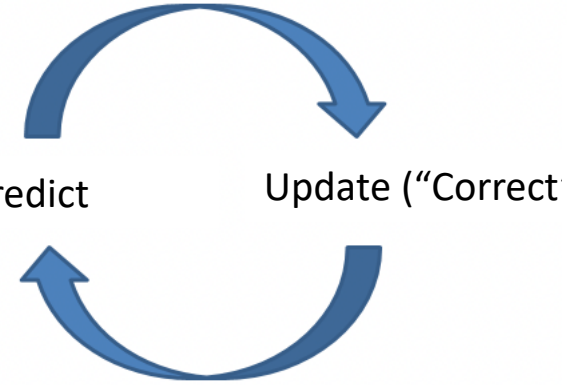
Intelligent Wireless Robotics

Lab -- Kalman filter

Spring 2023

Kalman Filter – Algorithm Overview



Algorithm Kalman_filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):

$\bar{\mu}_t = A_t\ \mu_{t-1} + B_t\ u_t$

$\bar{\Sigma}_t = A_t\ \Sigma_{t-1}\ A_t^T + R_t$

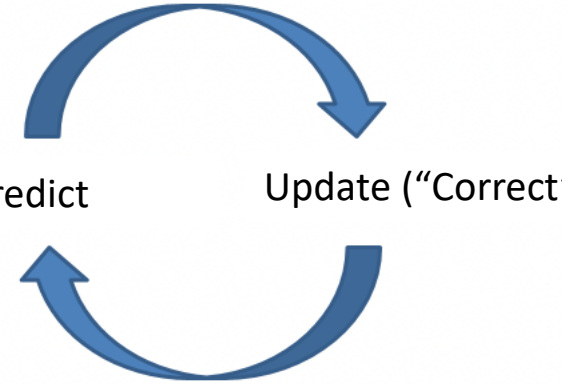
$K_t = \bar{\Sigma}_t\ C_t^T (C_t\ \bar{\Sigma}_t\ C_t^T + Q_t)^{-1}$

$\mu_t = \bar{\mu}_t + K_t(z_t - C_t\ \bar{\mu}_t)$

$\Sigma_t = (I - K_t\ C_t)\ \bar{\Sigma}_t$

return μ_t, Σ_t

Kalman Filter – Algorithm Overview



Algorithm Kalman_filter$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$**:**

$\bar{\mu}_t = A_t\ \mu_{t-1} + B_t\ u_t$

$\bar{\Sigma}_t = A_t\ \Sigma_{t-1}\ A_t^T + R_t$

$K_t = \bar{\Sigma}_t\ C_t^T (C_t\ \bar{\Sigma}_t\ C_t^T + Q_t)^{-1}$

$\mu_t = \bar{\mu}_t + K_t(z_t - C_t\ \bar{\mu}_t)$

$\Sigma_t = (I - K_t\ C_t)\ \bar{\Sigma}_t$

return μ_t, Σ_t

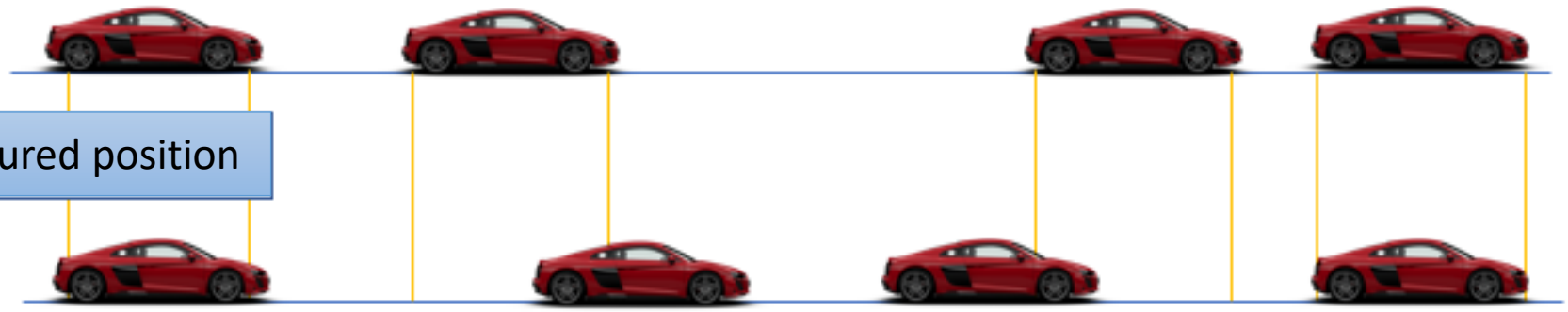
1. Predict the next state x'
2. Update by x' = x + K (z - x)

Toy Example

Tunnel Interference

Actual position

Measured position



Toy Example

```python
class KalmanFilterToy:
    def __init__(self):
        self.v = 0.  # velocity estimate
        self.prev_pos = 0.
        self.prev_t = 0.

    def predict(self, t):
        """predict position"""
        predict_pos = 0.
        return predict_pos


    def measure_and_update(self, measure_pos, t):
        """estimate velocity"""
        return
```

Estimate: velocity
Measurement: positions

Implement the predict function and the update function.

1. Predict function: predict the next position

   predict (t):

   x <- x + v Δt,

   where Δt = t – prev_t

2. Update function:

   update(measure_pos, t):

   z: velocity from measurement?

   v' <-- v + K (z - v)

   where the factor 0 < K < 1.

Do not forget:

self.prev_pos = measure_pos

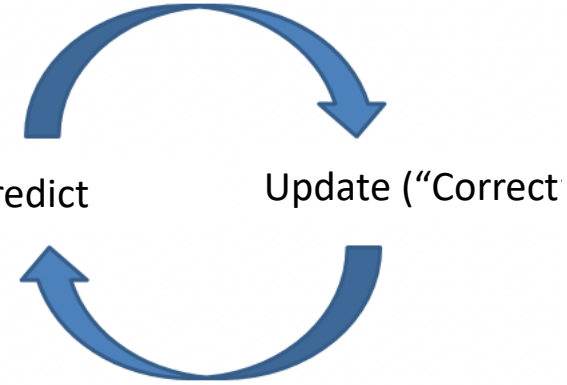
self.prev_t = t

Toy Example

Play with the following setting changes

- Change the values of alpha:
  - smaller, until K → 0
  - larger, until K → 1
- Set the speed constant or inconstant:
  - options['SPEED_CONSTANT'] = True
  - options['SPEED_CONSTANT'] = False

Kalman Filter – Algorithm Overview



Algorithm Kalman_filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):

$\bar{\mu}_t = A_t\ \mu_{t-1} + B_t\ u_t$

$\bar{\Sigma}_t = A_t\ \Sigma_{t-1}\ A_t^T + R_t$

$K_t = \bar{\Sigma}_t\ C_t^T (C_t\ \bar{\Sigma}_t\ C_t^T + Q_t)^{-1}$

$\mu_t = \bar{\mu}_t + K_t(z_t - C_t\ \bar{\mu}_t)$

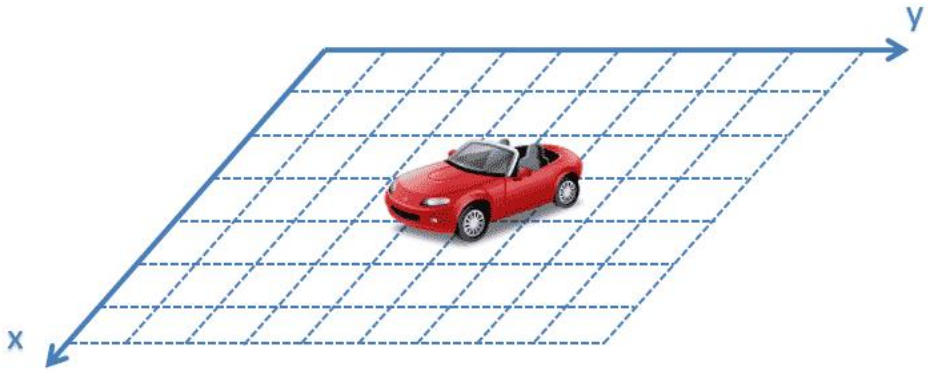
$\Sigma_t = (I - K_t\ C_t)\ \bar{\Sigma}_t$

return μ_t, Σ_t

Motion Model

- State Vector:

$$\mathbf{x_t} = \begin{bmatrix} position\_x \\ position\_y \\ velocity\_x \\ velocity\_y \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ \dot{x} \\ \dot{y} \end{bmatrix}$$



- Vehicle Motion Model:
  - No known control input (i.e., no pedaling or steering)
  - No known acceleration

$x_{t+1} = x_t + \dot{x}\,\Delta t$

$y_{t+1} = y_t + \dot{y}\,\Delta t$

$$\dot{x} = v = \frac{dx}{dt}$$

Kalman Filter – 1D Tracking

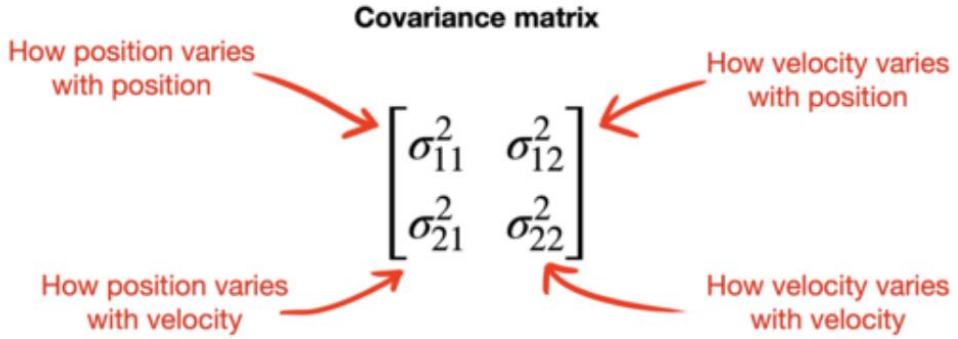
Scenario: 1D trajectory
Measurement: the position from GPS (may not accurate)
Unknown velocity; No control input

Implement a Kalman Filter to estimate the velocity and predict position

State vector

$$\hat{x} = \begin{bmatrix} position \\ velocity \end{bmatrix}$$

Covariance matrix



We assume the position and velocity are independent variables (we don't know their correlations), so ***SIGMA*** is initialized as a diagonal matrix.

$$\text{SIGMA}\_init = \begin{bmatrix} 1000 & 0 \\ 0 & 1000 \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$\mathbf{R} = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}$$ (may not optimal)

$$\boldsymbol{Q} = 0.01$$ (may not optimal)

Kalman Filter – 2D Tracking (homework)

Scenario: 2D trajectory in x and y directions;

Measurements: the positions in x and y directions from GPS (not quite accurate);
Unknown velocity

Task: Implement a Kalman filter in *kf2d_st.py,* used in *tracking2d.py* for position estimation.

$$\mathbf{x}_t = \begin{bmatrix} position\_x \\ position\_y \\ velocity\_x \\ velocity\_y \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

Hint: Matrices **A, SIGMA, C, Q, R ?**

Kalman Filter – Predict (not mandatory)

Scenario:

2D trajectory in both x and y directions;
measurement is the positions in x and y directions from GPS (not quite accurate);
unknown velocity
control input and acceleration need to be considered

Task: Implement the Kalman filter in *kf_predict_st.py*

- Implement a Kalman Filter that can estimate the car position
- Implement the *predict_light* function to predict whether the car should pass an intersection when the car camera captures the traffic light turning yellow. The car knows that the yellow light will last three seconds before turning to red, so the car needs to **predict its position for** 3 seconds later.
  - Return “False” to stop the car if predicted position < light position.
  - Return “True” to move forward if the predicted position > light position.

Kalman Filter – Predict (not mandatory)

- Suppose acceleration is allowed
  - set ALLOW_SPEED=true in *predict_light.py*
  - The car first check if it can pass the intersection with its current speed. If the current speed does not work, it is allowed to do reasonable acceleration, e.g., accelerate for 2 seconds with *acceleration=1.5*.

Task: Implement the predict function—*predict_light_speeding* in *kf_predict_st.py* —to predict whether the car should pass the intersection when the above speeding is allowed.