A GLIMPSE OF KALMAN FILTER

A PREPRINT

Jiachen Wang
SIST 2018533087
wangjch2@shanghaitech.edu.cn

April 16, 2021

ABSTRACT

State-space model has wide-application in the area of information science, which encompasses three components: transition probability, emission probability and initial distribution. For such problem, we have seen how to finish the inference and learning task. In this passage, I gonna introduce another particular type of task, namely filtering, while also give a specific type of algorithm for linear Gaussian models called "Kalman Filter". We first discuss the algorithm and then provide a demo of using kalman filter for object detection.

Keywords Kalman Filter · Object Detection

1 Introduction

Within the significant toolbox of mathematical tools that can be used for stochastic estimation from noisy sensor measurements, one of the most well-known and oftenused tools is what is known as the Kalman filter. The Kalman filter is named after Rudolph E. Kalman, who in 1960 published his famous paper describing a recursive solution to the discrete-data linear filtering problem Kalman [1960]. The Kalman filter is essentially a set of mathematical equations that implement a predictor-corrector type estimator that is optimal in the sense that it minimizes the estimated error covariance when some presumed conditions are met. Since the time of its introduction, the Kalman filter has been the subject of extensive research and application, particularly in the area of autonomous or assisted navigation. This is likely due in large part to advances in digital computing that made the use of the filter practical, but also to the relative simplicity and robust nature of the filter itself.

2 Problem Formulation

The Kalman filter aims to estimate the state $x \in R^n$ of a discrete-time contolled process which is described with a linear stochastic difference equation:

$$x_k = Ax_{k-1} + w_{k-1} \tag{1}$$

with a measurement $z \in \mathfrak{R}^m$ that is

$$z_k = Hx_k + v_k$$

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

$$p(w) \sim N(0, Q)$$
$$p(v) \sim N(0, R)$$

In practice, the process noise covariance Q and measurement noise covariance R matrices might change with each time step or measurement, however here we assume they are constant.

We can see that A, H, Q, R are all given parameters and our goal is to have an "robust" prediction for x_k at time step k based on historical knowledge of the system (things we learned from time 1 to $k-1$) and correct it after seeing the observation z_k. Kalman filter is powerful but is defined for linear models, as for non-linear models, another type of filter (particle filter) is preferred.

3 Method

The kalman filter has two processes:

1. Predict: give an (priori) estimation of x_k as $\hat{x_k}'$ based on historical knowledge
2. Update (Correct): give the corrected estimation (posteriori) of x_k as $\hat{x_k}$ based on observation z_k

3.1 Update(Correction)

If we define the co-variance P_k as the MSE error between posteriori estimation and true state value:

$$P_k = E\left[e_k e_k^T\right] = E\left[\left(x_k - \hat{x}_k\right)\left(x_k - \hat{x}_k\right)^T\right] \tag{2}$$

Then our goal is to minimize P_k so that our prediction is "robust". Also, assume the update $\hat{x_k}$ is a combination of priori estimation $\hat{x_k'}$ and the *measurement residual*:

$$\hat{x}_k = \hat{x}_k' + K_k\left(z_k - H\hat{x}_k'\right) \tag{3}$$

where K_k is the kalman gain or blending factor that minimizes the a posteriori error covariance equation.

Substitute the above definition into P_k we have:

$$\begin{aligned} P_k = E\left[\left[\left(I - K_k H\right)\left(x_k - \hat{x}_k'\right) - K_k v_k\right]\right. \\ \left.\left[\left(I - K_k H\right)\left(x_k - \hat{x}_k'\right) - K_k v_k\right]^T\right] \end{aligned} \tag{4}$$

Expanding P_k and set its derivatives of K_k to 0, we managed to compute Kalman Gain K_k:

$$K_k = P_k' H^T\left(H P_k' H^T + R\right)^{-1} \tag{5}$$

And subsisting it back to the expansion of P_K we managed to compute P_k:

$$\begin{aligned} P_k &= P_k' - P_k' H^T\left(H P_k' H^T + R\right)^{-1} H P_k' \\ &= P_k' - K_k H P_k' \\ &= \left(I - K_k H\right) P_k' \end{aligned} \tag{6}$$

3.2 Prediction

After having a posteriori estimation of x_k as $\hat{x_k}$, we should project it to state x_{k+1}, and this is done naturally with the state transition:

$$\hat{x}_{k+1}' = H\hat{x}_k \tag{7}$$

Also, we project the error covariance matrix into state x_{k+1}:

$$\begin{aligned} e_{k+1}' &= x_{k+1} - \hat{x}_{k+1}' \\ &= \left(H x_k + w_k\right) - H\hat{x}_k \\ &= H e_k + w_k \end{aligned} \tag{8}$$

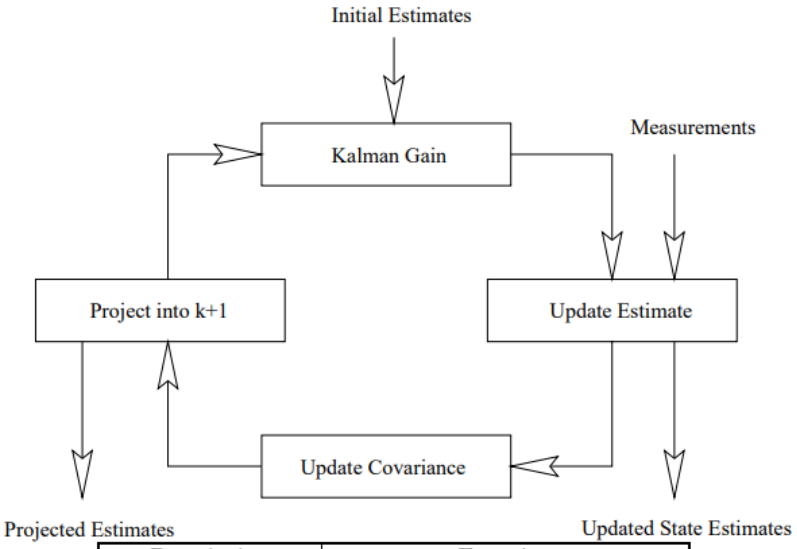
and

$$\begin{aligned} P'_{k+1} &= E\left[e'_{k+1}e'^{T}_{k+1}\right] \\ &= E\left[He_k\left(He_k\right)^T\right] + E\left[w_k w_k^T\right] \\ &= HP_kH^T + Q \end{aligned} \tag{9}$$

The last step comes from the fact that the cross-correlation between e_k and w_k is 0. This is because e_k is the error until time k and w_k is the noise introduced at time step $k+1$.

3.3 A milestone

Till now, we have the formal definition of the kalman filter.



Description	Equation
Kalman Gain	$K_k = P'_k H^T \left(HP'_k H^T + R\right)^{-1}$
Update Estimate	$\hat{x}_k = \hat{x}'_k + K_k\left(z_k - H\hat{x}'_k\right)$
Update Covariance	$P_k = \left(I - K_k H\right)P'_k$
Project into $k+1$	$\hat{x}'_{k+1} = \Phi\hat{x}_k$ $P_{k+1} = \Phi P_k \Phi^T + Q$

4 Demo: Object Detection with Kalman Filter

4.1 Implementing Kalman filter with Python

After understanding the mathematical foundation of Kalman filter, we can convert the above procedure into the python code.

We first initialize a kalman filter with the necessary parameters that describe a linear Gaussian model.

```
def __init__(self, state_estimateX, covariance_P,  transition_F, control_B,

noise_covariance_Q, measurement_Z, obs_model_H, obs_noise_covariance_R):
    """
     Initialize the filter manually to set up different parameters
    """
    self.state_estimateX = state_estimateX
    self.covariance_P = covariance_P
```

```
        self.transition_F = transition_F
        self.control_B = control_B

        self.noise_covariance_Q = noise_covariance_Q
        self.measurement_Z = measurement_Z
        self.obs_model_H = obs_model_H
        self.obs_noise_covariance_R = obs_noise_covariance_R
```

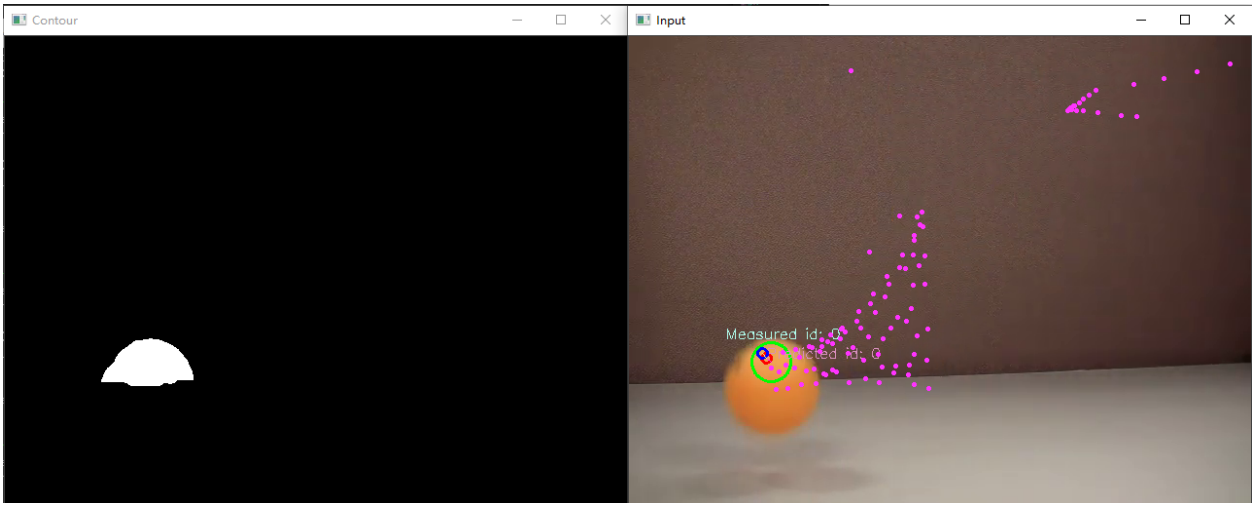
Then we wrap the predict and update procedure into two seperate functions.

```
    def predict(self):
        """
        predicts the future state

        gets the predicted state estimate
        """
        #no acceleration
        u = 0
        self.state_estimateX = np.dot(self.transition_F, self.state_estimateX) +

        np.dot(self.control_B, u)
        self.covariance_P = np.dot(np.dot(self.transition_F, self.covariance_P),

        self.transition_F.T) + self.noise_covariance_Q
        return self.state_estimateX

    def correct(self, Z):
        """
        corrects the kalman filter using the kalman gain
        returns the estimated state vector
        """
        n = self.transition_F.shape[1]
        y = Z - np.dot(self.obs_model_H, self.state_estimateX)
        kalman_brackets = self.obs_noise_covariance_R + np.dot(self.obs_model_H,

        np.dot(self.covariance_P, self.obs_model_H.T))

        Kalman_gain = np.dot(np.dot(self.covariance_P, self.obs_model_H.T),

        np.linalg.inv(kalman_brackets))

        self.state_estimateX = self.state_estimateX + np.dot(Kalman_gain, y)
        identity = np.eye(n)

        self.covariance_P = np.dot(np.dot(identity - np.dot(Kalman_gain,

        self.obs_model_H), self.covariance_P), (identity - np.dot(Kalman_gain,

        self.obs_model_H)).T) + np.dot(np.dot(Kalman_gain,

        self.obs_noise_covariance_R), Kalman_gain.T)

        return self.state_estimateX
```

4.2 Object Detection with Kalman filter

The key idea is to seperate the video into multiple frames and apply kalman filter to each frame. There are two steps to apply a kalman filter for object detection. First of all, we apply some image transformation to extract the contour of the

saliency object through median blur and Otsu's threshold. We then cast it with a rectangle and take its geometry center as the observation. Second, we simply apply kalman filter to predict its location and update the prediction for next-state processing.



As illustrated above, the left figure shows the saliency map and the right figure shows the kalman predictions. Pink dots represents historical trajectories while the green binding box represents the measured location and purple box shows the predicted location.

The program can be evoked with the following command:

```
$python main.py
Enter video name:
$xxx.mp4 // xxx.mp4 is the name for the video.
```

References

Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.