VEHICLE TRACKING USING KALMAN FILTER

IN THE PARTIAL FULFILLMENT OF REQUIREMENTS FOR AWARD

FOR DEGREE

OF

BACHELOR OF TECHNOLOGY

By

AKAASH NIDHISS PANDIAN
(2K19/IT/008)

ANASUYA MITHRA PARTHAJE
(2K19/IT/018)

AHMAD HABIB KHAN
(2K19/SE/003)

ANSHUMAN ROY
(2K19/CO/068)

UNDER THE GUIDANCE OF

PROF. **RAJIV KAPOOR**

Delhi Technological University
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042
Session: 2021-22

<u>CANDIDATES'S DECLARATION</u>

We, **Ahmad Habib Khan** (2K19/SE/003), **Akaash Nidhiss Pandian** (2K19/IT/008), **Anasuya Mithra Parthaje** (2K19/IT/018), **Anshuman Roy** (2K19/CO/068), hereby declare that the project Dissertation titled *"**Kalman Filter**"* submitted by us to Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of bachelor of technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma, Associateship, Fellowship or other similar title or recognition.

Place: Delhi

Date: 1/11/2021

CERTIFICATE

I hereby certify that the project Dissertation titled *"**Kalman Filter**"* which is submitted by **Ahmad Habib Khan** (2K19/SE/003), **Akaash Nidhiss Pandian** (2K19/IT/008), **Anasuya Mithra Parthaje** (2K19/IT/018), **Anshuman Roy** (2K19/CO/068), to Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Bachelor of Technology, is a record of the project work carried out by the students under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree and Diploma to this University or elsewhere.

Place: Delhi Prof. **Mr. Rajiv Kapoor**

Date: 1/11/2021 SUPERVISOR

ACKNOWLEGMENT

It is great pleasure for me to undertake this project we feel highly doing the project entitled- *“**Kalman Filter**”* I am grateful to my project guide prof. **Mr. Rajiv Kapoor**. This project would not have completed without their enormous help and worthy experience. Whenever I was in need, they were behind me. Although this report has been prepared with utmost care and deep routed interest. Even then I accept respondent and imperfection.

CONTENTS

ABSTRACT

Most modern systems are equipped with numerous sensors that provide estimation of hidden (unknown) variables based on a series of measurements. For example, a GPS receiver provides location and velocity estimation, where location and velocity are the hidden variables and differential time of the satellites' signals' arrival are the measurements.

One of the biggest challenges of tracking and control systems is providing accurate and precise estimation of the hidden variables in the presence of uncertainty. In GPS receivers, the measurement uncertainty depends on many external factors such as thermal noise, atmospheric effects, slight changes in satellite positions, receiver clock precision and many more.

The Kalman Filter is one of the most important and common estimation algorithms. The Kalman Filter produces estimates of hidden variables based on inaccurate and uncertain measurements. Also, the Kalman Filter provides a prediction of the future system state based on past estimations.

The filter is named after Rudolf E. Kálmán (May 19, 1930 – July 2, 2016). In 1960, Kálmán published his famous paper describing a recursive solution to the discrete-data linear filtering problem.

Today the Kalman filter is used in Tracking Targets (Radar), location and navigation systems, control systems, computer graphics and much more.

INTRODUCTION :

OVERVIEW

Kalman filtering is an algorithm that allows us to estimate the states of a system given the observations or measurements. It is a useful tool for a variety of different applications including object tracking and autonomous navigation systems, economics prediction, etc.

Even though it is a relatively simple algorithm, it's still not easy for some people to understand and implement it in a computer program such as Python. Therefore, the aim of this tutorial is to help some people to comprehend easily the implementation of Kalman filter in Python.

While there are some excellent references detailing the theory behind the Kalman filter, so we’re not going to dive deeply into the theoretical details. Instead, this article presents the Kalman filter from a practical usage perspective only.

BACKGROUND OF STUDY

Since our purpose of this tutorial is to implement the Kalman filter in computer programing code, we'll only consider this tutorial for the Discrete Kalman filter.

The basic idea of the Kalman filter is by using the prior knowledge of the state, the filter makes a forward projection state or predicts the next state.

The Kalman Filter is intended to estimate the state of a system at time k using the linear stochastic difference equation assuming that the state of a system at a time k evolved from the prior state at time k-1 as written in the following form:

$$\mathbf{x}_k = A\mathbf{x}_{k-1} + B\mathbf{u}_{k-1} + \mathbf{w}_{k-1}$$

It is always paired with the measurement model zk that describes a relation

between the state and measurement at the current step k. It is written as:

$$\mathbf{z}_k = H\mathbf{x}_k + \mathbf{v}_k$$

where:

- A, a matrix nxn, is the state transition matrix relating the previous time step k-1 to the current state k
- B, a matrix nxl, is a control input matrix applied to the optional control input u_{k-1}
- H, a matrix m x n, is a transformation matrix that transforms the state into the measurement domain
- w_k and v_k represent the process noise vector with the covariance Q and the measurement noise vector with the covariance R, respectively. They are assumed statistically independence Gaussian noise with the normal probability distribution.

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

PROJECT PLANNING

Project planning is part of project management, related to the use of schedules such as Gantt charts to plan and report progress within the project area. Initially, the scope of the project is defined and appropriate methods are determined to complete the project. Following this step, the length of time for the various activities required to complete the task is recorded and sorted according to the structure of the work division. Logical interdependence between tasks is defined using a work network diagram that allows to identify an important approach. Floating time or flexibility in schedule can be calculated using project management software. The required resources can then be estimated and the cost of each project can be allocated to each resource, giving the total cost of the project. At this stage, the project plan can be developed to achieve the right balance between resource utilization and project duration to align with project objectives. Once established and agreed upon, the system becomes what is known as the foundation. Progress will be measured against the foundation throughout the life of the project.

IMPLEMENTATION:

State transition matrix A and control matrix B

Remember, in 1-D Kalman Filter, we consider that both state and velocity are in one-direction only. Based on Kinematic equation, the relation between the position x and velocity x can be written as the following:

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ \dot{x}_k \end{bmatrix} = \begin{bmatrix} x_{k-1} + \dot{x}_{k-1}\Delta t + 1/2\ddot{x}_{k-1}\Delta t^2 \\ \dot{x}_{k-1} + \ddot{x}_{k-1}\Delta t \end{bmatrix}$$

Then we can write eq.(1) in the form of matrix multiplication as follows:

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ \dot{x}_k \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ \dot{x}_{k-1} \end{bmatrix} + \begin{bmatrix} \frac{1}{2}(\Delta t)^2 \\ (\Delta t \end{bmatrix} \ddot{x}_{k-1}$$

Now, we're going to focus on 2-D Kalman Filter. The equations of 2-D Kalman Filter whose position and velocity must be considered in 2-dimensional direction, the x– and y– directions, can be created by modifying the 1-D Kalman Filter equations. Meaning that instead of considering only for the position and velocity in one direction, let's say the x-direction, we need to take into account the position and velocity in the y-direction as well.

The state in time k can be predicted by the previous state in time k-1. Let x and y be the positions in the x– and y-directions, respectively, and also let x and \dot y be the velocities in x– and y-directions, respectively. Then the 2-D Kinematic equation for state x_y can be written as:

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ y_k \\ \dot{x}_k \\ \dot{y}_k \end{bmatrix} = \begin{bmatrix} x_{k-1} + \dot{x}_{k-1}\Delta t + 1/2\ddot{x}_{k-1}\Delta t^2 \\ y_{k-1} + \dot{y}_{k-1}\Delta t + 1/2\ddot{y}_{k-1}\Delta t^2 \\ \dot{x}_{k-1} + \ddot{x}_{k-1}\Delta t \\ \dot{y}_{k-1} + \ddot{y}_{k-1}\Delta t \end{bmatrix}$$

We can write eq.(3) into the form of matrix multiplication as follows:

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ y_k \\ \dot{x}_k \\ \dot{y}_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \dot{x}_{k-1} \\ \dot{y}_{k-1} \end{bmatrix} + \begin{bmatrix} \frac{1}{2}(\Delta t)^2 & 0 \\ 0 & \frac{1}{2}(\Delta t)^2 \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} \ddot{x}_{k-1} \\ \ddot{y}_{k-1} \end{bmatrix}$$

The eq.(4) can be simplified as follows:

$$\mathbf{x}_k = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{x}_{k-1} + \begin{bmatrix} \frac{1}{2}(\Delta t)^2 & 0 \\ 0 & \frac{1}{2}(\Delta t)^2 \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \mathbf{a}_{k-1}$$

Where x_k is the current state, x_{k-1} is the previous state, and a_{k-1} is a vector of the previous acceleration in x– and y-directions.

So, we the matrices A and B as follows:

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

$$\mathbf{B} = \begin{bmatrix} \frac{1}{2}(\Delta t)^2 & 0 \\ 0 & \frac{1}{2}(\Delta t)^2 \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix}$$

Transformation matrix H

The measurement model for 2-D Kalman filter is the same as the 1-D Kalman filter model:

$$\mathbf{z}_k = H\mathbf{x}_k + \mathbf{v}_k$$

In deriving the observation model, we assume that we're only measuring the position but not the velocity. Whereas, there are cases where the velocity must be taken into account to incorporate Doppler's effect, for example, airplane and satellite tracking systems.

We're now considering the particular case where the velocity will not be considered in the measurement process. So, we can write the measurement model as follows:

$$\mathbf{z}_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \\ \dot{x}_k \\ \dot{y}_k \end{bmatrix} + \mathbf{v}_k$$

So, we have the transformation matrix H as:

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

Process noise covariance matrix Q

The process noise covariance matrix Q for 2-D Kalman filter can be written as:

$$\mathbf{Q} = \begin{array}{c} \\ x \\ y \\ \dot{x} \\ \dot{y} \end{array} \begin{array}{c} \begin{array}{cccc} x & y & \dot{x} & \dot{y} \end{array} \\ \begin{bmatrix} \sigma_x^2 & 0 & \sigma_x\sigma_{\dot{x}} & 0 \\ 0 & \sigma_y^2 & 0 & \sigma_y\sigma_{\dot{y}} \\ \sigma_{\dot{x}}\sigma_x & 0 & \sigma_{\dot{x}}^2 & 0 \\ 0 & \sigma_{\dot{y}}\sigma_y & 0 & \sigma_{\dot{y}}^2 \end{bmatrix} \end{array}$$

By using the same philosophy as what we've discussed earlier in my previous tutorial for 1-D Kalman filter, we can rewrite the process noise covariance matrix for 2-D Kalman filter as:

$$\mathbf{Q} = \begin{bmatrix} \frac{\Delta t^4}{4} & 0 & \frac{\Delta t^3}{2} & 0 \\ 0 & \frac{\Delta t^4}{4} & 0 & \frac{\Delta t^3}{2} \\ \frac{\Delta t^3}{2} & 0 & \Delta t^2 & 0 \\ 0 & \frac{\Delta t^3}{2} & 0 & \Delta t^2 \end{bmatrix} \sigma_a^2$$

Where the σa is the magnitude of the standard deviation of the acceleration that is basically the process noise effecting on the process noise covariance matrix.

Measurement noise covariance matrix R

In the 2-D Kalman filter, we suppose that the measurement positions x and y are both independent, so we can ignore any interaction between them so that the covariance x and y is 0. We look at only the variance in the x and the variance in the y.

Then, the measurement noise covariance R can be written as follows:

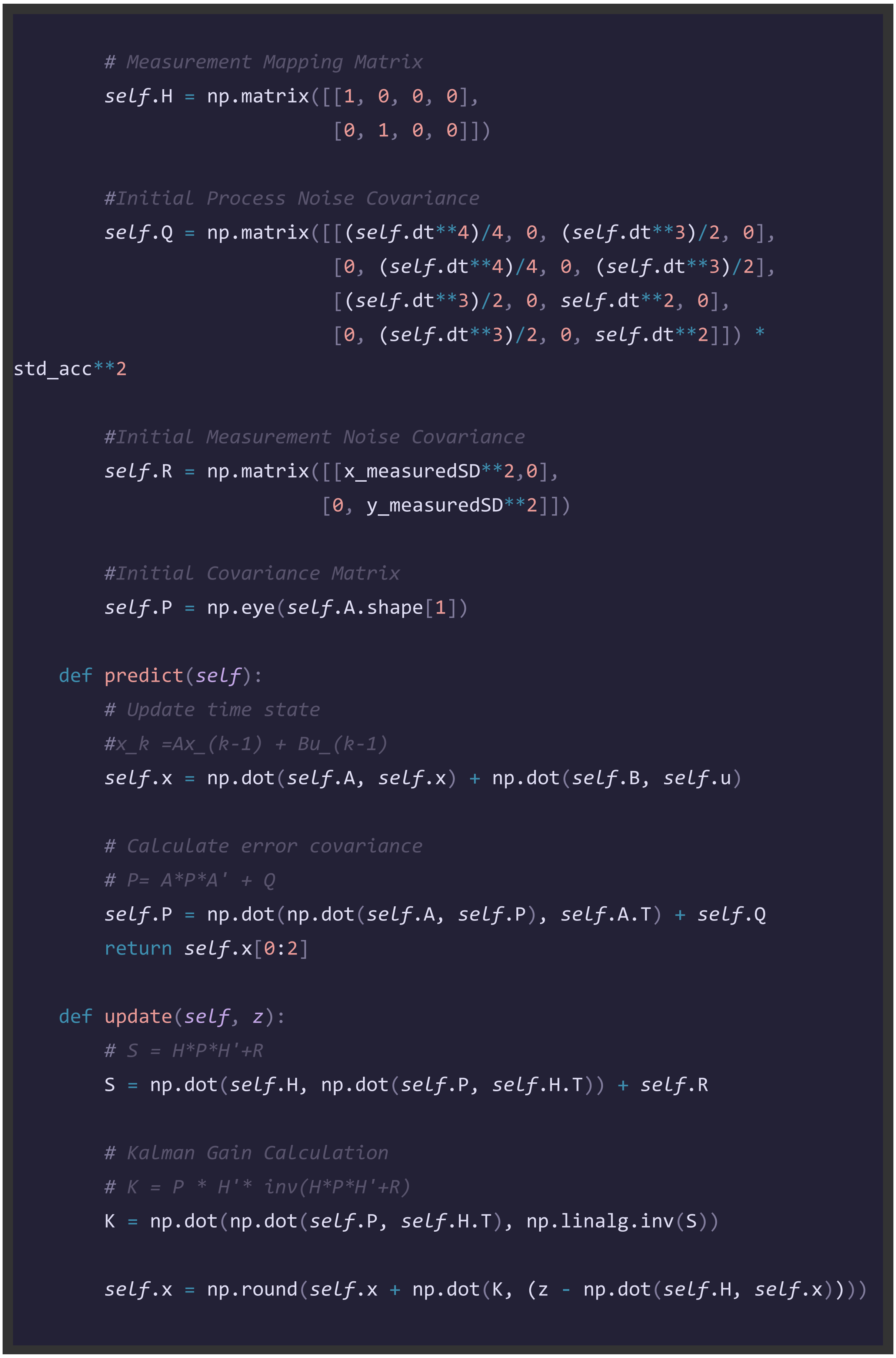
$$\mathbf{R} = \begin{array}{c c} & \begin{array}{cc} x & y \end{array} \\ \begin{array}{c} x \\ y \end{array} & \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix} \end{array}$$

PYTHON IMPLEMENTATION

This project has three files, namely:

- Kalman.py

```python
import numpy as np
import matplotlib.pyplot as plt

class KalmanFilter(object):
    def __init__(self, dt, ux,uy, std_acc, x_measuredSD, y_measuredSD):
        """
       dt -> sampling time (time for 1 cycle)
       ux -> acceleration in x-direction
       uy -> acceleration in y-direction
       std_acc -> process noise magnitude
       x_measuredSD -> standard deviation of the measurement in x-direction
       y_measuredSD -> standard deviation of the measurement in y-direction

       """
        # Sampling time
        self.dt = dt

        # The control input variables
        self.u = np.matrix([[ux],[uy]])

        # Intial State
        self.x = np.matrix([[0], [0], [0], [0]])

        # State Transition Matrix A
        self.A = np.matrix([[1, 0, self.dt, 0],
                            [0, 1, 0, self.dt],
                            [0, 0, 1, 0],
                            [0, 0, 0, 1]])

        # Control Input Matrix B
        self.B = np.matrix([[(self.dt**2)/2, 0],
                            [0,(self.dt**2)/2],
                            [self.dt,0],
                            [0,self.dt]])
```



```
        # Measurement Mapping Matrix
        self.H = np.matrix([[1, 0, 0, 0],
                            [0, 1, 0, 0]])

        #Initial Process Noise Covariance
        self.Q = np.matrix([[(self.dt**4)/4, 0, (self.dt**3)/2, 0],
                            [0, (self.dt**4)/4, 0, (self.dt**3)/2],
                            [(self.dt**3)/2, 0, self.dt**2, 0],
                            [0, (self.dt**3)/2, 0, self.dt**2]]) *
std_acc**2

        #Initial Measurement Noise Covariance
        self.R = np.matrix([[x_measuredSD**2,0],
                           [0, y_measuredSD**2]])

        #Initial Covariance Matrix
        self.P = np.eye(self.A.shape[1])

    def predict(self):
        # Update time state
        #x_k =Ax_(k-1) + Bu_(k-1)
        self.x = np.dot(self.A, self.x) + np.dot(self.B, self.u)

        # Calculate error covariance
        # P= A*P*A' + Q
        self.P = np.dot(np.dot(self.A, self.P), self.A.T) + self.Q
        return self.x[0:2]

    def update(self, z):
        # S = H*P*H'+R
        S = np.dot(self.H, np.dot(self.P, self.H.T)) + self.R

        # Kalman Gain Calculation
        # K = P * H'* inv(H*P*H'+R)
        K = np.dot(np.dot(self.P, self.H.T), np.linalg.inv(S))

        self.x = np.round(self.x + np.dot(K, (z - np.dot(self.H, self.x))))
        I = np.eye(self.H.shape[1])

        # Update error covariance matrix
        self.P = (I - (K * self.H)) * self.P
        return self.x[0:2]
```

- Detect.py

```
import cv2
import numpy as np

# Trained XML classifiers describes some features of the car object we want
to detect
car_cascade = cv2.CascadeClassifier( r'CarDetection/cars.xml')

def detect(frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detect cars
    cars = car_cascade.detectMultiScale(gray, 1.1, 1)
    centers = []

    # Drawing the border for the car object detected
    if len(cars)>0:
        x, y, w, h = cars[0]
        cv2.rectangle(frame, (x,y), (x+w,y+h), (0,0,255), 2)
        centrex = x + w/2
        centrey = y + h/2

        if (w > 20):
            print(w)
            centers.append(np.array([[centrex], [centrey]]))

    return centers
```

- main.py

```
import cv2
import numpy as np
from Detect import detect
from Kalman import KalmanFilter

def main():
    # Creating opencv video capture object
    vidCap = cv2.VideoCapture('car5.mp4')

    # Video reading speed
    vrs = 100
    hspeed = 100

    # Create KalmanFilter object kalmanF
    # KalmanFilter constructor -> KalmanFilter(dt, ux, uy, acceleration, x
measurement, y measurement)
    kalmanF = KalmanFilter(0.1, 1, 1, 1, 0.1,0.1)

    while(True):
        # Frame being read
        temp, imageSample = vidCap.read()

        # Detecting a car
        centreArr = detect(imageSample)

        # Centroid tracking if they are being detected by the code:
        if (len(centreArr) > 0):

            # Prediction function
            (x, y) = kalmanF.predict()
            x = int(x)
            y = int(y)

            # Rectangle depicting predicted car position
            cv2.rectangle(imageSample, (x - 15, y - 15), (x + 15, y + 15),
(255, 255, 0), 2)

            # Update function
            (x1, y1) = kalmanF.update(centreArr[0])
            x1 = int(x1)
            y1 = int(y1)

            # Rectangle depicting estimated car position
            cv2.rectangle(imageSample, (x1 - 15, y1 - 15), (x1 + 15, y1 +
15), (0, 255, 255), 2)
            center1 = int(centreArr[0][0])
            center2 = int(centreArr[0][1])

            cv2.putText(imageSample, "Estimated Position", (x1 + 15, y1 +
10), 0, 0.5, (0, 255, 255), 2)
            cv2.putText(imageSample, "Predicted Position", (x + 15, y), 0,
0.5, (255, 255, 0), 2)
            cv2.putText(imageSample, "Measured Position", (center1 + 15,
center2 - 15), 0, 0.5, (255,0,0), 2)

        cv2.imshow('image', imageSample)

        # To quit program
        if cv2.waitKey(2) & 0xFF == ord('q'):
            vidCap.release()
            cv2.destroyAllWindows()
            break

        cv2.waitKey(hspeed-vrs+1)



if __name__ == "__main__":
    main()
```

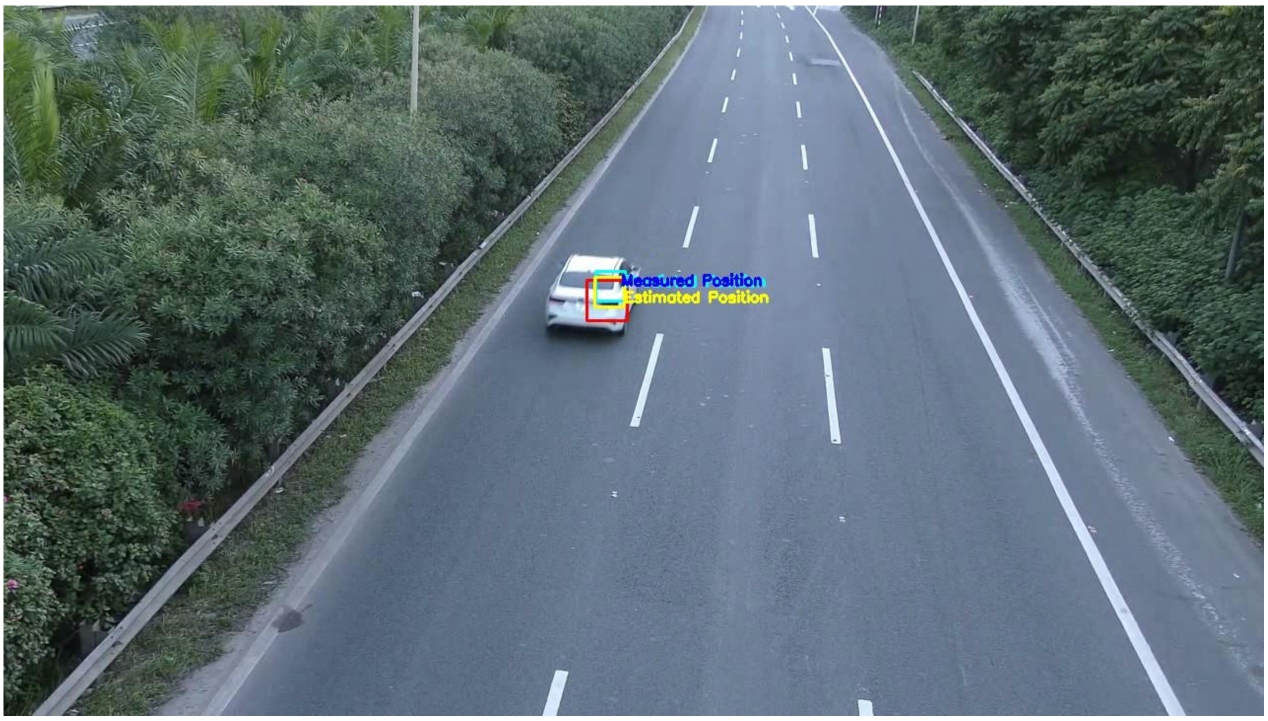
LEARNING OUTCOME:

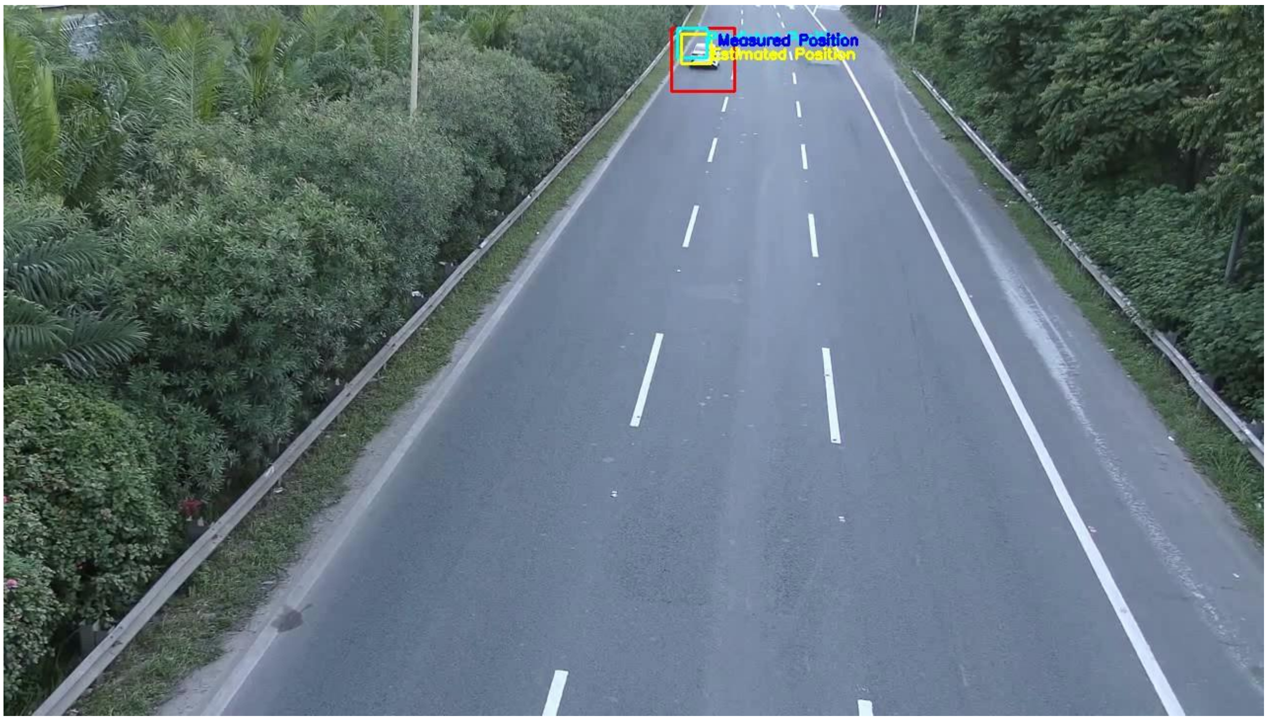
In the end, it is concluded that we have made effort on the following points:

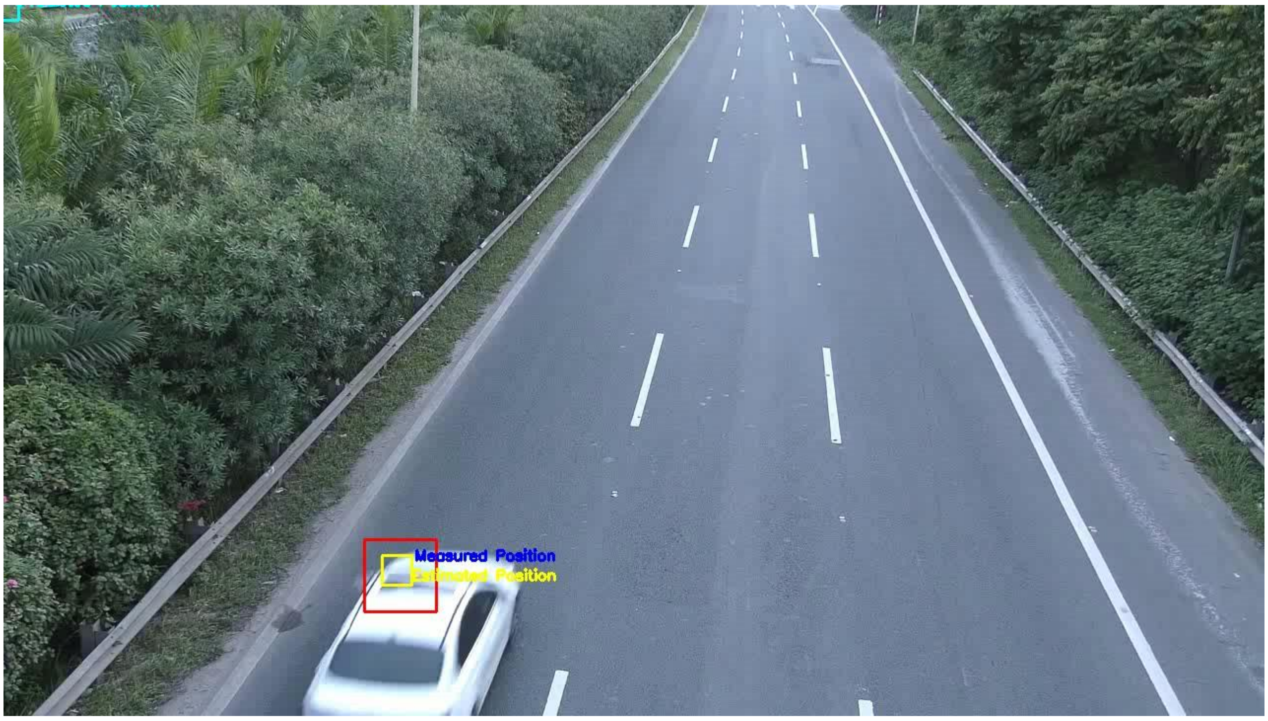
- A description of the background and context of the project and its relation to work already done in the area.
- Made a statement of the aims and objectives of the project.
- We define the problem on which we are working in the project.
- We describe the requirement Specifications of the system and the actions that can be done on these things.
- We understand the problem domain and produce a model of the system, which describes operations that can be performed on the system.
- We designed the user interface and security issues related to the system.
- Finally, the system is implemented and tested according to test cases.

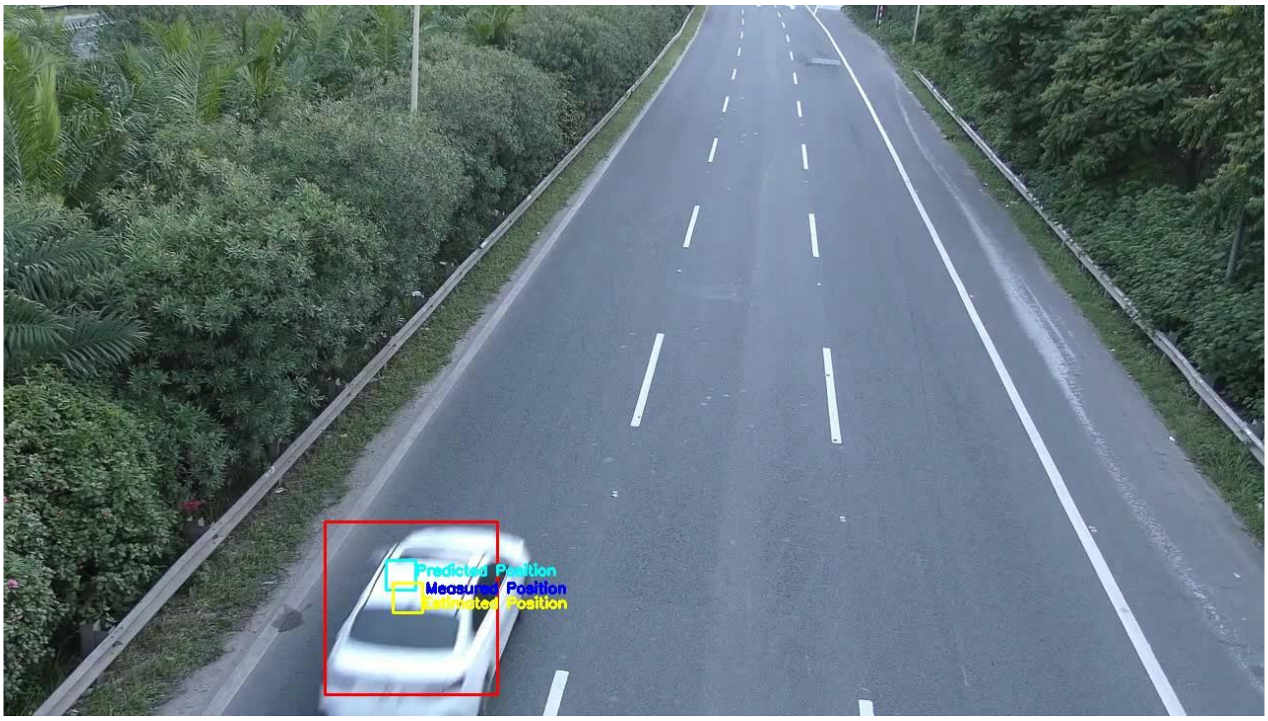
OUTPUT SCREENSHOTS:

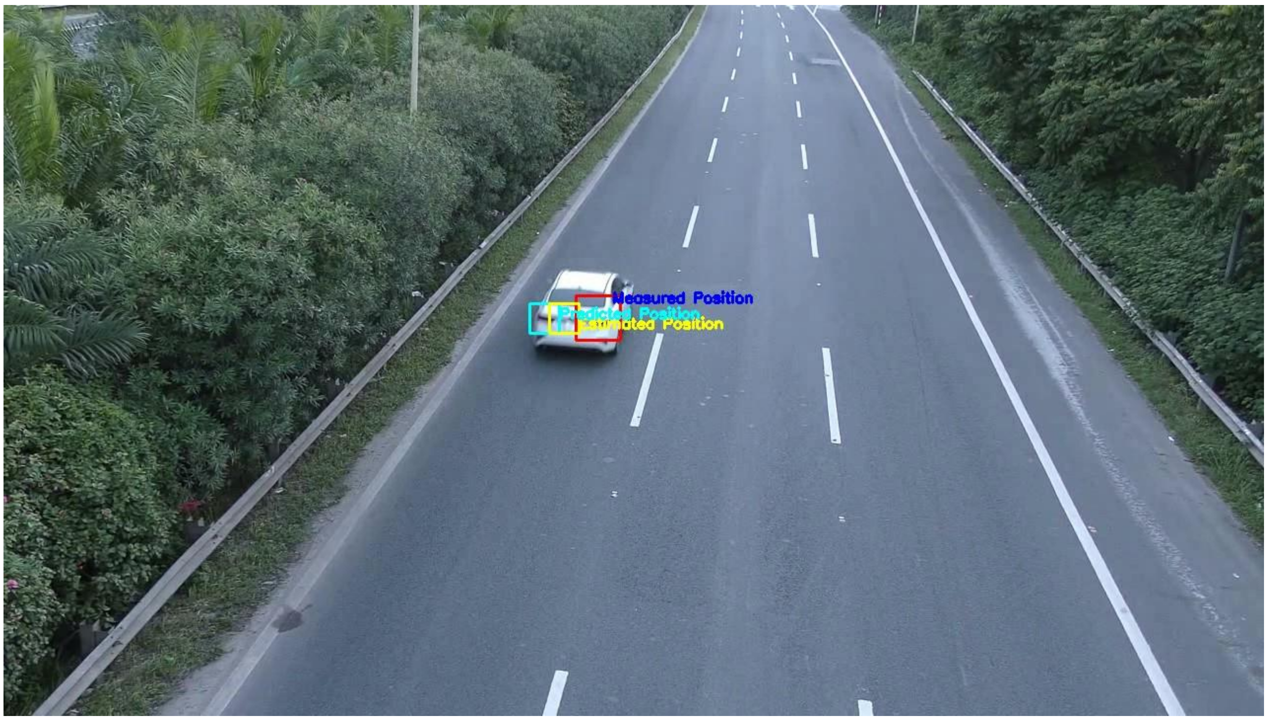
The following are some output screenshots displayed by our project code, illustrating the Estimated Position, Measured Position and Predicted Position of the car being tracked.

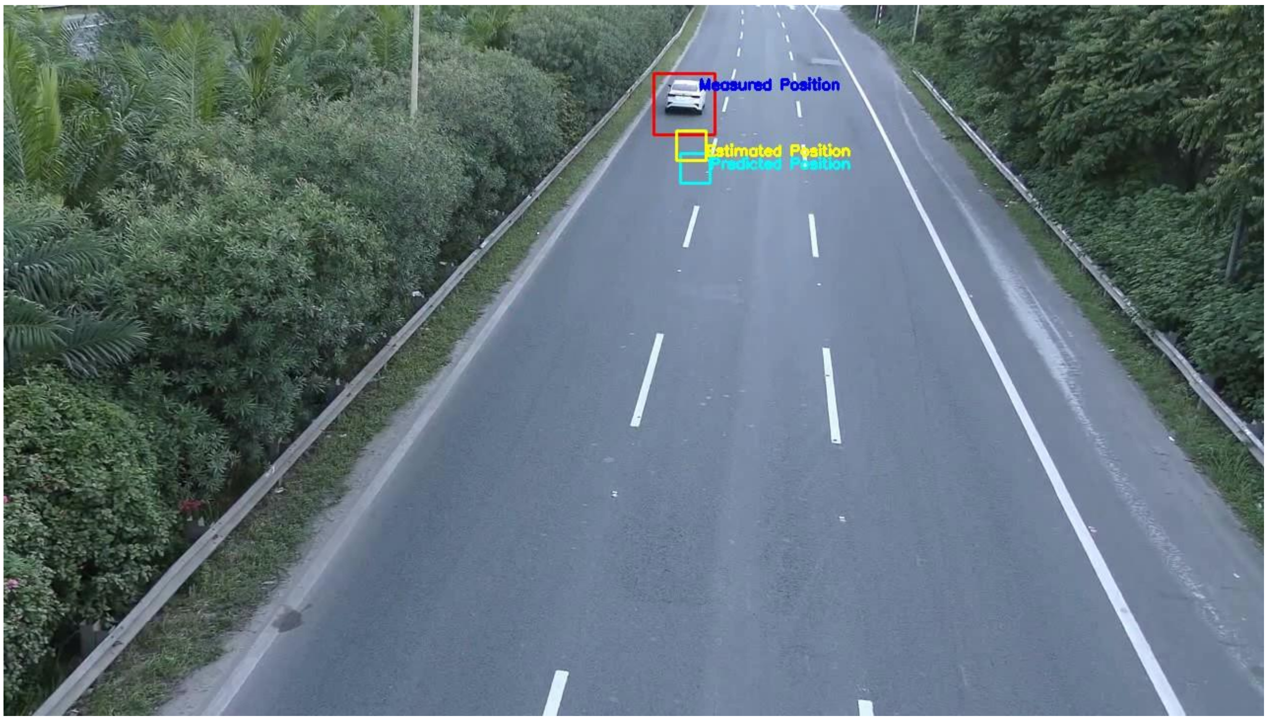





Measured Position
Estimated Position


Predicted Position
Measured Position
Estimated Position





A video featuring a sample output of our project code can be found here in the following google drive link:

carOutput.avi

CONCLUSION:

In conclusion, Kalman filtering is an algorithm that allows us to estimate the states of a system given the observations or measurements. It is a useful tool for a variety of different applications including object tracking and autonomous navigation systems, economics prediction, etc.

We implement the same in python, with the project having three files, namely: Kalman.py, Detect.py, and main.py.

Kalman.py contains one class called KalmanFilter consisting of three functions, __init__(), predict(), and update().

Detect.py file plays a role as an object detector and main.py is the main file of this project that we'll execute to track an object. At the beginning of this file, we import the function detect() from the file Detector.py, and class KalmanFilter from the file Kalman.py.

FUTURE ASPECTS:

In order to create an algorithm which reacted quickly to changes (Process Noise) in the object being detected, the Kalman Gain coefficient was accordingly increased in order to properly and quickly react to these changes. However, during the work on this project, it was realised that the car detector algo (contained in Detect.py) identified non-car objects and mistyped them as cars. Thus, the performance of the algorithm suffers when faced with high levels of measurement noise.

In order to enhance the working of the project code, necessary modifications should be performed to enhance the car detection algorithm and thus decrease overall measurement noise.

LITERATURE REVIEW:

The following papers were looked into to understand the Kalman Filter:

1. Welch, G., & Bishop, G. (1995). An introduction to the Kalman filter.
2. Meinhold, Richard J. & Singpurwalla, Nozer D. (1983). Understanding the Kalman Filter. The American Statistician, 37(2), 123–127.
3. Cox, Ingemar J. & Wilfong, Gordon T. (1990). Autonomous Robot Vehicles || The Kalman Filter: An Introduction to Concepts. , 10.1007/978-1-4613-8997-2(Chapter 15), 194–204.
4. Fitzgerald, R. (1971). Divergence of the Kalman filter. IEEE Transactions on Automatic Control, 16(6), 736–747.
5. Thacker, N. A., & Lacey, A. (1998). Tutorial: The kalman filter. *Imaging Science and Biomedical Engineering Division, Medical School, University of Manchester*, 61.
6. Yi, D. (1977). Kalman filter.
7. Rojas, R. (2003). The kalman filter. *available on www. robocup. mi. fu-berlin. de/buch/kalman. pdf*, 1-7.
8. Faragher, Ramsey (2012). Understanding the Basis of the Kalman Filter Via a Simple and Intuitive Derivation [Lecture Notes]. IEEE Signal Processing Magazine, 29(5), 128–132.
9. Humpherys, Jeffrey; Redd, Preston; West, Jeremy (2012). A Fresh Look at the Kalman Filter. SIAM Review, 54(4), 801–823.
10. Chan, Y.t.; Hu, A.g.c.; Plant, J.B. (1979). A Kalman Filter Based Tracking Scheme with Input Estimation. IEEE Transactions on Aerospace and Electronic Systems, AES-15(2), 237–244.

To improve the algorithm of the Kalman Filter used in the project, the following research papers were referred to:

1. Robustifying the Kalman Filter by developing a model where posterior distribution reverts to the prior distribution when extreme outlying observations are encountered.

   Meinhold, Richard J.; Singpurwalla, Nozer D. (1989). Robustification of Kalman Filter Models. Journal of the American Statistical Association, 84(406), 479–486.

2. Robust Bayesian Estimation for the Linear Model and Robustifying the Kalman Filter.

   Masreliez, C.; Martin, R. (1977). Robust bayesian estimation for the linear model and robustifying the Kalman filter. IEEE Transactions on Automatic Control, 22(3), 361–371.

3. Analysis and Improvement of the Consistency of Extended Kalman Filter based SLAM

   Huang, Guoquan P.; Mourikis, Anastasios I.; Roumeliotis, Stergios I. (2008). [IEEE 2008 IEEE International Conference on Robotics and Automation (ICRA) - Pasadena, CA, USA

(2008.05.19-2008.05.23)] 2008 IEEE International Conference on Robotics and Automation - Analysis and improvement of the consistency of extended Kalman filter based SLAM. , (), 473–479.

4. Improvement of Observer/Kalman Filter Identification (OKID) by Residual Whitening.

   Phan, M.; Horta, L. G.; Juang, J.-N.; Longman, R. W. (1995). Improvement of Observer/Kalman Filter Identification (OKID) by Residual Whitening. Journal of Vibration and Acoustics, 117(2), 232–.

5. An Adaptive Extended Kalman Filter Using Artificial Neural Networks.

   Stubberud, S.C.; Lobbia, R.N.; Owen, M. (1995). [IEEE 1995 34th IEEE Conference on Decision and Control - New Orleans, LA, USA (13-15 Dec. 1995)] Proceedings of 1995 34th IEEE Conference on Decision and Control - An adaptive extended Kalman filter using artificial neural networks. , 2(0), 1852–1856.

6. Improvement of Ensemble Kalman Filter for Improper Initial Ensembles

   Lee, Kyung-Book; Jo, Gyung-Nam; Choe, Jonggeun (2011). Improvement of Ensemble Kalman Filter for Improper Initial Ensembles. Geosystem Engineering, 14(2), 79–84.

7. Improved Kalman Filter Design for Three-Dimensional Radar Tracking.

   Seong-Taek Park, ; Jang Gyu Lee, (2001). Improved Kalman filter design for three-dimensional radar tracking. IEEE Transactions on Aerospace and Electronic Systems, 37(2), 727–739.

8. Improved Extended Kalman Filter Design for Passive Tracking.

   Weiss, H.; Moore, J. (1980). Improved extended Kalman filter design for passive tracking. , 25(4), 0–811.

9. An Improved Kalman Filter with Adaptive Estimate of Latency Probability.

   Wang, Zhao; Huang, Yulong; Zhang, Yonggang; Jia, Guangle; Chambers, Jonathon (2019). An Improved Kalman Filter with Adaptive Estimate of Latency Probability. IEEE Transactions on Circuits and Systems II: Express Briefs, (), 1–1.

10. Effective improvement of under-modeling frequency-domain Kalman filter

    Fan, Wenzhi; Chen, Kai; Lu, Jing; Tao, Jiancheng (2019). Effective improvement of under-modeling frequency-domain Kalman filter. IEEE Signal Processing Letters, (), 1–1.