IDENTIFICATION AND FORECASTING OF STATIONARY BANDS OF PRECIPITATION

A dissertation submitted to the University of Manchester
for the degree of Master of Science
in the Faculty of Engineering and Physical Sciences

2016

Nikolaos Theodorakopoulos Candidate
School of Mathematics

Contents

Word count xxxxx

List of Tables

List of Figures

Abstract

A forecast is an estimation of the future state of the atmosphere. It is created by estimating the present state of the atmosphere through observations, and then calculating how this state will change through time. Despite the technological advantages that exist today, weather forecast remain one of the most unexplored areas for the scientific computing.

In this work, we have a radar that captures pictures of the atmosphere every 5 minutes. After some modifications, the observed storm cells are stored as ellipsoids. We will use the Kalman filter as a forecast tool in order to track and predict the movement of these storm sells. At the same time we will test Kalman filter under different parameters to see how it responds and which of these parameters fits better our storm tracking problem. Kalman filter is ideal for now-casting and will allow you to model any linear system accurately.

Declaration

No portion of the work referred to in the dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Intellectual Property Statement

Acknowledgements

I would like to thank my supervisor Dr. Oliver Dorn for his valuable guidance and patience through this work.

I would like also to thank my parents, John and Helen who taught me to value science and knowledge.

Chapter 1

Introduction

1.0.1 Model the data into ellipses

Two professors from the School of Earth and Environmental Sciences, Dr. David Schultz [4] and Dr. Jonathan Fairman [5] work on weather forecast. They use a radar that captures the state of the atmosphere every 5 minutes and they need to do a weather forecast before they get the next observation. Also Dr. Fairman noticed that the data that he receives from the radar measurements for the position of the storm cells are sometimes slightly different from the real position. So he asked for help from his friend Dr. Oliver Dorn, from the School of Mathematics, in order to find a way to track and predict the movement of the storm cells for short time ahead. In order to have data to work on he sent us data information about the size and the position of the storm cells which he already modified into ellipses. We will work on data about the storm cells for all days of December 2015. In our case Kalman filter is the ideal tool in order to predict the movement of the storm cells and at the same time filter the noise that the radar sensors may have.

At first, we extracted from these data the position (x, y) of the ellipsoids, the orientation of the ellipses as computed from angle horizontal in x direction and the semi-axes of the ellipses. Using MATLAB we plotted the ellipsoids to get the following images for storm clouds above UK and Ireland:

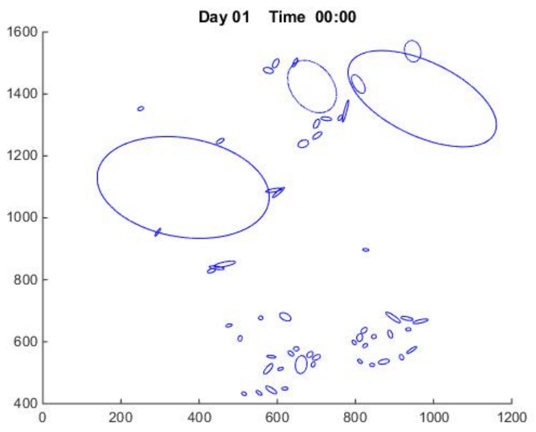

Figure 1.1: 1/12/2015 at 12:00 am

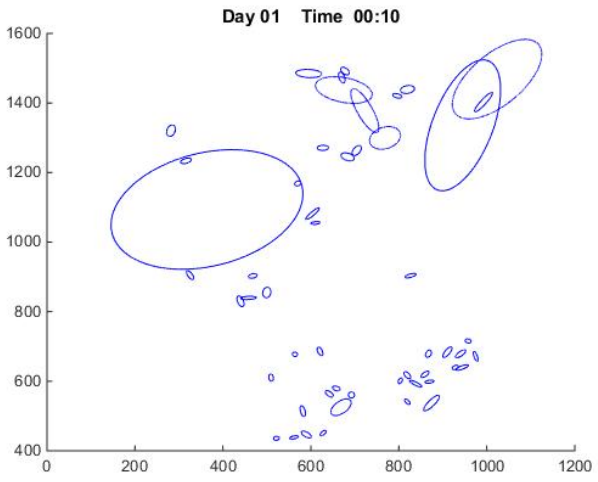

Figure 1.2: 1/12/2015 at 12:10 am

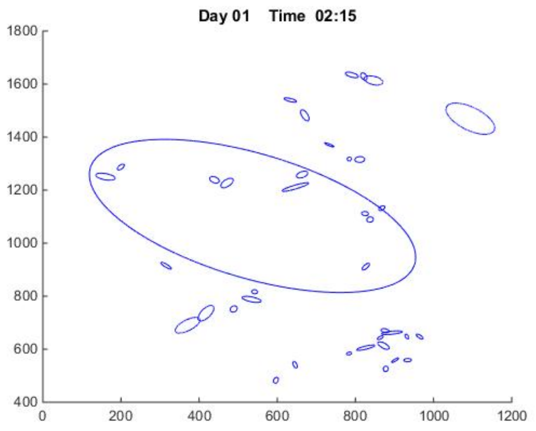

Figure 1.3: 1/12/2015 at 02:15 am

From the figures above we can see that after the first few minutes the ellipses have been slightly moved from the previous position. As time further passes we can see at Figure 1.3 that some new ellipses have been born while others died or combined together to a bigger or smaller ellipsoid. Pekka J. Rossi gives a good definition about the splitting and merging of a storm track: Convective storms tend to split and merge frequently. A split occurs if two cells belong to the same cluster in the time frame k but to different clusters at the next time step $k + 1$. The merging is analogous to the splitting; it is considered if cells belong to different cluster in the current frame k but to the same cluster in the next frame $k + 1$. The track is called complex if it includes splits or mergers [1].

At first we will try to track and predict only the center of the ellipsoid with the biggest surface using Kalman Filter. Then we will try to generalize Kalman filter in order to track the whole ellipse. Lets first see what the Kalman Filter actually is.

1.0.2 Kalman Filter

Kalman Filter is a powerful tool that can be used to predict a phenomenon or track an object due to a time period. It is also ideal for systems which continuously change and will allow you to model any linear system accurately. In practise KF evolves an initial state through time and is divided into two main steps. First, the prediction step where KF predicts the state k from a previous known state $k - 1$ based only on the model calculations. Second, the update step where KF compares the predicted state that just calculated with noisy observations that we got for the state k and corrects the predicted state. So now the final updated state becomes the initial state for the next prediction.

All we need to start is a state vector $\vec{x}$. This vector could contain any number of variables and represent anything we want. Usually, when we need to track an object this state vector should contain the initial *position* and *velocity*. Remember that for now we want to track only the center of the biggest ellipse at each time step. Here, (x, y) is the center position of the ellipse and $(\dot{x}, \dot{y})$ are the corresponding velocity components. All can be arranged together as the following state vector:

$$\vec{x} = (x, y, \dot{x}, \dot{y}) \in R^4$$

Kalman Filter assumes that the variables are *random* and *Gaussian distributed.* This means that each of these variables have a mean value μ and a variance σ^2.

In fact, we can assume that position and velocity are correlated: The likelihood of observing a particular position depends on what velocity you have. This kind of situation might arise if, for example, we are estimating a new position based on an old one. If our velocity was high, we probably moved farther, so our position will be more distant. If we are moving slowly, we did not get as far [2]. This correlation is captured by the **Covariance Matrix** [3].

Now in order to describe our problem with matrices at time κ, we will call our best estimate $\vec{x}_\kappa$ which is the mean and its covariance matrix P_κ.

$$\vec{x}_\kappa = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}^T$$

The matrix P_κ captures the variances and covariance of the state variables and is by definition symmetric and positive semi definite. Note that in this work we will refer to P_κ as the state covariance matrix or the priori covariance matrix.

$$P_\kappa = \begin{bmatrix} \sigma^2_{x_\kappa} & \sigma_{x_\kappa y_\kappa} & \sigma_{x_\kappa \dot{x}_\kappa} & \sigma_{x_\kappa \dot{y}_\kappa} \\ \sigma_{x_\kappa y_\kappa} & \sigma^2_{y_\kappa} & \sigma_{y_\kappa \dot{x}_\kappa} & \sigma_{y_\kappa \dot{y}_\kappa} \\ \sigma_{x_\kappa \dot{x}_\kappa} & \sigma_{y_\kappa \dot{x}_\kappa} & \sigma^2_{\dot{x}_\kappa} & \sigma_{\dot{x}_\kappa \dot{y}_\kappa} \\ \sigma_{x_\kappa \dot{y}_\kappa} & \sigma_{y_\kappa \dot{y}_\kappa} & \sigma_{\dot{x}_\kappa \dot{y}_\kappa} & \sigma^2_{\dot{y}_\kappa} \end{bmatrix} \in R^{4x4}$$

Now we need to look at the current state at time $\kappa - 1$ and then predict the future state at time κ. We can do this by using a simple kinematic formula for position and velocity:

$$x_\kappa = x_{\kappa-1} + \dot{x}_{\kappa-1}\Delta t$$
$$y_\kappa = y_{\kappa-1} + \dot{y}_{\kappa-1}\Delta t$$

$$\dot{x}_\kappa = \dot{x}_{\kappa-1}$$
$$\dot{y}_\kappa = \dot{y}_{\kappa-1}$$

And so we have:

$$\vec{x}_\kappa = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\kappa-1} \\ y_{\kappa-1} \\ \dot{x}_{\kappa-1} \\ \dot{y}_{\kappa-1} \end{bmatrix} = F_\kappa \vec{x}_{\kappa-1}$$

Where F_κ is the **Prediction Matrix** and gives us our next state. It takes every variable of our initial state vector and moves it to a new predicted location, which is where the system would have moved if that initial estimate was the right one. Notice that the determinant of the prediction matrix $det(F_\kappa) = 1$ which means that the system has unique solution.

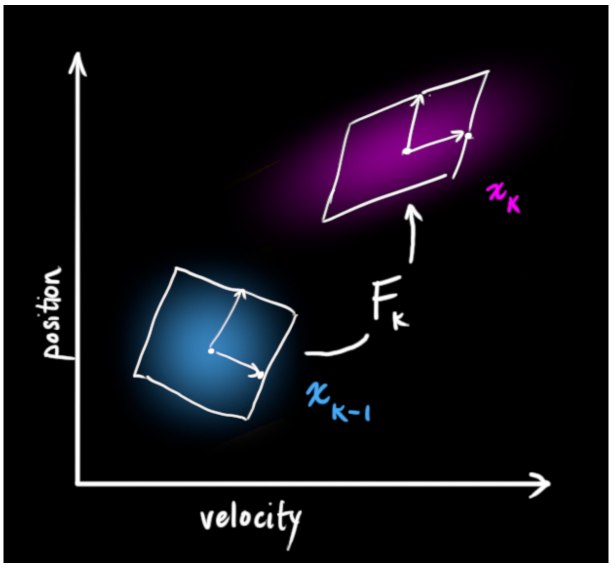

Figure 1.4: The impact of Prediction Matrix F_κ taken from [2].

Also, we need to update the Covariance matrix P_κ. To do this we need to multiply $P_{\kappa-1}$ with the the prediction matrix F_κ. So we have:

$$P_\kappa = F_\kappa P_{\kappa-1} {F_\kappa}^T$$

External Influences

There might be some external **known** influences that we need to include in our calculations. For example, we could know that at some point of time there will be an acceleration by the wind to our storm cells, lets say α. To add this detail in the prediction step we can use again the following kinematic formula:

$$\begin{aligned}
x_\kappa &= x_{\kappa-1} + \dot{x}_{\kappa-1}\Delta t + \tfrac{1}{2}\alpha\Delta t^2 \\
y_\kappa &= y_{\kappa-1} + \dot{y}_{\kappa-1}\Delta t + \tfrac{1}{2}\alpha\Delta t^2 \\
\dot{x}_\kappa &= \dot{x}_{\kappa-1} + \alpha\Delta t \\
\dot{y}_\kappa &= \dot{y}_{\kappa-1} + \alpha\Delta t
\end{aligned}$$

Which in matrix form is:

$$\vec{x}_\kappa = F_\kappa \vec{x}_{\kappa-1} + \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \frac{1}{2}\Delta t^2 \\ \Delta t \\ \Delta t \end{bmatrix} \alpha = F_\kappa \vec{x}_{\kappa-1} + B_\kappa \vec{u}_\kappa + \vec{w}_{\kappa-1}$$

Where B_κ is called the **Control Matrix** and $\vec{u}_\kappa$ the **Control Vector**. Vector $\vec{u}_\kappa$ is an input to our algorithm and measures the user control on the situation [2]. The random vector $\vec{w}_{\kappa-1}$ is the process noise which is added to describe the uncertainties of the system. It is assumed that it follows the multivariate Gauss distribution with zero mean and measurement noise covariance $Q_{\kappa-1}$.

$$\vec{w}_{\kappa-1} \sim N(0, Q_{\kappa-1})$$

External Uncertainty

There might be some **unknown** external forces. In our work, our purpose is to track and predict storm cells that randomly buffeted around by wind. As you understand we have a plenty of random and unknown forces that we do not know but we will capture in a matrix $Q_{\kappa-1}$. We will introduce this uncertainty into our algorithm as the error of the process. We will expand the previous covariance:

$$P_\kappa = F_\kappa P_{\kappa-1} {F_\kappa}^T + Q_{\kappa-1}$$

by simply adding $Q_{\kappa-1}$. Every state in our original estimate could have moved to a range of states. We will say that each point in $\vec{x}_{\kappa-1}$ is moved to somewhere inside a Gaussian blob with covariance $Q_{\kappa-1}$. Because we have defined the position components x and y to be independent to each other there is no correlation between them and so we set their correlation to zero. The same thing is true for the velocities $\dot{x}$ and $\dot{y}$ and so we set their correlation to zero. But there is a correlation between x

and $\dot{x}$ and y and $\dot{y}$. Finally we can define the error of the processes as the covariance matrix $Q_{\kappa-1}$

$$Q_{\kappa-1} = \begin{bmatrix} \frac{\Delta t^4}{4} & 0 & \frac{\Delta t^3}{2} & 0 \\ 0 & \frac{\Delta t^4}{4} & 0 & \frac{\Delta t^3}{2} \\ \frac{\Delta t^3}{2} & 0 & \Delta t^2 & 0 \\ 0 & \frac{\Delta t^3}{2} & 0 & \Delta t^2 \end{bmatrix}$$

This matrix shows how imperfect the model itself is. If the matrix contains only zeros, it shows that the model is perfect, but if it contains non-zeros, it shows that the model can not entirely be trusted, and that more weight should be given to the observations. The success or failure of the Kalman filter depends on this matrix. As we said, matrix $Q_{\kappa-1}$ captures the error of the process. There is a great difference if the matrix $Q_{\kappa-1}$ is going to be initialised as time invariant **or** updated inside the recursion of the Kalman filter as time variant for $\Delta t = 5, 10, 15...$ Time invariant Q means that we will not introduce extra uncertainty to our filter at each time step.

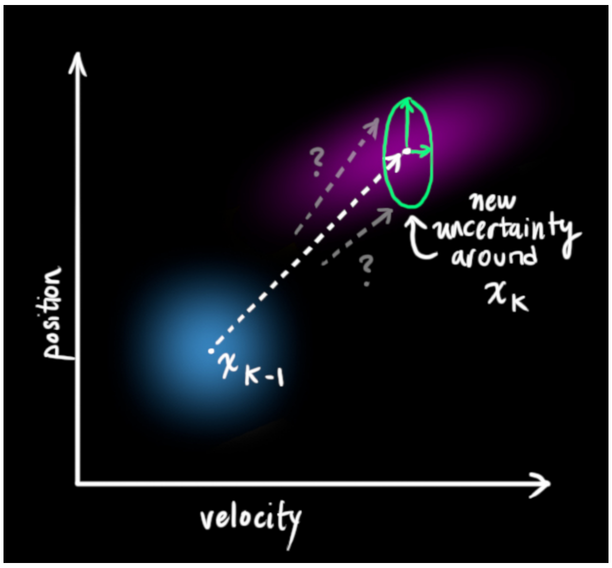

Figure 1.5: Each point in $\vec{x}_{\kappa-1}$ is moved to somewhere inside a Gaussian blob with covariance $Q_{\kappa-1}$ taken from [2].

The Complete Prediction Step

$$\vec{x}_\kappa = F_\kappa \vec{x}_{\kappa-1} + B_\kappa \vec{u}_\kappa + \vec{w}_{\kappa-1} \tag{1.1}$$

$$P_\kappa = F_\kappa P_{\kappa-1} {F_\kappa}^T + Q_{\kappa-1} \tag{1.2}$$

In other words, the new best estimate is a prediction made from previous best estimate, plus a correction for known external influences.
And the new uncertainty is predicted from the old uncertainty, with some additional uncertainty from the environment [1].

Notice that $\vec{x}_\kappa$ is the model prediction of the state **before** updated with the new observations. Next we will see how the new observations affect the final estimation of Kalman Filter.

Now it is important to mention that we will use the properties of the state covariance P_κ to obtain informations about the **reliability** of the Kalman filter **model**. The matrix P_κ measures the confidence of our model to predict the state $\vec{x}_\kappa$. If the entries of P_κ are large it means that there is large uncertainty from our model about the predicted state. Otherwise, if we have small variance then we can trust the calculations of our model for the predicted state. Another important thing is that the trace of the state covariance matrix P_κ will converge to a number. But why is this so important? The answer lies in the following definition: The matrix P_κ represents the variances of the state variables [3]. The $trace(P_\kappa)$ is the sum of the diagonal elements of the matrix P_κ. So, if the sum of the diagonal elements converge towards some value, then we reason that the single elements also converge towards a solution [3].

We can see from equation (2) that P_κ depends on $Q_{\kappa-1}$. In our algorithm if we initialise Q as time invariant it means that at each time step we introduce to our algorithm the same fixed amount of the uncertainty of the process and as we will see later the trace of P_κ **converge** to a number.

On the other hand if Q is time variant i.e putting $\Delta t = 5,\ 10,\ 15..$ then at each time step we introduce to our algorithm more and more uncertainty (more error of the process) which means that we are very unsure for our model calculations. Later we will see that in this case the trace of the state covariance matrix P_κ will finally diverge and the Kalman filter will fail.

The Complete Update Step

Kalman Filtering can be regarded as statistical inversion method where the unknown state $\vec{x}_\kappa$ of a system is estimated through noisy measurements $\vec{z}_\kappa$ [1]. The measurement update model $\vec{z}_\kappa$ will give us at each time step the observed data that we are interested for. Until now we want to track only the center of the ellipse, so $\vec{z}_\kappa$ will contain only measurements for the position (x, y) plus a random vector $\vec{v}_\kappa$ which is the measurement noise. This noise $\vec{v}_\kappa$ it is assumed to follow the multivariate Gauss distribution with zero mean and measurement noise covariance R.

$$\vec{v}_\kappa \sim N(0, R_\kappa)$$

We can not say that $\vec{z}_\kappa$ is the true position of the storm cells. In fact it is the measurement that we obtained from the radar that might have a lot of noise. We define these noisy measurements as $\vec{z}_\kappa = Hx^*_\kappa + \vec{v}_\kappa$:

$$\vec{z}_\kappa = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_\kappa \\ y_\kappa \\ \dot{x}_\kappa \\ \dot{y}_\kappa \end{bmatrix} + \vec{v}_\kappa = Hx^*_\kappa + \vec{v}_\kappa$$

Where H is the **observation model matrix** which we use to import the observed data into our algorithm and defines how the observed variables are mapped into the measurements. From each information that we get from the observation model, we might guess that our system was in a particular state. But because there is uncertainty, some states are more likely than others. We will call the covariance of this uncertainty (i.e. of the sensor noise) R_κ which is also a covariance matrix, but it is the variances and covariances of our sensor measurements. For now we have measurements for only two things: The x position and the y position and so it is like we have two sensors where the first measures the x and the second measures the y. We do not have any measurents for the velocities $\dot{x}$ and $\dot{y}$. So R_κ is a 2x2 matrix. In fact R_κ measures our confidence about the accuracy of the sensors and therefore the Kalman filter algorithm does not change R_κ due to the process. In our work we will define the matrix R_κ as following:

$$R_\kappa = R = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}$$

In a few words, the matrix R_κ captures the error in the measurements. In our work we are only measuring two things: The x position and the y position which, as we mentioned earlier, are independent to each other. Because they are defined as independent there is no interaction between them and all we are going to do in this model is to use the variance σ_x^2 and the variance σ_y^2 which is the variance of the radar for x and y coordinate respectively. Later we will define what these variances we want to be.

So now, the thing is that we have two distributions: The **predicted** measurement with $(\mu_0, \Sigma_0) = (H\vec{x}_\kappa, H_\kappa P_\kappa H_\kappa^T)$ and the **observed** measurement with $(\mu_1, \Sigma_1) = (\vec{z}_\kappa, R_\kappa)$.

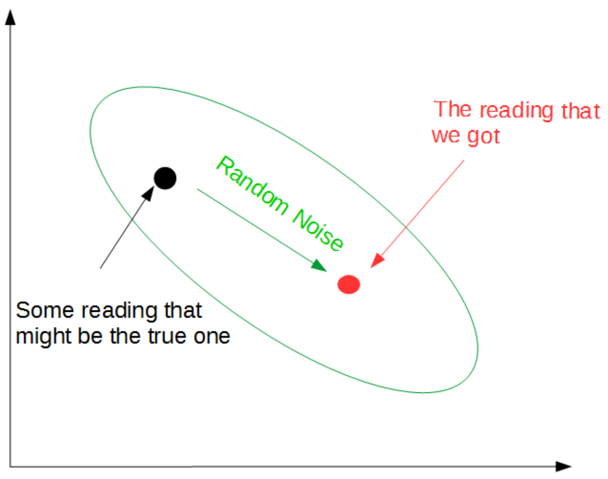

Figure 1.6: The effect of sensor noise R_κ.

After combining the Gaussian distributions we get the following complete equations for the update step:

$$K = P_\kappa H_\kappa^T (H_\kappa P_\kappa H_\kappa^T + R_\kappa)^{-1} \tag{1.3}$$

$$\hat{x}_\kappa = \vec{x}_\kappa + K(\vec{z}_\kappa - H_\kappa \vec{x}_\kappa) \tag{1.4}$$

$$\hat{P}_\kappa = (I - KH_\kappa)P_\kappa \tag{1.5}$$

Matrix $K \in R^{4x2}$ is the Kalman Gain, which defines the weight given to the observation with respect to predicted state. In fact, Kalman gain is the regulator in the filter. It determines how much information to use from the model estimation versus how much information to use from the noisy observations. So the Kalman Gain depends on the accuracy of our model and the accuracy of our sensors i.e to the priori covariance P_κ and the error in the observations R. Note that in this work we will call $\widehat{P}_\kappa$ as the posteriori covariance.

The Kalman Gain will be:

- Large if the measurements are accurate (the variances of the sensor noise σ_x^2 and σ_y^2 are small) **and** if the state estimation $\vec{x}_\kappa$ which is calculated in the prediction step is inaccurate (has a large variance i.e priori covariance matrix P_κ has large index). By "large" we mean that the entries of K will have values close to 1. So if $K_1 = 1$ then the updated estimation $\widehat{x}_\kappa$ will be the observed data and so $\widehat{x}_\kappa = \vec{z}_\kappa$

- Small if the measurements are inaccurate (the variances of the sensor noise are big) **and** if the state estimation $\vec{x}_\kappa$ is accurate (Priori covariance matrix P_κ has small index). Now by "small" we mean that the index of K will have values close to 0. So if $K_1 = 0$ then the updated estimation $\widehat{x}_\kappa$ is based wholly on the system model and so $\widehat{x}_\kappa = \vec{x}_\kappa$.

In fact, the index of the Kalman gain matrix K will lie between 0 and 1 i.e.

$$0 \leqslant K(i,j) \leqslant 1.$$

But notice that if we want the Kalman filter to be successful then the Kalman gain must take values strictly less than 1 and strictly more than zero. This means that our filter will neither follow completely the observations nor completely the model process. It will combine both the model process and the observations in order to make the final estimation about the state.

We can not say that an ideal index for Kalman gain would be the 0.5 in order to use half information from the observation measurements and half from the model prediction. Because in problems that involve observations maybe we could have more noise in the observations or our model could be very accurate and vice versa. If we want to initialise K as a fixed matrix then we should make a research to know where we have the most noise in our system i.e to the observations or to the noise of the process and then put an appropriate index. Later we will see that if our system is stable then the Kalman gain also converge to a number not close to the extreme values of 0 or 1 which of course is closely related with the convergence of the trace of P_κ.

Synopsis and Pseudo-code

Prediction step

$\vec{x}_\kappa = F_\kappa \vec{x}_{\kappa-1} + B_\kappa \vec{u}_\kappa + \vec{w}_{\kappa-1} \quad$ where $\quad \vec{w}_{\kappa-1} \sim N(0, Q_{\kappa-1})$

$P_\kappa = F_\kappa P_{\kappa-1} {F_\kappa}^T + Q_{\kappa-1}$

Update Step

$K = P_\kappa H_\kappa^T (H_\kappa P_\kappa H_\kappa^T + R_\kappa)^{-1}$

$\widehat{x}_\kappa = \vec{x}_\kappa + K(\vec{z}_\kappa - H_\kappa \vec{x}_\kappa) \quad$ where $\quad \vec{z}_\kappa = H\vec{x}_\kappa + \vec{v}_\kappa$ and $\vec{v}_\kappa \sim N(0, R_\kappa)$

$\widehat{P}_\kappa = (I - KH_\kappa)P_\kappa$

Kalman filter in the **prediction step** predicts a range of possible future states based on the previous state. This range is defined by the the process noise covariance P_κ (red). On the other hand Kalman filter in the **update step** creates a region around the noisy observation that we got from the radar. The range of this region depends on the value that we putted as σ_x^2 and σ_y^2. Putting a small value for σ_x^2 and σ_y^2 means that we create a very small region around the noisy measurement. Finally our Kalman filter estimation lives inside the **intersection** of the two regions. The intersection range is in fact the posteriori covariance $\widehat{P}_\kappa$.

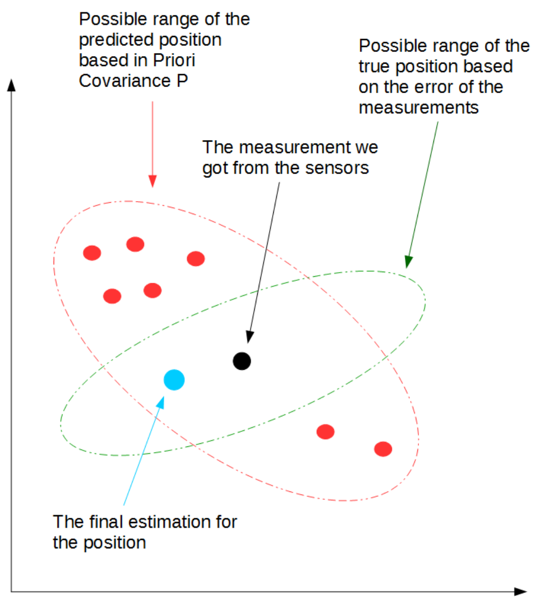
Possible range of the predicted position based in Priori Covariance P
Possible range of the true position based on the error of the measurements
The measurement we got from the sensors
The final estimation for the position

Figure 1.7: Kalman Filter

Chapter 2

Apply the Kalman Filter

2.0.1 Tuning the Kalman Filter

Before we proceed to the numerical computation, we need to discuss which variables, vectors and matrices we will initialise. Some of these data that we initialise play a key role in the success or failure of the Kalman Filter while others are indifferent. The difference between what is important to our filter and what is not is given by the answer to the question: Which initial data are updated inside the filter recursion and which are not updated. The data that are not updated and are initialised once in the beginning are the most important because they tuning Kalman Filter.

The data that are **updated** in the Kalman filter are:

- The state vector $\vec{x_\kappa} \in R^4$. It does not matter how will we initialise the state vector $\vec{x_\kappa}$, because Kalman Filter in the first step will update it close to the noisy measurements $\vec{z_\kappa}$. In our work we will initialise $\vec{x_0} = [1, 1, 0, 0]$ which is the x and y position and the initial velocity which is set to zero.

- The covariance Matrix P_κ. Again it does not matter how will we initialise the covariance matrix. In this work we will initialise $P_0 = I \in R^{4x4}$ where I is the identity matrix. Otherwise P_0 can be defined as $P_0 = Q_{\kappa-1}$ indicating that there is some uncertainty in the initial position of the storm cell.

The choice of the index of $\vec{x_\kappa}$ and P_κ may vary from very big values to very small according to the information about the initial states. It has further been observed that the choices of $\vec{x_0}$ and P_0 mainly affect the first part of the estimation so these are usually not very critical unless the initial estimation exceeds acceptable limits [1].

The data that are **not updated** in the Kalman filter are:

- The error in the observations $R \in R^{2x2}$. This matrix is important in our calculations. It defines the weight that the Kalman gain is going to give in our observed data. In other words, it measures how much we trust the observed data and how much the Kalman filter will try to be close to that data. The index of R is actually the covariance of all the censors. For now, we have sensors only to measure the position: one sensor for x and one sensor for y position. Previously, we defined R as:

$$R_\kappa = R = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}$$

  Putting a small number as $\sigma_x^2 \ll 1$ and $\sigma_y^2 \ll 1$ means that we have a very good confidence about our measurements $\vec{z_\kappa}$. The smallest the index of matrix R, the most confident we are for our measurements. If we want to be more precise about the variance of our radar measurements $\vec{z_\kappa}$ we need to test the accuracy of the sensors of our radar or read a manual that they have. In our work we do not have this kind of information and so we will run Kalman filter for different values of σ_x^2 and σ_y^2 and we will see how it responds. We will run our code for $\sigma_x^2 = \sigma_y^2 = 10^{-2}, 1, 100$.

- The process noise covariance $Q_{\kappa-1} \in R^{4x4}$. This is the most critical initialization of the tuning process. The choice of the noise covariance $Q_{\kappa-1}$ captures all the model uncertainties, inaccuracies and also the noise of the process. In short, the process noise covariance measures how much we trust the calculations of our model. In this work we will initialise the process error to be a realistic number which we will define as:

$$Q_{\kappa-1} = \begin{bmatrix} \frac{\Delta t^4}{4} & 0 & \frac{\Delta t^3}{2} & 0 \\ 0 & \frac{\Delta t^4}{4} & 0 & \frac{\Delta t^3}{2} \\ \frac{\Delta t^3}{2} & 0 & \Delta t^2 & 0 \\ 0 & \frac{\Delta t^3}{2} & 0 & \Delta t^2 \end{bmatrix} \cdot 10^{-6}$$

where the time step $\Delta t = 5$. If we put a much smaller error then our filter is lagging and delays significantly to catch up with the observations. When $Q_{\kappa-1}$ is large, the Kalman Filter tracks large changes in the data more closely than for smaller $Q_{\kappa-1}$. We will keep this number fixed and we will change the error in the observations to see how the Kalman Filter responds. Another way to define the process noise covariance matrix is:

$$Q_{\kappa-1} = I \cdot 10^{-6}.$$

It does not matter how will we define $Q_{\kappa-1}$ as long as is **time invariant** as the priori covariance P_κ and the posteriori covariance $\widehat{P}_\kappa$ will either way converge to a **steady state** after a few steps. The priori covariance P_κ displays the possible range of the predicted state $\vec{x_\kappa}$ while the posteriori covariance $\widehat{P}_\kappa$ displays the uncertainty of our final updated estimation $\widehat{x}_\kappa$. Notice that in the following cases we have defined $Q_{\kappa-1}$ to be time invariant.

Ideally we would like to achieve a good balance between the error in the observation R and process noise $Q_{\kappa-1}$. The first measures our trust in observations and the other measures our trust in our model process. For this kind of balance we will talk in the conclusion section.

2.0.2 Before numerical computation

We need to mention here that at the following section where we track only the center of the biggest ellipse some of the jumps in the data are not because of the failure of the radar sensors.

At each time step our algorithm calculates the surface of all the ellipses and chooses the biggest one to track. As we mentioned earlier, storm sells split and merge all the time and so at one time step we have the biggest ellipse and then some smaller ellipses may merge and construct an even bigger ellipse at the next time step. Equivalently our biggest ellipse may split into smaller ellipses and therefore at the next time step we have an other ellipse as the biggest to track. The new biggest ellipse is constructed between the time period of our radar observations.

There are some days in our data that the biggest ellipse remains the biggest ellipse through the day and so we do not have big jumps. We will use these **stable** days to see how Kalman filter works under different parameters for the noise of the measurements.

In some other days the ellipses - storm cells merge and split and in fact it is like tracking the center of a different ellipse (which now is the biggest one) that suddenly appeared in the next time step. But these days are not useless in our problem. We will try to overcome this difficulty by using the Kalman filter to track the center of **multiple** ellipses. We will tune our code to track the center of the three biggest ellipses at each time step.

Now we will apply the Kalman filter in the **stable** days to see how it evaluates under different parameters. Next we will apply the Kalman filter for **multiple** tracking.

2.0.3 Track the center of the biggest ellipse

Case 1 : $\sigma_x^2 = \sigma_y^2 = 10^{-2}$

Putting a small value as the observation error means that we are tuning Kalman filter to trust a lot our sensors. We are looking forward to see that Kalman filter will fit our observation measurements almost perfectly.

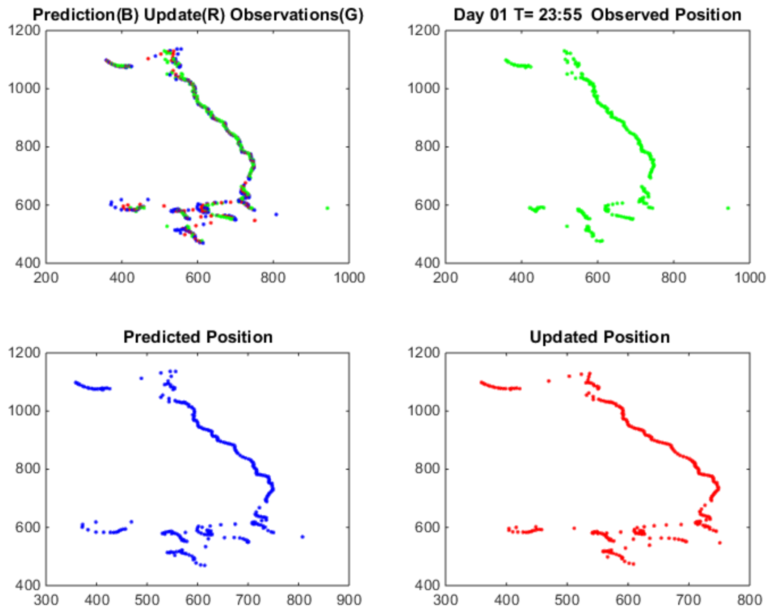

Figure 2.1: Tracking of the center of Biggest Ellipse : Day 1

In the figure above we can see that the Kalman filter fits the observed data well as we expected. But is this the true route of the center of the storm cell?

The Met office says that the radars used for forecast are very sensitive and prone to errors. They can be affected by daylight, nearby trees and buildings, level ground or even by an aeroplane that flies at that time above the radar.

So as you understand, in this case we putted a very small and unrealistic value as the error of the sensors and we trusted too much our observations.

Lets see what happens to the priori covariance matrix P_κ and to the posteriori covariance $\widehat{P}_\kappa$.

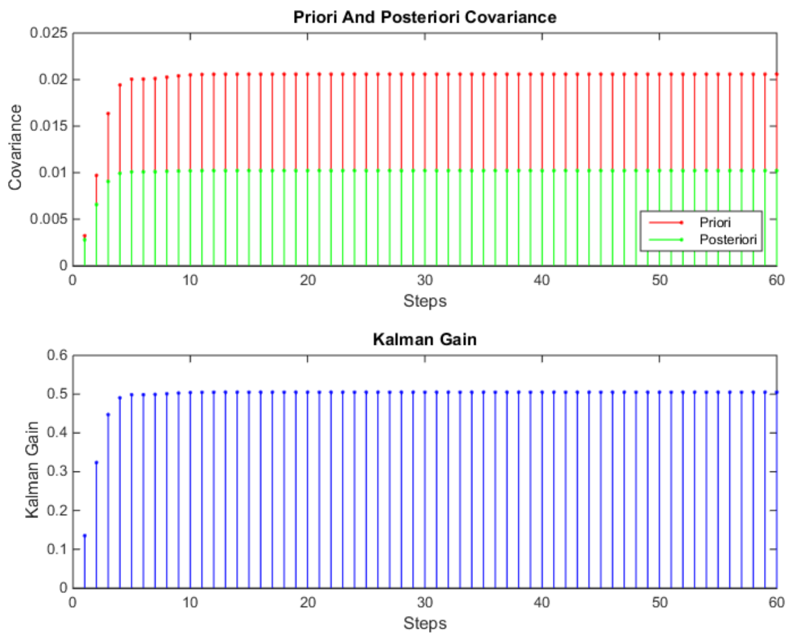

Figure 2.2: Covariances and Kalman gain converge to a steady state for $\sigma_x^2 = \sigma_y^2 = 10^{-2}$

We can see that after some steps both priori and posteriori covariance converge to a steady state. But what the above graph actually means? Shortly, it shows that our system is in balance, which means that KF uses both the model process **and** the observations for the final estimation of the state $\widehat{x}_\kappa$. The priori covariance P_κ shows the amount of uncertainty that we had before the update from the observations. Then our algorithm calculated the Kalman gain which used the noisy observations in order to correct the prior uncertainty into the posterior uncertainty. Finally, the posteriori covariance $\widehat{P}_\kappa$ shows our uncertainty about the final estimation.

The Kalman gain index for the position converge to a number close to 0.5051 which in fact is the ratio between the model usage versus how much observation used. We can say that in this particular case where $\sigma_x^2 = \sigma_y^2 = 10^{-2}$ the Kalman filter has a good balance of both model and observations.

In order to proceed we need to define two estimation errors. We will call as:

- A priori estimate error: $e_\kappa^- = x_\kappa^* - \vec{x}_\kappa$. This is the error between the observation and the prediction of the model only. At this stage the prediction $\vec{x}_\kappa$ has not yet updated with the new observations.

- A posterior estimate error: $e_\kappa^+ = x_\kappa^* - \widehat{x}_\kappa$. This is the error between the observation and Kalman filter final estimation after updating with the observation. Of course we expect this error to be smaller than the previous one.

Where x_κ^* is the observation. Notice that x_κ^* is also a vector which has as index the variables that we want to measure. For now these variables are the position x and the position y. In the following table we present the priori and posteriori estimations and the errors only for the x position. Analogous results we have for y position. Notice that the following table is on the estimations that KF did after the convergence of the Kalman gain.

Table 2.1: Estimations and Errors

x_κ^*	$\vec{x}_\kappa$	$\widehat{x}_\kappa$	e_κ^-	e_κ^+
380.87	381.68	381.27	0.812832	0.404029
388.44	384.68	386.58	3.766409	1.867543
390.34	390.64	390.49	0.308735	0.152888
393.96	394.50	394.23	0.546871	0.270678
399.39	398.15	398.77	1.243523	0.615403
410.69	402.91	406.84	7.782910	3.851574
399.49	412.35	405.85	12.860594	6.364396
404.74	409.10	406.89	4.361953	2.158617
408.60	409.37	408.98	0.768083	0.380101
414.42	411.33	412.89	3.090509	1.529390
417.92	415.77	416.86	2.148488	1.063210

Now we can observe three things. The first is that the Kalman filter did a very good prediction $\vec{x}_\kappa$. The second is that the error of the updated estimation e_κ^+ is smaller than the error in the prediction step e_κ^- as we expected. The second is that e^+ is approximately the half of the e_κ^- and so:

$$e_\kappa^+ \approx 0.5 \cdot e_\kappa^-$$

We said earlier that the Kalman gain entry for the position converge to 0.5051. Lets take the last entry of our table:

$$e_{11}^- \cdot K_1 = 2.148488 \cdot 0.5051 = 1.0852012888 \approx 1.063210 = e_{11}^+ \tag{2.1}$$

$$e_\kappa^- \cdot K_1 \approx e_\kappa^+ \tag{2.2}$$

Equation (2.2) of course is a sloppy equation from a mathematical aspect. As we said earlier Kalman gain is the ratio of the model usage against the observation usage. Even if equation (2.2) is correct and also holds for equality, we should not expect that our numerical calculations in the equation (2.1) would be equal because of the machine **round-off errors**. The calculations in the next case will help us understand if we can create a formula involving the Kalman gain and the estimation errors.

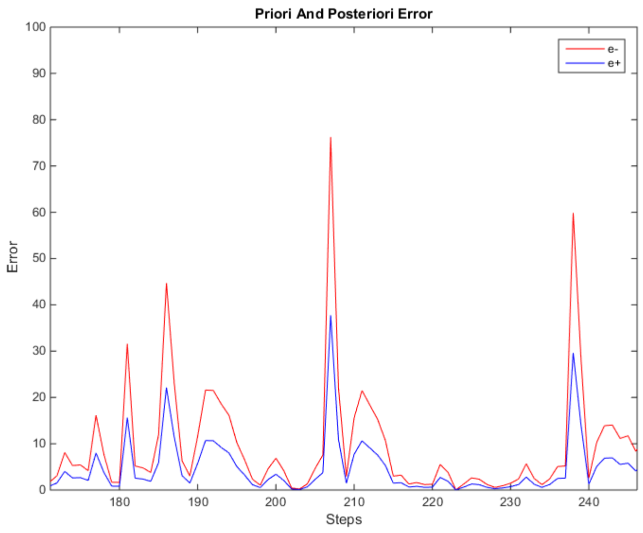

Figure 2.3: A *zoom* in the plotted errors.

As we said earlier, putting a so small variance as the error in the observations is too unrealistic. Lets put a more realistic value.

Case 2 : $\sigma_x^2 = \sigma_y^2 = 1$

In this case we put a bigger value as the observation error and so we expect to see Kalman filter not to follow the observation as much as before.

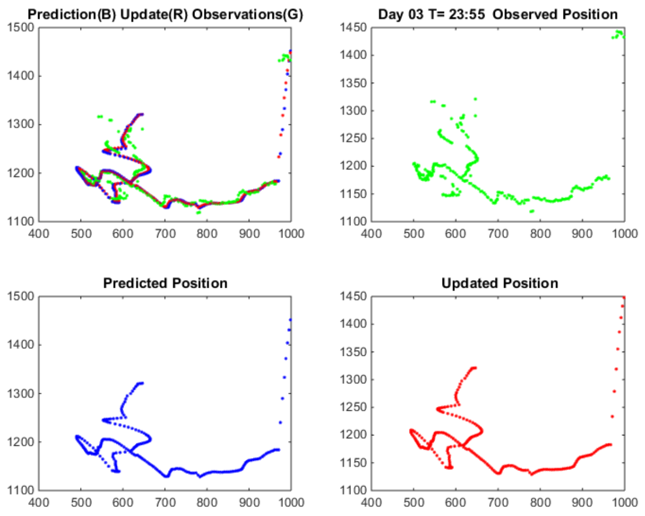

Figure 2.4: Tracking of the center of Biggest Ellipse : Day 3

Now that we have more uncertainty in our observations we also expect that the covariances will be bigger than before and the Kalman filter will not make a large correction in the update step. The amount of this correction is also captured in the bellow figure.

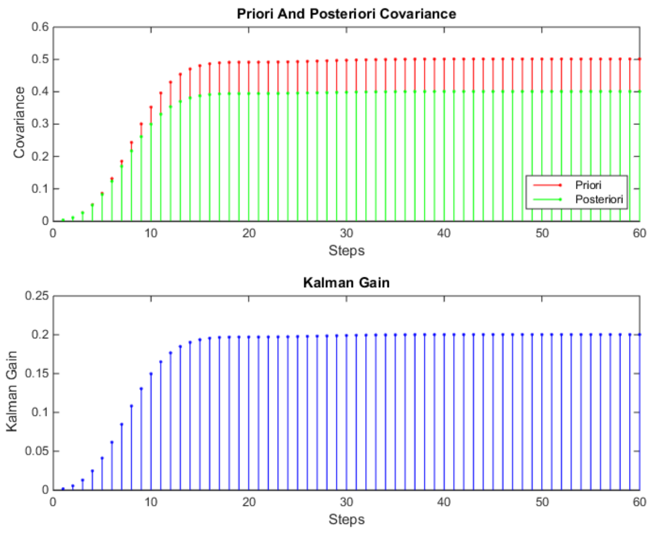

Figure 2.5: Covariances and Kalman gain converge to a steady state for $\sigma_x^2 = \sigma_y^2 = 1$.

Notice that the priori covariance is closer to posteriori covariance than before. This mean that in the update step we have bigger variance than before and the KF trusted more the model than the observations. Also it takes more time for both the Kalman gain and the covariances to converge to a steady state than before. In the following table we present the first estimations before the convergence of the Kalman gain and we also present later estimations made after the convergence of the Kalman gain and we will further discuss these differences.

Table 2.2: Estimations and Errors

k	x^*_κ	$\vec{x}_\kappa$	$\widehat{x}_\kappa$	e^-_κ	e^+_κ
1	646.37	646.37	646.37	0.000000	0.000000
2	542.48	646.37	645.81	103.892000	103.327742
3	549.12	645.52	644.27	96.398496	95.157593
4	568.90	643.51	641.67	74.615239	72.773294
...	...	...	...	...	...
25	560.16	577.62	574.17	17.456368	14.009395
26	560.44	570.36	568.40	9.919251	7.958300
27	561.84	564.38	563.88	2.536010	2.033983
28	564.37	559.80	560.70	4.572024	3.665631
29	565.12	556.72	558.39	8.393853	6.727338
30	565.68	554.60	556.80	11.079969	8.877022

In this case where we have larger uncertainty in the observations, we also have smaller amount of correction for the updated estimation than the previous case. Especially in the first iterations where the Kalman gain has not converge yet we see that the amount of correction is smaller than the late iterations. Lets see now if the equation (2.2) from the previous case holds.

$$e^-_\kappa \cdot K_1 \approx e^+_\kappa$$

Now Kalman gain converge to the number 0.200277510282359. We will work at the 30th iteration from the above table but it is already obvious that the equation (2.2) does not hold now. So is it wrong? No it is not! We just got a bit confused in the previous case because Kalman gain was a number close to 0.5 (half). So now we apply equation (2.2) in our 30th iteration to obtain the following:

$$e_{\kappa}^{-} \cdot K_1 = 11.079969 \cdot 0.200277510282359 = 2.219068605325719 \neq e_{\kappa}^{+}$$

Now notice that:

$$e_{\kappa}^{+} + e_{\kappa}^{-} \cdot K_1 = 8.877022 + 2.219068605325719 = 11.096090605325719 \approx e_{\kappa}^{-}$$

Which can be rearranged to the final form as:

$$e_{\kappa}^{-} - e_{\kappa}^{-} \cdot K_1 \approx e_{\kappa}^{+} \tag{2.3}$$

$$e_{\kappa}^{-}(1 - K_1) \approx e_{\kappa}^{+} \tag{2.4}$$

In an ideal case with no round off errors we can expect that the equation (2.4) holds for equality. Moreover we can rearrange equation (2.4) into:

$$\frac{e_{\kappa}^{-}}{e_{\kappa}^{+}} = \frac{1}{(1 - K_1)} \tag{2.5}$$

Now it is clear that as K_1 goes to 1 then the ratio of the priori and posteriori errors goes to infinity. Later we will see that this happens because of the priori state covariance matrix. If the trace of the matrix P_{κ} diverge then the Kalman gain will be close to 1 and our filter will fail. In other words it means that we had huge uncertainty for the prediction estimation $\vec{x}_{\kappa}$ and so Kalman filter will follow perfectly the observation. In practise our model fails and so KF follows only the observations.

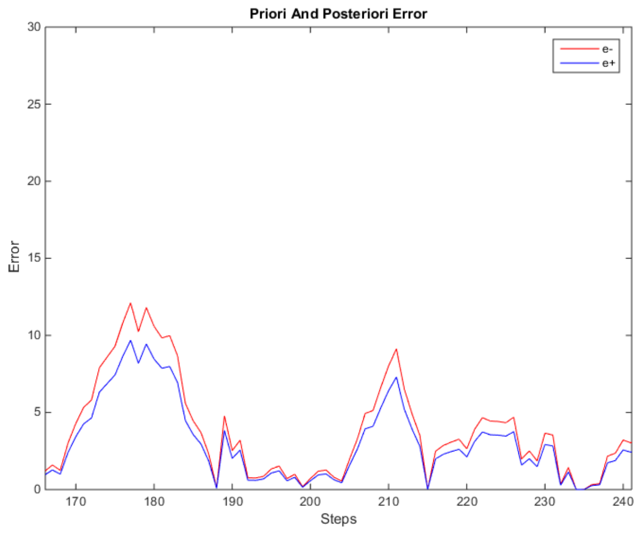

Figure 2.6: A *zoom* in the plotted errors.

Bigger error in observations means bigger uncertainty and so bigger covariance. In this case we have a more realistic route of the center of the biggest ellipse. Now Kalman filter does not follow exactly the observed data because we have increased the covariance of the sensors. By increasing the covariance of the sensors σ_x^2 and σ_y^2 we create a larger possible space for the observed data to be. The Kalman filter follows less and less the observed data as the observation error grows.

Case 3 : $\sigma_x^2 = \sigma_y^2 = 100$

Now that have a large value as the variance in the observation we expect that the Kalman filter will rely much more on the model than in the observations.

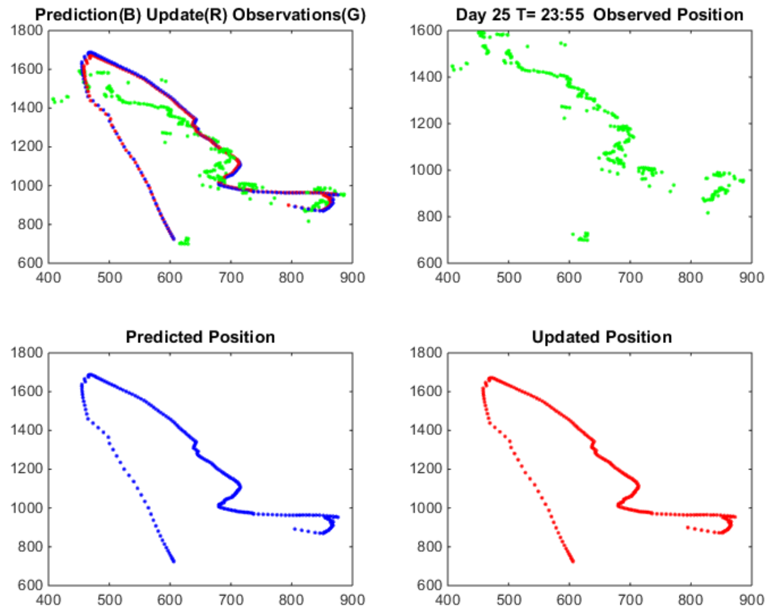

Figure 2.7: Tracking of the center of Biggest Ellipse : Day 25

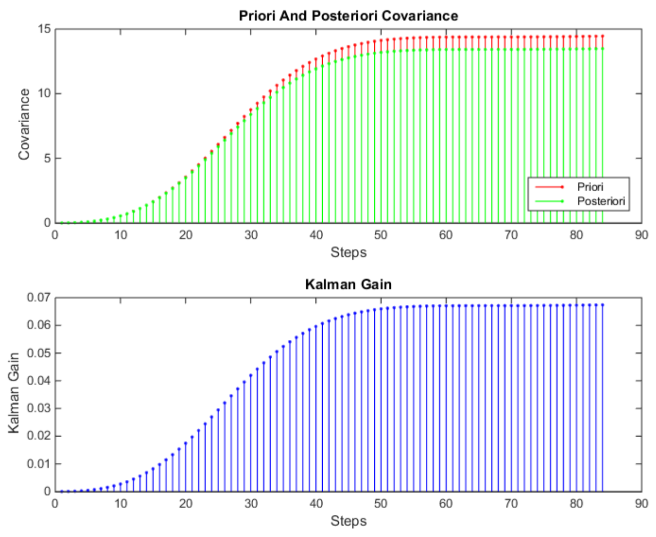

Figure 2.8: Covariances and Kalman gain converge to a steady state for $\sigma_x^2 = \sigma_y^2 = 100$.

Table 2.3: Estimations and Errors

k	x_κ^*	$\vec{x}_\kappa$	$\widehat{x}_\kappa$	e_κ^-	e_κ^+
1	605.92	605.92	605.92	0.000000	0.000000
2	625.83	605.92	605.93	19.904000	19.902912
3	629.05	605.93	605.93	23.125352	23.122317
4	627.15	605.93	605.94	21.217601	21.212134
...	...	...	...	...	...
276	835.74	861.73	859.96	25.993726	24.219260
277	822.12	860.98	858.33	38.863369	36.210356
278	822.83	859.26	856.77	36.432682	33.945599
279	820.20	857.62	855.06	37.416007	34.861798
280	826.86	855.81	853.84	28.958915	26.982030
281	831.12	854.52	852.92	23.396265	21.799115

As you can see we have a very small correction in our update $\widehat{x}_\kappa$ state because Kalman fitler does not trust the observation very much in this case. The truth is that KF trust much more the model for the final updated estimation. Now Kalman gain converge to 0.068265145102466. Lets see now if the equation (2.4) holds in this case. We will take the last entry of the above table. So we have that:

$$e^{-}_{281}(1 - K_1) = 23.396265 \cdot (1 - 0.068265145102466.) = 21.799115574919256 \approx e^{+}_{281}$$

So equation (2.4) holds.

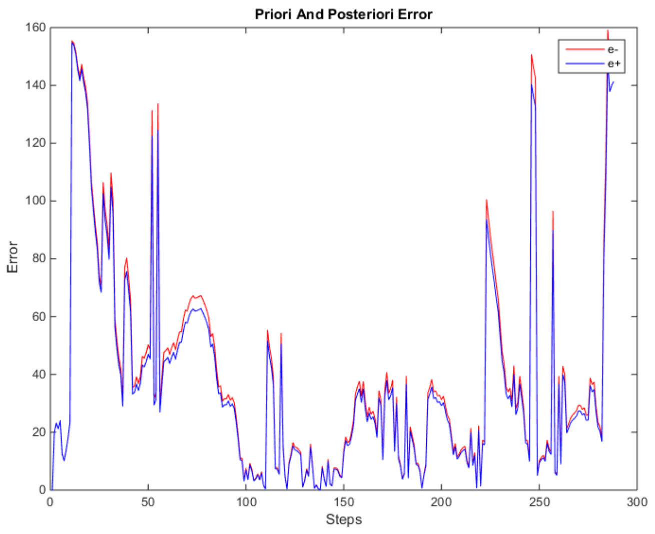

Figure 2.9: The posteriori error is very close to the priori error.

2.0.4 Time variant process noise Q

In this section we will define the uncertainty of the process - the process noise covariance Q to be time variant. This mean that at each step of the Kalman filter we will introduce more uncertainty than the previous step. We will define Q as before:

$$Q_{\kappa-1} = \begin{bmatrix} \frac{\Delta t^4}{4} & 0 & \frac{\Delta t^3}{2} & 0 \\ 0 & \frac{\Delta t^4}{4} & 0 & \frac{\Delta t^3}{2} \\ \frac{\Delta t^3}{2} & 0 & \Delta t^2 & 0 \\ 0 & \frac{\Delta t^3}{2} & 0 & \Delta t^2 \end{bmatrix} \cdot 10^{-6}$$

But this time we will update matrix Q inside the recursion of the Kalman filter with $\Delta t = 5, 10, 15..$

In this case we will put as error in the observations $\sigma_x^2 = \sigma_y^2 = 100$. Our previous calculations showed that when Q is time invariant and we put a big number as the error in the observation then the Kalman filter will trust more the model process and less the observations. But lets have a look in the following figure:

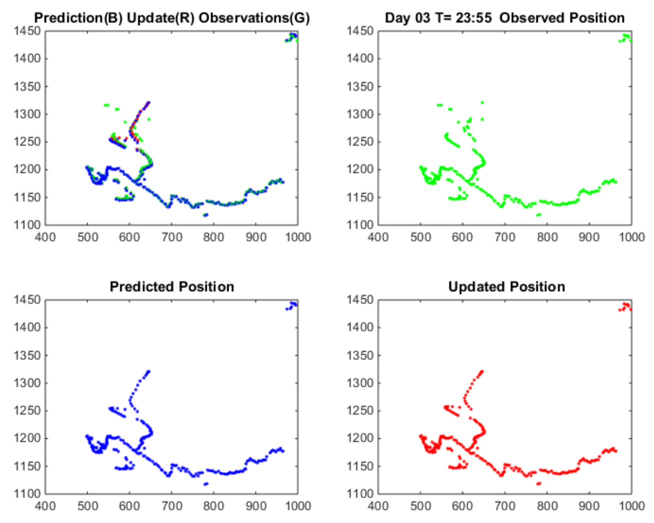

Figure 2.10: Tracking Day 1 with time variant process noise Q

We can see that only in the first few steps we have **filtered** tracking. After a while we have exact tracking at the observations and of course this is not the required result because the role of KF is to filter the noise. We can see that very fast our model fails and the KF start to track only the observations. This happens because of the extra amount of the uncertainty that we introduce at each time step to our algorithm and so the model quickly become unreliable.

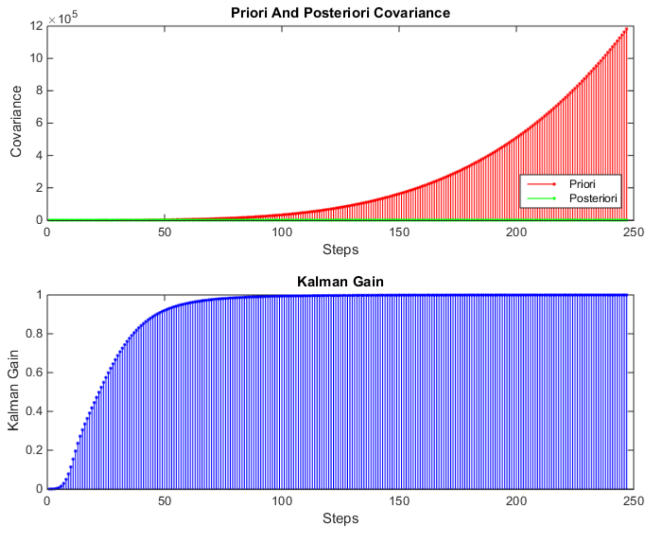

Figure 2.11: Priori covariance diverge and Kalman gain converge to 1 for time variant Q.

Now Kalman gain converge to a number very close to 1 as expected.

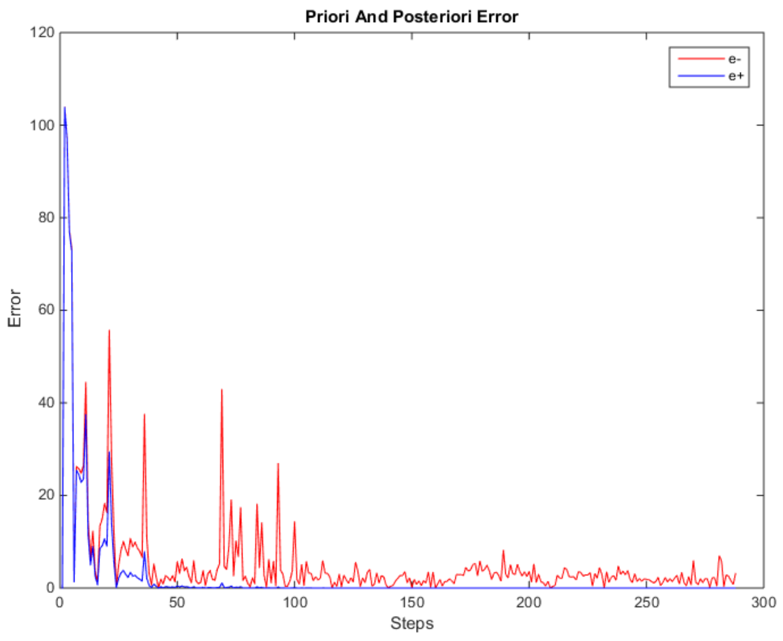

Figure 2.12: The posteriori error converge to 0.

The posterior error converge close to zero after the first iterations. The Kalman filter does not follow the model.

2.0.5 Track the center of multiple Ellipses

For some days in our data we have a problem. Between the time steps some ellipses merge or other split and then we track a different ellipse as the biggest one. In order to overcome this difficulty we are going to run Kalman filter to track the center of the 3 biggest ellipses - storm cells at each time step.

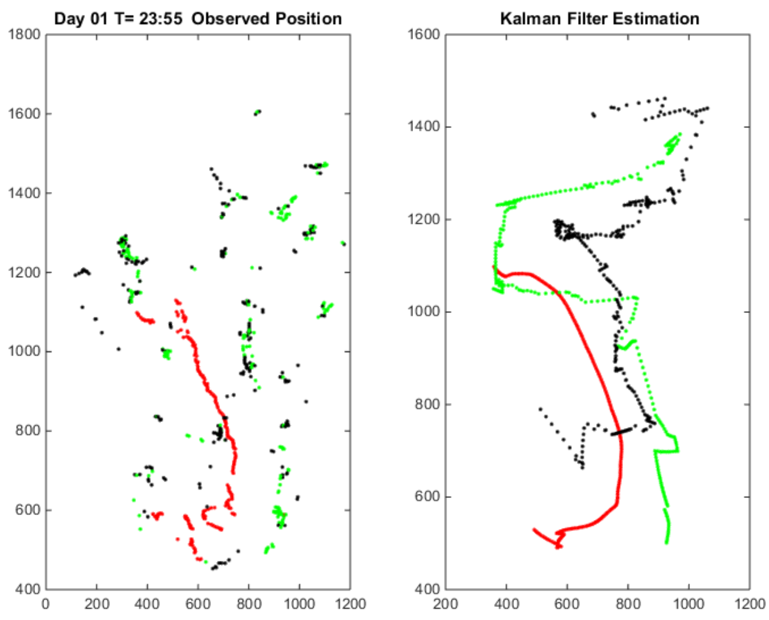

Figure 2.13: Radar Measured position and Kalman Fitler.

In the figure above we see that there is still some interchange between colours. We clearly see that at some time steps the 2nd biggest ellipse (black) became the 3rd biggest ellipse (green) and vice versa. So tracking multiple ellipses did not solved our problem. This happens because of two reasons: The first is that there might be a merge between the 3rd ellipse with another ellipse (not calculated) and so the 3rd ellipse becomes the 2nd biggest in some steps. The second reason is because of the randomness of the weather which means that the 2nd biggest storm cell might rain in some point of time and so became a bit smaller than before giving the 2nd biggest position to the 3rd ellipse and vice versa.

However, we will present this situation in a way to show another important ability of the Kalman Filter: The ability to track objects and display their route even if we have very sparse observations to update it. In the figure above, on the left, we can see that, except the red observed position which is smooth, the other two black and green are not smooth and we have some data about them randomly and sparsely distributed. We will put a big number as the error of the observations i.e $\sigma_x^2 = \sigma_y^2 = 100$ and we will keep the process noise fixed i.e $Q_{\kappa-1} = I \cdot 10^{-6}$ as usual. Remember that in this case we only know what happens at the time of the radar observation and we want to know what happens between these observations. What happened between the observations and how the storm cell moved? Lets see our next figure.

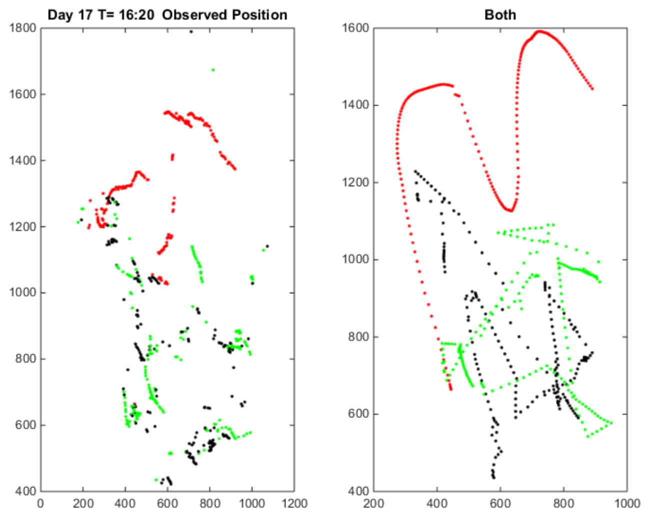

Figure 2.14: Tracking the 3 biggest ellipses

In cases that we have very sparce observations, Kalman filter is able to show a possible route that the tracking object used to move between the observations. An other thing that we can say from our experience so far is that at each day there is one ellipse which is the biggest one and the other ellipses changes between 2nd the 3rd biggest place.

2.0.6 Track the whole ellipses

In this section we will use the Kalman filter to track the whole ellipse and not only the center as we did in the previous sections. In order to do that we need first to do some modifications in our algorithm. Before, we wanted to track only the center of the ellipse and so we used a state vector $\vec{x}_\kappa \in R^4$ with index the (x, y) position and $\dot{x}$ and $\dot{y}$ the corresponding velocities. Now that we want to track the whole ellipse we have many more parameters to put as index in the state vector. So our state vector will have the following parameters as an index:

- The x position and the corresponding velocity $\dot{x}$.
- The y position and the corresponding velocity $\dot{y}$.
- The angle θ of the ellipse and the corresponding angle velocity $\dot{\theta}$.

- The big semi axis r_a of the ellipse and the corresponding velocity $\dot{r_a}$.
- The small semi axis r_b of the ellipse and the corresponding velocity $\dot{r_b}$.

So we have 5 parameters and 5 velocities to track. As you understand, now we will have a state vector $\vec{x}_\kappa \in R^{10}$ which is:

$$\vec{x} = (x, y, \dot{x}, \dot{y}, r_a, r_b, \dot{r_a}, \dot{r_b}, \theta, \dot{\theta}) \in R^{10}$$

We will initialise the other parameters and matrices exactly in the same way that we did before. The only thing that we will change now is the dimensions. So now we have the following matrices:

- The error in the observations will be the matrix $R \in R^{5x5}$ because we have 5 parameters that we want to measure now. It is like we have a sensor for each of the 5 above parameters. Notice that we do not have any observation for the velocities and so they are initialised to zero.
- The process noise covariance $Q_{k-1} \in R^{10x10}$ because we have uncertainty for all the 10 above variables. We will apply our code for $R(i,i) = 100$.

Now we can represent our ellipses in two different ways. As an explicit representation and as an implicit representation and we will discuss which of these two fits best our problem of storm tracking.

Explicit representation

For the explicit representation we will use the ellipse.m file that it takes as input the center, both semi axes and the angle of the ellipse and displays the full ellipse.

Implicit representation

For the implicit representation we practically evolve a function through time and the final ellipses are displayed from a level set function. We will see that in our case the implicit representation is much more useful but lets first see what the implicit form is:

The implicit equation of an ellipse at the center is:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \qquad a, b \in R \tag{2.6}$$

Now we will use translation of coordinate system by (s, t) and also we will define the rotation matrix as following:

$$T_{s,t}(x, y) = (x - s, y - t)$$

$$R_\theta = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & \cos\theta \end{bmatrix}$$

So now we will create the general implicit form by first translate the coordinate system and then rotate.

$$E[a, b, \theta, s, t](x, y) = E_{a,b}(R_\theta(T_{s,t}(x, y))) =$$

$$\frac{((x - s)cos(\theta) + (y - t)sin(\theta))^2}{a^2} + \frac{((x - s)sin(\theta) - (y - t)cos(\theta))^2}{b^2} = 1 \qquad (2.7)$$

Now we will define a basis function as the following:

$$\Gamma[a, b, \theta, s, t](x, y) = E[a, b, \theta, s, t](x, y) - 1$$

Which can be rearranged as the following function:

$$\Gamma[a, b, \theta, s, t](x, y) = \frac{((x - s)cos(\theta) + (y - t)sin(\theta))^2}{a^2} + \frac{((x - s)sin(\theta) - (y - t)cos(\theta))^2}{b^2} - 1 \qquad (2.8)$$

Now a cloud of elliptic shape can be represented as:

$$S_{ell} = \{(x, y) \in R^2 : \Gamma[a, b, \theta, s, t](x, y) \leqslant 0\}$$

Finally we can define a general cloud formation by superposition:

$$\Psi(x, y) = \min\big(\Gamma[a, b, \theta, s, t](x, y)\big)$$

And so:

$$S_{dom} = \{(x, y) \in R^2 : \Psi[a, b, \theta, s, t](x, y) \leqslant 0\}$$

In practise if we introduce in MATLAB the implicit representation of the ellipse then the visualised output will be a cone. We will get the required ellipse by creating a level set function which is like taking the intersection of this cone with the plane.

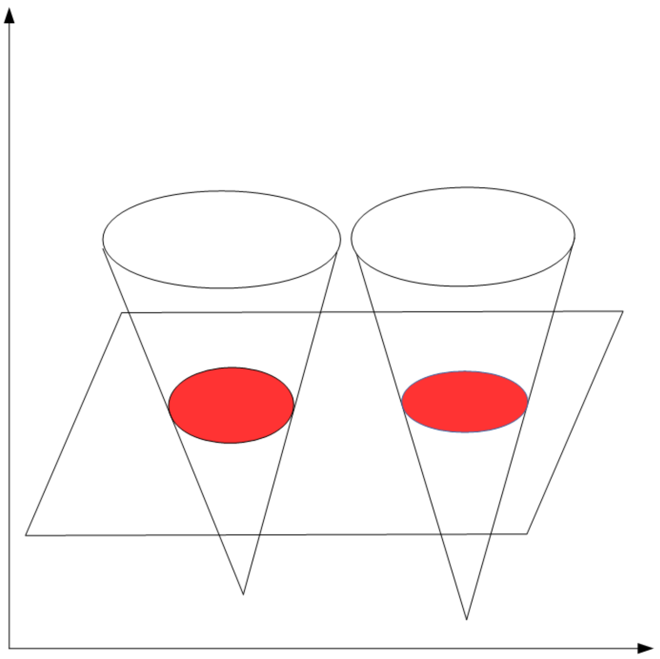

Figure 2.15: The level set function displays the required ellipses

Now we are ready to present our ellipses. We will try to track the 10 biggest ellipses at each time step.

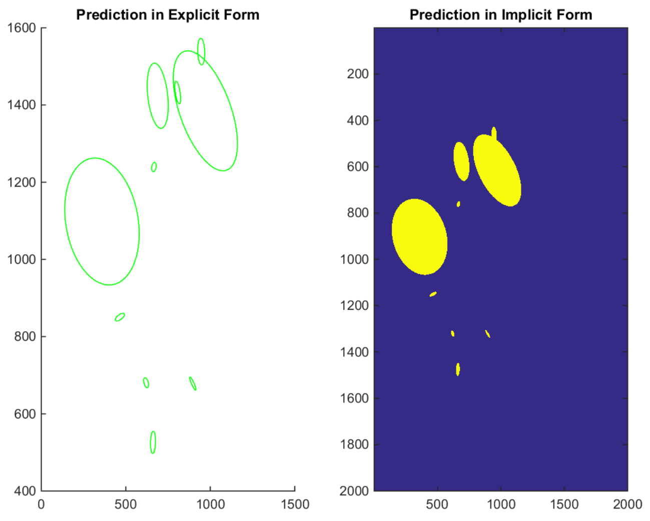

Figure 2.16: Explicit and Implicit representation for the prediction step

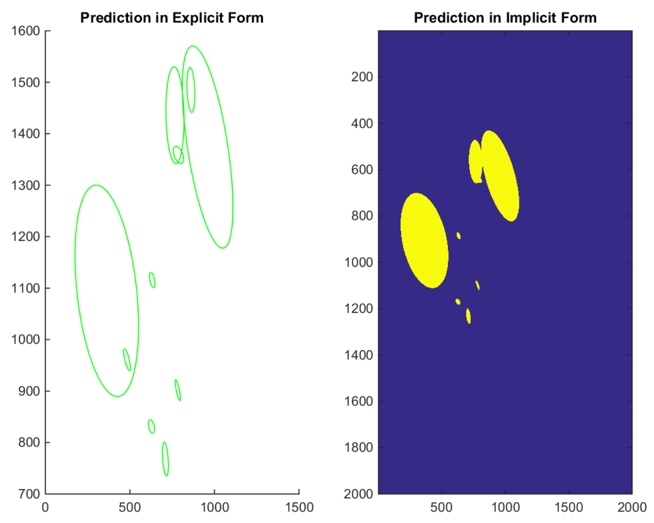

Figure 2.17: Explicit and Implicit representation for the prediction step

From the above figures we can see that there is a difference in the way the storm cells are represented. We can see that in the explicit representation we have some ellipses inside other bigger ellipses. But remember that in our work we are working on storm cells. From a natural aspect it does not make sense to track a storm cell inside another storm cell. So the radar gives pictures of ellipses inside other ellipses which is equivalent to storm cells inside other storm cells. How this happens? Maybe the small storm cell was closer to the radar sensor and the information come back to the radar faster than the information that came from the bigger storm cell. Another possible case is that the radar captures black and grey pictures of the storm cells and then modifies them into ellipses. Maybe the small storm cell was darker or lighter than the bigger one and so this difference captured. Now we present the following slide-show figures for 8 consecutive steps of our algorithm.

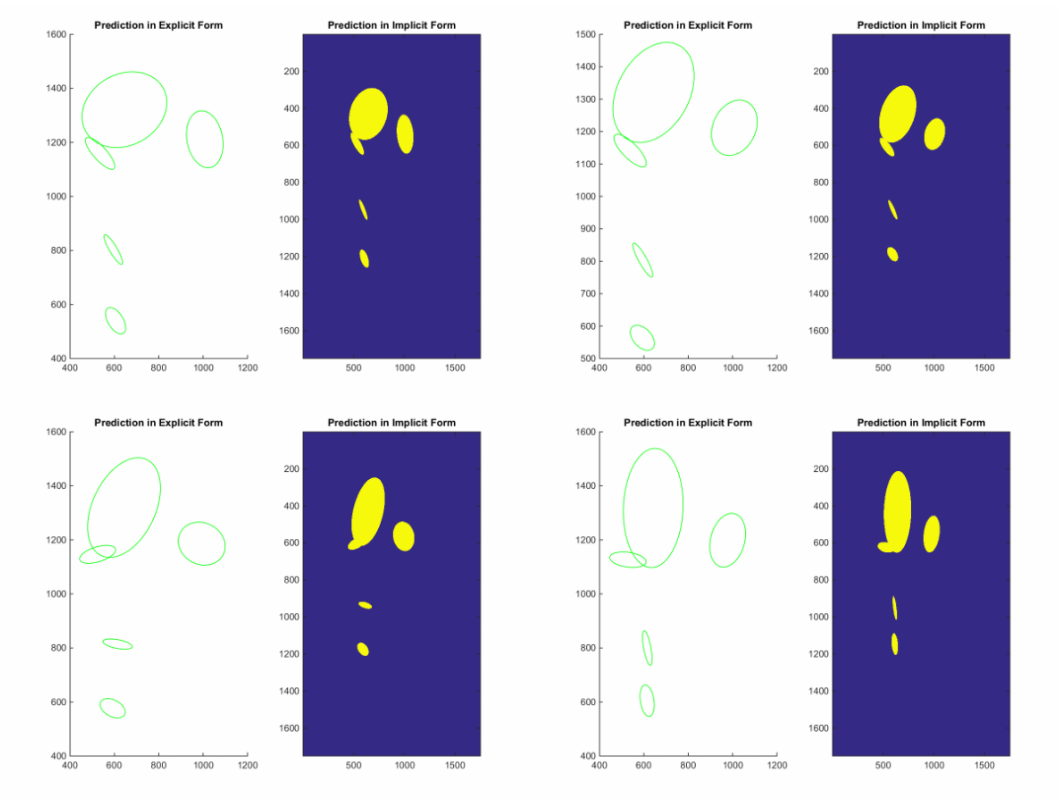

Figure 2.18: Four consecutive time steps for the 5 biggest ellipses

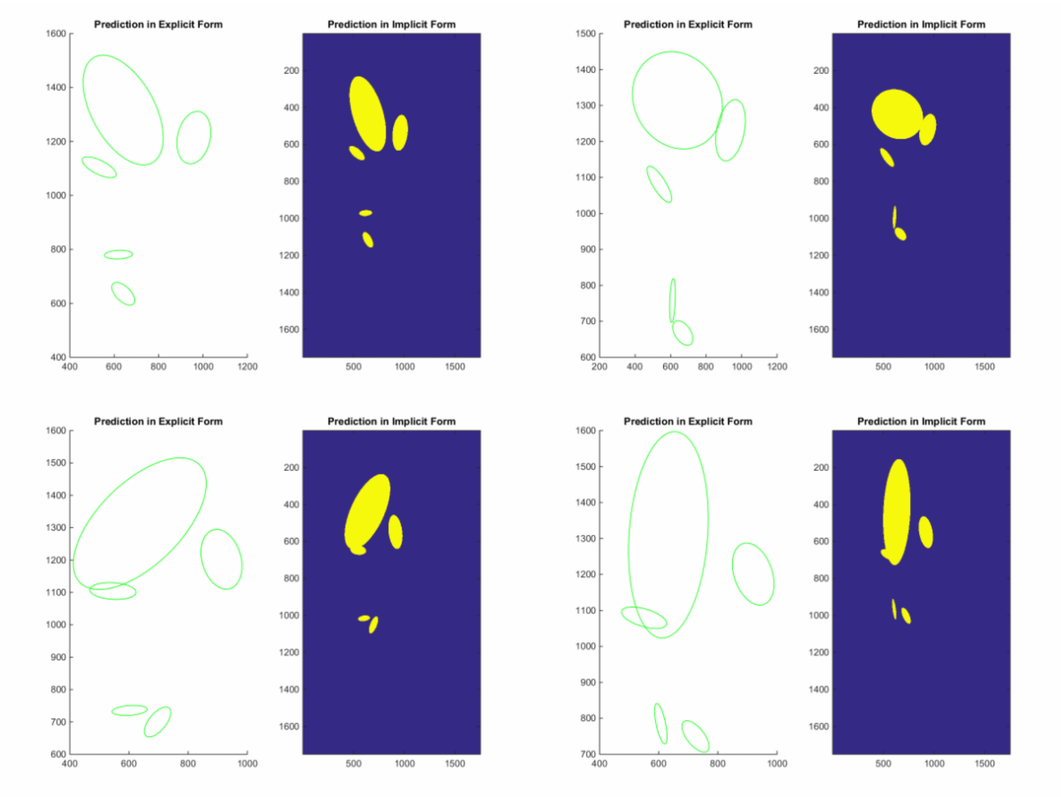

Figure 2.19: Four consecutive time steps for the 5 biggest ellipses

Chapter 3

Conclusions

We successfully applied the Kalman filter in order to track the center and the whole ellipses. We also tested how the KF performs under different parameters and at the same time we tried to figure out which of these parameters fits better our storm cell tracking problem. We can not say for sure which parameter for the error in the observation is closer to the reality because the covariance of the sensors of a radar is known by the developer of the radar or by a manual that comes with the radar. In our work we did not have these kind of information. But taking seriously the sensitivity of the forecast radars we would recommend a number close to $R(i,i) = 100$. We can say that the choice of 10^{-6} as coefficient for the error in the process $Q_{\kappa-1}$ was satisfactory.

We also verified the following things about the Kalman filter:

- The priori and posteriori covariance converge to a steady state for time invariant process noise Q.
- The Kalman gain converge to a steady state always. It is closer to 1 for accurate observation or model failure while it is closer to 0 for noisy observations or accurate model.
- The posteriori error of the updated estimation is always smaller that the priori error of the prediction.
- We found a sloppy correlation between the priori and posteriori error which involves the Kalman gain entry for the corresponding measurement: $e^-_\kappa(1-K_1) \approx e^+_\kappa$.

- The implicit form for the representation of the ellipses is a more realistic way to fit the requirements of storm cell tracking.

For any future work on storm tracking with Kalman filter we would recommend the following:

- Know the covariance of the radar sensors for more accurate and realistic predictions.

- Try to demonstrate that the final state estimation is bounded between one or two standard deviations which are given by : $\sigma_\kappa(i) = \sqrt{P_k(i,i)}$

- Improve the algorithm speed of implicit representation. In order to improve the speed of our algorithm we can modify our program to ignore and stop tracking-predicting a small ellipse that is inside a bigger ellipse.

Bibliography

[1] P.J. Rossi, V. Chandrasekar, V. Hasu, D. Moisseev, 2015. Kalman Filtering-Based Probabilistic Nowcasting of Object-Oriented Tracked Convective Storms. Journal of atmospheric and oceanic technology, Volume 32

[2] T. Babb, 2012. http://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/

[3] S. Gamse, F. Nobacht-Ersi, M. Sharifi, 2014. Statistical Process control of a Kalman Filter Model. Open access Sensors, Volume 14

[4] Prof. David M. Schultz, Chair in Synoptic Meteorology, Centre for Atmospheric Science, School of Earth, Atmospheric and Environmental Sciences, The University of Manchester. http://weather.seaes.manchester.ac.uk/schultz/

[5] Dr Jonathan Fairman, School of Earth, Atmospheric and Environmental Sciences, The University of Manchester. http://personalpages.manchester.ac.uk/staff/jonathan.fairman/bio.html

Appendix A

MATLAB Code

A.0.1 Code for tracking and predicting the center

```
% written by Nikolaos Theodorakopoulos, The University of
    Manchester, UK,
% nikostheodorakopoulos1990@hotmail.com
% Special Thanks to Dr. Oliver Dorn for his notes and
    corrections.
%%%%%%%%%%%%%---Kalman Center Only-----%%%%%%%%%%
kfinit=1
O=0
dt = 5;
noise =0.001; %Process noise big noise=time lag
tkn_x = 1;  %Covariance of sensor in X
tkn_y = 1;  %Covariance of sensor in Y
Ez = [tkn_x 0; 0 tkn_y]; %Error in the observations Matrix
Q= [dt^4/4 0 dt^3/2 0; ...
     0 dt^4/4 0 dt^3/2; ...
     dt^3/2 0 dt^2 0; ...
     0 dt^3/2 0 dt^2]*noise^2 %Process Noise
Ppost = Q; % Initialise Priori Covairance
F = [1 0 dt 0; 0 1 0 dt; 0 0 1 0; 0 0 0 1]; %State update
    matrice
```

```
H = [1  0  0  0;  0  1  0  0];

 for  day  =1:31
     close  all
   for  hour  =  0:  23
      for  shot  =  0:5:55
P  =  sprintf('201512%s%s%s_all.txt',num2str(day,'%02i'),num2str
    (hour,'%02i'),num2str(shot,'%02i'))

A= importdata(P);

 for  i=1:size(A.data)

A.data(i,3);  %Center  X
A.data(i,4);  %Center  Y
A.data(i,7);  %Orientation  of  the  ellipse
A.data(i,10);  %Semimajor  axis  length  in  X  direction  (Cartesian
    )
A.data(i,11);  %Semimajor  axis  length  in  Y  direction  (Cartesian
    )



k(i)=A.data(i,10)*A.data(i,11)*pi;  %Surface  of  Ellipses
end

%%%Comment  this  2  lines  for  multiple  tracking%%%
%%%%%%%-----Find  The  Biggest  Ellipse-----%%%%%%%%%
[num]  =  max(k(:))  ;  %Find  The  location  of  the  Biggest  Ellipse
[x]  =  ind2sub(size(k),find(k==num))
%  %%%%%%%%%-----------------------------------------------%%%%%%%%%

%%%Uncomment  this  2  lines  for  multiple  tracking%%%
```

```
% [sortedValues,sortIndex] = sort(k(:),'descend')  %# Sort the
    values in descending order
% maxIndex = sortIndex(1:3)  %# Get a linear index into A of
    the 10 largest values
%%%%%%%%%%%%%%%%%---------------%%%%%%%%%%%%%%%%%%


 O=O+1;

%%%%%-------Kalman Filter for Center-------%%%%%%%


 for n=1:1 %Change n range for multiple center tracking

  if kfinit==1
% x=maxIndex(n) %Uncomment for multiple tracking
  mat(n,:) =[A.data(x,3),A.data(x,4),0,0]' ;

  else
            % x=maxIndex(n) %Uncomment for multiple tracking
   mat(n,:) = F * mat2(n,:)'; %%Prediction of state
   figure(1)
 subplot(2,2,1);plot(mat(n,1),mat(n,2),'.b');
 hold on

 subplot(2,2,3);plot(mat(n,1),mat(n,2),'.b');
  title(['Predicted Position' ])
 hold on
  end
 Pprior = F * Ppost * F' +Q; %%Covariance

K = Pprior*H'/(H*Pprior*H' + Ez);%Kalman Gain
```

```

%Update State and covariance estimation.
mat2(n,:)=mat(n,:)' + K*([A.data(x,3),A.data(x,4)]'-H*mat(n,:)
   ')
Ppost =  (eye(4)-K*H)*Pprior;



 subplot(2,2,1);plot(mat2(n,1),mat2(n,2),'.r');
 hold on
 plot(A.data(x,3),A.data(x,4),'.g');
  title(['Prediction(B) Update(R) Observations(G)' ])
 hold on

subplot(2,2,2); plot(A.data(x,3),A.data(x,4),'.g'); % the
   actual tracking
title(['Day ' num2str(day,'%02i') ' T= ' num2str(hour,'%02i')
   ':' num2str(shot,'%02i') '  Observed Position' ])
hold on

subplot(2,2,4);plot(mat2(n,1),mat2(n,2),'.r'); % the kalman
   filtered tracking
title([ ' Updated Position' ])
hold on

end

 kfinit=2;

%%%%%%%%%%%%%%%%-----%%%%%%%%%%%%%%%%%%%%%%
 e1(O)=abs(A.data(x,3)-mat(n,1)); %%%A priori Error
 e2(O)=abs(A.data(x,3)-mat2(n,1));%%%A posteririori Error
```

```

%%%%%——Error Staf——%%%%%
figure(2);
subplot(2,1,1)
 title(['Priori And Posteriori Covariance' ])
 ylabel('Covariance');
 xlabel('Steps');
  legend('Priori','Posteriori','Location','southeast')

stem(O,trace(Pprior),'.R')
hold on
stem(O,trace(Ppost),'.G')
 hold on
subplot(2,1,2)
 title(['Kalman Gain' ])
 ylabel('Kalman Gain');
 xlabel('Steps');
stem(O,K(1,1),'.B')
hold on

  figure(3)
 title(['Priori And Posteriori Error' ])
 ylabel('Error');
 xlabel('Steps');
legend('e−','e+')
plot(e1,'r')
hold on
plot(e2,'b')
hold on

pause (.1)

```

```
k=0;

      end

   end
end
```

A.0.2 Code for tracking whole ellipse

At first you will need the implicit.m and the ellipse.m

```
% written by Nikolaos Theodorakopoulos, The University of Manchester, UK,
% nikostheodorakopoulos1990@hotmail.com
% Special Thanks to Dr. Oliver Dorn for his notes and corrections.
function p=implicit(mat)
p = cell(5, 1) ; %Change this index if you want to track more than 5
kif=0;
for n=1:5 %Change this index if you want to track more than 5


ck=cos(mat(n,9)); %%Calculate cos
sk=sin(mat(n,9)); %% Calculate sin
ak=mat(n,5)^2; %%Semi axes
bk=mat(n,6)^2; %%semi axes

for z=1:1750
   for w=1:1750
 h(z,w)=((1/ak)*((z-mat(n,1))*ck+(w-mat(n,2))*sk)^2  + (1/bk)*((z-mat(n,1))*sk-(w-mat(n,2))*ck)^2) -1   ;
end
 end

 p(n)={h};
   if  kif==0
         PSI=p{1};
   else
   PSI=min(PSI,p{n}); %%%%%% LEVEL SET FUNCTION %%%%%%%
 end

if  n==5 %Change this index if you want to track more than 5
         PSI=rot90(PSI);
       p=imagesc(PSI<=0)
end
kif=1;
 end
end
```

```
function  h=ellipse(ra,rb,ang,x0,y0,C,Nb)

% written by D.G. Long, Brigham Young University, based on the
% CIRCLES.m original
% written by Peter Blattner, Institute of Microtechnology,
    University of
% Neuchatel, Switzerland, blattner@imt.unine.ch



% Check the number of input arguments

if nargin<1,
   ra=[];
end;
if nargin<2,
   rb=[];
```

```
end;
if nargin<3,
   ang=[];
end;

%if nargin==1,
%  error('Not enough arguments');
%end;

if nargin<5,
   x0=[];
   y0=[];
end;

if nargin<6,
   C=[];
end

if nargin<7,
   Nb=[];
end

% set up the default values

if isempty(ra),ra=1;end;
if isempty(rb),rb=1;end;
if isempty(ang),ang=0;end;
if isempty(x0),x0=0;end;
if isempty(y0),y0=0;end;
if isempty(Nb),Nb=300;end;
if isempty(C),C=get(gca,'colororder');end;

```

```
% work on the variable sizes

x0=x0(:);
y0=y0(:);
ra=ra(:);
rb=rb(:);
ang=ang(:);
Nb=Nb(:);

if isstr(C),C=C(:);end;

if length(ra)~=length(rb),
  error('length(ra)~=length(rb)');
end;
if length(x0)~=length(y0),
  error('length(x0)~=length(y0)');
end;

% how many inscribed elllipses are plotted

if length(ra)~=length(x0)
  maxk=length(ra)*length(x0);
else
  maxk=length(ra);
end;

% drawing loop

for k=1:maxk



```

```
if length(x0)==1
   xpos=x0;
   ypos=y0;
   radm=ra(k);
   radn=rb(k);
   if length(ang)==1
      an=ang;
   else
      an=ang(k);
   end;
elseif length(ra)==1
   xpos=x0(k);
   ypos=y0(k);
   radm=ra;
   radn=rb;
   an=ang;
elseif length(x0)==length(ra)
   xpos=x0(k);
   ypos=y0(k);
   radm=ra(k);
   radn=rb(k);
   an=ang(k)
else
   rada=ra(fix((k-1)/size(x0,1))+1);
   radb=rb(fix((k-1)/size(x0,1))+1);
   an=ang(fix((k-1)/size(x0,1))+1);
   xpos=x0(rem(k-1,size(x0,1))+1);
   ypos=y0(rem(k-1,size(y0,1))+1);
end;

co=cos(an);
si=sin(an);
```

```
   the=linspace(0,2*pi,Nb(rem(k-1,size(Nb,1))+1,:)+1);
%   x=radm*cos(the)*co-si*radn*sin(the)+xpos;
%   y=radm*cos(the)*si+co*radn*sin(the)+ypos;
   h(k)=line(radm*cos(the)*co-si*radn*sin(the)+xpos,radm*cos(
       the)*si+co*radn*sin(the)+ypos);
   set(h(k),'color',C(rem(k-1,size(C,1))+1,:));

end;
```

```
% written by Nikolaos Theodorakopoulos, The University of
    Manchester, UK,
% nikostheodorakopoulos1990@hotmail.com
% Special Thanks to Dr. Oliver Dorn for his notes and
    corrections.

%%%%%—Kalman filter init For WHOLE ELLIPSE—%%
 kfinit=1;
dt=5;
F = [1  0  dt  0  0  0  0  0  0  0;
       0  1  0  dt  0  0  0  0  0  0;
       0  0  1  0  0  0  0  0  0  0;
       0  0  0  1  0  0  0  0  0  0;
       0  0  0  0  1  0  dt  0  0  0;
       0  0  0  0  0  1  0  dt  0  0;
       0  0  0  0  0  0  1  0  0  0;
       0  0  0  0  0  0  0  1  0  0;
       0  0  0  0  0  0  0  0  1  dt;
       0  0  0  0  0  0  0  0  0  1];

  H = [1  0  0  0  0  0  0  0  0  0;
        0  1  0  0  0  0  0  0  0  0;
        0  0  0  0  1  0  0  0  0  0;
        0 0 0 0 0 1 0 0 0 0;
        0 0 0 0 0 0 0 0 1 0];


B = [(dt^2/2); (dt^2/2); dt; dt]; %Control Matrix
noise = 0.001;
tkn_x =100;  %measurement noise in thehorizontal direction (x
    axis).
tkn_y =100;  %measurement noise in the horizontal direction (y
     axis).
Ez = [tkn_x 0 0 0 0;
        0 tkn_x 0 0 0;
        0 0 tkn_x 0 0;
        0 0 0 tkn_x 0;
        0 0 0 0 tkn_x]

  O=0

 Q = [dt^4/4 0 dt^3/2 0 0 0 0 0 0 0; ...
         0 dt^4/4 0 dt^3/2 0 0 0 0 0 0;...
         dt^3/2 0 dt^2 0 0 0 0 0 0 0; ...
         0 dt^3/2 0 dt^2 0 0 0 0 0 0;...
         0 0 0 0 dt^4/4 0 dt^3/2 0 0 0;...
         0 0 0 0 0 dt^4/4 0 dt^3/2 0 0;...
         0 0 0 0 dt^3/2 0 dt^2 0 0 0;...
         0 0 0 0 0 dt^3/2 0 dt^2 0 0;...
         0 0 0 0 0 0 0 0 dt^4/4 dt^3/2;...
         0 0 0 0 0 0 0 0 dt^3/2 dt^2]*noise^2

 P0 = Q; % estimate of
for day =3:3
    close all
```

```
   for hour = 0: 23
     for shot = 0:5:55
P = sprintf('201512%s%s%s_all.txt',num2str(day,'%02i'),num2str
    (hour,'%02i'),num2str(shot,'%02i'))

A= importdata(P);

for i=1:size(A.data)

A.data(i,3); %Center X
A.data(i,4); %Center Y
A.data(i,7); %Orientation of the ellipse
A.data(i,10); %Semimajor axis length in X direction (Cartesian
    )
A.data(i,11); %Semimajor axis length in Y direction (Cartesian
    )


k(i)=A.data(i,10)*A.data(i,11)*pi; %Surface of Ellipses
end

[sortedValues,sortIndex] = sort(k(:),'descend')  %# Sort the
    values in descending order
maxIndex = sortIndex(1:5)  %Change this index if you want to
    track more ellipses


%%%--------Kalman 2 FULL ELLIPSE------%%%%%%%%%
for n=1:5 %Change this index if you want to track more than 5
        x=maxIndex(n)
              if kfinit==1
x=maxIndex(n)
```

```
mat(n,:) =[A.data(x,3),A.data(x,4),0,0,A.data(x,10),A.data(x
    ,11),0,0,A.data(x,7),0]' ;

    else
 mat(n,:) = F * mat2(n,:)';

    end
    figure(1)
    title([ 'Prediction in Explicit Form' ])
    subplot(2,2,[1 3])
ellipse(mat(n,5),mat(n,6),mat(n,9),mat(n,1),mat(n,2),'g');
    Ppre = F * P0 * F' + Q;
    %Kalman Gain
K = Ppre*H'/(H*Ppre*H' + Ez);

%Update State and covariance estimation.
mat2(n,:)=(mat(n,:)' +K*([A.data(x,3),A.data(x,4),A.data(x,10)
    ,A.data(x,11),A.data(x,7)]'-H*mat(n,:)'))';
P0 =  (eye(10)-K*H)*Ppre;

if n==5 %%Change this if you want to track more ellipses.(Also
    check implicit.m)
 FIG=   figure(1)
    subplot(2,2,[2 4]);
implicit(mat)
 title([ 'Prediction in Implicit Form' ])
hold on
end
end

 kfinit=2;
pause (.001)
```

```
 clf(figure(1))
k=0;
  end

    end
end
```