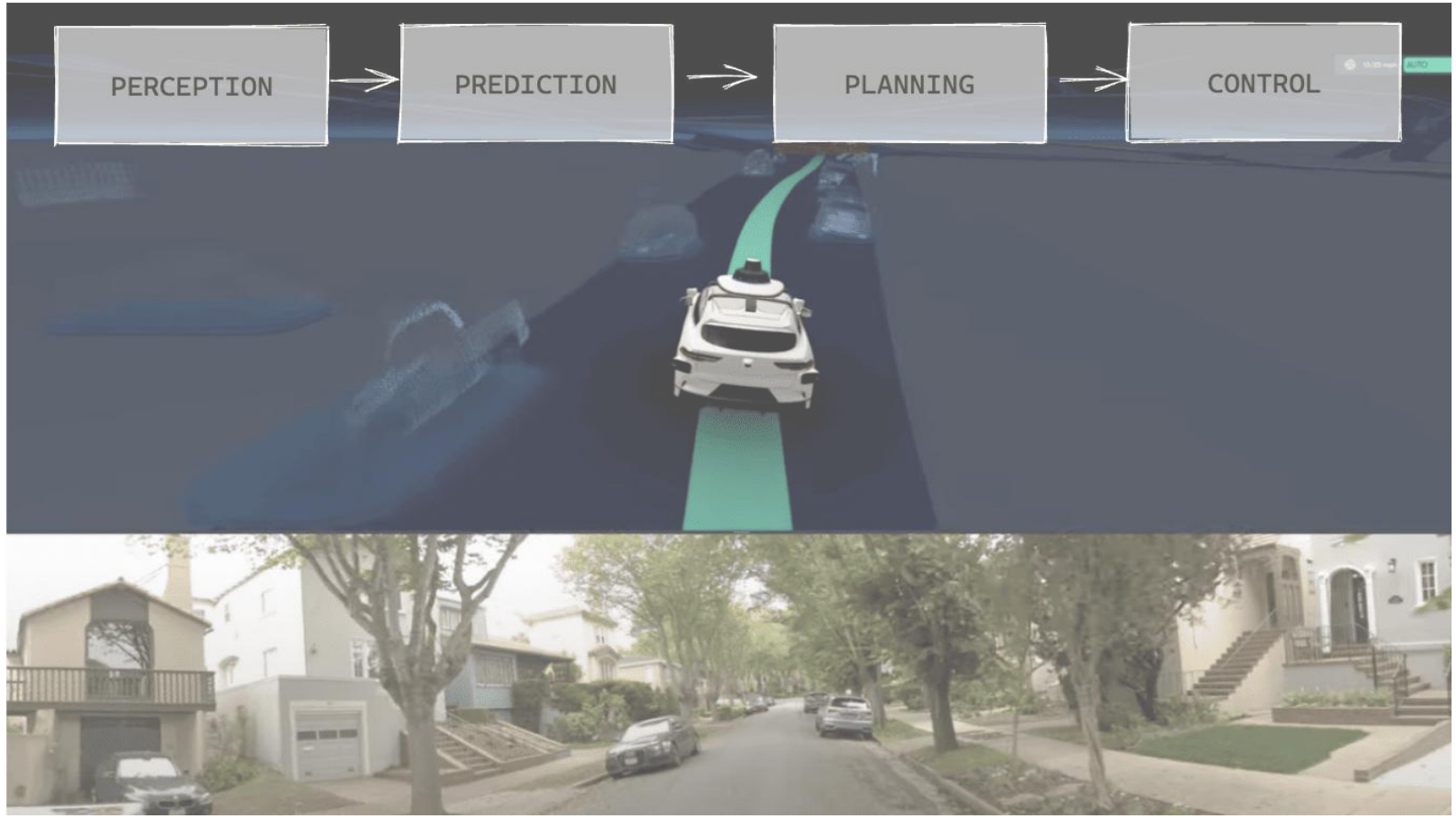
Advanced Deep Learning

week 4.1

Dongchan Kim

AiMIND Lab

Hanyang University Erica



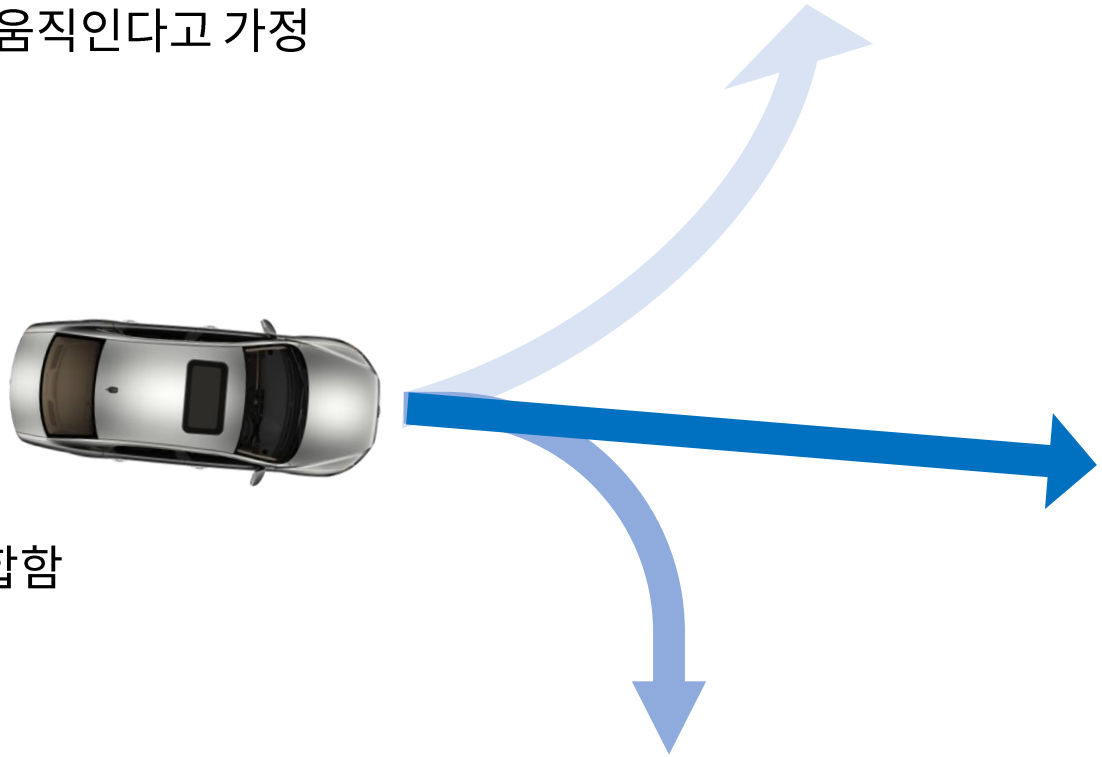
Rule-based Prediction Methods

Physics-based method (직진모델: CV, CA model)

Overview

- Object의 움직임은 물리 법칙을 따름
  - 물리 법칙을 표현하는 모델을 통해 미래의 움직임을 예측
  - 미래 움직임의 변화를 알 수 없기 때문에 특정 모델과 같이 움직인다고 가정
  - 현재 상태를 바탕으로 미래의 움직임을 예측

- 특징
  - 주변 Object나 Map에 대한 고려가 없음
  - 고려하는 변수가 적기 때문에 연산 시간이 짧음
  - Object의 상태만을 이용하기 때문에 긴 시간 예측은 부적합함



Overview

- Input
  - current dynamic object state (position, velocity, heading)
  - prediction horizon (T)
  - time step between predictions (dt)
- Output
  - list of predicted future vehicle states

$$
\begin{aligned}
&t \leftarrow 0 \\
&x_0 = x_{measure} \\
&while\ (t * dt < T)\ do \\
&\quad t = t + 1 \\
&\quad x_t = model(x_{t-1}) \\
&end\ \ while \\
&return\ \ x_{1:T}
\end{aligned}
$$

CV Model

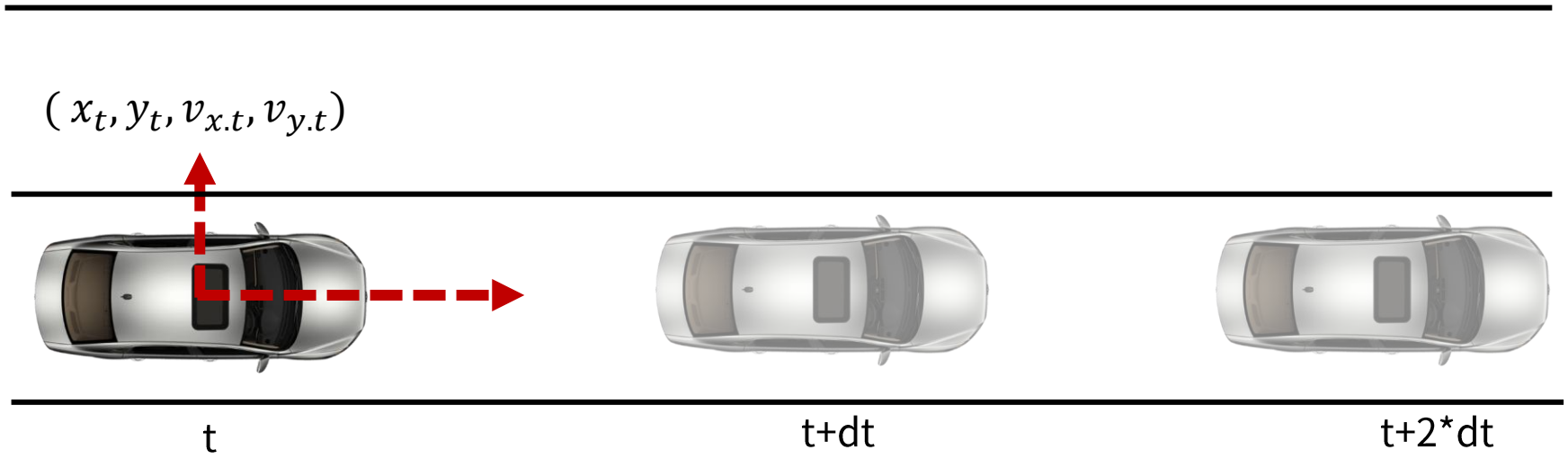
- CV (Constant Velocity) Model
  - 일반적인 직진 주행을 하는 경우에 대한 모델
  - 현재의 속도를 유지한다고 가정
  - 가장 간단한 예측 모델

State–space equation

State : x= $[\, x, y, v_x, v_y \,]^T$

$$\begin{cases} x_{t+1} = x_t + v_{x.t} * dt \\ y_{t+1} = y_t + v_{y.t} * dt \end{cases}$$

$$\begin{cases} v_{x.t+1} = v_{x.t} \\ v_{y.t+1} = v_{y.t} \end{cases}$$



$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ v_{x,t+1} \\ v_{y,t+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \\ v_{x,t} \\ v_{y,t} \end{bmatrix}$$

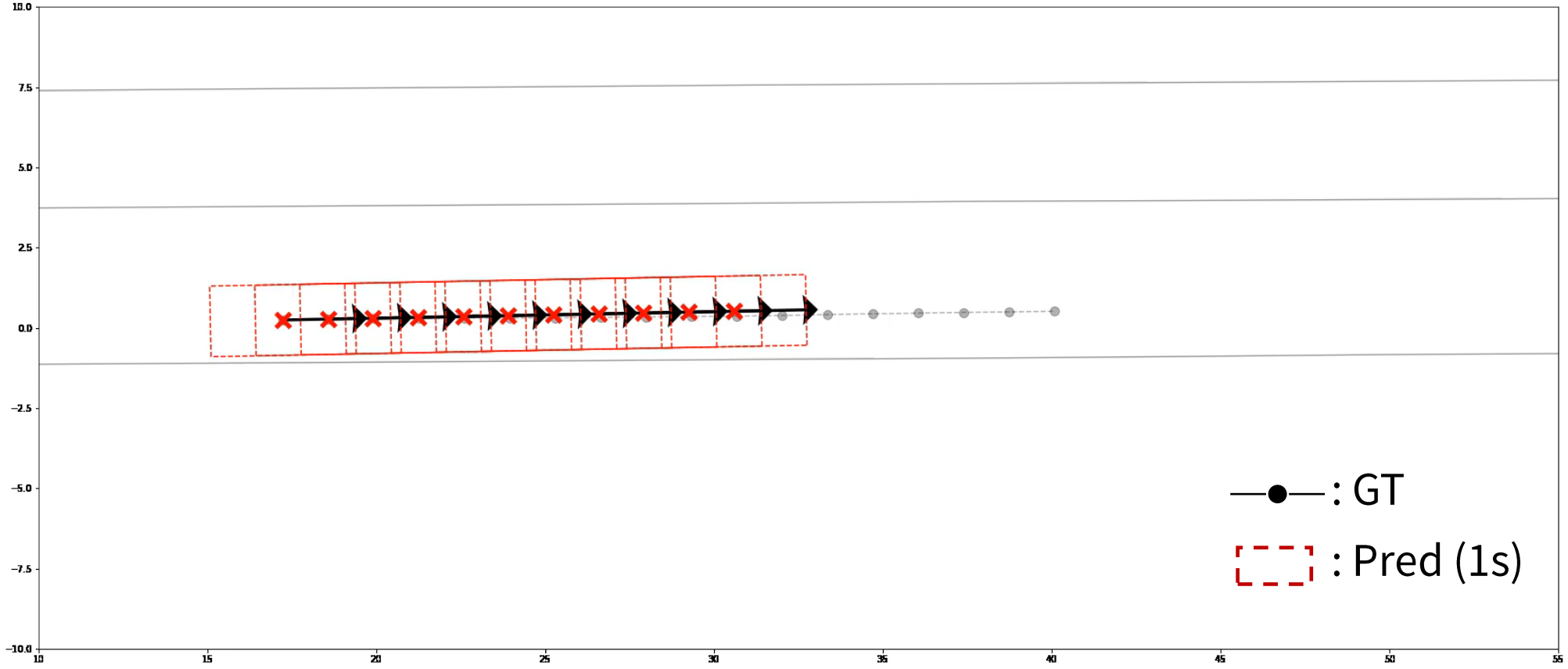
$$X_{t+1} = AX_t$$

$$X_{t+2} = AX_{t+1} = A(AX_t) = A^2 X_t$$

$$X_{t+n} = A(\cdots(AX_t)) = A^n X_t$$

CV Model

- CV (Constant Velocity) Model



CA Model

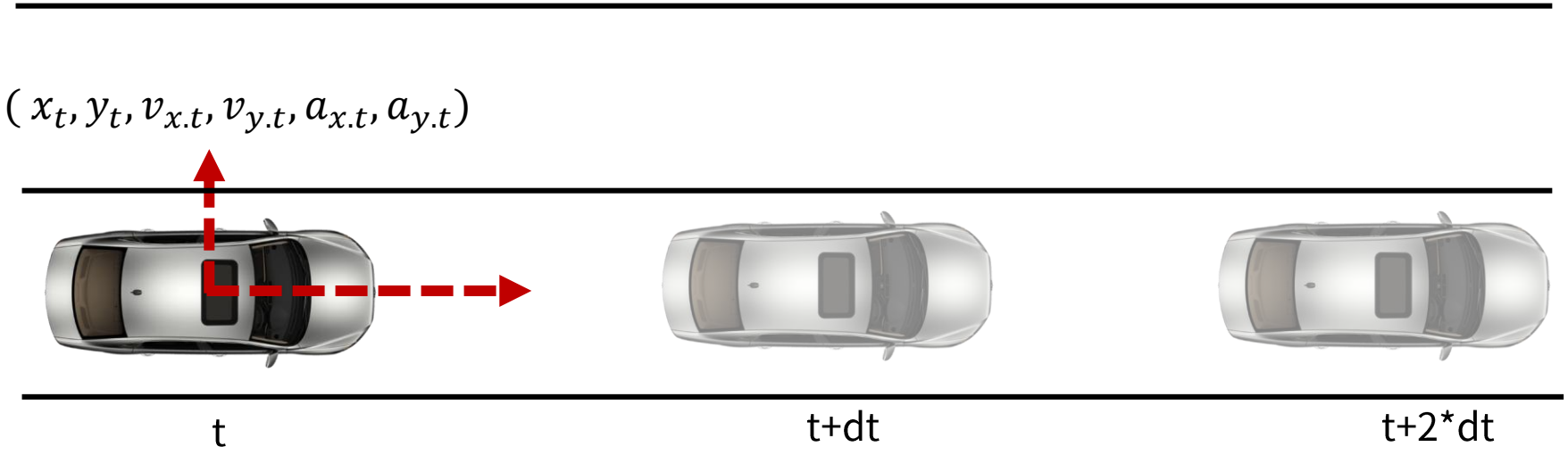
- CA (Constant Acceleration) model
  - 일반적인 주행 + 감 가속 상황
  - 현재의 가속도를 유지한다고 가정

State–space equation

State : X = $[x, v, a]^T$

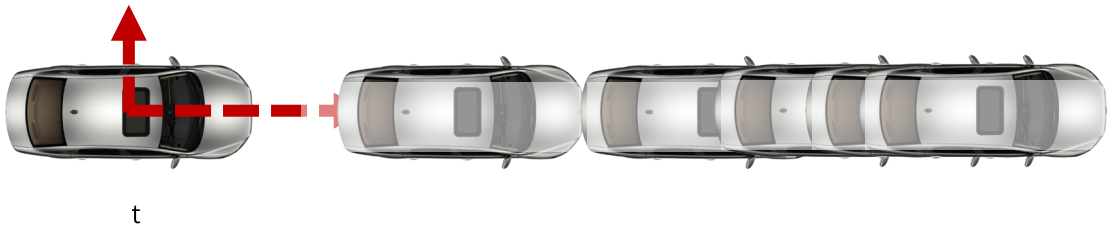
$$\begin{cases} x_{t+1} = x_t + v_t * dt + \frac{1}{2} a_t * dt^2 \\ v_{t+1} = v_t + a_t * dt \\ a_{t+1} = a_t \end{cases}$$

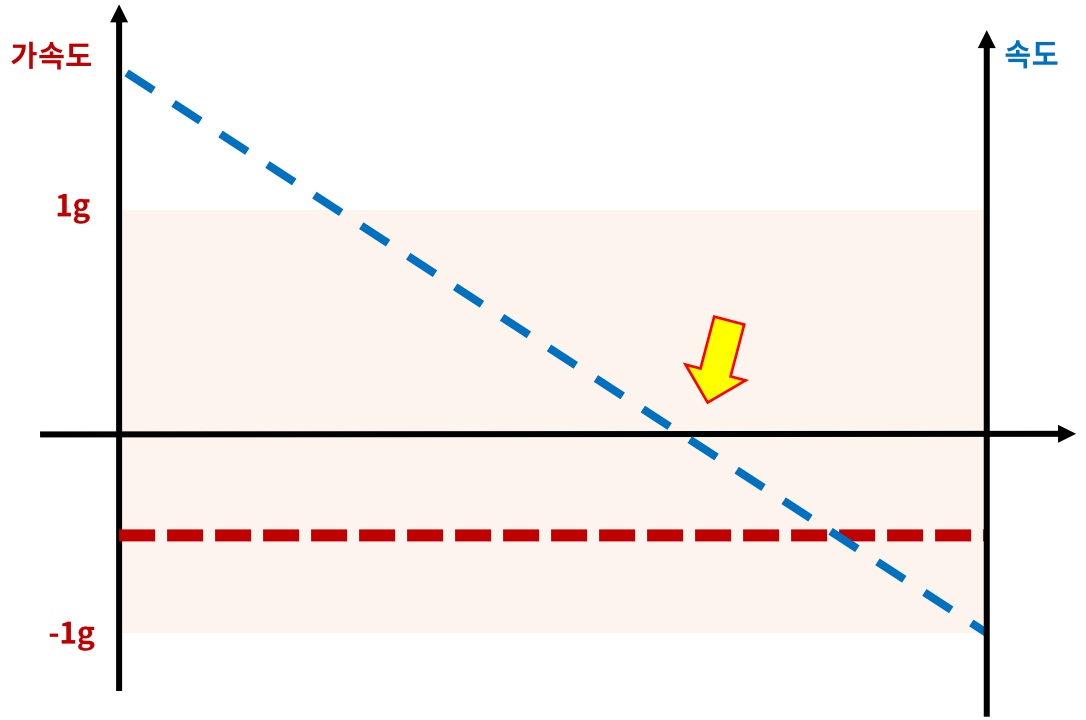
$$\begin{bmatrix} x_{t+1} \\ v_{t+1} \\ a_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & dt & dt^2/2 \\ 0 & 1 & dt \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ v_t \\ a_t \end{bmatrix}$$



CA Model

- 가속도의 한계
  - 차량이 가질 수 있는 감/가속도의 절대값은 $1g(9.8\,m/s^2)$ 이하
- 속도 부호의 변화
  - 전진하던 차량이 후진을?





CA Model

- 가속도의 한계
  - 차량이 가질 수 있는 감/가속도의 절대값은 $1g(9.8\,m/s^2)$ 이하

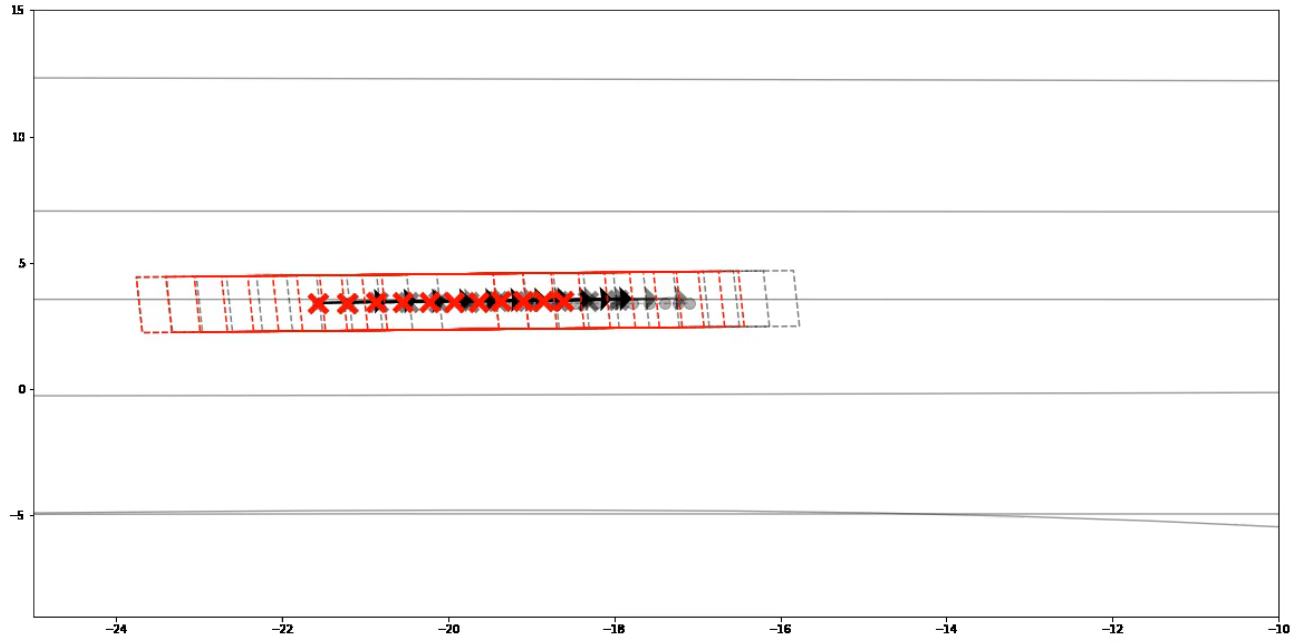
$$
\begin{aligned}
&if\ \ |a| > 9.8 \\
&\quad a = clip(a, [-9.8, 9.8]) \\
&end
\end{aligned}
$$

- 속도 부호의 변화
  - 전진하던 차량이 후진을?

$$
\begin{aligned}
&if\ \ (v + a * dt) < 0 \\
&\quad a = -v / dt \\
&end
\end{aligned}
$$

$$
\begin{aligned}
&t \leftarrow 0 \\
&x_0 = x_{measure} \\
&while\ (t * dt < T)\ do \\
&\quad t = t + 1 \\
&\quad x_t = model(x_{t-1}) \\
&end\ \ while \\
&return\ \ x_{1:T}
\end{aligned}
$$

CA Model



—●— : GT

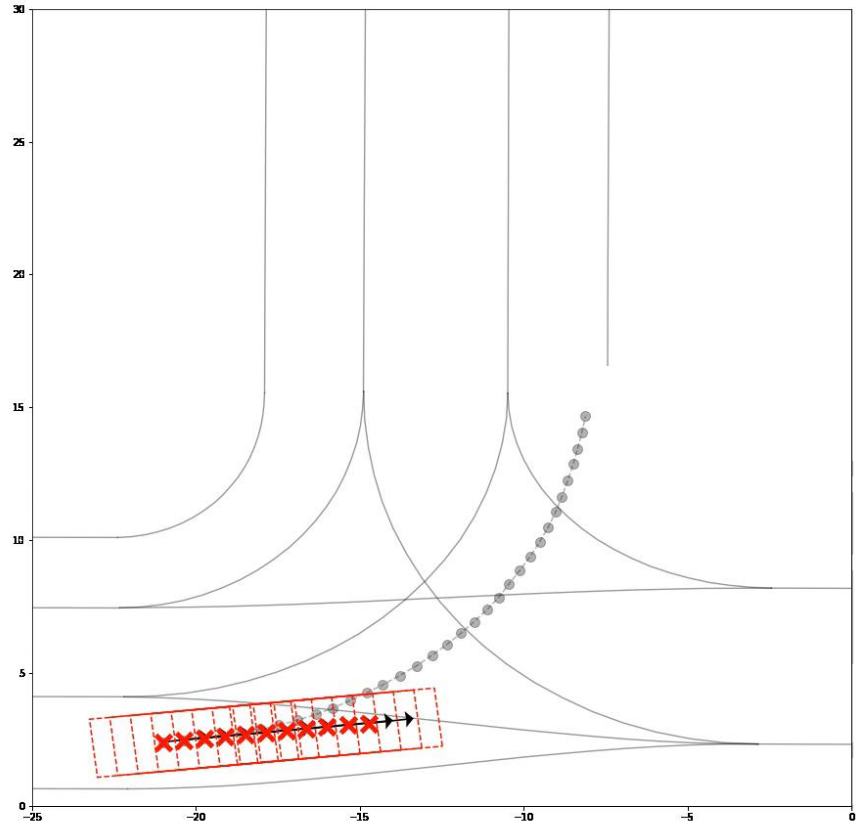
: Pred (CA)

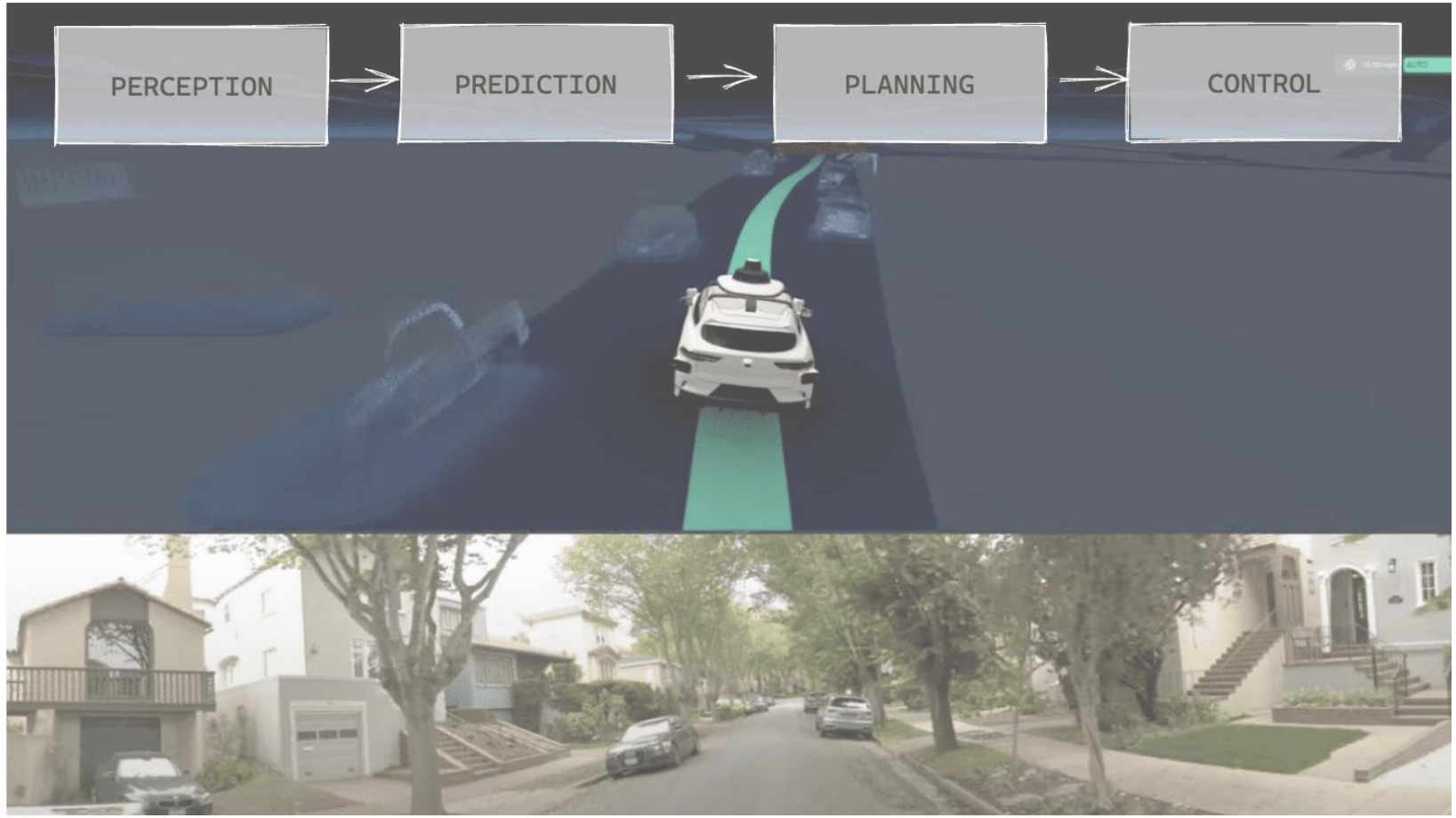
: Pred (CV)

HYU

Limitation

- 차량이 선회하는 움직임을 모사할 수 없음



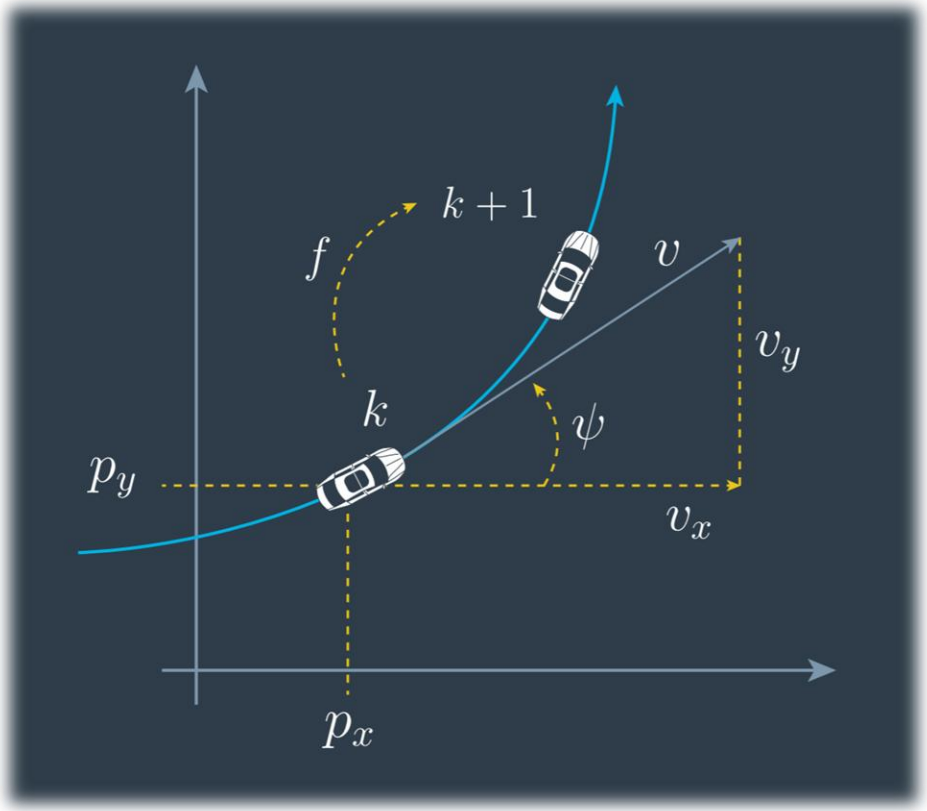


Rule-based Prediction Methods

Physics-based method (선회모델: CTRV, CTRA model)

CTRV Model

- CTRV (Constant Turn Rate & Velocity) Model
  - 현재의 속도와 turn rate(yaw rate) 을 유지한다고 가정



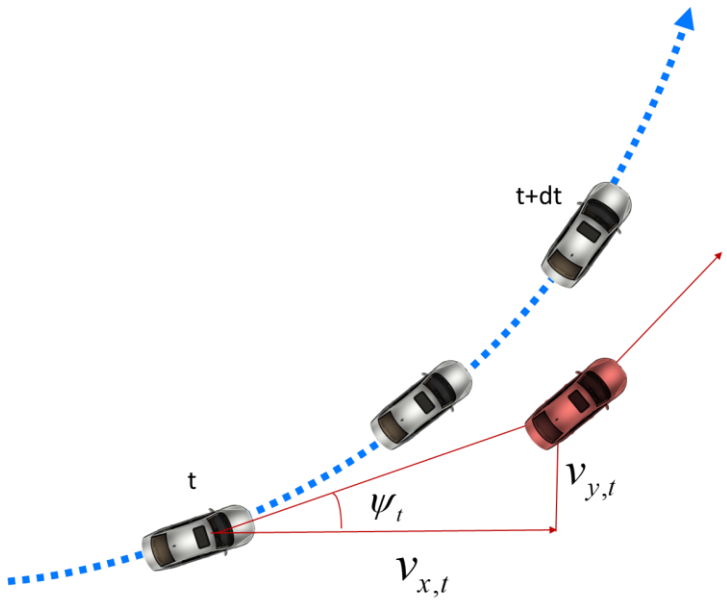
State–space equation

State : X= $\left[x,\ y,\ v,\ \psi,\ \dot{\psi}\right]^T$

$$\begin{cases} x_{t+1} = x_t + \frac{v_t}{\dot{\psi}_t}(sin(\psi_t + \dot{\psi}_t dt) - sin(\psi_t)) \\ y_{t+1} = y_t + \frac{v_t}{\dot{\psi}_t}(-cos(\psi_t + \dot{\psi}_t dt) + cos(\psi_t)) \\ v_{t+1} = v_t \\ \psi_{t+1} = \psi_t + \dot{\psi}_t dt \\ \dot{\psi}_{t+1} = \dot{\psi}_t \end{cases}$$

CTRV Model

- 작은 구간으로 나눠서 위치 변화를 추적



$$v_{x,t} = v_t cos(\psi_t)$$
$$v_{y,t} = v_t sin(\psi_t)$$

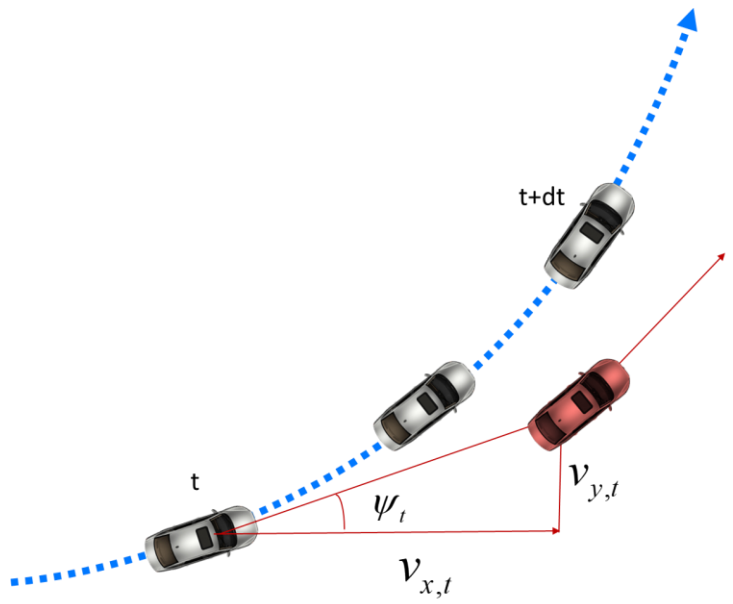
$$x_{t+1} = x_t + \sum_k v_{t+k} cos(\psi_{t+k})$$
$$y_{t+1} = y_t + \sum_k v_{t+k} sin(\psi_{t+k})$$

$$x_{t+dt} = x_t + \int_0^{dt} v_{t+k} cos(\psi_{t+k}) dk$$
$$y_{t+dt} = y_t + \int_0^{dt} v_{t+k} sin(\psi_{t+k}) dk$$

$$v_{t+k} = v_t$$
$$\psi_{t+k} = \psi_t + \dot{\psi}_t \cdot k$$

CTRV Model

- 작은 구간으로 나눠서 위치 변화를 추적



$$x_{t+dt} = x_t + \int_0^{dt} v_t cos(\psi_t + \dot{\psi}_t \cdot k) dk$$

$$y_{t+dt} = y_t + \int_0^{dt} v_t sin(\psi_t + \dot{\psi}_t \cdot k) dk$$

$$\int \sin(ax)dx = -\cos(ax)/a$$

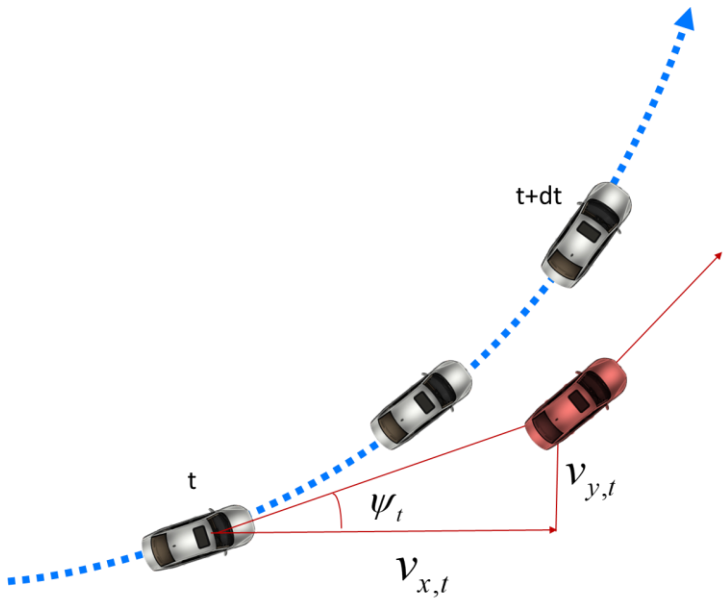
$$\int \cos(ax)dx = \sin(ax)/a$$

$$x_{t+1} = x_t + \frac{v_t}{\dot{\psi}_t}(sin(\psi_t + \dot{\psi}_t dt) - sin(\psi_t))$$

$$y_{t+1} = y_t + \frac{v_t}{\dot{\psi}_t}(-cos(\psi_t + \dot{\psi}_t dt) + cos(\psi_t))$$

CTRV Model

- 작은 구간으로 나눠서 위치 변화를 추적



State–space equation

State : X= $[x,\ y,\ v,\ \psi,\ \dot{\psi}]^T$

$$\begin{cases} x_{t+1} = x_t + \frac{v_t}{\dot{\psi}_t}(sin(\psi_t + \dot{\psi}_t dt) - sin(\psi_t)) \\ y_{t+1} = y_t + \frac{v_t}{\dot{\psi}_t}(-cos(\psi_t + \dot{\psi}_t dt) + cos(\psi_t)) \\ v_{t+1} = v_t \\ \psi_{t+1} = \psi_t + \dot{\psi}_t dt \\ \dot{\psi}_{t+1} = \dot{\psi}_t \end{cases}$$

CTRV Model

- 만약 yaw rate이 0 근처로 가게 되면 어떤 현상이 일어날까?

$$\begin{cases} x_{t+1} = x_t + \frac{v_t}{\dot{\psi}_t}(sin(\psi_t + \dot{\psi}_t dt) - sin(\psi_t)) \\ y_{t+1} = y_t + \frac{v_t}{\dot{\psi}_t}(-cos(\psi_t + \dot{\psi}_t dt) + cos(\psi_t)) \\ v_{t+1} = v_t \\ \psi_{t+1} = \psi_t + \dot{\psi}_t dt \\ \dot{\psi}_{t+1} = \dot{\psi}_t \end{cases}$$

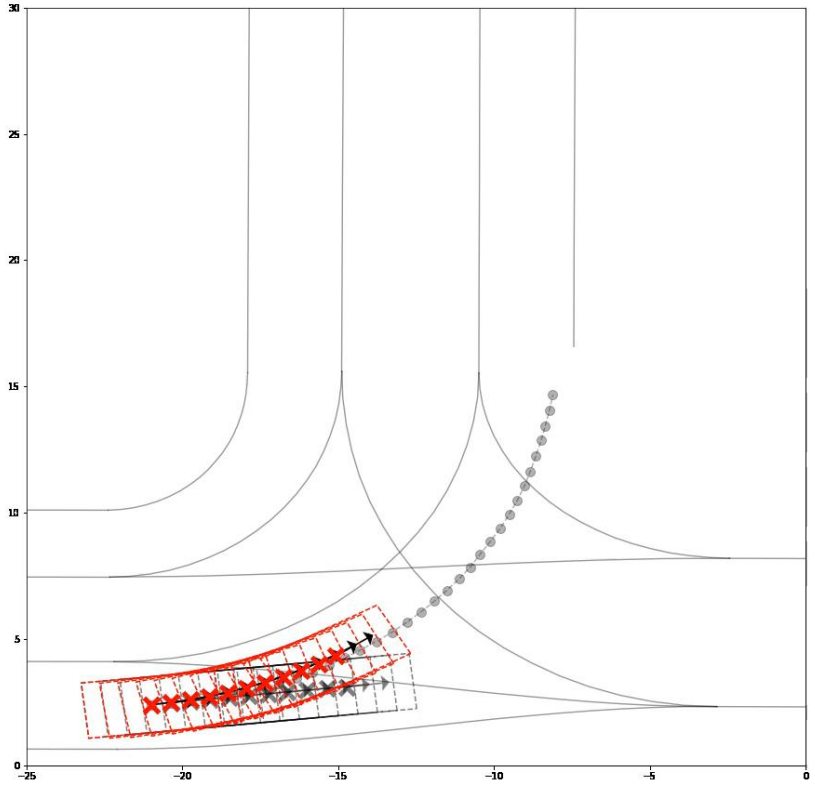
$$x_{t+dt} = x_t + \int_0^{dt} v_t(cos(\psi_t)cos(\dot{\psi}_t \cdot k) - sin(\psi_t)sin(\dot{\psi}_t \cdot k))dk$$

$$y_{t+dt} = y_t + \int_0^{dt} v_t(sin(\psi_t)cos(\dot{\psi}_t \cdot k) + cos(\psi_t)sin(\dot{\psi}_t \cdot k))dk$$

1 0 1 0

$$\begin{cases} x_{t+1} = x_t + vcos(\psi_t)dt \\ y_{t+1} = y_t + vsin(\psi_t)dt \\ v_{t+1} = v_t \\ \psi_{t+1} = \psi_t + \dot{\psi}_t dt \\ \dot{\psi}_{t+1} = \dot{\psi}_t \end{cases}$$

CTRV Model



: GT(1s)

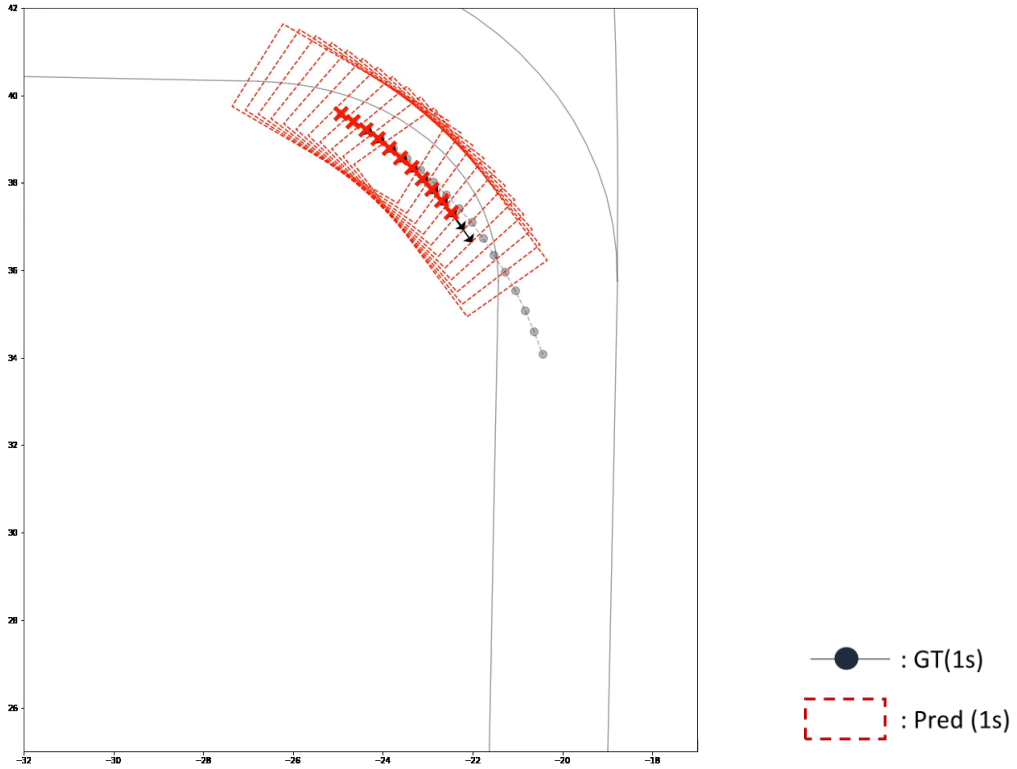
: Pred_CTRV (1s)

: Pred_CA (1s)

HYU

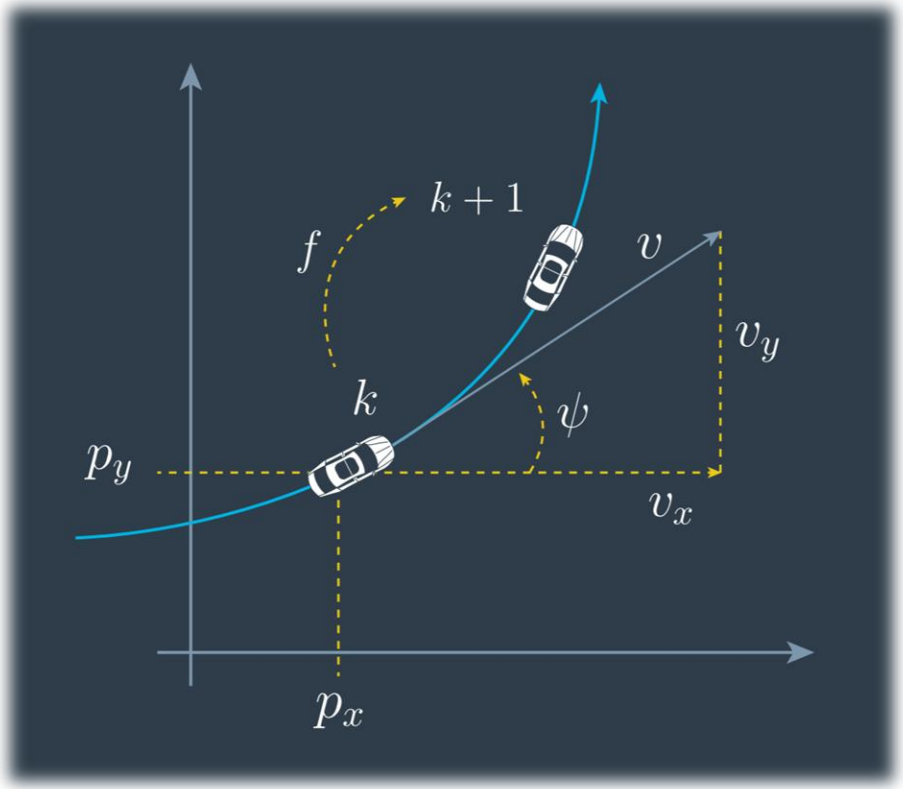
CRTV Model (Limitation)

- 코너에 진입/ 코너에 진출 시 잘 반영을 못함
  - 실제 운전시에는 코너 진입을 하면서 감속, 진출을 하면서 가속을 함



CTRA Model

- CTRA (Constant Turn Rate & Acceleration) Model
  - 현재의 turn rate(yaw rate)과 가속도를 유지한다고 가정



State–space equation

State : X = $[x, y, v, a, \psi, \dot{\psi}]T$

$$
\begin{cases}
x_{t+1} = x_t + \frac{1}{\dot{\psi}_t}((v_t + a_t dt)sin(\psi_t + \dot{\psi}_t) - v_t sin(\psi_t)) + \frac{a_t}{\dot{\psi}_t^2}(cos(\psi_t + \dot{\psi}_t dt) - cos(\psi_t)) \\
y_{t+1} = y_t + \frac{1}{\dot{\psi}_t}(-(v_t + a_t dt)cos(\psi_t + \dot{\psi}_t) + v_t cos(\psi_t)) + \frac{a_t}{\dot{\psi}_t^2}(sin(\psi_t + \dot{\psi}_t dt) - sin(\psi_t)) \\
v_{t+1} = v_t + a_t dt \\
a_{t+1} = a_t \\
\psi_{t+1} = \psi_t + \dot{\psi}_t dt \\
\dot{\psi}_{t+1} = \dot{\psi}_t
\end{cases}
$$

CTRA Model

$$x_{t+dt} = x_t + \int_0^{dt} v_{t+k} cos(\psi_{t+k}) dk$$

$$y_{t+dt} = y_t + \int_0^{dt} v_{t+k} sin(\psi_{t+k}) dk$$

$$v_{t+k} = v_t + a_t k$$

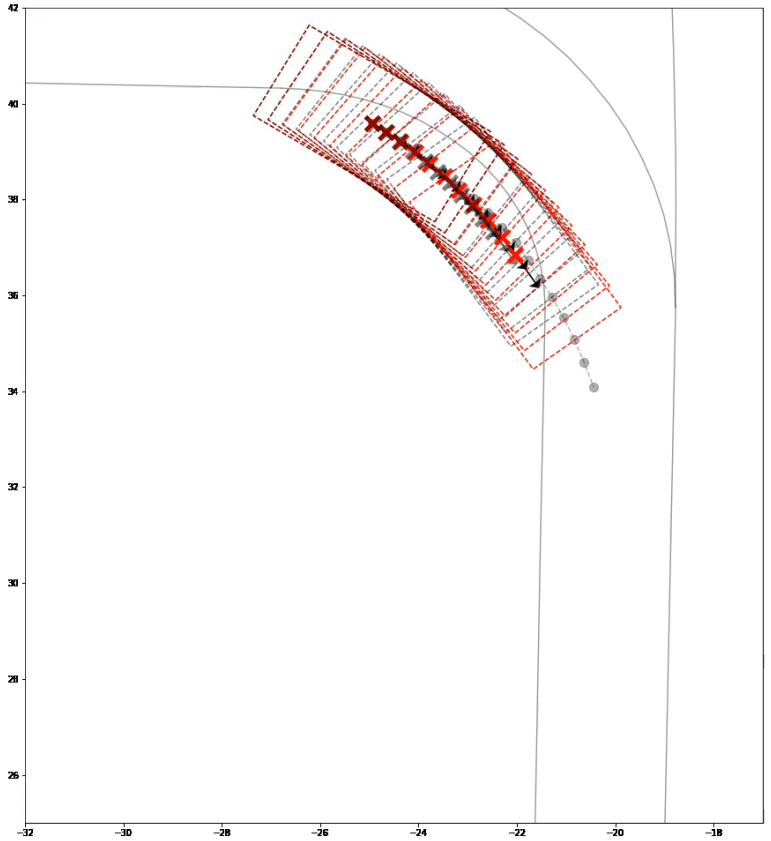
$$\psi_{t+k} = \psi_t + \dot{\psi}_t \cdot k$$

$$\int x sin(ax) dx = \frac{sin(ax)}{a^2} - \frac{x cos(ax)}{a}$$

$$\int x cos(ax) dx = \frac{cos(ax)}{a^2} + \frac{x\,sin(ax)}{a}$$

$$\begin{cases} x_{t+1} = x_t + \frac{v_t}{\dot{\psi}_t}(sin(\psi_t + \dot{\psi}_t dt) - sin(\psi_t)) + \frac{a_t}{\dot{\psi}^2}(cos(\psi + \dot{\psi} dt) + \dot{\psi} dt(sin(\psi + \dot{\psi} dt)) - cos(\psi)) \\ y_{t+1} = y_t + \frac{v_t}{\dot{\psi}_t}(-cos(\psi_t + \dot{\psi}_t dt) + cos(\psi_t)) + \frac{a_t}{\dot{\psi}^2}(sin(\psi + \dot{\psi} dt) + \dot{\psi} dt(-cos(\psi + \dot{\psi} dt)) - sin(\psi)) \\ v_{t+1} = v_t + a_t dt \\ a_{t+1} = a_t \\ \psi_{t+1} = \psi_t + \dot{\psi}_t dt \\ \dot{\psi}_{t+1} = \dot{\psi}_t \end{cases}$$

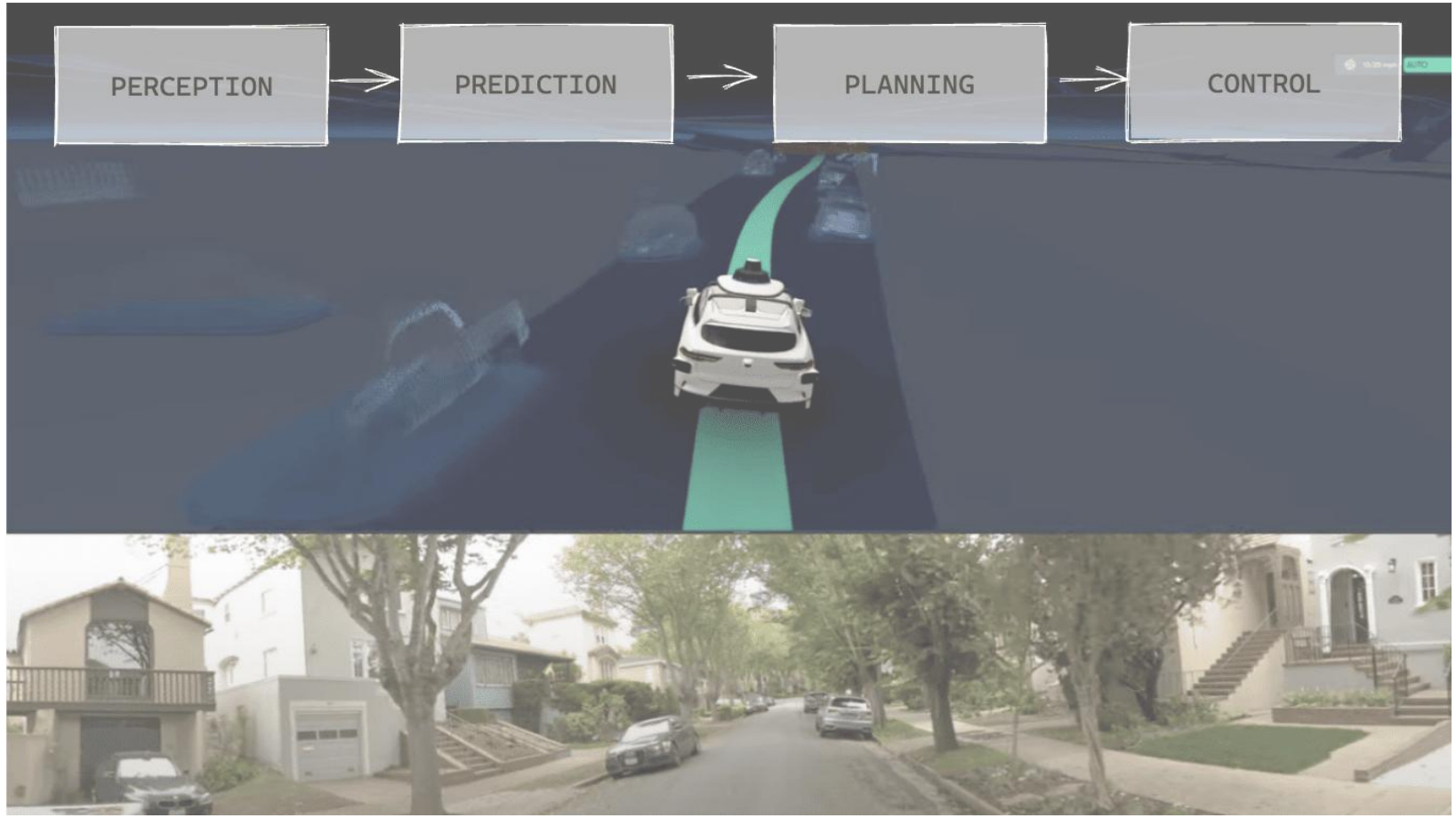
CTRA Model



: GT(1s)

: Pred_CTRA (1s)

: Pred_CTRV (1s)

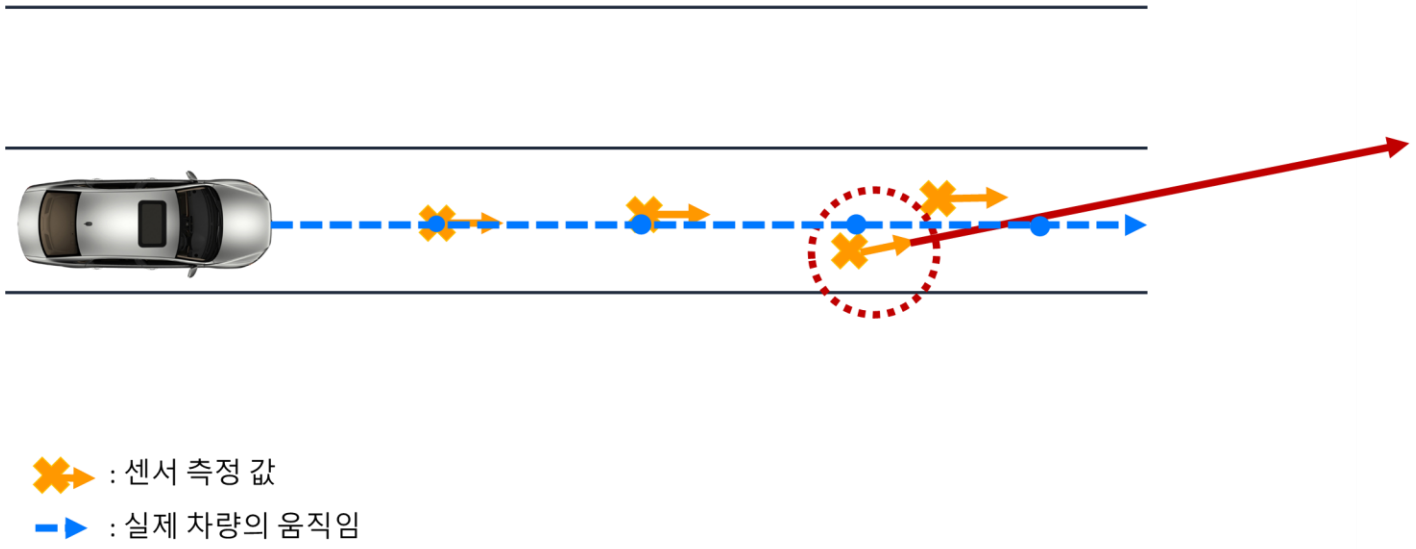


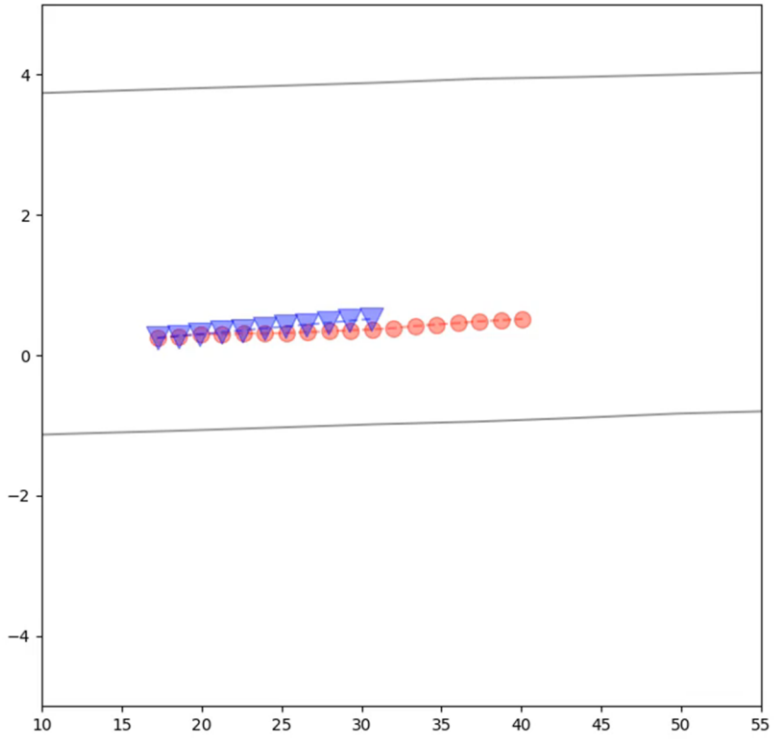
Rule-based Prediction Methods

Kalman Filter (KF)

센서 값으로 충분?

- 경로 예측은 센서에서 측정된 값을 사용해서 진행됨
- 그러나 센서에는 노이즈가 존재





센서 값으로 충분?

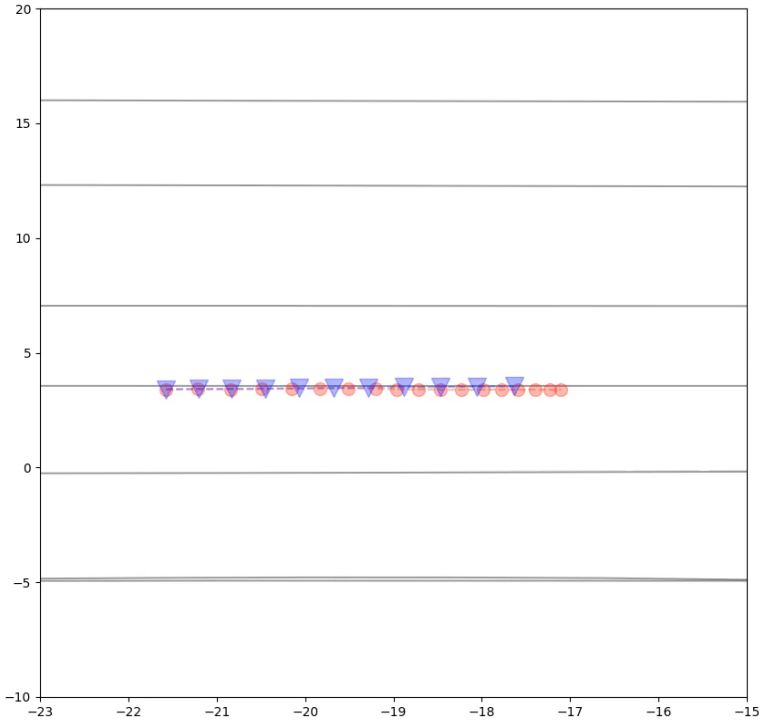
- 예측 모델에 사용되는 모든 변수를 센서에서 받을 수 없음
- 일부는 정확한 값을 받기가 어려움

CTRA 모델

$$
\begin{cases}
x_{t+1} = x_t + \frac{v_t}{\dot{\psi}_t}(sin(\psi_t + \dot{\psi}_t dt) - sin(\psi_t)) + \frac{a_t}{\dot{\psi}_t^2}(cos(\psi_t + \dot{\psi}_t dt) - cos(\psi_t)) \\
y_{t+1} = y_t + \frac{v_t}{\dot{\psi}_t}(-cos(\psi_t + \dot{\psi}_t dt) + cos(\psi_t)) + \frac{a_t}{\dot{\psi}_t^2}(sin(\psi_t + \dot{\psi}_t dt) - sin(\psi_t)) \\
v_{t+1} = v_t + a_t dt \\
a_{t+1} = a_t \\
\psi_{t+1} = \psi_t + \dot{\psi}_t dt \\
\dot{\psi}_{t+1} = \dot{\psi}_t
\end{cases}
$$

센서 값으로 충분?

- 예측 모델에 사용되는 모든 변수를 센서에서 받을 수 없음
- 일부는 정확한 값을 받기가 어려움

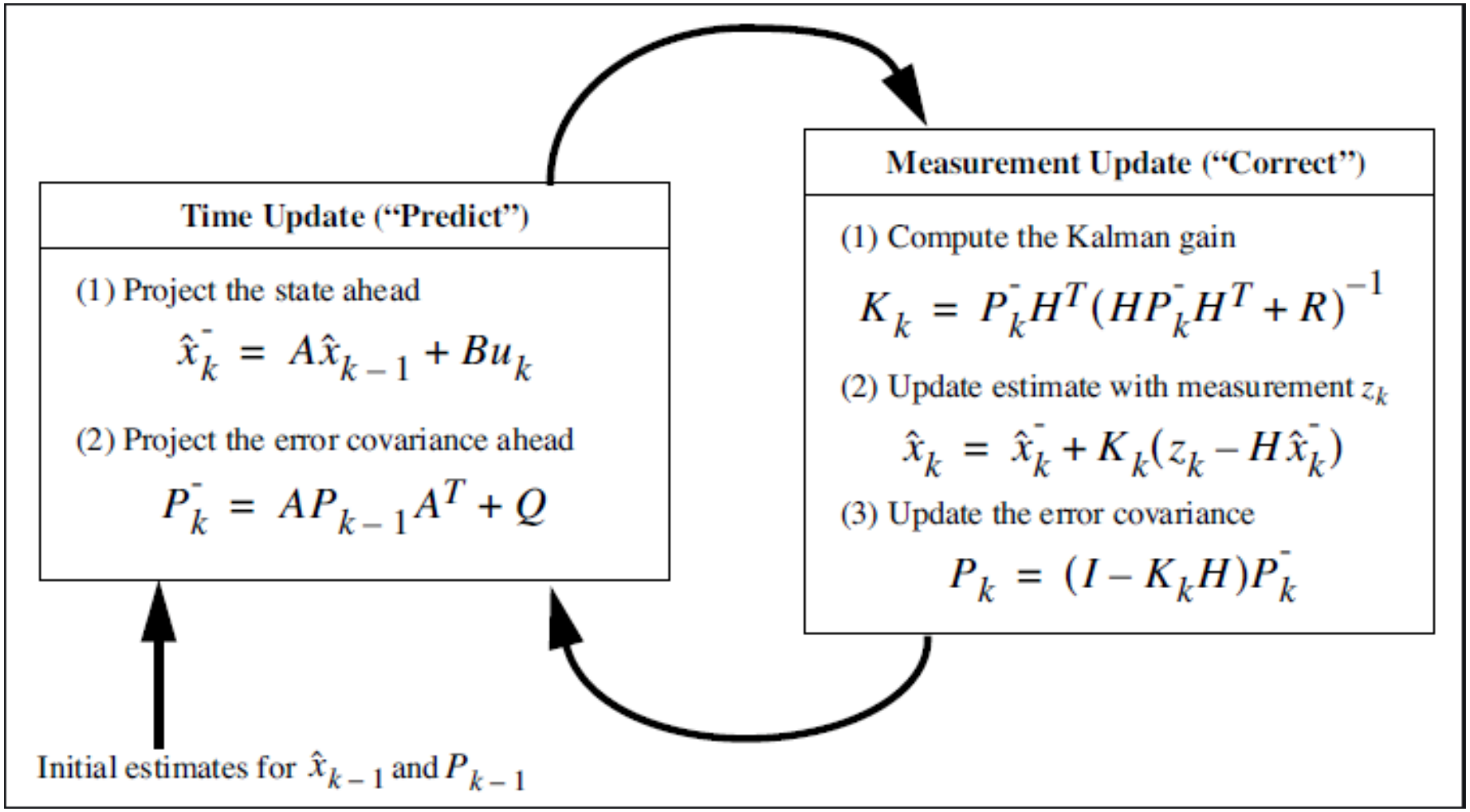


센서 값으로 충분? Summary

- 센서의 측정 값 자체에 노이즈가 포함되어 있음
  - smoothing이 필요

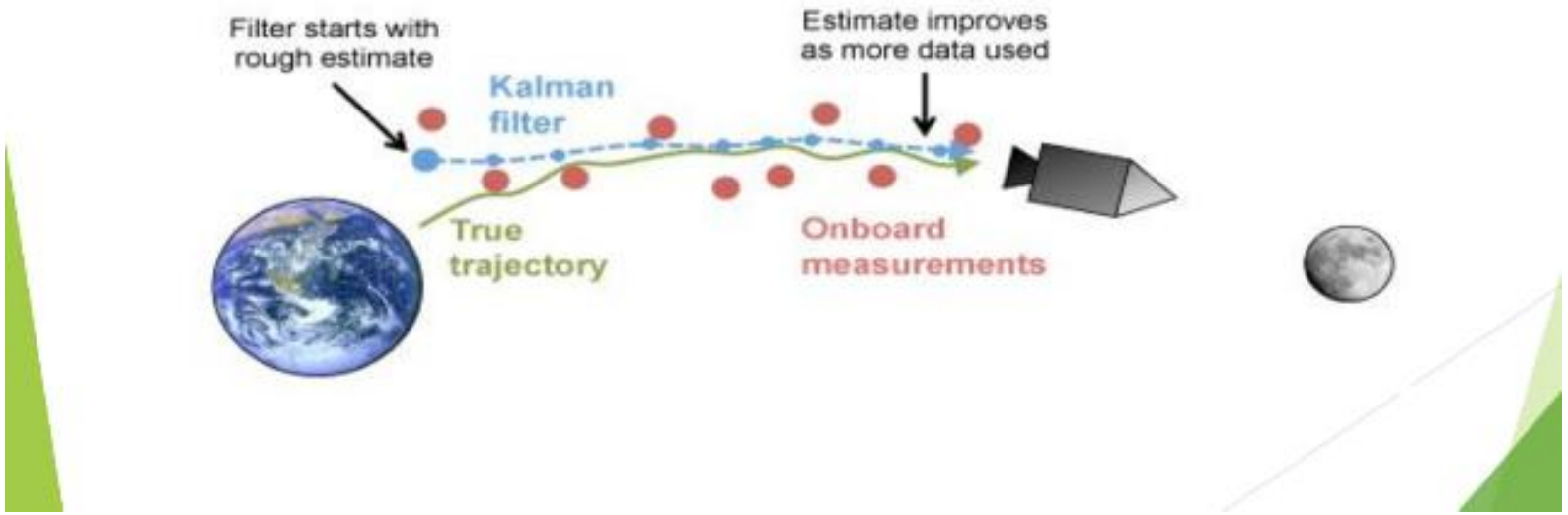
- 예측 모델에 사용되는 모든 변수를 센서에서 받을 수 없음
  - 변수에 대한 estimation이 필요

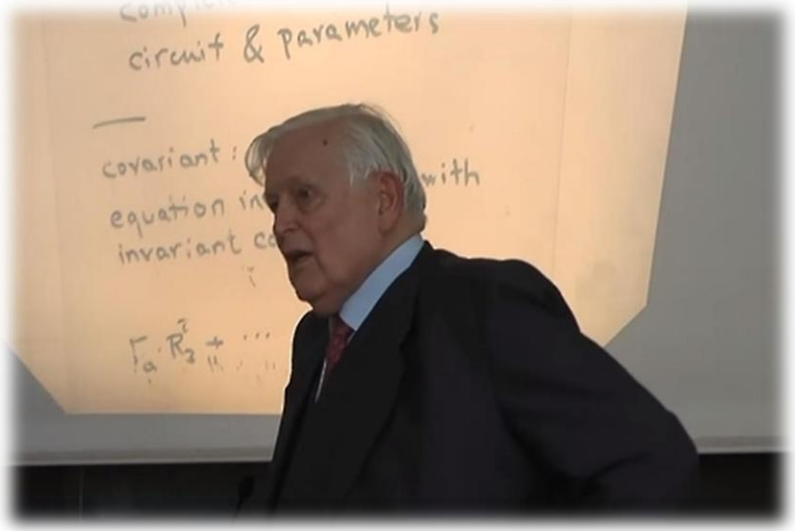
KF Overall Step



Kalman Filter

- 잡음이 포함되어 있는 측정치를 바탕으로 상태를 추정하는 필터로 1960년대 초, 루돌프 칼만이 개발
- 칼만 필터는 컴퓨터 비전, 로봇 공학, 레이다 등의 여러 분야에 사용됨
- 과거에 수행한 측정값을 바탕으로 현재의 상태 변수의 분포를 recursive하게 추정
- 대표적인 예시는 Apollo project 로, Apollo 11호 달착륙선의 궤적을 추정하는데 사용됨





KF Intuition

- 무엇을 믿을 것인가?
- 차량의 상태를 알아내는 방법
  - 알고 있는 모델을 바탕으로 이전 상태를 이용하여 연산을 수행
  - 다음 시점에 들어오는 센서의 측정값을 활용
- 그러나 모델과 센서는 각각의 오차가 존재

t

Model

Measure

t+1

KF Intuition

- 모델 : 실제 움직임을 단순하게 모사했기 때문에 생기는 오차
- 센서 : 센서 자체에서 생기는 noise로 인한 오차
- 따라서 실제 위치로 옮기기 위해서는 오차만큼 더해줘야 한다.

$$\begin{cases} x_{t+1} = x_t + v_{x.t} * dt & +\alpha \\ y_{t+1} = y_t + v_{y.t} * dt & +\beta \end{cases} \qquad \begin{matrix} m_x + \gamma \\ m_y + \epsilon \end{matrix}$$

t

t+1

센서 측정값이 100% 정확하다?

다음 상태는 Measure된 위치

모델이 100% 정확하다?

다음 상태는 모델을 통해 알아낸 위치

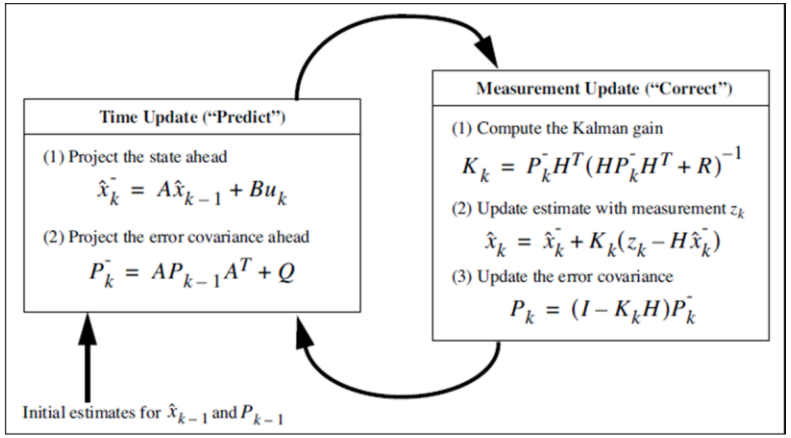
센서 값이 모델보다 더 정확하다?

다음 상태는 센서 값에 더 가까울 것이다.

-> 더 정확한 쪽에 가중치를 두고 합치자! 어떻게?

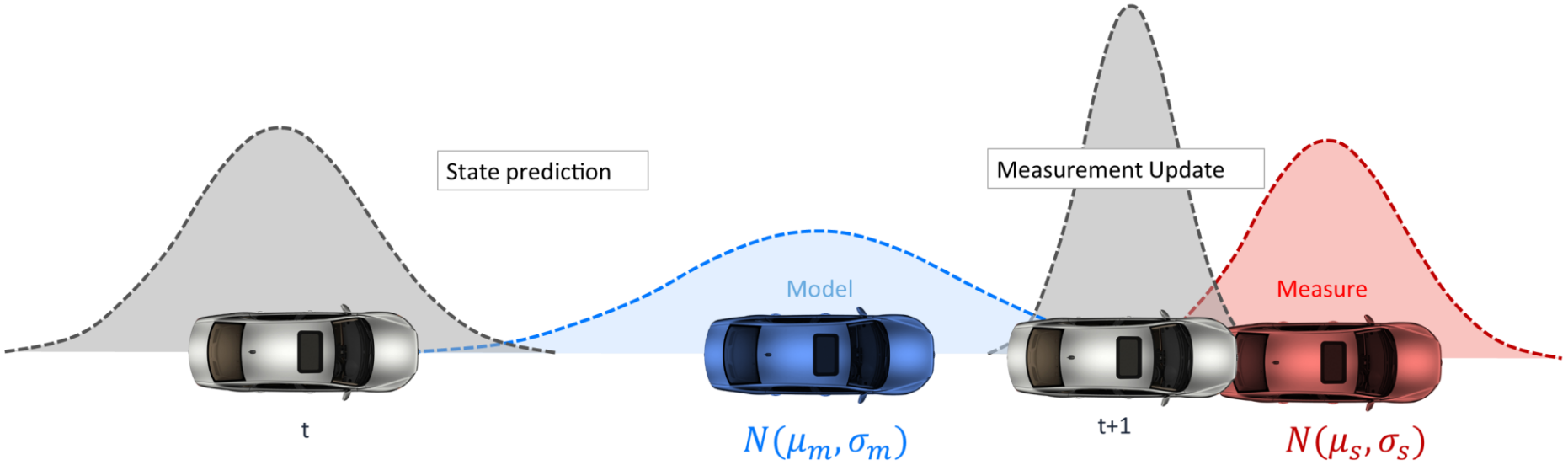
KF 구조

- state prediction과 measurement update 부분으로 나누어져 있음
- 각각의 파트에서 분포의 업데이트가 발생



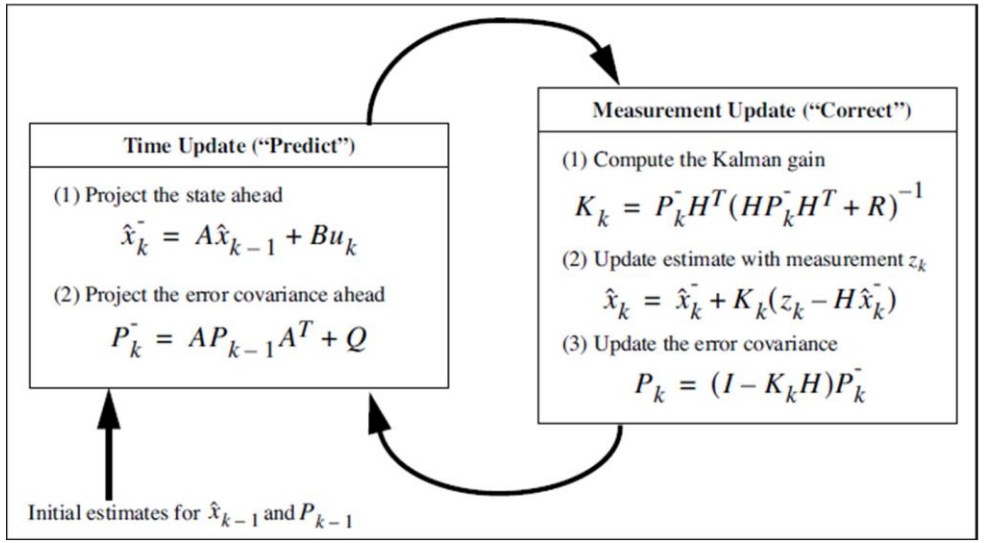
State prediction
: 이전 상태 값에서 모델을 통해 새로운 상태를 예측

Measurement Update
: 예측된 값과 새로 들어온 센서 값을 통해 상태를 보정



KF 구현

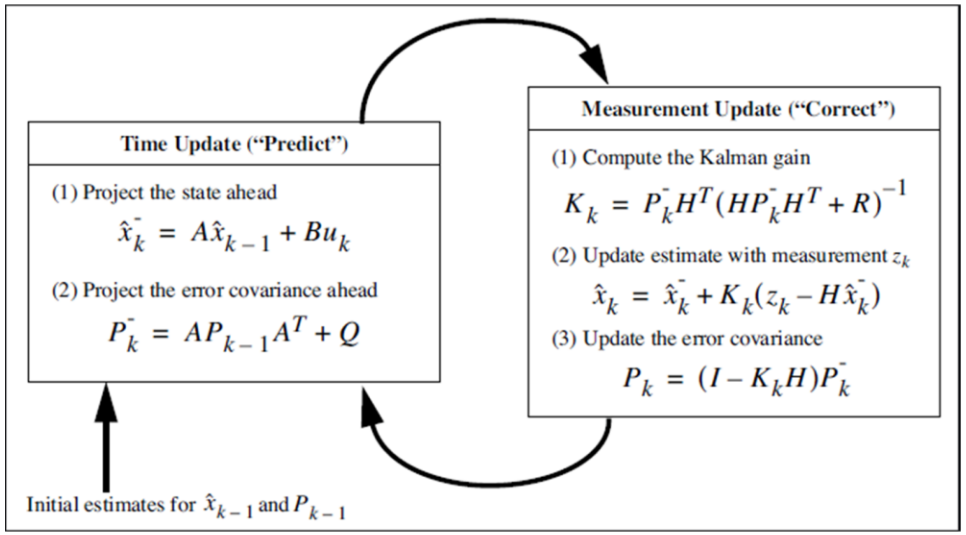
- Python의 class를 이용하여 Kalman Filter 구현




```python
class KalmanFilter():
    def __init__(self, x_dim, z_dim):

        """
        필요한 변수들 initialization

        """


    def predict(self, u=None, B=None, A=None, Q=None):

        """
        state prediction

        """


    def correction(self, z, R=None, H=None):

        """
        measurement update

        """
```

KF 구현

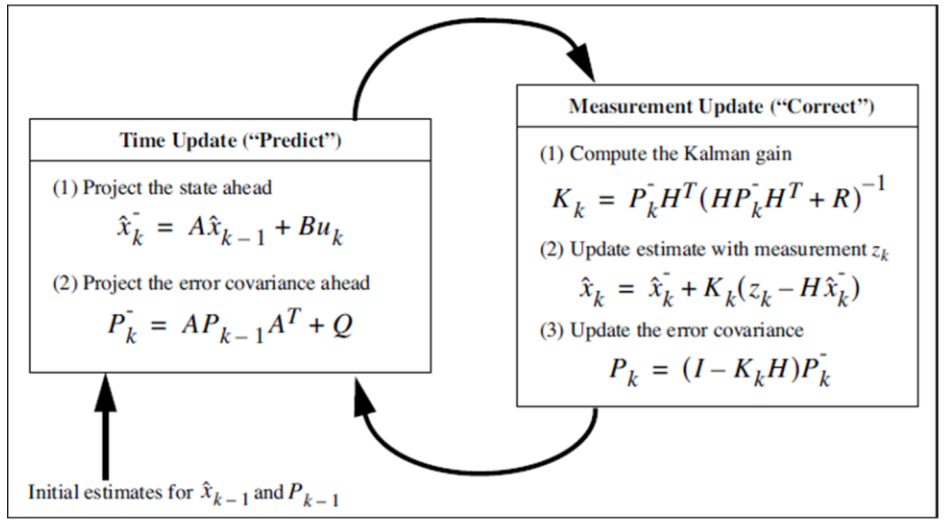
- Initialization




```python
def __init__(self, x_dim, z_dim):

    self.Q = np.eye(x_dim)
    self.R = np.eye(z_dim)
    self.B = None
    self.P = np.eye(x_dim)
    self.A = np.eye(x_dim)
    self.H = np.zeros((z_dim,x_dim))

    self.x = np.zeros((x_dim,1))
    self.y = np.zeros((z_dim,1))

    self.K = np.zeros((x_dim, z_dim))
    self.S = np.zeros((z_dim, z_dim))

    self._I = np.eye(x_dim)

    self.x_prior = self.x.copy()
    self.P_prior = self.P.copy()

    self.x_post = self.x.copy()
    self.P_post = self.P.copy()

    self.SI = np.zeros((z_dim, z_dim))
    self.inv = np.linalg.inv
```

KF 구현

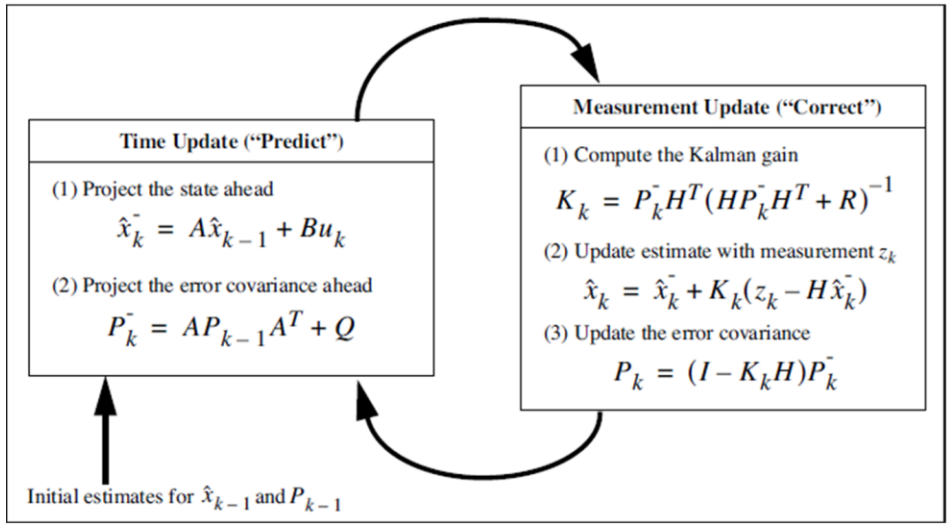
- State prediction




```
def predict(self, u=None, B=None, A=None, Q=None):

    if B is None:
        B = self.B
    if A is None:
        A = self.A
    if Q is None:
        Q = self.Q


    if B is not None and u is not None:
        self.x = np.dot(A, self.x) + np.dot(B, u)
    else:
        self.x = np.dot(A, self.x)


    self.P = np.dot(np.dot(A, self.P), A.T) + Q


    self.x_prior = self.x.copy()
    self.P_prior = self.P.copy()
```

KF 구현

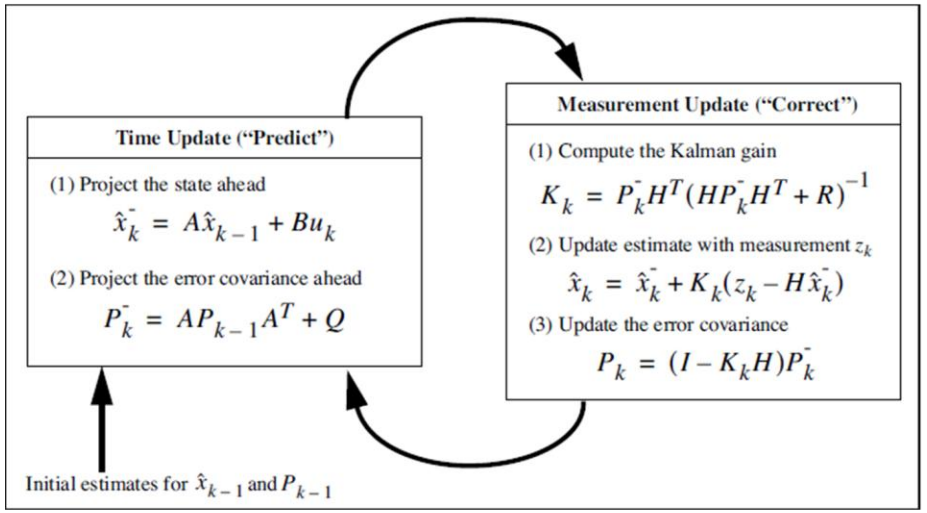
- Measurement update




```
def correction(self, z, R=None, H=None):

    if R is None:
        R = self.R

    if H is None:
        H = self.H

    self.y = z - np.dot(H, self.x)

    PHT = np.dot(self.P, H.T)

    self.S = np.dot(H, PHT) + R
    self.SI = self.inv(self.S)

    self.K = np.dot(PHT, self.SI)

    self.x = self.x + np.dot(self.K, self.y)

    I_KH = self._I - np.dot(self.K, H)

    self.P = np.dot(np.dot(I_KH, self.P), I_KH.T) +\
    np.dot(np.dot(self.K, R), self.K.T)

    self.x_post = self.x.copy()
    self.P_post = self.P.copy()
```

KF 구현

- Set up and implementation




```
# CA
x = [pose[0,0], pose[0,1], vel[0]*np.cos(theta[0]), vel[0]*np.sin(theta[0]),
     0, 0]
dt=0.1
A = np.array([[1,0,dt,0,1/2*(dt**2),0],
              [0,1,0,dt,0,1/2*(dt**2)],
              [0,0,1,0,dt,0],
              [0,0,0,1,0,dt],
              [0,0,0,0,1,0],
              [0,0,0,0,0,1]])
H = np.array([[1,0,0,0,0,0],[0,1,0,0,0,0],[0,0,1,0,0,0],[0,0,0,1,0,0]])
```

```
kf = KalmanFilter(6,4)
kf.A = A
kf.H = H
kf.x = x
X = [x]
for i in range(1,len(pose)):
    z= [pose[i,0], pose[i,1], vel[i]*np.cos(theta[i]), vel[i]*np.sin(theta[i])]

    kf.predict(Q=np.diag([0.00001,0.00001,0.001,0.001,0.1,0.1]))
    kf.correction(z=z, R=np.diag([0.1,0.1,0.1,0.1]))

    X.append(kf.x)

X = np.array(X)
```

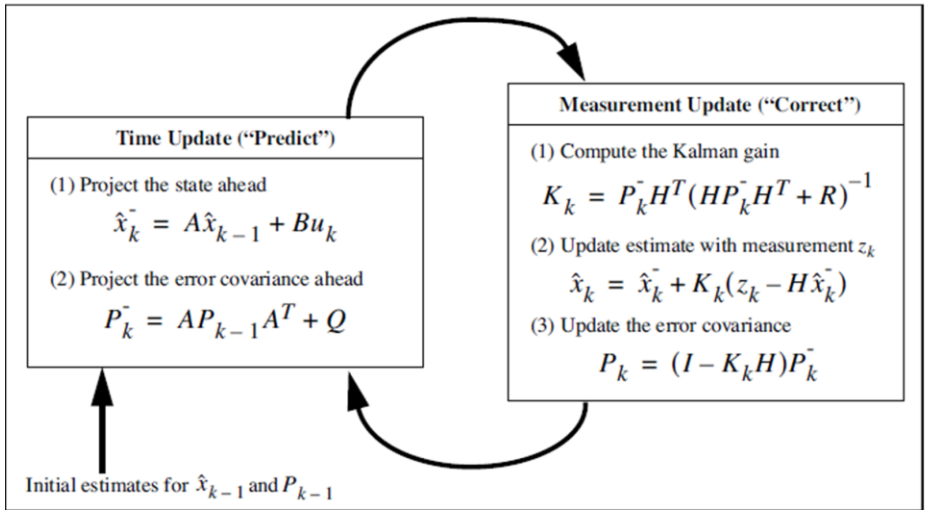
KF 구현

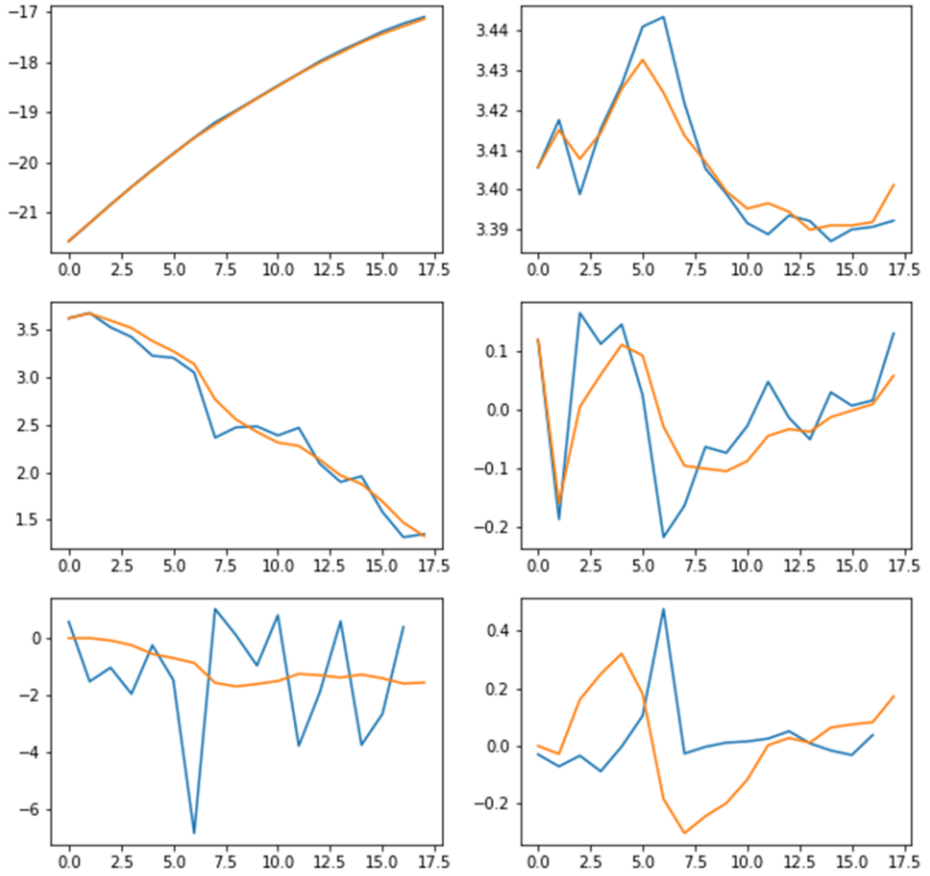
1) 현재 상태를 바탕으로 initial state, Q, R을 설정

2) Prediction Step

$$\begin{bmatrix} x_{t+1} \\ v_{t+1} \\ a_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & dt & dt^2/2 \\ 0 & 1 & dt \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ v_t \\ a_t \end{bmatrix}$$

3) Measure가 들어왔을 때 업데이트 수행





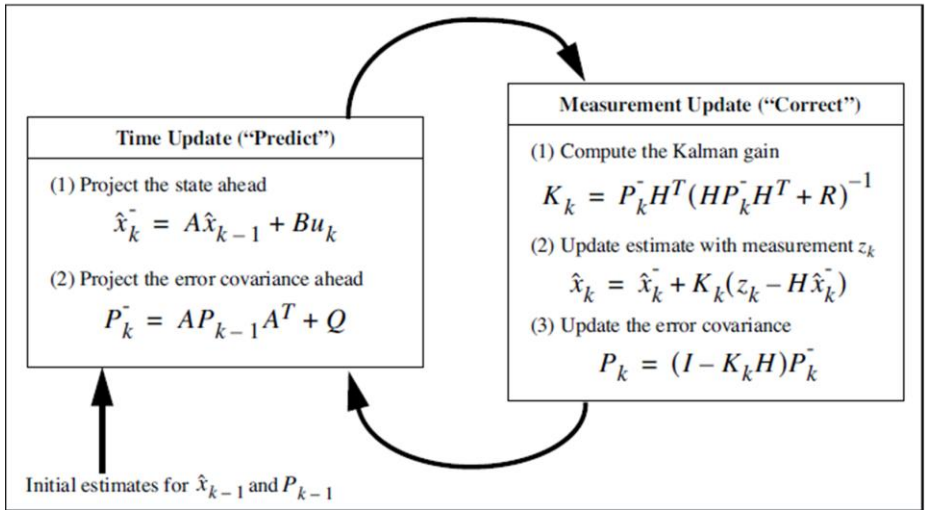
KF 구현

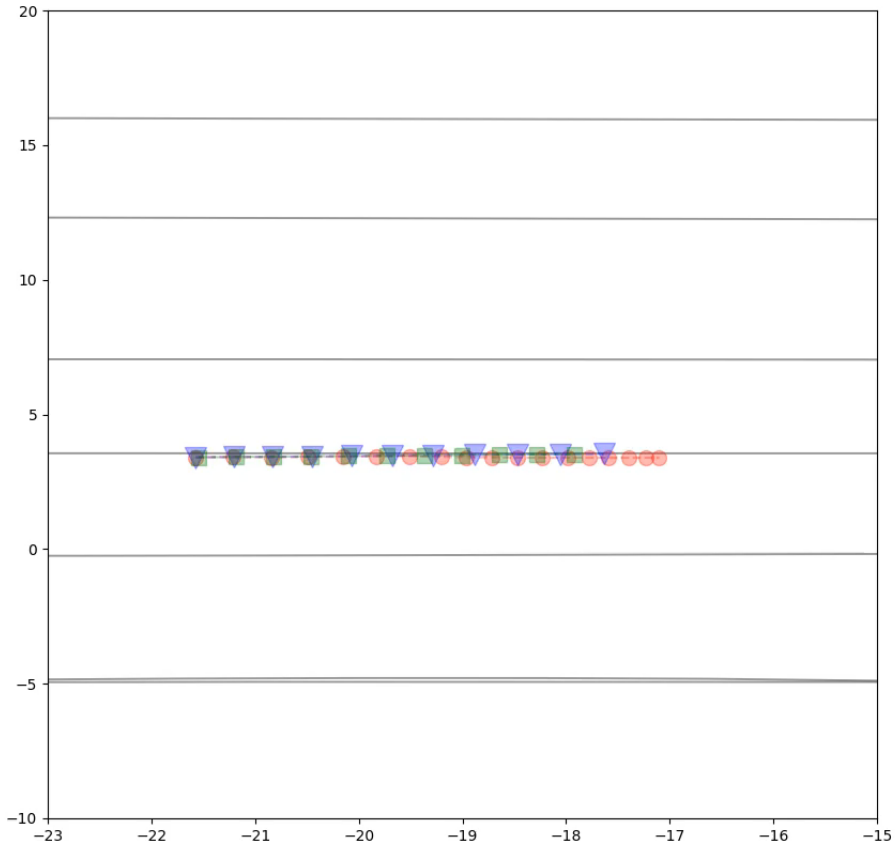
1) 현재 상태를 바탕으로 initial state, Q, R을 설정

2) Prediction Step

$$\begin{bmatrix} x_{t+1} \\ v_{t+1} \\ a_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & dt & dt^2/2 \\ 0 & 1 & dt \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ v_t \\ a_t \end{bmatrix}$$

3) Measure가 들어왔을 때 업데이트 수행



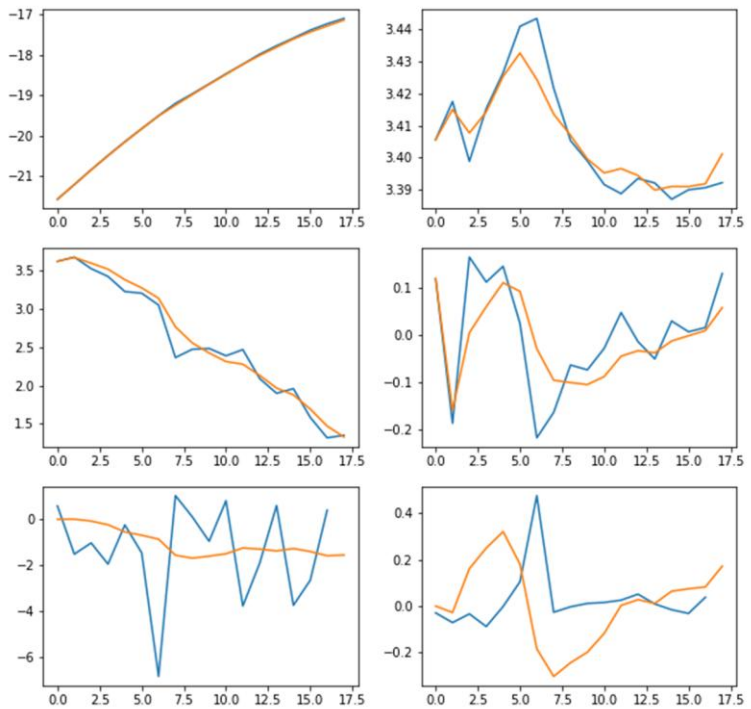


결과 및 고찰

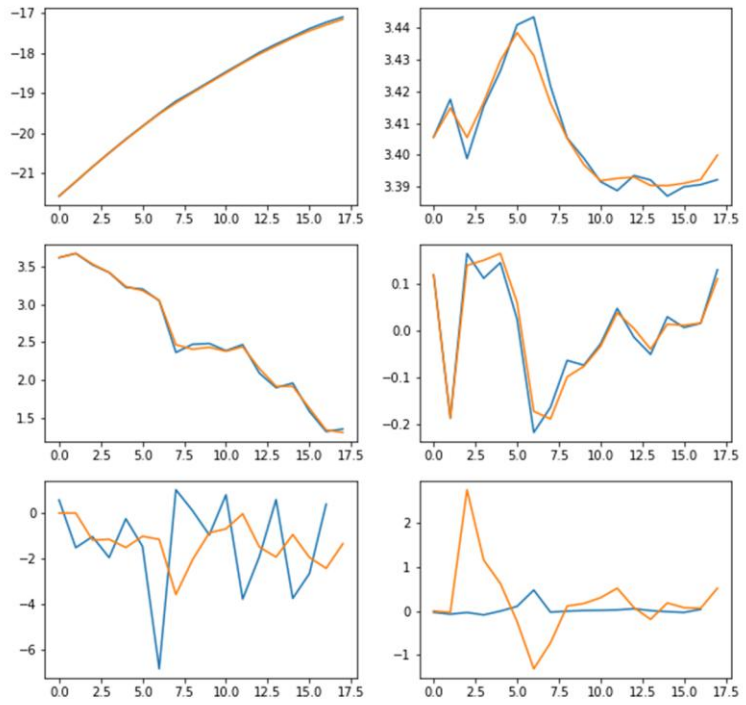
- Q , R은 어떻게 설정해주어야 하는가?
  - Q : 모델의 정확도로 인한 오차
  - R : 센서 노이즈로 인한 오차

$$\begin{bmatrix} x_{t+1} \\ v_{t+1} \\ a_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & dt & dt^2/2 \\ 0 & 1 & dt \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ v_t \\ a_t \end{bmatrix} + \begin{bmatrix} dt^2/2 \\ dt \\ 1 \end{bmatrix} w_a$$

```
kf.predict(Q=np.diag([0.00001,0.00001,0.001,0.001,0.1,0.1]))
kf.correction(z=z, R=np.diag([0.1,0.1,0.1,0.1]))
```

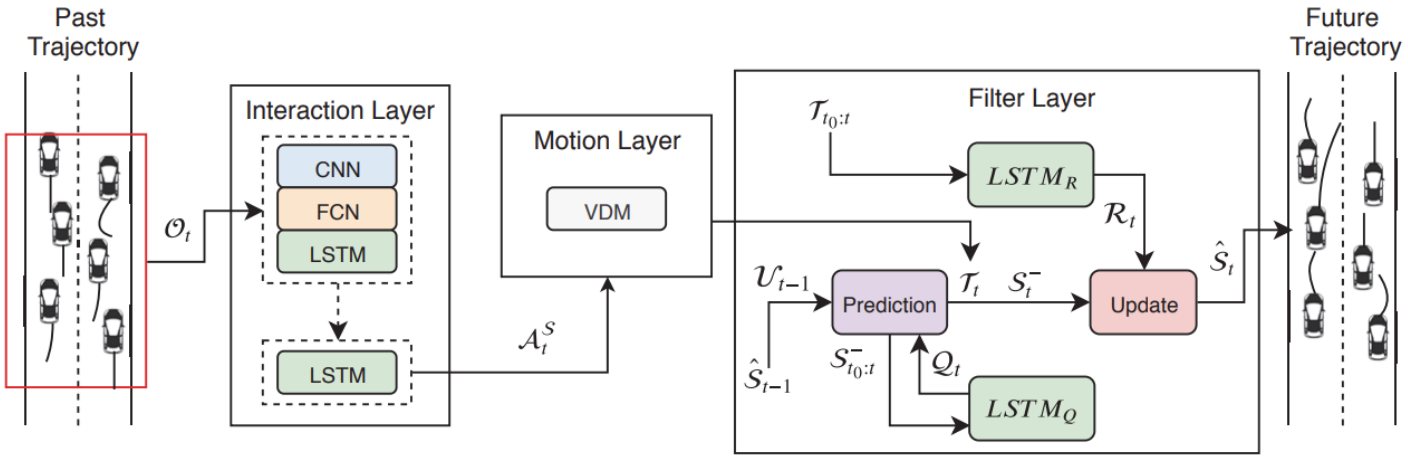


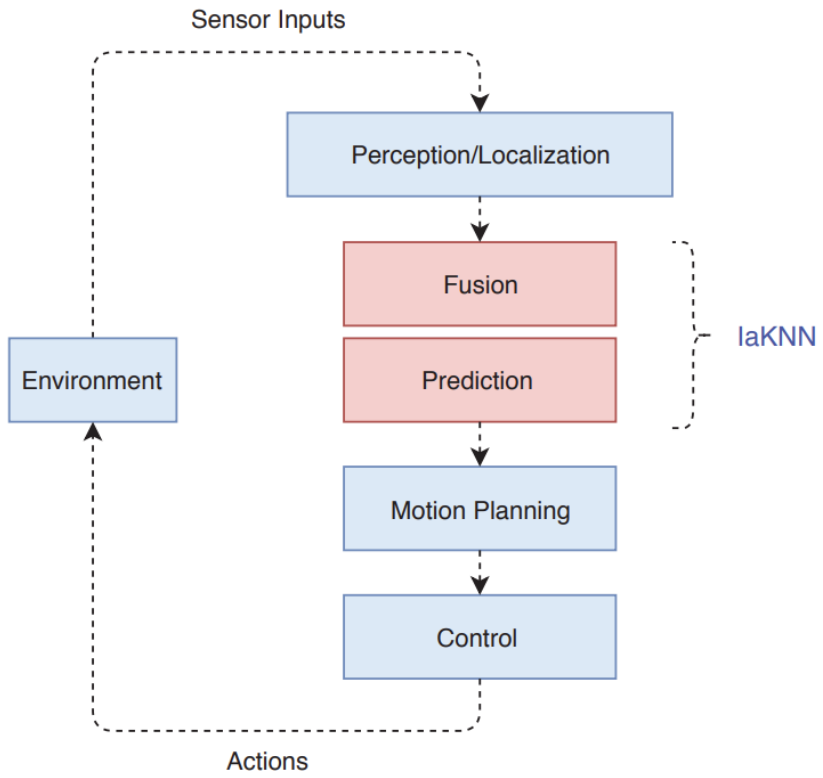
```
kf.predict(Q=np.diag([0.00001,0.00001,0.001,0.001,0.1,0.1]))
kf.correction(z=z, R=np.diag([0.1,0.1,0.001,0.001]))
```



Kalman Neural Networks

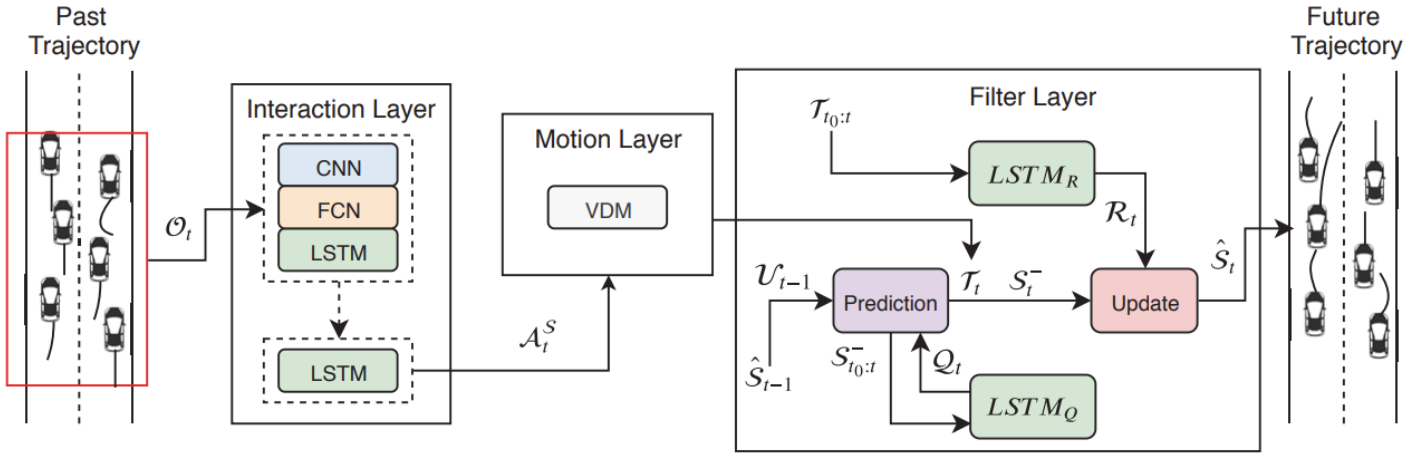
- Kalman Filter 의 Q, R matrix 를 추정하는 Neural Network 학습
- Measurement 용 경로를 Interaction layer network 로 학습





Kalman Neural Networks

- Interaction Layer



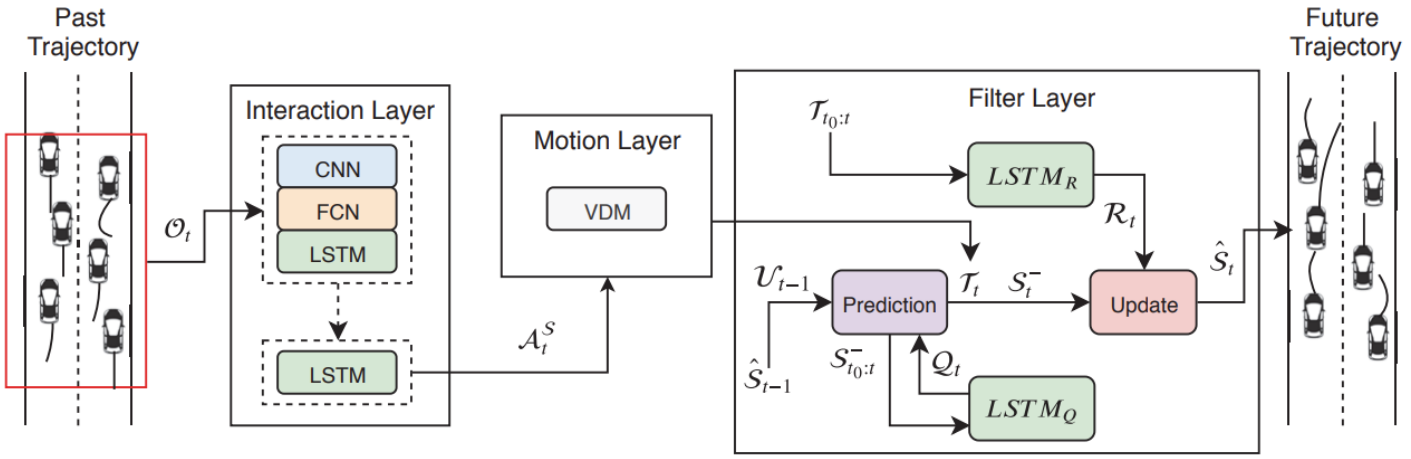
$$\mathcal{A}_t^{\mathcal{S}} = \textbf{Interaction}_{\{\mathcal{W},b\}}\left(a_{t_0:t}, w_{t_0:t}, l_{t_0:t}, d_{t_0:t}, e_{t_0:t}\right)$$

repulsive interaction forces

accelerations widths lengths relative distances

Kalman Neural Networks

- Motion Layer



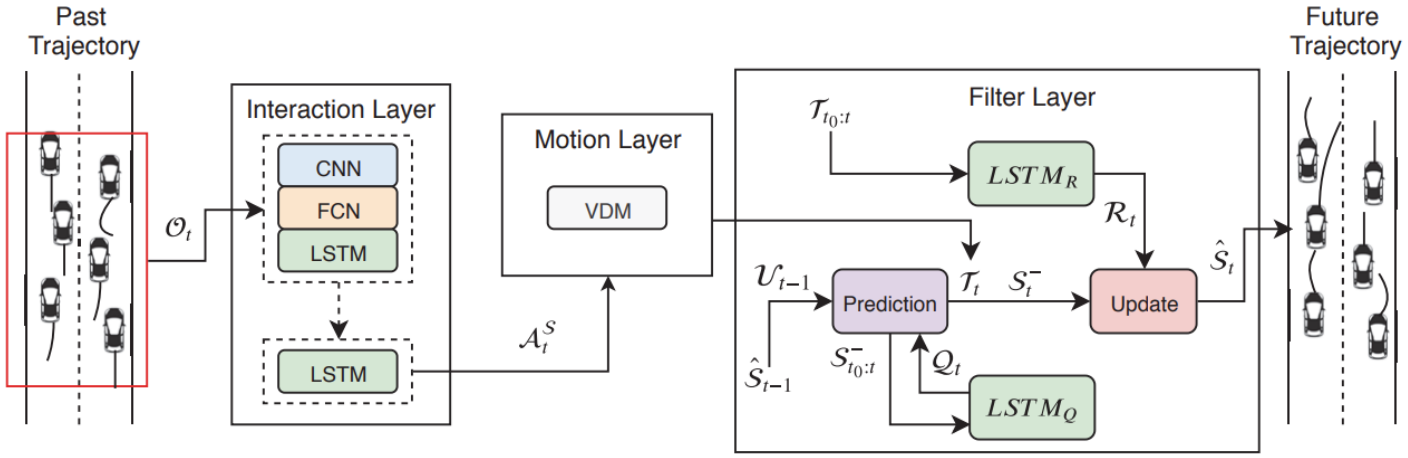
$$p_t = p_{t-1} + v_{t-1} \cdot \Delta t + \frac{1}{2} a_{t-1} \cdot \Delta t^2$$

$$v_{t+i} := \int_t^{t+i} \mathcal{A}^S dt$$

$$\mathcal{T}_t^i := \begin{bmatrix} \bar{p}_{t+1}^i \\ \bar{v}_{t+1}^i \\ \vdots \\ \bar{p}_{t+L}^i \\ \bar{v}_{t+L}^i \end{bmatrix}_{(2 \cdot L) \times 1}$$

Kalman Neural Networks

- Filter Layer



$$\mathcal{S}_t^i := \begin{bmatrix} p_{t+1}^i \\ v_{t+1}^i \\ \vdots \\ p_{t+L}^i \\ v_{t+L}^i \end{bmatrix}_{(2\cdot L)\times 1} \quad \text{and} \quad \mathcal{T}_t^i := \begin{bmatrix} \bar{p}_{t+1}^i \\ \bar{v}_{t+1}^i \\ \vdots \\ \bar{p}_{t+L}^i \\ \bar{v}_{t+L}^i \end{bmatrix}_{(2\cdot L)\times 1}$$

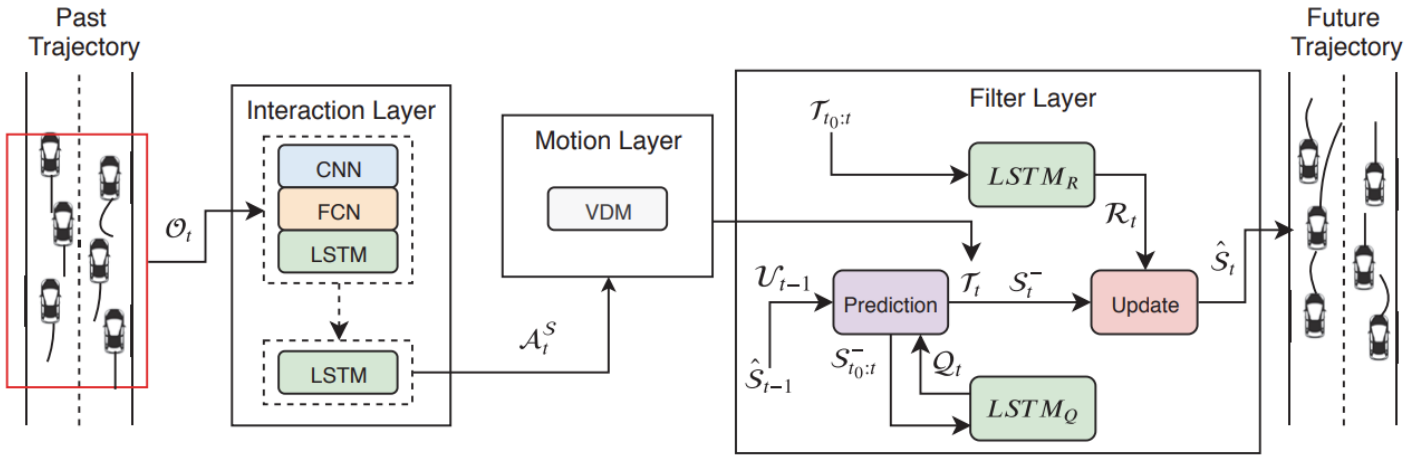
$$Q_t = \mathrm{LSTM}_Q\left(S_{t_0:t}^-\right), \qquad R_t = \mathrm{LSTM}_R(T_{t_0:t})$$

$$\mathcal{S}_t = \mathcal{F}\cdot\mathcal{S}_{t-1} + \mathcal{B}\cdot\mathcal{U}_{t-1} + \omega_t,$$
$$\mathcal{T}_t = \mathcal{S}_t + \eta_t$$

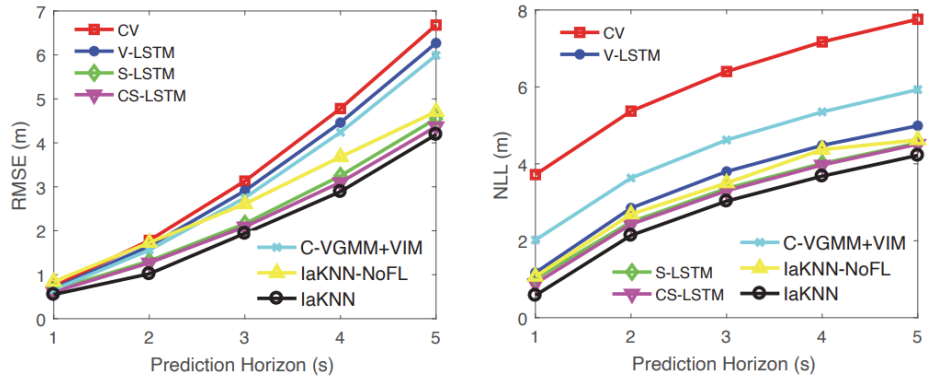
Kalman Neural Networks

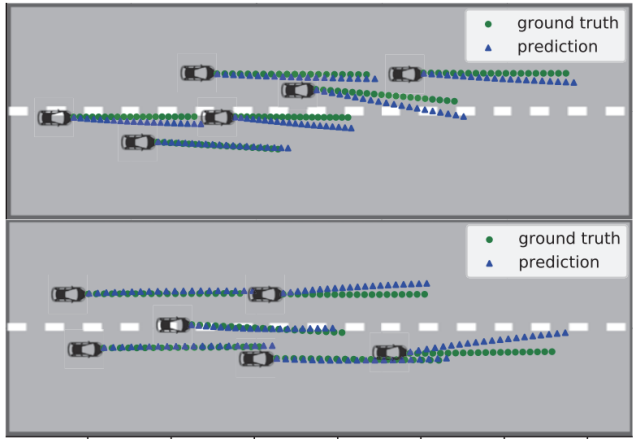
- Training loss

$$\mathcal{L}_{\{\mathcal{W},b\}} := \frac{1}{(L'+1)\cdot N} \cdot \sum_{i=1}^{N} \sum_{t=t_0}^{t_0+L'} ||\hat{\mathcal{S}}_t^i - G_t^i||^2,$$



- Results





Next class (week4.2)

- Prediction 2

Thanks for Listening!