

## Autonomous Robots Lab

HOME NEWS RESEARCH

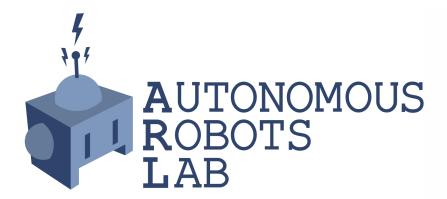
PUBLICATIONS

GROUP

**EDUCATION** 

RESOURCES

CONTACT





# The Kalman Filter

Basic Introduction to Kalman Filtering. The basic Kalman Filter structure is explained and accompanied with a simple python implementation.

# **Kalman Filter Basic Intro**

## Introduction

The Kalman Filter (KF) is a set of mathematical equations that when operating together implement a **predictor-corrector** type of estimator that

## Python Implementation

File: KalmanFilter\_Basic.py

```
from numpy import *
import numpy as np
from numpy.linalg import inv
from KalmanFilterFunctions import *
```

is optimal in the sense that it minimizes the estimated error covariance when some presumed conditions are met.

## Mathematical Formulation

The KF addresses the general problem of trying to estimate the state  $x \in \mathcal{R}^n$  of a discrete-time controlled process that is governed by the linear stochastic difference equation:

$$x_k = Ax_{k-1} + Bu_k + w_{k-1}$$
 with a measurement  $y \in \mathcal{R}^m$  that is:

$$y_k = Hx_k + v_k$$

The random variables  $w_k$  and  $v_k$  represent the process and measurement noise respectively. They are assumed to be **independent** of each other, white, and with normal probability distributions:

$$p(w) pprox N(0,Q)$$
  
 $p(v) pprox N(0,R)$ 

The  $n\times n$  matrix A relates the state at the previous time step to the state at the current step, in the absence of either a driving input or process noise. The  $n\times l$  matrix B relates the control input  $u\in \mathcal{R}^l$  to the state x. The  $m\times n$  matrix H in the measurement equation relates the state to the measurement  $y_k$ .

## How the KF works

The KF process has two steps, namely:

- \* **Prediction step:** the next step state of the system is predicted given the previous measurements
- \* **Update step:** the current state of the system is estimated given the measurement at that time step

These steps are expressed in equation-form as follows:

```
# time step of mobile movement
dt = 0.1
# Initialization of state matrices
X = array([[0.0], [0.0], [0.1], [0.1]))
P = diag((0.01, 0.01, 0.01, 0.01))
A = array([[1, 0, dt, 0], [0, 1, 0, dt], [0, 0])
Q = eye(X.shape[0])
B = eye(X.shape[0])
U = zeros((X.shape[0], 1))
# Measurement matrices
Y = array([[X[0,0] + abs(random.randn(1)[0])],
H = array([[1, 0, 0, 0], [0, 1, 0, 0]])
R = eye(Y.shape[0])
# Number of iterations in Kalman Filter
N iter = 50
# Applying the Kalman Filter
for i in range(0, N iter):
    (X, P) = kf predict(X, P, A, Q, B, U)
    (X, P, K, IM, IS, LH) = kf update(X, P, Y)
    Y = array([[X[0,0] + abs(0.1 * random.random.random])])
```

File: KalmanFilterFunctions.py

```
from numpy import dot, sum, tile, linalg, log,
from numpy.linalg import inv, det

def kf_predict(X, P, A, Q, B, U):
    X = dot(A, X) + dot(B, U)
    P = dot(A, dot(P, A.T)) + Q
    return(X, P)

def gauss_pdf(X, M, S):
    if M.shape[1] == 1:
        DX = X - tile(M, X.shape[1])
        E = 0.5 * sum(DX * (dot(inv(S), DX)), a
```

#### **Prediction**

$$\begin{split} X_k{}^- &= A_{k-1} X_{k-1} + B_k U_k \\ P_k{}^- &= A_{k-1} P_{k-1} A_{k-1}^T + Q_{k-1} \end{split}$$

### **Update**

$$egin{aligned} V_k &= Y_k - H_k - X_k^- \ S_k &= H_k P_k^- H_k^T + R_k \ K_k &= P_k^- H_k^T S_k^- 1 \ X_k &= X_k^- + K_k V_k \ P_k &= P_k^- - K_k S_k K_k^T \end{aligned}$$

where:

- \*  $X_k^-$  and  $P_k^-$  are the predicted mean and covariance of the state, respectively, on the time step k before seeing the measurement.
- \*  $X_k$  and  $P_k$  are the estimated mean and covariance of the state, respectively, on time step k after seeing the measurement.
- \*  $Y_k$  is the mean of the measurement on time step k.
- \*  $V_k$  is the innovation or the measurement residual on time step k.
- \*  $S_k$  is the measurement prediction covariance on the time step k.
- \*  $K_k$  is the filter gain, which tells how much the predictions should be corrected on time step k.

```
E - E + 0.5 * M.shape[0] * log(2 * pi)
        P = \exp(-E)
   elif X.shape[1] == 1:
        DX = tile(X, M.shape[1] - M)
        E = 0.5 * sum(DX * (dot(inv(S), DX)),
        E = E + 0.5 * M.shape[0] * log(2 * pi)
        P = \exp(-E)
    else:
        DX = X - M
        E = 0.5 * dot(DX.T, dot(inv(S), DX))
       E = E + 0.5 * M.shape[0] * log(2 * pi)
        P = \exp(-E)
    return (P[0],E[0])
def kf update(X, P, Y, H, R):
   IM = dot(H, X)
   IS = R + dot(H, dot(P, H.T))
   K = dot(P, dot(H.T, inv(IS)))
   X = X + dot(K, (Y-IM))
    P = P - dot(K, dot(IS, K.T))
   LH = gauss pdf(Y, IM, IS)
    return (X,P,K,IM,IS,LH)
```

Proudly powered by Weebly