KALMAN FİLTRESİ

Kalman Filtresi Nedir?

Kalman filtresi, bir sistemin mevcut durumunu, belirsizlikler (gürültü veya hata) içeren verilere dayanarak en iyi şekilde tahmin etmek için kullanılan bir matematiksel algoritmadır. **Tahmin** ve **ölçüm** verilerini birleştirerek, sistemin durumunu olasılıksal olarak günceller.

Neden Kullanılır?

- Gürültülü verilerden doğru sonuç çıkarmak.

- Sistemin mevcut ve gelecekteki durumunu tahmin etmek.

- Sensör hatalarını azaltmak ve doğruluğu artırmak.

Gerçek hayattan somut bir örnek;

Bir roketin fırlatılması sırasında yakıt yanmasıyla oluşan yüksek sıcaklık, roketin hassas elektronik komponentlerine ve diğer yapısal elemanlarına ciddi zarar verebilir. Eğer bu sıcaklık kontrol edilmezse, aşırı ısınma roketin patlamasına neden olabilir.

Ancak şöyle bir sorun var: Yakıt yanması sırasında sıcaklığı doğrudan ölçmek, aşırı sıcaklık ve ulaşılmaz alanlar nedeniyle pratikte imkânsızdır. İşte burada Kalman filtresi devreye giriyor.

Kalman Filtresinin İşleyişi Nasıldır?

Kalman filtresi, iki ana adımdan oluşur:

1-Zaman Güncellemesi / Tahmin (Prediction):

-Sistem modeline dayalı olarak, mevcut durumdan gelecekteki durum tahmin edilir.

-Bu adımda, sistemin **hareket denklemleri** ve kontrol girdileri kullanılır.

2.Ölçüm Güncellemesi / Düzeltme (Correction):

-Tahmin edilen durum, sensörlerden alınan ölçümlerle karşılaştırılır.

-Ölçümlerden gelen hata, tahmin edilen durum üzerinde düzeltmeler yapmak için kullanılır.

Bu iki adım sürekli tekrar eder ve **özyinelemeli bir yapı** oluşturur.

Başlangıç Durumu ve Kovaryans Tahmini:

Başlangıç tahmini : Sistemin ilk durumu hakkında tahminde bulunulur. Bu genellikle sıfır olabilir ya da sistemin başlangıç durumu hakkında bilinen bir değer olabilir.

Başlangıç kovaryansı : Bu, tahminin doğruluğunu belirleyen bir değerdir ve genellikle yüksek bir değerle başlanır, çünkü başlangıçta tahminlerin ne kadar belirsiz olduğunu gösterir.

Sistem modeli:

$$X_k = AX_{k-1} + BU_k + W_{k-1}$$

(Denklem 1)

Denklem 1 tahmin modelidir. X_k tahmin değeridir. Önceki tahmin değerinin (X_{k-1}) üzerine kontrol sinyali (U_k) ve önceki işlem gürültüsü (W_k) eklenerek oluşturulur.

Ölçüm Modeli:

$$Z_k = HX_k + V_k$$

(Denklem 2)

2 numaralı denklem ölçüm modelidir. Herhangi bir andaki ölçüm değeri Z_k, o andaki tahmin değeri (X_k) ve ölçüm gürültüsünün (V_k) toplamıdır.

A, B ve H elemanları matrislerin genel gösterimidir. A, durum dönüşüm matrisi olup durum vektörünün zamana bağlı ilişkisini yansıtmaktadır. B matrisi kontrol girdisi ile sistem arasındaki geçişi belirtmektedir. Ölçüm modelinde kullanılan H matrisi, gözlem matrisi olup sistem ile ölçüm arasındaki ilişkiyi yansıtmaktadır. W ve V ifadeleri, 0 ortalamalı ve sırasıyla Q ve R varyanslı gürültü bileşenleridir. Q durum hata kovaryansı, R ise ölçüm gürültü kovaryansı vektörü olup, sırasıyla durum ve ölçüm sayısı boyutlulardır.

Zaman Güncellemesi / Tahmin (Prediction):

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k$$

(Denklem 3)

$$P_k^- = AP_{k-1}A^T + Q$$

(Denklem 4)

Denklem 3 ve Denklem 4 ile sistemin bir sonraki filtre zaman adımındaki durum ve hata kovaryansı tahminleri yapılmaktadır. Bu eşitliklerdeki $\hat{x}k$ ve Pk ifadeleri, sırasıyla anlık durum ve hata kovaryansı tahminlerini belirtmektedir. $\hat{x}k-1$ ve $Pk-1$ ifadeleri, ölçüm güncellemesi aşamasında hesaplanarak geri beslenen durum ve hata kovaryansı değerleridir. Q durum hata kovaryansı matrisi olup durum sayısı boyutlu bir kare matristir.

Ölçüm Güncellemesi / Düzeltme (Correction):

Ölçüm güncellemesi aşamasında, zaman güncellemesi aşamasından gelen hata kovaryansı kullanılarak Kalman kazancı hesaplanmaktadır. Gelen anlık ölçümler, Kalman kazancı ve zaman güncellemesinde hesaplanan durum tahminleri kullanılarak durum kestirimleri yapılmaktadır. Ardından hata kovaryansı, hesaplanan Kalman kazancı ile güncellenerek zaman güncellemesi aşamasını tekrar beslemektedir.

$$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1}$$

(Denklem 5)

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$$

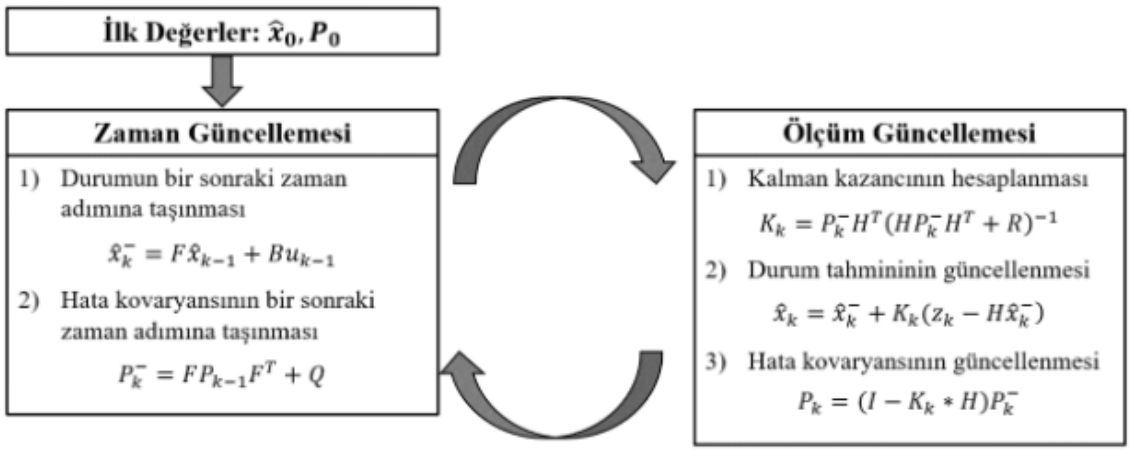
(Denklem 6)

$$P_k = (1 - K_k H) P_k^-$$

(Denklem 7)

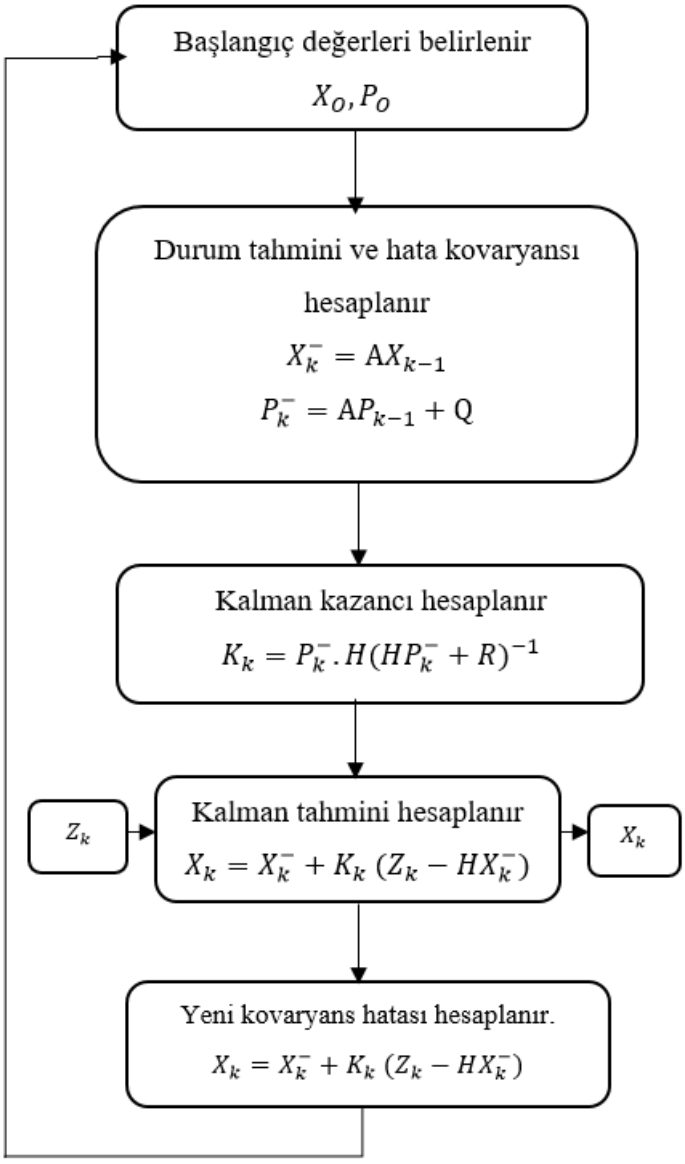
Yukarıdaki eşitliklerdeki Kk, $\hat{x}k$ ve Pk ifadeleri, sırasıyla Kalman kazancı, durum tahmini ve hesaplanan hata kovaryansını belirtmektedir. H gözlem matrisi, R ise ölçüm hatası kovaryans matrisidir. zk ifadesi anlık olarak gelen ölçüm girdilerini temsil etmektedir. I ise birim matristir.

Kalman Filtresi Diyagramı:



Şekil 1

Kalman Filtresi Algoritması



Şekil 2

Kalman Filtresi Yazılımı:

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <random>
#include <cmath>
#include <vector>

using namespace std;
using namespace cv;

// Ortalama ve standart sapma ile rastgele gürültü üretme
float generateNoise(float mean, float stddev) {
    static random_device rd;
    static mt19937 gen(rd());
    normal_distribution<float> dist(mean, stddev);
    return dist(gen);
}

// Sensör verilerini gürültü ve drift ile simüle etmek
Point2f simulateSensor(const Point2f& truePosition, float noiseStd, float drift) {
    return Point2f(truePosition.x + generateNoise(0, noiseStd) + drift,
        truePosition.y + generateNoise(0, noiseStd) + drift);
}

int main() {
    // Simülasyon parametreleri
    const int numSteps = 100;
    const float dt = 0.1f; // zaman adımı (değişken)
    const float processNoiseStd = 0.1f;
    const float measurementNoiseStd1 = 0.2f;
    const float measurementNoiseStd2 = 0.1f;
    const float sensorDrift1 = 0.05f;
```

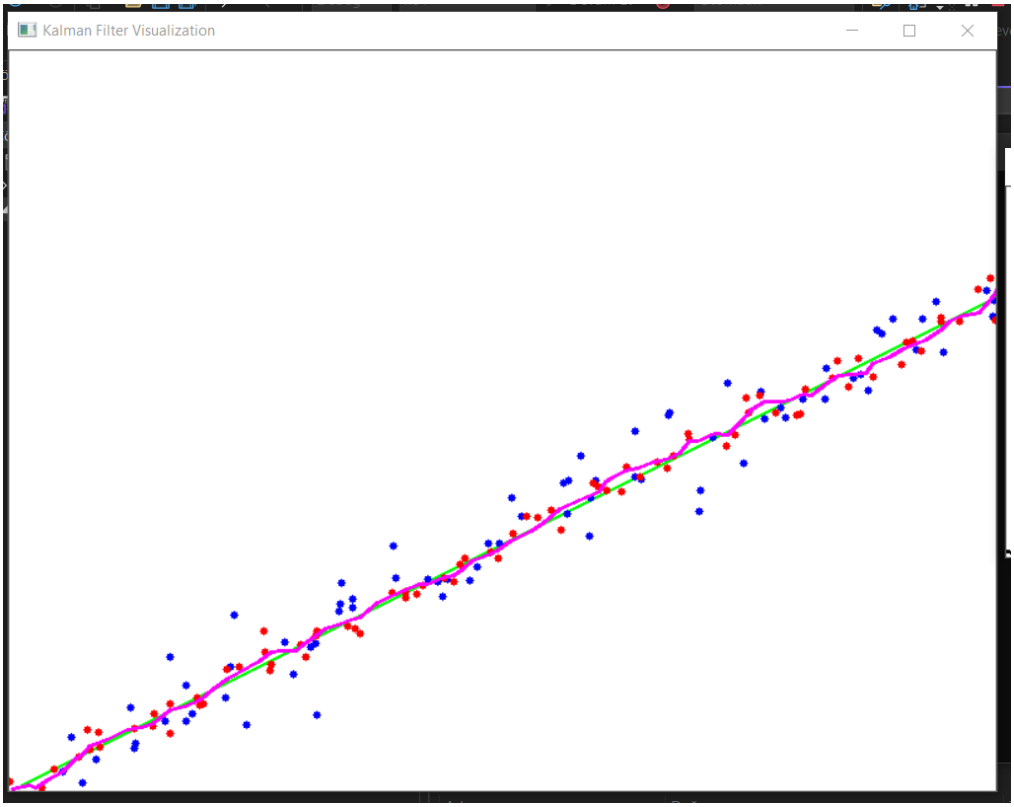
```
const float sensorDrift2 = 0.03f;

// Kalman Filtresini başlatma
KalmanFilter kf(4, 2, 0); // 4 durum değişkeni, 2 ölçüm
kf.transitionMatrix = (Mat_<float>(4, 4) <<
    1, 0, dt, 0,
    0, 1, 0, dt,
    0, 0, 1, 0,
    0, 0, 0, 1);

kf.measurementMatrix = (Mat_<float>(2, 4) <<
    1, 0, 0, 0,
    0, 1, 0, 0);

kf.processNoiseCov = (Mat_<float>(4, 4) <<
    processNoiseStd, 0, 0, 0,
    0, processNoiseStd, 0, 0,
    0, 0, processNoiseStd, 0,
    0, 0, 0, processNoiseStd);

// İlk hata kovaryansı
kf.errorCovPost = Mat::eye(4, 4, CV_32F);
kf.statePost = (Mat_<float>(4, 1) << 0, 0, 0, 0);  // Başlangıç durumu

// Gerçek Veri ve Sensör Ölçümleri
vector<Point2f> groundTruth;
vector<Point2f> measurements1, measurements2;
vector<Point2f> kalmanEstimates;
vector<float> errors;  // Hataları saklamak için bir vektör

Point2f currentPosition(0, 0);
for (int i = 0; i < numSteps; i++) {
```

```
        // Gerçek Pozisyonu Güncelleme
        currentPosition.x += 1.0f * dt;
        currentPosition.y += 0.5f * dt;
        groundTruth.push_back(currentPosition);

        // Gürültülü sensör ölçümlerini simüle etmek
        measurements1.push_back(simulateSensor(currentPosition, measurementNoiseStd1,
sensorDrift1));
        measurements2.push_back(simulateSensor(currentPosition, measurementNoiseStd2,
sensorDrift2));

        // İki sensör ölçümünü ortalamak (basitlik açısından)
        Point2f combinedMeasurement = (measurements1.back() + measurements2.back()) * 0.5;
        Mat measurement = (Mat_<float>(2, 1) << combinedMeasurement.x, combinedMeasurement.y);

        // Kalman filtresi prediction ve correction
        Mat prediction = kf.predict();

        // Ağırlıklı sensör verileri ile düzeltme adımı
        float weight1 = 1.0f / (measurementNoiseStd1 * measurementNoiseStd1);
        float weight2 = 1.0f / (measurementNoiseStd2 * measurementNoiseStd2);

        float totalWeight = weight1 + weight2;
        weight1 /= totalWeight;
        weight2 /= totalWeight;

        // Sensör ölçümlerinin ağırlıklı ortalaması
        Point2f weightedMeasurement = measurements1.back() * weight1 + measurements2.back() *
weight2;
        Mat weightedMeasurementMat = (Mat_<float>(2, 1) << weightedMeasurement.x,
weightedMeasurement.y);

        // Ağırlıklı ölçümle correction adımını gerçekleştirme
        Mat estimate = kf.correct(weightedMeasurementMat);
```

```
    kalmanEstimates.push_back(Point2f(estimate.at<float>(0), estimate.at<float>(1)));

    // Hata oranını hesapla (gerçek pozisyon ile tahmin arasındaki fark)
    float error = sqrt(pow(currentPosition.x - kalmanEstimates.back().x, 2) +
        pow(currentPosition.y - kalmanEstimates.back().y, 2));
    errors.push_back(error);
}

// Görselleştirme
Mat display(600, 800, CV_8UC3, Scalar(255, 255, 255));

// Gerçek yolun çizilmesi
for (size_t i = 1; i < groundTruth.size(); i++) {
    int y1 = display.rows - groundTruth[i - 1].y * 100;
    int y2 = display.rows - groundTruth[i].y * 100;
    line(display, Point(groundTruth[i - 1].x * 100, y1),
        Point(groundTruth[i].x * 100, y2), Scalar(0, 255, 0), 2);
}

// Sensör 1'in yolunu çizme (mavi noktalar)
for (size_t i = 0; i < measurements1.size(); i++) {
    int y = display.rows - measurements1[i].y * 100;
    circle(display, Point(measurements1[i].x * 100, y), 3, Scalar(255, 0, 0), -1);
}

// Sensör 2'nin yolunu çizme (kırmızı noktalar)
for (size_t i = 0; i < measurements2.size(); i++) {
    int y = display.rows - measurements2[i].y * 100;
    circle(display, Point(measurements2[i].x * 100, y), 3, Scalar(0, 0, 255), -1);
}

// Kalman filtresi tahmini çizme (mor çizgiler)
```

```
    for (size_t i = 1; i < kalmanEstimates.size(); i++) {
        int y1 = display.rows - kalmanEstimates[i - 1].y * 100;
        int y2 = display.rows - kalmanEstimates[i].y * 100;
        line(display, Point(kalmanEstimates[i - 1].x * 100, y1),
            Point(kalmanEstimates[i].x * 100, y2), Scalar(255, 0, 255), 2);
    }

    // Son görselleştirme
    imshow("Kalman Filter Visualization", display);

    // Hata oranlarını gösteren pencere
    Mat errorGraph(300, 600, CV_8UC3, Scalar(255, 255, 255));
    for (size_t i = 1; i < errors.size(); i++) {
        line(errorGraph, Point((i - 1) * 6, 300 - errors[i - 1] * 50),
            Point(i * 6, 300 - errors[i] * 50), Scalar(0, 0, 0), 2);
    }
    imshow("Error Graph", errorGraph);

    waitKey(0);

    return 0;
}
```

Görsel çıktı;



Şekil 3

Şekil 4

Referanslar:

- Akçay, H. (2023) Aldatma saldırısı tespiti ve aldatmaya karşı önlem için kalman filtresi tasarımı (TOBB Ekonomi ve Teknoloji Üniversitesi Fen Bilimleri Enstitüsü yüksek lisans tezi). YÖK tez merkezi.
- Çayıroğlu, İ. (2012). *Kalman filtresi ve bir programlama örneği*. Erişim adresi: https://www.ibrahimcayiroglu.com/Dokumanlar/Makale_BilgiPaylasim/(1-2012)-Kalman_Filtresi_Ve_Bir_Programlama_Ornegi-Ibrahim_CAYIROGLU.pdf