Decentralized tracking with feedback, adaptive sample rate and IMM

Erik Karlsson

Reg no: LiTH-ISY-EX-3049

2000-03-13
Decentralized tracking with feedback, adaptive sample rate and IMM

Master Thesis at the Division of Automatic Control at Linköping University (LiTH), Sweden, by:

Erik Karlsson

Reg no: LiTH-ISY-EX-3049

Supervisor FOA: Fredrik Lantz
Supervisor LiTH: Rickard Karlsson
Examiner: Niclas Bergman
Linköping, March 13 2000
Decentralized tracking with feedback, adaptive sample rate and IMM

(Decentraliserad målföljning med återkoppling, adaptiv sampling och IMM)

Erik Karlsson

This report compares filter architectures for target tracking with multiple sensors. Two decentralized architectures are implemented and evaluated in simulations against a central tracking filter. Data from an array radar and an infrared sensor are used. The radar has adaptive sample rate, i.e. the radar is only used when necessary.

The central filter is an interactive multiple models filter (IMM) with three models. These models are extended Kalman filters. The decentralized architectures have two sensor filters and one central filter that fuses the tracks from the sensor filters. Feedback to the sensor filters is used. The sensor filters are also IMM-filters. The difference between the two decentralized architectures is the amount of information that is communicated between the filters in the architecture.

The filter architectures are evaluated with respect to their accuracy and ability to perform adaptive sampling. The conclusion is that the decentralized filter that communicates the most is more accurate and needs fewer radar measurements than the filter that communicates less. The central filter is the best tracker as expected and the best decentralized filter is almost equally good.

decentralized tracking, multiple sensors, feedback, interactive multiple models, IMM, adaptive sample rate, track fusion, covariance intersection, extended Kalman filter, EKF
Abstract

This report compares filter architectures for target tracking with multiple sensors. Two decentralized architectures are implemented and evaluated in simulations against a central tracking filter. Data from an array radar and an infrared sensor are used. The radar has adaptive sample rate, i.e. the radar is only used when necessary.

The central filter is an interactive multiple models filter (IMM) with three models. These models are extended Kalman filters. The decentralized architectures have two sensor filters and one central filter that fuses the tracks from the sensor filters. Feedback to the sensor filters is used. The sensor filters are also IMM-filters. The difference between the two decentralized architectures is the amount of information that is communicated between the filters in the architecture.

The filter architectures are evaluated with respect to their accuracy and ability to perform adaptive sampling. The conclusion is that the decentralized filter that communicates the most is more accurate and needs fewer radar measurements than the filter that communicates less. The central filter is the best tracker as expected and the best decentralized filter is almost equally good.
Acknowledgements

This is a Master Thesis in Applied Physics and Electrical Engineering at Linköping University, Sweden. The study was performed at the Swedish Defence Research Establishment and the Division of Command and Control Warfare Technology (Försvarets Forskningsanstalt, FOA).

I would like to express my gratitude towards my supervisor at the Defence Research Establishment Fredrik Lantz for the giving discussions and the support, to Dan Strömberg for his ideas and enthusiasm and Pontus Hörling for the flight path file and for patiently letting me run simulations on his computer.

At Linköping University I would like to acknowledge my supervisor Rickard Karlsson and examiner Niclas Bergman for their ideas and help in the making of this report. I am also grateful to Fredrik Gustafsson for the copy of his upcoming book Adaptive Filtering and Change Detection [6] and other materials.

I would also like to thank Marcus Grahn and my opponent Robert Bärnskog for their reviews of the report.

Linköping, March 2000

Erik Karlsson
Introduction

The purpose of this report is to compare different tracking architectures. This chapter gives an introduction to target tracking and important concepts of this report, as well as the outline of the report.

1.1 Target tracking

A modern military aircraft has a wide variety of sensors in order to guide the pilot. In an ideal world where sensors are perfect, target tracking would be reduced to simple geometry. However, sensors are not perfect, their measures are cluttered with noise and a single sensor can rarely measure all interesting information about a target. Therefore filters and multiple sensors are used to enhance the situation awareness.

Target tracking is the process of filtering noisy measurements from one or more sensors to achieve the best possible estimate of the state of the target. The result is presented to the pilot as one estimate with high accuracy instead of several noisy sensor displays. A track is a state estimate for one target over time.

This report deals with filtering data from a radar and an infrared sensor in order to estimate the position and velocity of a target. The purpose of the report is to investigate different filter and data fusion architectures. The target is usually another aircraft.

1.2 Principles of target tracking

The basic principle of tracking is to combine measured data with a mathematical model of how the target is likely to move. A model is a set of differential equations that describes, for instance, the relations between position, velocity and acceleration. The model is used to calculate, to predict, where the target will be in the future. The prediction is then combined with sensor measurements to produce an estimate of the target state. This is done in a Kalman filter, which will be discussed in detail in chapter 2. The reason for using a model is to reduce the influence of the measurement noise. It also makes it possible to predict where the target will be in between measurements and thereby creating more of a smooth track rather than just estimates at discrete time points.

The main tracking sensor is the radar sensor. A radar is mainly an active sensor, which means that it emits signals and measures the reflected energy that comes back from the
measured object. A radar can measure bearing, elevation and distance to an object. A
doppler radar can also measure the derivative of the distance, i.e. the radial velocity or
range rate. A radar can typically measure distance with good resolution, but the angular
resolution is not very good. Since a radar measures both angle and distance to an object it
provides sufficient information to track an object. A radar can also be used passively, for
instance as a radar detector.

An infrared search and track sensor (IRST) on the other hand has a good angular resolu-
tion. An IRST is a passive sensor; it only registers incoming signals. The sensor measures
bearing and elevation of a target. It can therefore only provide the direction to a target and
not the exact position, but because of the good angular resolution this sensor can be a
valuable complement to a radar tracking system.

The precision of a track estimate can be greatly enhanced if both radar and IRST mea-
surements are available. However, if a reduction in precision is acceptable, the number of
radar measurements can be reduced if the IRST measures in between radar measure-
ments and thereby reduces the uncertainty of the predicted state.

A complete tracking system must be able to deal with more than one target. This fact
adds to the complexity of the system, since functions dealing with associating measure-
ments to the right tracks, track initiation and deletion has to be implemented.

The basic principle for data association is that if a measurement is close to a predicted
target then this measurement probably belongs to that target. There are several data asso-
ciation algorithms. The nearest neighbor algorithm simply associates a measurement
with the closest predicted target. The joint probabilistic data association algorithm forms
probabilistic hypotheses over every possible association combination. The multiple
hypotheses tracking algorithm forms probabilistic hypotheses over time and handles par-
allel hypotheses. A decentralized architecture would also need to perform track associa-
tions.

This report does not handle multiple targets or data association.

1.3 Electronically scanned antennas and adaptive sample rate
Conventional radars are mechanically scanned antennas (MSAs). The antenna is
mechanically pointed in the desired direction. Mechanical systems are always limited by
inertia. This inhibits the flexibility of a MSA since it is too slow to point to many differ-
tent targets while monitoring a search volume for new targets in a short period of time.
Therefore, a MSA is usually constrained to certain fixed scan patterns and to track targets
while scanning the search volume.

A new generation of radar is the electronically scanned antenna (ESA). An ESA does not
contain any moving parts, but consists of an array of emitters and it controls the radar
pulse by phase shifting the array elements. The ESA is not restrained by inertia and can
therefore be repositioned within microseconds. The idea behind ESA is old, but it was
not until the development of modern electronics that the idea was possible to implement.

The agility of an ESA makes a more flexible radar use possible. It is now attainable to
point the antenna to many different locations and still be able to scan the search volume
within a short period of time. In order to utilize the capacity of an ESA effectively other
tracking strategies have been developed. These strategies usually invoke adaptive sample rate as a mean to minimize the needed radar time to update a given track.

Adaptive sample rate ideally means that the radar should only be used when necessary, i.e. when the quality of the track is too low, rather than on a regular basis. Since the actual quality of a track is unknown it is being estimated as the uncertainty of the track states, or the covariance matrix of the corresponding state vector, see chapter 2.

Adaptive sample rate is a central issue in tracking system design and in this report. A reduction of radar measurements has at least two desired features. Since a radar is an active sensor its signals can be detected by other aircrafts. By reducing the number of radar measurements the risk of being detected is reduced. The other advantage is that the radar will have more time to perform other tasks such as: keep track of more targets, radar jamming, communication and intercepting other radars.


1.4 Tracking architectures
The conventional architecture for target tracking is a centralized architecture where all sensor data are put into one filter that produces a target state estimate. See Figure 1.1.

The central tracker is an optimal tracker, since the filter has access to all sensor data. Its estimate is therefore based on all available data. An attractive quality of this architecture is its simplicity. However, it is generally assumed that the filter calculations are performed in the central computer of the aircraft, since it is this system that requires the state estimate. The filter calculations can be quite extensive and for this reason a distributed architecture can be of interest.

![Figure 1.1 Central tracker](image)

There are many kinds of distributed architectures, but perhaps the most straightforward in a tracking application is a hierarchical decentralized architecture. See Figure 1.2.

![Figure 1.2 Decentralized tracker](image)
1 INTRODUCTION

In this architecture each sensor has its own tracking filter. These local filters produce tracks which are then fused into a central track and the central computer has only to perform the fusion. There are also other reasons for this type of architecture. For instance, if an aircraft already has a radar tracking system, it could easily be complemented with an IRST-tracker. The IRST-track can be fused with the existing system according to Figure 1.2 and only small alterations are needed in the original system. Other sensors, such as a radar warning sensor, or estimates from other aircrafts or ground control systems can readily be integrated into the system. This architecture provides graceful degradation, i.e. if one sensor malfunctions, there will still be a sensor and tracker that is operational. A central architecture is totally reliant upon the central node.

Even though the central computer only has to perform track fusion for tracking purposes, it is likely to contain a prediction model as well, similar to those in the local filters. One reason for this is that the pilot perhaps wants more frequent track updates. Another reason is that the system wants to predict future states of the target for weapon delivery or scheduling purposes. In this report, however, the central computer only performs track fusion for the decentralized architectures.

It is shown in [3] and [4] that feedback significantly enhances tracking quality for decentralized architectures and therefore only architectures with feedback are considered in this report. [3] and [4] concludes that a decentralized tracking system with feedback is well comparable to a central tracking system.

Modern sensors, as the ESA and IRST, usually contain a tracker, and some sensor systems provides only tracks and not observations [13]. They also have built-in logic and can make their own decisions regarding track quality and the need for measurements. Because of this, much is won if the sensors can have access to more information and thus can make better decisions. A decentralized system with feedback provides this information to the sensors.

However, feedback introduces cross-correlation between the sensor tracks. This cross-correlation must be dealt with in order to produce consistent tracks. It is especially important when tracking multiple targets where data association is needed. For instance, if one sensor filter makes an association with a low level of confidence and this association is fed back to the same filter it must not take this information as a confirmation of its own decision. It should not upgrade its level of confidence when actually no new information has been gained.

The improved tracking quality in the sensors will lead to better measurement-to-track association among other things [3].

Feedback can also be dangerous. If one sensor malfunctions or is being jammed, it will produce estimates that are incorrect. These erroneous estimates will then spread to the other sensor filters and the central estimate could very well be ruined. However, in this report all sensors produce unbiased measurements. A more realistic situation would require a sensor manager that monitors the local filters and sensors to prevent feedback in the case of malfunctioning local filters. This would naturally be required in a realistic centralized system as well, but most likely the detection of a malfunction in a feedback system will be more difficult to observe and locate.
1.5 The aim of the report

The aim of this report is to construct and to evaluate different decentralized tracking architectures and to compare them to each other and to a central tracking system. The decentralized systems should have feedback and their local filters should consist of multiple kinematic models (see chapter 2). The radar sensor should be an ESA radar and use adaptive update rate. For data fusion, a method called Covariance Intersection should be investigated and applied.

The architectures will be evaluated with respect to a single target with an idealized sensor model (see chapter 2.4.1) in a two dimensional scenario. See chapter 4.5 for further limitations.

The evaluation will be based on Matlab™ simulations. The simulations are further discussed in chapter 4 and the outcome is presented in chapter 5.

1.6 Outline

Chapter 2 considers the theory behind tracking. It discusses the choice of coordinates, the Kalman filter, the linearization and discretization that leads to the extended Kalman filter (EKF). It deals with the interactive multiple models (IMM) algorithm and also adaptive sample rate and sensor models.

Chapter 3 deals with decentralized architectures and their distinctive philosophies. Feedback is discussed, as is data fusion in general and Covariance Intersection in particular. The different decentralized architectures are presented and discussed.

Chapter 4 defines the simulations. What features and qualities are important to study and how can these qualities be measured? This chapter presents filter and sensor data, the scenario and limitations used in the simulations.

Chapter 5 presents the outcome of the simulations.

Chapter 6 concludes the results.

Appendix A is a list of symbols and abbreviations used in this report.

Appendix B, C and D contains Matlab™ code for the implemented filters.
2 Estimation

This report deals with tracking of aircrafts. To track is to estimate the state of a target, i.e. the position, velocity, direction, et cetera. In order to estimate the state of the target a target motion model and sensors are used. Sensor data and motion model are fused in a Kalman filter into a state estimate. The sensors are here modelled to be standing still. The model can readily be modified to let the sensors be mounted on an aircraft, see [4] for instance.

2.1 Kalman filter

The Kalman filter estimates the state by a mean, \( x(t) \) (the state vector), and covariance, \( P \) (the uncertainty matrix), which is a complete description if the state is normal distributed. In this report the state and covariance together are called an estimate or a state estimate.

2.1.1 Linear Kalman filter

A general motion model used in a Kalman filter for tracking is a continuous state space model:

\[
\begin{align*}
\dot{x}(t) &= f(\hat{x}(t)) + Bv(t) \\
y(t) &= h(\hat{x}(t)) + w(t)
\end{align*}
\]

In equation (2.1) \( \hat{x}(t) \) is the estimated state vector, \( v(t) \) is the system noise vector and \( B \) is a matrix. The system noise is zero-mean white noise with covariance \( Q \). The equation can be seen as a description of how the system’s current state will affect its derivatives, i.e. how the current state affects the change of the state. The noise vector is needed to model the uncertainty of the model. Equation (2.2) is the measurement vector. It describes the measured data in the state vector variables. \( w(t) \) is the measurement noise vector which is also zero-mean white noise with a covariance matrix \( R \).

Discretization

In many cases equations (2.1) and (2.2) are linear and time invariant and can be written as:

\[
\begin{align*}
\dot{x}(t) &= A\hat{x}(t) + Bv(t) \\
y(t) &= C\hat{x}(t) + w(t)
\end{align*}
\]
2 ESTIMATION

where $A$, $B$ and $C$ are matrices. To enable a computer to filter, equations (2.3) and (2.4) need to be discretized into:

$$
\dot{x}_{k+1} = F\tilde{x}_k + Gw_k 
$$ (2.5)

$$
y_k = H\tilde{x}_k + w_k
$$ (2.6)

where $\tilde{x}_k = \tilde{x}(t) = \tilde{x}(kT)$, $\tilde{x}_{k+1} = \tilde{x}(t+T)$ and $T$ is the period time. According to [8] this is accomplished if:

$$
F = e^{AT} = I + AT + \frac{A^2T^2}{2!} + \ldots + \frac{A^mT^m}{m!} + \ldots
$$ (2.7)

$$
G = \int_0^T e^{ABs} \cdot ds
$$ (2.8)

$$
H = C
$$

and $v(t)$ is constant during the sample period $T$. $w_k$ is naturally discrete since the sensors deliver discrete measurements. $I$ is the identity matrix.

The Kalman filter

The filter is derived by minimizing the expected estimation error:

$$
\min E\{(x_k - \hat{x}_k)^T(x_k - \hat{x}_k)\}
$$ (2.9)

$x_k$ is the unknown true state. The explicit derivation of the equations will not be performed in this report, but can be found in for example [6, 9, 10, 11 or 16].

The Kalman filter works in two steps at time $k$. The first step, called time update, is to predict the state from time $k-1$ to $k$ using the model. Then in the second step, called the measurement update, the measurements are included to update the prediction to an estimate for time $k$.

The following equations are borrowed from [9 and 10].

**Time update**

$$
\dot{x}_{k|k-1} = F\tilde{x}_{k-1}
$$ (2.10)

$$
P_{k|k-1} = FP_{k-1}F^T + GQ_{k-1}G^T
$$ (2.11)

Where $P_k$ is the covariance of $\tilde{x}_k$ and

$$
Q_k = E[v(t)v(t)^T] =
\begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & v_t^2 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & v_n^2
\end{bmatrix}
$$
assuming that the noise vector \( \nu(t) \) is constant between samples and chosen as in equation (2.18).

**Measurement update**
The innovation, \( \varepsilon \), is the difference between the measurement and the prediction expressed in the measurement variables:

\[
\varepsilon_k = y_k - H\hat{x}_{k|k-1}
\] (2.12)

The covariance of the innovation is:

\[
S_k = R + HP_{x_k|k-1}H^T
\] (2.13)

The Kalman gain, \( K_k \), is calculated to decide how much the innovation is to affect the state estimate.

\[
K_k = P_{x_k|k-1}H^T S_k^{-1}
\] (2.14)

Now, the state estimates can be calculated:

\[
\hat{x}_k = \hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \varepsilon_k
\] (2.15)

\[
P_k = P_{x_k|k} = P_{x_k|k-1} - P_{x_k|k-1}H^T S_k^{-1} HP_{x_k|k-1}
\] (2.16)

According to [11] equation (2.16) can be rewritten to perform better numerically:

\[
P_k = P_{x_k|k} = (I - K_k H)P_{x_k|k-1}(I - K_k H)^T + K_k R K_k^T
\] (2.17)

The Kalman filter algorithm has to evaluate equations (2.7), (2.8), (2.10)-(2.15) and (2.17) at every time interval.

**2.1.2 Choice of coordinates**
The previous chapter discussed the linear Kalman filter. However, the objects that we want to track move in a nonlinear manner. We must step back to equations (2.1) and (2.2) again and consider the nonlinearities, and how to capture these motions in a coordinate system.

The state vector can be chosen in many ways, but generally position and velocity variables are included. These variables are often expressed in a cartesian coordinate system. According to [5] it is better to use cartesian position and polar velocity coordinates to track coordinated turns. They perform better than cartesian position and velocity coordinates. Coordinated turns is the name for a motion model were the target moves in straight lines and circle segments. Coordinated turns describe most civilian aircraft motions accurately [5]. The main difference between civilian and military aircraft is that a military aircraft is more manoeuvrable. The difference is especially big tangential to the flight path and in altitude.

Since the coordinates in this report are not of main concern, a two dimensional model is used. An extension into three dimensions is easily accomplished but will only add to the complexity of this study.

The state vector will then be the following, with notations from Figure 2.1:

\[
\begin{align*}
\varepsilon &:= y_k - H\hat{x}_{k|k-1} \\
S &= R + HP_{x_k|k-1}H^T \\
K_k &= P_{x_k|k-1}H^T S_k^{-1} \\
\hat{x}_k &= \hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \varepsilon_k \\
P_k &= P_{x_k|k} = P_{x_k|k-1} - P_{x_k|k-1}H^T S_k^{-1} HP_{x_k|k-1} \\
P_k &= P_{x_k|k} = (I - K_k H)P_{x_k|k-1}(I - K_k H)^T + K_k R K_k^T
\end{align*}
\]
2 ESTIMATION

\[
\begin{bmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  x_4 \\
  x_5
\end{bmatrix} =
\begin{bmatrix}
  x \\
  y \\
  v \\
  \phi \\
  \omega
\end{bmatrix}
\]

Figure 2.1

with \( \omega = \dot{\phi} \). Equation (2.1) will then be:

\[
\dot{x}(t) = \begin{bmatrix}
  v \cos \phi \\
  v \sin \phi \\
  0 \\
  0 \\
  v_r
\end{bmatrix} +
\begin{bmatrix}
  0 \\
  0 \\
  v_r \\
  0 \\
  v_{\omega}
\end{bmatrix}
\]  
(2.18)

\( v_r \) is the acceleration in the tangential direction of the flight path and can be seen as the change in thrust. \( v_{\omega} \) is the acceleration normal to the trajectory and can be seen as the turn rate of the target.

A radar measures \((r, \dot{r}, b_r, e_r)\), but the elevation, \( e_r \), is not used in this 2-D scenario. An IRST measures \((b_{ir}, e_{ir})\) and only \( b_{ir} \) will be used. These measurements have to be transformed into the chosen coordinates. The measurement equation (2.2) will be:

\[
y(t) = \begin{bmatrix}
  r \\
  \dot{r} \\
  b_r \\
  b_{ir}
\end{bmatrix} + w(t) = h(x(t)) + w(t) = \begin{bmatrix}
  \sqrt{x_1^2 + x_2^2} \\
  \frac{x_1 v \cos \phi + x_2 v \sin \phi}{\sqrt{x_1^2 + x_2^2}} \\
  \frac{\sqrt{x_1^2 + x_2^2}}{x_1} \tan(\frac{x_2}{x_1}) \\
  \frac{\sqrt{x_1^2 + x_2^2}}{x_2} \tan(\frac{x_2}{x_1})
\end{bmatrix} + \begin{bmatrix}
w_1 \\
w_2 \\
w_3 \\
w_4
\end{bmatrix}
\]  
(2.19)

The measurement noise vector, \( w(t) \), can usually be established by observing the sensors. The system noise vector, \( v(t) \), can be seen as a design parameter that should be adjusted until a satisfactory result is achieved.

2.1.3 Nonlinear Kalman filter

Equations (2.1) and (2.2) are both time continuous and nonlinear. They need to be discretized as earlier and to be linearized to fit into the Kalman framework. A Kalman filter with linearization is called an extended Kalman filter (EKF).

There are basically two different strategies for turning a continuous nonlinear model into a discrete linear one: first linearize and then discretize or first discretize and then linearize. Since the performance of these strategies is not of main concern for this report, the method of linearizing first was chosen because it was easier.

The linearization is accomplished by a Taylor expansion of equation (2.1), which then is discretized according to chapter 2.1.1, resulting in the prediction equations [9]:

\[
\begin{align*}
x_1(t+1) &= x_1(t) + v_{\omega}(t) \\
x_2(t+1) &= x_2(t) + v_r(t) \\
x_3(t+1) &= x_3(t) \\
x_4(t+1) &= x_4(t) \\
x_5(t+1) &= x_5(t)
\end{align*}
\]
Time update

\[ \hat{x}_{k|k-1} = \hat{x}_{k-1} + Gf(\hat{x}_{k-1}) \] (2.20)

\[ P_{4k|k-1} = FP_{k|k-1}F^T + GQ_{k|k-1}G^T \] (2.21)

where

\[ F = e^{f(\hat{x})T} \] (2.22)

\[ G = \int_{0}^{T} e^{f(\hat{x})T} dt \] (2.23)

\( f(\hat{x}) \) is the Jacobian of \( f(\hat{x}(t)) \). The Jacobian is a matrix where the rows in \( f(\hat{x}(t)) \) are derived by all the variables in the state vector. With the chosen state vector, \( f(\hat{x}) \) becomes:

\[ f(\hat{x}) = \frac{d}{dx}f(\hat{x}) = \begin{bmatrix} 0 & 0 & \cos \phi & -v \sin \phi & 0 \\ 0 & 0 & \sin \phi & v \cos \phi & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = A \] (2.24)

\( F = e^{AT} \) is calculated in the same manner as in equation (2.7) and it can be noted that only the first 3 terms in the right side are needed, since \( A^3 = 0 \).

Measurement update

To enable a measurement update \( h(\hat{x}(t)) \) also needs to be linearized. This is accomplished by replacing \( H\hat{x}_{k|k-1} \) in equation (2.12) with \( h(\hat{x}_{k|k-1}) \) and \( H \) with the Jacobian of \( H \) around \( \hat{x}_{k|k-1} \), called \( H^f \).

\[ H^f = \begin{bmatrix} \frac{x_1}{r} & \frac{x_2}{r} & 0 & 0 & 0 \\ -\frac{x_1}{r^3} + \frac{x_1}{r} & -\frac{x_2}{r^3} + \frac{x_2}{r} & \frac{x_1 \cos \phi + x_2 \sin \phi}{r} & \frac{-x_1 v \sin \phi + x_2 v \cos \phi}{r} & 0 \\ -\frac{x_2}{r^3} & \frac{x_1}{r^3} & 0 & 0 & 0 \\ -\frac{x_2}{r^3} & \frac{x_1}{r^3} & 0 & 0 & 0 \end{bmatrix} \] (2.25)

where \( \alpha = x_1 \cos \phi + x_2 \sin \phi \) and \( r = \sqrt{x_1^2 + x_2^2} \).

Equations (2.12)-(2.15) and (2.17) are replaced by:

\[ \varepsilon_k = y_k - h(\hat{x}_{k|k-1}) \] (2.26)

\[ S_k = R + H^f P_{k|k-1} H^T \] (2.27)
2 ESTIMATION

\[ K_k = P_{k|k-1} H^T S_k^{-1} \]  
\[ \hat{x}_k = \hat{x}_{k|k-1} + K_k e_k \]  
\[ P_k = P_{k|k} = (I - K_k H') P_{k|k-1} (I - K_k H')^T + K_k R K_k^T \]

All equations in chapter 2.1.3 have to be evaluated at every time interval.

2.2 Multiple models

In order to get a good estimate from the Kalman filter, the process noise Q needs to be well adjusted. The choice of Q is however a delicate problem. On one hand, we would like the filter to be able to track very sharp turns, which it will if we allow Q to be large. On the other hand, we do not want the process noise to be bigger than necessary, since we would then rely more on the measurements and not filter that much, leaving a noisy estimate.

2.2.1 General ideas on multiple models

An idea on how to solve this dilemma is to adapt the system model to how the target maneuvers. The basic idea is to have a bank of Kalman filters with either different state vectors or different parameter settings. These filters are then combined to produce a better estimate. There are at least two alternative ways to model target maneuvers, the Markov model and the Jump model [12]. A Markov model models the transitions between different target maneuvers according to a Markov chain. That means that at any given time there are certain probabilities that the target changes its motion into a given number of predefined motion patterns. One example of a Markov model is the interacting multiple models (IMM) algorithm (IMM) where the filters interact each time period according to their probabilities of being true. This algorithm is probably the most common method of all multiple model filters.

A jump model on the other hand models the target as either proceeding its current motion or interrupting this motion, i.e. changing its turn rate. Each of these options lead to a Kalman filter, which results in a growing tree of filters. To keep the number of filters constant the least likely branches are pruned. An example of a jump model is the adaptive forgetting multiple models algorithm, see [9, 10 or 12] for more details.

According to [12] a Markov model (e.g. IMM) yields better estimation accuracy at the onset of a maneuver, while the jump model gives better steady state accuracy before, during and after a turn. Since the IMM algorithm is the dominant model and the choice of multiple model algorithm is not of main concern, the IMM method is chosen for this report.

2.2.2 Interacting Multiple Models (IMM)

When designing an IMM-filter there is a choice of how many models to use and how these models should be designed. A transition matrix that describes the likelihood of model interaction needs to be defined. In the vast literature concerning this method a number of model sets can be found. A common number of models seems to be three. Some designers use one filter for a straight trajectory, another for left hand turns with a specific acceleration and the third for a right hand turn with a specific acceleration. This
will result in element 3 and 5 in the state vector in equation (2.18) having a specific value and that the noise vector will be set to zero.

Another method is to adjust the process noise Q. One model can be used for straight trajectories, with very little system noise. Another can have more noise, i.e. larger Q matrix, which corresponds to normal acceleration. Then the third model can be used for large accelerations. By adjusting Q the relation between how much we trust the model and how much we trust the measurements is altered. A large Q means that the model error is large and the Kalman filter will use the measurements at a larger extent, and a small Q leads to the opposite. The implemented filters of this report uses this method.

The Algorithm
In the IMM-algorithm all models are associated with a probability of being the correct model, \( \mu_i(k) \). At the beginning of the algorithm cycle, at time \( k \), a mixing matrix, \( \mu_{ij}(k-1) \), is being calculated from \( \mu_i(k-1) \) and the predefined transition likelihoods \( p_{ij} \).

The number of models used in this report is \( n = 3 \).

\[
\mu_{ij}(k-1) = \frac{p_{ij}\mu_j(k-1)}{\sum_{i=1}^{n} p_{ij}\mu_i(k-1)} = \frac{p_{ij}\mu_j(k-1)}{C_j(k-1)} \quad i,j = 1, ..., n
\]  

(2.31)

This mixing matrix is then used to mix the states \( \hat{x}_j(k-1) \) into \( \hat{x}_{0j}(k-1) \) and \( P_j(k-1) \) into \( P_{0j}(k-1) \) as shown in Figure 2.2. Indices i and j indicate model.

\[
\hat{x}_{0j}(k-1|k-1) = \sum_{i=1}^{n} \hat{x}_i(k-1|k-1)\mu_{ij}(k-1)
\]  

(2.32)

\[
P_{0j}(k-1|k-1) = \sum_{j=1}^{n} [P_j(k-1|k-1) +

[\hat{x}_i(k-1|k-1) - \hat{x}_{0j}(k-1|k-1)][\hat{x}_i(k-1|k-1) - \hat{x}_{0j}(k-1|k-1)]^T] \mu_{ij}(k-1)
\]  

(2.33)

In this way the different filters affect each other with respect to their probabilities of being correct.

After the state mixing, each model is filtered according to chapter 2.1.3. The filter algorithm has to be complemented with a calculation of the model probability. The calculation of model probability is based on the innovation, which is assumed to be normal distributed. This gives the model probability, \( g_j \)
2 ESTIMATION

\[ g_j(k) = \frac{1}{\sqrt{(2\pi)^{dim(e_j)} |S_j|}} e^{-\frac{(e_j)^T e_j}{2}} \] (2.34)

The IMM-algorithm then updates and normalizes the model probabilities:

\[ \mu_j(k) = \frac{g_j(k)C_j(k - 1)}{\sum_{i=1}^{n} g_i(k)C_i(k - 1)} \quad i,j = 1, ..., n \] (2.35)

\( C \) is defined in equation (2.31). Now the models are filtered and each model has an updated probability of being the correct model. To produce one estimate the three models are interpolated into one. The interpolated state is then:

\[ \hat{x}_{k|k} = \sum_{j=1}^{n} \hat{x}_j(k|k)\mu_j(k) \] (2.36)

\[ P_{k|k} = \sum_{j=1}^{n} (P_j(k|k) + (\hat{x}_j(k|k) - \hat{x}_{k|k})(\hat{x}_j(k|k) - \hat{x}_{k|k})^T)\mu_j(k) \] (2.37)

The interpolated state is the output of the filter. The interpolation does not affect future values of the filter, since it is not used again in the cycle. For more details on the IMM-method and a derivation of the equations, see [10 or 11].

The algorithm can be summarized:

- Predict model probabilities
- Mix the models with one another according to these probabilities
- Filter the models separately and calculate their probabilities of being correct
- Update the model probabilities
- Interpolate the models into one estimate

2.3 Adaptive radar sample rate

As mentioned in chapter 1.3 the ESA radar is much faster than a conventional radar. In order to utilize this agility effectively, it is desirable to use adaptive sample rate. This means that the radar should measure a target only when it is necessary rather than on a regular basis; the radar should adapt its sample rate to the situation.

The reason for a desired lower sample rate is twofold. The radar will have more time to perform other tasks, e.g. keep track of more targets. The radar will also emit less signal energy and will therefore be less visible to enemy radars.

Adaptivity in this report only concerns the radar sensor. The IRST is thought to measure periodically. The IRST-sensor is mechanically steered. This sensor is faster than the MSA-radar and for the purpose of this report the IRST is considered able to measure whenever we wish.
2.3.1 Measure of uncertainty

To be able to measure only when necessary, it is required to have some sort of measure of the necessity of a radar update. There are different ideas of how to measure the update necessity. One idea is to monitor the innovation, in this case to monitor how well the prediction and the measurements of the IRST-sensor correspond. Other ideas utilize the fact that every state estimate has a corresponding covariance matrix in the Kalman filter to estimate the uncertainty of the state estimate. It is therefore natural to use this information as a measure of the urgency of a radar update. This strategy has the advantage that the measure of the need for a radar update is independent of the IRST-sensor. This corresponds well to the idea that the ESA-radar makes its own decisions of when to measure.

The idea of using the covariance is used in this report. To use the whole covariance matrix as a measurement can be inconvenient and often only the position elements in the covariance matrix are used. The standard deviation of the position is calculated from the position elements in the covariance matrix. This gives a measurement in meters which is intuitively good. Simulations suggest that this measure of uncertainty works well.

Equation (2.38) calculates the standard deviation of the position estimate $r = \sqrt{x_1^2 + x_2^2}$ [10]:

$$\text{std}(r) = \sqrt{H_r(x)P_H(x)^T}$$  \hspace{1cm} (2.38)

where:

$$H_r(x) = r' = \frac{1}{\sqrt{x_1^2 + x_2^2}} \begin{bmatrix} x_1 & x_2 & 0 & 0 & 0 \\ \end{bmatrix}$$  \hspace{1cm} (2.39)

The most straightforward method of implementing adaptive sample rate is to calculate equation (2.38) and use the radar if the standard deviation of the relative distance is bigger than some value. For the decentralized filters, equation (2.38) is performed in the radar filter and if a radar measurement is not performed the radar filter uses its prediction as an estimate.

This strategy waits until the uncertainty of the position is larger than some value, but sometimes it is unacceptable that the uncertainty exceeds the limit. It is therefore common to predict the uncertainty ahead. In the simplest form, the radar filter predicts the uncertainty at the beginning of the filter loop to see how big the uncertainty will be if a measurement is not performed. This predicted uncertainty is then evaluated. In a more complex situation with multiple targets, the uncertainty can be iteratively predicted ahead until it reaches a certain limit. This will result in a maximum update time that can be used for scheduling purposes.

In an IMM-filter the prediction can be made for one filter, e.g. the most probable, if a simple method is wanted. [17] suggests to perform an IMM-cycle with only predictions, i.e. to let the models interact, then predict all models and interpolate a predicted state and covariance. This interpolated prediction is then used as a measurement of the uncertainty.

2.3.2 Update strategy

The easiest update strategy is to evaluate the measurement calculated in chapter 2.3.1 against a fixed limit. A strategy could be: if the standard deviation of the position estimate
2 ESTIMATION

is bigger than, for instance 100 meters, use the radar. This strategy will result in more measurements if the target is far away and less measurements if the target is close.

Another strategy could be to evaluate eq. (2.38) against a relative limit. Perhaps it is not critical to know the position of a target within 100 meters if the target is 50 km away. However, if the target is 1 km away it is probably extremely important to know its position with very high accuracy. A relative limit also represents the actual beam of the radar. The radar beam spreads out in a cone-shaped manner. If the target is far away the beam width is very large and the target will more likely be in this section and reflect the radar pulse, than if the target is closer. If the beam width is taken into account in the position uncertainty limit, the likelihood of a hit is equally big for all distances.

In a realistic situation there is an upper limit of permitted uncertainty. This limit is set by the radar beam width and range gate. The longest sample period is calculated as the time at which the target might possibly exit the predicted region that the radar beam covers. If the uncertainty is bigger than the beam width or range gate the radar might fail to detect the target. If this is the case the radar must perform a number of measurements around the previous measurement to find the target again. This will occupy the radar for some time and use unnecessary radar resources.

This report uses fixed limit update strategies. Sometimes it is desired to force a measurement even if the uncertainty is low. This can, for instance, be done by the pilot or if for some reason the sensors and/or tracks are believed to be inaccurate. In the simulations in this report the radar is forced to make a measurement in the beginning, even though it starts with correct values. In a multiple target situation, the closeness of other targets will probably also affect a relative update limit.

2.4 Sensor models

The sensors used in the simulations are considered ideal. Real sensor are of course not ideal, but for the purpose of comparing the proposed tracking architectures the sensors can most likely be considered ideal. Since the sensors can measure many times faster than the system needs to work, the system can most likely always expect successful measurements when it needs them. However, a realistic radar model is investigated here to see how the radar can be affected and controlled for tracking purposes. A brief discussion about an IRST-model is also included. Chapters 2.4.2 and 2.4.3 are somewhat separate from the rest of the report and are basically just summaries of chapter 2.2 and 2.3 in [11] and can be seen as a sensor model introduction. The equations in these two chapters are from [11].

2.4.1 Ideal sensor model

A sensor is considered ideal in the sense that it always detects the target and never detects any false targets. The measurements are noisy, but they always correspond to one target. In the simulation an ideal radar model with fixed update limits of different values is used. The IRST is also ideal but measures regularly. The measurement noise is chosen similar to values found in various references and are presented in chapter 4.3.

2.4.2 Radar model

This chapter gives a more detailed discussion of the radar sensor. It considers the physical reality behind the measurement noise. It also considers the fact that a radar sometimes fail
to produce a measurement and that sometimes clutter in the measurement causes false echoes.

A radar measures range by filtering the reflected pulse through a set of discrete matched filters. These filters are delayed versions of the transmitted pulse. The delay of the matching filter corresponds to the range by \( R = ct/2 \) where \( c \) is the speed of light and \( t \) is the delay. Each filter correspond to a certain range, or range interval, called a range bin. Bearing and elevation are measured by noting the angles in which the radar aperture is directed at the time of detection. A doppler radar measures range rate by observing the difference in frequency between the transmitted and the received signal which is due to the relative target motion. The range rate is \( \dot{R} = -(\lambda \Delta f)/2 \) where \( \lambda \) is the wavelength of the transmitted energy and \( \Delta f \) is the difference in frequency. Closing motion is defined as positive doppler shift.

**Top-level view**

Two important factors that affect the radar’s ability to detect targets are thermal noise and clutter. The thermal noise is mainly caused by the radar itself and is considered in this top-level view of the radar. Clutter is mentioned in the low-level discussion below.

The relation between the received target signal power and thermal noise power is given by the *radar range equation* (2.40):

\[
\text{SNR} = \frac{P_{pk}G_T\rho\sigma}{(4\pi)^2R^2kT_0B_nFL_p} \quad (2.40)
\]

\( \text{SNR} \) stands for signal to noise ratio and is often converted to decibels: \( SNR_{dB} = 10\log(\text{SNR}) \). The signal power in equation (2.40) is:

\[
S = \frac{P_{pk}G_T\rho^2\sigma}{(4\pi)^2R^4L_p} \quad (2.41)
\]

\( P_{pk} \) is the transmitted power, \( G_T \) is the gain of the transmitting antenna, indicating to what extent the transmitted energy is concentrated in the transmission direction. \( A_e \) is the effective receiving antenna aperture, which indicates the amount of reflected energy that is detected. \( \rho \) is the pulse compression ratio, which here is assumed to be 1. \( \sigma \) is the target’s radar cross-section and is a measure of how much of the transmitted energy is reflected back in the direction of the receiver by the target. \((4\pi)^2R^4\) is the signal power loss due to the spreading of the beam over the distance \( R \) and \( L_p \) is any additional propagation loss.

The noise power in eq. (2.40) is:

\[
N = \rho kT_0B_nF \quad (2.42)
\]

\( k \) is Boltzmann’s constant. \( T_0 \) is the assumed temperature of 290 K. \( B_n \) is the receiver noise bandwidth; \( B_n = 1/\tau_p \) where \( \tau_p \) is the pulse width. \( F \) is the receiver noise figure, which accounts for any additional noise the receiver adds if the temperature is over the assumed 290 K.
2 ESTIMATION

The radar range equation (2.40) is fundamental to tracking analysis. Given the radar and target parameters, $SNR$ can be calculated as a function of the distance to the target. The $SNR$ is used together with a $P_D$-curve to determine the probability of detection, $P_D$.

A $P_D$-curve depends on the chosen probabilistic model for target motion, the noise characterization, receiver detection law and various waveform parameters. The probabilistic motion model is often chosen as one of the Swirling models, see [11]. A receiver detection law is the manner in which the radar detection mechanism works. One waveform parameter is the amount of noncoherent integration. The signal voltage of the received energy is often integrated over multiple pulses to increase the $SNR$. To actually increase the $SNR$, the signals need to be coherent during the period of integration $\tau$. It is assumed that the signal voltage from the target is phase coherent during the period of integration and its power increases as the square of $\tau$. The noise voltage is random and its power increases proportionally to $\tau$.

A typical $P_D$-curve looks like in Figure 2.3. This graph is a plot of the simple relationship $P_D = P_{FA}^{1/(1+SNR)}$ which is valid for a single coherent measurement assuming a Swerling 1 or 2 motion model. These models are valid for a wide range of airborne targets. $P_{FA}$ is the probability of false alarm, i.e. the likelihood of detecting a nonexisting target. The left curve is plotted using $P_{FA} = 10^{-2}$ and the right curve uses $P_{FA} = 10^{-6}$. $P_D$ and $P_{FA}$ depends on the detection threshold used. If the threshold is low then the likelihood of a detection is high but the likelihood of false detections is also high. The ESA-radar often uses a strategy called alert-confirm. First it makes a measurement with a low threshold. Then it rapidly measures all potential targets again with a higher threshold. If the second measurement detects the same potential target it is likely a true target. If the measurement fails the potential target was probably noise or clutter.

The probability of detection will affect the update strategy discussed earlier. The probability of missed detections must be taken into account when calculating the uncertainty measure. We can also note that the likelihood of a detection quickly rises towards 1 when

![Figure 2.3 A typical Pd-curve.](image)
a series of measurements are performed, even if the probability of detection for a single measurement is fairly low. \( P_D(l) = 1 - (1 - P_D)^l \) for \( l \) attempts and, for instance, \( P_D(3) = 0.992 \) and \( P_D(5) = 0.9997 \) if \( P_D = 0.8 \).

A radar often uses a series of pulses instead of a single pulse. This is done mainly to achieve better SNR, higher probability of detection and to resolve ambiguities in the doppler range rate measurement. The doppler shift is measured by calculating the difference between the transmitted and the incoming pulses. The top graph in Figure 2.4 illustrates transmitted and received pulses. The received pulses have smaller amplitude and a slightly different frequency than the frequency of the transmitted pulses. The pulse difference is then integrated for each pulse as illustrated in the lower graph. The pulses in the lower graph can be seen as samples of a signal with a frequency equal to the frequency shift. These samples are then converted into frequency domain using the fast Fourier transform algorithm (FFT). The FFT-spectrum of the signal is then filtered through matched filters to determine the doppler frequency shift. The time between the pulses is \( T \). \( 1/T \) is called the pulse repetition frequency. Often a number of measurements with different pulse repetitions frequencies are used.

\[
\begin{align*}
\text{Figure 2.4} & \quad \text{The top graph illustrates the transmitted and received} \\
& \quad \text{pulses. The lower graph illustrates the integrated difference} \\
& \quad \text{between the two signals for each pulse.}
\end{align*}
\]

**Measurement quality**

The angular resolution of the radar is mainly determined by the beam width. To be able to resolve two targets the distance between them must be larger than the 3 dB beam width of the antenna. The standard deviation for the angular error for a monopulse measurement during a linear scanning mode is related to the SNR:

\[
\sigma_\theta = \frac{\theta_{3\text{dB}} \sqrt{L_b}}{k_s \sqrt{2(S/N)_m n}}
\]

\( \theta_{3\text{dB}} \) is the two-sided 3 dB beam width. \( L_b \) is a beam shape loss factor. \( k_s \) is the discriminator gain slope. \( (S/N)_m \) is the peak SNR during the scan. The term \( k_s \sqrt{2} E_b = 1.4 \) for
2 ESTIMATION

optimal processing. \( n \) is the number of coherent integrations that occurs during the measure of a target, i.e. the number of pulses in Figure 2.4.

If the radar is performing a single measurement in a nonscanning mode the term \( k_s \sqrt{f_b} \) is replaced by \( k_m \) which ranges from 1.2 and 1.9. \((S/N)_m\) is then the \( S/N \) used in the measurement.

In practice the angular resolution is restricted by imperfections in the antenna and the standard deviation is not zero for a large \( SNR \). A good angular resolution measure includes a hardware term and a stable but unknown bias term:

\[
\sigma_{\theta_{\text{tot}}} = \sqrt{(\sigma_{\theta_{\text{bias}}})^2 + (\sigma_{\theta_{\text{hwa}}})^2}
\]  

(2.44)

For low or medium pulse repetition frequency modes the range resolution and accuracy depends primarily on the pulse width. The pulse width determines the range of the discrete matched filters discussed earlier. Two targets can be resolved if they are two or more range bins apart. The range resolution is \( \tau_p c \) where \( \tau_p \) is the pulse width and \( c \) the speed of light. If the target is detected by a single range bin the range accuracy is:

\[
\sigma_R = \frac{\Delta R}{\sqrt{1/2}}
\]  

(2.45)

where \( \Delta R \) is the range bin size. If the target is detected by multiple adjacent range bins the accuracy can be improved:

\[
\sigma_R = \frac{\Delta R}{k \sqrt{2(S/N)n}}
\]  

(2.46)

where \( k = 1.81 \) and \( n \) is the number of integrated pulses.

For a high pulse repetition frequency the range is highly ambiguous because it is not certain which incoming pulse corresponds to which transmitted pulse. This mode is mostly used for measuring range rate. To resolve the range ambiguity three measurements are performed. The first measurement determines the doppler shift. During the second measurement the frequency is ramped at a constant rate \( df_1/dt \). The ramping imposes an additional frequency shift, \( f_1 \), that is proportional to the target range. The third measurement uses a different ramp to resolve ambiguities that exist if there are two targets in the beam and gives \( f_2 \). To calculate the accuracy for a high pulse repetition frequency equation (2.45 or 2.46) is used with:

\[
\Delta R = \frac{c (f_2 - f_1)}{2 \frac{df_1}{dt}}
\]  

(2.47)

The range rate resolution corresponds to the frequency shift detection resolution:

\[
\Delta R = \frac{\lambda}{2} \Delta f
\]  

(2.48)

The range rate accuracy is calculated using eq. (2.45 or 2.46) with \( \Delta R \) instead of \( \Delta R \).
According to [11] this level of sensor understanding is sufficient to model the operation of the radar for a tracking system. However, to gain a deeper understanding for tracking system design and adaptive multiple sensor tracking systems a more detailed understanding is needed. This low-level discussion of the radar sensor is too extensive for this report but some important concepts are outlined below. Please refer to [11] for a more detailed discussion of the above and a complete discussion of low-level models.

**Low-level view**

The low-level model described in [11] discusses the concepts of measurement availability, clutter and eclipsing losses in further detail as well as other important concepts.

Clutter arises from the spreading of the radar beam. The radar beam consists of several lobes and these lobes can be reflected off different objects creating the illusion of several targets. Clutter is divided into four categories: main-lobe clutter, residual main-lobe clutter, side-lobe clutter and altitude return. The discussion in [11] leads to a more detailed version of the radar equation:

\[ SNR = \frac{S}{N + J_{SLC} + J_{MLC} + J_{RMLC}} \]  

(2.49)

where \(J_{SLC}, J_{MLC}\), and \(J_{RMLC}\) are the extra noise from the clutter categories. Altitude return is the doppler effect of the motion between the radar and the ground and is usually small.

Another important phenomenon that affects the measurement availability is eclipsing loss. Most radars cannot transmit and receive signals simultaneously. This is because the transmitted effect is so high that it would destroy the receivers to have these activated while transmitting. This means that the radar is blind while transmitting. Since measurements usually consist of several pulses, the radar will be unable to detect pulses from targets at certain ranges since it might still be transmitting when pulses return. This is called eclipsing loss and it is mainly a problem for higher pulse repetition frequencies.

**2.4.3 IRST-model**

An infrared sensor consists of an array of detectors which are sensible to heat radiation. The data from the sensor is conventionally presented on a monitor and each detector is presented as a pixel forming a TV-like image. The infrared search and track (IRST) sensor is developed for tracking purposes and this sensor also calculates bearing and elevation to detected targets. The incoming radiation is focused on the detection array by a set of scanning mirrors. This is a mechanical system which is limited by inertia. An IRST is therefore primarily a scanning sensor, but it might also lock on to a single target. Since this sensor is passive it must rely on signal processing of received signals.

The intercepted signal power for an infrared sensor is:

\[ S = \tau(R) \frac{\pi d^2 J_s}{4 R^2} \]  

(2.50)

\(\tau(R)\) is an atmospheric transmittance factor that is a function of the range \(R\). \(d\) is the optical diameter of the sensor. \(J_s\) is the radiant flux per unit solid angle.
2 ESTIMATION

The signal to thermal noise ratio is:

\[
SNR = K_0 \tau(R) \frac{\pi d D^* J_1(\frac{t_d}{\alpha_d K_f})}{4 R^2 f_N} \left(\frac{t_d}{\alpha_d K_f}\right)^{1/2}
\]  \hspace{1cm} (2.51)

\(K_0\) is the optical efficiency of the sensor. \(d\) is the aperture diameter. \(D^*\) is the sensor detectivity. \(f_N\) is the optical f-number. \(t_d\) is the time on target. \(\alpha_d\) is the instantaneous field of view in steradians. \(K_f\) is a proportionality constant which is usually close to 1.

The clutter power is:

\[
C = \tau(R_s) \frac{\pi d^2}{4} \alpha_d \int \frac{W(\lambda, T) d\lambda}{\lambda_2 - \lambda_1}
\]  \hspace{1cm} (2.52)

\(R_s\) is the range to the background. \(W\) is the radiant energy which can be calculated using Planck’s law. \([\lambda_1, \lambda_2]\) is the wavelength interval which the sensor is able to detect. \(T\) is the average background temperature.

The probability of detection is approximated as a Gaussian distribution:

\[
P_D = \int_{(T_D - S)/\sigma}^{\infty} \frac{1}{\sqrt{2\pi} \sigma} e^{-\frac{1}{2} t^2} dt
\]  \hspace{1cm} (2.53)

\(S\) is the target amplitude. \(T_D\) is the detection threshold, which is usually equal to \(K \sigma\) where \(K\) is usually between 3 and 5 and \(\sigma\) is the standard deviation of the estimated noise/clutter mean.

The angular accuracy of a typical IRST is so good that the contribution from the angular measurement error is negligible in comparison to the total tracking error. The angular resolution and accuracy is in the order of 0.1 mrad. The good angular resolution depends on the short wave length of infrared radiation.

Please refer to [11] for a more detailed discussion of the IRST.
3

Decentralized estimation

The concept of decentralized tracking was introduced in chapter 1.4 and is here discussed in further detail.

3.1 General discussion

As discussed in chapter 1.4, a centralized tracker has the advantage of being relatively simple and performing optimally, with respect to track quality. It also performs the least amount of calculations and there is no need for communication between different tracking elements, except with the sensors. Decentralized systems need to send estimates back and forth between the local and central filters. This is done through the aircraft’s communication network, since the track fusion is thought to be performed in the central computer. Low communication requirement is a desired filter feature, i.e. to burden the communication network as little as possible. The decentralized architecture is not optimal since it filters the information in several local places. The local filtering is based on the locally available information and does not consider all available data.

Why then, is a decentralized tracking system interesting? In a decentralized tracking system the filter calculations are performed at the sensor level. The central computer has only to fuse the local tracks. In a realistic situation the central node will most likely contain a prediction model as discussed in 1.4, but for the purpose of tracking the central node only needs to perform track fusion. The decrease of calculation burden is perhaps the main reason for a decentralized architecture. Other reasons are the flexibility, graceful degradation and increasing sensor autonomy mentioned in chapter 1.4. It is therefore clear that a decentralized architecture can be interesting and the remaining question is how such a system performs. [3 and 4] concludes that a system with feedback is well comparable in quality with a central system. It is the purpose of this report to investigate this further, including multiple models, adaptive sample rate and a more realistic radar model.

3.2 Track fusion

Since [3 and 4] concludes that feedback greatly enhances the tracking quality for single target tracking without association, only systems with feedback are considered in this report. In a system with feedback the states of the different nodes are highly correlated, because they are based on the same information. They can not be considered independent as is customary to do in a Kalman filter. A Kalman filter can handle cross-correlation if this is included in the process noise. If the cross-correlation can be established a Kalman
filter do provide an optimal track fusion, optimal in the least square sense. However, cross-correlation can be very difficult to establish and to quantify. It is practically impossible in a more complex architecture. Ignoring cross-correlation in a Kalman filter will result in over confident estimates that might lead to divergence of the filter.

Cross-correlation needs to be considered in order to produce a consistent track. Consistency means that the uncertainty of the state estimate corresponds to the actual estimation error, and will be discussed further in chapter 4.2. The consistency needs to be good in order to produce a track estimate with small errors.

There are different ways of dealing with the cross-correlation. [13] discusses two methods of removing cross-correlation by dividing old and new information, using only the new information gained since the last update in the fusion process. [4] implements a fusion filter where the tracks to be fused are assumed to be independent, like in a normal Kalman filter application. To compensate for the cross-correlation, the resulting covariance matrix is multiplied by a constant (larger than one). [3] evaluates these two methods and a third method called covariance intersection, which will be described in the next chapter.

A conclusion in [3] is that both the method of removing cross-correlation and the covariance intersection method produces consistent fusions. The method of assuming independence fail to produce consistent fusions in a feedback system. Covariance intersection proves to be somewhat better than removing cross-correlation and is therefore used in this report.

### 3.3 Covariance intersection

The covariance intersection method is a fusion method that produces consistent estimates for any cross-correlation without actually knowing the cross-correlation. Chapter 3.3 is based on [3, 14 and 15].

#### 3.3.1 The covariance intersection idea

Let’s assume that the estimated states that are to be fused are \( \hat{a} \) and \( \hat{b} \), with true values \( a \) and \( b \). The deviations from the true values are \( \hat{a} = a - \hat{a} \) and \( \hat{b} = b - \hat{b} \). Then:

\[
\begin{align*}
P_{aa} &= E(\hat{a}\hat{a}^T) & P_{ab} &= E(\hat{a}\hat{b}^T) = E(\hat{b}\hat{a}^T) = P_{ba} & P_{bb} &= E(\hat{b}\hat{b}^T)
\end{align*}
\]

These values are not known but are approximated by \( P_{aa} \) and \( P_{bb} \), where \( P_{ab} \) is approximated to be zero. The Kalman filter, uses a linear fusion in the form [14]:

\[
\dot{\hat{c}} = w_a \hat{a} + w_b \hat{b} \tag{3.1}
\]

\[
P_{cc} = w_a P_{aa} w_a^T + w_a P_{ab} w_b^T + w_b P_{ba} w_a^T + w_b P_{bb} w_b^T \tag{3.2}
\]

\( w_a \) and \( w_b \) are weights chosen to minimize the trace of \( P_{cc} \). If \( P_{ab} = 0 \) this fusion method is consistent, if \( \hat{a} \) and \( \hat{b} \) are consistent. However, if \( P_{ab} \neq 0 \) a consistent update is difficult to guarantee according to [14].

In the event that \( \hat{a} \) and \( \hat{b} \) are two-dimensional, the fusion update can be visualized by plotting the covariance ellipses of the corresponding states. It will then be seen that no
matter how $P_{ab}$ is chosen, $P_{cc}$ will always lie within the intersection of \(P_{aa}\) and \(P_{bb}\). An example of an intersection of \(P_{aa}\) and \(P_{bb}\) can be seen in Figure 3.1. If \(P_{cc}\) always lies within the intersection of \(P_{aa}\) and \(P_{bb}\), then a fusion algorithm that finds a \(P_{cc}\) that encloses the intersection will automatically be consistent, no matter what \(P_{ab}\) actually is. This is the idea behind covariance intersection. Covariance intersection is also known as Gauss intersection.

### 3.3.2 Covariance intersection equations

The covariance intersection method takes a convex combination of the inverse of the means and covariance:

\[
P_{cc} = \left[ \omega P_{aa}^{-1} + (1 - \omega) P_{bb}^{-1} \right]^{-1}
\]

\[
\hat{c} = P_{cc} \left[ \omega P_{aa}^{-1} \hat{a} + (1 - \omega) P_{bb}^{-1} \hat{b} \right]
\]

ω is chosen to minimize \(P_{cc}\) so that it will enclose the intersection as tightly as possible.

In the case of a two-dimensional covariance matrix, the covariance can be graphically expressed in terms of k-sigma contours, defined by:

\[
(x - a)^T A^{-1} (x - a) = k
\]

\(a\) is here the state and \(A\) the covariance of \(a\). See Figure 3.1 for a graphical illustration of equations (3.3 and 3.4) where \(k = 1\).

![Figure 3.1](image-url) The two solid ellipses are the covariance for \(a\) and \(b\). The dashed ellipse is the covariance for \(c\), calculated with the Covariance Intersection method.

The covariance is plotted as an ellipse with the locus at the state mean and the shape of the ellipse corresponds to the uncertainty in different directions. If the sensors of this report are located at the bottom of this page, the a-ellipse could represent the radar covariance and the b-ellipse the IRST-covariance. A conclusion that can be drawn from Figure 3.1 is that two sensors are better than one, if the sensors are unbiased.

It can be proven that the covariance intersection produces a consistent \(P_{cc}\), if \(\hat{a}\) and \(\hat{b}\) are consistent, for any choice of \(P_{ab}\) and \(\omega\). A proof thereof can be found in [14].
3 DECENTRALIZED ESTIMATION

Equations (3.1 and 3.2) can easily be expanded to fuse more than two tracks, if more than two sensor filters are used:

\[ P_{\text{tot}}^{-1} = \sum_n \omega_n P_n^{-1} \]  \hspace{1cm} (3.6)

\[ \sum_n \omega_n = 1 \]  \hspace{1cm} (3.7)

\[ \hat{x}_{\text{tot}} = P_{\text{tot}} \sum_n \omega_n P_n^{-1} \hat{x}_n \]  \hspace{1cm} (3.8)

3.3.3 The choice of the weight \( \omega \)

The efficiency of the covariance intersection method depends on the choice of \( \omega \). Figure 3.2 illustrates the resulting covariance for different choices of \( \omega \) in a 2-dimensional example. The solid ellipses are the covariance matrices to be fused and the dashed ellipse is the resulting covariance.

![Figure 3.2](image-url)

*Figure 3.2* The two solid ellipses are fused into the dashed ellipse using equations (3.3) and (3.4) for different \( \omega \).
To gain the most information out of the two original estimates, it is desired to choose $\omega$ in such a way that the resulting covariance is as small as possible, while enclosing the intersection. In order to minimize $P_{cc}$ it is necessary to define a norm of the size of $P_{cc}$.

There are several possibilities to choose a norm, for instance the determinant, trace or Frobenius norm of $P_{cc}$ or maximum value or eigenvalue for $P_{cc}$. The choice of norm is a choice between an accurate size measurement and computational burden. A norm that considers all elements, like determinant or Frobenius norm, should likely produce better estimates than the trace or maximal value norm. They are, however, more complex than methods that only considers one or a few of the elements in the matrix. A norm like the determinant is tolerant to differences in size of the elements in the state vector. In our state vector (2.18) element 1 and 2 are probably much larger than the other states. With the determinant as norm they will not be as dominant as with, e.g. the trace norm. In existing literature the trace and determinant seem to be the most common, see [3, 14 or 15].

In this report the determinant is used as a norm, and $P_{cc}$ is minimized by:

$$
\min_{[0,1]} \omega \frac{1}{\text{det}(\omega P_{aa}^{-1} + (1-\omega)P_{bb}^{-1})}
$$

(3.9)

Figure 3.2 d) illustrates $P_{cc}$ when $\omega$ is chosen by minimizing the determinant norm.

Cost functions, such as (3.9), which are convex with respect to $\omega$ only have one optimum in the range $0 \leq \omega \leq 1$ and are straightforward to optimize. In Matlab™, optimizing is easy, see appendix C and functions fusionfilter.m and covint.m.

The covariance intersection method produces consistent estimates for any $\omega$, even a constant one, and it might seem unnecessary to bother finding the minimizing $\omega$. However, by optimizing $\omega$, the performance of the fusion is optimized. If $\omega$ is chosen without consideration, the information in the different tracks may be lost, which could lead to divergence in a system with feedback.

Covariance intersection can also be used to triangulate a position using angle-only sensors at different locations. If the sensors measure direction exactly their covariance ellipses will be reduced to covariance lines. In Figure 3.3 sensor A measures the target to be somewhere along covariance line $a$ and sensor B somewhere along covariance line $b$. Then the target must be where the lines intersect. The fused mean is shown with an $x$ and the resulting covariance is invisibly small.

"Figure 3.3 Covariance intersection when the uncertainty is in one dimension."
3 DECENTRALIZED ESTIMATION

3.4 Central architecture with IMM

As a comparison to the decentralized architectures later on, the central architecture in Figure 1.1 can be illustrated with an IMM-structure as in Figure 3.4.

A corresponds to the calculation of the mixing matrix and B to the actual mixing of the states. C consists of the 3 filters. The estimates are then fed from C back to A and forwarded to D where the new model probabilities are calculated and where the interpolated state estimate is produced. D also controls the adaptivity of the radar sensor by F. G illustrates how the new model probabilities are used at the beginning of the next IMM cycle.

![Figure 3.4 Central tracking architecture with 3 IMM-filters](image)

3.5 Decentralized architecture 1

The first decentralized architecture to be evaluated looks like the one in Figure 1.2. It is, however, more complex since the local filters are IMM-filters with 3 models. See Figure 3.5.

3.5.1 The architecture

A is the radar filter. B is the IRST filter. C is the fusion node, where the radar track and IRST track are fused. The fusion is done in the central computer using covariance intersection as described in chapter 3.3. The only other function that the central computer must maintain is the feedback control, D. When the central track is fed back it has to be fused into the 3 IMM-filters in the local filters, E. [4] simply replaces the local tracks with the central feedback track. [3] fuses the feedback track with the local tracks with the same fusion method as in the fusion node C. Some simulations where the feedback track replaces the local tracks and where the tracks were fused were made during the implementation of these filters. They strongly suggest that the feedback track should be fused...
with the local tracks, like in [3]. Fusion in $E$ gave a much more stable filter structure. The somewhat unstable behavior of the system, if replacement is used, is probably due to the fact that the model probabilities and the new tracks do not correspond well with one another.

The local filters produce a track estimate each at every cycle, which are forwarded to the central node each cycle. If the radar uses adaptive sample rate, the radar estimate is only a prediction at the intervals when no measurement is performed, but it is still forwarded to the central node and fused with the IRST-track. Another option, which is not being investigated here, is to forward local estimates only when a measurement has been performed. However, if the feedback rate is low, the radar filter may have performed a radar measurement which is not yet fed back to the IRST-filter through the central filter. Since the central node does not contain a kinematic model, it can not receive a track and then predict it in order to feed it back later to the IRST-filter when this is possible, at least not in this system. The information will then be lost for the IRST-filter. Therefore, this architecture lets the local filters “remember” measurement data, which then are reflected in the estimates that are produced and fed forward at each cycle. The IRST is thought to measure every cycle without adaptivity.

To make communication easier between the different nodes, all nodes use the same state vector. This eliminates coordinate transformation. This is perhaps not an ideal solution, but it simplifies the filter structure. The state vector in the local filters is chosen as dis-

\[ \begin{align*} 
A & \quad \text{Radar filter} \\
B & \quad \text{IRST-filter} \\
C & \quad \text{Central filter} \\
D & \quad \text{Feedback control} \\
E & \quad \text{Track fusion} 
\end{align*} \]

Figure 3.5 Decentralized architecture 1
DECENTRALIZED ESTIMATION

Discussed for the central tracker in chapter 2.1.2. Since the radar filter has access to \((r, \dot{r}, b_r)\) it is natural to choose the same state vector as for the central tracker. For the IRST filter, which only has access to \(b_{ir}\), it will be impossible to track a target for a longer period of time. It can not estimate \(r\) accurately, which will eventually lead to divergence in the state estimate. For this reason it is possible to choose other coordinates that reduce the influence of \(r\) on the estimate, for instance modified spherical coordinates or to use an angle-only filter. Since the feedback rate in this architecture is thought to be relatively high, the feedback will most likely prevent the IRST-filter from diverging. The simplicity of using the same coordinates is the reason that the state vector (2.18) is chosen for both the radar and the IRST filter.

3.5.2 The radar filter

The radar filter looks basically like the central filter in Figure 3.4. It receives data from the radar sensor and controls when the radar should be used. When the central node feeds back a central, and better, track to the radar filter, the radar filter fuses the central track with each of its 3 filters and the IMM cycle rolls on. If a central track is not fed back, the 3 filters proceed like a normal IMM-filter.

As discussed, the same state vector is chosen for the radar filter as for the central architecture filter. However, since \(b_{ir}\) is not accessible the measurement vector will contain only the first three elements of equation (2.19):

\[
y(t) = \begin{bmatrix} r \\ \dot{r} \\ b_r \\ \dot{b_r} \end{bmatrix} + w(t) = \begin{bmatrix} x_1 \\ x_2 \\ v_\phi \cos \phi + x_3 v_\phi \sin \phi \\ \sqrt{x_1^2 + x_2^2} \end{bmatrix} + \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}
\] (3.10)

This also affects the Jacobian \(H'\):

\[
H' = \begin{bmatrix}
\frac{x_1}{r} & \frac{x_2}{r} & 0 & 0 \\
-\frac{x_1}{r^3} \alpha + \frac{\ddot{x}_1}{r^2} \alpha + \frac{\ddot{x}_1}{r} \cos \phi + \frac{\ddot{x}_2}{r} \sin \phi - \frac{x_1 v_\phi \sin \phi + x_3 v_\phi \cos \phi}{r} & 0 & 0 & 0 \\
-\frac{x_2}{r^2} & \frac{x_1}{r^2} & 0 & 0 \\
\end{bmatrix}
\] (3.11)

Apart from this difference, the filter is implemented as in chapter 2.

3.5.3 The IRST filter

The fusion of feedback tracks into the local tracks are performed as for the radar filter. The IRST filter has only access to \(b_{ir}\) and therefore the measurement vector looks like:

\[
y(t) = \begin{bmatrix} b_{ir} \\ \dot{b}_{ir} \end{bmatrix} + w(t) = \begin{bmatrix} \tan(x_2/x_1) \end{bmatrix} + \begin{bmatrix} w_4 \end{bmatrix}
\] (3.12)

and the Jacobian \(H'\):

\[
\]
The idea of splitting a central filter into two local filters is further described in [4].

### 3.6 Decentralized architecture 2

Any kind of preprocessing leads to a lesser amount of information in the central node. The ideal situation is naturally that the preprocessing leaves an estimate where all important information is still included. Perhaps architecture 1 filters too hard, leaving insufficient information for the central node. Because of this suspicion another filter architecture is investigated.

This decentralized architecture uses more communication between the nodes, providing the central node with more information, and perhaps with a better estimate. Figure 3.6 illustrates this architecture.

![Figure 3.6 Decentralized architecture 2](image)

The main difference between this architecture and the previous one is that here all 3 filters in the local filters are fed forward to the central node \( C \). The radar filter, \( A \), does not interpolate its multiple models at the end of the IMM-cycle. Neither does the IRST filter, \( B \). The first radar model is fused with the first model in the IRST-filter, the second to the second and third to the third. The feedback of all three tracks are controlled by \( G \). Each local

\[
H' = \begin{bmatrix}
-x_1 & x_2 \\
-x_3 & 0 \\
\end{bmatrix} \begin{bmatrix}
r_1 \\
r_2 \\
0 \\
0
\end{bmatrix}
\]  

(3.13)
3 DECENTRALIZED ESTIMATION

filter calculates its model probabilities, $E$, which are also fed to a central node, $F$. $F$ averages the two sets of model probabilities and feed these new probabilities back to the local filters when $G$ feedbacks the tracks. The probabilities in $F$ are also used in the model interpolation in $D$. $D$ is needed to present the result of the tracking system. The central computer has to control $C$, $D$, $F$ and $G$.

Architecture 1 can be seen as two separate IMM-filters which are fused and the fused track is fed back to the local filters. Architecture 2 can actually be seen as one IMM-filter, where the three IMM-models interact at the beginning of the cycle. Then each model is split in two where one model is updated with the radar sensor and the other with the IRST sensor. The two split models are then fused back into one track and the model probabilities are updated. Just like in the normal IMM-cycle, the three tracks are interpolated at the end to produce a result. When full feedback is used, that is. At periods without feedback, the local filters perform their own IMM-cycle.

This architecture does not need fusion between the local tracks and the central track that is fed back. This is because of the fusion and feedback of model probabilities. When full feedback is used, the two filters will have the same model probabilities and make the same predictions. When feedback is not full, but occurs at a fairly frequent rate, the probabilities are more or less the same in the two filters and fusion like in architecture 1 will hopefully not be required.

The local filters in the radar and IRST-filter in this architecture are the same as for architecture 1 in chapter 3.5.2 and 3.5.3.

3.7 A comparison of the decentralized architectures

Architecture 1 (type 1) is more simple than architecture 2 (type 2). It puts less computation burden on the central node and needs less communication. This all support type 1. The reason for investigating type 2 at all is the suspicion that perhaps the local filters filter too much and leaves the central node with an estimate that is not as accurate as it could be.

Therefore it is interesting to evaluate these two filters in simulations. Some interesting qualities to evaluate are: track quality, radar update rate, feedback rate, computation burden and communication load. If type 1 has an equal tracking quality to type 2, then there is no need for the complex type 2. If type 2 performs better then type 1, perhaps it does not need to have as high feedback rate or update rate as type 1, and the computation burden and the need for communication could be lowered. These are issues that will hopefully be answered by the simulations presented in chapter 5.

The two types are also evaluated against a central tracker to see if they can achieve a comparable tracking quality.

If type 2 performs better than type 1, it could be interesting to see if the performance of type 1 can be improved if the model probabilities are fused and fed back as in type 2. Another alternative version of type 1 is to eliminate the interaction in the beginning of the IMM-cycle in cycles with feedback. This may seem like an odd thing to do, but considering Figure 3.4 it is apparent that the three model estimates are fused once in the interpolation phase at the end of the IMM-cycle. This fused estimate is then being fed back and
fused with the local tracks which then again interact at the beginning of the IMM-cycle. The interpolation at the end of the IMM-cycle is similar to the interaction at the beginning of the IMM-cycle and is not a part of the normal filter cycle. It might be that the three different models are not very different after all in the original version of type 1.

The purpose of the simulations can be summarized:

Compare type 1 and its two alternatives to type 2 and both these types to the centralized architecture. Evaluate the architectures with respect to: track quality, radar update rate, feedback rate, computational burden and communication load.
3 DECENTRALIZED ESTIMATION
Simulation definitions

A simulation consists of a scenario which is defined by one or more flight paths representing one or more targets, i.e., aircrafts. These flight paths are vectors which contain measurement data used in the simulation of the filters. Gaussian noise is added to simulate measurement error. An ideal filter would then be able to reproduce the actual flight path out of a noisy measurement vector. In order to evaluate the performance of a filter a number of evaluation quantities and qualities need to be specified.

4.1 Measures of performance

State error

One obvious evaluation quantity is the position error, \( \tilde{d}(k) \) and its average.

\[
\tilde{d}(k) = \sqrt{(\hat{x}_1(k) - \hat{x}_1(k|k))^2 + (\hat{x}_2(k) - \hat{x}_2(k|k))^2}
\]

\[
\bar{d} = \frac{1}{N} \sum_{k=1}^{N} \tilde{d}(k)
\]

\( \hat{x} \) is the actual state, \( \hat{x} \) is the estimated state and \( N \) is the number of time steps that the simulation lasts. It can also be interesting to observe the velocity error:

\[
\hat{v}(k) = |\hat{x}_3(k) - \hat{x}_3(k|k)|
\]

Consistency

If the estimation error statistically correspond to the state vector’s covariance matrix the filter is said to be consistent. One measure of consistency is the normalized state error squared variable [16]:

\[
\eta_k = \tilde{x}^T_k P^{-1}_{kk} \tilde{x}_k
\]

where \( \tilde{x}_k = \hat{x}_k - \hat{x}_k \) is the full state error vector. \( \eta_k \) is \( \chi^2(n) \)-distributed, where \( n \) is the number of states, which in this case is 5. If the filter is consistent, \( \eta_k \) should stay inside its confidence interval. For these simulations a 95% confidence interval is used, i.e.
in 95% of the samples $\eta_k$ should stay within this region. The confidence interval is determined with a $\chi^2$-table.

**Adaptivity**

One very interesting quality to investigate is the radar update rate. It is desired to have a low measure frequency for the radar. How many radar measures are required for different architectures? How does the radar update rate affect the tracking quality and how does the feedback rate affect the update rate and tracking quality for the different architectures?

**Computation load and communication**

It is interesting to see how much computation the different architectures need to perform, and where in the structure it is performed. Does the decentralized approach ease the computation burden for the central computer? How do the decentralized architectures differ from one another? Also interesting to observe is how much data communication is required between different nodes in the decentralized architectures. Both computation and communication load will depend on the radar update rate and the feedback rate.

**Track losses**

Sometimes a filter is unable to successfully track a target. This is the case when the estimate diverges from the true state without the filter having the ability to correct it. Track loss is a serious negative quality of a filter, since a filter that loses tracks can not be trusted to the same extent as a filter that does not lose tracks in similar circumstances. In these simulations a track is considered lost when its average position error is more than 500 m. Position divergence arises when there is a dissonance between the actual position error and the estimated uncertainty.

### 4.2 Monte Carlo simulations

Since the state estimate and measurements are stochastic and not deterministic, it is interesting to run a series of simulations of the same scenario with each filter, but with different measurement noise realizations. Due to the measurement noise, simulations that are otherwise identical can behave quite differently and therefore a single simulation is not of great interest. A series of simulations can be averaged and the measures of performance can be expressed in statistical terms. Such a series of simulations with the same conditions is called a Monte Carlo simulation. In a Monte Carlo simulation equation (4.1) is replaced by the root mean square error [6]:

$$RMSE(k) = \left( \frac{1}{M} \sum_{i=1}^{M} ||x(k) - \hat{x}_i(k)||^2 \right)^{1/2}$$  \hspace{1cm} (4.4)

where $M$ is the number of simulations and $k$ is the time index. Equation (4.2) is replaced by:

$$RMSE = \left( \frac{1}{N} \sum_{k=1}^{N} \frac{1}{M} \sum_{i=1}^{M} ||x(k) - \hat{x}_i(k)||^2 \right)^{1/2}$$  \hspace{1cm} (4.5)

where $N$ is the number of time steps of the simulation.

The average normalized state error squared, $\bar{\eta}_k$, multiplied by the number of simulations, $M$, is $\chi^2(nM)$-distributed where $n$ is the number of states. In these simulations $n = 5, M =$
20. A table gives that $\eta_k \leq 6.2$ if a filter is consistent [16]. The number of radar measures are also averaged. The number of lost tracks during the simulations is counted.

### 4.3 Sensor and Filter Data

The sensor model used in the simulations have the following covariance: $\sigma_{\eta}^2 = 40^2 m$, $\sigma_{\eta}^2 = 20^2 m/s$, $\sigma_{\eta_{\eta}} = 0.02^2 rad$ and $\sigma_{\eta_{\eta_{\eta}}} = 0.002^2 rad$. The central architecture uses all four sensor data, while the decentralized architectures use the three first in their radar filter and the fourth in their IRST filter.

The system noise can, as previously mentioned, be seen as a design parameter and is adjusted separately for each filter architecture. The same system noise is used in both the radar and the IRST-filters for the decentralized architectures, since the target that the filters track is the same and moves according to the same kinematic model.

**Centralized Filter (described in chapter 3.4)**

Process noise for the central architecture that gave good results are for the three IMM-models:

\[
\begin{align*}
\nu_{t1}^2 &= 1 \times 10^{-6} & \nu_{n1}^2 &= 2 \times 10^{-6} \\
\nu_{t2}^2 &= 1 \times 10^{-2} & \nu_{n2}^2 &= 2 \times 10^{-2} \\
\nu_{t3}^2 &= 1 & \nu_{n3}^2 &= 2 \\
m/s^2 & & rad/s^2
\end{align*}
\]

The index $t$ indicates tangential direction and index $n$ normal direction. That is, the noise is thought tangential and normal to the flight direction. In the filter implementation, these process noise parameters are ordered in the matrix $Q$, see chapter 2.1.1.

The values of $\nu_t$ and $\nu_n$ may seem small. The chosen coordinates suggests a constant random turning and tangential acceleration and to model a straight trajectory or constant velocity, the system noise is low.

The predefined transition likelihood matrix for this architecture is:

\[
\begin{bmatrix}
0.90 & 0.05 & 0.05 \\
0.10 & 0.80 & 0.10 \\
0.01 & 0.09 & 0.90
\end{bmatrix}
\]

This is also a design parameter. The first row describes the mixing probabilities for filter model 1, the second the second, etcetera. Element (1,1) is the probability that the first model is the true model given that the first model was the true model in the previous cycle. Element (1,2) is the probability that model 2 is the true model given that model 1 was the true model in the previous cycle. And so on. Every row must summarize to 1.
4 SIMULATION DEFINITIONS

Decentralized filter 1 (described in chapter 3.5)
Process noise for decentralized architecture 1 are:

\[
\begin{align*}
&v_{11}^2 = 1 \times 10^{-6} & v_{21}^2 = 2 \times 10^{-6} \\
v_{12}^2 = 2 \times 10^{-2} & v_{22}^2 = 4 \times 10^{-2} \\
v_{13}^2 = 2 & v_{23}^2 = 5 \\
\text{m/s}^2 & \text{rad/s}^2
\end{align*}
\]

These values are a little bit bigger but they are of the same magnitude. The transition likelihood matrix for this architecture is:

\[
P_{ij} = \begin{bmatrix}
0.85 & 0.05 & 0.10 \\
0.10 & 0.80 & 0.10 \\
0.08 & 0.17 & 0.75
\end{bmatrix}
\]

Decentralized filter 2 (described in chapter 3.6)
Process noise for decentralized architecture 2 are:

\[
\begin{align*}
&v_{11}^2 = 1 \times 10^{-6} & v_{21}^2 = 2 \times 10^{-6} \\
v_{12}^2 = 2 \times 10^{-3} & v_{22}^2 = 4 \times 10^{-3} \\
v_{13}^2 = 0.5 & v_{23}^2 = 1 \\
\text{m/s}^2 & \text{rad/s}^2
\end{align*}
\]

These values are also similar to those of the other filters.

\[
P_{ij} = \begin{bmatrix}
0.85 & 0.05 & 0.10 \\
0.12 & 0.76 & 0.12 \\
0.08 & 0.12 & 0.80
\end{bmatrix}
\]

4.4 Scenario
The test scenario for the simulations is described in Figure 4.1. The flight path starts by the x and ends by the *. The sensors are located at the origin, o. The scenario lasts for 200 seconds and consists of 200 samples. The first turn is a left 5-g turn, followed by a slow 0.5-g turn left. The first right turn is a 2-g turn, followed by a sharp 8-g turn. The scenario ends with a right 3-g turn. The sharp 5 and 8-g turns should cause challenging problems, but it will also be interesting to see how well the filters can detect the slow 0.5-g turn and how it handles the mixture of different turns and straight flight paths. The aircraft flies at a constant speed during the scenario, with \(v = 291 \text{m/s}\).
The flight path is generated by a Matlab™ file developed by Defence Research Establishment. This file takes breakpoints, velocity and g-forces as arguments and generates a flight path with turns at the breakpoints with the stated g-forces. This means that the flight path is not generated using the same model as in the state vector.

Figure 4.2 illustrates a typical measurement noise realization for a target with the flight path in Figure 4.1, i.e. how wrong the radar sensor measures position during the flight scenario. The measurement error is clearly dependent upon the distance between the sensor and the target. The measurement noise is white, but because of the noise in the angle-measurement the absolute error is bigger with longer distance.
4 SIMULATION DEFINITIONS

4.5 Limitations

The simulations presented in chapter 5.1-5.4 uses an ideal sensor model, see 2.4.1. The filter nodes in the decentralized filters are synchronous with a period time of 1 s. The filters start with correct initial values and do not perform any initiation. The filters track a single target. A limitation of the simulation is also that the filters are only evaluated in one 2D-scenario and that this scenario uses a target that flies with a constant velocity.
Simulations

This chapter presents the outcome of the simulations. The filters in chapter 5.1-5.4 all use an ideal radar model and a fixed update limit. Chapter 5.1 presents the simulations for the central filter. Chapter 5.2 presents the decentralized filter 1 and chapter 5.3 decentralized filter 2. The simulations are divided into sections where the filters are evaluated with respect to full radar use and adaptive radar use. The decentralized filters are also evaluated with respect to different feedback rates. In chapter 5.4 the two alternatives of the decentralized filter 1 are simulated. Chapter 5.5 compares calculation and communication burden for the architectures.

5.1 Central filter

5.1.1 Regular radar use for central filter

A typical realization of the scenario described in chapter 4.4 looks like in Figure 5.1 when the radar is used on a regular basis every second with feedback also every second.

![Figure 5.1 A typical simulation for the central filter. The numbers in the graph represents time [s].](image)

Avg. est. error: 13.3 m, Number of radar: 200/200
5 SIMULATIONS

With the scale of Figure 5.1 it is difficult to spot any difference between the actual trajectory and the estimated track. The average position error is 13.3 m which is good considering that the measurement noise is similar to that in Figure 4.2.

Consistency, position and velocity error of the filter in a 20 run Monte Carlo (MC) simulation are presented in Figure 5.2. The top graph of Figure 5.2 shows the normalized state error squared variable, $\eta = \hat{x}^T P^{-1} \hat{x}$, along with its 95% confidence interval, see chapter 4.1. $\eta$ should stay within this interval in 95% of the samples. Generally, if $\eta$ is small, the process noise is over estimated and if $\eta$ is large the process noise is under estimated with respect to the actual error. Ideally $\eta$ should not be too small since the information from the prediction is then not considered enough in comparison to the measurements. The filtered state will depend heavily on the measurements and the filter will not be used. On the other hand, if $\eta$ is large the process noise is smaller than the actual error. This could result in a filter that is divergent.

In Figure 5.2 the filter is consistent for the most parts, with two exceptions. There is one peak around 25 s into the track and one between 140 and 150 s. These peaks correspond to the two sharpest turns in the trajectory; the first to the 5-g left turn and the second to the 8-g right turn. The reason that the 5-g turn is more difficult to track in this simulation is that it takes place much further away from the sensors and thus have bigger measurement error. Both these turns are relatively sharp and they proved to be difficult to successfully track. However, these tops are not alarmingly big and considering the position error in the middle graph, we can see that the filter actually performs very well.

The middle graph of Figure 5.2 illustrates the position deviation. The solid line is the position deviation and the dot-dashed line is the predicted covariance that is used as uncertainty limit for adaptive radar use, although adaptivity is not used in this particular simulation. We can see that the predicted uncertainty corresponds very well to the actual.
error, with the exception of the 8-g turn where the filter is too slow. The actual error is limited and the filter soon manages to estimate the uncertainty well after the turn.

The lower graph of Figure 5.2 illustrates the velocity deviation. The velocity error is very small throughout the simulation. This is perhaps not so strange since the target has a constant speed. This graph is perhaps more interesting in a scenario where the target accelerates tangential to the flight path. However, it is very important that the velocity estimation is accurate in order for the predictions to work. This graph is valuable in order to tune the filter parameters.

The average position error over the 20 simulations is 13.5 m and the standard deviation of the position error is 1.01. The filter manages to track all 20 runs and does not lose any targets.

5.1.2 Adaptive radar use for central filter

The simulation results of chapter 5.1.1 was achieved after using an update limit of zero meters, i.e. the radar was used as soon as the measure of uncertainty, see chapter 2.3.1, was greater than zero meters. This is of course always true, and therefore the radar was used every second. This chapter presents the results of simulations of various update limits. Table 5.1 presents the data from these simulations.

<table>
<thead>
<tr>
<th>Update limit [m]</th>
<th>No. of measures</th>
<th>Avg. pos. error [m]</th>
<th>Std of pos. error [m]</th>
<th>Lost tracks</th>
<th>MC runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>200</td>
<td>13.5</td>
<td>1.01</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>30</td>
<td>100.5</td>
<td>15.2</td>
<td>1.08</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>50</td>
<td>37.8</td>
<td>28.2</td>
<td>6.49</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>100</td>
<td>15.9</td>
<td>49.5</td>
<td>16.8</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>150</td>
<td>9.4</td>
<td>87.7</td>
<td>31.1</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>200</td>
<td>7.2</td>
<td>115</td>
<td>27.5</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>300</td>
<td>5.3</td>
<td>130</td>
<td>27.6</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>400</td>
<td>3.8</td>
<td>148</td>
<td>43.6</td>
<td>0</td>
<td>20</td>
</tr>
</tbody>
</table>

The update limit was discussed in chapter 2.3.2. Column two, three and five are discussed in chapter 4.1 and 4.2. Column four presents the standard deviation of the position errors that are averaged in column three. Column six is the number of Monte Carlo simulations performed.

The adaptivity results of this filter must be considered good. By allowing the position uncertainty to be 30 m instead of zero we reduce the number of radar measurements by half. The resulting position error is only slightly bigger, 15.2 m instead of 13.5 m. By allowing larger and larger update limits the filter performs fewer and fewer radar measures ending in a remarkable 3.8 average when the update limit is 400 m. As the number of measurements decrease the position error naturally increase and the standard deviation of the position error also increase. The outcome of a simulation run with a high update limit varies more than for a low update limit, i.e. the result is more dependent on the actual measurement noise realization when the filter makes fewer measurements.
5 SIMULATIONS

It can be interesting to evaluate the filter a little closer than in table 5.1 to observe consistency, position deviation for different update limits. Monte Carlo simulations with an update limit of 30 m is presented in Figure 5.3. The consistency is very similar to that of an update limit of zero. The top around 25 s is slightly bigger, but the top between 140 and 150 s is actually a little bit better. We can see in the middle graph that the uncertainty level is higher between 40 and 110 s. It is interesting to see that the filter qualities are almost unchanged even though the radar only measures every other second. The velocity deviation is virtually the same.

Figure 5.4 illustrates one estimated track and the radar measures that were made in a typical run. The scale is still too large to separate the track estimate from the actual track. The scale is however enough to illustrate the measurement noise. If the measurements would have been without noise, they would all lie on the solid line. We can see that a great number of measurements have been saved between 30 and 110 s in the scenario where the trajectory is relatively easy to track. Between 130 and 145 the filter does not make any radar measurements. Apparently the short distance and the angle between the sensors and target are good for the IRST-sensor. The uncertainty is relatively big after the sharp 8-g turn and it takes the filter many measurements after the turn in order to lower the uncertainty.

In Figure 5.6 the filter qualities of Monte Carlo simulations with an update limit of 150 m are presented. Now we can see that $\eta$ is bigger; it is very close to the 95% confidence interval at all times during the simulation runs. The estimation error variable $\eta$ is actually outside the interval for more than 95% of the time, but on the other hand between 30 and 100 it is just barely outside. $\eta$ is limited at all times and the filter can be considered consistent with just a little bit of good will. The position error is larger than the estimated uncertainty between 25 and 100. This is what causes the $\eta$ to be so big during this period. It seems like the IRST-sensor has difficulties in detecting the slow changes that take place during this period. Altogether the IRST-sensor is a valuable component in the filter struc-
ture, as can be seen in Figure 5.5. The IRST-sensor is able to track the target the 90 first seconds with only the radar measurement at the start as help.

In Figure 5.5 it is possible to see the position error for one typical simulation. The drifting estimate between 40 and 90 s and the difficulties with the turns at 120 and after 145 s can be seen.

It is clear that adaptivity can save much radar resources for a central filter. This filter structure makes the radar use flexible. If a high accuracy estimate is required the radar can measure every one or two seconds, but if the target is distant or not considered dangerous it might be enough to measure every 10 or even 20 seconds on average and still have an average position error of less than 100 m.

Figure 5.6 Consistency, position and velocity deviation for the central filter using an update limit of 150 m for 20 Monte Carlo runs.
5 SIMULATIONS

5.2 Decentralized filter 1

5.2.1 Regular radar use for decentralized filter 1

The filter in this chapter uses the radar, IRST-sensor and feedback every second. A typical realization of this filter can be seen in Figure 5.7. This track looks very similar to the result in Figure 5.1. The average position error over 20 Monte Carlo simulations is 17.7 m.

Figure 5.8 shows the filter qualities consistency, position and velocity error over 20 Monte Carlo simulations. The consistency of this filter is good as can be seen in the top graph, although the scale is a bit unfortunate. In fact the top around 25 s is better than for the central filter. The middle graph illustrates that the estimated uncertainty very well covers the actual error. The velocity error in the lower graph is somewhat bigger, but is still small.

Figure 5.7 A typical simulation for decentralized filter 1

Figure 5.8 Consistency, position and velocity deviation of decentralized filter 1 for 20 Monte Carlo simulations.
5.2.2 Adaptive radar use for decentralized filter 1

In this chapter the filter is evaluated with respect to its radar update adaptivity. Feedback is still performed every second. Table 5.2 presents the data from these simulations.

The two most striking differences between the filters in table 5.1 and 5.2 are that this decentralized filter needs more radar measurements and that it looses some tracks. The fact that the filter looses tracks is a disturbing quality. These tracks are lost in the sharp 8-g turn. If turns like these are sharper than what a tracker is actually expected to handle there is no problem. Benchmark scenarios usually do not contain turns that are this sharp, but it is naturally a big drawback of this filter compared to the central filter, which manages perfectly. The standard deviation of the position error is also larger than for the central filter, suggesting that this filter is more influenced by the noise realization. Each row of data in the table are based on 20 successful simulations. If a track is lost it is run again and the data is discarded. This means that the filter lost 3 out of 23 simulations for update limit 30 m and 17 out of 37 for update limit 400 m.

Table 5.2

<table>
<thead>
<tr>
<th>Update limit [m]</th>
<th>No. of measures</th>
<th>Avg. pos. error [m]</th>
<th>Std of pos. error [m]</th>
<th>Lost tracks</th>
<th>MC runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>200</td>
<td>17.7</td>
<td>1.47</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>30</td>
<td>195</td>
<td>19.0</td>
<td>2.84</td>
<td>3</td>
<td>23</td>
</tr>
<tr>
<td>50</td>
<td>99.8</td>
<td>29.1</td>
<td>14.6</td>
<td>1</td>
<td>21</td>
</tr>
<tr>
<td>100</td>
<td>74.9</td>
<td>37.8</td>
<td>28.7</td>
<td>1</td>
<td>21</td>
</tr>
<tr>
<td>150</td>
<td>35.4</td>
<td>67.6</td>
<td>44.0</td>
<td>1</td>
<td>21</td>
</tr>
<tr>
<td>200</td>
<td>24.4</td>
<td>77.2</td>
<td>50.4</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>300</td>
<td>27.3</td>
<td>164</td>
<td>75.7</td>
<td>12</td>
<td>32</td>
</tr>
<tr>
<td>400</td>
<td>16.0</td>
<td>201</td>
<td>117</td>
<td>17</td>
<td>37</td>
</tr>
</tbody>
</table>

The large number of track losses for update limits of 300 and 400 m makes it meaningless to raise the update limit further to see if a smaller number of radar measures can be accomplished. If one or two lost tracks out of 20 is acceptable or if the filter only tracks targets that perform turns around 5-g then this filter is a fully functioning filter. However, in comparison to the central filter this filter needs a lot more radar measurements to produce tracks of similar quality.

With an update limit of 50 m the number of radar measures is halved. Figure 5.9 presents further filter qualities for this update limit. It is clear that the consistency of the filter with this update limit is not as good as for the central filter. The consistency is not terrible and since the average position error is relatively small and the position uncertainty fairly well covers the position error the filter must be said to be reasonably well functioning. The extra top at 60 s in the consistency graph corresponds to some difficulties in the estimate of the targets velocity. Figure 5.9 suggests that the filter could be better tuned for the situation. It is clear that it is too slow to handle the 8-g turn properly.
Figure 5.10 presents a typical simulation output. In comparison to Figure 5.4 this filter uses the radar in a more regular manner, suggesting that it is not as well adjusted to the different situations that arises in the scenario, or simply that the update limit is so small that the uncertainty easily rises above it.

Figure 5.11 presents a typical simulation output for an update limit of 200 m. Here it is apparent that the radar use is more adjusted to the scenario. The filter has difficulties at the two sharp turns which it is too slow to track satisfactorily.

Figure 5.12 illustrates the Monte Carlo qualities of the filter. The top graph indicates that the filter is not consistent with this update limit. This means that the filter is not well adjusted to the scenario using this low number of radar measurements. The filter has obvious problems in estimating the position during the 5-g and 8-g turns, problems visible in
both Figure 5.11 and in the middle graph of Figure 5.12. Despite the fact that the filter is not ideally trimmed, it worked for the 20 simulations; the average position error is modest in comparison to the number of radar measurements performed and not a single track was lost. Although judging from the rest of table 5.2 the filter will probably lose a track or two if the number of simulations is increased and considering the poor consistency it is not for certain that the filter can be trusted with these parameter settings.

5.2.3 Different feedback rates for decentralized filter 1

The feedback rate affects the amount of communication that is required on the aircraft’s communication network. The effect of the feedback rate is interesting to investigate as a comparison between the different decentralized filters. Table 5.3 and Figure 5.13 illustrates the how the feedback period affects the number of required radar measurements for different update limits.

![Figure 5.12 Consistency, position and velocity deviation for the decentralized filter 1 with update limit 200 m over 20 MC runs.](image)

<table>
<thead>
<tr>
<th>Feedback period</th>
<th>1 s</th>
<th>2 s</th>
<th>3 s</th>
<th>4 s</th>
<th>5 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>30</td>
<td>195</td>
<td>197</td>
<td>197</td>
<td>198</td>
<td>198</td>
</tr>
<tr>
<td>50</td>
<td>100</td>
<td>104</td>
<td>103</td>
<td>103</td>
<td>103</td>
</tr>
<tr>
<td>100</td>
<td>74.9</td>
<td>83.3</td>
<td>84.1</td>
<td>86.2</td>
<td>83.8</td>
</tr>
<tr>
<td>150</td>
<td>67.6</td>
<td>51.7</td>
<td>58.0</td>
<td>61.5</td>
<td>62.2</td>
</tr>
<tr>
<td>200</td>
<td>77.2</td>
<td>44.1</td>
<td>48.7</td>
<td>49.4</td>
<td>56.4</td>
</tr>
<tr>
<td>300</td>
<td>27.3</td>
<td>38.1</td>
<td>40.0</td>
<td>38.8</td>
<td>43.0</td>
</tr>
<tr>
<td>400</td>
<td>16.0</td>
<td>18.3</td>
<td>35.3</td>
<td>33.0</td>
<td>41.5</td>
</tr>
</tbody>
</table>
5 SIMULATIONS

As expected the number of radar measurements generally increase with the feedback period. This reflects the fact that the radar filter can rely less on the supporting information of the IRST-filter. Naturally the IRST-filter gets less information from the radar filter as well. To compensate for the lesser amount of information the uncertainty is bigger and the radar is therefore forced to be used more frequently.

Figure 5.13 is an attempt to visualize table 5.3.

![Figure 5.13 A visualization of table 5.3](image.png)

Table 5.4 and Figure 5.14 illustrates the effect of different feedback periods on the average position error. Generally, the average position error does not seem to be dependent on the feedback rate. This means that the adaptivity works good. When the local filters have access to less information their uncertainty rises and as a consequence the radar sensor is used more frequently to maintain the desired track quality.

<table>
<thead>
<tr>
<th>Feedback period [s]</th>
<th>1 s</th>
<th>2 s</th>
<th>3 s</th>
<th>4 s</th>
<th>5 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>17.7</td>
<td>26.3</td>
<td>21.3</td>
<td>20.3</td>
<td>17.0</td>
</tr>
<tr>
<td>30</td>
<td>19.0</td>
<td>18.1</td>
<td>25.6</td>
<td>17.2</td>
<td>17.4</td>
</tr>
<tr>
<td>50</td>
<td>29.1</td>
<td>32.8</td>
<td>23.7</td>
<td>23.7</td>
<td>24.8</td>
</tr>
<tr>
<td>100</td>
<td>37.8</td>
<td>42.1</td>
<td>24.9</td>
<td>41.5</td>
<td>24.0</td>
</tr>
<tr>
<td>150</td>
<td>67.6</td>
<td>59.9</td>
<td>40.6</td>
<td>56.6</td>
<td>68.5</td>
</tr>
<tr>
<td>200</td>
<td>77.2</td>
<td>81.9</td>
<td>49.1</td>
<td>62.4</td>
<td>112</td>
</tr>
<tr>
<td>300</td>
<td>164</td>
<td>93.3</td>
<td>62.1</td>
<td>73.1</td>
<td>141</td>
</tr>
<tr>
<td>400</td>
<td>201</td>
<td>78.5</td>
<td>112</td>
<td>139</td>
<td>154</td>
</tr>
</tbody>
</table>

Table 5.4 - Position error [m]
Table 5.5 and Figure 5.15 presents how the number of lost tracks depends on the feedback rate.

Table 5.5 - Number of lost tracks

<table>
<thead>
<tr>
<th>Feedback period [s]</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>3</td>
<td>3</td>
<td>7</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>4</td>
<td>27</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>46</td>
<td>15</td>
</tr>
<tr>
<td>150</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>8</td>
<td>29</td>
</tr>
<tr>
<td>200</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>7</td>
<td>28</td>
</tr>
<tr>
<td>300</td>
<td>12</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>400</td>
<td>17</td>
<td>5</td>
<td>8</td>
<td>5</td>
<td>60</td>
</tr>
</tbody>
</table>

Table 5.5 illustrates that this filter structure does not handle long feedback periods very well. The number of lost tracks do generally increase with increased feedback periods and the filter can, for instance, not be run with a feedback period of 5 seconds. The big peaks in Figure 5.15 for feedback periods of 3 and 4 s probably makes these filter settings useless for any practical purposes. These peaks are not coincidental. Repeated simulations results in similar data. If a small number of track losses is acceptable the filter should have feedback every second or perhaps every other second and the update limit should be 200 m or less.
5.3 Decentralized filter 2

5.3.1 Regular radar use for decentralized filter 2

The filter in this chapter uses the radar, IRST-sensor and feedback every second. A typical simulation of this filter is seen in Figure 5.16.

The average position error for this filter over 20 Monte Carlo simulations is 14.9 m. This is a result that is between that of the central filter and the decentralized filter 1 which was 13.5 and 17.7 m respectively. The consistency, position and velocity error of this filter is shown in Figure 5.17 for 20 Monte Carlo simulations.

The consistency for this filter is excellent, it is better than both the other filters and is the only filter that manages to track the target through the 5-g turn consistently. The correspondence between the predicted uncertainty and the actual position error is also good. The predicted uncertainty is a bit low at the beginning of both the sharp turns but the
actual discrepancy is small, even less than 5 m. The estimation of the targets velocity is good. It is disturbed in the 8-g turn and it takes a rather long time before it is “back to normal”, but even during the turn the velocity error is fairly small.

5.3.2 Adaptive radar use for decentralized filter 2

This chapter presents the filter qualities when the radar is used adaptively. Feedback is performed every second. Table 5.6 presents data from these Monte Carlo simulations.

<table>
<thead>
<tr>
<th>Update limit [m]</th>
<th>No. of measures</th>
<th>Avg. pos. error [m]</th>
<th>Std of pos. error [m]</th>
<th>Lost tracks</th>
<th>MC runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>200</td>
<td>14.9</td>
<td>0.98</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>30</td>
<td>198</td>
<td>14.8</td>
<td>0.99</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>50</td>
<td>104</td>
<td>17.7</td>
<td>1.93</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>100</td>
<td>86.4</td>
<td>17.8</td>
<td>1.42</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>200</td>
<td>42.9</td>
<td>24.1</td>
<td>3.46</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>400</td>
<td>15.5</td>
<td>45.5</td>
<td>16.1</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>600</td>
<td>9.2</td>
<td>99.4</td>
<td>42.7</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>800</td>
<td>6.4</td>
<td>109</td>
<td>60.5</td>
<td>6</td>
<td>26</td>
</tr>
</tbody>
</table>

This filter performs better than the other decentralized filter with respect to radar adaptivity and is more stable to track losses. Therefore the update limit was increased until track losses occurred to see how small the number of radar measurements could be made. The predicted uncertainty is higher than for the other decentralized filter and therefore the
5 SIMULATIONS

update limit must be higher to achieve a low number of radar measurements. In a realistic situation an update limit as high as 800 m can probably not be allowed, as discussed in chapter 2.3.2.

The standard deviation of the position error is also small. The filter output is hence stable and not too dependent on the measurement noise realization. The standard deviation is very similar to that of the central filter. This is good and a related quality is the fact that this filter does not loose any tracks until the allowed uncertainty is very big and the number of radar measurements is quite low.

To see how well this filter is adjusted to the different parameter settings the consistency, position and velocity errors are presented in Figure 5.18 for update limit 50 m. Here the filter uses half the number of radar measurements of Figure 5.17

Figure 5.18 Consistency, position and velocity deviation for decentralized filter 2 with an update limit of 50 m over 20 MC runs.

Figure 5.19 A typical simulation with update limit of 50 m.

Figure 5.20 A typical simulation with update limit of 600 m.

The consistency is very good here as well. The filter now has a bigger problem with the 5-g turn than before, but not significantly. The consistency is better for this filter than for the
central filter for the same number of radar measurements. The predicted position uncertainty covers the actual error, but is perhaps generally too big to be ideal. The velocity error is relatively small; it increases during the two sharp turns and recovers slowly afterwards. A typical simulation with these settings looks like in Figure 5.19. The radar measurements are relatively evenly distributed throughout the flight path.

Figure 5.20 illustrates a typical simulation when the update limit is 600 m and the number of radar measurements is 9. The estimated track is very similar to that of the central track in Figure 5.5 which uses 8 radar measurements.

The consistency must be considered very good for these filter settings as can be seen in Figure 5.21, even a little better than for the central filter. From the middle graph it is clear that the predicted position uncertainty well covers the actual position error. The error in the velocity estimate is about twice as big as for the central filter, but it is still relatively small and clearly bounded.

5.3.3 Different feedback rates for decentralized filter 2

As for the decentralized filter 1 it is interesting to simulate the effects of a reduced feedback rate. Table 5.8 and Figure 5.22 illustrates how the number of radar measurements depends on the feedback period.
5 SIMULATIONS

Table 5.8 - Number of radar measurements

<table>
<thead>
<tr>
<th>Feedback period [s]</th>
<th>1 s</th>
<th>2 s</th>
<th>3 s</th>
<th>4 s</th>
<th>5 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update limit [m]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>30</td>
<td>198</td>
<td>197</td>
<td>197</td>
<td>198</td>
<td>198</td>
</tr>
<tr>
<td>50</td>
<td>105</td>
<td>103</td>
<td>103</td>
<td>105</td>
<td>103</td>
</tr>
<tr>
<td>100</td>
<td>86.4</td>
<td>84.3</td>
<td>72.5</td>
<td>77.0</td>
<td>75.8</td>
</tr>
<tr>
<td>200</td>
<td>42.3</td>
<td>53.2</td>
<td>52.5</td>
<td>54.6</td>
<td>52.9</td>
</tr>
<tr>
<td>400</td>
<td>15.5</td>
<td>21.9</td>
<td>29.8</td>
<td>34.0</td>
<td>36.3</td>
</tr>
<tr>
<td>600</td>
<td>9.3</td>
<td>12.4</td>
<td>18.8</td>
<td>22.3</td>
<td>24.3</td>
</tr>
<tr>
<td>800</td>
<td>6.4</td>
<td>8.5</td>
<td>11.4</td>
<td>14.3</td>
<td>19.1</td>
</tr>
</tbody>
</table>

The different feedback periods do not seem to affect the number of radar measurements for update limits of 100 m and less. With update limits of 200 m and higher the number of radar measurements are dependent upon the feedback period. The longer period the more number of measurements. In comparison to decentralized filter 1 this filter is able to use fewer radar measurements for all feedback periods.

Table 5.9 and Figure 5.23 presents the effect of the feedback rate on the average position error.
The average position errors are similar for feedback periods of 1, 2 and 3 s. The errors are generally bigger for feedback periods of 4 and 5 s, at least for higher update limits. For small update limits the position error is fairly independent of the feedback period, suggesting that the adaptivity of the radar works according to the discussion in chapter 5.2.3.

Table 5.9 - Position error [m]

<table>
<thead>
<tr>
<th>Feedback period [s]</th>
<th>1 s</th>
<th>2 s</th>
<th>3 s</th>
<th>4 s</th>
<th>5 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>14.9</td>
<td>17.2</td>
<td>17.5</td>
<td>21.1</td>
<td>19.5</td>
</tr>
<tr>
<td>30</td>
<td>14.8</td>
<td>15.6</td>
<td>15.7</td>
<td>21.1</td>
<td>18.5</td>
</tr>
<tr>
<td>50</td>
<td>17.7</td>
<td>35.2</td>
<td>22.8</td>
<td>60.6</td>
<td>25.8</td>
</tr>
<tr>
<td>100</td>
<td>17.8</td>
<td>21.4</td>
<td>57.7</td>
<td>105</td>
<td>62.2</td>
</tr>
<tr>
<td>200</td>
<td>24.0</td>
<td>34.5</td>
<td>66.8</td>
<td>134</td>
<td>122</td>
</tr>
<tr>
<td>400</td>
<td>45.5</td>
<td>62.1</td>
<td>45.1</td>
<td>117</td>
<td>190</td>
</tr>
<tr>
<td>600</td>
<td>99.4</td>
<td>75.1</td>
<td>133</td>
<td>88.2</td>
<td>167</td>
</tr>
<tr>
<td>800</td>
<td>109</td>
<td>111</td>
<td>103</td>
<td>172</td>
<td>150</td>
</tr>
</tbody>
</table>

The average position errors are similar for feedback periods of 1, 2 and 3 s. The errors are generally bigger for feedback periods of 4 and 5 s, at least for higher update limits. For small update limits the position error is fairly independent of the feedback period, suggesting that the adaptivity of the radar works according to the discussion in chapter 5.2.3.

Table 5.10 and Figure 5.24 illustrates the influence of the feedback period on the number of lost tracks. It is apparent that this filter is more stable with respect to successfully tracking targets when the feedback period increases. If the update limit is zero the filter manages to successfully track targets for all feedback periods between 1 and 5 s. If a small number of lost tracks is allowed then this filter using the radar every or every other second with any feedback period between 1 and 5 s works good. Using the radar about every 10 seconds with feedback periods between 1 and 3 s is then also acceptable.
5 SIMULATIONS

Table 5.10 - Number of lost tracks

<table>
<thead>
<tr>
<th>Feedback period [s]</th>
<th>1 s</th>
<th>2 s</th>
<th>3 s</th>
<th>4 s</th>
<th>5 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>200</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>400</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>19</td>
</tr>
<tr>
<td>600</td>
<td>0</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>800</td>
<td>6</td>
<td>21</td>
<td>4</td>
<td>16</td>
<td>11</td>
</tr>
</tbody>
</table>

Figure 5.24 A visualization of table 5.10

5.4 Alternatives of decentralized filter 1

As discussed in chapter 3.7 there are two alternatives to the decentralized filter type 1 (filter 1). The first alternative, called decentralized filter 1a, fuses and feeds back the model likelihoods. The second alternative, called decentralized filter 1b, skips the model mixing in the beginning of the IMM-cycle when feedback is performed.
5.4.1 Decentralized filter 1a

The simulation data from this filter type is presented shortly in tables 5.11-5.13.

### Table 5.11 - Number of radar measurements

<table>
<thead>
<tr>
<th>Feedback period limit</th>
<th>1 s</th>
<th>2 s</th>
<th>3 s</th>
<th>4 s</th>
<th>5 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>30</td>
<td>196</td>
<td>197</td>
<td>197</td>
<td>198</td>
<td>197</td>
</tr>
<tr>
<td>50</td>
<td>102</td>
<td>104</td>
<td>105</td>
<td>104</td>
<td>104</td>
</tr>
<tr>
<td>100</td>
<td>88.9</td>
<td>88.8</td>
<td>88.9</td>
<td>89.6</td>
<td>87.3</td>
</tr>
<tr>
<td>150</td>
<td>56.6</td>
<td>61.6</td>
<td>61.7</td>
<td>64.3</td>
<td>64.9</td>
</tr>
<tr>
<td>200</td>
<td>32.4</td>
<td>48.1</td>
<td>53.0</td>
<td>51.0</td>
<td>52.7</td>
</tr>
<tr>
<td>300</td>
<td>19.1</td>
<td>34.0</td>
<td>41.2</td>
<td>40.5</td>
<td>46.4</td>
</tr>
<tr>
<td>400</td>
<td>13.5</td>
<td>22.7</td>
<td>31.9</td>
<td>31.4</td>
<td>40.8</td>
</tr>
</tbody>
</table>

The difference in number of radar measurements between this filter and filter 1 is not enormous, as seen in table 5.11. This filter performs somewhat fewer radar measurements for a feedback period of 1 or 2 s, but somewhat more for feedback periods 3, 4 and 5 s. Table 5.12 presents the average position errors for this filter which are also better than for filter 1 for feedback periods of 1 or 2 s and about the same for feedback periods of 3-5 s.

### Table 5.12 - Position error [m]

<table>
<thead>
<tr>
<th>Feedback period limit</th>
<th>1 s</th>
<th>2 s</th>
<th>3 s</th>
<th>4 s</th>
<th>5 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>17.4</td>
<td>17.2</td>
<td>17.3</td>
<td>16.8</td>
<td>22.1</td>
</tr>
<tr>
<td>30</td>
<td>17.2</td>
<td>17.1</td>
<td>18.8</td>
<td>17.2</td>
<td>18.4</td>
</tr>
<tr>
<td>50</td>
<td>20.0</td>
<td>26.9</td>
<td>29.7</td>
<td>22.1</td>
<td>25.7</td>
</tr>
<tr>
<td>100</td>
<td>27.2</td>
<td>31.3</td>
<td>27.5</td>
<td>54.1</td>
<td>37.3</td>
</tr>
<tr>
<td>150</td>
<td>37.4</td>
<td>43.3</td>
<td>39.0</td>
<td>52.3</td>
<td>68.4</td>
</tr>
<tr>
<td>200</td>
<td>44.0</td>
<td>38.9</td>
<td>61.4</td>
<td>31.3</td>
<td>48.1</td>
</tr>
<tr>
<td>300</td>
<td>62.4</td>
<td>57.0</td>
<td>111</td>
<td>104</td>
<td>135</td>
</tr>
<tr>
<td>400</td>
<td>59.6</td>
<td>91.8</td>
<td>100</td>
<td>130</td>
<td>202</td>
</tr>
</tbody>
</table>

Also when considering the number of lost tracks in table 5.13 this filter performs better than filter 1 for feedback periods of 1 and 2 s. It is marginally better for a feedback period of 3 s. It is similar to filter 1 for periods of 4 and 5 s.

This filter clearly outperforms filter 1 for feedback periods of 1 and 2 s for all update limits. For feedback periods above 2 s these two filter types performs relatively equal. The fact that this filter does not lose tracks for a feedback period of 1 s, except for the highest update limit, suggest that this filter version is the version to use of the two.
5.4.2 Decentralized filter 1b

The simulation results of this filter are presented in tables 5.14-5.16.

Table 5.13 - Number of lost tracks

<table>
<thead>
<tr>
<th>Feedback period</th>
<th>1 s</th>
<th>2 s</th>
<th>3 s</th>
<th>4 s</th>
<th>5 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>0</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>50</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>41</td>
<td>12</td>
</tr>
<tr>
<td>150</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>16</td>
<td>28</td>
</tr>
<tr>
<td>200</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>300</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>28</td>
</tr>
<tr>
<td>400</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>9</td>
<td>36</td>
</tr>
</tbody>
</table>

Table 5.14 - Number of radar measurements

<table>
<thead>
<tr>
<th>Feedback period</th>
<th>1 s</th>
<th>2 s</th>
<th>3 s</th>
<th>4 s</th>
<th>5 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>30</td>
<td>195</td>
<td>194</td>
<td>191</td>
<td>190</td>
<td>185</td>
</tr>
<tr>
<td>50</td>
<td>100</td>
<td>103</td>
<td>103</td>
<td>106</td>
<td>107</td>
</tr>
<tr>
<td>100</td>
<td>73.9</td>
<td>76.6</td>
<td>77.2</td>
<td>80.5</td>
<td>84.3</td>
</tr>
<tr>
<td>150</td>
<td>37.1</td>
<td>51.8</td>
<td>66.8</td>
<td>59.0</td>
<td>63.0</td>
</tr>
<tr>
<td>200</td>
<td>28.4</td>
<td>46.1</td>
<td>59.8</td>
<td>54.8</td>
<td>59.4</td>
</tr>
<tr>
<td>300</td>
<td>24.9</td>
<td>25.1</td>
<td>41.7</td>
<td>52.6</td>
<td>48.5</td>
</tr>
<tr>
<td>400</td>
<td>15.7</td>
<td>18.5</td>
<td>33.2</td>
<td>45.5</td>
<td>43.0</td>
</tr>
</tbody>
</table>

The data in table 5.14 is virtually the same as the data for filter 1 in table 5.3.

With the exception of a few values the data in table 5.15 is worse than the data for filter 1 in table 5.4.

In table 5.16 the number of lost tracks is presented. This filter loses about the same number of tracks for a feedback period of 1 s. For a feedback period of 2 s there is a mysteriously big number of track losses for update limit 30 m. For feedback periods of 3 and 4 s the number of lost tracks decrease with an increase of the update limit which goes against all previous simulations. Interestingly, the longest feedback period has the smallest number of track losses. This goes against the very idea that this filter version was based on. Compared to filter 1 this filter was modified just at the time at which the feedback is made. When no feedback is performed this filter works like filter 1. The fact that this filter
performs better when the alteration is used less suggests that this filter structure is not good.

Table 5.15 - Position error [m]

<table>
<thead>
<tr>
<th>Feedback period limit</th>
<th>1 s</th>
<th>2 s</th>
<th>3 s</th>
<th>4 s</th>
<th>5 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>18.1</td>
<td>21.7</td>
<td>26.5</td>
<td>32.2</td>
<td>18.2</td>
</tr>
<tr>
<td>30</td>
<td>18.6</td>
<td>31.1</td>
<td>48.7</td>
<td>36.9</td>
<td>23.8</td>
</tr>
<tr>
<td>50</td>
<td>32.2</td>
<td>34.7</td>
<td>46.7</td>
<td>104.6</td>
<td>83.6</td>
</tr>
<tr>
<td>100</td>
<td>31.6</td>
<td>38.9</td>
<td>38.1</td>
<td>75.2</td>
<td>87.6</td>
</tr>
<tr>
<td>150</td>
<td>80.5</td>
<td>34.1</td>
<td>38.3</td>
<td>79.0</td>
<td>93.7</td>
</tr>
<tr>
<td>200</td>
<td>108</td>
<td>43.5</td>
<td>50.5</td>
<td>119</td>
<td>74.0</td>
</tr>
<tr>
<td>300</td>
<td>201</td>
<td>58.7</td>
<td>114</td>
<td>152</td>
<td>102</td>
</tr>
<tr>
<td>400</td>
<td>225</td>
<td>155</td>
<td>146</td>
<td>82.8</td>
<td>124</td>
</tr>
</tbody>
</table>

Table 5.16 - Number of lost tracks

<table>
<thead>
<tr>
<th>Feedback period limit</th>
<th>1 s</th>
<th>2 s</th>
<th>3 s</th>
<th>4 s</th>
<th>5 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
<td>9</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>4</td>
<td>30</td>
<td>5</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>100</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>150</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>200</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>300</td>
<td>14</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>400</td>
<td>18</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

5.5 Communication and computation burden

5.5.1 Communication burden

The comparison of the need for communication is theoretical and calculated per target.

For the central filter the communication is the sensor data that is sent to the central computer. With an ideal sensor model the amount of data is 3 floating point values (floats) per interval for the radar and 1 float for the IRST. The sensors report all their detections and if the sensors detect clutter the amount of information that needs to be sent on the network is bigger, but should still be much smaller than for the decentralized architectures.

For the decentralized filters sensor data is not sent on the communication network, but kept inside the local filters. Only tracks of established targets are sent between the different nodes in the architecture. The local filters initiate new targets, but this information is
5 SIMULATIONS

probably not communicated before a measurement is a confirmed target with an established state.

Filter 1 sends the state vectors and covariance matrices from the local filters to the central at every interval. The central state vector and covariance matrix are also sent to the local filters at every interval. However, an \( n \times n \) covariance matrix only contains \( (n \cdot (n + 1))/2 \) unique values since it is symmetrical. This means that the number of elements that has to be sent at every feedback interval is: \( 3 \cdot ((5 \cdot 6)/2 + 5) = 60 \) floats/interval. Filter 1a uses an additional \( 3 \cdot 3 \) floats/interval to communicate the model likelihoods. Filter 1b uses the same number of floats/interval that filter 1 uses.

Filter 2 sends 3 state vectors and 3 covariance matrices from and to each local filter. It also sends 9 times 3 floats of model likelihoods resulting in 207 floats/interval. Filter 2 needs to send 3.5 times as many floats per interval as the other filter. Consider the fact that filter 2 preforms equal to filter 1 when filter 2 uses feedback every third second and filter 1 uses feedback every second. Then the communication difference is almost reduced to zero.

This means that filter 2 can combine the advantage of its own higher tracking quality and the communication benefits of filter 1 by using a lower feedback rate when the load on the communication system must be low.

5.5.2 Computation burden

The comparison of computation burden is very approximate and based on one target. It is based on the number of matrix multiplications and inversions of the implemented Matlab\textsuperscript{TM} filters per filter cycle. Also considered is the number of calls to the Matlab\textsuperscript{TM} function ‘fmin’ which is quite time consuming. The result is presented in table 5.17. The ‘fmin’ function is the most and a matrix multiplication the least time consuming of the three column elements. Simple simulations using values that are similar to those in the filter simulations show that one matrix inversion performs about 1.6 times the calculations that a matrix multiplication uses. ‘fmin’ uses about 15 times as many calculations as a matrix multiplication. Taking this into account, a total burden variable is calculated and presented in the table.

<table>
<thead>
<tr>
<th></th>
<th>Matrix multiplications</th>
<th>Matrix inversions</th>
<th>‘fmin’</th>
<th>Total burden</th>
</tr>
</thead>
<tbody>
<tr>
<td>Central filter</td>
<td>147</td>
<td>12</td>
<td></td>
<td>166</td>
</tr>
<tr>
<td>Decentralized filter 1 - total</td>
<td>217</td>
<td>24</td>
<td>7</td>
<td>360</td>
</tr>
<tr>
<td>Decentralized filter 1 - central node</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>Decentralized filter 2 - total</td>
<td>186</td>
<td>12</td>
<td>3</td>
<td>250</td>
</tr>
<tr>
<td>Decentralized filter 2 - central node</td>
<td>19</td>
<td>9</td>
<td>3</td>
<td>78</td>
</tr>
</tbody>
</table>

It is clear that the central filter performs the least amount of computation and the decentralized filter 1 the most. However, a reason for the decentralized architecture was to ease the computational burden for the central computer and both the decentralized architectures manages to do that. Decentralized filter 1 reduces the amount of calculations signif-
icantly to only 14% of that of the central filter. Decentralized filter 2 halves the amount of required calculation for the central computer.

In the central filter sensor data from clutter and multiple targets are sent to the central computer and the central computer needs to perform data association and track initiation. Since these functions are not simulated in this report they are not considered in table 5.17. However, they are an additional burden for the central computer and a burden that is not negligible.

The decentralized filters perform data association and track initiation in the local filters. The central computer has to perform data association as well in a decentralized architecture, since it has to associate target tracks from different sensor filters correct with one another.

The central filter has to associate measurements to tracks. The central computer in a decentralized filter needs to associate tracks from the local filters with one another. It is most likely easier to associate different tracks than to associate noisy measurements to tracks. Including the fact that the central filter also needs to perform track initiations in the central computer, clutter and multiple targets are a bigger problem for the central filter than for the decentralized filters when considering the amount of calculations performed in the central computer.

5.6 Comments

It is difficult to draw any absolute conclusions regarding these filter structures since the evaluation is based on tests. The result depends heavily on the empirically chosen filter parameters. All filters can naturally be adjusted to perform better. The process of filter tuning is very time consuming and results in a compromise between filter performance and available time.

The consistency graphs of the central and decentralized filter 2 indicates that these filters are well tuned. This means that conclusions regarding these two filters probably can be considered valid, at least for a 2D-scenario.

The consistency and number of lost tracks for the decentralized filter 1 suggest that this filter is not that well adjusted. However, the amount of time spent on adjusting this filter was great; greater than for the other filters. Also this filter is intolerant to different parameter settings, while the other filter structures were more stable. It was difficult to even get the decentralized filter 1 to successfully track the target during the whole scenario.

This discussion suggests that the comparison of the central filter and decentralized filter 2 is valid although it is only based on simulations. The difficulties and the poor tracking result for decentralized filter 1 suggest that this filter structure is in fact inferior to the other filters.
5 SIMULATIONS
Conclusions

6.1 Filter comparisons
This chapter summarizes the results from chapter 5. Decentralized filter 1b was not an improvement of decentralized filter 1 and is not considered in this comparison.

Filter accuracy
The filters work well with respect to tracking accuracy. There are differences among them and table 6.1 makes an approximative comparison of their average position errors for different numbers of radar measurements. It is clear that the central filter is the most accurate filter and that the decentralized filter 2 outperforms decentralized filter 1, it is almost as good as the central filter. Decentralized filter 1a is more accurate than its original version, but it is still not as accurate as the other architectures.

<table>
<thead>
<tr>
<th>No. of measures</th>
<th>Central filter [m]</th>
<th>Decentralized filter 1 [m]</th>
<th>Decentralized filter 2 [m]</th>
<th>Decentralized filter 1a [m]</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>13.5</td>
<td>17.7</td>
<td>14.9</td>
<td>17.4</td>
</tr>
<tr>
<td>~100</td>
<td>15</td>
<td>29</td>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>~40</td>
<td>28</td>
<td>67</td>
<td>24</td>
<td>40</td>
</tr>
<tr>
<td>~10</td>
<td>88</td>
<td>-</td>
<td>99</td>
<td>60</td>
</tr>
</tbody>
</table>

Table 6.1

Adaptivity
The central filter is the filter that needs the least amount of radar measurements to maintain a good tracking accuracy. It manages to track a target virtually without loss of tracking accuracy when the number of radar measurements is halved. The fact that it manages to track the target with an average number of radar measurements of 3.8 is remarkable.

The decentralized filter 1 performs well with respect to adaptivity with a high feedback rate. It is able to lower the number of measurements to one eighth, before loosing too many tracks. That means that on average the filter only needs to perform a radar measurement every 8 seconds. In comparison to the other architectures this is not great, but it must be considered okay.
6 CONCLUSIONS

The decentralized filter 2 performs very well with respect to adaptivity. It can lower the number of radar measurements to one twentieth before losing any tracks. This is almost as good as for the central filter.

**Track losses**

The central filter does not lose any tracks in the simulations, which is good. Decentralized filter 1 loses tracks occasionally when the feedback rate is high and it loses many tracks when the feedback rate is low. This is perhaps the most negative quality of this filter architecture. Filter version 1a does not lose as many tracks as version 1. This is the most important difference between version 1 and 1a. Filter version 1a performs almost as good as decentralized filter 2 with respect to the number of lost tracks. The decentralized filter 2 is the decentralized filter that loses the fewest number of tracks.

Using full feedback in the decentralized filters the central filter and decentralized filters 1a and 2 manages to track the target without losing it, except for when using the highest update limits. For feedback periods between 1 and 3 s the decentralized filter 2 works good if the update limit is equal to or less than 400 m and a small number of track losses is acceptable. If the update limit is 50 m or less this filter type works with feedback periods as long as 5 s. Decentralized filter 1a can operate with a feedback period of 1 or 2 s without update limit restrictions if a small number of track losses is acceptable. Decentralized filter 1 should probably use feedback every second.

**Communication**

According to the discussion in chapter 5.5.1 the central filter needs very little communication. Decentralized filter 1 needs about ten times as much data communication even if the sensors would fail almost every other measurement. Filter version 1a needs a little more than version 1. Decentralized filter 2 needs 3.5 times as much communication as decentralized filter 1.

For the decentralized architectures the amount of data communication clearly affects the performance. The more communication the better the filter performs. The decentralized filter 2 performs approximately equally good using a feedback period of 3 s as the decentralized filter 1 using feedback every second. When the decentralized filter 2 uses a feedback period of 3 s it uses the same amount of communication as the decentralized filter 1 does using full feedback.

**Computation**

Table 5.17 in chapter 5.5.2 presents the computation burden for each filter structure. It is clear that both decentralized architectures do lower the calculation burden for the central computer. Decentralized filter 1 reduces the amount of calculations to 14% of that in the central filter. Filter version 1a performs a few more calculations, but basically it is similar to version 1. The decentralized filter 2 performs about half the calculations that are performed in the central filter.

When considering the total amount of computation performed in each architecture the decentralized filters performs more.

**Feedback rate**

The feedback period is limited by the fact that the filters loose tracks. As mentioned under Track losses the best filter with respect to longer feedback periods is the decentralized fil-
Decentralized filter 1a manages to deal with longer update periods better than version 1. Version 1 should probably use feedback every second.

6.2 Filter conclusions

The central filter is the best tracker of the proposed filter architectures. The decentralized filter 2 is nearly as good as the central filter. The decentralized filter 2 has an advantage over the central filter in that it performs less calculations in the central computer, but it requires much more data communication.

Decentralized filter 2 produces tracks of higher quality than the decentralized filter 1 does. Including the large number of track losses for decentralized filter 1 it can be said that the tracking quality for the decentralized filter 2 is much higher. Decentralized filter 1 has the advantage that is performs even less calculations in the central computer. It also requires less data communication than the decentralized filter 2 requires.

If possible, the decentralized filter 1 should communicate and fuse the local model likelihoods as does filter version 1a. This improves the filter quality. It also increases the amount of communication, but the increase in track quality is most likely worth the extra communication.

It should be noted that these conclusions are based on empirical testing and can not be guaranteed as general truths as discussed in chapter 5.6. The fact that the consistency is good for the filters, except for decentralized filter 1 with large update limits, suggests that the conclusions drawn here are valid.

6.3 Suggested improvements

The simulations used a two-dimensional scenario. In any real application the filter must track targets in three dimensions. Will the extra dimension strike differently upon the different architectures? It should be possible to draw more general conclusions based on 3D-simulations.

In this report only one set of state variables is used, but state variables can be chosen in many ways. Modified spherical coordinates are often used for an angle-only sensor as the IRST-sensor. The IRST-filter could also be an angle-only filter. These designs are created to prevent an angle-only sensor filter from diverging. The fact that the decentralized filter 2 manages to track the target with a very small number of radar measurements suggests that it is enough with an occasional radar measurement to prevent divergence in the angle-only sensor filter. On the other hand, it would be interesting to see if the number of track losses decreases with a different state vector in the IRST-filter.

A tracking filter should be able to deal with multiple targets. The filters in this report should be complemented with data association algorithms, such as the nearest neighbor, joint probabilistic data association or the multiple hypotheses tracking algorithms. If more than one target is present the large update limits used for adaptivity in this report are probably too big to be acceptable. The risk of assigning a radar measurement to the wrong target would be significant. The use of more realistic sensor models would also require data association, since one measurement could result in many targets or potential
6 CONCLUSIONS

targets. In a realistic situation sensor management would be required to maintain the
tracks.

The reason that the improvements above are not included in this report is that time is lim-
ited. Although the ideas presented here are not implemented, it does not seem likely that
they would change the conclusions that the central filter is the best filter and that decen-
tralized filter 2 is better than decentralized filter 1.
References


[5 and 12] could in Nov. 1999 be found on: http://www.rt.isy.liu.se/~fredrik
[14 and 15] as well as other material on covariance intersection, could in Nov. 1999 be

Matlab is a registered trademark of MathWorks Inc., 24 Prime Park Way, Natick, MA
01760, USA. http://www.mathworks.com
Appendix A - Notations

This appendix summarizes the notations used throughout the report.

Operators: (x can be a vector or matrix)
- \( \dot{x} \) The dot is for time derivative
- \( x' \) The ′ symbolizes the Jacobian
- \( x^T \) The T symbolizes transposition
- \( det(\cdot) \) The determinant of a matrix
- \( std(\cdot) \) The standard deviation of a vector
- \( E(\cdot) \) The expectation value
- \( min_{\alpha}\beta(\alpha) \) Minimize the function \( \beta \) with respect to the variable \( \alpha \)

Symbols: (x can be a vector or matrix)
- \( \hat{x} \) The hat symbolizes an estimate
- \( \bar{x} \) The line symbolizes a true value
- \( \tilde{x} \) The tilde symbolizes error
- \( x(t) \) Time-continuous
- \( x_k \) Time-discrete
- \( x_{k|k-1} \) One-step prediction
- \( x_{k|k} \) Updated vector or matrix

Scalars:
- \( r \) The distance measure from the radar
- \( \dot{r} \) The distance rate measure from the radar
- \( b_r \) The bearing measure from the radar
- \( b_{ir} \) The bearing measure from the infrared sensor
- \( \sigma_{r}^2 \) The distance measurement noise for the radar
- \( \sigma_{\dot{r}}^2 \) The distance rate measurement noise for the radar
- \( \sigma_{br}^2 \) The bearing measurement noise for the radar
- \( \sigma_{\dot{b}_r}^2 \) The bearing measurement noise for the infrared sensor
- \( v_i^i \) The process noise for model \( i \) in the tangential direction
- \( v_i^j \) The process noise for model \( i \) normal to the tangential direction
- \( \mu_{j}(k) \) The model likelihood for model \( j \) at discrete time \( k \) (ordered in a vector)
- \( \mu_{ij}(k) \) Element \((i,j)\) of the mixing matrix at discrete time \( k \) (ordered in a matrix)
- \( p_{ij} \) Transition likelihood from model \( j \) to model \( i \) (ordered in a matrix)
$g_j(k)$ The model probability for model $j$ at discrete time $k$

$\omega$ The weight used in covariance intersection data fusion

$\tilde{a}(k)$ Position error at discrete time $k$

$\bar{a}$ Mean position error

$RMSE(k)$ The root mean square error at discrete time $k$

$RMSE$ The time mean of the root mean square

$P_D$ Probability of detection

$P_{FA}$ Probability of false alarm

**Vectors and matrices:**

$x$ The time-continuous state vector

$y$ Measurement vector

$v$ System noise vector

$w$ Measurement noise vector

$f$ Model function

$h$ Measurement function

$A, B, C$ Time-continuous model matrices. $A$ is also the Jacobian of $f$.

$F, G, H$ Time-discrete model matrices

$P$ The covariance matrix of the state vector $x$

$Q$ The system noise matrix

$R$ The measurement noise matrix

$\epsilon$ The innovation vector

$\eta$ The normalized state error squared variable

$S$ The covariance matrix of the innovation vector

$K$ The Kalman gain matrix

**Abbreviations:**

DRE Defence Research Establishment (Försvarsmakten, FOA)

EKF Extended Kalman Filter

ESA Electronically Scanned Antenna

IMM Interactive Multiple Model

IRST InfraRed Search and Track

MC Monte Carlo

MSA Mechanically Scanned Antenna

RMSE Root Mean Square Error

SNR Signal to Noise Ratio

avg. average

est. estimate or estimation

pos. position

std standard deviation
The implemented central filter consist of the file `central.m`. This file uses `ekf.m` as a subroutine and it runs the file `measure.m` to create a measurement data vector. The file also needs access to trajectory data. The trajectory data used in these simulations has 3 cartesian position and 3 cartesian velocity variables. The data file with the trajectory data also needs to contain a variable T with the sample period. The function `measure.m` and trajectory data, `flightpath.mat`, are used by all filters.

```matlab
function [Xfilt, Y, radar_meas, P_pos_pred, pos_err, vel_err, eps] = central(limit)
% [Xfilt, Y, radar_meas, P_pos_pred, pos_err, vel_err, eps] = central(limit)
% % limit is the update limit (e.g. 0-400 m)
% % Xfilt is the filtered state
% % Y is the true trajectory
% % radar_meas is the position of each radar measurement
% % P_pos_pred is the predicted uncertainty measurement
% % pos_err is the position error
% % vel_err is the velocity error
% % eps is the normalized error squared variable

% Initiation
%%%%%%
% Create measurement data in cartesian/polar coordinates
load flightpath %load variables Y and T
meas_noise = [20 10 1e-2 1e-3]; %measurement noise
Z = measure(Y, meas_noise);
% Sensor variances
Wr = 40^2; % Radar distance variance
Wrr = 20^2; % Radar distance rate variance
Wb = 0.02^2; % Radar bearing variance
Wbir = 0.002^2; % IRST bearing variance
R = diag([Wr Wrr Wb Wbir]);
% Process noise
B = zeros(5,2); B(3,1) = 1; B(5,2) = 1;
Q1 = B * diag([0.000001 0.000002]) * B'; %process noise for model 1
Q2 = B * diag([0.01 0.02]) * B'; %process noise for model 2
Q3 = B * diag([1 2]) * B'; %process noise for model 3
AllQ = zeros(5*5, 3); %matrix with all Q-matrices
AllQ(:,1) = Q1(:);
AllQ(:,2) = Q2(:);
AllQ(:,3) = Q3(:);
% Matrices and variables
trans_prob = [0.90 0.05 0.05; 0.10 0.80 0.10; 0.01 0.09 0.90]; %Transition probabilities
X0 = [Y(1,1); Y(2,1); sqrt(Y(4,1)^2+Y(5,1)^2); atan2(Y(5,1),Y(4,1)); 0];
P0 = eye(5);
AllX = zeros(5,3); %matrix with all X-vectors
AllP = zeros(5*5,3); %matrix with all P-matrices
AllPP = AllP;
for m=1:3
    AllX(:,m) = X0;
    AllP(:,:,m) = P0(:,:,m);
end
model_prob = [0.2; 0.2; 0.6]; %initial model probabilities
prob = zeros(3,1);
P = zeros(5,5);
Qk = zeros(5,5);
Pest = zeros(5,5);
N = size(Y);
Xfilt = zeros(5, N(2));
```

### Appendix B - Matlab™ code for central filter

The implemented central filter consist of the file `central.m`. This file uses `ekf.m` as a subroutine and it runs the file `measure.m` to create a measurement data vector. The file also needs access to trajectory data. The trajectory data used in these simulations has 3 cartesian position and 3 cartesian velocity variables. The data file with the trajectory data also needs to contain a variable T with the sample period. The function `measure.m` and trajectory data, `flightpath.mat`, are used by all filters.
Xfilt(:,1) = X0;
radar_meas = [ ];
% vector to keep the radar measures in
pos_err = zeros(1,N(2));
vel_err = zeros(1,N(2));
eps = zeros(1,N(2));
P_pos_pred = zeros(1, N(2));
Ppred = zeros(5,5); % Matrices used for deciding if
M3_Xpred = zeros(5,3);
M3_Ppred = zeros(25,3);
Ppred_tot=zeros(25,N(2)); %

% Filter loop
% Filter loop
% Filter loop
for i=2:N(2)

% Calculate the mixingsmatrix
 model_prob_pred = trans_prob'*model_prob;
 mix_prob = trans_prob.*(model_prob*(model_prob_pred.^(-1))');
% Mix states
 X0hat = AllX*mix_prob;
 for k=1:3
    while X0hat(4,k) <= -pi X0hat(4,k) = X0hat(4,k) + 2*pi; end
    while X0hat(4,k) > pi X0hat(4,k) = X0hat(4,k) - 2*pi; end
    dx = AllX(:,k) - X0hat(:,k);
    while dx(4) <= -pi dx(4) = dx(4) + 2*pi; end
    while dx(4) > pi dx(4) = dx(4) - 2*pi; end
    dx2 = dx*dx';
    AllPP(:,k) = dx2(:);
 end
 AllP = (AllP + AllPP)*mix_prob;
% Should the radar be used?
 %-----------------------------
 %Predict all 3 models
 for m=1:3
    X = X0hat(:,m);
    P(:) = AllP(:,m);
    Qk(:) = AllQ(:,m);
    [X, P] = ekf(X, P , Qk, R, Z(:,i), T, 0);
    M3_Xpred(:,m) = X;
    M3_Ppred(:,m) = P(:);
 end
%Let the predictions interact
 Xpred = M3_Xpred*model_prob;
 while Xpred(4) <= -pi Xpred(4) = Xpred(4) + 2*pi; end
 while Xpred(4) > pi Xpred(4) = Xpred(4) - 2*pi; end
 for m=1:3
    x = M3_Xpred(:,m);
    x2 = x*x';
    Px(:,m) = x2(:);
 end
 Ppred(:) = (M3_Ppred + Px)*model_prob;
Ppred = Ppred - Xpred*Xpred';
Ppred_tot(:,i) = Ppred(:);
%Is the uncertainty big enough?
r = sqrt( Xpred(1,:)*Xpred(1,:) );
 dr = [Xpred(1,:)*Xpred(2,:);];
P_pos = dr'*Ppred*(1:2,1:2)*dr';
if P_pos > limit^2
    sensor = 1;
else
    sensor = 2;
end
if i == 2 sensor = 1; end
if sensor == 1
    radar_meas = [radar_meas [Z(1,i)*cos(Z(3,i)) Z(1,i)*sin(Z(3,i))]]; end
end
P_pos_pred(i) = sqrt(P_pos);
end
% Filter the 3 models
%-----------------------
for k=1:3
    % use the data from filter k
    Xest = Xhat(:,k);
    Pest(:) = AllP(:,k);
    Qk(:) = AllQ(:,k);
    while Xest(4) <= -pi Xest(4) = Xest(4) + 2*pi; end
    while Xest(4) > pi Xest(4) = Xest(4) - 2*pi; end
    % Filter
    [Xest, Pest, prob(k)] = ekf(Xest, Pest, Qk, R, Z(:,i), T, sensor);
    % store the data for filter k
    AllX(:,k) = Xest;
    AllP(:,k) = Pest(:);
end

% Update the model likelihoods
%------------------------------------
model_prob = model_prob_pred.*prob;
model_prob = model_prob/sum(model_prob);
if sign(AllX(4,1))~=sign(AllX(4,2)) | sign(AllX(4,1))~=sign(AllX(4,3)) | sign(AllX(4,2))~=sign(AllX(4,3))
    if model_prob(1) >= model_prob(2)
        AllX(4,2) = AllX(4,1);
    else
        AllX(4,1) = AllX(4,2);
    end
    if model_prob(2) >= model_prob(3)
        AllX(4,3) = AllX(4,2);
    else
        AllX(4,2) = AllX(4,3);
        AllX(4,1) = AllX(4,3);
    end
end

% Model interpolation
%-------------------------
Xest = AllX*model_prob;
while Xest(4) <= -pi Xest(4) = Xest(4) + 2*pi; end
while Xest(4) > pi Xest(4) = Xest(4) - 2*pi; end
for k=1:3
    dx = AllX(:,k) - Xest;
    while dx(4) <= -pi dx(4) = dx(4) + 2*pi; end
    while dx(4) > pi dx(4) = dx(4) - 2*pi; end
    dx2 = dx*dx';
    AllPP(:,k) = dx2(:);
end
P(:) = (AllP + AllPP)*model_prob;

% Create output data
%-----------------------
Xfilt(:,i) = Xest;
pos_err(i) = sqrt((Y(1,i)-Xfilt(1,i)).^2 + (Y(2,i)-Xfilt(2,i)).^2);
vel_err(i) = abs(Xfilt(3,i) - sqrt(Y(4,i).^2 + Y(5,i).^2));
% consistency
omega(i) = atan2(Y(5,i),Y(4,i)) - atan2(Y(5,i-1),Y(4,i-1));
if i>70 & i<126 omega(i)=omega(70);
end
if i>120 & i<126 omega(i)=omega(120);
end
X_err = [Y(1,i) Y(2,i) sqrt(Y(4,i).^2 + Y(5,i).^2) atan2(Y(5,i),Y(4,i)) omega(i)] - Xfilt(:,i);
while X_err(4) <= -pi X_err(4) = X_err(4) + 2*pi;
end
while X_err(4) > pi X_err(4) = X_err(4) - 2*pi;
end
eps(i) = X_err'*inv(Ppred)*X_err;
function [Xest, Pest, prob_i] = ekf(Xest,Pest,Qi,Ri,Zi,T,sensor)

%[Xest, Pest, prob_i] = ekf(Xest,Pest,Qi,Ri,Zi)
%
%This function is used by imm.m and performs one time step Kalman filtering
%
% Linearize around previous estimate
%---------------------------------------------------------------
A = zeros(5,5);
A(1,3:4) = [cos(Xest(4)) -Xest(3)*sin(Xest(4))];
A(2,3:4) = [sin(Xest(4)) Xest(3)*cos(Xest(4))];
A(4,5) = 1;
A2 = A * A;
F = eye(5) + A*T + A2*(T^2)/2;
G = eye(5)*T + A*(T^2)/2 + A2*(T^3)/6;
f = [Xest(3)*cos(Xest(4)); Xest(3)*sin(Xest(4)); 0; Xest(5); 0];
%
% Predict
%--------
Xpred = Xest + G*f;
Ppred = F*Pest*F’ + G*Qi*G’;
r = sqrt( Xpred(1)^2 + Xpred(2)^2 );
%
% Filter
%-------
if sensor==1 % radar and IRST

% Calc. the innovation
h = zeros(4,1);
h(1) = r;
h(2) = ( Xpred(1)*Xpred(3)*cos(Xpred(4)) + Xpred(2)*Xpred(3)*sin(Xpred(4)) ) / r;
h(3) = atan2(Xpred(2), Xpred(1));
h(4) = atan2(Xpred(2), Xpred(1));
inno = Zi - h;

% Estimate
s = Xpred(1)*Xpred(3)*cos(Xpred(4)) + Xpred(2)*Xpred(3)*sin(Xpred(4));
H = zeros(4,5);
H(1,1:2) = [Xpred(1)/r Xpred(2)/r];
H(2,1:2) = [ -Xpred(1)*s/r^3+Xpred(3)*cos(Xpred(4))/r -Xpred(2)*s/r^3+Xpred(3)*sin(Xpred(4))/r ];
H(2,3:4) = [Xpred(1)*cos(Xpred(4))+Xpred(2)*sin(Xpred(4))]/r; %->
H(3,1:2) = [ -Xpred(2)/(r^2) Xpred(1)/(r^2) ];
H(4,1:2) = [ -Xpred(2)/(r^2) Xpred(1)/(r^2) ];
S = H*Ppred*H’ + Ri;
K = Ppred*H’*(eye(4,4) + H*Ppred*H’)^-1;
Xest = Xpred + K*inno;
Pest = ( eye(5) - K*H’ ) * Ppred*( eye(5) - K*H’ )’ + K*Ri*K’;
else % only IRST

h = atan2(Xpred(2), Xpred(1));
inno = Zi(4,1) - h;
H = [Xpred(1)*r^2 Xpred(2)*r^2 Xpred(1) r^2 0 0 0];
S = eye(4) + H*Ppred*H’;
K = Ppred*H*S;
Xest = Xpred + K*inno;
Pest = ( eye(5) - K*H’ ) * Ppred*( eye(5) - K*H’ )’ + K*Ri(4,4)*K’;
end

% Correct estimat that are out of bounds
%-----------------------------------------------
if Xest(3)<100 Xest(3)=100; end
while (Xest(4) > pi)
    Xest(4) = Xest(4) - 2*pi;
end
while (Xest(4) <= -pi)
    Xest(4) = Xest(4) + 2*pi;
end
function [Z] = measure(traj, noise)

% [Z] = measure(traj, noise)
% Generates measurements, Z, given the true state in traj and noise.
% 
% Z(1,:) = radar distance
% Z(2,:) = radar distance rate
% Z(3,:) = radar bearing
% Z(4,:) = IRST bearing

N=size(traj);

% Noise
%--------
w1 = noise(1)*randn(1, N(2));
w2 = noise(2)*randn(1, N(2));
w3 = noise(3)*randn(1, N(2));
w4 = noise(4)*randn(1, N(2));

% True states
%-------------
x = traj(1,:);
y = traj(2,:);
v = sqrt( traj(4,:).*traj(4,:)+traj(5,:).*traj(5,:)) ;
phi = atan2( traj(5,:), traj(4,:)) ;
V(1,:) = sqrt( x.*x + y.*y);
V(2,:) = (x.*v.*cos(phi) + y.*v.*sin(phi))./sqrt(x.*x + y.*y);
V(3,:) = atan2(y,x);
V(4,:) = atan2(y,x);

% Measurements
%--------------
Z(1,:) = V(1,:) + w1;
Z(2,:) = V(2,:) + w2;
Z(3,:) = V(3,:) + w3;
Z(4,:) = V(4,:) + w4;
Appendix C  Matlab™ code for decentralized filter 1

This decentralized filter consists of the main file decentralized.m which runs the files localfilter.m and fusionfilter.m. localfilter.m uses the file ekf.m and fusionfilter.m uses the file covint.m. decentralized.m also uses measure.m, described in appendix B, and needs access to trajectory data.

function [Xfilt, Y, radar_meas, P_pos_pred, pos_err, vel_err, eps] = decentralized(limit, T_fb)
%
% [Xfilt, Y, radar_meas, P_pos_pred, pos_err, vel_err, eps] = decentralized(limit, T_fb)
%
%limit is the update limit (e.g. 0-400)
%T_fb is the feedback period
%Xfilt is the filtered state
%Y is the true state
%radar_meas is the position of the radar measurements
%P_pos_pred is the predicted uncertainty measurement
%pos_err is the position error
%vel_err is the velocity error
%eps is the normalized error squared variable
%
% Initiation
%%% %Create measurement data in cartesian/polar coordinates
load flightpath %load variables Y, T, trajectories
meas_noise = ([20 10 1e-2 1e-3]); %measurement noise
Z = measure(Y, meas_noise); %Variables, vectors and matrices
N = length(Y);
X0 = [Y(1,1); Y(2,1); sqrt(Y(4,1)^2 + Y(5,1)^2); atan2(Y(5,1),Y(4,1)); 0];
P0 = eye(5);
Xfilt = zeros(5, N);
Xfilt(:,1) = X0;
R_Xfilt = zeros(5, N);
IR_Xfilt = zeros(5, N);
Tot_Pest = P0;
Tot_Xest = X0;
R_Xest = X0;
R_Pest = P0;
IR_Xest = X0;
IR_Pest = P0;
R3_Pest = zeros(25,3);
R3_Xest = zeros(5,3);
IR3_Pest = zeros(25,3);
IR3_Xest = zeros(5,3);
for k=1:3
    R3_Pest(:,k) = P0(:,);
    R3_Xest(:,k) = X0;
    IR3_Pest(:,k) = P0(:,);
    IR3_Xest(:,k) = X0;
end
R_model_prob = [0.7; 0.2; 0.1];
IR_model_prob = [0.7; 0.2; 0.1];
RP = zeros(5,5);
RP = zeros(5,5);
do_meas = 1; %flag that indicates that the radar should be used
did_meas = 0; %flag that indicates if the radar was used
radar_meas = [[1]];
% Filter loop
% for i=2:N
  % Feedback
  % ------------------------
  if mod(i,T_fb) == 0
    for l=1:3
      IRP(:) = IR3_Pest(:,l);
      RP(:)  = R3_Pest(:,l);
      if sign(Tot_Xest(4)) ~= sign(R3_Xest(4,l))
        R3_Xest(4,l) = Tot_Xest(4);
      end
      if sign(Tot_Xest(4)) ~= sign(IR3_Xest(4,l))
        IR3_Xest(4,l) = Tot_Xest(4);
      end
      [IR3_Xest(:,l), IRP] = fusionfilter(IR3_Xest(:,l), IRP, Tot_Xest, Tot_Pest);
      [R3_Xest(:,l), RP] = fusionfilter(R3_Xest(:,l), RP, Tot_Xest, Tot_Pest);
      IR3_Pest(:,l) = IRP(:);
      R3_Pest(:,l)  = RP(:);
    end
  end
  did_meas = 0;
  if do_meas == 1
    radar_meas = [radar_meas [Z(1,i).*cos(Z(3,i)) Z(1,i).*sin(Z(3,i))]' ];
    did_meas = 1;
  end
  % Radar filter
  % ----------------------
  sensor = 1;
  [R_Xest, R_Pest, R3_Xest, R3_Pest, R_model_prob, do_meas, P_pos] = localfilter(R3_Xest, R3_Pest, R_model_prob, Z(:,i), T, sensor, do_meas, limit);
  % IRST filter
  % ----------------------
  sensor = 2;
  [IR_Xest, IR_Pest, IR3_Xest, IR3_Pest, IR_model_prob] = localfilter(IR3_Xest, IR3_Pest, IR_model_prob, Z(:,i), T, sensor, do_meas, limit);
  % Track fusion
  % ----------------------
  if sign(R_Xest(4)) ~= sign(IR_Xest(4))
    if did_meas == 1
      IR_Xest(4) = R_Xest(4);
    else
      R_Xest(4) = IR_Xest(4);
    end
  end
  [Tot_Xest, Tot_Pest] = fusionfilter(IR_Xest, IR_Pest, R_Xest, R_Pest);
  % Output data
  % ----------------------
  Xfilt(:,i) = Tot_Xest;
  P_pos_pred(i) = P_pos;
  % Consistency
  omega(i) = atan2(Y(5,i),Y(4,i)) - atan2(Y(5,i-1),Y(4,i-1));
  if i>70 & i<85
    omega(i) = omega(70);
  end
  if i>120 & i<126
    omega(i) = omega(120);
  end
  X_err = [Y(1,i) Y(2,i) sqrt( Y(4,i)^2+Y(5,i)^2 ), atan2(Y(5,i),Y(4,i)), omega(i)]' - Xfilt(:,i);
  while X_err(4) <= -pi
    X_err(4) = X_err(4) + 2*pi;
  end
  while X_err(4) > pi
    X_err(4) = X_err(4) - 2*pi;
  end
  eps(i) = X_err'*inv(Tot_Pest)*X_err;
pos_err = sqrt( (Xfilt(1,2:N) - Y(1,2:N)).^2 + (Xfilt(2,2:N) - Y(2,2:N)).^2 );
vel_err = abs( Xfilt(3,2:N) - sqrt( Y(4,2:N).^2 + Y(5,2:N).^2 ) );

function [Xest, Pest, M3_Xest, M3_Pest, model_prob, do_meas, P_pos] = localfilter(M3_Xest, M3_Pest, model_prob, Zi, T, sensor, do_meas, limit)

% IMM-filter used by decentral.m

% Definitions
% Sensor variance
if sensor == 1
    Wr = 40^2;
    Wrr = 20^2;
    Wb = 0.02^2;
    R = diag([Wr Wrr Wb]);
elseif sensor == 2
    Wir = 0.002^2;
    R = Wir;
end

% Process noise
B = [0 0; 0 0; 1 0; 0 0; 0 1];
Q1 = B*diag([0.000001 0.000002])*B';
Q2 = B*diag([0.02 0.04])*B';
Q3 = B*diag([2 5])*B';
AllQ = zeros(25,3);
AllQ(:,1) = Q1(:);
AllQ(:,2) = Q2(:);
AllQ(:,3) = Q3(:);
Q = zeros(5,5);

% Variables and matrices
trans_prob = [0.85 0.05 0.10;
              0.10 0.80 0.10;
              0.08 0.17 0.75];
prob = zeros(3,1);
Px = zeros(25,3);
Xest = zeros(5,1);
Pest = zeros(5,5);
P = zeros(5,5);
M3_Xpred = zeros(5,3);
M3_Ppred = zeros(25,3);
Ppred = zeros(5,5);

% Filtering
% Calc. model likelihoods
% Calc. model likelihoods
model_prob_pred = trans_prob' * model_prob;
max_prob = trans_prob' * model_prob' * (model_prob_pred.' - 1)';

% Mix states
X0hat = M3_Xest * mix_prob;
for n=1:3
    while X0hat(4,n) <= -pi X0hat(4,n) = X0hat(4,n) + 2*pi; end
    while X0hat(4,n) > pi X0hat(4,n) = X0hat(4,n) - 2*pi; end
    dx = M3_Xest(:,n) - X0hat(:,n);
    while dx(4) <= -pi dx(4) = dx(4) + 2*pi; end
    while dx(4) > pi dx(4) = dx(4) - 2*pi; end
    dx2 = dx'*dx;
Px(:,n) = dx2;
end
M3_Pest = (M3_Pest + Px) * mix_prob;
% Use radar?
% ----------------
if sensor == 1
    if do_meas == 1; sensor = 1;
    else sensor = 0; end
end
% Filter
% ---------
for n=1:3
    % use data for filter k
    Xest = X0hat(:,n);
    Pest(:) = M3_Pest(:,n);
    Q(:) = AllQ(:,n);
    % Filter
    [Xest, Pest, prob(n)] = ekf(Xest, Pest, Q, R, Zt, T, sensor, model_prob_pred(n));
    % store data for filter k
    M3_Xest(:,n) = Xest;
    M3_Pest(:,n) = Pest(:,n);
end
% Update model likelihoods
% -----------------------------
model_prob = model_prob_pred.*prob;
model_prob = model_prob/sum(model_prob);
if sign(M3_Xest(4,1))~=sign(M3_Xest(4,2)) | sign(M3_Xest(4,1))~=sign(M3_Xest(4,3))
    if model_prob(1) >= model_prob(2)
        M3_Xest(4,2) = M3_Xest(4,1);
    else
        M3_Xest(4,1) = M3_Xest(4,2);
    end
    if model_prob(2) >= model_prob(3) | model_prob(1) >= model_prob(3)
        M3_Xest(4,3) = M3_Xest(4,2);
    else
        M3_Xest(4,2) = M3_Xest(4,3);
        M3_Xest(4,1) = M3_Xest(4,3);
    end
end
% Model interpolation
% ---------------------
Xest = M3_Xest*model_prob;
while Xest(4) <= -pi Xest(4) = Xest(4) + 2*pi; end
while Xest(4) > pi Xest(4) = Xest(4) - 2*pi; end
for n=1:3
    dx = M3_Xest(:,n) - Xest;
    while dx(4) <= -pi dx(4) = dx(4) + 2*pi; end
    while dx(4) > pi dx(4) = dx(4) - 2*pi; end
    dx2 = dx*dx';
    Px(:,n) = dx2(:);
end
Pest(:) = (M3_Pest + Px)*model_prob;
% Should the radar be used next cycle?
% --------------------------------------
if sensor == 0 | sensor == 1
    % Predict each model
    for m=1:3
        X = M3_Xest(:,m);
        P(:) = M3_Pest(:,m);
        Q(:) = AllQ(:,m);
        [X, P] = ekf(X, P, Q, 0, 0, T, 0, 0);
        M3_Xpred(:,m) = X;
        M3_Ppred(:,m) = P(:,m);
    end
    % Let the predictions interact
    Xpred = M3_Xpred*model_prob;
    for m=1:3
        x = M3_Xpred(:,m);
        x2 = x*x';
        Px(:,m) = x2(:,);
    end
\[
P_{\text{pred}}(\cdot) = (M3_{\text{Ppred}} + P_x) \times \text{model\_prob}; \\
P_{\text{pred}} = P_{\text{pred}} - X_{\text{pred}} \times X_{\text{pred}}'; \\
\% \text{Is the uncertainty too high?} \\
r = \sqrt{(X_{\text{pred}}(1))^2 + (X_{\text{pred}}(2))^2}; \\
dr = \{X_{\text{pred}}(1)/r \times X_{\text{pred}}(2)/r\}; \\
P_{\text{pos}} = dr \times P_{\text{pred}}(1.2.1.2)/dr'; \\
\text{if} P_{\text{pos}} > \text{limit}^2 \\
\quad \text{do\_meas} = 1; \\
\text{else} \\
\quad \text{do\_meas} = 0; \\
\text{end} \\
P_{\text{pos}} = \sqrt{P_{\text{pos}}}; \\
\]

\text{function} \ [X_{\text{est}}, \ P_{\text{est}}, \ \text{prob}_i] = \text{ekf}(X_{\text{est}}, \ P_{\text{est}}, \ Q, \ R, \ Zi, \ T, \ \text{sensor}, \ \text{prob}) \\
\%
\% \text{This function is used by imm.m and performs one step Kalman filtering} \\
\%
\% \text{Linearize around previous estimate} \\
\%
\% \text{-----------------------------------------} \\
A = \text{zeros}(5,5); \\
A(1,3:4) = [\cos(X_{\text{est}}(4)) -X_{\text{est}}(3) \times \sin(X_{\text{est}}(4))]; \\
A(2,3:4) = [\sin(X_{\text{est}}(4)) \times X_{\text{est}}(3) \times \cos(X_{\text{est}}(4))]; \\
A(4,5) = 1; \\
A2 = A \times A; \\
F = \text{eye}(5) + A \times T + A2 \times (T^2)/2; \\
G = \text{eye}(5) \times T + A \times (T^2)/2 + A2 \times (T^3)/6; \\
f = [X_{\text{est}}(3) \times \cos(X_{\text{est}}(4)); \ X_{\text{est}}(3) \times \sin(X_{\text{est}}(4)); 0; X_{\text{est}}(5); 0]; \\
\%
\% \text{Predict} \\
\%
\% \text{-----------------------------------------} \\
X_{\text{pred}} = X_{\text{est}} + G \times f; \\
P_{\text{pred}} = F \times P_{\text{est}} \times F' + G \times Q \times G'; \\
r = \sqrt{(X_{\text{pred}}(1))^2 + (X_{\text{pred}}(2))^2}; \\
\%
\% \text{Estimate} \\
\%
\% \text{-----------------------------------------} \\
\% \text{RADAR} \\
\% \text{calc. innovation} \\
h = \text{zeros}(3,1); \\
h(1) = r; \\
h(2) = (X_{\text{pred}}(1) \times X_{\text{pred}}(3) \times \cos(X_{\text{pred}}(4)) + X_{\text{pred}}(2) \times X_{\text{pred}}(3) \times \sin(X_{\text{pred}}(4))) / r; \\
h(3) = \text{atan2}(X_{\text{pred}}(2), X_{\text{pred}}(1)); \\
inno = Zi(1:3,1) - h; \\
\% \text{estimate} \\
s = X_{\text{pred}}(1) \times X_{\text{pred}}(3) \times \cos(X_{\text{pred}}(4)) + X_{\text{pred}}(2) \times X_{\text{pred}}(3) \times \sin(X_{\text{pred}}(4)); \\
H = \text{zeros}(3,5); \\
H(1,1:2) = [X_{\text{pred}}(1:2) X_{\text{pred}}(3) \times \cos(X_{\text{pred}}(4)) \times T - X_{\text{pred}}(2) \times \sin(X_{\text{pred}}(4)) \times T]; \\
H(2,3:4) = [X_{\text{pred}}(1) \times \cos(X_{\text{pred}}(4)) \times T \times X_{\text{pred}}(3) \times \sin(X_{\text{pred}}(4)); \\
H(3,1:2) = [X_{\text{pred}}(2) \times r \times T \times X_{\text{pred}}(1) \times (r^2) / 2]; \\
\%
\% \text{IRST} \\
\% \text{calc. innovation} \\
h = \text{atan2}(X_{\text{pred}}(2), X_{\text{pred}}(1)); \\
inno = Zi(4,1) - h; \\
\%
\% \text{estimate} \\
H = [X_{\text{pred}}(2) \times T \times (r^2) \times X_{\text{pred}}(1) \times r \times T \times 2 \times 0 \times 0]; \\
S = H \times P_{\text{pred}} \times H' + R; \\
K = P_{\text{pred}} \times H' \times (S \times S^{-}); \\
X_{\text{est}} = X_{\text{pred}} + K \times inno; \\
P_{\text{est}} = (\text{eye}(5) - K \times H') \times P_{\text{pred}} \times (\text{eye}(5) - K \times H') + K \times R \times K'; \\
\text{elseif sensor == 2} \\
% \text{calc. innovation} \\
h = \text{atan2}(X_{\text{pred}}(2), X_{\text{pred}}(1)); \\
inno = Zi(4,1) - h; \\
% \text{estimate} \\
H = [X_{\text{pred}}(2) \times T \times (r^2) \times X_{\text{pred}}(1) \times r \times T \times 2 \times 0 \times 0]; \\
S = H \times P_{\text{pred}} \times H' + R; \\
K = P_{\text{pred}} \times H' \times (S \times S^{-});
\[ X_{est} = X_{pred} + K \cdot \text{inn}o; \]
\[ P_{est} = (\text{eye}(5) - K \cdot H)' \cdot P_{\text{pred}} \cdot (\text{eye}(5) - K \cdot H) + K \cdot R \cdot K'; \]
\[ S_{\text{inv}} = 1/S; \]

else
\[
\text{% PREDICTION} \\
X_{est} = X_{pred}; \\
P_{est} = P_{\text{pred}}; \\
\text{prob}_i = \text{prob}; \\
\]
end

\% Correct estimates that are out of bounds
\%-------------------------------------------------
if X_{est}(3)<200 X_{est}(3)=200; end
if X_{est}(3)>400 X_{est}(3)=400; end
while (X_{est}(4) > \text{pi})
X_{est}(4) = X_{est}(4) - 2*\text{pi};
end
while (X_{est}(4) <= -\text{pi})
X_{est}(4) = X_{est}(4) + 2*\text{pi};
end

\% calc model likelihoods
\%-------------------------------
if sensor==1 | sensor==2
\[
\text{norm} = \sqrt{\text{det}(2\pi S)}; \\
\text{exponent} = -(\text{inn}o' \cdot S_{\text{inv}} \cdot \text{inn}o)/2; \\
\text{prob}_i = 1/\text{norm} \cdot \exp(\text{exponent}); \\
\]
end

\textbf{function} \[ \text{Tot}_X_{est}, \text{Tot}_P_{est} = \text{fusionfilter}(A_{X_{est}}, A_{P_{est}}, B_{X_{est}}, B_{P_{est}}) \]
%\[ \text{Tot}_X_{est}, \text{Tot}_P_{est} = \text{fusionfilter}(A_{X_{est}}, A_{P_{est}}, B_{X_{est}}, B_{P_{est}}) \]
%\% Used by decentral.m to perform covariance intersection
%\%Find optimizing omega
%\%-------------------------------
\text{A}_{P_{est}}_{\text{inv}} = (A_{P_{est}})\text{eye}(5);
\text{B}_{P_{est}}_{\text{inv}} = (B_{P_{est}})\text{eye}(5);
w = \text{fmin('covint', 0, 1, [], \text{A}_{P_{est}}_{\text{inv}}, \text{B}_{P_{est}}_{\text{inv}}); }

%Fuse
%-------
\text{Tot}_P_{est} = (w*A_{P_{est}}_{\text{inv}} + (1-w)*B_{P_{est}}_{\text{inv}})\text{eye}(5);
\text{Tot}_X_{est} = \text{Tot}_P_{est}*(w*A_{P_{est}}_{\text{inv}}*A_{X_{est}} + (1-w)*B_{P_{est}}_{\text{inv}}*B_{X_{est}});

\textbf{function} a = covint(w, A, B)
%a = covint(w, A, B)
%\% Used by fusionfilter.m to calculate the weight w:
\text{a} = 1/\text{det}(w*A + (1-w)*B);
Also this decentralized filter consists of the main file decentralized.m which runs the files localfilter.m and fusionfilter.m. localfilter.m uses the file ekf.m and fusionfilter.m uses the file covint.m. decentralized.m also uses measure.m, described in appendix B, and needs access to trajectory data. The files ekf.m, fusionfilter.m and covint.m are described in appendix C.

function \([X_{filt}, Y, \text{radar}\_\text{meas}, \text{P\_pos\_pred}, \text{pos\_err}, \text{vel\_err}, \text{eps}] = \text{decentral}(\text{limit}, \text{T\_fb})\)

%function \([X_{filt}, Y, \text{radar}\_\text{meas}, \text{P\_pos\_pred}, \text{pos\_err}, \text{vel\_err}, \text{eps}] = \text{decentral}(\text{limit}, \text{T\_fb})\)

% limit is the update limit (e.g. 0-800)
% T\_fb is the feedback period
% X\_filt is the filtered state
% Y is the true state
% radar\_meas is the position of the radar measurements
% P\_pos\_pred is the predicted uncertainty measurement
% pos\_err is the position error
% vel\_err is the velocity error
% eps is the normalized error squared variable

% Initiation
%%%%%
% Create measurement data in cartesian/polar coordinates
load flightpath
meas\_noise = ([20 10 1e-2 1e-3]);
Z = measure(Y, meas\_noise);
N = length(Y);
% Sensor variances
Wr = 40^2;
Wrr = 20^2;
Wb = 0.02^2;
R\_R = diag([Wr Wrr Wb]);
Wr = 0.002^2;
IR\_R = Wr;
% Variables and matrices
model\_prob = [0.4 0.3 0.3]';
R\_model\_prob = model\_prob;
IR\_model\_prob = model\_prob;
trans\_prob = [0.85 0.05 0.10; 0.12 0.76 0.12; 0.08 0.12 0.80];
prob = zeros(3,1);
P_0 = zeros(25,3);
X0 = [Y(1,1); Y(2,1); sqrt(Y(4,1)^2 + Y(5,1)^2); atan2(Y(5,1),Y(4,1)); 0];
P_0 = diag([1 1 1 1 1]);
Xest = zeros(5,1);
Pest = zeros(5,5);
M3\_Xest = zeros(5,3);
M3\_Pest = zeros(25,3);
R3\_Xest = zeros(5,3);
R3\_Pest = zeros(25,3);
IR3\_Xest = zeros(5,3);
IR3\_Pest = zeros(25,3);
for j=1:3
M3\_Xest(:,j) = X0;
M3\_Pest(:,j) = P_0;
R3_Xest(:,j) = X0;
R3_Pest(:,j) = P0(:,j);
IR3_Xest(:,j) = X0;
IR3_Pest(:,j) = P0(:,j);
end
do_meas = 1;
did_meas = 0;
Xfilt = zeros(5,N);
Xfilt(:,1) = X0;
P_pos_pred = zeros(1,N);
radar_meas = [[1];[]];
did_break = 0;

% Filter loop
% %%%%%%%%%%%%%%%%%
for i=2:N

% Feedback
% %%%%%%%%%%%%%%%%%
if rem(i, T_fb/T) == 0
  R3_Xest = M3_Xest;
  R3_Pest = M3_Pest;
  IR3_Xest = M3_Xest;
  IR3_Pest = M3_Pest;
  R_prob = model_prob;
  IR_prob = model_prob;
else
  R_prob = R_model_prob;
  IR_prob = IR_model_prob;
end

% Radar
% %%%%%%%%%%%%%%%%%
did_meas = 0;
if do_meas == 1
  did_meas = 1;
sensor = 1;
  radar_meas = [radar_meas [Z(1,i)*cos(Z(3,i)) Z(1,i)*sin(Z(3,i))]];''
else sensor = 0; end
[R3_Xest, R3_Pest, R_model_prob, R_prob, P_pos, do_meas] = localfilter(R3_Xest, R3_Pest, R_R, Z(:,i), --->
  ---> T, sensor, R_prob, do_meas, limit);

% IRST
% %%%%%%%%%%%%%%%%%
sensor = 2;
[IR3_Xest, IR3_Pest, IR_model_prob, IR_prob] = localfilter(IR3_Xest, IR3_Pest, IR_R, IR_R, Z(:,i), T, sensor, --->
  ---> IR_prob, do_meas, limit);

% Fusion filter
% %%%%%%%%%%%%%%%%%
for n=1:3
  R_Xest = R3_Xest(:,n);
  R_Pest(:,n) = R3_Pest(:,n);
  IR_Xest = IR3_Xest(:,n);
  IR_Pest(:,n) = IR3_Pest(:,n);
  if sign(R_Xest(4)) ~= sign(IR_Xest(4))
    if did_meas == 1
      IR_Xest(4) = R_Xest(4);
    else
      R_Xest(4) = IR_Xest(4);
    end
  end
  [F_Xest, F_Pest] = fusionfilter(R_Xest, R_Pest, IR_Xest, IR_Pest);
  while F_Xest(4) <= -pi F_Xest(4) = F_Xest(4) + 2*pi; end
  while F_Xest(4) > pi F_Xest(4) = F_Xest(4) - 2*pi; end
  M3_Xest(:,n) = F_Xest;
  M3_Pest(:,n) = F_Pest(:,n);
end
% Update model likelihoods
%-------------------------------
model_prob_pred = trans_prob'*model_prob;
prob = (R_prob+IR_prob)/2;
model_prob = model_prob_pred.*prob;
model_prob = model_prob/sum(model_prob);

% Model interpolation
%------------------------
if sign(M3_Xest(4,1)) ~= sign(M3_Xest(4,2)) | sign(M3_Xest(4,1)) ~= sign(M3_Xest(4,3))
    if model_prob(1) >= model_prob(2)
        M3_Xest(4,2) = M3_Xest(4,1);
    else
        M3_Xest(4,1) = M3_Xest(4,2);
    end
    if model_prob(2) >= model_prob(3) | model_prob(1) >= model_prob(3)
        M3_Xest(4,3) = M3_Xest(4,2);
    else
        M3_Xest(4,2) = M3_Xest(4,3);
        M3_Xest(4,1) = M3_Xest(4,3);
    end
end

Xest = M3_Xest*model_prob;
while Xest(4) <= -pi Xest(4) = Xest(4) + 2*pi; end
while Xest(4) > pi Xest(4) = Xest(4) - 2*pi; end
for n=1:3
dx = M3_Xest(:,n) - Xest;
while dx(4) <= -pi dx(4) = dx(4) + 2*pi; end
while dx(4) > pi dx(4) = dx(4) - 2*pi; end
dx2 = dx*dx';
Px(:,n) = dx2(:);
end
Pest(:) = (M3_Pest + Px)*model_prob;

% Output data
%-------------
Xfilt(:,i) = Xest;
P_pos_pred(i) = P_pos;
% Consistency
omega(i) = atan2(Y(5,i),Y(4,i)) - atan2(Y(5,i-1),Y(4,i-1));
if i>70/T & i<85/T
    omega(i) = omega(70/T);
end
if i>120/T & i<126/T
    omega(i) = omega(120/T);
end
X_err = [Y(1,i) Y(2,i) sqrt( Y(4,i).^2 + Y(5,i).^2 ) atan2(Y(5,i), Y(4,i)) omega(i)] - Xfilt(:,i);
while X_err(4) <= -pi
    X_err(4) = X_err(4) + 2*pi;
end
while X_err(4) > pi
    X_err(4) = X_err(4) - 2*pi;
end
eps(i) = X_err'*inv(Pest)*X_err;

pos_err = sqrt( (Y(1,2:N)-Xfilt(1,2:N)).^2 + (Y(2,2:N)- Xfilt(2,2:N)).^2 );
vel_err = abs( Xfilt(3,2:N) - sqrt( Y(4,2:N).^2 + Y(5,2:N).^2 ) );
function [M3_Xest, M3_Pest, model_prob, prob, P_pos, do_meas] = localfilter(M3_Xest, M3_Pest, R, Zi, T, sensor, model_prob, do_meas, limit)

%M3_Xest, M3_Pest, model_prob, prob, P_pos, do_meas] = localfilter(M3_Xest, M3_Pest, R, Zi, T, sensor, model_prob, do_meas, limit)
%
% IMM-filter used by decentral.m
%
% Definitions
%\%\%\%\%\%\%
% Process noise
B = [0 0 0 0 0; 0 0 0 0 0];
Q1 = B*diag([0.000001 0.000002])*B';
Q2 = B*diag([0.002 0.004])*B';
Q3 = B*diag([0.5 1])*B';
AllQ = zeros(25,3);
AllQ(:,1) = Q1(:);
AllQ(:,2) = Q2(:);
AllQ(:,3) = Q3(:);

% Variables and matrices
Q = zeros(5,5);
P = zeros(5,5);
Px = zeros(25,3);
Xest = zeros(5,1);
Pest = zeros(5,5);
Xpred = zeros(5,1);
Ppred = zeros(5,5);
M3_Xpred = zeros(5,3);
M3_Ppred = zeros(25,3);
prob = zeros(3,1);
trans_prob = [0.85 0.05 0.10; 0.12 0.76 0.12; 0.08 0.12 0.80];

% Filter
%\%\%\%\%
% Calc. model likelihoods
%----------------------------
model_prob_pred = trans_prob'*model_prob;
mix_prob = trans_prob.*(model_prob*(model_prob_pred.^(-1))');

% Mix states
%-----------------
X0hat = M3_Xest*mix_prob;
for n=1:3
while X0hat(4,n) <= -pi X0hat(4,n) = X0hat(4,n) + 2*pi; end
while X0hat(4,n) > pi X0hat(4,n) = X0hat(4,n) - 2*pi; end
dx = M3_Xest(:,n) - X0hat(:,n);
while dx(4) <= -pi dx(4) = dx(4) + 2*pi; end
while dx(4) > pi dx(4) = dx(4) - 2*pi; end
dx2 = dx*dx';
Px(:,n) = dx2(:,n);
end
M3_Pest = (M3_Pest + Px)*mix_prob;

% Filter
%-------
for m=1:3
Xest = X0hat(:,m);
Pest(:,m) = M3_Pest(:,m);
Q(:,m) = AllQ(:,m);
[Xest, Pest, prob(m)] = ekf(Xest, Pest, Q(:,m), R, Zi, T, sensor, model_prob_pred(m));
M3_Xest(:,m) = Xest;
M3_Pest(:,m) = Pest(:,m);
end

% Update model likelihoods
%----------------------------
model_prob = model_prob_pred.*prob;
model_prob = model_prob/sum(model_prob);
Should the radar be used next cycle?
if sensor == 0 | sensor == 1
% Model interpolation
Xest = M3_Xest*model_prob;
while Xest(4) <= -pi Xest(4) = Xest(4) + 2*pi; end
while Xest(4) > pi Xest(4) = Xest(4) - 2*pi; end
for n=1:3
    dx = M3_Xest(:,n) - Xest;
    while dx(4) <= -pi dx(4) = dx(4) + 2*pi; end
    while dx(4) > pi dx(4) = dx(4) - 2*pi; end
    dx2 = dx*dx';
    Px(:,n) = dx2(:);
end
Pest(:) = (M3_Pest + Px)*model_prob;
% Predict each model
for m=1:3
    X = M3_Xest(:,m);
    P(:) = M3_Pest(:,m);
    Q(:) = AllQ(:,m);
    [X, P] = ekf(X, P, Q, 0, 0, T, 0, 0);
    M3_Xpred(:,m) = X;
    M3_Ppred(:,m) = P(:);
end
% Let the predictions interact
Xpred = M3_Xpred*model_prob;
while Xpred(4) <= -pi Xpred(4) = Xpred(4) + 2*pi; end
while Xpred(4) > pi Xpred(4) = Xpred(4) - 2*pi; end
for m=1:3
    x = M3_Xpred(:,m);
    x2 = x*x';
    Px(:,m) = x2(:);
end
Ppred(:) = (M3_Ppred + Px)*model_prob;
Ppred = Ppred - Xpred*Xpred';
% Is the uncertainty too high?
if P_pos > limit^2
    do_meas = 1;
else
    do_meas = 0;
end
else
    P_pos = 0;
end