한 정확한 예측을 가능하게 한다.
센서는 위치만 보고 움직이기에, 속도를 판정하지 않는다.
칼만 필터는 속도를 포함한 미래 위치에 대한 예측을 할수 있다는것