

**Single Object Tracking Performance Analysis
For Kalman,Extended Kalman,Alpha-Beta
filters and LSTM**

Presented by:

Bheeshmaraya Bheemasamudra Vruksharaj

7206909

Supervisor: Prof.Dr.Andreas Becker

A research thesis presented for the degree of Master of
Engineering

Faculty of Information Technology

Fachhochschule Dortmund

Germany

May 2021

Contents

Bibliography	1
1 Introduction	4
1.1 Motivation	4
1.2 Literature overview	5
1.3 Outline	6
1.4 Problem statement	6
2 Alpha Beta filter	7
2.1 Filter equations	7
3 Kalman Filter	9
3.1 Prediction	9
3.2 Kalman Gain	10
3.3 Update step	11
4 Extended Kalman Filter	12
4.1 Prediction	12
4.2 Kalman Gain	13
4.3 Update Step	14
5 Long short-term Memory	15
5.1 Introduction	15
5.2 LSTM basics	15
6 Data preparation	18
6.1 Matlab script for data generation	18

CONTENTS

7	Implementation	21
7.1	Alpha beta filter	21
7.2	Kalman filter	22
7.3	Extended Kalman Filter	23
7.4	LSTM	24
8	Results	27
8.1	Performance analysis for less maneuvers tracks	28
8.2	Performance analysis for more maneuvers tracks	30
9	Future scope	32
	Bibliography	34

List of Figures

5.1	LSTM unit with 3 gates [3]	16
5.2	LSTM unit forget gate [3]	16
5.3	LSTM unit input gate [3]	16
5.4	LSTM update state [3]	17
5.5	LSTM output state [3]	17
6.1	Example1 track generated from matlab	20
6.2	Example2 track generated from matlab	20
7.1	LSTM model	26
8.1	Track of less Maneuvers with $M_{rmse} = 3.64$	29
8.2	Track of more Maneuvers with $M_{rmse} = 3.79$	31

Chapter 1

Introduction

1.1 Motivation

Object tracking is well-studied topic with many different approaches to solve the challenges it poses. With increase in safety and comfort functions added to cars the use of object tracking is increased. Ensuring safety is a top priority for autonomous driving and advanced driver assistance systems (ADAS). To promise a high degree of safety, the ability to perceive surrounding situations and predict their development in the future is critical. The vehicle on driving encounters various types of dynamic traffic participants such as car, motorbike, and pedestrian which could be a potential threat to safe driving. To avoid an accident, the system should be able to analyze the pattern in their motion and predict the future trajectories in advance. If the system predicts where the surrounding vehicles are heading in the near future, the vehicle can plan its driving path in response to the situation to come such that the probability of collision is minimized. Object tracking is also used in many applications like planning route, surveillance etc, which can be done using information provided by different sensors.

Object tracking can be achieved using camera by change in position of object in consecutive frames, LIDAR point clouds and radar point cloud can also be used. In the current work radar point cloud data is used for object tracking. Each of the sensors have their own pros and cons based on their working principle and

constraints of operation. To achieve single object tracking, classical approaches of object tracking that is Kalman filter, Extended kalman filter, $\alpha - \beta$ filter is used and recently machine learning techniques like RNN (Recurrent neural network), LSTM (Long short term memory) which are sequence based prediction techniques are also being used. Since there are different possible solutions there is need to understand which of these methods could be used for better performance and scientifically differentiate them. And conclude a good approach that fits for object tracking using radar in different scenarios. Efforts are made in such a direction in this thesis.

1.2 Literature overview

There are many proposed solutions of object tracking and some of the relevant solutions are quoted here. The work [7] deals with object tracking and evaluation of performance between the different object tracking methods like Kalman filter, KNN (K nearest Neighbour) and LSTM (Long-Short Term Memory), the data is based on series of image frames from different data set and in [14] the tracking is done by sensor fusion of camera and mmwave radar which uses both of them for more accurate tracking details using LSTM. In [6] there is interesting method to improve extended kalman filter by introducing LSTM for prediction step in pose regularization application.

The work [19] investigates the trajectory tracking problem of a flight vehicle performing complex maneuvers. A learnable Extended Kalman Filtering (L-EKF) method is proposed. First, two recurrent neural networks, named Input Modification Network and Gain Modification Network, are designed to identify the model inaccuracy and compensate for the estimation error of the EKF. Then, the L-EKF algorithm is proposed by embedding the two networks into the EKF algorithm. Then, the proposed L-EKF method is applied to a trajectory tracking problem of a gliding vehicle with complex maneuvers. Another possibility of object tracking using camera is video based which is discussed in [8]. Similar to [7] there is a comparison of performance of object tracking between Kalman filter and RNN (Recurrent neural network) in [1]. The LSTM model used in current work is developed in reference to [7].

1.3 Outline

The work is divided into 9 chapters. Chapter one deals with motivation, literature review and also gives an insight of problem statement dealt in this thesis. The second to fifth chapters provide an overview of Alpha-beta filter, Kalman filter, Extended kalman filter and the LSTM network used in the current work. Sixth and seven chapters gives details of data preparation and implementation respectively and finally in last two chapters we discuss about the results, feature scope.

1.4 Problem statement

Radar based object tracking which can be used in different environment conditions has its advantages and some disadvantages as well when compared to camera based object tracking. Based on the information the radar gives which is range (Distance of the object detected), range rate (relative velocity of detected object) and azimuth angle (Angle of detected object) goal is to estimate the nearest position object in comparison ground truth in Cartesian coordinate system based on available measurements.

The measurements data of radar detection for object tracking and ground truth data is generated by Matlab simulation, we can see the implementation details of it in chapter 6. To give an overview of the data, it is a series of iterations (tracks) each with certain number of points which are noisy (to simulate the real-time measurements). We also have ground truth for similar tracks. The noisy measurements are to be used to predict the next real position of the object which is the ground truth. The data contains tracks of both linear and non-linear object movement.

The linear motion of object can be better estimated by kalman filter. To handle non-linearity in some tracks there is a need for a better method than just a kalman like Extended kalman filter and LSTM based deep neural network. The detailed explanation and analysis is done on all these methods in next chapters.

Chapter 2

Alpha Beta filter

Alpha-Beta filtering algorithm is a simple, easy-to-implement, constant-gain filtering method. This method has been widely used in the design of tracking filters for various systems. The biggest advantage is that the calculation is small. It is a simplified scheme of Kalman tracking filter. Its principal advantage is that it does not require a detailed system model as we see in the kalman filter later.

The basic idea of the Alpha-Beta filtering algorithm is when tracking the target, there will be a target position prediction value and a target position measurement value in the case of the target position. In the case of the target velocity, there will be a target velocity at the previous time and it is assumed that the target is moving at a constant speed.

2.1 Filter equations

There are four state variables that are to be predicted that is position and velocity in two directions of Cartesian coordinate system. The position and velocity is predicted using below equation

$$\hat{X}_n = X_{n-1} + dt * V_{n-1} \quad (2.1)$$

$$\hat{V}_n = V_{n-1} \quad (2.2)$$

dt is the time difference between consecutive measurements. X_{n-1} and V_{n-1} are the position and velocity in previous (n-1) time stamp. The prediction done in eqn 2.1 will have error when compared to measurement and that error is calculated as difference and it used to correct the final estimation using below equations

$$\hat{e}_n = X_n - \hat{X}_n \quad (2.3)$$

Using this error the final estimations are calculated using eqn 2.4 and eqn 2.5

$$X_n = \hat{X}_n + \alpha * \hat{e}_n \quad (2.4)$$

$$V_n = \hat{V}_n + (\beta/dt) * \hat{e}_n \quad (2.5)$$

The corrections can be considered in small steps along an estimate of the gradient direction. As these adjustments accumulate, error in the state estimates is reduced. For convergence and stability, the values of the alpha and beta multipliers should be positive and small. Usually alpha and beta fall in the range $0 < \alpha < 1$ and $0 < \beta < 2$. All the above equations are applied for both x and y Cartesian coordinates. The values for alpha and beta are decided based on experiment trails. Larger values provide good transient changes while tracking a object and smaller values gives reduction in noise of estimates.

Chapter 3

Kalman Filter

Kalman filter is algorithm that uses series of measurements which are observed over time having noise and uncertainties to estimate more accurate values of certain variables nearer to ground truth.

The steps involved in final estimation is as follows

- Predict current state using previous state and the dynamic equation.
- Calculate Kalman Gain
- Update the current state using prediction step, kalman gain and measurements.

3.1 Prediction

Prediction step is where the state variables are estimated based on dynamic model and in vehicle tracking the dynamic model is $s = s_{n-1} + v_{n-1} * t + 0.5at^2$, Where s and s_{n-1} are current position and previous position respectively, v_{n-1} is previous velocity. t is delta time between 2 consecutive measurements. a is acceleration and the state vector is $X_n = \begin{bmatrix} x & v_x & y & v_y \end{bmatrix}$

where the x, y are position coordinates of object detected and v_x, v_y are the velocity components in x and y coordinates respectively.

The prediction of next state is given by equation

$$\hat{X}_n = A * X_{n-1} + w_n \quad (3.1)$$

A is the transition matrix and it is given by

$$\begin{bmatrix} 1 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

But in reality vehicle does not follow constant velocity because of the acceleration and this factor is added as noise w_n which will be considered in calculation of Q matrix or process noise in eqn 3.2 . X_{n-1} is the previous state vector.

Predicted state error co-variance matrix \hat{P}_n which tells about estimate uncertainties of current state prediction and is given by eqn(3.2). P_{n-1} is previous state error co-variance matrix. Q is called process noise co-variance which is calculated by $B\sigma B^T$. Value of Q decides the level of uncertainty in the dynamic model.

$$\hat{P}_n = AP_{n-1}A^T + Q \quad (3.2)$$

σ is possible co-variance is acceleration in x and y direction i.e $\begin{bmatrix} \sigma_{ax}^2 & 0 \\ 0 & \sigma_{ay}^2 \end{bmatrix}$

B matrix is obtained based on dynamic model that is given as below

$$\begin{bmatrix} 0.5dt^2 & 0 \\ dt & 0 \\ 0 & 0.5dt^2 \\ 0 & dt \end{bmatrix}$$

3.2 Kalman Gain

Kalman gain decides the factor by which prediction and measurement is to be considered for final estimate of data point. It is the relative weight given to the measurements and current state estimate, and can be "tuned" to achieve a particular performance. With a high gain, the filter places more weight on the most recent measurements, and thus follows them more responsively.

Kalman gain is calculated by the below equation

$$KG = \frac{\hat{P}_n H}{H \hat{P}_n H^T + R} \quad (3.3)$$

Here H is observation matrix which transforms state space to measurement space. For example, the digital electric thermometer measures the electric cur-

rent, while the system state is the temperature. There is a need for a transformation of the system state (input) to the measurement (output). In present problem the H is given by **Identity matrix of size 4** which means the measured and required system state are same because radar measurements are converted to Cartesian coordinates in the form of state vector.

R is co-variance matrix of measurement uncertainty which gives the possible error in measurement which can be tuned like Q matrix since we have 4 inputs so the size of the R matrix is 4×4 .

3.3 Update step

In this step final estimation is calculated based on prediction done and the measurement obtained including kalman Gain. The update equations for state vector and estimate co-variance is give as below

$$X_n = \hat{X}_n + KG(y - H\hat{X}_n) \quad (3.4)$$

X_n is final current state vector , KG is kalman gain, y is obtained measurement. \hat{X}_n is prediction done in predict step. H is the observation matrix. The final estimate error co-variance matrix is give by below equation

$$P_n = (1 - KG * H)\hat{P}_n \quad (3.5)$$

Chapter 4

Extended Kalman Filter

When the system has nonlinear dynamic model where it is not able to define either the process model with multiplication of vectors and matrices as in eqn 3.1 .The extended Kalman filter provides a tool for dealing with such nonlinear models in an efficient way.

The extended Kalman filter can be termed as a nonlinear version of the Kalman filter that linearize the models about a current state.

4.1 Prediction

The prediction for state vector is given by 4.1 equation and constant velocity model is followed and change in acceleration will be considered as process noise in eqn 4.2 with state vector being $\begin{bmatrix} x & y & v_x & v_y \end{bmatrix}$

$$\hat{X}_n = f(X_{n-1}) + W_n = AX_{n-1} + W_n \quad (4.1)$$

The equations are similar to kalman filter equations with difference in handling of measurements since the relation between measurement and state elements is non-linear. f is a function of non-linear motion model.

A is transition matrix $\begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Predicted state error Co-variance matrix is similar to kalman filter and is given by eqn 4.2.

$$\hat{P}_n = AP_{n-1}A^T + Q \quad (4.2)$$

process noise co-variance Q is given by $B\sigma B^T$. σ is possible variance is acceleration in x and y direction i.e $\begin{bmatrix} \sigma_{ax}^2 & 0 \\ 0 & \sigma_{ay}^2 \end{bmatrix}$ and B is given by $\begin{bmatrix} 0.5dt^2 & 0 \\ 0 & 0.5dt^2 \\ dt & 0 \\ 0 & dt \end{bmatrix}$

4.2 Kalman Gain

Kalman gain is calculated using equation 4.3.

$$KG = \frac{\hat{P}_n H}{H \hat{P}_n H^T + R} \quad (4.3)$$

The measurements here are in the form of range(ρ), azimuth angle (ϕ) and range rate ($\dot{\rho}$), are in non-linear form of state variables as shown in below

$$h(x) = \begin{bmatrix} (\rho) \\ (\phi) \\ (\dot{\rho}) \end{bmatrix} = \begin{bmatrix} \sqrt{x^2 + y^2} \\ \arctan(x/y) \\ \frac{xv_x + yv_y}{\sqrt{x^2 + y^2}} \end{bmatrix} \quad (4.4)$$

The H matrix in equation 4.3 is the Jacobian of matrix $h(x)$ which is given by the eqn 4.5

$$\begin{bmatrix} \frac{\partial \rho}{\partial x} & \frac{\partial \rho}{\partial y} & \frac{\partial \rho}{\partial v_x} & \frac{\partial \rho}{\partial v_y} \\ \frac{\partial \phi}{\partial x} & \frac{\partial \phi}{\partial y} & \frac{\partial \phi}{\partial v_x} & \frac{\partial \phi}{\partial v_y} \\ \frac{\partial \dot{\rho}}{\partial x} & \frac{\partial \dot{\rho}}{\partial y} & \frac{\partial \dot{\rho}}{\partial v_x} & \frac{\partial \dot{\rho}}{\partial v_y} \end{bmatrix} \quad (4.5)$$

$$\begin{bmatrix} \frac{x}{\sqrt{x^2 + y^2}} & \frac{y}{\sqrt{x^2 + y^2}} & 0 & 0 \\ \frac{-y}{\sqrt{x^2 + y^2}} & \frac{x}{\sqrt{x^2 + y^2}} & 0 & 0 \\ \frac{y(v_x y - v_y x)}{(x^2 + y^2)^{3/2}} & \frac{x(v_y x - v_x y)}{(x^2 + y^2)^{3/2}} & \frac{x}{\sqrt{x^2 + y^2}} & \frac{y}{\sqrt{x^2 + y^2}} \end{bmatrix} \quad (4.6)$$

In Eqn 4.3 R is the measurement noise co-variance matrix which tells the possible variance in error of the range(ρ), azimuth angle(ϕ) and in range rate($\dot{\rho}$) measurements and is supplied usually by radar supplier. Here it is 3x3 matrix. It

is calculated manually for our data and tuned for better performance.

4.3 Update Step

In update step using kalman gain, prediction and measurement the final posterior of state vector is calculated as in by Eqn 4.7

$$X_n = \hat{X}_n + KG(y - h(x)) \quad (4.7)$$

As seen earlier $h(x)$ is raw measurements in polar coordinate form.

The final estimate covariance matrix remains as in Eqn 4.2

$$P_n = (1 - KG * H)\hat{P}_n \quad (4.8)$$

Here H is jacobian as in Eqn 4.6.

Chapter 5

Long short-term Memory

5.1 Introduction

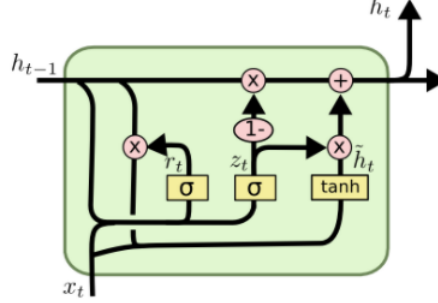
The LSTM performance is evaluated for the single object tracking. Unlike Kalman filter LSTMs make no assumption about motion model of the objects but try to learn motion model from the data given in terms of tracks. In essence it should work for both linear and non-linear motion. Because of recurrent nature LSTM can use the window of previous history to get to know future position. LSTM network is a type of recurrent neural network capable of learning order dependence in sequence prediction problems. LSTMs are majorly used in machine translation, speech recognition where data is sequence dependent. In current approach a sequence of positions is passed from noisy measurement to LSTM network, on which future position is predicted and compared with ground truth.

5.2 LSTM basics

All recurrent neural networks have the form of chain of repeating models of neural network. Since LSTM is advanced version of RNN it also has repeating model like structure instead of having one neural network layer, it has multiple layers as shown in fig 5.1

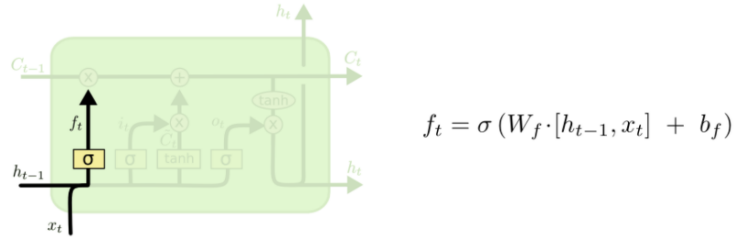
The important point in LSTM is the cell state which runs from input to output with some minor linear interactions. The LSTM does have ability to remove or add information to cell based on controls called gates.

Figure 5.1: LSTM unit with 3 gates [3]



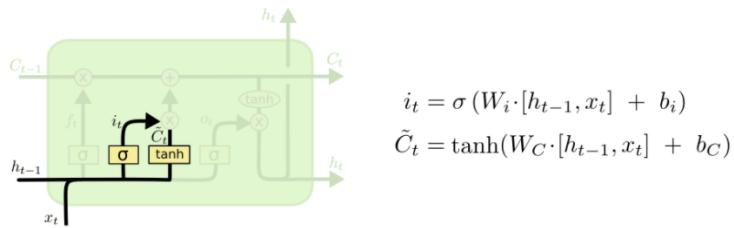
To decide what to remove from cell state forget gate is used as shown in the fig 5.2

Figure 5.2: LSTM unit forget gate [3]



Next step is to decide what information is to be stored in the cell state, it has two parts first is sigmoid layer which is "input gate", it decides which values needs to be updated. Next is tanh layer which creates a vector of new candidates (fig 5.3). And eventually these two are combined to update the cell state.

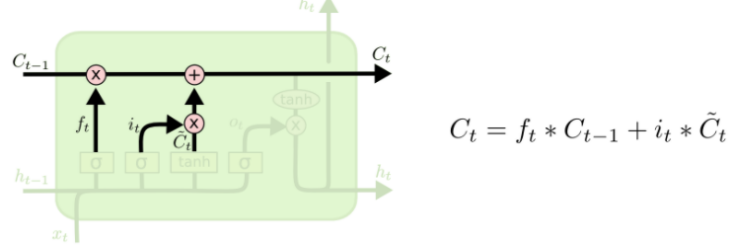
Figure 5.3: LSTM unit input gate [3]



Once values to be updated are ready, the task is to remove old and update with new values which is shown in 5.4. Old state is multiplied by forget gate and new state is stored in cell state.

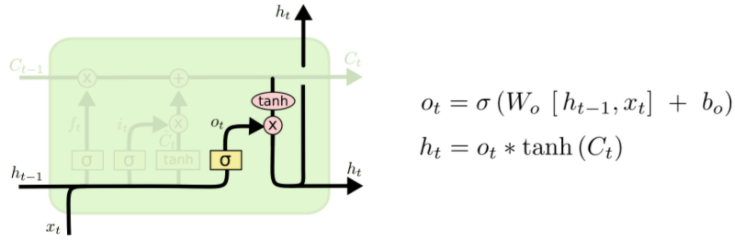
Finally the output is to be decided which is based on the cell state. And final

Figure 5.4: LSTM update state [3]



output is given as in Fig 5.5. In current model 256 such LSTM units were used and sequence length of 5 is used. These are hyper parameters which can change according to model and data which we will discuss in chapter 7.

Figure 5.5: LSTM output state [3]



The details of symbols used in the equation are explained below:

$i_t \rightarrow$ represents input gate

$f_t \rightarrow$ represents forget gate

$o_t \rightarrow$ represents output gate

$\sigma \rightarrow$ represents sigmoid function

$W_x \rightarrow$ represents weight of respective gate(x) neurons

$h_{t-1} \rightarrow$ represents output from previous LSTM blocks (at time stamp (t-1))

$x_t \rightarrow$ represents input at current time-stamp

$b_x \rightarrow$ represents biases of respective(x) gates

$C_t \rightarrow$ represents the cell state

$C_{t-1} \rightarrow$ represents the previous cell state (at time stamp (t-1))

$\tilde{C}_t \rightarrow$ represents the new cell state candidate)

Chapter 6

Data preparation

For the performance analysis of object tracking radar point cloud data is used. The data is generated from Matlab script, which gives multiple tracks or iterations and each track consist of 101 points which simulates position of single vehicle in time.

The matlab script gives the measurement and ground truth values for range, azimuth angle, range rate and corresponding Cartesian coordinate values that is position in x direction, position in y direction , velocity in x direction and velocity in y direction. So the .csv file generated will contain 14 values in total for each point generated in a track or iteration.

6.1 Matlab script for data generation

The script is available in matlab file called 'targetSimulation.m'. 'targetSimulation' class has various methods to generate the required tracks and possibility to set required parameters like number of points in each iteration, time interval between 2 points generated, randomness in values for ground truth and measurement for range, range rate, azimuth angle and acceleration of target. *getData()* function in *targetSimulation.m* does the work of generating ground-truth and measurements.

The *getData()* function using the number of points per iteration as decided above generates the ground-truth using *getGroundtruth()* function. This function makes sure it generates increasing ,decreasing and no acceleration values

within a iteration of 101 points. To control randomness a parameter *sigma_gt_a* is used as show below for each track

```
obj.groundTruth.a(obj.t < P(1)*obj.Tmax) = 0.2;
obj.groundTruth.a(obj.t > P(2)*obj.Tmax) = -0.2;
obj.groundTruth.a(obj.t > P(3)*obj.Tmax) = 0;

obj.groundTruth.a = obj.groundTruth.a + randn(size(obj.t))*obj.settings.sigma_gt_a;
```

With the acceleration values, the range rate is calculated by just integrating with respect to time and again integrating range rate ,the range is obtained. Azimuth angle is generated similarly to acceleration which in-turn is used to get position and velocities in Cartesian form using sine and cosine angles as shown in below code snippet:

```
obj.groundTruth.phidotdot(obj.t < P(1)*obj.Tmax) = 0.0001;
obj.groundTruth.phidotdot(obj.t > P(2)*obj.Tmax) = -0.0001;
obj.groundTruth.phidotdot(obj.t > P(3)*obj.Tmax) = 0;
obj.groundTruth.phidotdot = obj.groundTruth.phidotdot ...
    + randn(size(obj.t))*obj.settings.sigma_gt_phi_dot_dot;
obj.groundTruth.phidot = cumsum(obj.groundTruth.phidotdot)*obj.dt;

obj.groundTruth.phi = cumsum(obj.groundTruth.phidot)*obj.dt;
obj.groundTruth.x = obj.groundTruth.r .* cos(obj.groundTruth.phi);
obj.groundTruth.y = obj.groundTruth.r .* sin(obj.groundTruth.phi);
obj.groundTruth.dx = obj.groundTruth.rr .* cos(obj.groundTruth.phi);
obj.groundTruth.dy = obj.groundTruth.rr .* sin(obj.groundTruth.phi);
```

For generating measurements a white noise is added for ground truth range, range rate, acceleration and azimuth angle. And using these Cartesian coordinates are generated. The randomness is controlled by parameters of each of them as shown below in *getMeasurements()* function of the *targetSimulation* class.

```
function obj = getMeasurement(obj)
    obj.measurement.r = obj.groundTruth.r + randn(size(obj.groundTruth.r))*obj.settings.sigma_m_r;
    obj.measurement.rr = obj.groundTruth.rr + randn(size(obj.groundTruth.rr))*obj.settings.sigma_m_rr;
    obj.measurement.a = obj.groundTruth.a + randn(size(obj.groundTruth.a))*obj.settings.sigma_m_a;
    obj.measurement.phi = obj.groundTruth.phi + randn(size(obj.groundTruth.phi))*obj.settings.sigma_m_phi;
    obj.measurement.x = (obj.measurement.r*cos(obj.measurement.phi));
    obj.measurement.y = (obj.measurement.r*sin(obj.measurement.phi));
    obj.measurement.dx = (obj.measurement.rr*cos(obj.measurement.phi));
    obj.measurement.dy = (obj.measurement.rr*sin(obj.measurement.phi));
```

Many tracks with required randomness for linear and non-linear motion of vehicle can be simulated. An example of track with ground truth and corresponding measurements simulated from matlab is show figures 6.1 and 6.2

Figure 6.1: Example1 track generated from matlab

A track with more maneuvers(non-linear)

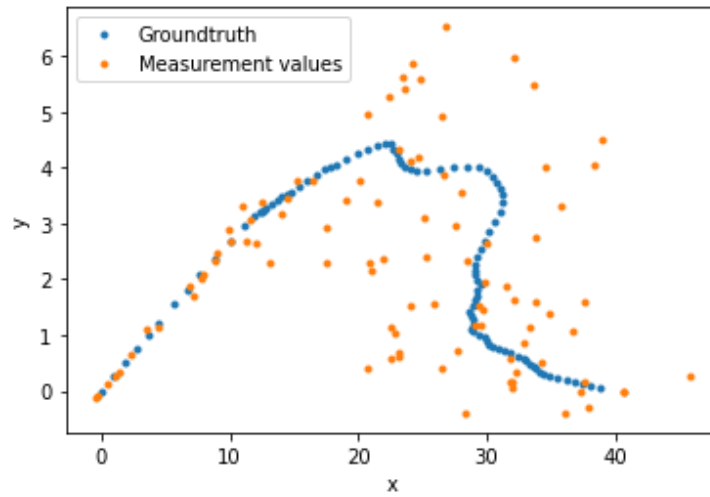
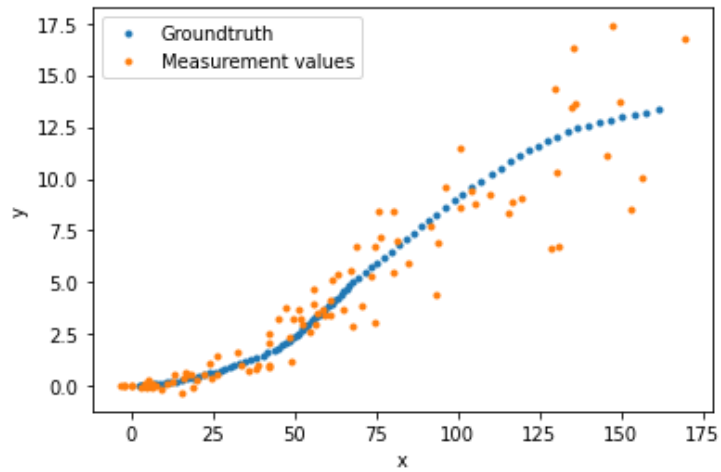


Figure 6.2: Example2 track generated from matlab

A track with less maneuvers



Chapter 7

Implementation

In this chapter the implementation details of Alphabeta filter, Kalman filter, extended kalman filter and LSTM model is explained with evaluation criteria. Python language with version 3 is used for implementing all the four approaches and LSTM model is implemented using Keras library with Tensorflow backend.

7.1 Alpha beta filter

Alpha beta filter is implemented in *alpha_beta* file. The file contains a *AlphaBetaFilter* class with all required methods for estimation. From the data generated from matlab in *data.csv* file only the required columns *x, y* of measurements and ground truth are used. Ground truth is used in RMSE calculation for performance evaluation.

The estimation is done for individual tracks or iterations which has 101 points. Since our final goal is to compare with LSTM performance and LSTM uses first 5 measurements to predict sixth data point and compare it to ground truth. So only 96 points are considered.

Since the estimation for data points depends on the previous value of position and velocity the first estimation would be same as measurement for all state variable the is x, y, v_x, v_y .

Eqn 2.1 is implemented in *predictions()* method and other Eqns from 2.2 to 2.5 are implemented in *add_sample()* method of *AlphaBetaFilter* class. Plotting

and RMSE calculation on estimation, ground truth and measurements functions are available in same file.

7.2 Kalman filter

Kalman filter implementation can be seen in *kalman_filter.py* in *kalman_filter* folder. The file has all required functions to estimate the position on an object based on measurements. From the *test_data.csv* file only required columns of measurements in Cartesian coordinate system i.e x(position in x direction), y(position in y direction), dx(velocity in x direction), dy(velocity in y direction) and for ground truth only x and y positions are used.

For estimation of each point in a track the python function *kalman_estimation()* is used with measurements. This function in-turn calls all other functions and for the first estimation the first measurement itself is the final estimation. Prediction error Co-variance matrix P being an identity matrix for first estimation of a point in a track.

Having initial values after first estimation of a point in the track, prediction is carried out for the 2nd point in the track and further points of track using equation 3.1 and is implemented in *prediction_State_Vector()* function which returns the prediction of state vector i.e x, y, v_x, v_y . The prediction error co-variance matrix prediction is calculated using eqn 3.2 in the *prediction_process_covariance()* function. The process noise Q is calculated as mentioned in section 3.1 and is implemented in the same function.

The kalman gain calculation is implemented using eqn 3.3. Measurement error co-variance matrix R is usually provided by the measuring device supplier. In present case since we know the measurement and ground truth its easy to get the R matrix. It is variance of error between ground truth and the measurement of all state variables x, y, v_x, v_y for each iteration or track which is tuned after multiple trails. So that R is not calculated for each track separately.

The kalman gain is calculated *kalman_gain()* function and is used in the update step as in equation 3.4. y in the equation is the real measurement that is sent out by radar for all state variables. Similarly update of prediction error co-variance matrix P of estimation error of state variables is also calculated. The

measurement matrix H in eqn 3.3 is identity matrix of size 4 because we have 4 states in our state vector and there is no conversion needed between measurement equipment and estimating function which makes it an identity matrix.

Based on the Kalman gain calculation next step is to update the state vector as in eqn 3.4 and that is calculated in *Update_State_Vector()* function. The updated process co-variance matrix P is implemented in *Update_Process_Covar()* function.

Each iteration is of 101 points of radar point cloud but only 96 points are considered leaving the first 5 points since the LSTM gives its first prediction after a sequence of 5 points. So in order to evaluate performance in each iteration(track) first 5 points are not considered in Kalman filter.

The evaluation is based on Root mean square error(RMSE) between the prediction and the ground truth in each iteration. Similarly the RMSE is calculated between measurements and ground truth. Implementation to calculate RMSE is in *calculate_rmse()* function.

7.3 Extended Kalman Filter

In further to discussion of extended kalman filter(EKF) in chapter 4 in this section the implementation details would be discussed. The complete implementation of EKF is split in 5 files and they are *main.py*, *Extended_kf.py*, *Kalman-filter.py*, *DataSample.py* and *Toolkit.py*. The data preparation explained in chapter 6 is still valid for EKF as well. The .csv needs to be converted to a text file since here data is read line by line. Here also radar point cloud information of the tracks or iterations of 96 points are sent to EKF to get the filtered estimations.

Going through details of implementation, *main.py* has required functions for processing data, estimation related functions and plotting functions to visualize data. *Parse_data()* function needs the path for data file in folder where script is present. First initialization for prediction error co-variance matrix P is a Identity matrix of 4x4 and measurement noise co-variance is tuned for better performance based on the error between measurement and the ground truth. The transition matrix (A) and process noise(Q) are initialized to identity matrix and

get_estimations() is a function which sends measurement to get the estimate of state variables using EKF.

There are 2 types of data first is the one which is raw data from radar in polar form and another is the ground truth of position and velocity of object in Cartesian form. In *DataSample* class of *DataSample.py* file each data point is processed based on if it is radar measurement or ground truth for further comparison of EKF performance and usage. In the *Toolkit.py* there are functions for polar to Cartesian and vice versa conversions and to calculate Jacobian of $h(x)$ as in eqn 4.6 matrix.

When a data point of radar point cloud is passed to estimate the filtered position in *main.py* in *get_estimations()* function it uses *process()* from *Extended_kf* class. If it is the first estimation then final updated estimation would be same as first measurement. From second estimation the *update()* function is called which takes care of updating the transition matrix A and Q matrix based on the time difference between the consecutive measurements as in 4.1 and 4.2 respectively for calculating the prediction of state vectors. But here the measurements are available at equal intervals making these matrix constants after first computation. With required information the Jacobian is calculated for the obtained raw measurements. Having all these the final step is to calculate the posterior and that is done using *update()* method of *Kalmanfilter* class.

The above process is applied to all the remaining data points of a particular iteration. And finally the RMSE is calculated using the *combined_RMSE()* function in *main.py* for each iteration of 96 data points.

7.4 LSTM

The LSTM model and data preparation is available in *LSTM_object_tracking.ipynb* file in LSTM folder. The LSTM model is implemented using Keras library using Tensorflow back-end. The tensor flow version used is 2.4.1. And other python libraries like sklearn, numpy and other generic libraries were used.

The data is generated from matlab is available in .csv format. The .csv file contains track number, point number in the track, range, range rate, azimuth angle, x, y, v_x, v_y for both measurement and ground truth. For training LSTM

model Cartesian coordinates of measurements is used. For training and validation a set of around 970 iterations (tracks) are used of which 80 percent is used for training an LSTM model and 20 percent was used for validation from *training_data.csv*. The data set with window size of 5 was prepared where in 6th is predicted by LSTM using the previous 5 measurements and label for that is 6th element in the Ground truth. This is done separately for each track. So there will be 96 points predicted since first 5 points are used for predicting the 6th point in each track. Testing was done on separate data set. Similar data set for testing is prepared from another *testing_data.csv* file having many iterations.

To train a model, some training or hyper parameters were tuned and they are usually decided randomly in beginning and tuned based on data used to train the model. The training parameters are :

- *LSTM Units*: It is the number of LSTM units in each layer of the model.
- *Window size*: It the length of sequence based on which the next prediction is calculated in sequence.
- *Hidden layers*: It the number of hidden layers between the input layer and the output layer in neural network.
- *Batch size* : This is number of samples considered at a time for training for each forward and backward propagation.
- *Learning rate*: It decided the rate of gradient descent.
- *Number of epochs* : Number of iterations of training.
- *Optimizer*: Algorithms used to change the attributes of the neural network such as weights and learning rate to reduce the losses.

<i>Parameter</i>	<i>Value</i>
Batch size	96
<i>Learning Rate</i>	0.001
<i>Number of Epochs</i>	60
<i>Optimizer</i>	Adam
<i>Hidden layers</i>	2
<i>Window size</i>	5

The model used is shown below in code snippet where first the sequence of x and y values are sent in (T=5,2) shape to LSTM layer of 256 units with default activation *tanh* function to learn the uncertainty in the measurement sequence and followed by a dense layer of 128 units with *ReLU* function is used. Here (5,2) represents a sequence of length 5 in dimensions x and y. And at end we have 2 dense units since we have 2 output x,y positions.

Figure 7.1: LSTM model

```
adm = Adam(lr=0.001)
LSTM_model = Sequential()
LSTM_model.add(LSTM(256,input_shape=(T,2)))
LSTM_model.add(Dense(128,activation='relu'))
LSTM_model.add(Dense(2))
LSTM_model.compile(loss='mse',optimizer=adm)
rtu= LSTM_model.fit(train_input_seqs, train_gd_seq, epochs=60, batch_size = 96,verbose=2 ,
                    validation_data=(valid_input_seqs,valid_gd_seq))
```

Performance for object tracking is evaluated based on RMSE as it was used in Kalman filter also. The RMSE is calculated between the prediction of LSTM model with ground truth and Measurement with ground truth for each iteration or track.

Chapter 8

Results

Performance analysis for single object tracking using radar point cloud data was done for 4 possible approaches Alphabeta filter Kalman filter, Extended kalman filter and tracking based on Long short term memory. The project explains the details about object tracking and 4 possible approaches for single object tracking quoting details of equations used in all the approaches. The radar point cloud data was simulated using the Matlab script. The evaluation was done based on Root mean square error between the predictions done by each one of them with respect to ground truth and RMSE between the Measurement and ground truth.

The basic idea is to compare the RMSE between the prediction-ground truth of all the approaches individually and measurement-ground truth. To consider estimation method to be good the RMSE for prediction-ground truth should be less than measurement-ground truth.

The Kalman filter is usually designed for linear motion models and it was proved that it performs better in linear motion of vehicle and performance decreases in case of non-linear motion i.e when there are more maneuvers in vehicle or object being tracked.

The LSTM performs better than Kalman filter in non-linear cases, it does not depend on whether the motion is linear or non-linear so basically it learns the motion model irrespective of the type of motion of vehicle or object. It turns out that LSTM performance is comparable with kalman filter in less maneuver cases.

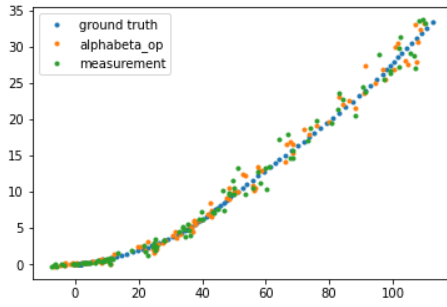
Extended kalman filter performs better for both linear and non-linear motions and its is even better than LSTM for the data set it was tested on. To understand the performance we use some symbols like P_{rmse} which is RMSE between the prediction done by estimation method to ground truth. And the M_{rmse} is the RMSE between the track measurement and ground truth.

8.1 Performance analysis for less maneuvers tracks

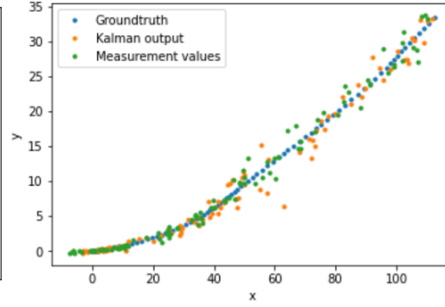
From the fig 8.1(a) and fig 8.1(d) tracks with less maneuvers, RMSE of Alphabeta filter estimation to ground truth is more than RMSE of LSTM prediction to ground truth by using same measurements. It indicates the LSTM has better performance than Alphabeta filter.

From the fig 8.1(b) and fig 8.1(d) tracks with less maneuvers, RMSE of kalman filter estimation to ground truth is comparable to RMSE of LSTM prediction to ground truth by using same measurements. It indicates the LSTM performance is similar to Kalman in less maneuvers.

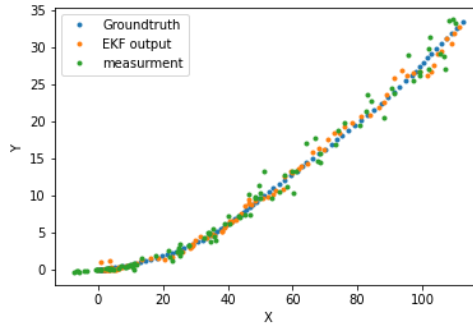
EKF performance is better for less maneuvers as seen in fig 8.1(c) when compared to Kalman filter and LSTM. Since it linearizes small non-linearities and performs better.



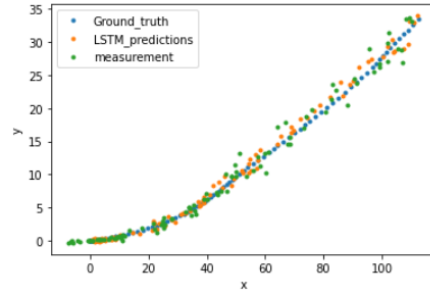
(a) Alphabeta Filter $P_{rmse} = 3.10$



(b) Kalman filter $P_{rmse} = 2.47$



(c) Extended Kalman filter $P_{rmse} = 1.93$



(d) LSTM $P_{rmse} = 2.23$

Figure 8.1: Track of less Maneuvers with $M_{rmse} = 3.64$

8.2 Performance analysis for more maneuvers tracks

From fig 8.2(a) and fig 8.2(d) it can be seen that LSTM performs better than Alphabeta filter in tracks with more maneuvers. And also Alphabeta performs better than kalman in more maneuvers as seen in fig 8.2(a) and 8.2(b)

From fig 8.2(b) and 8.2(d) it can be seen that LSTM performs better than Kalman filter in tracks with more maneuvers. Kalman filter performance decreases since it cannot adjust its update step to the sudden change in the direction of travelling or longitudinal acceleration of vehicle, it takes some measurement data points to correct its kalman gain and estimate updated values nearer to ground truth. On the other hand LSTM makes its decision based on the previous measurements and predict the next nearest position in comparison with ground truth.

EKF performs better than kalman filter as seen in fig 8.2(c) and 8.2(b) as expected since it depends on Jacobian for update step of estimation and takes care of non-linear motions of vehicle.

EKF also performs better than LSTM as seen in as fig 8.2(c) and fig 8.2(d) this might be because the LSTM could not completely learn the variance in track so that it can outperforms the EKF.

To summarize LSTM outperforms Kalman filter in tracks having more maneuvers and LSTM performs almost similar to kalman filter for the tracks which has less maneuver but EKF performs better than LSTM Still LSTM is good approach too and can be relied on for all sought of tracks to get optimum RMSE but not the best RMSE . Of-course there are some advanced LSTM architecture can be tested for better performance than EKF which can be seen in chapter future scope.

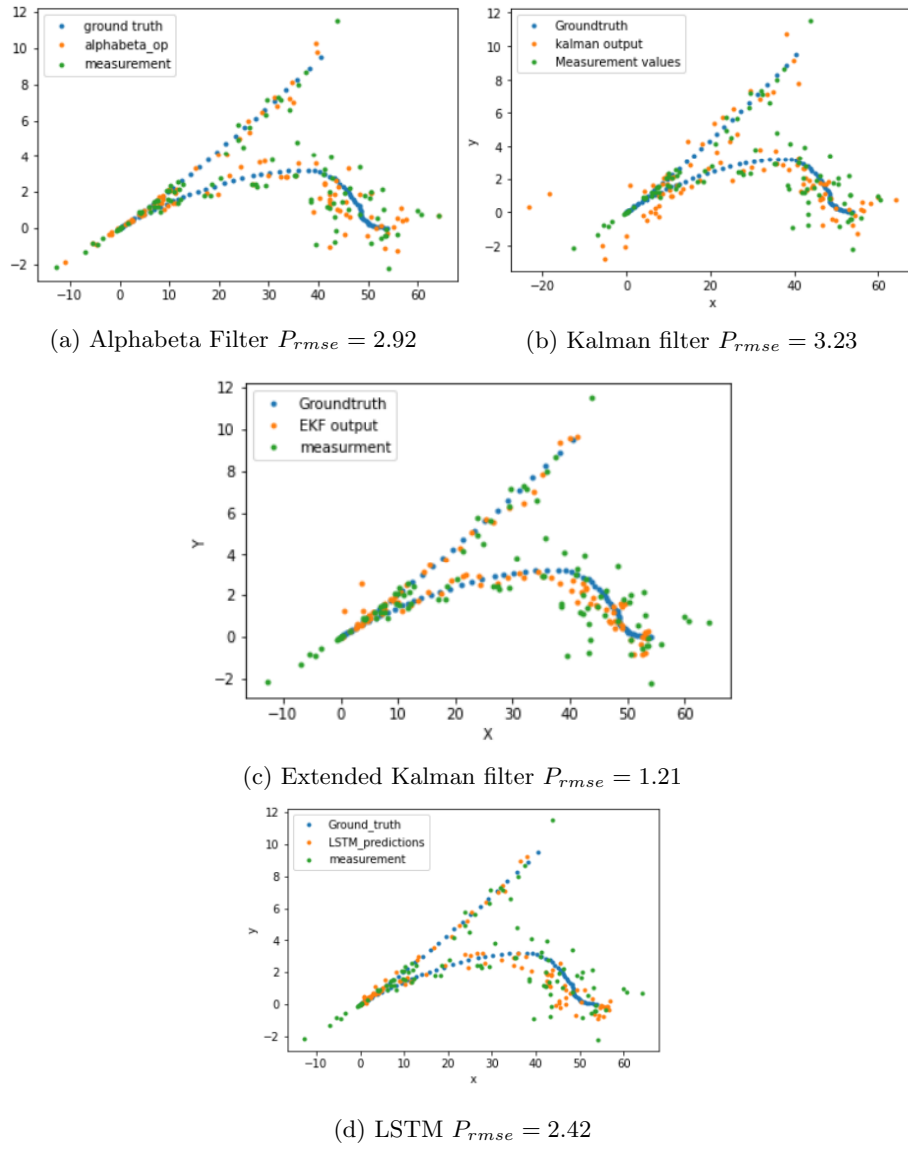


Figure 8.2: Track of more Maneuvers with $M_{rmse} = 3.79$

Chapter 9

Future scope

The single object tracking using kalman filter,extended kalman filter are already been proven as an effective means to estimate position of vehicle and LSTM is comparatively new approach using radar point cloud data.With the analysis LSTM also proves to provide optimum estimation.There were many impediments faced while implementing this project. While some of those were resolved within the scope of this research thesis, solutions to some of the unresolved complications are mentioned as a future scope of this project.

The data that was used was simulated data using matlab and always there is difference in results for real case scenarios ,so there is possibility to get real data using any of possible mmwave radars to get radar point cloud data and perform analysis on it. The data generation can also be done on simulated platforms like Carla, AirSim which can also be used to get data more nearer to real-time data and perform analysis.There are still more advanced filters like adaptive kalman filter , Unscented kalman filter on which performance analysis can be carried out.The approach followed can be extended to multiple object tracking using clustering and data association and perform analysis on these different approaches and along with other filters quoted earlier.Object tracking using sensor fusion can be done which increase accuracy and acts as back up in failure of any one of them.Along with radar, camera can be used which tracks objects by consecutive image frames.

The LSTM network can be used to learn the more nonlinear motion models

for which getting a dynamic motion model is difficult and accuracy can be increased by following advanced LSTM architecture like encoder and decoder LSTM to perform better than EKF shown in [5] .

We faced difficulty in getting labelled radar data, and the dataset which we found had very less frames by which it was difficult to train the neural network. To develop and test more number of applications using deep learning based on the radar there is necessity of more open source labelled radar data like how we have for camera.

Bibliography

- [1] RajaGopalReddy B et al. *State Estimation and Tracking using Recurrent Neural Networks*.
- [2] Alex Becker. <https://www.kalmanfilter.net/default.aspx>.
- [3] Jason Brownlee. <https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/>.
- [4] Jason Brownlee. <https://machinelearningmastery.com/tune-lstm-hyperparameters-keras-time-series-forecasting/>.
- [5] Juan Pedro Llerena Caña, Jesús García Herrero, and José Manuel Molina López. *Forecasting Nonlinear Systems with LSTM: Analysis and Comparison with EKF*.
- [6] Huseyin Coskun, Felix Achilles, and Robert DiPietro. *Long Short-Term Memory Kalman Filters Recurrent Neural Estimators for Pose Regularization*.
- [7] Dan Iter, Jonathan Kuck, and Philip Zhuang. *Target Tracking with Kalman Filtering, KNN and LSTMs*.
- [8] Ming xin Jiang et al. *Multiobject Tracking in Videos Based on LSTM and Deep Reinforcement Learning*.
- [9] Keras. https://keras.io/api/layers/recurrent_layers/lstm/.
- [10] Keras. <https://keras.io/api/optimizers>.
- [11] Intech Open. <https://www.intechopen.com/books/introduction-and-implementations-of-the-kalman-filter/introduction-to-kalman-filter-and-its-applications>.
- [12] Maria Isabel Ribeiro. *Kalman and Extended Kalman Filters: Concept, Derivation and Properties*.

BIBLIOGRAPHY

- [13] towards data science. <https://towardsdatascience.com/alpha-beta-filter-21d3276cf35e>.
- [14] Arindam Sengupta, Feng Jin, and Siyang Cao. *A DNN-LSTM based Target Tracking Approach using mmWave Radar and Camera Sensor Fusion*.
- [15] Gabriel A. Terejanu. <https://www.cse.sc.edu/terejanu/files/tutorialEKF.pdf>.
- [16] Udacity. <https://www.udacity.com/course/self-driving-car-engineer-nanodegree--nd013>.
- [17] wiki. https://en.wikipedia.org/wiki/Alpha_beta_filter.
- [18] Christian Wolff. <https://www.radartutorial.eu/index.en.html>.
- [19] Tianyu Zheng et al. *An RNN-based Learnable Extended Kalman Filter Design and Application*.