Time series analysis: Kalman Filter

ΕΠΛ 428: IOT PROGRAMMING

Dr. Panayiotis Kolios
Assistant Professor, Dept. Computer Science,
KIOS CoE for Intelligent Systems and Networks
Office: FST 01, 116
Telephone: +357 22893450 / 22892695
Web: https://www.kios.ucy.ac.cy/pkolios/

Πανεπιστήμιο Κύπρου

- Introduction to the Kalman Filter
- Assumptions and Model components
- The Kalman Filter algorithm
- Application to static and dynamic one-dimensional data
- Application to higher-dimensional data

```
# Imports
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import cv2
np.random.seed(0) # for reproducibility
```

- The Kalman Filter (KF) is an algorithm that processes time series measurements, containing statistical noise and other inaccuracies
- It produces estimates of unknown variables that tend to be more accurate than those based only on measurements
- Developed by Rudolf E. Kálmán in the late 1950s
- Applications:
  - tracking objects (Apollo project, GPS, self driving cars)
  - image processing
  - processes where model can be extracted from dynamics

- Dynamic systems are systems that change over time
- They are described by a set of equations that predict the future state of the system based on its current state
- The KF assumes the system is a linear dynamic system with Gaussian noise
- Gaussian distributions are used because of their nice properties when dealing with averages and variances

What are the ingredients?

- **State**
  - True value of the variables we want to estimate
  - State vector represents all the information needed to describe the current state of the system
- **Observation model**
  - Relates the current state to the measurements or observations
- **Noisy measurements**
  - Process and measurement noise (assumed to be Gaussian) represent the uncertainty in our models and measurements

- Example
    - Car moving at a constant velocity
- Model:
    - Car position = velocity × time × system noise
- Measurements:
    - Position and velocity (which are also noisy)

- The tradeoff between the influence of the model and the measurements is determined by noise
  - If the model has relatively large errors, more importance is given to the latest measurements in computing the current estimate
  - If the measurement have larger errors, more importance is given to the model in making the current estimate
- Therefore, you need to estimate not only your state but also the errors (the covariance) for both the model and the measurements
- Must be updated at each time step, too!

- KF is a recursive algorithm:
  - Uses information from previous time step to update the estimates

- Does not keep in memory all the data acquired so far

- Has predictor-corrector structure:
  1. Make a prediction based on the model
  2. Update the prediction with the measurements
  3. Repeat

- KF is based on three fundamental assumptions:
    1. The system can be described or approximated by a linear model
    2. All noise (from both the system and the measurements) is white, i.e., the values are not correlated
    3. All noise is Gaussian
- **Assumption 1: Linearity**
    - Each variable at the current time is a linear function of the variables at previous times
    - Many systems can be approximated this way
    - Linear systems are easy to analyze
    - Nonlinear systems can often be approximated by linear models around a current estimate (extended KF)
- **Assumption 2: Whiteness**
    - The noise values are not correlated in time
    - If you know the noise at time t, it doesn't help you to predict the noise at future times $t + \tau$
    - White noise is a reasonable approximation of the real noise
    - The assumption makes the mathematics tractable

- **Assumption 3: Gaussian Noise**

At any point in time, the probability density of the noise is a Gaussian

  - System and measurement noise are often a combination of many small sources of noise
  - The combination effect is approximately Gaussian
  - If only mean and variance are known (typical case in engineering systems), Gaussian distribution is a good choice as these two quantities completely determine the Gaussian distribution
  - Gaussian distribution have nice properties and are easy to treat mathematically

Car moving at a constant velocity (combining two sources)

- Assume the car has an initial position $x_0 = 0$ and initial velocity $\dot{x_0} = 60\text{km/h}$
- If the speed is constant, we have:

$$x_t = 1 \cdot x_{t-1} + \delta_t \cdot \dot{x}_{t-1}$$
$$\dot{x}_t = 0 \cdot x_{t-1} + 1 \cdot \dot{x}_{t-1}$$

- And in matrix form:

$$x_t = \begin{bmatrix} x_t \\ \dot{x}_t \end{bmatrix} = \begin{bmatrix} 1 & \delta_t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ \dot{x}_{t-1} \end{bmatrix} = A x_{t-1}$$

- New position and speed will evolve according to the linear dynamical system:

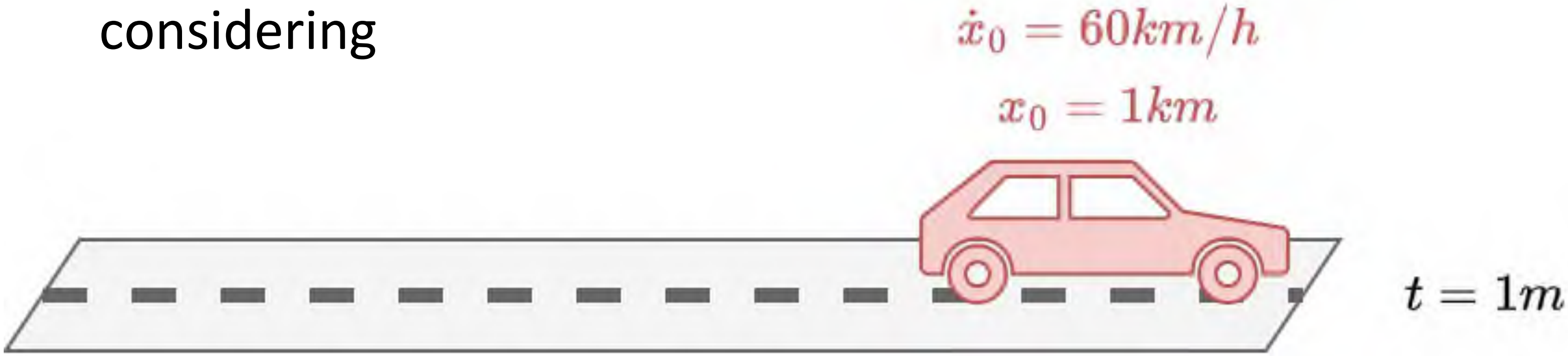
$$x_t = Ax_{t-1} + w_{t-1}$$

- where $w_{t-1} \sim N(0, Q)$, is the process noise (things that have not been modelled)
- $x_t \in \mathbb{R}^N$ represents the state of the system
- $Q \in \mathbb{R}^{N \times N}$ is the *process noise covariance*, where N represents the number of state variables (2 in our case, position and speed)

Πανεπιστήμιο Κύπρου

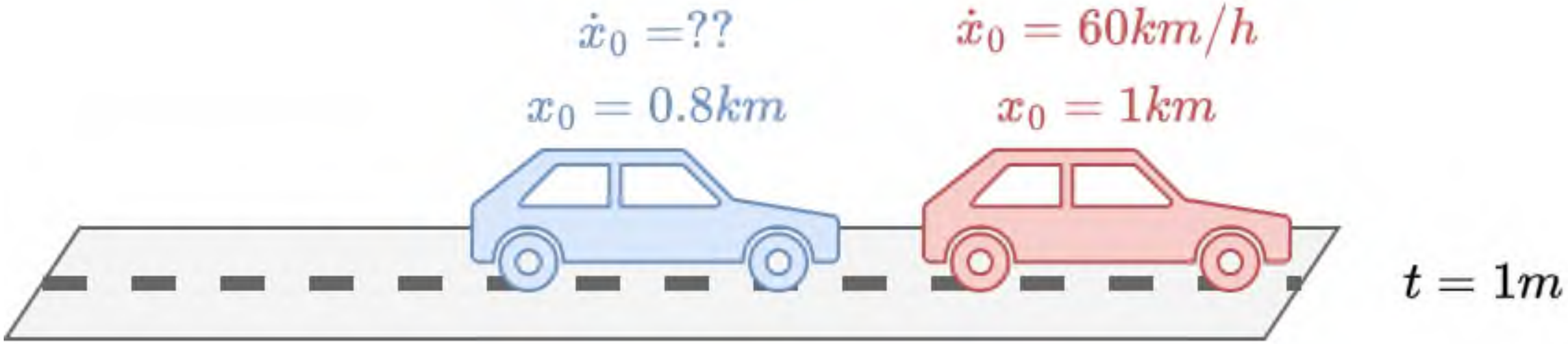
Where is the car when 1 minute passed?

- If we rely only on our simple system $x_{t+1} = Ax_t$ without even consider the noise, the car will keep moving at 60km/h and will be displaced 1km across the 1d direction we are considering



- Consider the GPS measurement of a car receiver
  - As all instruments, our GPS will be affected by errors and gives us measurements with uncertainties in it
  - In addition, our GPS can only measure the position but not the speed
  - Based on our GPS, after 1 minute we are at 0.8km from where we started

Πανεπιστήμιο Κύπρου



- We can model GPS measurement as follows:

$$z_t = Hx_t + v_t$$

- where $v_t \sim N(0, R)$, is uncertainty in the measurements
- $z_t \in \mathbb{R}^M$ is the measurement vector
- Since GPS measures only position, $H = [1\ 0]$ and $\mathrm{R} = [r_{xx}]$
- $H \in \mathbb{R}^{M,N}$ is a matrix that maps the N state variables into the M measurements and $\mathrm{R} \in \mathbb{R}^{M,M}$ is the measurement noise covariance

- Hence, we have two different sources of information:
  - a linear stochastic difference equation, representing our imprecise knowledge of a discrete-time controlled process; **model**
  - a source of measurements, that are noisy

$$x_t = Ax_{t-1} + w_{t-1}$$

$$z_t = Hx_t + v_t$$

- Combine them to get the best possible estimate of our system variables!

- KF is an iterative procedure that consists of two steps:
  - Predict (Time update)
  - Correct (Measurement update)
- These two steps are used to update two quantities:
  - State estimate
  - Uncertainty about our state estimate, called covariance estimate

- Covariance estimate
  - Two sources of error in the estimate of the state of our system
  - Prior estimate error $e_t^- = x_t - \hat{x}_t^-$
  - Posterior estimate error $e_t = x_t - \hat{x}_t$
- where
  - $\hat{x}_t^-$ is the estimate of the state based only on the knowledge of the system, e.g., dynamics model
  - $\hat{x}_t$ is the estimate based on the measurement z_t e.g. GPS
- Each type of error is associated with a covariance matrix, which reflects the amount of uncertainty in the state estimates
  - $P_t^- = \mathbb{E}\left[e_t^- {e_t^-}^T\right] \in \mathbb{R}^{N \times N}$(prior covariance estimate)
  - $P_t = \mathbb{E}[e_t e_t^T] \in \mathbb{R}^{N \times N}$(posterior covariance estimate)

Πανεπιστήμιο Κύπρου

COVARIANCE ESTIMATE FOR CAR EXAMPLE

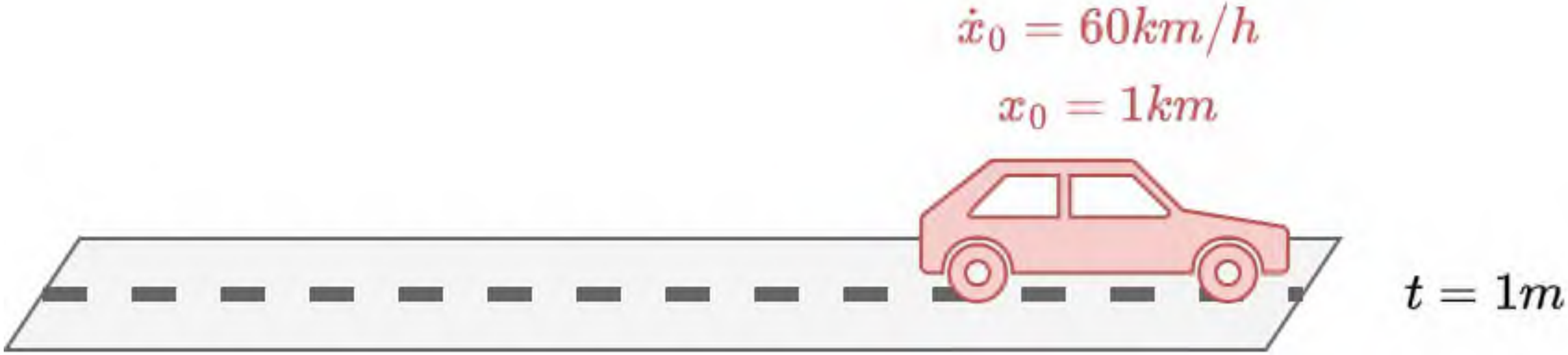
- $P_t = \begin{bmatrix} p_{xx} & p_{\dot{x}x} \\ p_{\dot{x}x} & p_{\dot{x}\dot{x}} \end{bmatrix}$, time t omitted on the matrix elements
  - p_{xx} uncertainty about the position
  - $p_{\dot{x}\dot{x}}$ uncertainty about the velocity
  - $p_{\dot{x}x}$ and $p_{\dot{x}x}$ represent the correlation between the noise in the measurements of the position and the speed

- P_t^- has the same form

COVARIANCE ESTIMATE FOR CAR EXAMPLE

- Predict
  - The dynamics equations are responsible for projecting forward (in time) the current state and error covariance estimates to obtain the prior estimate for the next time step
  - Such time update equations can also be thought of as predictor equations
- State estimate update:

$$\hat{x}_t^- = A\hat{x}_{t-1}$$



- Covariance estimate update:

$$P_t^- = \mathbb{E}\left[e_t^- {e_t^-}^T\right]$$

$$= \mathbb{E}[(x_t - \hat{x}_t^-)(x_t - \hat{x}_t^-)^T]$$

$$= \mathbb{E}\left[(Ax_{t-1} + w - A\hat{x}_{t-1})(Ax_{t-1} + w - A\hat{x}_{t-1})^T\right]$$

$$= \mathbb{E}\left[(Ae_{t-1})(A^T e_{t-1}^T)\right] + \mathbb{E}[ww^T]$$

$$= A\mathbb{E}\left[(e_{t-1}e_{t-1}^T)\right]A^T + \mathrm{Q}$$

$$= AP_{t-1}A^T + \mathrm{Q}$$

- Predict step consists of two updates:
  - State estimate $\hat{x}_t^- = A\hat{x}_{t-1}$
  - Covariance estimate $P_t^- = AP_{t-1}A^T + \mathrm{Q}$

Πανεπιστήμιο Κύπρου

Correction step

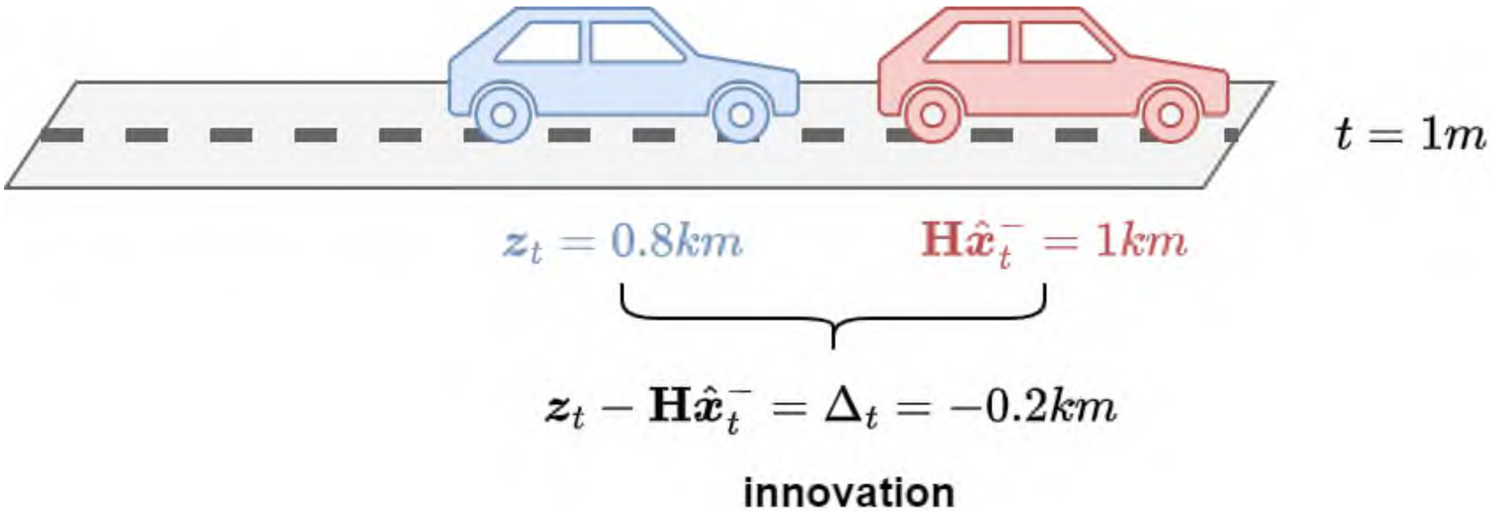
- The measurement update equations are responsible for the feedback, i.e. for incorporating a new measurement into the prior estimate to obtain an improved posterior estimate.
- The posterior estimate $\hat{x}_t$ is a linear combination of:
  - The prior estimate $\hat{x}_t^-$
  - A weighted difference between the actual measurement z_t and a measurement prediction $H\hat{x}_t^-$

$$\hat{x}_t = \hat{x}_t^- + K_t(z_t - H\hat{x}_t^-)$$

  - with the difference $\Delta_t = z_t - H\hat{x}_t^-$ is called measurement *innovation* or *residual*
  - Δ_t reflects the difference between our imprecise prediction $H\hat{x}_t^-$ and the actual noisy measurement z_t
  - Residual of zero means that the two are in complete agreement

Πανεπιστήμιο Κύπρου

KF EXECUTION STEPS



- Matrix $K_t \in \mathbb{R}^{N \times M}$ is chosen to be the **gain** that minimizes the posterior error covariance
- Execution steps:
  1. Consider the posterior error $e_t = x_t - \hat{x}_t$
  2. Consider the posterior error covariance $P_t = \mathbb{E}[e_t e_t^T]$
  3. Substitute value $\hat{x}_t = \hat{x}_t^- + K_t \Delta_t$in 1.
  4. Substitute the e_t value obtained from 3. in 2. and take the expectation
  5. Take the derivative of the trace of the result with respect to K_t
  6. Set the result equal to zero
  7. Solve for K_t

Πανεπιστήμιο Κύπρου

- Overall

$$K_t = P_t^- H^T (H P_t^- H^T + R)^{-1} = \frac{P_t^- H^T}{H P_t^- H^T + R}$$

- In the moving car example:

$$K_t = \frac{\begin{bmatrix} p_{xx}^- & p_{\dot{x}x}^- \\ p_{\dot{x}x}^- & p_{\dot{x}\dot{x}}^- \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}}{[1\ 0] \begin{bmatrix} p_{xx}^- & p_{\dot{x}x}^- \\ p_{\dot{x}x}^- & p_{\dot{x}\dot{x}}^- \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + [r_{xx}]} = \frac{\begin{bmatrix} p_{xx}^- \\ p_{\dot{x}x}^- \end{bmatrix}}{p_{xx}^- + r_{xx}}$$

- time index t has been omitted from matrix components for readability
- the update equations for the position and speed become
  - position: $\hat{x}_t = \hat{x}_t^- + \dfrac{p_{xx}^-}{p_{xx}^- + r_{xx}} \Delta_t$
  -
  - speed: $\hat{\dot{x}}_t = \hat{\dot{x}}_t^- + \dfrac{p_{x\dot{x}}^-}{p_{xx}^- + r_{xx}} \Delta_t$

Πανεπιστήμιο Κύπρου

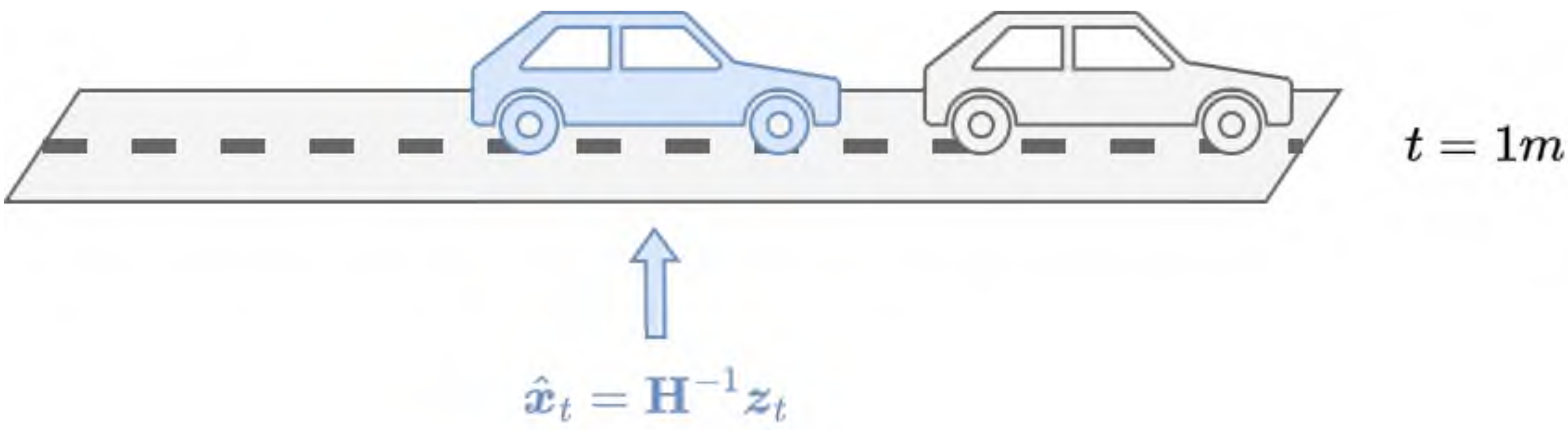
- One key observation is that the speed is updated even if we do not have a direct measure for it in our measurement z_t
- The update is possible thanks to the term $p^{-}_{\dot{x}x}$ in K_t that relates the position (and its measurement) to the velocity
- This showcases one of the main strengths of KF: it can handle partial observations
- Let’s consider the two extreme cases for the values of K_t

1. the measurement error covariance R approaches zero, i.e., there is no noise in the measurements
   - Measurements are completely reliable
   - We have $\lim_{R \to 0} K_t = \frac{P_t^- H^T}{H P_t^- H^T + R} = H^{-1}$
   - K_t weights the residuals more heavily:

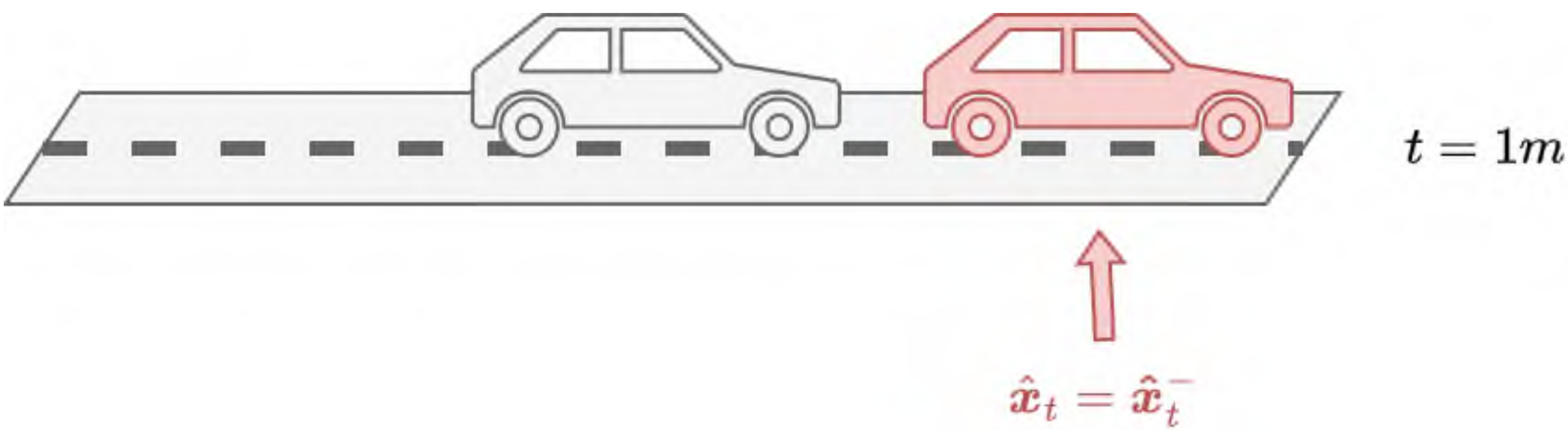
     $\hat{x}_t = \hat{x}_t^- + H^{-1}\Delta_t = \hat{x}_t^- + H^{-1}(z_t - H\hat{x}_t^-)$

     $= \hat{x}_t^- + H^{-1}z_t - H^{-1}H\hat{x}_t^- = \hat{x}_t^- + H^{-1}z_t - \hat{x}_t^-$

     $= H^{-1}z_t$



2. Covariance estimate P_t^- approaches zero
   - the model is completely reliable and the measurements are not accounted for
   - We have $\lim_{P_t^- \to 0} K_t = \frac{P_t^- H^T}{H P_t^- H^T + R} = 0$
   - K_t does not give importance to residuals $\hat{x}_t = \hat{x}_t^- + 0\Delta_t = \hat{x}_t^-$



- Final step is to update is the posterior covariance estimate to be used in the next predict step
- The formula is:

$$P_t = (I - K_t\mathrm{H})P_t^-$$

- Consider again the two extreme cases for the values of K_t
- $K_t = H^{-1} \rightarrow P_t = 0$
  - measurements are completely reliable
  - Only source of uncertainty at the next Predict step will be the one of the model: $P_t^- = AP_{t-1}A^T + \mathrm{Q} = \mathrm{Q}$
- $K_t = 0 \rightarrow P_t = P_t^-$
  - measurements are completely unreliable
  - Posterior covariance estimate same as prior covariance estimate obtained in the Predict step using the model and not modified with the new measurements in the Correct step

Πανεπιστήμιο Κύπρου

KF ALGORITHMIC PROCEDURE

Predict

$$\hat{\boldsymbol{x}}_t^- = \mathbf{A}\hat{\boldsymbol{x}}_{t-1}$$

$$\mathbf{P}_t^- = \mathbf{A}\mathbf{P}_{t-1}\mathbf{A}^T + \mathbf{Q}$$

Correct

$$\mathbf{K}_t = \mathbf{P}_t^-\mathbf{H}^T(\mathbf{H}\mathbf{P}_t^-\mathbf{H}^T + \mathbf{R})^{-1}$$

$$\hat{\boldsymbol{x}}_t = \hat{\boldsymbol{x}}_t^- + \mathbf{K}_t(\boldsymbol{z}_t - \mathbf{H}\hat{\boldsymbol{x}}_t^-)$$

$$\mathbf{P}_t = (\mathbf{I} - \mathbf{K}_t\mathbf{H})\mathbf{P}_t^-$$

Initial estimates for

$$\hat{\boldsymbol{x}}_{t-1}, \mathbf{P}_{t-1}$$

System

$$\boldsymbol{x}_t = \mathbf{A}\boldsymbol{x}_{t-1} + \boldsymbol{w}_{t-1}$$

$$\boldsymbol{z}_t = \mathbf{H}\boldsymbol{x}_t + \boldsymbol{v}_t$$

Πανεπιστήμιο Κύπρου

- In general, both the measurement noise covariance R and the process noise covariance Q are **unknown**
- Estimating R is usually done prior to operation of the filter
- Measuring R is usually possible because we measure the process anyway (while operating the filter)
- We can take some offline sample measurements to determine the variance of the measurement noise
- Determining the process noise covariance Q is generally more difficult
- We typically do not have the ability to directly observe the process we are estimating
- Sometimes a relatively simple process model can produce acceptable results if one “injects” enough uncertainty into the process
- This works well if the process measurements are reliable

- Extended Kalman Filter (EKF) is used for nonlinear systems by linearizing about the current estimate

- Ensemble Kalman Filter (EnKF) replaces the covariance matrix with the sample covariance to be used in problems with a large number of variables

- Particle Filter handles arbitrary noise distributions (other than Gaussian noise processes); computationally expensive

- Unscented Kalman Filter (UKF) uses a deterministic sampling technique to capture the mean and variance of the state distribution – Balances efficiency of Kalman and accuracy of particle filter

- Designing an accurate model and having reliable measurements are crucial for effective filtering
- Dealing with nonlinearities can be challenging and may require the EKF or UKF
- Tuning process and measurement noise covariances Q and R is essential for optimal performance
- Python implementation
    1. Make an initial estimate of your state vector and covariance matrix
    2. Predict the state and covariance for the next time step
    3. Computer the Kalman gain
    4. Make a measurement
    5. Update estimates of state and covariance
    6. Repeat from step 2.

- Python implementation for 1D examples
    1. Make an initial estimate of your state vector (**single number**) and covariance matrix (**reduces to variance**)
    2. Predict the state and covariance for the next time step
    3. Computer the Kalman gain
    4. Make a measurement
    5. Update estimates of state and covariance
    6. Repeat from step 2.
- No matrices involved thus normal multiplication

- Considering a simple example: determine the position of a stationary car
- 1D example so we need the distance along (a one-dimensional value) along a stretch of road from a landmark
- Need to predict the true position based on some noisy measurements of the distance
- We assume the measurements follow the distribution:

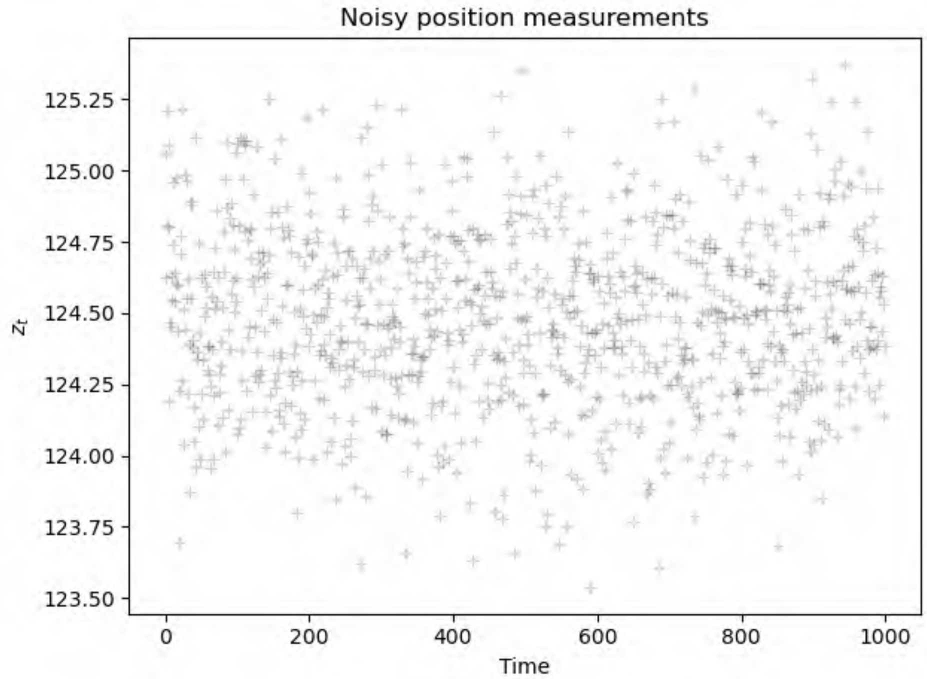
$$z_t = \mu + v_t \text{ with } v_t \sim N(0, R)$$

- where μ is the actual position and R is the measurement noise covariance
- these two values are unknown to the KF

Πανεπιστήμιο Κύπρου

- Generate the dataset

```
mu = 124.5 # Actual position
R = 0.1    # Actual standard deviation of actual measurements (R)

# Generate measurements
n_measurements = 1000 # Change the number of points to see how the convergence changes
Z = np.random.normal(mu, np.sqrt(R), size=n_measurements)

plt.plot(Z,'k+',label='measurements $z_t$',alpha=0.2)
plt.title('Noisy position measurements')
plt.xlabel('Time')
plt.ylabel('$z_t$')
plt.tight_layout();
```



- Since in reality the actual measurement noise covariance is unknown, we have to provide an estimate
- Estimate R from the available measurements
- In addition, we need to estimate the process noise covariance Q, which is usually harder to guess
- Remember that Q represents the noise of the model used to describe the actual position
- In our case:

$$x_t = x_0 + w_t \text{with } w_t \sim N(0, Q)$$

- We can see the effect of making a good or a bad estimate for R
- Also, we choose the actual position of our car, so we can see how quickly the KF converges to the true value
- Finally, we need to guess initial values for the initial position of the car and the variance in this initial estimate

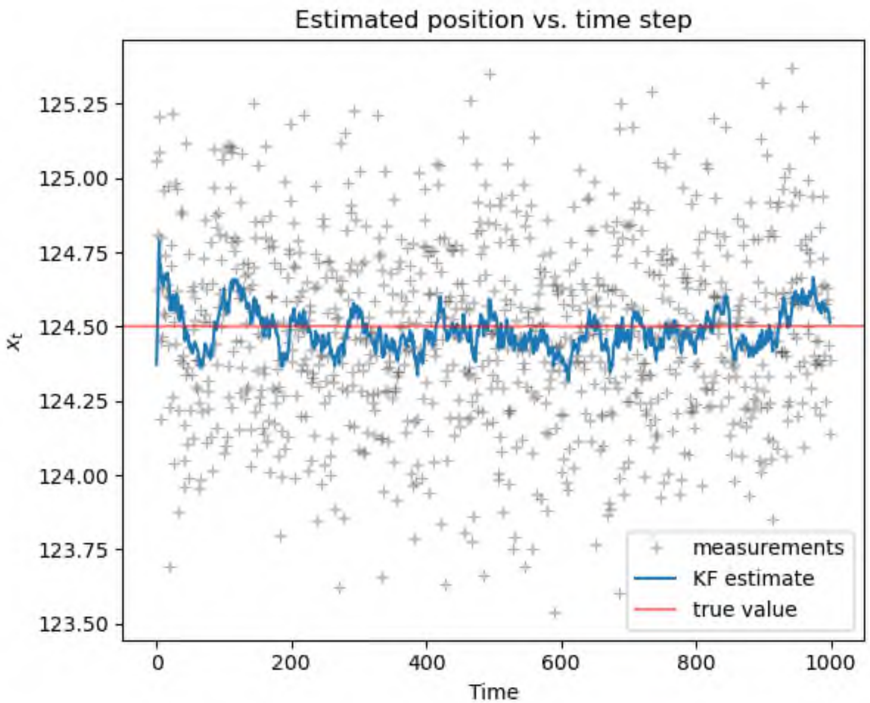
```
# Estimated covariances
Q_est = 1e-4
R_est = 2e-2
```

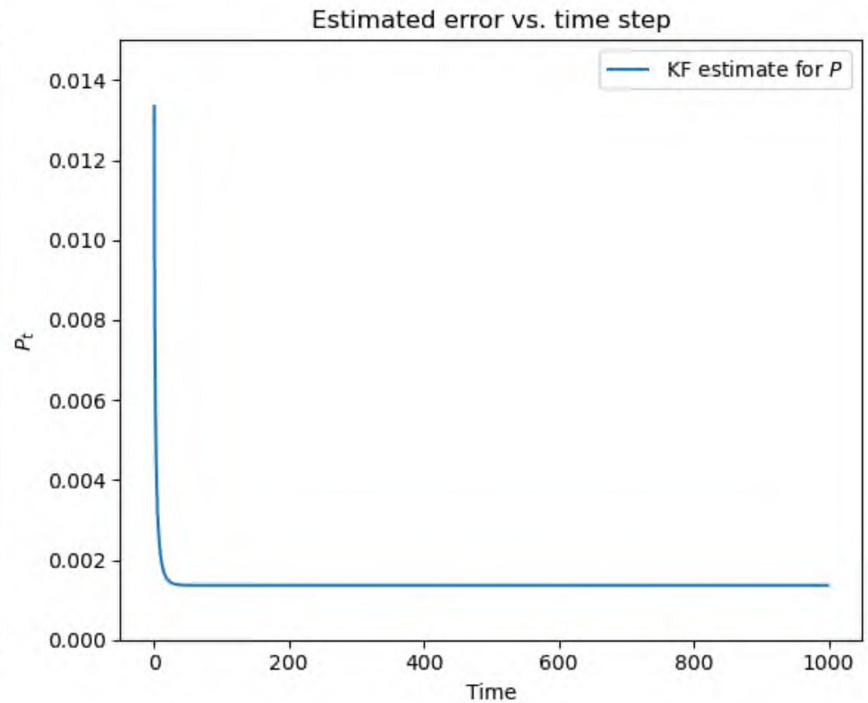
```
# Function to compute the estimated position and associated error
def kalman_1d(x, P, measurement, R_est, Q_est):
    # Prediction
    x_pred = x
    P_pred = P + Q_est
    # Update
    K = P_pred / (P_pred + R_est)
    x_est = x_pred + K * (measurement - x_pred)
    P_est = (1 - K) * P_pred

    return x_est, P_est
```

```
# Apply KF to the 1D measurements
# initial guesses
x = 123 # Use an integer (imagine the initial guess is determined with a meter stick)
P = 0.04 # error covariance P

KF_estimate=[] # To store the position estimate at each time point
KF_error=[] # To store estimated error at each time point
for z in Z:
    x, P = kalman_1d(x, P, z, R_est, Q_est)
    KF_estimate.append(x)
    KF_error.append(P)
```

```
# Function to plot estimates and measurements
def plot_1d_comparison(measurements_made, estimate, true_value, axis):
    axis.plot(measurements_made,'k+',label='measurements',alpha=0.3)
    axis.plot(estimate,'-',label='KF estimate')
    if not isinstance(true_value, (list, tuple, np.ndarray)):
        # plot line for a constant value
        axis.axhline(true_value,color='r',label='true value', alpha=0.5)
    else:
        # for a list, tuple or array, plot the points
        axis.plot(true_value,color='r',label='true value', alpha=0.5)
    axis.legend(loc = 'lower right')
    axis.set_title('Estimated position vs. time step')
    axis.set_xlabel('Time')
    axis.set_ylabel('$x_t$')

def plot_1d_error(estimated_error, lower_limit, upper_limit, axis):
    # lower_limit and upper_limit are the lower and upper limits of the
vertical axis
    axis.plot(estimated_error, label='KF estimate for $P$')
    axis.legend(loc = 'upper right')
    axis.set_title('Estimated error vs. time step')
    axis.set_xlabel('Time')
    axis.set_ylabel('$P_t$')
    plt.setp(axis,'ylim',[lower_limit, upper_limit])
```

```
fig, axes = plt.subplots(1,2, figsize=(12, 5))
plot_1d_comparison(Z, KF_estimate, mu, axes[0])
plot_1d_error(KF_error, 0, 0.015, axes[1])
plt.tight_layout();
```





- For the selected parameters we see that the filter converges to the true value quickly and the noise is filtered out
- Fluctuations around the true value are approximately the size of the standard deviation of the estimate, $\sqrt{P_t}$

Example 2: Dynamic one-dimensional data

- Considering another example: determine the position of a car moving at constant velocity v_0
- position changes over time according to the linear model:

$$x_t = x_{t-1} + \delta t \cdot v_0 + w_t \text{ with } w_t \sim N(0, Q)$$

- w_t is the model noise
- We assume to have measurements of position only:

$$z_t = z_t + \delta t \cdot v_t \text{ with } v_t \sim N(v_0, R)$$

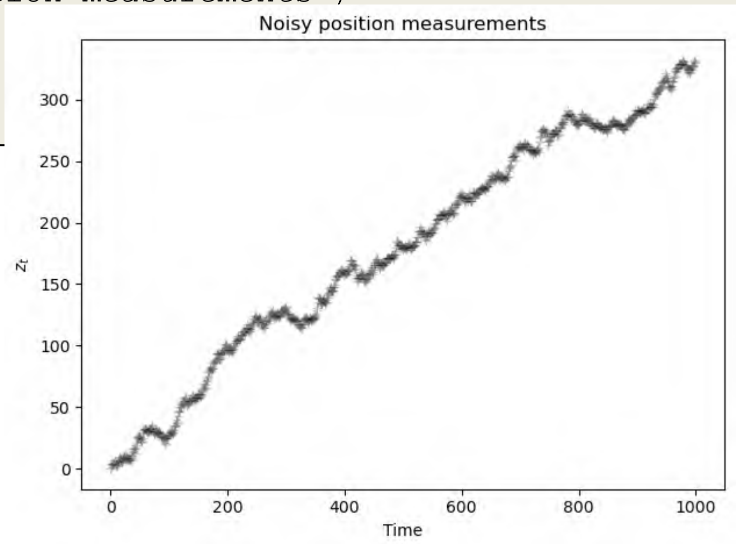
- where v_t is the noise measurement of velocity not observed directly and R is the variance of measurement errors

- At each time point, we will generate a random value v_t for the velocity measurement
- We will assume that the car moves at velocity v_t until the next time point, which allows us to calculate the distance traveled
- By summing up all of the distances traveled, we can calculate the measured position of the car z_t

- Then apply the KF and compare the KF estimate for the position with both the model position $x_t = x_0 + t \cdot v_0$ and the measured positions z_t

Example 2: Dynamic one-dimensional data

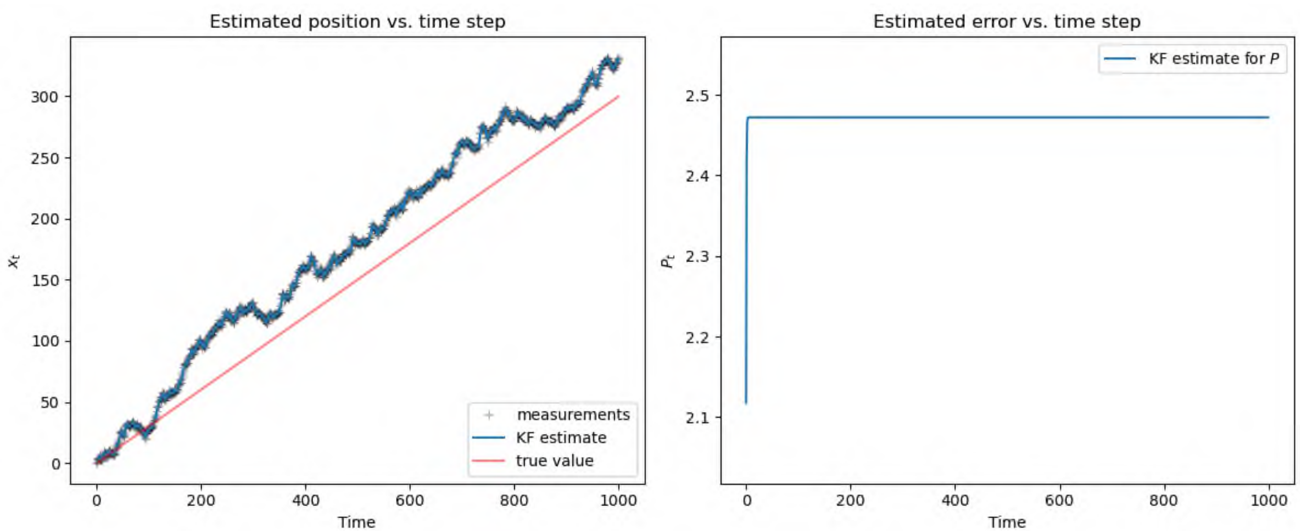
```
# initial parameters
v0 = 0.3
x0 = 0.0
R = 4.0

# generate noisy measurements
n_measurements = 1000
Zv = np.zeros(n_measurements) # velocity measurements
Zx = np.zeros(n_measurements) # position measurements
for t in range(0, n_measurements-1):
    Zv[t] = np.random.normal(v0, np.sqrt(R))
    Zx[t+1] = Zx[t] + Zv[t] * 1 # delta_t = 1

# generate true positions
Xt = np.zeros(n_measurements)
for t in range(0, n_measurements):
    Xt[t]= x0 + v0*t

plt.plot(Zx,'k+',label='measurements $z_t$',alpha=0.2)
plt.title('Noisy position measurements')
plt.xlabel('Time')
plt.ylabel('$z_t$')
plt.tight_layout();
```



Πανεπιστήμιο Κύπρου

Example 2: Dynamic One-Dimensional Data

```
# initial guesses and estimates
x = 0
P = 0.5
Q_est = 4
R_est = 4
KF_estimate = [] # To store the position estimate at each time point
KF_error = [] # To store estimated error at each time point

# Kalman filter
for z in Zx:
    x, P = kalman_1d(x, P, z, R_est, Q_est)
    KF_estimate.append(x)
    KF_error.append(P)

fig, axes = plt.subplots(1,2, figsize=(12, 5))
plot_1d_comparison(Zx, KF_estimate, Xt, axes[0])
plot_1d_error(KF_error, min(KF_error)-0.1, max(KF_error)+0.1, axes[1])
plt.tight_layout();
```



- The measurements are not close to the true value because the variance of the measurements error R is large compared to the velocity v_0
- The KF estimate is tracking the measurements, so it won't be close to the true value
- In addition, we do not have measurements for the instantaneous velocity
- With dynamic models, there are more parameters to tune, so it can be more challenging to reach convergence
- If we had a way to measure velocity, we could use that information too to improve the estimates
- Nevertheless, the KF can work with incomplete observations, which is one of its main advantages

Πανεπιστήμιο Κύπρου

- Now we consider an example that is closer to real-world applications: estimating the motion of a point in two dimensions, x and y

- Derive the algorithm to track objects
  - a mouse on a screen
  - objects in a video

- Use instantaneous measurements for the position (x, y)

- Going to use the Kalman filter built into the OpenCV library

Πανεπιστήμιο Κύπρου

- System is defined by the following quantities:
- State vector x
  representing the 2D position and velocity

$$x = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

- Measurement vector z
  for measurements of 2D position

$$z = \begin{bmatrix} z_x \\ z_y \end{bmatrix}$$

- Transition matrix A

$$A = \begin{bmatrix} 1 & 0 & \delta_t & 0 \\ 0 & 1 & 0 & \delta_t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Measurement matrix H

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

- Covariance of the model noise Q

$$Q = \begin{bmatrix} q & 0 & 0 & 0 \\ 0 & q & 0 & 0 \\ 0 & 0 & q & 0 \\ 0 & 0 & 0 & q \end{bmatrix}$$

- Covariance of the measurements noise R

$$R = \begin{bmatrix} r & 0 \\ 0 & r \end{bmatrix}$$

```
# KF implementation in OpenCV
KalmanFilter(state_size, measurements_size, control_size)
```

Πανεπιστήμιο Κύπρου

Example 3: Dynamic two-dimensional data

```
# KF implementation in OpenCV
KalmanFilter(state_size, measurements_size, control_size)
```

- state_size is the dimension of the state vector x, which is 4 in our case
- measurements_size is the dimension of the state vector x, which is 2 in our case
- control_size, which is 0 since we do not at this point do not control our car (u, not discussed in this lecture)

```
kalman = cv2.KalmanFilter(4,2,0) # 4 states, 2 measurements, 0 controls
q = 1 # the variance in the model
r = 20 # the variance in the measurement
dtime = 1 # size of time step
kalman.measurementMatrix = np.array([[1,0,0,0],
                                     [0,1,0,0]],np.float32) #  H
kalman.transitionMatrix = np.array([[1,0,dtime,0],
                                    [0,1,0,dtime],
                                    [0,0,1,0],
                                    [0,0,0,1]],np.float32) # A
kalman.processNoiseCov = np.array([[1,0,0,0],
                                   [0,1,0,0],
                                   [0,0,1,0],
                                   [0,0,0,1]],np.float32) * q # Q
kalman.measurementNoiseCov = np.array([[1,0],
                                       [0,1]],np.float32) * r # R
KF_estimate_xy = [] # To store the position estimate at each time point
```

Example 3: Dynamic two-dimensional data

```
# Load some pre-computed data
xy_motion =
pd.read_csv('https://zenodo.org/records/10951538/files/kf_ts1.csv?download=1',
                              header = None).values.astype('float32')
```

```
for i in xy_motion:
    pred = kalman.predict()   # predicts new state using the model
    kalman.correct((i))       # updates estimated state with the measurement
    KF_estimate_xy.append(((pred[0]),(pred[1]))) # store the estimated
position
```

```
x_est, y_est = zip(*KF_estimate_xy)
x_true, y_true = zip(*xy_motion)
plt.scatter(x_est, y_est, marker= '.', label = 'KF estimate', alpha = 0.5)
plt.scatter(x_true, y_true,marker= '.', label = 'true value', alpha = 0.5)
plt.legend(loc = 'lower center')
plt.title('2D position')
plt.xlabel('x coordinate')
plt.ylabel('y coordinate')
plt.tight_layout();
```