B20AI013_B20AI061_A2_Kalman

Team Members

Ishaan Shrivastava - B20AI013

Vikash Yadav - B20AI061

Report on Kalman Filter for Monitoring Enemy Aircraft Position and Velocity

The Kalman filter is a recursive algorithm that uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more precise than those based on a single measurement alone. It estimates the state of a dynamic system in the presence of uncertain and noisy measurements. The main idea is to have a model of the system, and the measurements are used to update the model. The state of the system is represented by a Gaussian distribution, which is updated with every measurement. The Kalman filter uses two sets of equations: the state transition equations and the observation equations.

Part 1: Equations of Kalman Filter for the given problem

The problem statement involves monitoring the position and velocity of an enemy aircraft using a radar that takes time T to make a full rotation while scanning. The ASAM is a Kalman filter that updates the position every Δt time steps. It is given that $\Delta t < T$, and

$$k \times \Delta t = T$$

for some positive integer constant k. The state of the enemy can be modeled as a discrete dynamic system with state given by

$$S_t = \begin{bmatrix} x_t \\ v_t \end{bmatrix}$$

where v_t and x_t are the 3D velocity and position vectors at that given time t.

The acceleration control $u_t = a_t$ acting on the target is also known to an exact degree with zero uncertainty.

The equations to deduce the next corresponding values of the state are given by

$$\begin{aligned} v_{k+1} &= v_k + a_t \times \Delta t \\ x_{k+1} &= x_k + v_t \times \Delta t \end{aligned}$$

This can be generalised to the same form as the **state transition equations** in the Kalman Filter given below

$$S_{k+1} = F_k S_k + G_k u_k + \xi_k$$

, where $\xi_k \sim \mathcal{N}(0, Q_k)$ is the system noise, and valid values for F_k, G_k, and u_k boil down to:-

$$F_k = \begin{bmatrix} I_3 & \Delta t \\ 0_3 & I_3 \end{bmatrix}, G_k = \begin{bmatrix} 0_3 & 0_3 \\ 0_3 & I_3 \Delta t \end{bmatrix}, u_k = \begin{bmatrix} 0 \\ 0 \\ 0 \\ a_{k_x} \\ a_{k_y} \\ a_{k_z} \end{bmatrix}$$

Where 0_n refers to a $n \times n$ matrix of zeros.

The observation equations for the given problem are:

$$y_k = H_k S_k + \eta_k$$

where $\eta_k \sim \mathcal{N}(0, R_k)$ is the measurement noise (provided in the question itself), and the measurement matrix H_k for our problem is a simple identity transformation.

$$H_k = I_6$$

The observations in the code are noised by the addition of gaussian noise with standard deviations $\eta_x = \text{noise\_position}$ and $\eta_v = \text{noise\_velocity}$, so the covariance matrix R_k can be obtained as:

$$R_k = \begin{bmatrix} I_3 * \eta_x^2 & 0_3 \\ 0_3 & I_3 * \eta_v^2 \end{bmatrix}$$

The Kalman filter equations are:

Propagation step

$$P_{k+1|k} = F_k P_k F_k^T + Q_k$$
$$\hat{S}_{k+1|k} = F_k \hat{S}_{k|k} + G_k u_k$$

Update step

$$P_{k|k}^{-1} = P_{k|k-1}^{-1} + H_k^T R_k^{-1} H_k$$
$$K_k = P_{k|k} H_k^T R_k^{-1}$$
$$\hat{S}_{k|k} = \hat{S}_{k|k-1} + K_k(y_k - H_k \hat{S}_{k|k-1})$$

Where K_k is the kalman gain matrix which specifies the amount by which the residue between the observation and the posterior estimate, $y_k - H_k \hat{S}_{k|k}$, must be multiplied to obtain the correction term that transforms the prior estimate $\hat{S}_{k|k-1}$ into the posterior estimate $\hat{S}_{k|k}$.

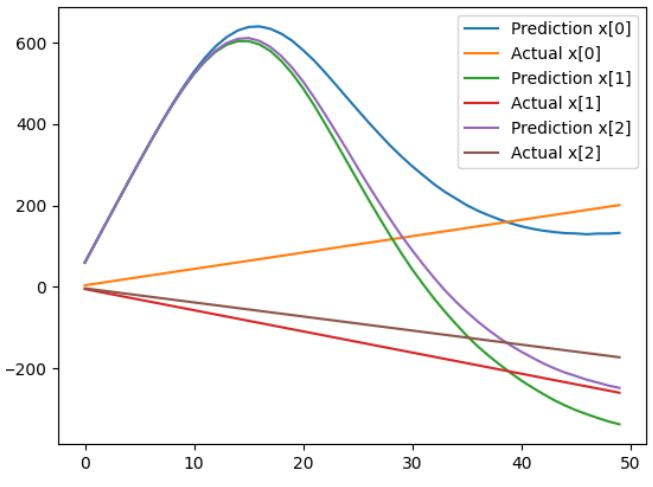
In our problem, the ASAM will **(1) Run the propagate step k times** before every reading from the RADAR allows us to update our running estimate using the **(2) update step**. This process will loop indefinitely for as long as we want to track the object and readings are available.

Here is a screenshot of the different tests being run on the model -

```
Error after test run: 91.91022388412128
Error after test run: 88.46522358045377
Error after test run: 131.79656553295678
Error after test run: 1.643155988573289e+149
Error after test run: 2.3341605348689643e+149
```

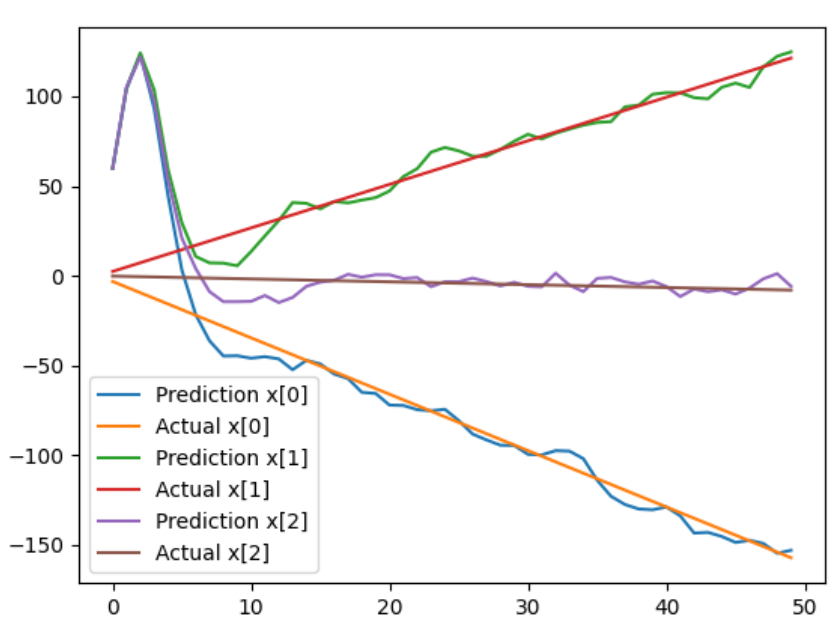
I have additionally included a new parameter β which controls the size of the covariance matrix Q_k for the noise associated with propagation. I have included plots of the kalman filter for varying values of Q_k and P_0 , with a bad initial estimate to highlight the trends, below:

$Q_k = P_0 = 0.0001 * I_6$



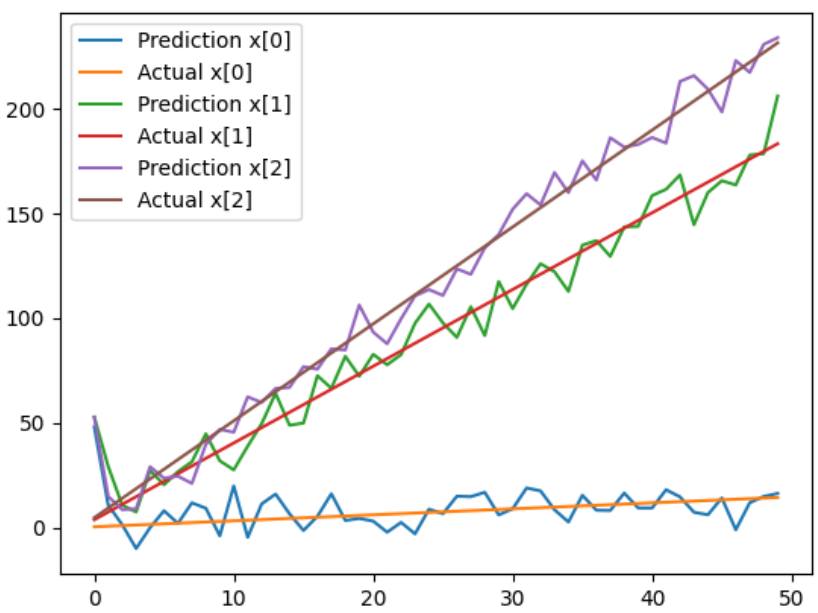

```
Error after test run: 394623.614667733
Error after test run: 283314.29360503197
Error after test run: 250955.98548485216
Error after test run: 4.998700584877458e+147
Error after test run: 9.006838334479301e+147
```

$Q_k = P_0 = 0.01 * I_6$



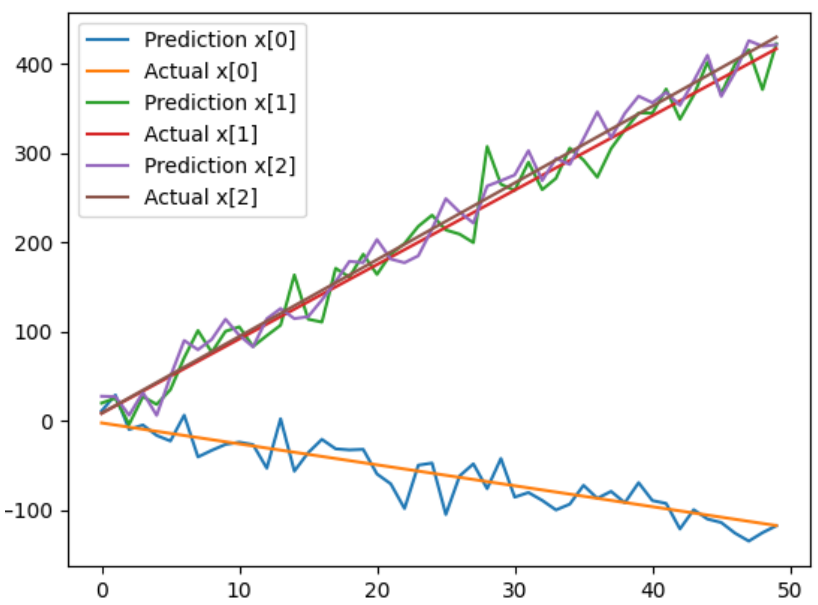

```
Error after test run: 2692.994428117097
Error after test run: 2152.2411534126595
Error after test run: 1569.9993125224803
Error after test run: 1.1192438931170447e+148
Error after test run: 2.228971999237419e+149
```

$Q_k = P_0 = 1.0 * I_6$




```
Error after test run: 341.17573987931814
Error after test run: 278.2374775934765
Error after test run: 278.32944499606776
Error after test run: 5.914697129625672e+148
Error after test run: 1.7647477243008092e+147
```

$Q_k = P_0 = 100.0 * I_6$




```
Error after test run: 1234.6549200055763
Error after test run: 1071.424794119989
Error after test run: 1337.2861508167357
Error after test run: 7.515754848816892e+148
Error after test run: 3.087499758612907e+146
```

This behaviour can be explained by considering two things:

1. The dynamic uncertainty Q_k controls how much information passes through between RADAR readings. If the uncertainty is small, propagate steps do not add much uncertainty to the state estimate in the equation $P_{k+1|k} = F_k P_{k|k} F_k^T + Q_k$. Hence, the filter is able to make stabler estimates using information from a larger number of time steps prior to the current, with the tradeoff of requiring a larger number of observations to incorporate information from observations into its estimates. This is perfectly exhibited by the plot corresponding to small β in column 1 where the predictions of the filter are initially way off but get smoother than the other two columns as time progresses. However, care should be taken to make sure that β is not too small in order to avoid the filter getting too confident in its estimates in which case the filter will take too long to adapt to information about any sharp changes in the projected trajectory of the state. This can be seen above for $\beta = 0.0001$, where the filter fails to forget the bad initial estimate even for a simple test case:
2. Conversely, a larger uncertainty means that the noise associated with propagation will bottleneck information from previous timesteps from being used in every update step as the estimate from the observation will be much less uncertain. This can be visible in the second and third columns where the estimates are very noisy due to a bias towards information obtained in the most recent update step. This means that the filters learn to fit the state of the system fairly quickly compared to the first column, however, this comes with the tradeoff of losing information with each timestep leading to noisier results in the long run.

Part 2:

Included in the `kalman.py` file. Snippets of the relevant functions have been attached here for reference as well.

```
class KalmanFilter:
    def __init__(self, noise_velocity, noise_position) -> None:
        self.nv = noise_velocity
        self.np = noise_position
        self.Hk = np.eye(6)
        self.Rk = np.concatenate(
            [
                np.concatenate([np.eye(3) * self.np**2, np.zeros_like(np.eye(3))], axis=1),
                np.concatenate([np.zeros_like(np.eye(3)), np.eye(3) * self.nv**2], axis=1),
            ],
            axis=0,
        )
        self.delta_t = 1
        self.Fk = np.concatenate(
            [
                np.concatenate([np.eye(3)       , np.eye(3) * self.delta_t], axis=1),
                np.concatenate([np.zeros([3, 3]), np.eye(3)               ], axis=1),
```

```
            ],
            axis=0
        )
        self.Gk = np.concatenate(
            [
                np.concatenate([np.zeros([3, 3]), np.zeros([3, 3])        ], axis=1),
                np.concatenate([np.zeros([3, 3]), np.eye(3) * self.delta_t], axis=1),
            ]
            , axis=0
        )
        self.beta = 100
        self.Qk = np.eye(6) * self.beta
        self.Pk = np.eye(6) * self.beta
        self.state = np.ones([6, 1]) * 10
```

```
    def input(self, observed_state:State, accel:numpy.ndarray, justUpdated:bool):
        # propagate step
        self.state = self.Fk @ self.state + self.Gk @ np.concatenate([np.zeros([3, 1]), accel.reshape([3, 1])], axis=0)
        self.Pk = self.Fk @ self.Pk @ self.Fk.T + self.Qk

        if justUpdated:
            # update step
            self.yi = np.concatenate([observed_state.position.reshape([-1, 1]), observed_state.velocity.reshape([-1, 1])], axis=0)
            self.Pk = np.linalg.inv(np.linalg.inv(self.Pk) + self.Hk.T @ np.linalg.inv(self.Rk) @ self.Hk)
            self.kalman_gain = self.Pk @ self.Hk.T @ np.linalg.inv(self.Rk)
            self.state = self.state + self.kalman_gain @ (self.yi - self.Hk @ self.state)
```

```
def get_current_estimate(self)->State:
        return State(x = np.array([si[0] for si in self.state[:3]]), v = np.array([si[0] for si in self.state[3:]]))
```

Part 3: Equations of Kalman Filter for a Falling Projectile

Consider the case of a falling projectile. It is noted that to have a velocity at a given position, the only acceleration force acting on this body is the force of gravity, but due to reaching its terminal velocity (a velocity where the air resistance cancels out all downwards acceleration), the total acceleration of the body is 0. Assuming that it will stay that way, we need to rewrite the Kalman filter equations where the state of the object is observed from the RADAR.

The main differences between part 1 and part 3 will be the state transition equations and the additional assumption of a terminal velocity v^o. Hence, acceleration of the projectile in this case is known and the state equations can be modelled with the help of the new state transition:

$$x_{k+1} = x_k + \Delta t \times v_k$$
$$v_{k+1} = v_k - \Delta t \times \left(g \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \frac{g v_k}{v^o} \right)$$

, where v^o is the scalar value of the terminal velocity and the drag coefficient for velocity is expressed as the fraction $\frac{g}{v^o}$.

Hence, the state transition equations get remodelled as:

$$S_{k+1} = F_k S_k + G_k u_k + \xi_k$$

, where $\xi_k \sim \mathcal{N}(0, Q_k)$ is the system noise, and new valid values for F_k, G_k, and u_k are:-

$$F_k = \begin{bmatrix} I_3 & \Delta t \\ 0_3 & I_3 \left(1 - \frac{g \Delta t}{v^o}\right) \end{bmatrix}, G_k = \begin{bmatrix} 0_3 & 0_3 \\ 0_3 & I_3 \Delta t \end{bmatrix}, u_k = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -g \end{bmatrix}$$

The key difference between the equations in part 1 and part 3 is that since this system has a fixed acceleration, the control input matrix u_k now becomes constant and fixed whereas in part 1 it had varying acceleration components.
Also, the state transition matrix F_k changes to incorporate a new drag term in it.