Digital Signal Filtering

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it

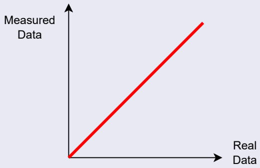
Robotic Systems

Measures and Errors

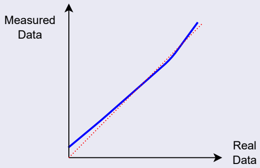
- Measure of state variables is a fundamental aspect of control systems
- If the measure is not precise or affected by a significant amount of noise, the whole control system cannot work properly
- However **any measurement** system is always affected by **errors**
- **Measurement errors** have different characteristics and depends on the kind of sensor used

Sensor Characteristics

- The behaviour of a sensor is in general represented by its **characteristic curve**
- It is plot over a *XY* chart that reports in the *X* axis the **real data** and in the *Y* axis the **measured data**
- For an **ideal** sensor, the characteristic is a 45-degrees straight line
- But for a **real** sensor, the characteristic is a **curve** the is **close** to the 45-degrees straight line



Ideal Sensor

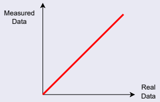


Real Sensor

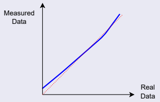
Sensor Errors

- **Offset:** it is the **non-zero value** given by the sensor when the real data is zero
- **Non-linearity:** it is the difference between the ideal and real characteristic
- **Noise:** it the variation of the measured data when the real data is **constant**

- Offset and Non-linearity can be reduced by means of **sensor calibration**
- Noise (that is harder to be removed) can be reduced by means of **digital filters**



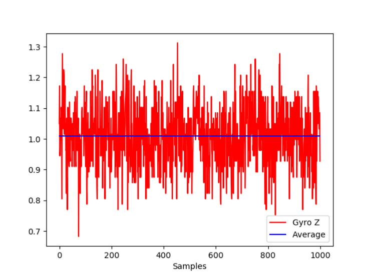
Ideal Sensor



Real Sensor

Noise Errors

- Here is a plot of the data sampled by a gyroscope, Z-axis, when the system is **stopped**

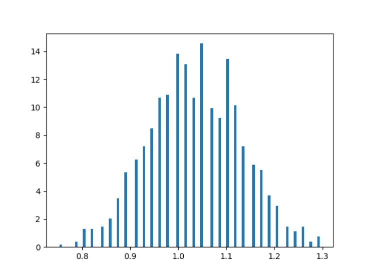


- The **real data** should be **zero**, but we have an **offset** and a certain amount of **noise**
- The **blue line** is the **average**, if we subtract it, we can remove the offset but not the noise

Measures and Errors

Noise Characteristics

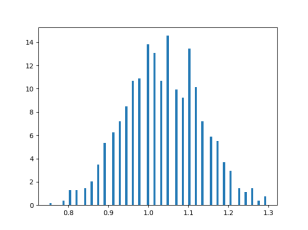
- If we plot the **histogram** of sampled data, we obtain the following chart



- Here we have, on *X* axis, the values of a sampled data, and, on *Y* axis, the number of times that value is got (data is organized in subintervals)
- The plot is the classical **Gaussian Curve** or **Normal Distribution** that is a common characteristic of noise errors

Noise Characteristics

- The plot is the classical **Gaussian Curve** or **Normal Distribution** that is a common characteristic of noise errors



- This curve is characterised by the **average** and the **variance**
- Given that we have N measures, and given x_i the measures, we have:

$$\overline{x} = \frac{1}{N}\sum_i x_i$$

$$\sigma_x^2 = \frac{1}{N}\sum_i (x_i - \overline{x})^2$$

Noise Filters

A Noisy Process

Robot over a XY plane

Let us consider a robot moving over a *XY* plane with a straight trajectory and using a constant speed:

$$x(k+1) = x(k) + v_x \, \Delta T$$
$$y(k+1) = y(k) + v_y \, \Delta T$$

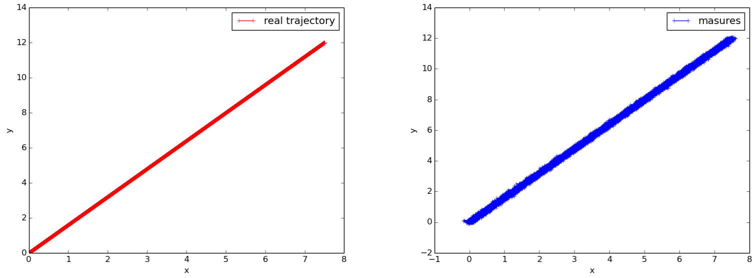
with v_x, v_y constants

Let us consider a position sensor the however has a gaussian noise:

$$\hat{x}(k) = x(k) + \sigma_x$$
$$\hat{y}(k) = y(k) + \sigma_y$$

- $\hat{x}(k)$, $\hat{y}(k)$: measured values
- σ_x, σ_y: random gaussian variables with zero average

Ideal and Measured Trajectory



(see **`filtering/noise_plot.py`**)

Digital Filters

Digital Filters

- A digital filter is discrete dynamic system characterised by the following relation:

$$y(k) + a_1 y(k-1) + a_2 y(k-2) + \cdots + a_m y(k-m) =$$
$$= b_0 u(k) + b_1 u(k-1) + \cdots + b_n u(k-n)$$

$$y(k) = -a_1 y(k-1) - a_2 y(k-2) + \cdots - a_m y(k-m) +$$
$$+ b_0 u(k) + b_1 u(k-1) + \cdots + b_n u(k-n)$$

- $u(k)$, input
- $y(k)$, output
- $max(n, m)$ **filter order**
- Coefficients $a_i, b_i \in [0, 1] \subset \mathcal{R}$ determine the **filter type** and the **noise cut capability**
- The filter behaves as a **weighted average** of past inputs and outputs

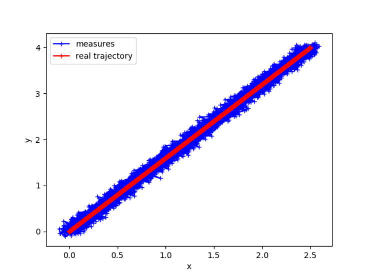
Order-1 Digital Filter

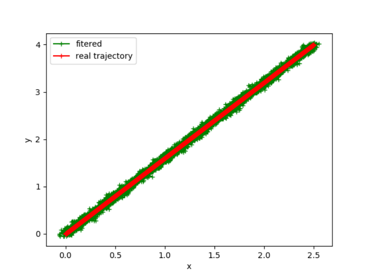
$$
\begin{aligned}
y(k) &= (1-\alpha)y(k-1) + \alpha u(k) \\
\alpha &\in [0, 1]
\end{aligned}
$$

- if α is close to 0, the new sampled data (input) is weighed **less** than the previous output: high filtering capability but measured values are propagated slowly
- if α is close to 1, the new sampled data (input) is weighed **more** than the previous output: low filtering capability but measured values are propagated fast

Ideal, Measured and Filtered Trajectory

$\alpha = 0.5$

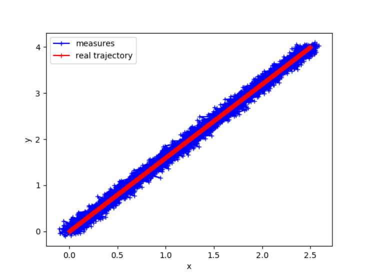


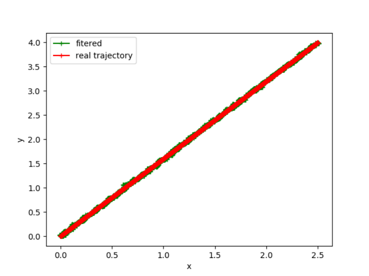


(see **`filtering/first-order-filter.py`**)

Ideal, Measured and Filtered Trajectory

$\alpha = 0.1$

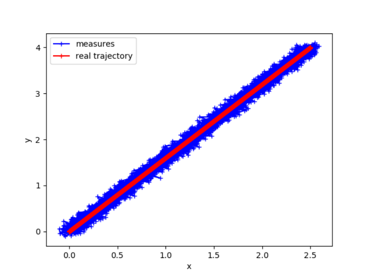


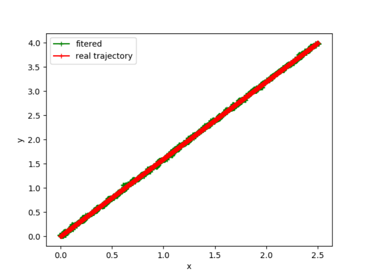


(see `filtering/first-order-filter.py`)

Ideal, Measured and Filtered Trajectory

$\alpha = 0.05$



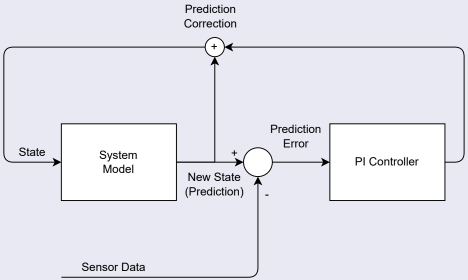


(see **`filtering/first-order-filter.py`**)

Prediction and Adaptation

Predictive Filters

- **Predictive filters** are bsaed on a model of the behaviour of the system
- By means of the model, an estimate of the state variable is performed
- The estimate is compared with sensor data and an **error** is computed
- The error is then passed to a **PI controller** and the output is **(agebrically) added** to the estimate



Kalman Filters

- Predictive filter are able to reduce measurement error on the basis of the knowledge of the system model
- But they require the tuning of controller constants K_P e K_I
- **Kalman Filters** are predictive filters with a P-controller whose K_P is updated at each iteration
- K_P is computed by using an approach that has the objective on **minimising the error**
- The approach is based on the statistic characterisation of the system model and the measure

System Model

The system is considered characterised by the discretized model

$$x(k+1) \quad = \quad A\,x(k) + w$$

where

- x is the state vector
- A is the state matrix
- w is a vector representing the uncertainty of the behaviour of the system

Measure Model

Sensors are considered characterised by the following relation

$$z(k) = H\,x(k) + v$$

where

- z is the vector of the measured values
- H is a matrix that specifies the state variables measured
- v is a vector representing the uncertainty (noise) of the behaviour of the measure

Statistic Characterisation

- In the previous models w and v are random variables that se suppose gaussian and with zero average
- The Kalman filter needs the knowledge of the statistics of w and v
- In details, it is necessary to have the variance and **covariance** of each element of w and v
- In other words, the self and mutual variance of the system and measure for each of the state variables

Variance

Given a random variable with $x_1, x_2, ..., x_n$ a set of samples and $\overline{x}$ the mean, the **variance** is:

$$\sigma_x^2 = \frac{1}{N}\sum_i (x_i - \overline{x})^2$$

Covariance

Given two random variables with $x_1, x_2, ..., x_n$ and $y_1, y_2, ..., y_n$ the set of samples and $\overline{x}$, $\overline{y}$ the means, the **covariance** is:

$$\sigma_{xy}^2 = \frac{1}{N}\sum_i (x_i - \overline{x})(y_i - \overline{y})$$

if the two variables are **statistically independent**, then the covariance **is close to 0**

Covariance Matrix

The Kalman Filter needs two matrices

- Q is the covariance matrix of the uncertainty of the system w
- R is the covariance matrix of the uncertainty of the measure v

In these matrices

- element (i, i) is the variance of the i^{th} variable
- element (i, j) is the covariance of the i^{th} and j^{th} variables

Kalman Filter Algorithm

Let

- x the state vector **real** (that is unknown)
- $\hat{x}$ the **estimated** state vector

The Kalman filter algorithm works as follows

1. $\hat{x} = A\,\hat{x}$ new estimate
2. $E = z - H\hat{x}$ error computation with respect to the estimate
3. $K = ...$ computation of the **optimal gain (K_P)**
4. $\hat{x} = \hat{x} + K\;E$ estimate correction

Step 3 has the objective of **minimising the error** between the real and predicted state:

$$min\{\hat{x}(k) - x(k)\}$$

but x is not known and we have only its statistical characterisation w and Q

The Optimal Gain

Starting form covariance matrix Q, the **error covariance** P is determined, then the algorithm computes the K such that P is minimised

1. $P = A\,P\,A^T + Q$
   Error covariance estimate
2. $K = P\,H^T\,(H\,P\,H^T + R)^{-1}$
   optimal gain
3. $P = (I - K\,H)\,P$
   Update of the error covariance on the basis of the optimal gain

Kalman Filter

Complete Algorithm

1	$\hat{x} = A\,\hat{x}$	Prediction
2	$P = A\,P\,A^T + Q$	Update of Error Covariance
3	$K = P\,H^T\,(H\,P\,H^T + R)^{-1}$	Gain
4	$\hat{x} = \hat{x} + K(z - H\hat{x})$	Measure Correction
5	$P = (I - K\ H)\ P$	Error Covariance Correction

An Example: Robot over a XY plane

Let us consider a robot moving over a *XY* plane with a straight trajectory and using a constant speed:

$$x(k+1) = x(k) + v_x \, \Delta T$$
$$y(k+1) = y(k) + v_y \, \Delta T$$

with v_x, v_y constants

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

State matrix:

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

An Example: Robot over a XY plane

Let us consider a robot moving over a XY plane with a straight trajectory and using a constant speed:

$$x(k+1) = x(k) + v_x\,\Delta T$$
$$y(k+1) = y(k) + v_y\,\Delta T$$

with v_x, v_y constants

State Equation:

$$\begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta T & 0 \\ 0 & 1 & 0 & \Delta T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix}$$

Implementation

```
class KalmanFilter:

    def __init__(self, delta_t):
        # state vector x y vx vy, initial state to 0
        self.x = np.matrix( [ 0, 0, 0, 0 ]).transpose()
        # process matrix
        self.A = np.matrix( [ [1, 0, delta_t, 0      ],
                              [0, 1, 0      , delta_t],
                              [0, 0, 1      , 0      ],
                              [0, 0, 0      , 1      ] ])

        # process covariance
        self.Q = np.eye(4,4) * 0.05

        # measure covariance (initially high)
        self.R = np.eye(4,4) * 1000

        # measure matrix (only x and y masured)
        self.H = np.matrix( [ [1, 0, 0, 0],
                              [0, 1, 0, 0],
                              [0, 0, 0, 0],
                              [0, 0, 0, 0] ])
        # error covariance matrix
        self.P = np.matrix( [ [0, 0, 0, 0],
                              [0, 0, 0, 0],
                              [0, 0, 0, 0],
                              [0, 0, 0, 0] ])
```

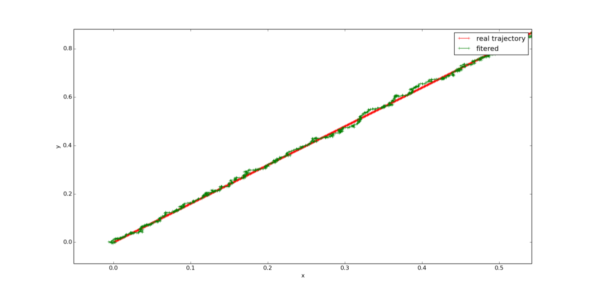
Implementation

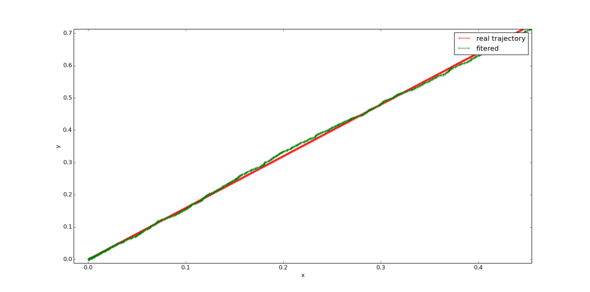
```
class KalmanFilter:
    ...
    def prediction(self):
        self.x = self.A * self.x
        self.P = self.A * self.P * self.A.transpose() + self.Q
        S = self.H * self.P * self.H.transpose() + self.R
        self.K = (self.P * self.H.transpose()) * S.I


    def measure(self, measures):
        measures = np.matrix(measures).transpose()
        self.x = self.x + self.K * (measures - self.H * self.x)


    def update(self):
        self.P = (np.eye(4,4) - self.K * self.H) * self.P
```

Low-pass vs. Kalman





Digital Signal Filtering

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it

Robotic Systems