# Plateforms and Algorithms for Autonomous Driving-Module 1

2. KF and EKF

**Asif Khan Pattan**

Masters in Artificial Intelligence

University of Bologna

August 29, 2023

# Objective

The objective of the assignment is to

1. Implement a Kalman Filter using the data from the LiDAR:
   implement the formulas and define the parameters needed to run the KF
2. Implement the EKF using the data from the LiDAR and radar:
   implement the formulas and define the parameters needed to run the EKF

# KF

### 0.0.1 kf.cpp

A Kalman filter is a mathematical algorithm used for estimating the state of a dynamic system based on noisy measurements. Here's a brief explanation of the key parts of the code:

1. KalmanFilter::Predict(): This function predicts the next state of the system using the prediction equations of the Kalman filter.

   - 'x_' is the state vector of the system.
   - 'F_' is the state transition matrix that relates the current state to the next state.
   - 'P_' is the covariance matrix of the state, representing the uncertainty in the state estimation.
   - 'Q_' is the process noise covariance matrix, accounting for the uncertainty in the process model.
   The prediction steps include:
   - Updating the state estimate: 'x_ = F_ * x_'.
   - Updating the covariance estimate: 'P_ = F_ * P_ * F_.transpose() + Q_'.

2. KalmanFilter::Update(const VectorXd &z): This function updates the state estimate based on a new measurement using the Kalman filter update equations.

   - 'z' is the measurement vector.
   - 'H_' is the measurement matrix that relates the state to the measurements.
   - 'R_' is the measurement noise covariance matrix, representing the uncertainty in the measurements.
   The update steps include:
   - Calculating the measurement residual: 'y = z - H_ * x_'.
   - Calculating the innovation covariance: 'S = H_ * P_ * H_.transpose() + R_'.
   - Calculating the Kalman gain: 'K = P_ * H_.transpose() * S.inverse()'.
   - Updating the state estimate: 'x_ = x_ + K * y'.
   - Updating the covariance estimate: 'P_ = (I - K * H_) * P_'.

   In summary, the Predict step estimates the next state of the system based on the previous state and the system dynamics, while the Update step corrects the prediction using the measurement information to provide a more accurate state estimate. Kalman filters are widely used in various applications, such as tracking, navigation, and control systems, to improve the accuracy of state estimation in the presence of noise and uncertainty.

### 0.0.2 tracking.cpp

1. Initializing State Covariance Matrix 'P_':
   Here, the state covariance matrix P_ is initialized with a diagonal matrix. Diagonal entries

represent the variances of the corresponding state variables. The non-diagonal elements are set to 0, implying no initial correlations between state variables.

2. Initializing Transition Matrix 'F_':
   The transition matrix 'F_' is initialized as a 4x4 matrix representing the state transition equations. The matrix describes how the state evolves over time. In this case, it's a simple constant velocity model where the first two rows correspond to position updates, and the last two rows correspond to velocity updates.

3. Calculating Process Covariance Matrix 'Q_':
   The process covariance matrix Q_ captures the uncertainty in the process model. It's a measure of how much the actual state can deviate from the predicted state due to process noise. In this case, it's derived from the constant acceleration motion model. The matrix is computed using the time step dt and acceleration noise terms noise_ax and noise_ay.

4. Updating Transition Matrix 'F_' with Time Integration:
   The transition matrix F_ is updated to integrate time into the state transition equations. This step reflects the relationship between position, velocity, and time in the state update equations. The positions are updated based on the current velocities and the time step dt.

   These solutions ensure that the Kalman filter is properly initialized and updated for accurate state estimation, considering the dynamics of the system, measurement noise, and process noise.

### 0.0.3 Results:

Figure 1 shows the output of KF with default values where round truth x, y data in the green, sensor measurement output in the orange and KF calculations in blue.

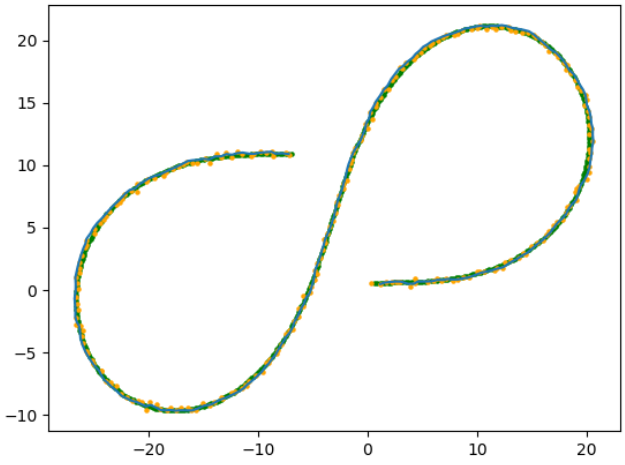


Figure 1: KF output(default values)

1. When Sensor Noise is very low ie R_ = [0.1 0 0 0.1]:

   When the sensor noise values are low the KF estimates are perfectly in alignment with the ground truth values as shown in Figure 2.

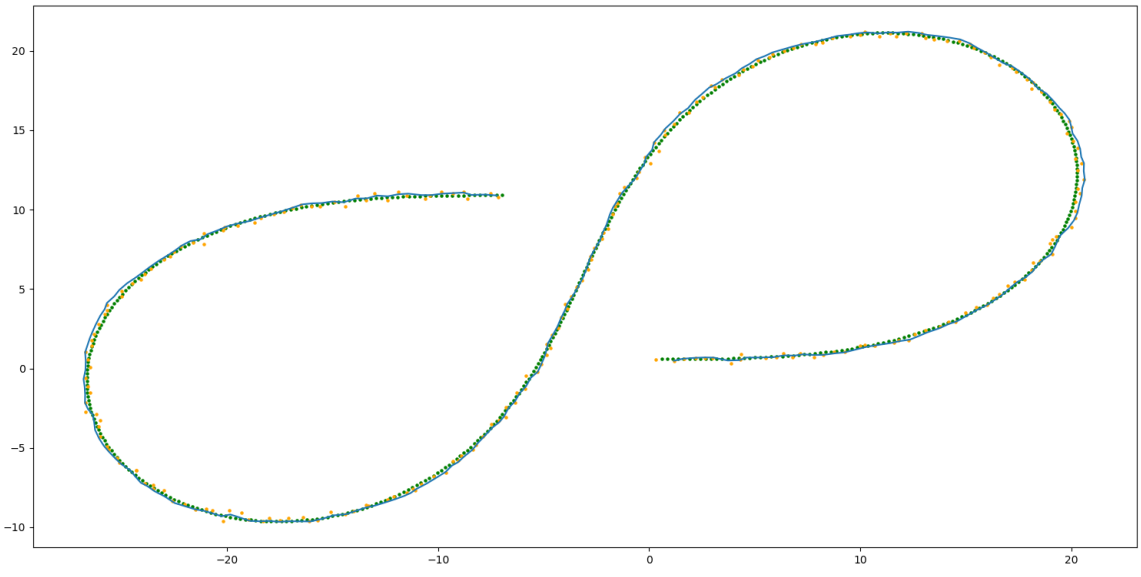


Figure 2: KF output(Low sensor noise)

2. When Sensor Noise is high ie R_ = [1.0 0 0 1.0]:

   When the sensor noise values are high the KF estiamtes are farther from the ground truth/measurement values but follows the path with a disparity as shown in figure 3
   High sensor noise implies that the measurements from the sensor are less reliable and have a higher degree of uncertainty. The Kalman filter will assign less weight to the measurements due to their higher uncertainty, causing the estimated state to rely more on the predicted state. The estimated trajectory might become smoother but less responsive to changes in measurements, resulting in slower adjustments to the true state.

When Motion noise is high:

When the noise_ax and noise_ay values are set to 30, the predictions of KF are close to that of ground truth/mesurement values but overshoots more often due to higer acceleration noise as shown in figure 4
With high motion noise (acceleration noise), the filter becomes more sensitive to sudden changes in the state of the tracked object. The filter might exhibit faster responses to changes in motion, leading to quicker adjustments in the estimated state. However, overly high motion noise can also lead to instability and erratic behavior, causing the filter to overreact to noise and produce less accurate results.

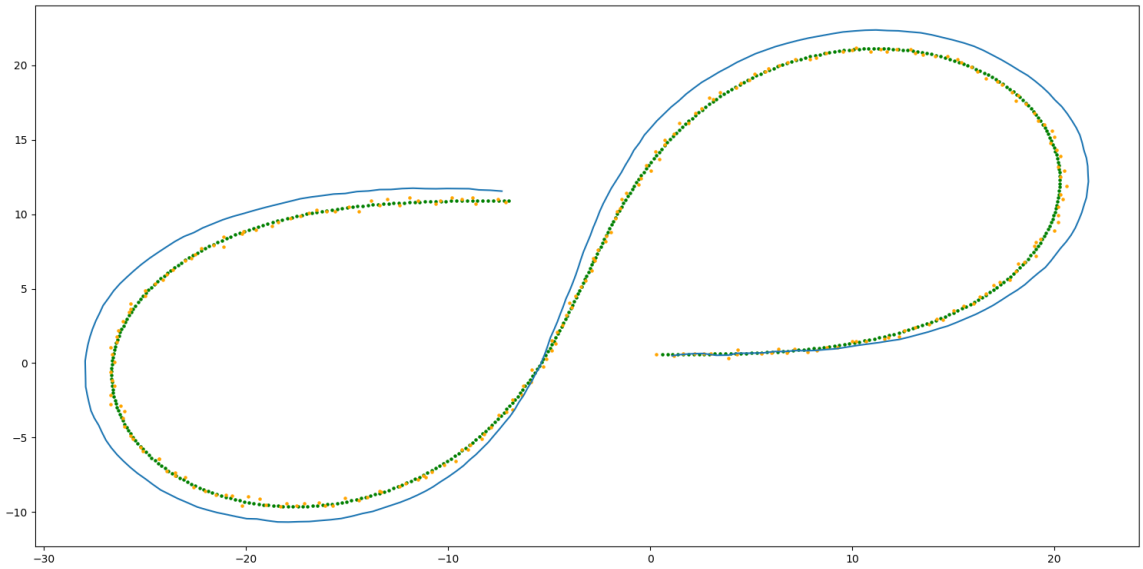


Figure 3: KF output(High sensor noise)

## EKF

**Results:**

1. EKF output with default values:

   As shown in figure 5, the predictions from the sensor fusion of Extended kalman filter are mostly overlapping to that of ground truth.
   The 1st plot shows the ground truth and predicted values while the plots 2 and 3 show the Xrms and Yrms.

2. EKF output with high LiDAR sensor Noise:

   Figure 6 shows the plots when laser noise is set to 1 instead of 0.0225 and radar left unchanged.
   Increasing LIDAR sensor noise to a higher value (e.g., 1.0) will cause the EKF to assign less weight to LIDAR measurements. The filter will become more reliant on the predicted state, leading to smoother but potentially less accurate trajectory estimates. The trajectory might exhibit reduced responsiveness to LIDAR measurements, which can cause the filter to perform less well in scenarios where LIDAR measurements provide accurate information.

3. EKF output with high LiDAR and radar sensor Noise:

   Figure **??** shows the plots when both laser sensor noise and radar sensor noise is set to 1 instead of default values.
   Increasing noise levels for both LIDAR and radar measurements will lead to the filter considering measurements from both sensors as less reliable. The EKF will tend to trust its predicted state more, leading to smoother trajectory estimates. The filter's ability to

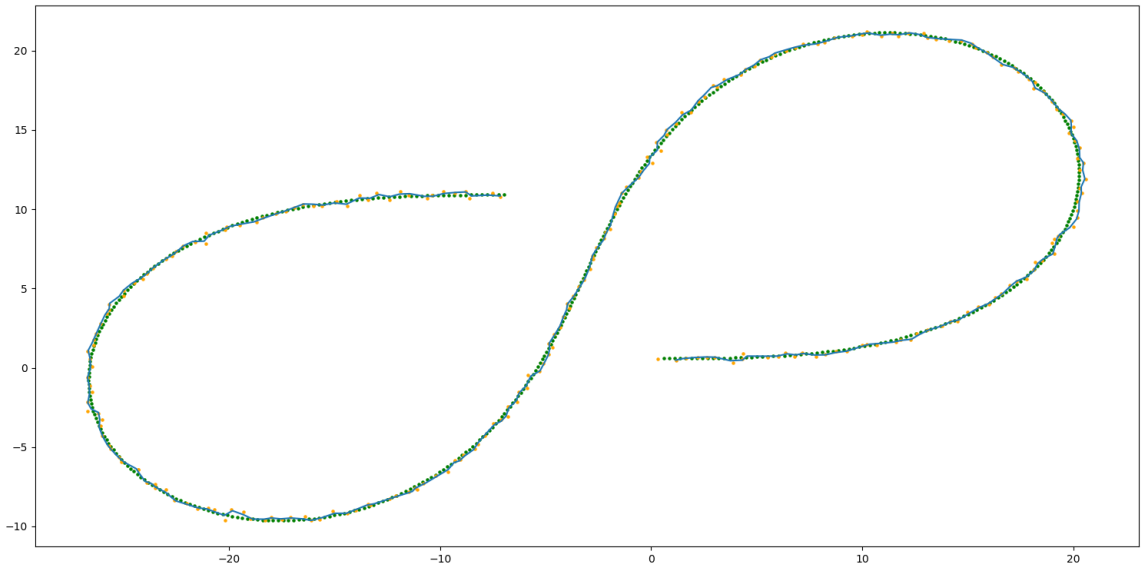


Figure 4: KF output(High motion noise)

differentiate between accurate and noisy measurements will decrease, potentially resulting in less accurate state estimates.

4. Increasing noise_ax and noise_ay to Higher Values:

   noise_ax and noise_ay are set to 30 and the output plots can be seen in figur 8 where the xrms and yrms plots vary(stay higher) compared to that of low process noise. Increasing the values of noise_ax and noise_ay will lead to higher assumed process noise in the prediction step of the EKF as shown in The filter will respond more readily to changes in motion, leading to more sensitive trajectory adjustments. However, excessively high noise_ax and noise_ay values can lead to instability and erratic behavior. The filter might overreact to noise, causing the estimated trajectory to diverge from the true state.

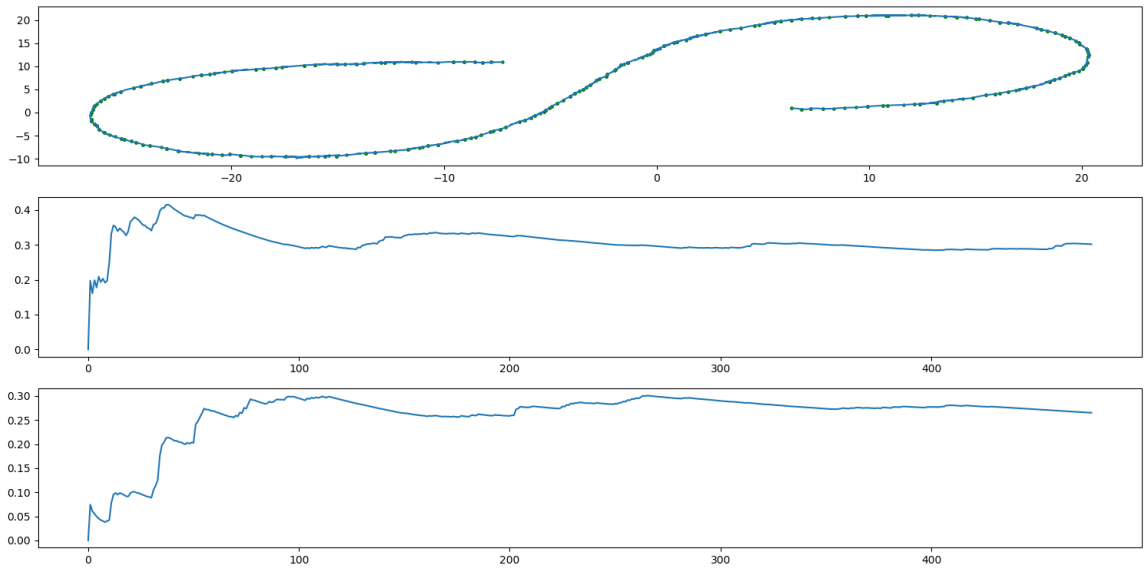


Figure 5: EKF output(default)

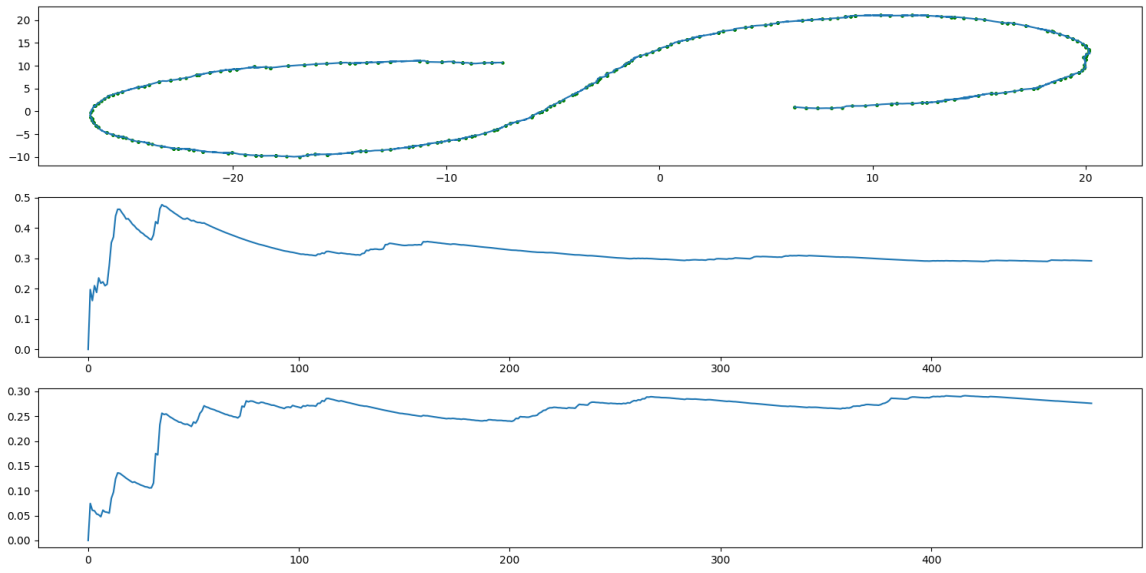


Figure 6: EKF output(High Lidar sensor noise with default radar sensor noise)

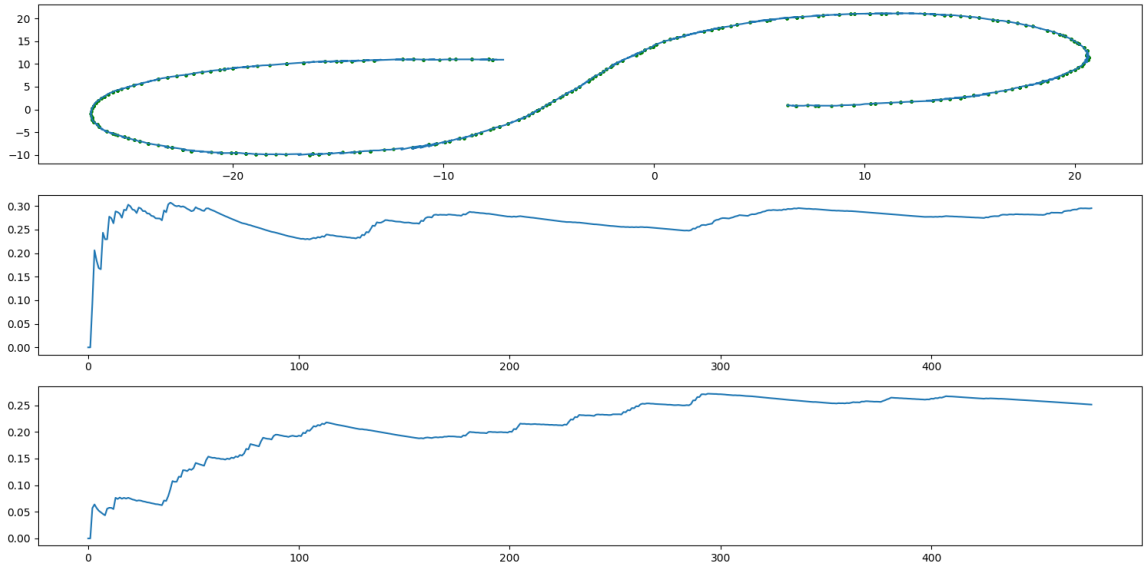

Figure 7: EKF output(High LiDAR and radar sensor noise)

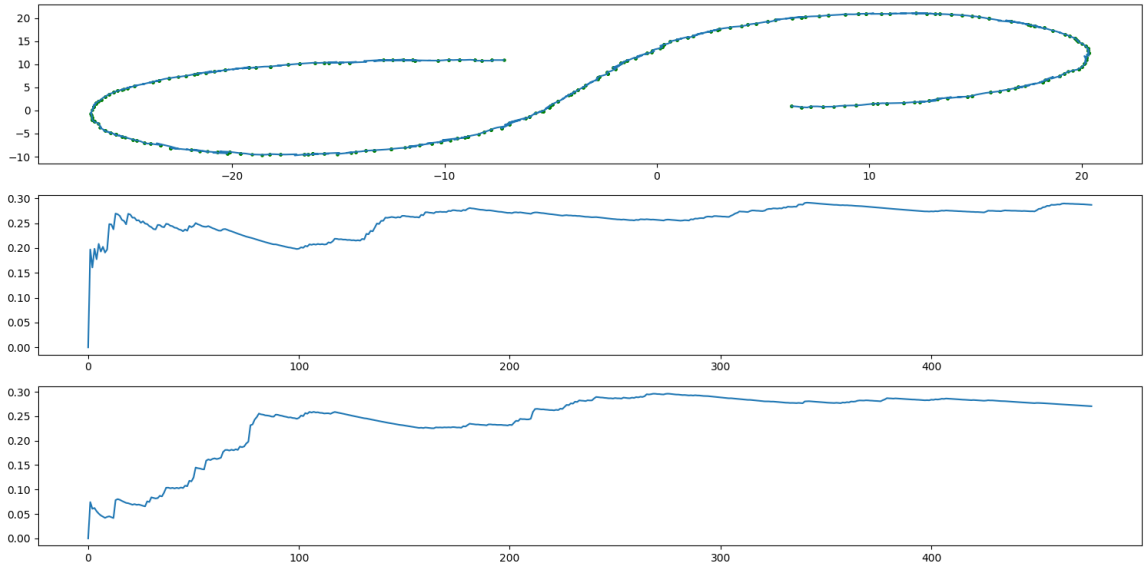

Figure 8: EKF output(High process noise)