NTNU – Trondheim
Norwegian University of
Science and Technology

Tracking A Person Using UWB Pulse Doppler Radar

Morten Åsheim

Master of Science in Electronics
Submission date: June 2015
Supervisor: Lars Magne Lundheim, IET
Co-supervisor: Jan Roar Pleym, Novelda As

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Abstract

There are today a large number of different tracking systems and the subject of tracking targets is not a new one. A lot of the tracking done is in connection with aviation and military uses, but as the "internet of things" has become more popular the possibility to expand to households has appeared. To be able to track people in buildings or your own house opens a lot of possibilities for different applications, for example in home security.

The usage of small radars, in the size of a few square centimeters, for tracking in households was an interesting topic for Novelda AS, which this thesis was written in collaboration with. The task was therefore to create a tracking system using Novelda's Xethru short-range radars, to track a walking person in a room.

A tracking system was developed to be used in said environment. The total system consisted of four main parts. The first part was the kalman filter. The kalman filter smoothed out noisy data and provided estimates of the range and speed of the target. Then the gating was implemented, allowing only measurements from the radar that was within a certain range from the predicted position. Then, in the case when there was received multiple measurements at the same time, the global nearest neighbour method chose the measurement closest to the predicted position. In the end, a triangulation function was made to give out a final position (x, y) using the two radars.

The thesis first gave a thorough description of the problem. Important parts of the tracking system was then explained and implemented. The system was then tested in simulations and later with real radar measurements. All the simulation and real radar measurements results were then presented in various plots and then discussed. Topics for further work were discussed and in the end a conclusion was drawn. It was concluded that the tracking system provided satisfactory results relative to the SNRs given. The simulation gave a good track estimation up to 5.66 m distance from origo, while the real radar measurements gave the same at 4 m. The author then concluded that the tracking system algorithm performed well, but not well enough for it to be used in any household tracking system with its current SNR values. The performance could be improved by following some of the suggestions mentioned in further work.

Acknowledgements

There are two people that has been of great help to me during the process of working with this thesis. I would like to thank Jan Roar Pleym, from Novelda AS, and my project supervisor Prof. Lars Lundheim, from the Norwegian University of Science and Technology (NTNU), for the guidance and help I have received throughout the whole project.

Contents

List of Figures

List of Tables

Dedicated to my family

Chapter 1

Introduction

1.1 Background

The task presented in thesis was made in collaboration with Novelda As, an up and coming Norwegian company that has created a high precision sensor technology. Common sensor technologies such as infrared, ultrasound and microwave, are mainly designed to perform a single task involving detection of either range, motion or presence. These sensors usually work under very limiting conditions, such as limited range, only moving or only stationary objects and so on. Each of the different sensors have their advantages and disadvantages, and ultimately one have to chose what feature is most important for the application.

Some applications need a combination of the different features given by the different sensor technologies, and it is here that more advanced radars come into play. Advanced impulse radars has the capabilities to measure even the most minute movements of objects in its vicinity. They can detect proximity, distance, presence and motion for stationary and moving targets, at different ranges. It can also see through objects and can therefore easily be hidden out of sight for security or esthetic purposes.

For a radar that is so small, powerful and precise, it is applicable in a vast number of different systems. For example, the radar is proposed to work as a tracking system. Tracking itself is not a new topic, but it has not been done using the Xethru ("see-through") radars yet. The radar is supposed to work in an indoor environment with low power consumption and without radiation that is dangerous to people. Therefore, it is both interesting for the author and Novelda to explore these possibilities further.

1.2 History

Radar and tracking using radar are not new concepts. They can be traced back to a handful of clever scientists in the epoch before and during the World Wars.

Often, the first person mentioned in this regard is the German physicist Heinrich Rudolf Hertz. In 1887, his experiment showed that electricity could be transmitted using electromagnetic waves. These waves would pass through some materials and would be reflected by others. He was the first person to generate and detect these signals later known as radio waves [1].

The next man to make a name for himself in radar technology, was the Italian Guglielmo Marconi. Inspired by Hertz's work, Marconi began building his own radio devices. In 1899, his radio signal broadcasts was received over the English channel. He presented the accumulation of his work i New York City in 1922, and that meeting is now referred to as the event that began the widespread of interest in radar development [2].

Building on the work done by Hertz and Marconi, the German engineer Christian Hülsmeyer was in 1904 able to demonstrate his invention which he called the "telemobiloscope". This invention was a transmitter-receiver system that was able to detect distant metallic objects as far as 3 km away by sending radio waves. It was designed to detect ships at sea to avoid collision. Mounted on a ship, its antenna would rotate and send out a signal. When a similar signal was received a bell would ring meaning a object was in that direction. The "telemobiloscope" was not able to detect the range, but Hülsmeyer solved this problem by using a triangulation system. The antenna would be mounted on a tower and scan the horizon. Knowing the height of the tower and the angle of the most intense return signal, the range could be calculated [3].

A lot of research and development of radar systems was done in the years leading up to WW2. The radar would prove its usefulness in the UK where the British were in need of a defense-system against air-raids from Germany. The British were outnumbered in the sense of the number of available fighter-planes compared to the Germans. To deal with this problem they figured that if they could have a system that would in advance notify the military about the raids, they could gather their forces and fend off the the attacks. This led up to the creation of the "Chain home" defense system.

The "Chain home" system [4] consisted of many 110 m tall steel towers, spaced out along the coastline of the UK. These towers flooded the airspace with radio frequency pulsed energy. Air crafts within the range of the towers would reflect the signals to a set of crossed dipole antennas. Using the time of travel of the reflected pulse and the angle calculated from the X-and Y- components of the crossed antennas, they were able

to pinpoint the position of the incoming air crafts. This gave the British air forces an much needed advantage over the Germans and it had a huge impact on the war, at least for the British.

In the years following WW2, the cold war came. In this period the plan was to use radars and tracking systems defensively against missiles and other threats. In 1955 Nelson Wax found a way to automatically predict aircraft's future positions using mathematical functions. Later, in 1964 Robert Sittler used Bayesian statistics to connect radar observations to existing targets. This work, in combination with the success of the Kalman filter, has enabled Yaakov Bar-Shalom and Robert Singer to develop the foundation of today's modern tracking systems[5].

In the later years, it has become more relevant to use short range radars, as the ones used in this thesis. For applications where there are people exposed, in home or medical situations, the short range ultra wide band (UWB) radar is a good choice since the radar pulses have such low energy that they are not harmful.

1.3 Thesis Structure

A short overview of the thesis structure is given

1. **Introduction**. A brief introduction about the background of the sensors used and the history behind radars.
2. **Problem Description**. A thorough description of the problem that is to be solved in the thesis.
3. **Tracking Filter**. Three different Kalman filter versions are introduced here, and how they are used in this thesis is explained.
4. **Methods for Data Association**. The principle of data association is explained. The gating and global nearest neighbour methods are then presented.
5. **Simulations**. The total simulation setup and all the necessary assumptions are explained. Then the actual results from the simulations are presented and discussed.
6. **True Radar Tracking**. The true radar setup with real radar measurements are shown and the results of the tracking are presented and discussed.
7. **Further Work**. Some thoughts about how to improve the tracking are made, and expansion possibilities are discussed.

8. **Conclusive Remarks**. A short conclusion at the end.

Chapter 2

Problem Description

2.1 General Task

The main problem is revolved around creating a functional two-sensor system and develop a tracking algorithm that uses this system to track a moving target, in this case a walking person. The target does not have to be a walking person, but the simulation values, such as speed, are based on this.

The problem focus of the thesis is centered around the output values from the radars and will not include changes to the actual radar hardware itself.

The tracking itself is a noisy operation and a filter will be needed.

The radars are stationary sensors in the sense that they do not rotate or give any information about where in the field of view the reflected signal arrives from. This is the reason why two sensors will be used to gain the actual position of the target using triangulation.

2.2 Description of Radar System

Small and energy saving Ultra Wide Band (UWB) radars are well suited for detecting objects at a few meters distance. They separate moving objects from stationary ones by distance- and Doppler-analysis. In surveillance and alarm applications it is important to follow position and speed of targets in the xy-plane. A Cartesian coordinate system will be used to represent the position as seen in figure 2.1. In this figure the walking target is seen from above in a bird's eye view.

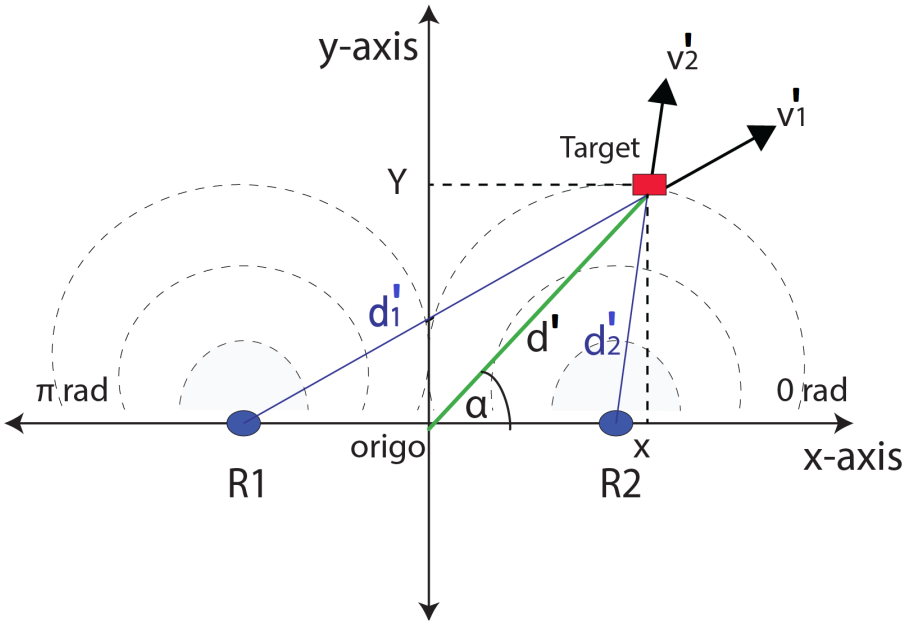


FIGURE 2.1: Cartesian coordinate system

The target is the red square and d' is the distance from origo to the target, represented by a green line. The target will never go below the x-axis since the radars do not have any vision there. The position of the target (x, y) will in the end always be given relative to the origo of the coordinate system, regardless of the position of the sensors used, represented in the figure as two blue circles on the x-axis. Upward will represent positive values on the y-axis and positive values on the x-axis will be to the right. The angle of the target α is 0 radians in the x-direction and increase counterclockwise to π.

The information of the target state (position and velocity) given by the radar sensors, R_1 and R_2, are in the form of range d_1 and d_2 and radial velocity v_1 and v_2, found in figure 2.1. Using these measurements, the Cartesian (x, y) position can be found with the use of trigonometry.

The radars used operate with spherical coordinates. Spherical coordinates means that the position is usually given by range, azimuth and elevation, but for the stationary radars used here the azimuth and elevation is not available for each of them alone. If the radar sends out a pulse to detect the target, the pulse propagates in the antenna direction until it is reflected back to the receiver antenna. There is no elevation or angle information here, only range and velocity. This means that the radar does not know from where in the radars vision the reflection came from. It is important to note that the radars still operates with a spherical system, because the range measurement can come from any point in the radars field of view, not just from a zero-elevation plane in front of the radar.

The key quantities, d_1', d_2', v_1' and v_2', have been introduced and figure 2.2 has been made to illustrate the total problem to be solved in this thesis. The difference between the notation d' and d is that the former is the actual distance from the target while the latter is the distance delivered by the radar. The same notation counts for v and v'. In general situations, where it does not matter whether it is radar $R1$ or $R2$ that is being talked about, d and v will be used instead of d_1, d_2, v_1 and v_2.

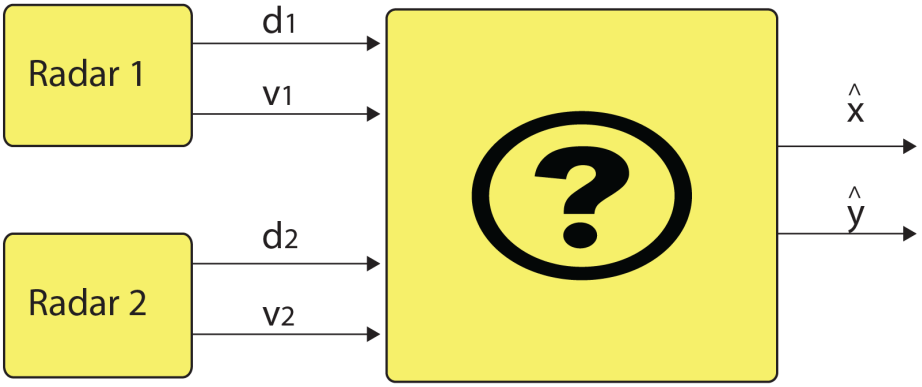


FIGURE 2.2: Problem overview

In figure 2.1 the d_1, d_2, v_1 and v_2 are shown. These measurements are the ones given by the radars and represents the starting point of the problem. These are the input of the question mark block in figure 2.2. The question mark represents all the processing and algorithms done to finally get the estimated outputs $\hat{x}$ and $\hat{y}$ which is the estimated Cartesian position of the target.

As portrayed here, the radars just give of one value each for d and v, but this is not always the case. There can be many potential values for d and v at each time iteration and it can be hard to know which of the values to chose. d_1 and v_1 can for example be as shown below

$$\boldsymbol{d_1} = [d_{1,1}, d_{1,2}, d_{1,2}, ..., d_{1,N}] \tag{2.1}$$

$$\boldsymbol{v_1} = [v_{1,1}, v_{1,2}, v_{1,2}, ..., v_{1,N}] \tag{2.2}$$

The same goes for d_2 and v_2. This will be addressed later in chapter 4.

Now that the starting point and the end product of the problem is defined, the solutions to the problem, residing in the question mark box, can be introduced. Things

like Kalman filters, gating, data association and triangulation will all be a part of the solution.

Chapter 3

Tracking Filter

The purpose of tracking filters is to to improve our estimate of what state the target is in [5]. This state is defined in this thesis as the actual position and velocity of the target. This will be the first step to eventually gain a good estimate $(\hat{x}, \hat{y})$, shown in figure 2.2, of the Cartesian (x, y) position of the target.

It is important to understand, before going through this chapter, that the different filters can operate on each of the radars separately with d_1 and v_1 in parallel with d_2 and v_2, and then using triangulation to find x, y, v_x and v_y, or do the triangulation first and then use the filters on the resulting d and α shown in figure 2.1. It is easier to explain the latter option first, and then simplify the filters to work with the first option after. Which option that will be implemented as a final solution will be discussed in a later chapter, but as the filters are explained here, no option is ruled out.

A Kalman filter, no matter what type[6], will work in the fashion shown in figure 3.1 below

FIGURE 3.1: Kalman general process

Here the vector $\mathbf{z}(k)$ contains the measurements from the radar, whether it is $\mathbf{z}(k) = [d(k), v(k)]^T$ before any triangulation or $\mathbf{z}(k) = [x(k), y(k), v_x(k), v_y(k)]^T$ after triangulation, and the state vector $\hat{\mathbf{x}}(k)$ contains the accompanying updated filter estimates of the measurements $\mathbf{z}(k)$ holds.

3.1 Model

When making measurements with any sensor, noise will be introduced. To negate that noise, many different versions of tracking filters has been developed. The filters all have in common that they need a model to describe the system they are working on[7]. The model explains how the sensor observations relates and translates to the actual state of the target (range and radial velocity). The model that will be used here is a discrete-time Markov process written out as

$$\mathbf{x}(k) = f[\mathbf{x}(k-1)] + \mathbf{w}(k) \tag{3.1}$$

$$\mathbf{z}(k) = h[\mathbf{x}(k)] + \mathbf{v}(k) \tag{3.2}$$

Equation 3.1 is a recurrence equation that describes the target dynamics using the actual target states in terms of a Markov process. In this case it is defined as the process where the current state $\mathbf{x}(k)$ is completely determined by the previous state $\mathbf{x}(k-1)$, and the process noise $\mathbf{w}(k)$. The process noise is denoted as $\mathbf{w}(k)$ and the measurement noise is $\mathbf{v}(k)$, more about the noise later in this chapter.

To keep track of which iteration of the radar measurements that is being worked on, the value k has been made. The first radar measurement taken into the Kalman filter will be the first iteration and represented as $k = 1$. $k-1$ represents the previous target iteration, while k represents the current one.

$h(\cdot)$ is called the measurement function, and describes the relationship between the measurements and the actual state. $f(\cdot)$ is called the state transition function and is assumed known. This function takes advantage of the physical aspect of the states. Using time, velocity and the laws of physics, the next state can be found. Note that there are many different versions of the $h(\cdot)$ and $f(\cdot)$ functions, corresponding to the different situations the kalman filter is used in.

Noise is added in order to make the model realistic. The process noise $\mathbf{w}(k)$ and measurement noise $\mathbf{v}(k)$ are added on in equation 3.1 and 3.2, respectively. The measurement noise $\mathbf{v}(k)$ is the uncertainty of the sensor. So the hardware and physical environment

determines the measurement noise and the statistical properties of the noise can sometimes be found by reading the sensor specifications. The process noise $\mathbf{w}(k)$ on the other hand, is not that easily found. It represents the error in the model itself and therefore the difference between the actual state and the state transformed by the state transition function, as seen in equation 3.1. The actual state is the true position and velocity of the target. This noise becomes more prominent when the target is moving in an uneven track because the transition function is based on linear lines.

Depending on the application, $\mathbf{z}(k)$ will vary and therefore the components in $\hat{\mathbf{x}}(k)$ and $\mathbf{x}(k)$ vary. For this thesis, where the tracking is done in the XY-plane, only the range and velocity are used. The acceleration is assumed low and not necessary for the tracking of low speed targets such as a person. So the state vector will consist of $\mathbf{x}(k) = [x(k), y(k), v_x(k), v_y(k)]^T$ or $\mathbf{x}(k) = [d(k), v(k)]^T$, depending on if the triangulation is done before or after the kalman filter, respectively. Measurements from the sensor are a combination of the systems state components and uncorrelated noise and is stored in the measurement vector $\mathbf{z}(k)$ found in equation 3.2. The model will be used in the Kalman filters shown next.

3.2 Kalman Filter

Even though the Kalman filter is over 50 years old, it is still a very important data fusion algorithm today. Data fusion is defined as the process of integrating multiple data of the same real-world object into an accurate and useful representation. The filter was created by Rudolph E. Kalman and found great success because of its simplicity and small computational complexity. The typical uses are smoothing out noisy data and giving estimates of important parameters. The filter is used in applications like global positioning system receivers, phase locked loops in radio equipment and tracking of objects, as in this thesis.

The Kalman filter is an algorithm which gives an estimate of the wanted parameters, and by learning from the previous iteration, decides how much to trust the estimated values compared to the measured values [8]. By doing this, it can update its estimate which is supposed to be better than the previous estimation. A more detailed explanation about how this works, will be gone through in the following chapters.

Before using the model in the kalman filters, some assumptions are made about the noise. If the noise is white Gaussian, the kalman filter minimizes the mean square errors of the position and velocity estimates. Both the process and measurement-noise, $w(k)$ and $v(k)$, needs to be white Gaussian. White noise is defined as noise with many frequencies

with equal intensities. If the noise is not Gaussian, the Kalman filter will still give the best linear estimator, but a nonlinear estimator would prove to be better. Because the noises $\mathbf{v}(k)$ and $\mathbf{w}(k)$ are white they are uncorrelated both with themselves and each other. This is written out as

$$\mathbf{w}(k) = \mathcal{N}(0, Q) \tag{3.3}$$

$$\mathbf{v}(k) = \mathcal{N}(0, R) \tag{3.4}$$

where R and Q are the matrices shown below and also used later in the filters. R and Q could change over the different iterations k, but for simplicity they are chosen to be constant through out all iterations. R and Q are first made with the assumption that the two sensors has been used in a triangulation function so that the state vector is $\mathbf{x}(k) = [x(k), y(k), v_x(k), v_y(k)]^T$. This gives the following

$$R = \begin{bmatrix} {\sigma_d}^2 & 0 \\ 0 & {\sigma_\alpha}^2 \end{bmatrix}$$

$$Q = \begin{bmatrix} {\sigma_x}^2 & 0 & 0 & 0 \\ 0 & {\sigma_y}^2 & 0 & 0 \\ 0 & 0 & {\sigma_{v_x}}^2 & 0 \\ 0 & 0 & 0 & {\sigma_{v_y}}^2 \end{bmatrix}$$

As shown above, in R, the variances of the noise in the distance d and angle α measurements shown in figure 2.1, are along the diagonal. In Q, the variances of the different states components can be found along the diagonal. Note that these two matrices will be subjects to change according to the type of Kalman filter being used, but also according which measurement components that are being used. As previously stated, if the filters are used on each of the sensors in parallel, before any triangulation, the state vector becomes $\mathbf{x}(k) = [d(k), v(k),]^T$ and

$$R = \begin{bmatrix} {\sigma_d}^2 \end{bmatrix}$$

$$Q = \begin{bmatrix} {\sigma_d}^2 & 0 \\ 0 & {\sigma_v}^2 \end{bmatrix}$$

In the following chapters, as the standard and extended kalman filters are introduced, the state vector is assumed to be $\mathbf{x}(k) = [x(k), y(k), v_x(k), v_y(k)]^T$.

3.3 Standard Kalman filter

As previously stated, the kalman filter is an optimal estimator which calculates parameters of interest using inaccurate and uncertain observations. The simplest version of it is the linear Kalman filter, henceforth called the standard Kalman filter. For the standard Kalman filter to work optimally, the model used must be linear. This is not necessarily the case for the model presented in chapter 3.1 because $f(\cdot)$ and $h(\cdot)$ can be non-linear. Therefore, some small alterations has to be made to assure linearity.

$$\mathbf{x}(k) = F\mathbf{x}(k-1) + \mathbf{w}(k) \tag{3.5}$$

$$\mathbf{z}(k) = G\mathbf{x}(k) + \mathbf{v}(k) \tag{3.6}$$

The nonlinear functions $f(\cdot)$ and $h(\cdot)$ has been replaced with the linear matrices F and G, respectively.

The Kalman filter algorithms works in a cyclic fashion. It consists of two main steps; the time update (prediction) step and measurement update (correction) step as shown in figure 3.2.

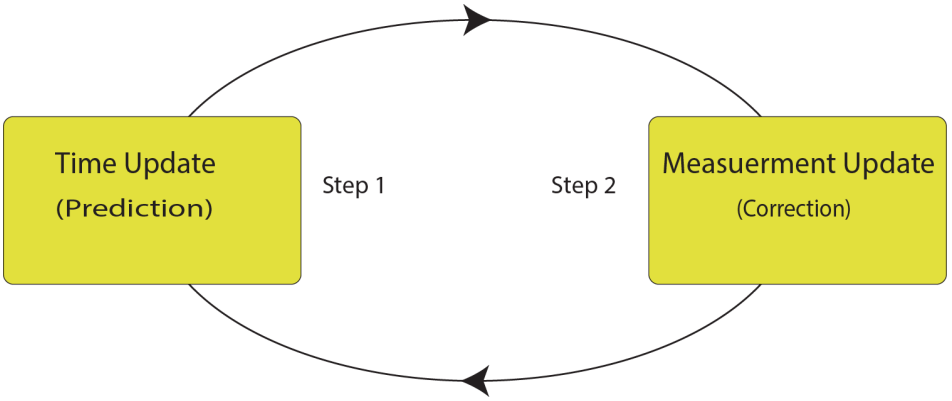


FIGURE 3.2: Kalman Cycle

Before going into the cycle, a few definitions are made. A is just a dummy character used for exemplification. $\overline{A}$ refers to a prediction, $\hat{A}$ is a updated estimation, $\tilde{A}$ is a error variable, A^T means a transposed matrix and A^{-1} is a inverse matrix. Now that the definitions are in order, the cycle can be explained step by step starting of with the prediction part.

To make the next few equations easier to understand, a block diagram of the Kalman filter process has also been made. The most important aspects are shown in this figure

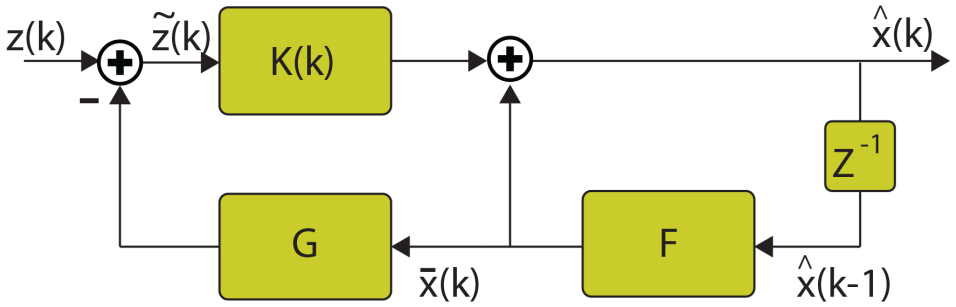


FIGURE 3.3: Kalman Block Diagram

3.3.1 Time Update (Prediction)

The first equation in the time update uses the system model to predict the future state of the system in the following way

$$\overline{\mathbf{x}}(k) = F \cdot \hat{\mathbf{x}}(k-1) \tag{3.7}$$

Here the $\overline{x}(k)$ was predicted using the previous state and the transition matrix F. F is created using laws of physical motion that connects the previous state $\mathbf{x}(k-1)$ to the next state $\mathbf{x}(k)$. In this case, with the chosen state components (range and velocity) and F equation 3.7 looks like this

$$\overline{\mathbf{x}}(k) = \begin{bmatrix} 1 & 0 & \Delta T & 0 \\ 0 & 1 & 0 & \Delta T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix} = \begin{bmatrix} x + v_x \Delta T \\ y + v_y \Delta T \\ v_x \\ v_y \end{bmatrix}$$

where ΔT is the time between samples and F is a $m \times m$ matrix where m is the length of the state vector $\mathbf{x}(k)$.

The second equation of the prediction step is a bit more complicated. It is where the covariance matrix $P(k)$ is calculated, which also has to be predicted for the next estimate. This matrix tells the algorithm how much to trust the current prediction and is written out as follows

$$\overline{P}(k) = F\overline{P}(k-1)F^T + Q \tag{3.8}$$

Although equation 3.8 shows the way $P(k)$ is used, it can be easier to understand using its definition

$$P(k) = E[(\mathbf{x}(k) - \hat{\mathbf{x}}(k))(\mathbf{x}(k) - \hat{\mathbf{x}}(k))^T] \tag{3.9}$$

The information of the covariance matrix lies in the diagonal. The diagonal will contain the variances for all the state components from the estimated $\hat{\mathbf{x}}(k)$. Which means the variance of x, y, v_x and v_y, in that order. In the end the process noise matrix Q is added. Q is a difficult matrix to calculate, and it is supposed to take account for the uncertainties in the model. The Q can therefore affect the whole filter to a great extent and is often used as a tuning tool. In practice, Q might change for each iteration, but it is assumed to be constant, shown on page 12.

3.3.2 Measurement Update (Correction)

The correction part of the algorithm is, in terms of computation, the bigger part. It starts with

$$\tilde{\mathbf{z}}(k) = \mathbf{z}(k) - G\overline{\mathbf{x}}(k) \tag{3.10}$$

where $\tilde{\mathbf{z}}(k)$ is the error between the observation $\mathbf{z}(k)$ from the sensor and the predicted $G\overline{\mathbf{x}}(k)$. The measurements from the sensor $\mathbf{z}(k)$ has to be in the same format as the state components in $\overline{\mathbf{x}}(k)$. Since the sensor uses spherical coordinates, the $\mathbf{z}(k)$ values has to be transformed into Cartesian coordinates before they are used in the filter and the G matrix has to transfer only the position states from $\overline{\mathbf{x}}(k)$. This makes sure all the values are in the same format.

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

G is a $h \times m$ matrix where h is the dimension of $\mathbf{z}(k)$ and m is the dimension of $\mathbf{x}(k)$.

The next thing to find, much like the $P(k)$, is a measure for how accurate the measurements from the sensor are. This comes in the form of $S(k)$

$$S(k) = G\overline{P}(k)G(k)^T + R \tag{3.11}$$

Again, as for $P(k)$, it can help to see the definition for a better understanding of $S(k)$

$$S(k) = E[(\mathbf{z}(k) - G\overline{\mathbf{x}}(k))(\mathbf{z}(k) - G\overline{\mathbf{x}}(k))^T] \tag{3.12}$$

$S(k)$ is called the innovation covariance matrix and it holds the information about how much to trust the measured values $\mathbf{z}(k)$ compared to the model. The measurement noise matrix R is also added on here which means that R can have a big impact on how much the algorithm trusts the measurements.

Both $P(k)$ and $S(k)$ will now be combined and used in what is called the Kalman gain.

$$K(k) = \overline{P}(k)G^T S(k)^{-1} \tag{3.13}$$

It is in the Kalman gain $K(k)$ where the the decision of how much to weight the measurement compared to the predicted estimate is taken, and it is the heart of the algorithm. The bigger the values in the $K(k)$ matrix becomes the more the measurements will be weighted compared to the model as seen in the next equation.

$$\hat{\mathbf{x}}(k) = \overline{\mathbf{x}}(k) + K(k)\tilde{\mathbf{z}}(k) \tag{3.14}$$

$$\hat{P}(k) = \overline{P}(k) - K(k)S(k)K(k)^T \tag{3.15}$$

These last two equations are the corrected updates of the system. $\hat{\mathbf{x}}(k)$ and $\hat{P}(k)$ will be the improved state estimate and the improved covariance , respectively, and used as an input for the next cycle of the algorithm. If everything works properly the result will be a state estimate with reduced model- and measurement-noise, which is the whole purpose of the Kalman filter.

To sum all this up in an orderly fashion, the pseudo code is given below. The pseudo code will work as the cycle figure 3.2 illustrates.

TABLE 3.1: **Standard Kalman filter Algorithm**

Time Update (Prediction)

$$\overline{\mathbf{x}}(k) = F \cdot \hat{\mathbf{x}}(k-1)$$
$$\overline{P}(k) = F\hat{P}(k-1)F(k)^T + Q$$

Measurement Update (Correction)

$$\tilde{\mathbf{z}}(k) = \mathbf{z}(k) - G\overline{\mathbf{x}}(k)$$
$$S(k) = G\overline{P}(k)G^T + R$$
$$K(k) = \overline{P}(k)G^T S(k)^{-1}$$
$$\hat{\mathbf{x}}(k) = \overline{\mathbf{x}}(k) + K(k)\tilde{\mathbf{z}}(k)$$
$$\hat{P}(k) = \overline{P}(k) - K(k)S(k)K(k)^T$$

New Cycle...

Using only one radar at the time makes the vector $\mathbf{x}(k) = [d(k), v(k)]^T$. Which is a simpler form of the standard Kalman filter that has been described above. Some of the matrices has to be scaled down to match the matrix dimension, but the filter works in the same way.

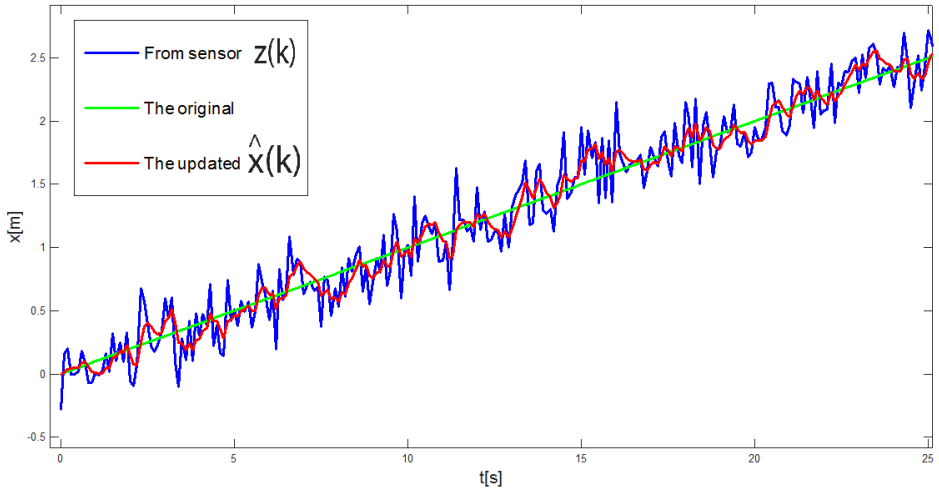


FIGURE 3.4: Standard Kalman, 1 radar example

The green straight line in figure 3.4 is the path that has been simulated and works as the original path. The blue graph is the noisy measurements from the sensor $\mathbf{z}(k)$ and the red graph is the Kalman filtered result $\hat{\mathbf{x}}(k)$. On the x-axis the discrete time is

multiplied with the sample interval ΔT and on the y-axis the distance from the radar is shown in meters. As seen in figure 3.4 there has been a decent improvement the red graph compared to the blue one.

3.4 Extended Kalman filter

As previously mentioned, the radar requires its measurements in spherical coordinates and because the coordinate system used requires Cartesian coordinates, the model is nonlinear[9]. The standard Kalman filteris optimal if the system is linear, but in a non-linear environment the extended Kalman filter may perform better, and it will therefore be implemented next.

The extended Kalman filter is very similar to the standard Kalman filter, but there are a few important differences. The measurement function $h(\cdot)$ becomes unlinear, but the transition function $f(\cdot)$ stays linear. This results in a new model where $F(k)$ is still the same matrix, but $h(\mathbf{x}(k))$ needs a new definition.

$$\mathbf{x}(k) = F\mathbf{x}(k-1) + \mathbf{w}(k) \tag{3.16}$$

$$\mathbf{z}(k) = h(\mathbf{x}(k)) + \mathbf{v}(k) \tag{3.17}$$

The cycle goes on in the same way as show before, but the first part of the measurement update (correction) is different from equation 3.10 and is now written as

$$\tilde{\mathbf{z}}(k) = \mathbf{z}(k) - h[\overline{\mathbf{x}}(k)] \tag{3.18}$$

The function $h(\cdot)$ is supposed to transform the Cartesian estimates into spherical values, so that the error between measurement and state can be estimated. Using the following spherical transformation rules this can be done.

$$\mathbf{z}(k) = h[\overline{\mathbf{x}}(k)] = \begin{bmatrix} d \\ \alpha \end{bmatrix} = \begin{bmatrix} h_d[\overline{\mathbf{x}}(k)] \\ h_\alpha[\overline{\mathbf{x}}(k)] \end{bmatrix}$$

$$h_d[\overline{\mathbf{x}}(k)] = d = \left[\sqrt{x(k)^2 + y(k)^2}\right]$$

$$h_\alpha[\overline{\mathbf{x}}(k)] = \alpha = \left[\arctan\left(\frac{y(k)}{x(k)}\right)\right]$$

where d and α refers to the values presented in figure 2.1. The error $\tilde{\mathbf{z}}(k)$ can now be estimated, but there are still places in the algorithm where the outdated matrix G is used. A new matrix that operates with spherical coordinates has to replace G. This will be the Jacobian linearized measurement matrix $H(k)$ and it is defined as

$$H(k) = \left.\frac{\partial h}{\partial \mathbf{x}}\right|_{\overline{\mathbf{x}}} \tag{3.19}$$

The $H(k)$ is a $h \times m$ matrix, where h and m is the dimension of $\mathbf{z}(k)$ and $\mathbf{x}(k)$, respectively. With $\mathbf{x}(k) = [x(k), y(k), v_x(k), v_y(k)]$ and $\mathbf{z}(k) = [d(k); \alpha(k)]$, $H(k)$ will end up looking like this

$$H(k) = \begin{bmatrix} \frac{\partial d(k)}{\partial x(k)} & \frac{\partial d(k)}{\partial y(k)} & \frac{\partial d(k)}{\partial v_x(k)} & \frac{\partial d(k)}{\partial v_y(k)} \\ \frac{\partial \alpha(k)}{\partial x(k)} & \frac{\partial \alpha(k)}{\partial y(k)} & \frac{\partial \alpha(k)}{\partial v_x(k)} & \frac{\partial \alpha(k)}{\partial v_y(k)} \end{bmatrix}_{\mathbf{x}=\overline{\mathbf{x}}(k)} = \begin{bmatrix} \frac{\overline{x}(k)}{\overline{d}(k)} & \frac{\overline{y}(k)}{\overline{d}(k)} & 0 & 0 \\ \frac{-\overline{y}(k)}{\overline{d}(k)^2} & \frac{\overline{x}(k)}{\overline{d}(k)^2} & 0 & 0 \end{bmatrix}$$

With these alterations the algorithm changes as seen in the next table.

TABLE 3.2: **Extended Kalman filter Algorithm**

Time Update (Prediction)	
	$\overline{\mathbf{x}}(k) = F \cdot \hat{\mathbf{x}}(k-1)$
	$\overline{P}(k) = F\hat{P}(k-1)F^T + Q$
Measurement Update (Correction)	
	$\tilde{\mathbf{z}}(k) = \mathbf{z}(k) - H(k)\overline{\mathbf{x}}(k)$
	$S(k) = H(k)\overline{P}(k)H(k)^T + R$
	$K(k) = \overline{P}(k)H(k)^T S(k)^{-1}$
	$\hat{\mathbf{x}}(k) = \overline{\mathbf{x}}(k) + K(k)\tilde{\mathbf{z}}(k)$
	$\hat{P}(k) = \overline{P}(k) - K(k)S(k)K(k)^T$
New Cycle...	

It is also possible with the extended Kalman filter to use $\mathbf{x}(k) = [d(k), v(k)]^T$ when using measurements from only one radar.

As a proof of concept, as for the Standard Kalman filter, the extended Kalman filter produced the following figure

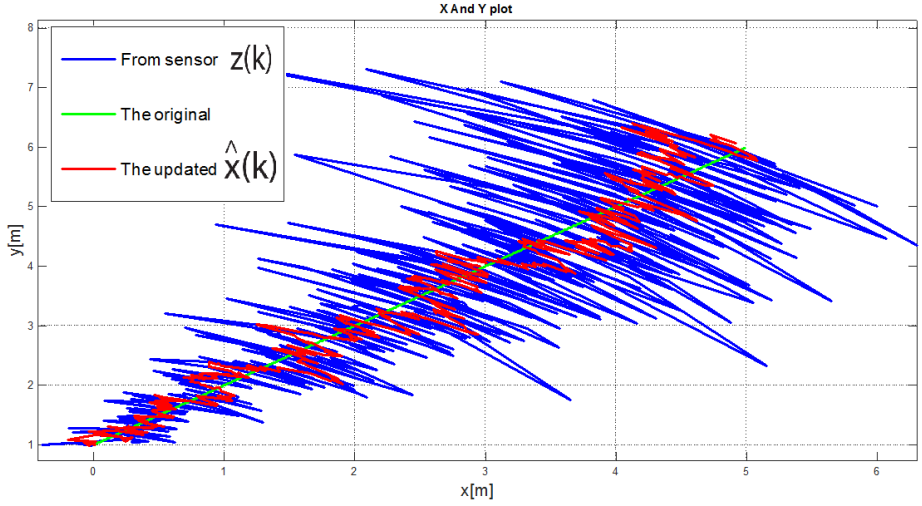


FIGURE 3.5: Extended Kalman, 2 radars example

In this case the x values and y values are given in meters on the x-and y-axis, respectively. The x and y values represent the estimated position of the target in the Cartesian coordinate system. The green graph is the original simulated track, the blue line represents the measurements $z(k)$ and the red graph comes from the updated filter estimate $\hat{x}(k)$.

3.5 Include Radial Velocity

Until this point, only the range measurements, d_1 and d_2, have been actively used to give an improvement in the tracking. Since the pulse Doppler radar also gives information about the Doppler speed, an even better tracking performance could be achieved utilizing both range and Doppler speed.

From the Beijing institute of technology, J. Wang, P. He and T. Long have described a way to add the Doppler speed, or radial velocity as it will be called from now on, to the Kalman filter[10]. It is done in a sequential way where the states first are predicted using the range, and afterwords the radial velocity is added to the prediction. The general way that the algorithm works makes it possible to add more measurements to the state prediction if more information about the target is available. For this thesis though, only the range and radial velocity will be used.

The sequential kalman filter is proposed superior to the conventional extended Kalman filter and it is close to an ideal filter[10]. It begins with the same algorithm as shown

in table 3.2 as the first sequence. Then the next sequence, which will include the radial velocity, continues.

3.5.1 Pseudoradial Velocity Measurement Update

After the measurement update in table 3.1, which is really the position measurement update, the new cycle began. Now, five more lines will be added before the new cycle begins. This part will be called the pseudoradial velocity measurement update.

There are a lot of new definitions made in this section, and they will be explained one by one until they are all stringed together in the end. The first new component is called the directional cosine

$$\Lambda(k) = \sqrt{\big(Y(k)^T Y(k)\big)} Y(k)^T \tag{3.20}$$

where $Y(k)$ is the filtered position estimate after the position measurement update. The algorithm is originally proposed to work with x, y and z. In the case where the radar is 1 dimensional, the $Y(k) = d_1$ or $Y(k) = d_2$ and it will be for the rest of this implementation, just because the complexity increases a lot with the increase of dimensions. Even just using x and y values, as was done with the previous linear and extended kalman filters, proved more difficult than first explained in the paper [10].

Next, a variable denoted $A(k)$ will be defined as following

$$A(k) = \sqrt{\big(Y(k)^T Y(k)\big)} \big(I - \Lambda(k)^T \Lambda(k)\big) \tag{3.21}$$

which will later be used in the creation of a new measurement matrix. I is simply the identity matrix.

Let

$$B(k) = \begin{bmatrix} 0 & 0.5A(k) \\ 0.5A(k) & 0 \end{bmatrix}$$

and define $C(k) = B(k)\hat{P}(k)$ where $\hat{P}(k)$ is the filtered state covariance from the position measurement update. c_{ij} will be the element on row i column j of the $C(k)$ matrix. The new measurement matrix is defined

$$H_2^c(k) = \begin{bmatrix} \big(A(k)\dot{Y}(k)\big)^T & \Lambda(k) \end{bmatrix}$$

where $\dot{Y}(k) = v_1$ or $\dot{Y}(k) = v_2$ contains the velocity estimate. In the end, a new noise variance matrix R needs to be defined

$$R_2^c(k) = \sigma_v^2 + 2\sum_i^I \sum_j^J c_{ij}^2 \tag{3.22}$$

where σ_v^2 is the variance of the velocity measurements and I and J are the lengths of the row and column of the C matrix, respectively.

These new definitions will now be used in equations familiar to those shown in chapter 3.3 and 3.4.

$$S_2(k) = H_2^c(k)\hat{P}(k)H_2^c(k)^T + R_2^c(k) \tag{3.23}$$

which is the innovation covariance used in the next Kalman gain equation

$$K_2(k) = \hat{P}(k)H_2^c(k)^T S_2(k)^{-1} \tag{3.24}$$

Now, the measurement error denoted $\tilde{z}^c(k)$ is defined as

$$\tilde{z}^c(k) = z(k) - \text{trace}(C(k)) \tag{3.25}$$

Finally, the last two equations can be found

$$\hat{\mathbf{x}}^v(k) = \hat{\mathbf{x}}(k) + K_2(k)\tilde{z}^c(k) \tag{3.26}$$

$$\hat{P}^v(k) = \hat{P}(k) - K_2(k)S_2(k)K_2(k)^T \tag{3.27}$$

where $\hat{\mathbf{x}}^v(k)$ and $\hat{P}^v(k)$ are the state estimate and the filtering error covariance after the radial velocity measurements are taken into account.

This algorithm is called the practical sequential pseudomeasurement filter (PSPMF) and its pseudo code is shown below.

TABLE 3.3: **PSPMF Algorithm**

Time Update (Prediction)	
	$\overline{\mathbf{x}}(k) = F \cdot \hat{\mathbf{x}}(k-1)$
	$\overline{P}(k) = F\hat{P}(k-1)F^T + Q$
Measurement Update (Correction)	
	$\tilde{\mathbf{z}}(k) = \mathbf{z}(k) - H(k)\overline{\mathbf{x}}(k)$
	$S(k) = H(k)\overline{P}(k)H(k)^T + R$
	$K(k) = \overline{P}(k)H(k)^T S(k)^{-1}$
	$\hat{\mathbf{x}}(k) = \overline{\mathbf{x}}(k) + K(k)\tilde{\mathbf{z}}(k)$
	$\hat{P}(k) = \overline{P}(k) - K(k)S(k)K(k)^T$
Pseudoradial Velocity Measurement Update	
	$\tilde{z}^c(k) = z(k) - \text{trace}(C(k))$
	$S_2(k) = H_2^c(k)\hat{P}(k)H_2^c(k)^T + R_2^c(k)$
	$K_2(k) = \hat{P}(k)H_2^c(k)^T S_2(k)^{-1}$
	$\hat{\mathbf{x}}^v(k) = \hat{\mathbf{x}}(k) + K_2(k)\tilde{z}^c(k)$
	$\hat{P}^v(k) = \hat{P}(k) - K_2(k)S_2(k)K_2(k)^T$
Other Measurement Updates	
	...
New Cycle ...	

Again, as a proof of concept the following figure is produced using one simulated radar.

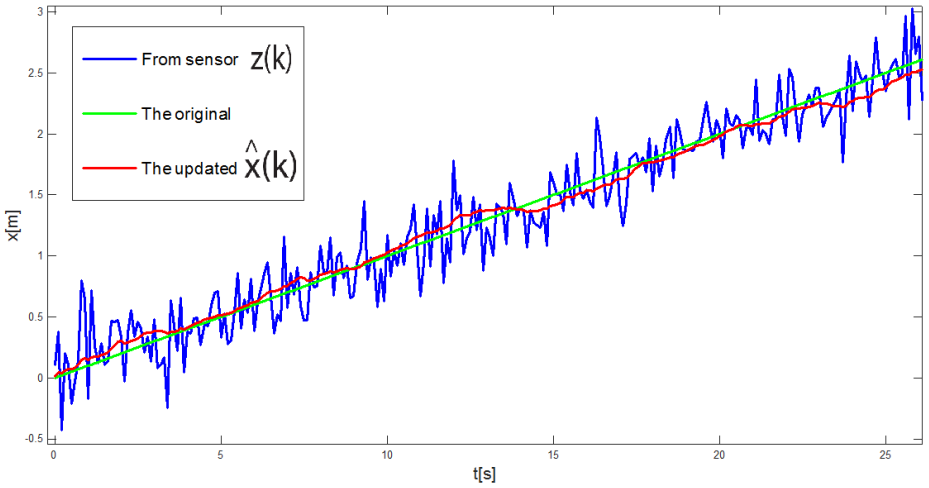


FIGURE 3.6: Extended Kalman with pseudoradial velocity added, 1 radar example

Without doing any analysis of figure 3.6 in this chapter, it can already be seen that this is a improvement from the standard Kalman filter shown in figure 3.4.

Chapter 4

Methods for Data Association

In a radar tracking system, such as the one in this thesis, there can be multiple measurements at each iteration k which can come from the actual state of the target or from noise around it. As mentioned in chapter 2 this is represented as

$$\boldsymbol{d_1} = [d_{1,1}, d_{1,2}, d_{1,2}, ..., d_{1,N}] \tag{4.1}$$

$$\boldsymbol{v_1} = [v_{1,1}, v_{1,2}, v_{1,2}, ..., v_{1,N}] \tag{4.2}$$

Consequently, for every iteration there are lots of possible data for the target state, for both multiple- and single-targets systems. The methods where this data is interpreted are called methods for data association[6]. There are many methods within this category, but in this thesis only a few are used.

The two different data association techniques are showed in the following figure

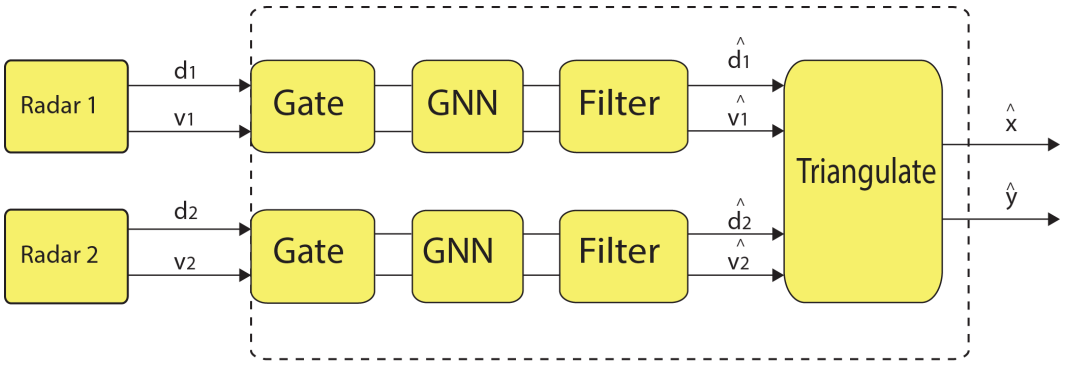


FIGURE 4.1: The total system path

Figure 4.1 can be used as a overview as the techniques are explained, but it also shows that it has been decided to use a separate Kalman filter for each of the radars and then triangulate the estimations. This means that the state vector, introduced in chapter 3, is $\mathbf{x}(k) = [d(k), v(k)]^T$ for both filters. This is an important decision as the option to triangulate and then use the filter is from this point on ruled out. The option to filtrate the radar measurements separately and then triangulate was chosen because of complexity considerations.

4.1 Gating

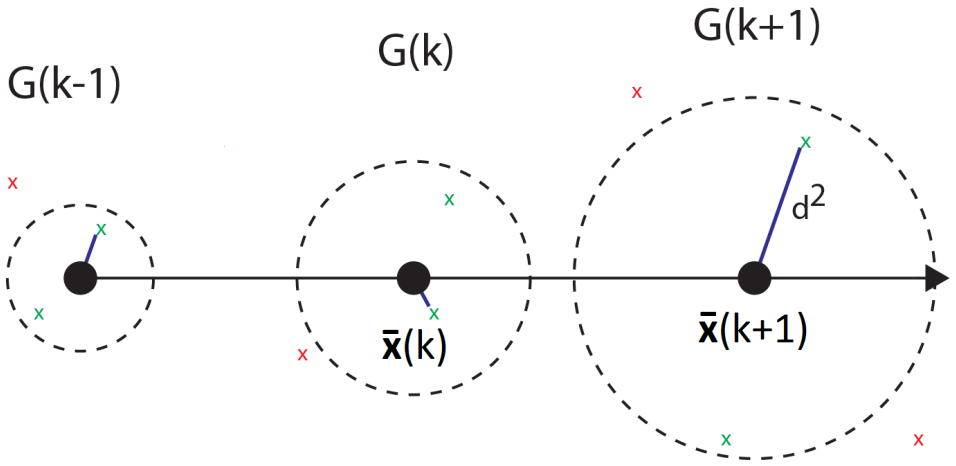


FIGURE 4.2: Illustration of gating

The first data association method used is called gating. It is based on a simple principle and has very good synergy with the Kalman filters. The basic idea is to create a range centered around the predicted state $\overline{\mathbf{x}}(k)$ of the target, where only the data that is within that range will be considered further. As shown i figure 4.2, the gates are centered around the predicted target states, $\overline{\mathbf{x}}(k)$ from the Kalman filter, represented as black filled circles. When the current state $\mathbf{x}(k) = [d(k), v(k)]^T$ is filtered through the Kalman filter, only probable values that has passed the gate will be considered. The complexity of the gate can vary a lot, but a simple and effective way of creating it is by using the innovation covariance matrix S from the Kalman filters. This way, the more error introduced in the system, the bigger the gate will be. The gate is a circle with radius G which is decided by the following formula

$$G = c \cdot \sigma \cdot S(k+1)^{-1} \tag{4.3}$$

where c is a constant used for tweaking, and chosen to be $c = 4$ here. $S(k+1)$ is the predicted innovation covariance matrix and σ is the standard deviation of either $d(k)$ or $v(k)$ from $\mathbf{z}(k)$. $S(k+1)$ is found by the following equations

$$\hat{P}(K+1) = F(k)\hat{P}(k)F(k)^T + Q(k) \tag{4.4}$$

$$S(k+1) = H(k)\hat{P}(K+1)H(k)^T + R(k) \tag{4.5}$$

where these equations should be familiar from chapter 3.

Now that the gate around the estimated position is set, only a measure to find out if the data is inside or outside the gate is needed. This measure is found in the following way.

$$d^2 = \tilde{\mathbf{z}}(k+1)^T S(k+1)^{-1}\tilde{\mathbf{z}}(k+1) \tag{4.6}$$

What remains now is testing whether d^2 is smaller than the radius G of the gate , if it is, then the value tested is within the gated region and can be accepted as a possible candidate for the actual position of the target.

A check needs to be done for each iteration k to see if the measurements are within reasonable distance from the predicted position of the target. As seen i figure 4.2 the gates are the dotted circles created with radius G and are calculated in equation 4.3. Every possible measurement value is evaluated using d^2 found in equation 4.6. If the measurement falls within the gate it is validated, represented by a green x, if not it is discarded and represented in the figure as a red x. Sometimes there are multiple validated measurements, and for this the global nearest neighbour is used, as discussed later in subchapter 4.2.

In the case that no measurement is within the calculated gate, there will be no output value $d(k)$. This happens when the target is very far away from the radar and the measurements are quite noisy. To deal with this, the following quick addition is made to the kalman filter. Before the values are used in the filter, a check is done to find out whether the value is empty or a valid number. If it is empty, the next state is solely predicted from the the previous state $\hat{\mathbf{x}}(k) = \overline{\mathbf{x}}(k)$. This will only help if there are not too many consecutive empty values in a row. If too many empty values appear in a row, the state prediction becomes increasingly inaccurate.

When using these gates, very noisy measurements, that are most likely false targets, are disregarded before they enter the Kalman filter. This way, the values that has the potential to cause big errors are never taken into the filter, resulting in a better performance.

The gating explained until now has just been gating with the distance d in mind. Since both the distance d and velocity v is available for each sensor, the gating can be further improved by using both the position and velocity. The way to implement this is to use exactly the same procedure as before, but this time use velocity instead of position. Then there will be two gating criteria, one for position and one for velocity, where both has to be within their gates for the state to be accepted as valid. The gating is this way stricter and demands more in the sense of precision from the filters.

4.2 Global Nearest Neighbour

After the gating has been completed, there are three possible outcomes. First, it is possible that none of the measurements has been validated within the gate and a weak prediction of the state has to be used, as mentioned in section 4.1. It is also possible that only one value has been validated, which makes that the only possible choice. Lastly, there could be a lot of validated measurements within a gate, which makes the Global Nearest Neighbour data association method necessary.

The GNN (Global Nearest Neighbour) is intuitively easy to understand by looking at figure 4.3.

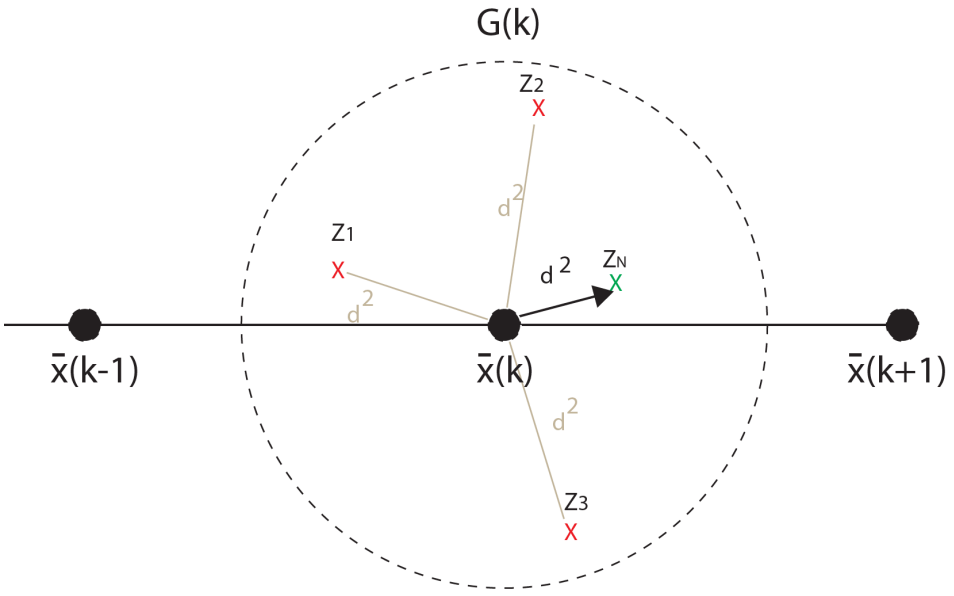


FIGURE 4.3: Illustration of GNN

It is an algorithm which will chose one of all the measurements received represented as $\mathbf{Z} = [Z_1, Z_2, Z_3, ..., Z_N]$. Z can contain either d, v or both depending on what is chosen

to be gated. In this thesis both d and v are gated. The GNN will chose the measurement that is closest to the predicted position, the black filled circle. Since the gating algorithm already has calculated all the distances d^2, it is easy to test these distances sequentially to see which are nearest to the predicted position. The algorithm will therefore chose, as the name suggests, the nearest neighbour. It is a simple technique and yet very effective.

Chapter 5

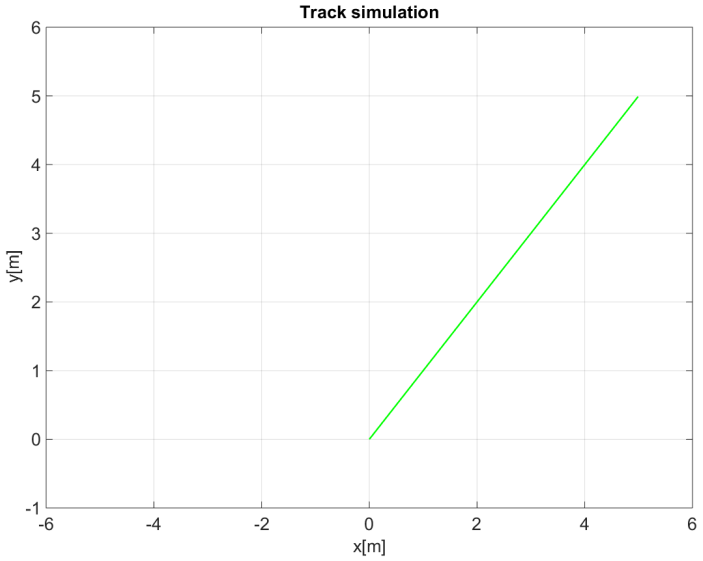
Simulations

An extensive part of this thesis, has revolved around creating realistic simulations of the tracking. Using realistic simulations is a good way to ensure that your filter- and data association-algorithms work properly in a controllable environment, before using the algorithms in a real life situation. The simulations and results from these simulations will be addressed in this chapter. The simulation function codes can be found in appendix A

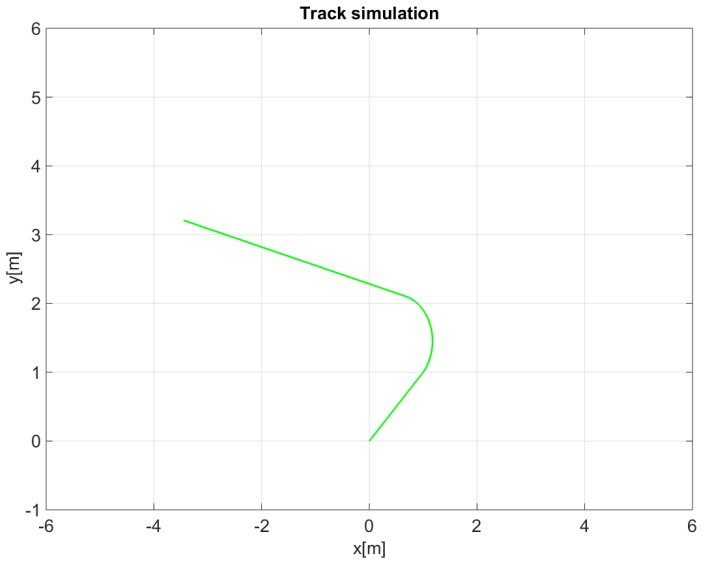
5.1 Track Simulation

The intention of the thesis is to track only one walking person at the time, and the simulation has been done with that in mind. The simulations create values for said walking person that can be used to test the filters and algorithms. The initial speed (v_x, v_y), position (x, y) can be set in the simulator, found in appendix A.2, and therefore indirectly the direction of that person, but also if and when that person will maneuver during the path. A maneuver is defined here as a deliberate change in the persons direction or speed. As previously discussed, the end product of the algorithms will be a position $(\hat{x}, \hat{y})$ in the Cartesian coordinate system.

The next figure shows an example track.



(a) A straight track



(b) A maneuver track

FIGURE 5.1: Track Simulations

In figure 5.1(a), the starting point is given in Cartesian coordinates $(x, y) = (0, 0)$. The speed is $(v_x, v_y) = (0.1, 0.1)$ meters/second and the target continues in the same direction throughout the whole path. The second track, in figure 5.1(b), has a 120 degree maneuver in the middle of the path, but the starting values are still the same.

5.2 Noise

An important aspect of making the simulation realistic is the addition of noise in the system. The way the noise is added, has a big impact on the performance of the system and therefore needs to be as accurate as possible.

The Xethru radars being used in this thesis has a signal to noise ratio, SNR, that varies with respect to the distance d the target is from the radar, given by the following formula [11].

$$\mathrm{SNR}_{dB}(d) = 52 - d \times 12 \tag{5.1}$$

The formula is plotted in the next figure.

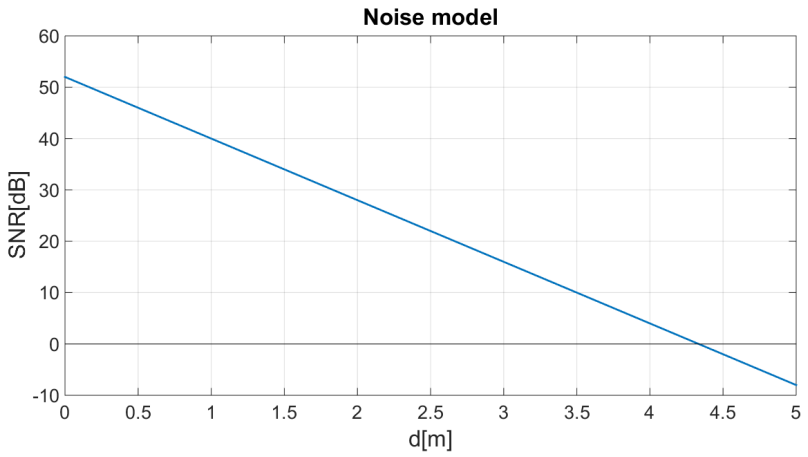


FIGURE 5.2: Illustration of the noise magnitude

As shown in figure 5.2 the noise, blue line, decays linearly with respect to the distance d. At 1 meters distance the $SNR_{dB}(d)$ is 42 dB and decays 12 dB per meter. As the $SNR_{dB}(d)$ goes below about 10 dB, the $SNR_{dB}(d)$ is really too low to give any reasonable value, and really noisy values are to be expected here.

The noise has to be transformed into linear SNR_{lin} before it can be added on to the track simulation. The transformation is done in the following way

$$\mathrm{SNR}_{lin} = 10^{\frac{\mathrm{SNR}_{dB}(d)}{10}} \tag{5.2}$$

The SNR_{lin} and the values calculated from it will all be functions of the distance d because the $SNR_{dB}(d)$ is, but will be written without the (d) for simplicity sake.

This SNR_{lin} will be used in two different noise generators. They both use the same SNR_{lin}, but one will generate the standard deviation for the distance σ_d and the other will generate σ_v for the velocity. First, the distance noise will be generated.

To do this, the range resolution for the Xethru radar has to be found. The pulse length will be called T_{pulse}. It is set to $T_{pulse} = 1$ nanosecond . The distance resolution becomes this

$$\Delta R = \frac{c \times T_{pulse}}{2} = \frac{3 \times 10^8 \times T_{pulse} \times 1 \times 10^9}{2} = 0.15m \tag{5.3}$$

where c is the speed of light. δR can then be used in the following equation [11]

$$\sigma_d = \frac{\delta R}{\sqrt{2 \times \mathrm{SNR}_{lin}}} \tag{5.4}$$

To calculate the σ_v, a speed resolution is needed. The radio frequency wavelength $\lambda = \frac{c}{f} = \frac{3 \times 10^8}{7 \times 10^9} = 0.04929m$ where f is the radars operating frequency and the integration time in pulse Doppler $T_{int} = 1sec$. The T_{int} basically decides how many pulses received in the radar that are used to generate each output of the radar. So the bigger the T_{int} the better the speed resolution gets. These values are used in the next equation

$$\delta V = \frac{\lambda}{2 \times T_{int}} = \frac{0.04929}{2 \times 1} = 0.0215m/s \tag{5.5}$$

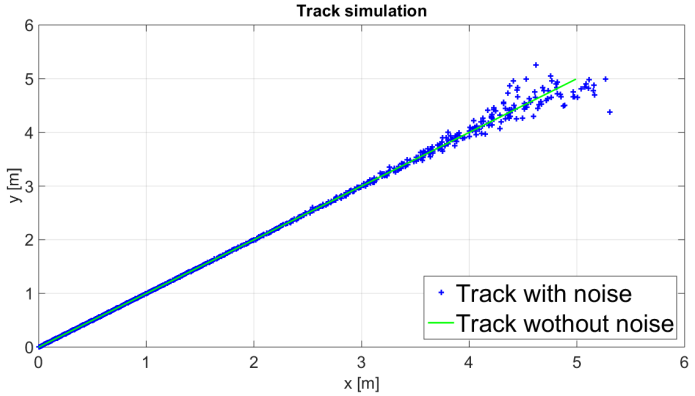
and then [11]

$$\sigma_v = \frac{\delta V}{\sqrt{2 \times \mathrm{SNR}_{lin}}} \tag{5.6}$$

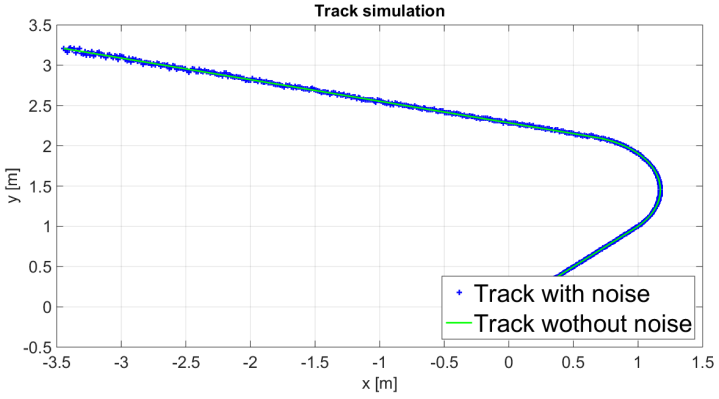
σ_d and σ_r will be the standard deviations of the noises $u_d(k)$ and $u_r(k)$, respectively, added on to the tracks in figure 5.1. The noise is added on in the following way

$$\begin{aligned} x(k) &= x'(k) + u_d(k) \\ y(k) &= y'(k) + u_d(k) \\ v_x(k) &= v'_x(k) + u_r(k) \\ v_y(k) &= v'_y(k) + u_r(k) \end{aligned} \tag{5.7}$$

This will result in a random noise where the standard deviation increases as the target moves away from the radar. When this noise is used with the simulated noiseless tracks it can look like the graphs in the next figure.



(a) A straight track with noise



(b) A maneuver track with noise

FIGURE 5.3: Track Simulations with noise

In figure 5.3 we see the same tracks as in figure 5.1 only this time the noise has been added, seen as the blue + signs. The origo of the Cartesian coordinate system has here been chosen as the zero point of the distance d and d decides the standard deviation of the noise. This will change as the radar positions are decided later. Then d_1 and d_2 will be zero at the position of the radars R_1 and R_2 from figure 2.1.

The figure 5.3 shows that the noise acts as anticipated. The noise becomes greater as the track moves further away from the origo. At the end of the green graph in figure 5.3(a), the target is about $d = \sqrt{(5m)^2 + (5m)^2} = 7.07m$ from the origo. This results in all the scattered noise close to the end of the track. At the end of the track, the green graph in figure 5.3(b), the noise tends to stay closer to the track, than what is seen in figure 5.3(a). This is because the total distance $d = \sqrt{(3.2m)^2 + (3.5m)^2} = 4.74m$ and

therefore does not produce as much noise. The further away the radar is from the target, the more noise is generated, and this is the wanted realistic effect of the noise.

5.3 Simulated Radar Setup

In the simulation, the radars can be placed at any desired position. The radars are placed first and then the coordinate system is defined as if these two radars were placed on the x-axis, as shown in figure 2.1 This is a reasonable placement because they are not in each others field of view and also gives a big joint field of view. It is important to place them at such a distance apart that both radars are close to the area where the tracking will be done. If for example they are 10 m apart, a target that goes between them will be really noisy. Also, if they are too close, even the smallest noise will effect the triangulation function. So for this simulation it has been chosen to put them no more that $1m$ apart. That means, R_1 at (-0.5,0) and R_2 at (0.5,0).

The track simulation has until now been made without any regard to the fact that there is actually two radars R_1 and R_2. Therefore, the original track is converted with a algorithm, found in appendix A.3, into measurements representing the same track, but with two radars placed on the x-axis. . Radar R_1 will have the values d_1 and v_1, and R_2 has d_2 and v_2 for distance and velocity respectively. Now that there are two separate radars with different measurements representing the same track, the simulated noise will also vary depending on the distances d_1 and d_2.

Previously, in chapter 2, figure 2.2 showed the radar measurements d_1, v_1 ,d_2 and v_2 from the two radars going into a question mark box. It is now time to look further into the content of that box. Note that this is the same figure as figure 4.1.

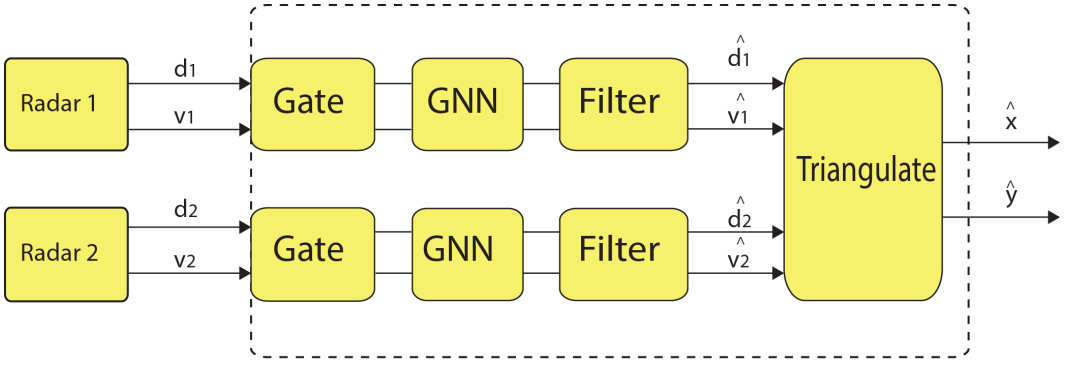


FIGURE 5.4: The total system path

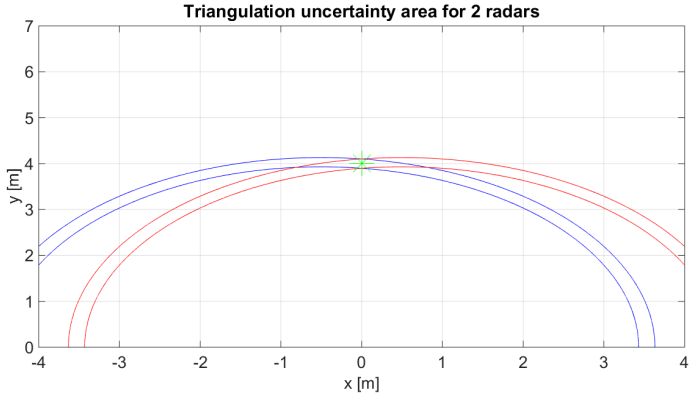
As shown in figure 5.4 the previously unknown box is replaced with two separate paths which ends up in a triangulation function. The two paths are identical except that the inputs d_1, d_2, v_1 and v_2 are different. First, the radar measurements d and v goes into each path. The values are then gated as explained in chapter 4. The gated values are then further processed by the GNN (global nearest neighbour) function. After this point there is supposed to be one value for d and one value for v for each path. As discussed before in chapter 3, sometimes there are no values within the gate and sometimes there is only one single value.

When the filtered values $\hat{d_1}$, $\hat{d_2}$, $\hat{v_1}$ and $\hat{v_2}$ are found, they are sent into a triangulation function found in appendix A.8, which, using common trigonometry transforms the separate radar measurements into one single estimated position $(\hat{x}, \hat{y})$ which is the goal of the thesis.

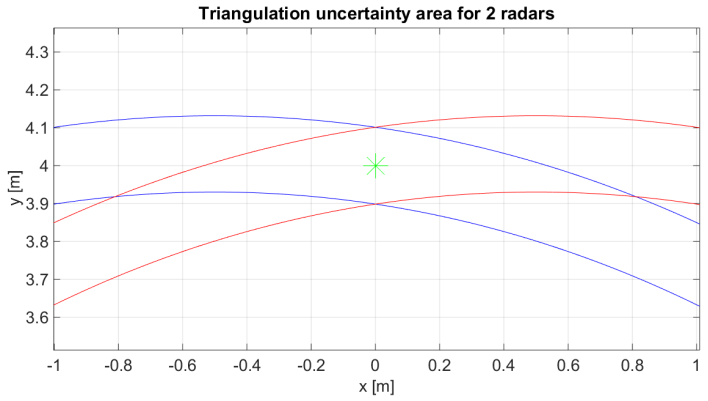
5.4 Triangulation Expected Error

The two radars R1 and R2 both have their own noise and uncertainties. This will result in an accumulated error in the triangulation function. What this error will look like and how big it will be is at this point unknown. To get a better understanding of what errors to expect in the triangulation, a function has been made which produces some interesting figures of the triangulation concept, found in appendix C

The figure's goal is to exemplify the concept of triangulation and the errors generated from that. When the two radars are 1 m apart, as the case is for the simulations, the following figure is produced.



(a) Triangulation error area



(b) Triangulation error area, zoom

FIGURE 5.5: Triangulation error area

Figure 5.5 contains two figures which show the same plot, but one is zoomed in. 5.5(a) shows four lines. The two blue lines are from radar $R1$, and the position of the target can be anywhere between the two lines. How big the gap between the lines of the same colors are, is decided by SNR_{dB}. The red lines are from the other radar, $R2$. 5.5(b) shows the intersection of the lines. The diamond shape area with a green star in the middle, is the area where the targets can appear due to noise. The green star is the actual position of the target. If there were no noise in the system, the gap between the two blue lines, and also between the red ones, would be gone, and there would only be one intersection point on top of the green star, the actual target position. Since there is noise in the system, we get 4 intersections points, which all confines the uncertainty area of the triangulation.

Figure 5.5(b) shows the area when we have a $\text{SNR}_{dB} = 10$ at the position $(0, 4)$. So in a worst case scenario, the estimated position can be at the point furthest away from the actual target, at the intersection all the way to the right or left of the figure. At this distance the worst error in the x-direction is 0.8 m and 0.1 m in the y-direction.

So, in the simulation or for the real radar values, one can expect error close to those values at about 4 m distance from origo. This figure is very handy when checking if the triangulation gives out reasonable values for different ranges.

5.5 Simulation Results and Discussion

Before showing any results, there are a few settings that needs to be explained.

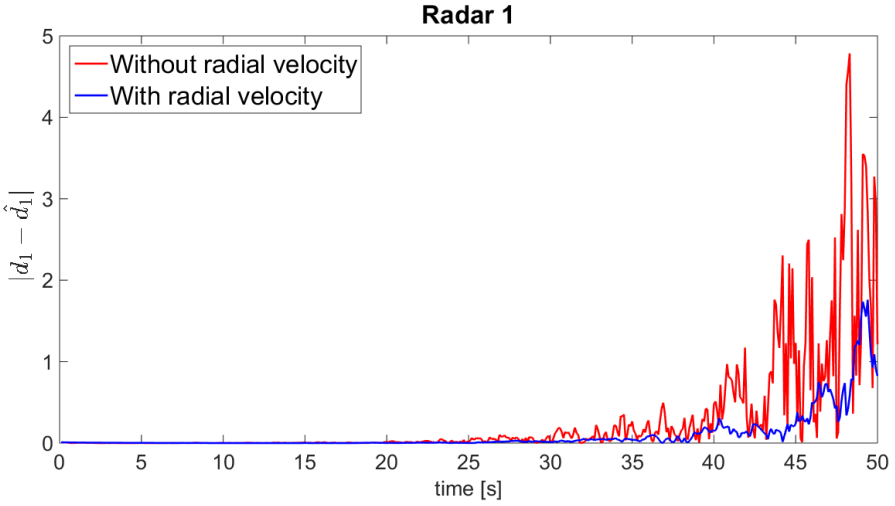
5.5.1 Settings

Since the system is set up in the way as shown in figure 5.4, with two separate paths joined into one in the triangulation function, the state for one path is just $\mathbf{x}(k) = [d(k), v(k)]^T$. This again means that, as mentioned in chapter 3, the R and Q becomes this

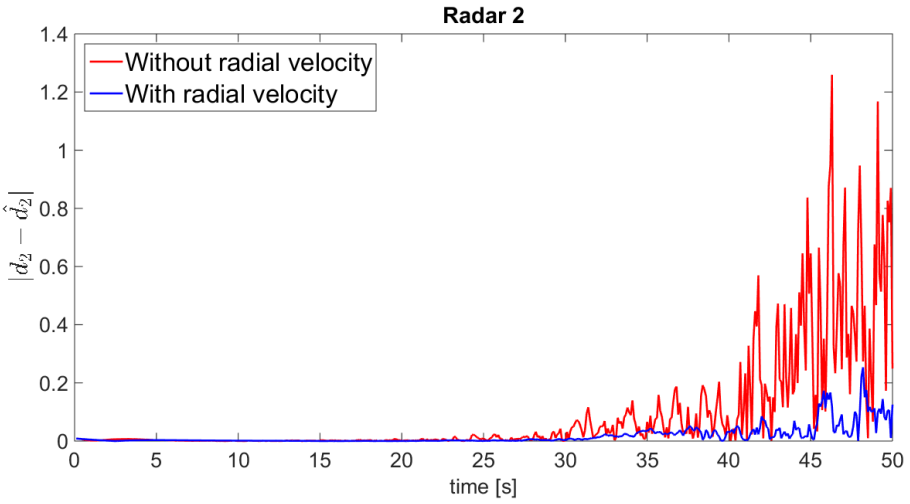
$$R = \begin{bmatrix} {\sigma_d}^2 \end{bmatrix} = \begin{bmatrix} 0.2^2 \end{bmatrix}$$

$$Q = \begin{bmatrix} {\sigma_d}^2 & 0 \\ 0 & {\sigma_v}^2 \end{bmatrix} = \begin{bmatrix} 0.2^2 & 0 \\ 0 & 0.1^2 \end{bmatrix}$$

Also, the decision to use or to not to use the radial velocity has to be taken. In the next figure a comparison between d and $\hat{d}$ with and without the addition of radial velocity has been made for both radars.



(a) $|d_1 - \hat{d}_1|$ in Radar 1, Straight track



(b) $|d_2 - \hat{d}_2|$ in Radar 2, Straight track

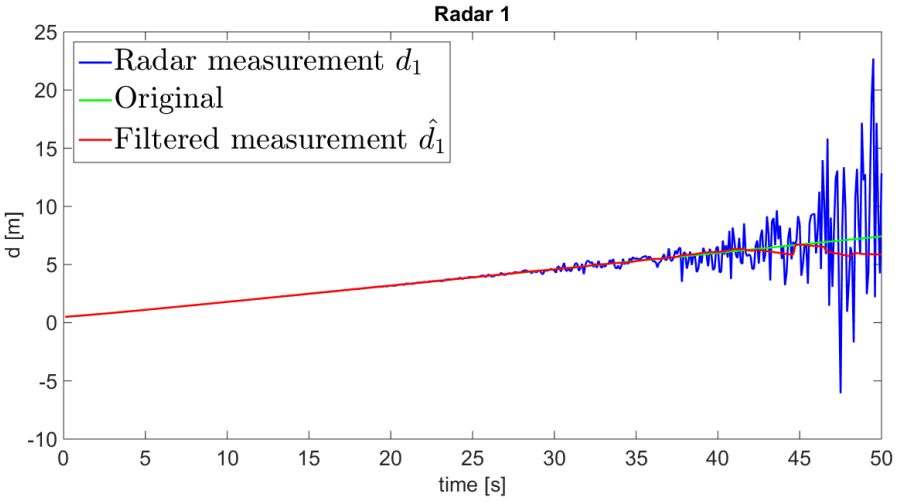
FIGURE 5.6: With and without radial velocity

Both these subfigures in figure 5.6 has been simulated with the straight path shown in figure 5.1(a). They show the difference $|d_1 - \hat{d}_1|$ and $|d_2 - \hat{d}_2|$ where the addition of radial velocity in the system is toggled on and off. d_1, $\hat{d}_1$, d_2 and $\hat{d}_2$ are found in figure 5.4. As seen there the values has not been run through the triangulation function yet. The red line represents the error without the use of radial velocity in the filter, while the blue line is with the use of radial velocity.

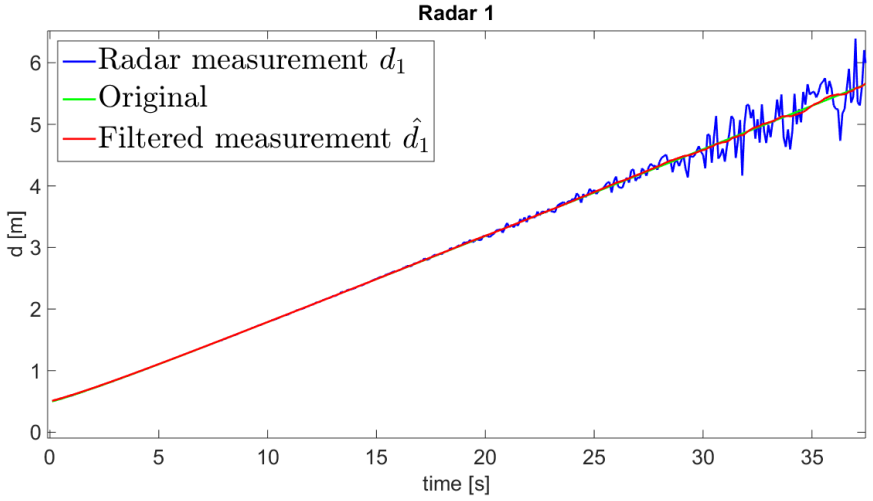
Figure 5.6 shows that the inclusion of radial velocity gives less error that without it, for both radars, and it is from this point on chosen to use the addition of radial velocity in the Kalman filters.

5.5.2 Gate, GNN and Kalman Filter

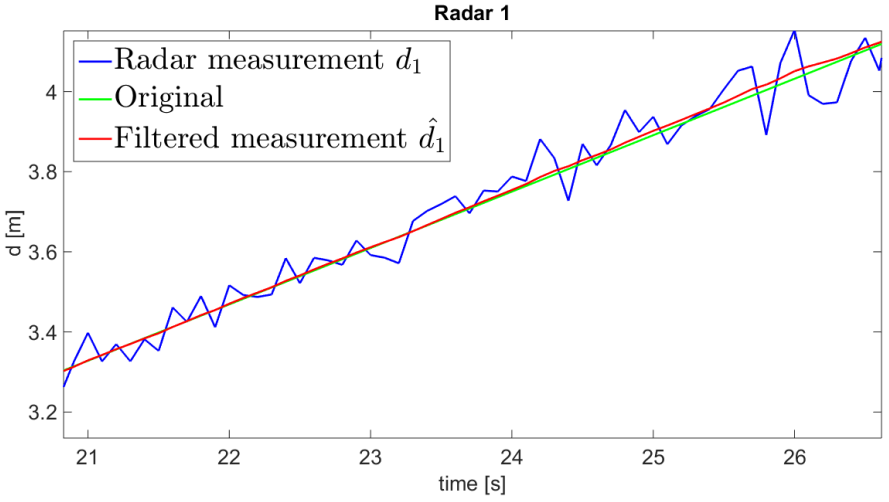
Now that Q, R and the use of radial velocity in the kalman filters has been decided, a plot of the path from each radar through the gate, GNN and filters can be plotted.



(a) Radar 1, whole figure

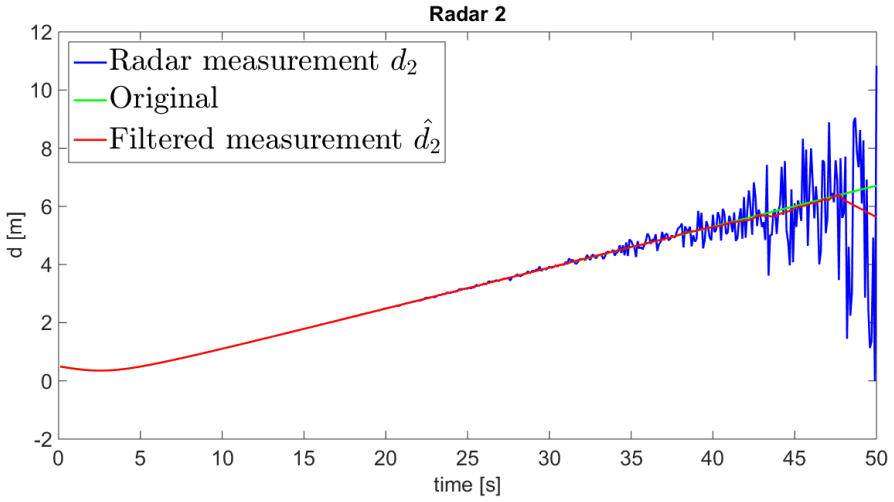


(b) Radar 1, 1x zoom

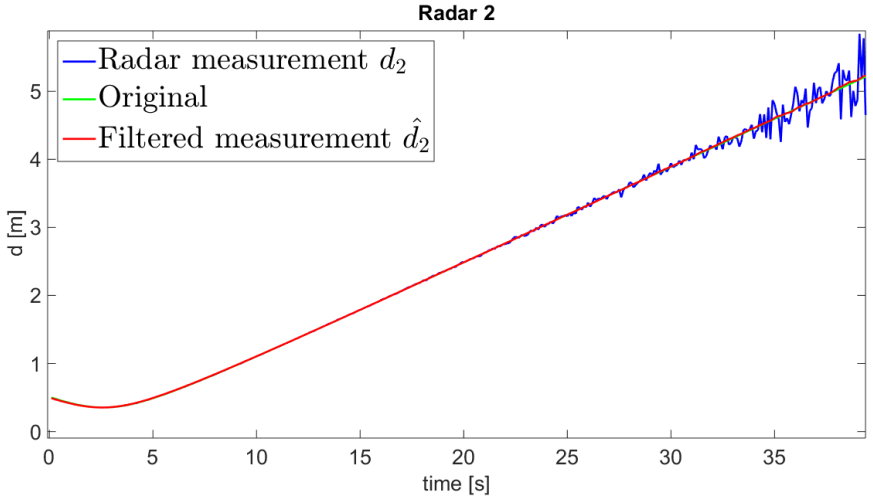


(c) Radar 1, 2x zoom

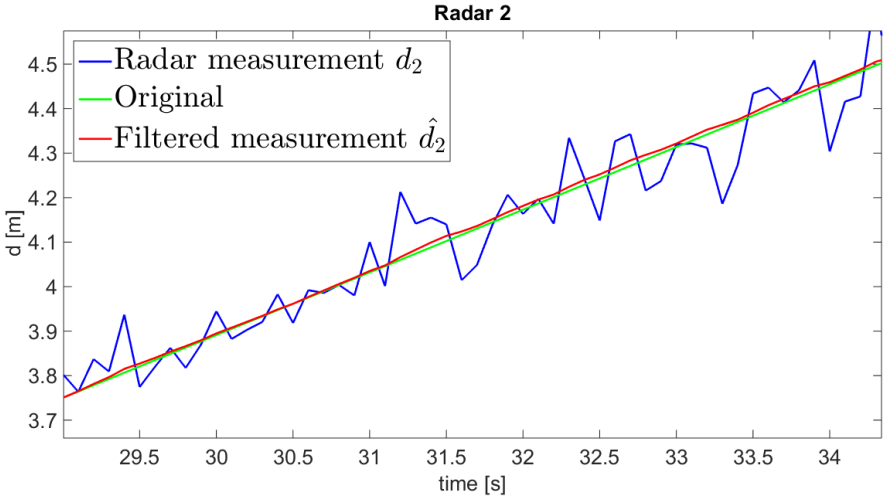
FIGURE 5.7: Gating, GNN and Filter of Straight track from Radar 1



(a) Radar 2, whole figure



(b) Radar 2, 1x zoom



(c) Radar 2, 2x zoom

FIGURE 5.8: Gating, GNN and Filter of Straight track from Radar 2

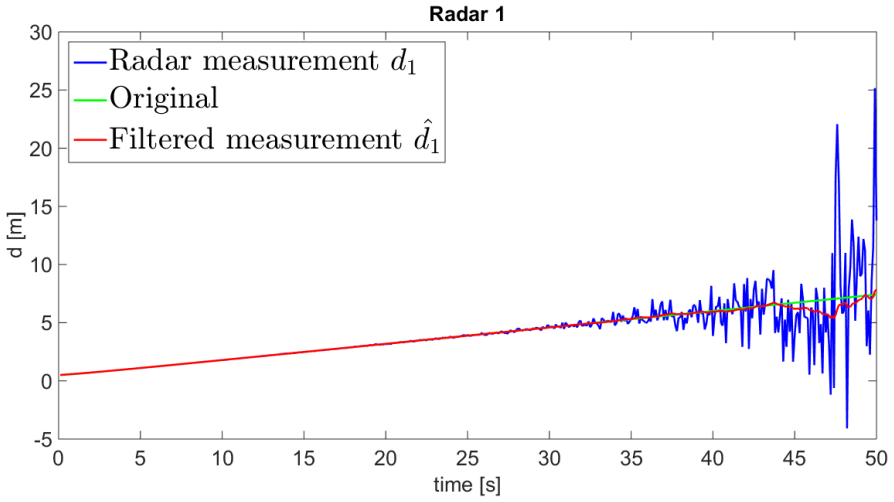
The two figures 5.7 and 5.8 both show the same principles, except that the measurements comes from radar $R1$ and radar $R2$, respectively. Radar $R1$ is positioned at $(-0.5, 0)$ and radar $R2$ is at $(0.5, 0)$. There will therefore be a slight difference between the two figures other than the difference created by the noise. This is because the simulated track, shown in figure 5.1(a), is moving closer to R2 than R1. The difference is seen as the target continuously moves away from $R1$, but for $R2$ the distance decreases before it increases.

The subfigures in figure 5.7 are all taken from the same plot, but with different scales. Subfigure 5.7(a) is the graph with no zoom at all. It is seen that after about 46 seconds, when the target is about $6m$ away from radar $R1$, the filtered measurement $\hat{d}_1$ no longer follows the original green track. This is because the noisy measurements d_1 differ too much from the predicted measurements and they end up outside the gate. As previously mentioned, when there are no values inside the gate the kalman filter uses $\overline{\mathbf{x}}(k)$ to estimate $\hat{\mathbf{x}}(k)$. This is sometimes a good enough estimate, except when there are no values inside the gate for many iterations in a row. This happens after 46 seconds, and $\hat{d}_1$ becomes increasingly worse.

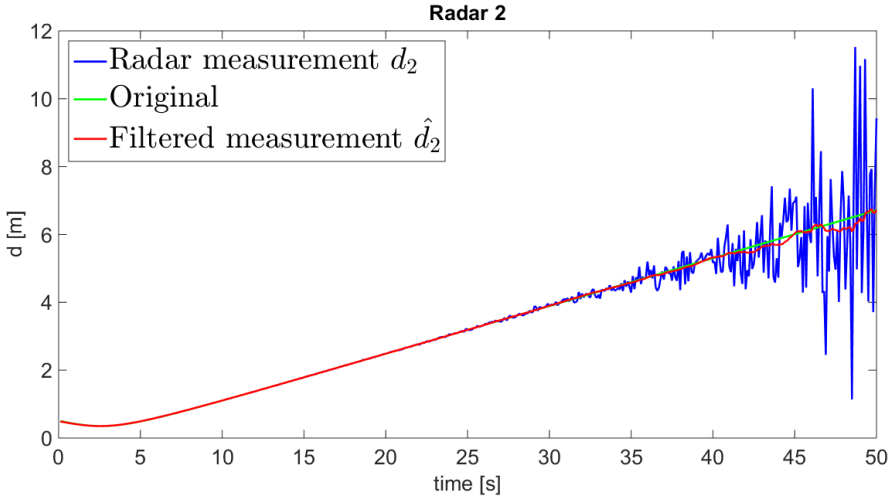
The intention of figure 5.7(b) is to zoom in on the ranges where the noise is still at a reasonable level. The filtered measurements $\hat{d}_1$ stays close to the green original track as wanted as long as the target is closer than $6m$ from the radar, which matches the noise model in figure 5.2. Subfigure 5.7(c) is a even more zoomed in version of the graph and was made to exemplify that the even when the noise conditions are good, the kalman filter still improves the outcome $\hat{d}_1$.

The subfigures in figure 5.8 are also from the same plot and the subfigures here are made for the same reasons as for the subfigures in figure 5.7. There are a lot of similarities, but one difference can be seen in figure 5.8(b) when the target moves closer to the radar, which is as expected. This results in a longer range with reasonable noise levels compared to radar $R1$. $R1$ kept its measurements d_1 inside the gate for about 43-47 seconds, while $R2$ keeps them inside the gate for about 49 seconds. This translates into $6m$ versus $6.5m$ acceptable distances for radar $R1$ and $R2$, respectively.

Different gates and gating methods gives a lot of different performances. To prove that is is the gating that is the part of the path that holds back the rest, a new plot where the gating has been turned off.



(a) Radar 1 without gating



(b) Radar 2 without gating

FIGURE 5.9: Paths shown without gating

In the figures 5.9(a) and 5.9(b) it can be seen that $\hat{d}_1$ and $\hat{d}_2$ does a good job of following the original tracks. They are different from figure 5.7(a) and 5.8(a) in the way that towards the end of the track, the noisy values are still used in the kalman filter. This can be seen as the filtered measurement $\hat{d}$ is still trying to follow the original track. While with the gates on, the estimates are moving steadily away from the track after about 47 seconds since no values are within the gate. So the question is then, why is the gate even needed? The reason is that the estimated $\hat{d}$'s are still too noisy to get a good final position $(\hat{x}, \hat{y})$ after they are run through the triangulation function, which will be shown soon. Also, when we are using real radar measurements, some unexpected noise signals seem to appear at random times during the tracking and without gating away

those values, they would distort the whole estimation process, but more about that in chapter 6. The next part of the path is the triangulation and will be shown next.

5.5.3 Triangulation

Now that the radar measurements from the two radars has been through gating, GNN and Kalman filtering, it is time to join the two paths with the triangulation function.

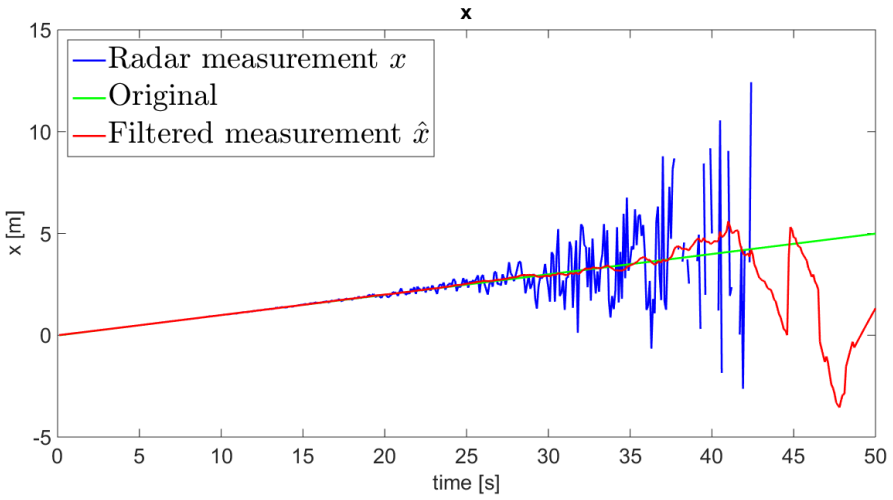
There is no magic to this function. It uses the two radar measurements $\hat{d}_1$ and $\hat{d}_2$ and the two radar positions $s_1 = (-0.5, 0)$ and $s_2 = (0.5, 0)$ to find the actual position $(\hat{x}, \hat{y})$.

The main part of the triangulation formula is shown below. Note that d_0 is the distance between the radars divided by two, $d_0 = \frac{|s_2 - s_1|}{2}$.

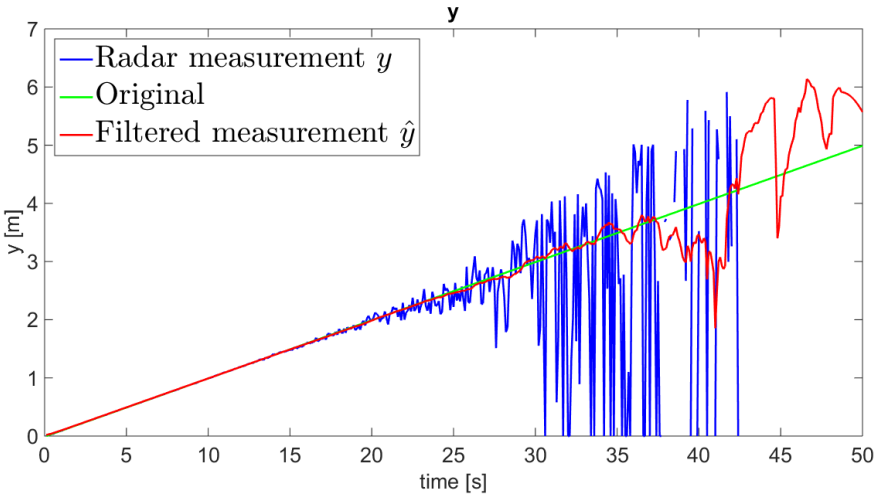
$$\begin{aligned} \hat{x}(k) &= \frac{\hat{d}_1(k)^2 - \hat{d}_2(k)^2}{4d_0} \\ \hat{y}(k) &= \sqrt{\hat{d}_2(k)^2 - d_0^2 + \frac{\hat{d}_1(k)^2 - \hat{d}_2(k)^2}{2} - \left(\frac{\hat{d}_1(k)^2 - \hat{d}_2(k)^2}{4d_0}\right)^2} \end{aligned} \tag{5.8}$$

Because of the independent noise from both radars the final target might be in two different positions depending on if d_1 or d_2 is prioritized. Depending on the target position compared to s_1 and s_2 the function prioritizes different values. The function prioritizes the measurements from the radar that is closest to the target and therefore minimizes the error.

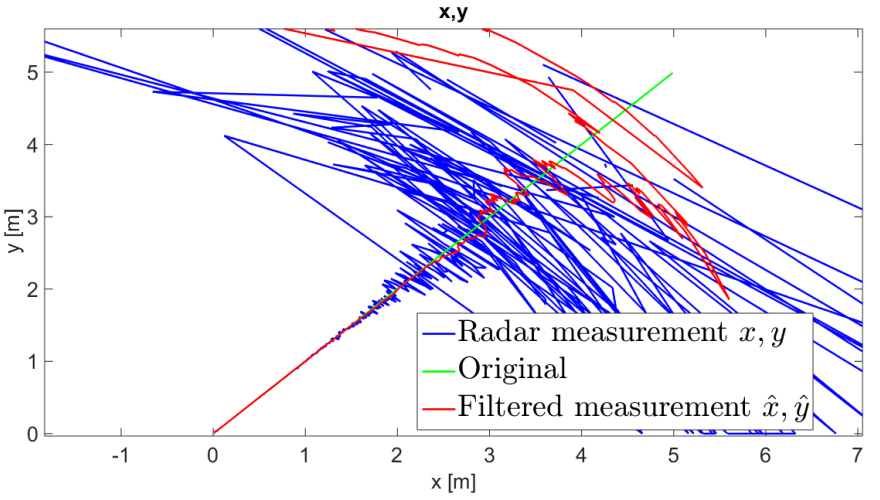
The next figure shows the result of the triangulation function from the same path as before, shown in 5.1(a).



(a) Triangulated x values



(b) Triangulated y values



(c) Triangulated final position

FIGURE 5.10: Triangulated values

Figure 5.10 shows three subfigures that represents the final position $(\hat{x}, \hat{y})$ after the triangulation. Figure 5.10(b) holds the y values, 5.10(a) holds the x values and 5.10(c) holds both the x and y values.

Since the estimated position $(\hat{x}, \hat{y})$ is generated from a combination of both radars $R1$ and $R2$, the accuracy of the final estimated positions will be effected by the accuracy of either of the radars. If for instance $R1$ gives a good estimation, but $R2$ does not, the triangulated final position $(\hat{x}, \hat{y})$ will also be inaccurate . It can already be seen in both figure 5.10(b) and 5.10(a) that the radar measurements x and y start becoming bigger than the range that the gates allows. For example if $R1$ has really noisy measurements after $5m$, the whole system will be affected by that. Since this system is based around origo this will result in very noisy measurements already at $4.5m$ distance. Imagine that the simulated track just follows the x-axis in the positive direction. So when the target is $4m$ away from origo, it is actually $4.5m$ away from $R1$ and $3.5m$ away from $R2$. $R1$ will in this case be the main contributor of really noisy measurements since the noise model varies with distance.

The way the noise model is defined, the resulting filtered measurement $\hat{x}, \hat{y}$ does not give any position worth trusting after about $4m$ distance on both axis. This amounts to $\sqrt{4m^2 + 4m^2} = 4 \times \sqrt{2}m = 5.66m$ distance from the origo.

When only looking at the radar measurements, the blue lines, the position becomes untrustworthy way sooner than the estimated position, red lines, in all three figures in figure 5.10. This means that the gate, GNN and kalman filters contributes positively towards a better position estimate.

The figure 5.10(c) gives out reasonable values compared with the triangulation simulations from section 5.4. At the position $(2.8, 2.8)$ the target is about 4 m away from origo. As the simulation shows in figure 5.5(b), the expected error is about 0.8 m in the x-direction. This is in the same ballpark as the difference between the radar measurements (x, y) and the original track in figure 5.10(c). It was not expected that the error was exactly what was predicted, but it helps to know that they are not too far off.

As mentioned before, the necessity of the gate shows itself when looking at figure 5.10(c) compared to the next figure

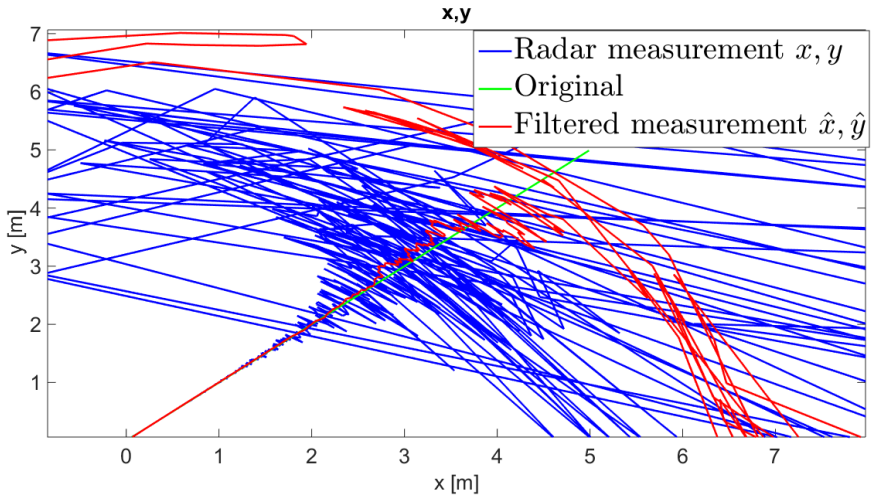


FIGURE 5.11: Triangulation without gating

Figure 5.11 shows the triangulation results when no gating is being used. This is much worse than what is seen in figure 5.10(c). The radar measurements, blue lines, are all over the plot and the filtered measurements are also worse.

Having worked with the filters a lot, it has become evident that it is possible to tweak the filters into giving more accurate estimations. By altering Q, R and some of the initial conditions one can get a even better estimation than what has been shown here. The problem is that these settings and performances are based on unrealistic settings, and almost psychic initial guesses. Given that the goal of these simulations were not to provide the best estimation of the position, but rather the best estimates given realistic settings and noises, the result is more than acceptable.

The reason the simulations are set to be realistic is because the simulation process is merely a means to develop algorithms that can be used in real situations with real radars. If the simulations only focused on giving the best possible estimations, the algorithms would most likely not work in real situations. The way the author use the simulations is to provide the most robust algorithms to be used in said real situations. This is easier to do in a simulation environment when the inputs are controlled and the outputs are predictable.

The next step is to use the system of algorithms in a real situation where a walking person is tracked using the real radars.

Chapter 6

True Radar Tracking

Now that the complete path from radar to position $(\hat{x}, \hat{y})$, shown in figure 5.4, has been proven to work during simulation, it is time to try it out with real radars.

This will be done with the same radar setup as shown in figure 5.4, step by step starting out with only one radar at the time, testing out gating, GNN and kalman filtering. In the end the two paths will be joint in the triangulation function. The real radar measurement functions can be found in appendix B.

Before the results are shown it is important to note that for the actual radar measurement the time between samples is 1 second, compared to the simulated 0.1 seconds. That is just the way the radars are programmed at the moment, but it is enough as long as the person is walking. Therefore, the simulation samples are spaced closer together and can in theory handle faster targets. Another note is that the radar operates in the space 1.5 m- 4.5 m. So values outside those ranges will not be detected.

6.1 Radar Results and Discussion

The results will be divided into two sections, one for the single radar and another for both radars working simultaneously. First, it will be shown that 1 radar works properly by itself. In other words, that one of the paths gives reasonable values $\hat{d}$.

6.1.1 Gating, GNN and Kalman Filter for One Radar

Figure 6.1 shows how the raw values from the radar comes in to be processed. It represents a track made of a person walking in a straight path, starting close to the radar, walking away from it and then walking back to it.

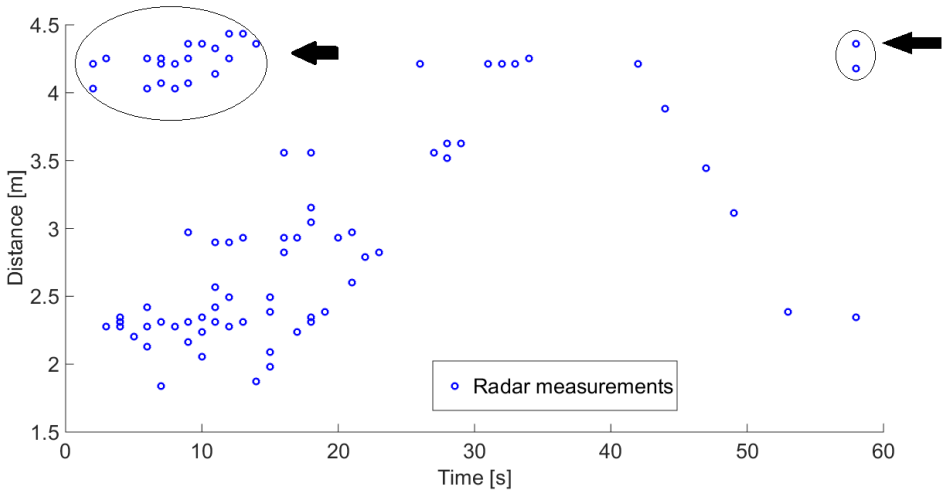


FIGURE 6.1: Raw values

It is seen that multiple measurements can appear at the same instant. This means that the GNN is highly necessary. Without it the algorithm would not know which measurements to filter. The cluster of values up in the top right and top left corner of the figure, pointed at by the arrows, are unfortunate false values. These appear because the current radar still has some unresolved bugs when the target is outside the given range $1.5-4.5m$. These false values will somewhat be resolved by the gating algorithms.

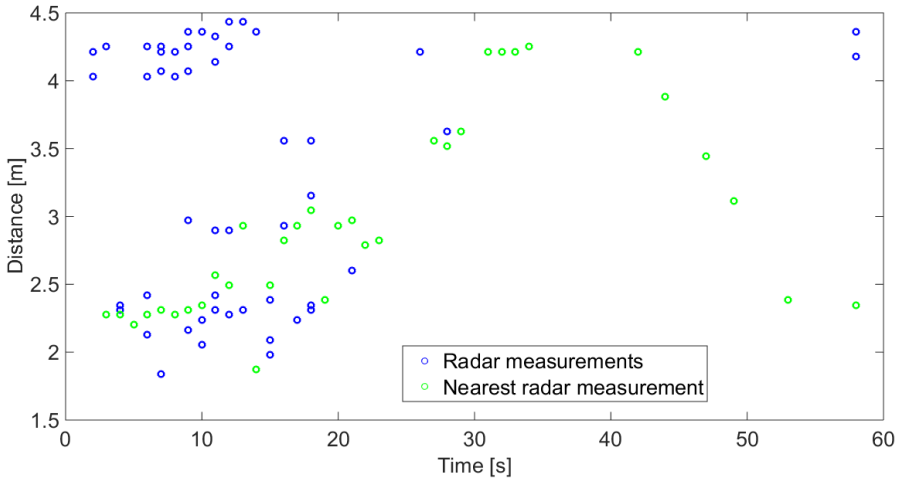
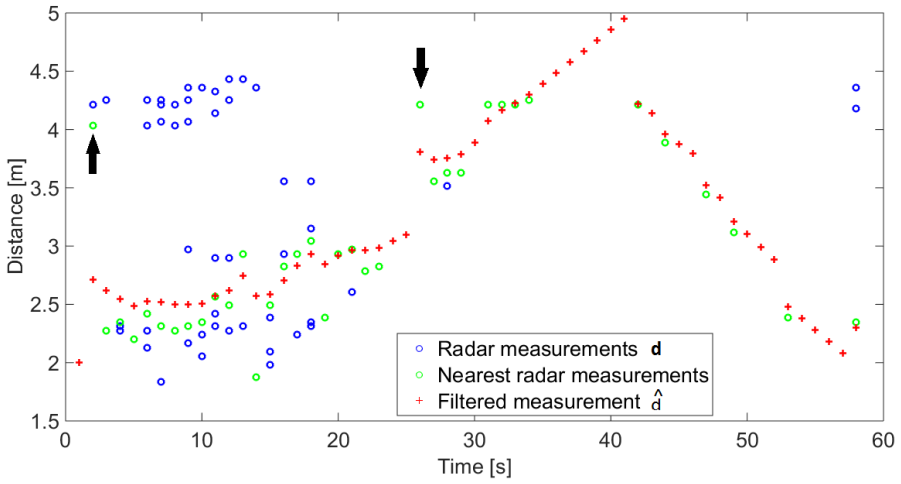


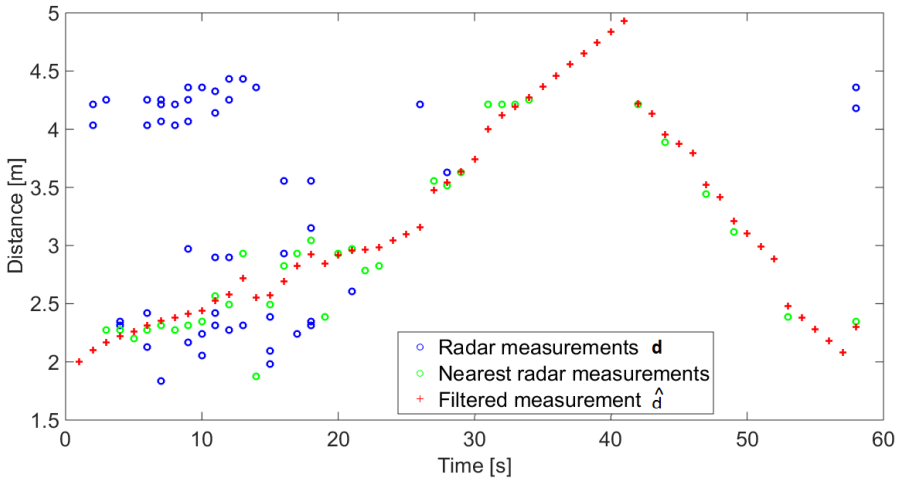
FIGURE 6.2: Raw values and GNN

Figure 6.2 above shows the effect of the GNN. The measurements closest to the predicted state $\overline{x}(k)$ will be stored and colored like a green ring. Only the green rings will be considered by the Kalman filter and the two clusters of measurements are not corrupting the system as much anymore.

Now, the values can be run through the filter as shown in the next figure.



(a) Radar 1 without gating



(b) Radar 1 with gating

FIGURE 6.3: Walking path without and with gating

In figure 6.3 there are two subfigures 6.3(a) and 6.3(b) which represents the complete system of algorithms before triangulation without and with gating, respectively.

In the first figure 6.3(a) the red crosses, representing the filtered position $\hat{d}$, does a decent job of finding a reasonable position for a person walking back and forth. Still, it can be seen at several points in the graph, showed with arrows, that the filtered positions gets pulled away from what is most likely the correct path. For example, after 3 seconds the green circle pulls the filtered measurements away from the rest of the values. The same happens again after 26 seconds. This may be because there are few values at those times and the closest measurement given by the GNN is still really far away from the path.

To fix this problem, the gating is activated. This has been done in the second figure 6.3(b). It can be seen that the previous green circles at for example the 3rd and 26th second is no longer green, but blue. This means that they will no longer be evaluated by the Kalman filter and as a result gives a better filtered measurement representing the targets position. The improvement is clear when comparing the evenness of the path made by the red crosses in bot figures.

After about 25 seconds, in both 6.3(a) and 6.3(b) there seems to be no values for a long time. This may be because the person has walked outside the set range of the radar. There, the Kalman filter uses its previous estimations and the F matrix from chapter 3. It does this until the person appears again in the range of the radar at the 42nd second.

So far the the system path, starting from the radar to the triangulation function, has worked properly. Some problems have appeared, but they have been solved by different parts of the path, either gating, GNN or filtering. Some of these problems has occurred because of how the radar is not completely tuned to be doing tracking.

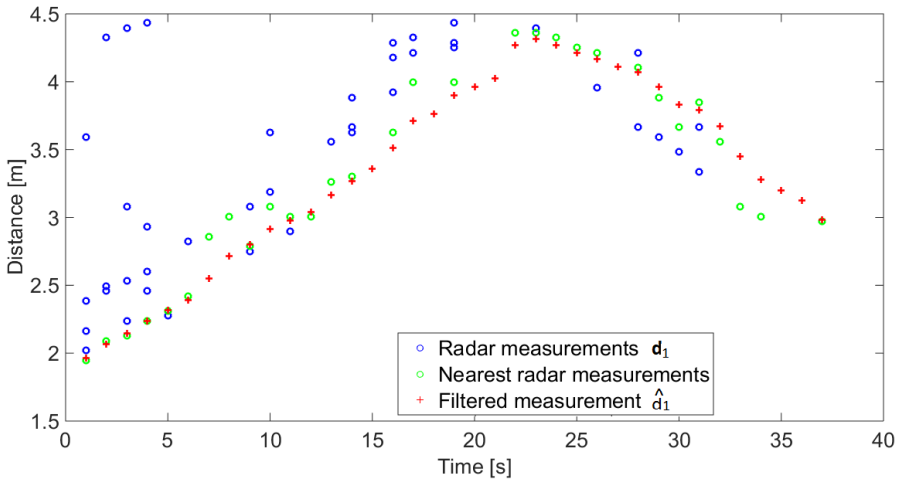
Individually the radars works properly and it is time to test out two radars at the same time.

6.1.2 Two Radars and Triangulation

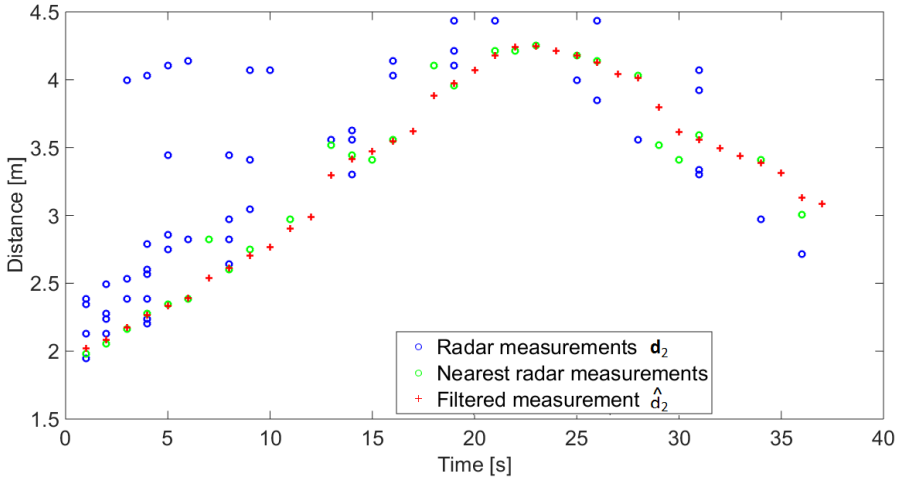
During the simulations, the radars were stationed 1 m apart from each other. Due to space limitations in the office, it was decided to place them 0.6 m apart for the actual radar measurements. This is still a perfectly good placement and going from 1 m to 0.6 m will not have a critical impact on the results.

The target used was a average sized person. His starting position was about $(x, y) = (0, 2)$. He was told to walk away from the radars and stop, then turn around and walk back towards the radars. There was not used a secondary system to track the person, so there is no original track to compare the results to, other than the description of the path above.

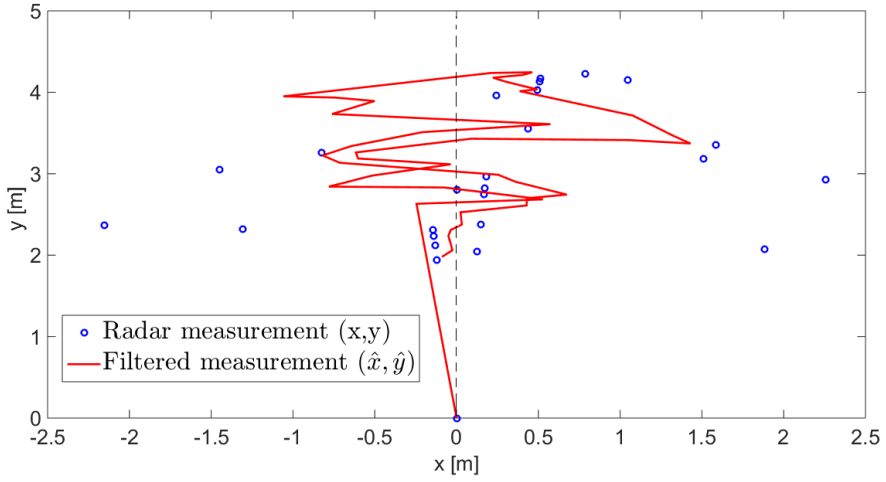
The results are shown in the next figure



(a) From radar R1



(b) Fron radar R2



(c) Triangulated final position

FIGURE 6.4: Triangulated values

Figure 6.4 contain three subfigures. The two first are figure 6.4(a) and 6.4(b). These should show familiar values as the ones seen in figure 6.3(b) and what we see in them has previously been discussed. The two first figures are also quite similar. This is because the walked path has been in the middle of both radars and the distances are supposed to be similar. The reason figure 6.4(a) and 6.4(b) is shown, is because they are the exact values that was used to create the third figure 6.4(c).

Figure 6.4(c) shows the final position estimate that is calculated by the triangulation function. The dotted black line is there as a visual tool to imagine where the supposedly walked path was. As the distance increases, the estimated $(\hat{x}, \hat{y})$ stray further from the path, which is as expected, and backed up by the simulations. At about $(0, 2)$ the difference between the original path and filtered measurement is really accurate. About 0.1 m from the supposed track.

The radar measurements, blue circles, are at most up to 2 m away from the original path, and the algorithms do a decent job to correct for this. The difference between the radar measurements and the original path in figure 6.4(c), are a bit higher than predicted by the simulated in section 5.4. This means that the actual SNR might be a bit lower than what was simulated with in figure 5.5.

6.1.3 Comparison to Simulations

Comparing the final position $(\hat{x}, \hat{y})$ from the real radars to the simulated final position is not a trivial problem. For the true radar tracking, values for a original path is not available and it is hard to directly compare how similar the performances are to the simulation. The simulation has been used as a tool to create a safe environment to build the algorithms used in the true radar tracking, where the outputs and inputs are controlled. The point was not to make the simulation perform exactly as the true radar tracking, but rather make them work in the same manner, and have similar values. Therefore a direct comparison has not been made in this thesis between the results from the simulation and the true radar tracking.

Using the various figures from chapter 5 and 6, some important points can still be made when comparing the results from the two chapters.

The results show that the simulated noise acts close to realistic. Looking at figure 6.4(c) and figure 5.10(c) the noise increases greatly in both figures as the distance from origo increases. When the target is about 2-3 m away from origo, the radar measurements are about 0.1-0.2 m away from the original tracks. As seen in figure 5.10(c), the noise increases a lot when the target is more than 4 m away from origo. The position $(\hat{x}, \hat{y})$ is

about 1-2 m away from the original track. This is what we see in figure 6.4(c) as well, but the filtered estimate $(\hat{x}, \hat{y})$ is a lot better in the simulation at that distance.

The figures in 6.4 and 5.10 also show that the gating, GNN and kalman filters work properly and improve the $(\hat{x}, \hat{y})$. Still, it seems that the kalman filters do a better job with the simulation values. The simulations gives a better general performance than the real radar setup, but this is to be expected.

Chapter 7

Further Work

The investigations and results made in this thesis show that there are several areas for further work that may lead to some improvements. These topics might be interesting to follow up on at a later time.

First of all, in this thesis the focus has been on tracking one target. In advanced tracking systems, it is normal to have the ability to keep track of multiple targets. The implementation of multiple target is definitely worth looking into.

Another part of the system that has room for improvement is the gating. The gating function used in this thesis is a circular one. Tracking can be done with for example elliptical or asymmetrical gating which is better when the target is maneuvering. It might at least be worth trying out different gating methods.

The global nearest neighbour (GNN) method is well suited for the tracking of one target. If the goal is to track multiple targets, a closer look into different data association methods is needed. In this area there are a lot of different association methods that would be worth researching further.

This thesis did not focus on the actual performance of the sensors. It is still important to note that the radars used has not been optimized to be used in tracking. If these radars are to be used for tracking in the future, the radars themselves has a lot of potential to improve and a lot of further work could be done with these.

Chapter 8

Conclusive Remarks

The study set out to create a target tracking system using two stationary pulse Doppler radars. Using range and Doppler speed, a walking person was to be tracked. Key elements, such as Kalman filtering, data association and triangulation has been implemented and tested in simulations and with real radar measurements.

The Kalman filter is used to filter out noise from the measurements and provide a better estimation of the position of the target. It is found that a extended kalman filter (EKF) is needed in this system. With the inclusion of radial velocity into the EKF, there is a considerable improvement in the estimated position.

Two data association methods proves to be crucial to the tracking system. The gating shows great synergy with the Kalman filter by using components calculated in the filter in its own gating functions. During the real tracking, the gate shows that it can remove false and really noisy measurements preventing these measurements from distorting the estimated track. The global nearest neighbour method shows its usefulness in the real radar tracking as well, being able to chose the measurement that is closest to the predicted position among many potential values.

The simulation and true radar tracking results has a lot of similar values. The simulation does give a better estimation of the track, which means the simulated noise is most likely smaller that for the real radars.

Individually, the simulated radars are able to follow the original track up to about 6m distance seen in figure 5.7(a) and 5.8(a). After triangulation, this distance is reduced to 4 m on each axis which accumulates to 5.66 m from origo, seen in figure 5.10.

The true radar tracking is able to follow the target up to about 4.5 m from origo, as seen in figure 6.4(c). At 4 m distance the estimated position is about 1 m away from

the original track. That is a big error, but it is about the same error that the simulation predicts. To improve the tracking and minimize the error there has to be less noise from the actual radars.

The proposed tracking system proves to perform well in both the simulations and for the real tracking, but at these given SNR levels it is not be beneficial to use the system to track targets. By improving the SNR in the radars themselves and following some of the proposed suggestions for further work, a robust system that can be used in many different tracking applications, may be developed.

Appendix A

Matlab Simulation Code

A lot of different codes and versions of those codes has been made in this thesis. It has been chosen to only include the most essential ones and also leave out the plot functions. In appendix A, the simulation codes will be presented.

A.1 Main code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Morten Aasheim, filter then gather with gating, GNN Triangulate       %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
close all
T=0.1;
for t=1:1
%% Values from the two sensors
out=XY_sim();                        % [x;y;vx;vy]
[d1, d2]=twoSensorSim(out);          % change radar pos here.make the original sim
                                     % into two sensors worth

%% Add realistic noise
x1_1=noise_gen(d1);
%x1_1=d1;  % test without noise
x1_2=noise_gen(d2);
%x1_2=d2;  % test without noise


%% gate, GNN and filter the two sensor readings
```

```
single=1;             % decides if we use more signals or just a single signal
L=length(x1_1);
if (single==1)
    x2_1=zeros(2,L);
    x2_2=zeros(2,L);
    x3_1=zeros(2,L);
    x3_2=zeros(2,L);
end
[Nearest_1,X_upd_1,X_next_1]=gate_GNN_filter(x1_1,x2_1,x3_1,single); % first sensor

[Nearest_2,X_upd_2,X_next_2]=gate_GNN_filter(x1_2,x2_2,x3_2,single); % second sensor

%% Gather the two sensor measurements
L=length(Nearest_1);
x_upd=zeros(1,L);
y_upd=zeros(1,L);
x=zeros(1,L);
y=zeros(1,L);

for i=1:L

    [x_upd(i), y_upd(i)]=angleTriangle(X_upd_1(1,i),X_upd_2(1,i),-0.5,0.5); %from 2 sensors
                                                                            %to one from origo
    [x(i), y(i)]=angleTriangle(Nearest_1(1,i),Nearest_2(1,i),-0.5,0.5); %from 2 sensors
                                                                            %to one from origo
end
```

A.2 Track simulation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Morten Aasheim simulation of movement of walking human
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function out=XY_sim()
%% Startup %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x_start=0;  %m
y_start=0;  %m
vx_start=0.1; %m/s
vy_start=0.1; %m/s
v_abs=sqrt(vx_start^2 + vy_start^2); %no more than 0.2 m/s
T=0.1; %step time s
turn=0; %degrees , counterclockwise

turn_start=10; %when the maneuver is starting seconds
turn_stop=20; %when the maneuver is stopping seconds
```

```
turn_rate=turn/(turn_stop-turn_start); %degrees/s
turn_rate_rad=turn_rate*(pi/180);
T_tot=50; % total amount of seconds

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Storage
L=T_tot/T;
x=zeros(1,L);
y=zeros(1,L);
vx=zeros(1,L);
vy=zeros(1,L);

%% simulate
vx_temp=vx_start;
vy_temp=vy_start;
x(1)=x_start;
y(1)=y_start;
vx(1)=vx_start;
vy(1)=vy_start;
rad=atan2(vy_start,vx_start); %start angle

for t=2:L
    if((t*T) >=turn_start && (t*T)<turn_stop)
        rad=(turn_rate_rad*T)+rad;
        vx_temp=v_abs*cos(rad);
        vy_temp=v_abs*sin(rad);
    end
    x(t)= x(t-1)+(vx_temp*T);
    y(t)= y(t-1)+(vy_temp*T);

    vx(t)=vx_temp;
    vy(t)=vy_temp;
end
out=[x;y;vx;vy];

end
```

A.3 Two Radars Track simulation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%                                                                  %%%
%%% Morten Aasheim, simulation with two sensors                      %%%
%%%                                                                  %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [d1, d2]=twoSensorSim(out)

%out=XY_sim();  % [x;y;vx;vy]
L=length(out);
%% Sensor position
s1=[-0.5,0];                    %[x,y]
s2=[0.5,0];

%% find distance to the sensors
d1=zeros(2,L);                  % [r,vr]
d2=zeros(2,L);
T=0.1;                          % seconds between samples

for i=1:L
    temp=[out(1,i), out(2,i)];

    d=temp-s1;
    d1(1,i)= sqrt(sum(d.^2));        %sensor 1 distance

    d=temp-s2;
    d2(1,i)=sqrt(sum(d.^2));         %sensor 2 distance
end
%% sensor velocity
for j=1:L-1
    % sensor 1
    d1(2,j)=(d1(1,j+1)-d1(1,j))/T; %speed in sensor 1
    % sensor 2
    d2(2,j)=(d2(1,j+1)-d2(1,j))/T; %speed in sensor 2
end
d1(2,L)=d1(2,L-1); %just make the last speed equal to the previous one...
d2(2,L)=d2(2,L-1);

end
```

A.4 Generate Noise

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Morten Aasheim , Generate noise based on distance from radar %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Test
%  out=XY_sim();
%  x(1,:)=out(1,:);
%  x(2,:)=out(3,:);
function y=noise_gen(x)     %take in a vector of range and velocity
```

```
pulse_length_ns=1;
dR=3e8*pulse_length_ns*1e-9/2; % range resolution

lambda=0.0429; % lambda=3e8/7GHz= 0.4929 m - RF radar wavelength
T=1;            % integrasjonstid i PulseDoppler (6 sekund dettype1 og 1 sekund dettype 2)
dV=lambda/(2*T); % speed resolution

L=length(x);
E_d=zeros(1,L);
E_v=zeros(1,L);
y=zeros(2,L);
for i=1:L
    R=x(1,i);
    SNR=10^((52-R*12)/10);        % function to decide linear SNR based on range

    sigma_d=dR/sqrt(2*SNR);       % make sigma for the range
    error_d=sigma_d*randn(1,1);   % random distance error with this sigma

    sigma_v=dV/sqrt(2*SNR);
    error_v=sigma_v*randn(1,1);
    y(1,i)=x(1,i)+error_d;
    y(2,i)=x(2,i)+error_v;
    E_d(i)=error_d;
    E_v(i)=error_v;
end

end
```

A.5 Gate, GNN and Kalman Filter

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Morten Aasheim, apply gating, GNN and filtering                %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Nearest,X_upd,X_next]=gate_GNN_filter(x1,x2,x3,single)
L=length(x1);
%% matrixes
T=0.1;                          % seconds per measurement
F=[1 T; 0 1];                   % transition matrix, Newton
s_x=0.2;                        % sigma r
s_v=0.1;                        % sigma v
Q=0.1*[(T^2)/3 T/2; T/2 1];     % Process noise,liten Q - lite avvik,
                                    % stor Q - stort avvik
%Q=1*[s_x 0; 0 s_v];
bUseVr=1;  % 0 - dont use vr measurement, 1 -use vr measurement
```

```
%% predict the first P_next
P_upd=0.1*[0.1 0; 0 0.1];
x_upd=1.0*x1(:,1);  % MULIG FEIL HER

%% the loop
X_next=zeros(2,L);
X_upd=zeros(2,L);
Nearest=zeros(2,L);
first=1;       % takes care of the first iteration skip
if (single==1)
    non_gated=x1;
else
    non_gated=[x1;x2;x3];     %[x1;vx1,x2,vx2,x3,vx3];
end
nearest=x1(:,1); % just for the first iteration

for i=1:L
    if (first==0)
    nearest=gate_and_GNN(P_upd,H1,H2,F,Q,s_x,s_v,non_gated(:,i),x_pred);% find nearest neig
    end
    first=0;
    %% Kalman

   [x_upd,P_upd,x_pred,H1,H2]= ...
       extended_Kalman_pseudo_multivar(nearest(:,1),x_upd,P_upd,F,Q,s_x,s_v,bUseVr);
   X_next(:,i)=x_pred;
   X_upd(:,i)=x_upd;
   Nearest(:,i)=nearest;
end
end
```

A.6 Gate and GNN

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Morten Aasheim, gate and Global nearest neighbour  %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function nearest=gate_and_GNN(P_upd,H1,H2,F,Q,s_r,s_v,non_gated,x_pred)

[row, col]=size(non_gated);
gated=zeros(row,1);
%nearest=[NaN;NaN];
nearest=non_gated(:,1);
    %% Gate the position
```

```
    bound=1e20;    % set so high it will pass no matter what the first time
    for k=1:2:row                          % go through the three different xes
            P_pred=F*P_upd*transpose(F)+Q;
            %% Range gate
            S_r=H1*P_pred*transpose(H1)+ s_r;     % predict S
            z_r=non_gated(k,1)-x_pred(1,1);       % the current residual in x
            dr2=z_r'*inv(S_r)*z_r;                % use predicted S
            Gr=4*s_r*inv(S_r); % Gate for distance
           % Gr=1.0e20;
            %% velocity gate
            S_v=H2*P_pred*transpose(H2);
            z_v=non_gated(k+1,1)-x_pred(2,1);
            dv2=z_v'*inv(S_v)*z_v;
            Gv=4*s_v*inv(S_v);                    % Gate for velocity
           % Gv=1.0e20;
            if(dr2<Gr && dv2<Gv)                  % keep value if d2 is within the gates
                 gated(k)=non_gated(k,1);            % x
                 gated(k+1)=non_gated(k+1,1);        % vx

            else                           % outside the gate, replaced with NaN
                gated(k)=NaN;
                gated(k+1)=NaN;
            end
            %% GNN
            if (dr2<bound)                 % the nearest point is found
                 nearest(1,1)=gated(k);
                 nearest(2,1)=gated(k+1);
                 bound=dr2;
            end
    end

end
```

A.7 Kalman Filter w/Radial Velocity

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Morten Aasheim, extended kalman filter w/radial velocity %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [x_upd, P_upd, x_pred, H1, Hk4]=...
    extended_Kalman_pseudo_multivar(z,x_prev,P_prev,F,Q,sigma_r,sigma_vr,bUseVr)

%% Predict
x_pred=F*x_prev; %2x1                      %State pred
```

```
P_est=F*P_prev*transpose(F)+Q; %2x2       %State pred. cov.


%% Calculations
h= x_pred; %2x1
H1=[1 0];
%% Check if the value is NaN
if (isnan(z))
    x_upd=x_pred;
    P_upd=P_est;
    Hk4=H1;         % So velocity gate can be used even if the value is NaN
    return;
end                   %1x2
%% Update
z_upd=z(1,1) -h(1,1); %1x1

S=H1*P_est*transpose(H1)+ sigma_r; %1x1       % Innovation cov.
K=P_est*transpose(H1)*inv(S);%2x1             % Kalman gain

x_upd=x_pred+ K*z_upd;%2x1                % Updated state estimate
P_upd=P_est-K*S*K';  %2x2 JRP             % Updated state covariance

if (bUseVr==0) %if we only want to use x, not Vr
 x_pred=F*x_upd;  % not pseudo      % State prediction to next time
 Hk4=H1;
 %P_pred=F*P_upd*transpose(F)+Q;       % not pseudo      % State pred. cov. to next time
 return;
end

%% Radial velocity update
xk3=x_upd;     %temporal storage
Pk3=P_upd;     %temporal

Yk_k=xk3(1,1);          % x values
Ypk_k=xk3(2,1);         % vr values
DirCosk_k=(Yk_k'*Yk_k)^(-0.5)*Yk_k';             % Directional cosine, eq 11
I=eye(1);                                        % identity matrix
Ak=(Yk_k'*Yk_k)^(-0.5)*(I-DirCosk_k'*DirCosk_k); % text after eq 14
Bk=[zeros(1,1) 0.5*Ak;0.5*Ak zeros(1,1)];        % 2x2, eq 15
Ck=Bk*Pk3; %2x2
Hck2=[(Ak*Ypk_k)' DirCosk_k];                    % text after eq 17
Hk4=Hck2;

l=length((diag(Ck))); %length of diagonal
sum_cij=0.0;
for i1=1:l %2 atm
  for j1=1:l
    sum_cij=sum_cij+Ck(i1,j1)^2;      % text after eq 17
```

```
  end
end
Rck2=sigma_vr^2+2*sum_cij;            % eq 18
sigma2_4=Rck2;
Sk4=(Hk4*Pk3*Hk4'+sigma2_4);          %eq 30, innovation cov
Kk4=Pk3*Hk4'/Sk4;                     %eq 30, kalman gain

trace_Ck=trace(Ck);              % the diagonal vector of Ck
Zck=z(2)-trace_Ck;               % text after eq 17

xk4=xk3+Kk4*(Zck-Hck2*xk3); %eq 28, updated state estimate
Pk4=Pk3-Kk4*Sk4*Kk4';       %eq 29, updated cov,Pk4=Pk3-Kk4*Hck2*Pk3;?

x_upd=xk4;                  % new x with radial velocity included
P_upd=Pk4;                  % new P with radial velocity included

x_pred=F*x_upd;              % State prediction to next time

end
```

A.8 Triangulation

```
% find the angles of the triangle given the side lengths
% d1 and d2 are lengths of the "legs", s1 and s2 are the koorrdinates of the sensors
function [x,y]=angleTriangle(d1,d2,s1,s2)

d0=abs(s1)+abs(s2); %s1 and s2 are the koordinates of the two sensors on the x axis
d=d0/2;             % distance to origo
% %use the cosine rule
% a1=real(acos((d1^2 + d0^2 -(d2^2))/(2*d1*d0))); % in radians, only real values
% a2=real(acos((d2^2 + d0^2 -(d1^2))/(2*d2*d0))); % in radians


%% fix1, if value d2 is empty
if(isempty(d2))
    d2=NaN;
end
if(isempty(d1))
    d1=NaN;
end
%% Main
%x= (1/2)*(d2^2+d0^2-d1^2)/d0-d0/2;
%y= (1/2)*sqrt(4*(d2^2)-(d2^2+d0^2-d1^2)^2/(d0^2));
```

```
x=real(((d1^2)-(d2^2))/(4*d));
y=real(sqrt((d2^2)-(d^2)+ 2*(((d1^2) - (d2^2))/4) - ((((d1^2) - (d2^2))/(4*d))^2)));

%% Main
% if (d2<=d1 || isnan(d1))            % use the most accurate angle
%     x= s2-cos(a2)*d2;
%     y= sin(a2)*d2;
% elseif (d1<d2 || isnan(d2))
%     x= s1+cos(a1)*d1;
%     y= sin(a1)*d1;
% end


%% fix2, if x and y is empty
if(isempty(x))
    x=NaN;
end
if(isempty(y))
    y=NaN;
end

end
```

Appendix B

Matlab Real Radar Measurement Code

In appendix B, the real radar measurement codes will be presented and they are intended to perform the same task as the ones in appendix A.

B.1 Main Code, 1 Radar

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%                                                                    %%%%
%%%       Morten Aasheim , find values from radar                        %%%
%%%        Used with : Radar_gate_GNN_filter.m                           %%%
%%%                    Radar_gate_and_GNN.m                              %%%
%%%                    extended_Kalman_pseudo_radar.m                    %%%
%%%%                                                                    %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all
clear all
% function
%% original
%load('c:\Users\Morten\Documents\NoveldaRadarJRP\Morten\XtMatlab\ConceptTestPlatform2\File_
%% Test
load('c:\Users\Morten\Documents\NoveldaRadarJRP\Morten\XtMatlab\ConceptTestPlatform2\File_A
Ndet2=length(AccumDetList2); % decided to use detlList2
onlyHighestSNR=0;            % 1-only use the value with the highest SNR, 0- use all


% for i=1:Ndet2-2                % makes the sets start at the same time
%     AccumDetList2{i}=AccumDetList2{i+2};
```

```
% end
[Nearest,X_upd,X_next]=Radar_gate_GNN_filter(AccumDetList2,onlyHighestSNR);

%% for the legend only
figure();
plot(-1,1.5,'bo');
hold on
plot(-1,1.5,'go');
hold on
plot(-1,1.5,'r+');
%% Lets plot stuff
plot_trix=0;
for i=1:Ndet2-6          % chose what values to plot
    time=AccumDetList2{i}.Time+1;          % It starts at 0

%% plots all the possible values
rangevec=AccumDetList2{i}.DetList2.Range_m;
if (isempty(rangevec)==false)          % if it is not empty
    len=length(rangevec);
    index=1:len;
    plot(time-plot_trix,rangevec(index),'bo','linewidth',2);
    hold on
end

%% plots the nearest
time=AccumDetList2{i}.Time+1;          % It starts at 0
if (isempty(Nearest{1,i})==false)      % if its not empty
    plot(time-plot_trix,Nearest{1,i},'go','linewidth',2)
end
hold on
 %% Plots the filtered
plot(time-plot_trix,X_upd(1,i),'r+','linewidth',2);
hold on
end
a=axis;
axis([0 a(2) a(3) a(4)])
h=legend('Radar measurements','Nearest radar measurements','Filtered measurement');
%h=legend('Radar measurements','Nearest radar measurement');
set(h,'Position',[0.4044 0.1317 0.3787 0.1517])
set(h,'Fontsize',20)
xlabel('Time [s]','Fontsize',20)
ylabel('Distance [m]','Fontsize',20)
set(gca,'Fontsize',20);
%end
```

B.2 Main Code, 2 Radars

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%                                                                 %%%%
%%%       Morten Aasheim , find values from two radars                %%%
%%%        Used with : Radar_gate_GNN_filter.m                        %%%
%%%                    Radar_gate_and_GNN.m                           %%%
%%%                    extended_Kalman_pseudo_radar.m                 %%%
%%%%                                                                 %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all
clear all


%% Load radars , gate, GNN and Filter
%% R1
%load('c:\Users\Morten\Documents\NoveldaRadarJRP\Morten\XtMatlab\ConceptTestPlatform2\File_
load('c:\Users\Morten\Documents\NoveldaRadarJRP\Morten\XtMatlab\ConceptTestPlatform2\File_A
Ndet1=length(AccumDetList2); % decided to use detlList2
onlyHighestSNR=0;            % 1-only use the value with the highest SNR, 0- use all

tempR1=AccumDetList2;        % Length of R1
%% R2
%load('c:\Users\Morten\Documents\NoveldaRadarJRP\Morten\XtMatlab\ConceptTestPlatform2\File_
load('c:\Users\Morten\Documents\NoveldaRadarJRP\Morten\XtMatlab\ConceptTestPlatform2\File_A
Ndet2=length(AccumDetList2); % decided to use detlList2. Length of R2
Diff=Ndet1-Ndet2;
for i=1:Ndet2-2                % makes the sets start at the same time
    AccumDetList2R2{i}=AccumDetList2{i+2};
    AccumDetList2R1{i}=tempR1{i+Diff-1+1};   % remember the .time is still be off
    AccumDetList2R1{i}.Time=AccumDetList2R1{i}.Time-Diff+1;     % time is fixed
end

%% R1
[Nearest_1,X_upd_1,X_next_1]=Radar_gate_GNN_filter(AccumDetList2R1,onlyHighestSNR);
%% R2
[Nearest_2,X_upd_2,X_next_2]=Radar_gate_GNN_filter(AccumDetList2R2,onlyHighestSNR);

%% Triangulate
X_upd=zeros(1,Ndet2);
Y_upd=zeros(1,Ndet2);
X=zeros(1,Ndet2);
Y=zeros(1,Ndet2);
%figure();
for i=1:Ndet2-2
    [x_upd, y_upd]=angleTriangle(X_upd_1(1,i),X_upd_2(1,i),0.3,-0.3); %from 2 sensors
```

```
                                                          % to one from origo
    [x, y]=angleTriangle(Nearest_1{1,i},Nearest_2{1,i},0.3,-0.3); %from 2 sensors
                                                          % to one from origo
    X_upd(i)=x_upd;
    Y_upd(i)=y_upd;
    X(i)=x;
    Y(i)=y;
%     plot(x_upd,y_upd,'r+','linewidth',2)
%     hold on
%     plot(x,y,'bo','linewidth',2)
%     hold on
end
```

B.3 Radar Gate, GNN and Kalman Filter

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%                                                                                  %%%%
%%%   Morten Aasheim,radar gate, GNN and filter with the actual radar                   %%%
%%%%      used in: XY_realRadar.m                                                     %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Nearest,X_upd,X_next]=Radar_gate_GNN_filter(AccumDet2,onlyHighestSNR)
%% matrixes
T=1;                            % 1 seconds per measurement, THIS is different from sim
F=[1 T; 0 1];                   % transition matrix, Newton
s_x=0.2;                        % sigma r
s_v=0.1;                        % sigma v
Q=0.1*[(T^2)/3 T/2; T/2 1];     % Process noise,liten Q - lite avvik,
                                % stor Q - stort avvik

bUseVr=1;  % 0 - dont use vr measurement, 1 -use vr measurement
Ndet2=length(AccumDet2);
X_next=zeros(2,Ndet2);
X_upd=zeros(2,Ndet2);
Nearest{2,Ndet2}=[];  % this is a cell array
for i=1:Ndet2
%% Find the values to be gated
    speedvec=AccumDet2{i}.DetList2.Speed_m_s;
    rangevec=AccumDet2{i}.DetList2.Range_m;
    SNRvec=AccumDet2{i}.DetList2.SNR_dB;
    if (isempty(rangevec)==false)            % go in if there are values here
        [~,predIndex]=max(SNRvec);           % used to predict the first x_upd
        if (onlyHighestSNR)
            [~,indexvec]=max(SNRvec);
```

```
        else
            indexvec=1:length(rangevec);
        end
        non_gated=[rangevec(indexvec); -speedvec(indexvec)]; % speed negative because
                                                             % they were the wrong way
    else
    non_gated=[rangevec; -speedvec];    % these are supposed to be empty
    end

%% predict the first P_next
    if (i==1)                    % only go in here the first iteration
        P_upd=0.1*[0.1 0; 0 0.1];

        if (isempty(rangevec)==false)                          % enter if it has values
            x_upd=[rangevec(predIndex) ; -speedvec(predIndex)];  % minus because they were
            nearest=x_upd; % set nearest for first iteration

        else
            x_upd=[1.9; 0.1];    % supposed to be empty, if not set reasonable values
            nearest=non_gated;   % its empty
        end
    end
%% Gate and GNN
    if (i~=1)                                                % dont go in the first iteration
        nearest=Radar_gate_and_GNN...                        % gives empty or 1 value(x;vx)
            (P_upd,H1,H2,F,Q,s_x,s_v,non_gated,x_pred);  % find the nearest neighbour
    end

%% Kalman filter
    [x_upd,P_upd,x_pred,H1,H2]= ...
       extended_Kalman_pseudo_radar(nearest,x_upd,P_upd,F,Q,s_x,s_v,bUseVr);
    X_next(:,i)=x_pred;
    X_upd(:,i)=x_upd;
    if (isempty(nearest))            % if there is no values here
        Nearest{1,i}=nearest;        % this is now a cell, because some values are empty
        Nearest{2,i}=nearest;
    else
        Nearest{1,i}=nearest(1,1);
        Nearest{2,i}=nearest(2,1);
    end


end %for
end %function
```

B.4 Radar Gate and GNN

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%                                                                          %%%%
%%%  Morten Aasheim, gate and Global nearest neighbour, with the actual radar   %%%
%%%%            used in: Radar_gate_GNN_filter.m                              %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function nearest=Radar_gate_and_GNN(P_upd,H1,H2,F,Q,s_r,s_v,non_gated,x_pred)
%% Empty
[~,len]=size(non_gated);   %mulig feil her   !!!!!!!!!!!!!!!%!%!%!%%!
if (isempty(non_gated))              % go in if its empty
    nearest=non_gated;  % if there are no values, then just pass that on
    return;
end

%% Gate the position
bound=1e20;     % set so high it will pass no matter what the first time
for i=1:len
    P_pred=F*P_upd*transpose(F)+Q;
    %% Range gate
    S_r=H1*P_pred*transpose(H1)+ s_r;     % predict S
    z_r=non_gated(1,i)-x_pred(1,1);       % the current residual in x
    dr2=z_r'*inv(S_r)*z_r;                % use predicted S
    Gr=4*s_r*inv(S_r);                    % Gate for distance
   % Gr=1.0e20;                            % infinite gate
    %% velocity gate
    S_v=H2*P_pred*transpose(H2);
    z_v=non_gated(2,i)-x_pred(2,1);
    dv2=z_v'*inv(S_v)*z_v;
    Gv=4*s_v*inv(S_v);                    % Gate for velocity
   % Gv=1.0e20;                            % infinite gate
    if(dr2<Gr && dv2<Gv)                  % keep value if d2 is within the gates
        gated=non_gated(:,i);
    else                                  % outside gate
        gated=[];
    end
    %% GNN
    if (dr2<bound)                 % the nearest point is found
        nearest=gated;
        bound=dr2;
    end
end
end
```

B.5 Radar Kalman filter

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%                                                                                  %%%%
%%%   Morten Aasheim, extended kalman filter w/radial velocity, with the actual radar %%%
%%%%   used in: Radar_gate_GNN_filter.m                                               %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [x_upd, P_upd, x_pred, H1, Hk4]=...
    extended_Kalman_pseudo_radar(z,x_prev,P_prev,F,Q,sigma_r,sigma_vr,bUseVr)

%% Predict
x_pred=F*x_prev; %2x1                    %State pred
P_est=F*P_prev*transpose(F)+Q; %2x2      %State pred. cov.


%% Calculations
h= x_pred; %2x1
H1=[1 0];
%% Check if its empty
if (isempty(z))
    x_upd=x_pred;
    P_upd=P_est;
    Hk4=H1;         % So velocity gate can be used even if the value is empty
    return;
end                   %1x2
%% Update
z_upd=z(1,1) -h(1,1); %1x1

S=H1*P_est*transpose(H1)+ sigma_r; %1x1      % Innovation cov.
K=P_est*transpose(H1)*inv(S);%2x1            % Kalman gain

x_upd=x_pred+ K*z_upd;%2x1                % Updated state estimate
P_upd=P_est-K*S*K';  %2x2 JRP             % Updated state covariance

if (bUseVr==0) %if we only want to use x, not Vr
 x_pred=F*x_upd;  % not pseudo      % State prediction to next time
 Hk4=H1;
 %P_pred=F*P_upd*transpose(F)+Q;       % not pseudo      % State pred. cov. to next time
 return;
end

%% Radial velocity update
xk3=x_upd;     %temporal storage
Pk3=P_upd;     %temporal
```

```
Yk_k=xk3(1,1);          % x values
Ypk_k=xk3(2,1);         % vr values
DirCosk_k=(Yk_k'*Yk_k)^(-0.5)*Yk_k';            % Directional cosine, eq 11
I=eye(1);                                       % identity matrix
Ak=(Yk_k'*Yk_k)^(-0.5)*(I-DirCosk_k'*DirCosk_k); % text after eq 14
Bk=[zeros(1,1) 0.5*Ak;0.5*Ak zeros(1,1)];       % 2x2, eq 15
Ck=Bk*Pk3; %2x2
Hck2=[(Ak*Ypk_k)' DirCosk_k];                   % text after eq 17
Hk4=Hck2;

l=length((diag(Ck))); %length of diagonal
sum_cij=0.0;
for i1=1:l %2 atm
  for j1=1:l
    sum_cij=sum_cij+Ck(i1,j1)^2;      % text after eq 17
  end
end
Rck2=sigma_vr^2+2*sum_cij;            % eq 18
sigma2_4=Rck2;
Sk4=(Hk4*Pk3*Hk4'+sigma2_4);          %eq 30, innovation cov
Kk4=Pk3*Hk4'/Sk4;                     %eq 30, kalman gain

trace_Ck=trace(Ck);              % the diagonal vector of Ck
Zck=z(2)-trace_Ck;               % text after eq 17

xk4=xk3+Kk4*(Zck-Hck2*xk3); %eq 28, updated state estimate
Pk4=Pk3-Kk4*Sk4*Kk4';       %eq 29, updated cov,Pk4=Pk3-Kk4*Hck2*Pk3;?

x_upd=xk4;                  % new x with radial velocity included
P_upd=Pk4;                  % new P with radial velocity included

x_pred=F*x_upd;              % State prediction to next time

end
```

B.6 Triangulation

```
% find the angles of the triangle given the side lengths
% d1 and d2 are lengths of the "legs", s1 and s2 are the koorrdinates of the sensors
function [x,y]=angleTriangle(d1,d2,s1,s2)

d0=abs(s1)+abs(s2); %s1 and s2 are the koordinates of the two sensors on the x axis
d=d0/2;             % distance to origo
% %use the cosine rule
```

```
% a1=real(acos((d1^2 + d0^2 -(d2^2))/(2*d1*d0))); % in radians, only real values
% a2=real(acos((d2^2 + d0^2 -(d1^2))/(2*d2*d0))); % in radians


%% fix1, if value d2 is empty
if(isempty(d2))
    d2=NaN;
end
if(isempty(d1))
    d1=NaN;
end
%% Main
%x= (1/2)*(d2^2+d0^2-d1^2)/d0-d0/2;
%y= (1/2)*sqrt(4*(d2^2)-(d2^2+d0^2-d1^2)^2/(d0^2));

x=real(((d1^2)-(d2^2))/(4*d));
y=real(sqrt((d2^2)-(d^2)+ 2*(((d1^2) - (d2^2))/4) - ((((d1^2) - (d2^2))/(4*d))^2)));

% if (d1 > d2)
%     x=abs(x);
% else
%     x=-abs(x);
% end
% y=abs(y);
%% Main


%% fix2, if x and y is empty
if(isempty(x))
    x=NaN;
end
if(isempty(y))
    y=NaN;
end
end
```

Appendix C

Matlab Triangulation Error

This function was provided by the supervisor at Novelda AS, but few minor tweeks has been done.

C.1 Triangulation Concept

```
d=1.0; % distance between radar 0 and 1. Assumed placed at (0,0) and (0,d)
pulse_length_ns=1;
SNR_dB=10;

xt=0;
yt=4;

% xt=1;
% yt=0;

Nsigma=3;

%%%%%%%%%%%%%%%%%%%%%%

dR=3e8*pulse_length_ns*1e-9/2; % range resolution
SNR=10^(SNR_dB/10);
sigma_r=dR/sqrt(2*SNR);

R0=sqrt((xt-(-d/2))^2+yt^2);
R1=sqrt((xt-d/2)^2+yt^2);

xt= (1/2)*(R1^2+d^2-R0^2)/d-d/2;
yt= (1/2)*sqrt(4*R1^2-(R1^2+d^2-R0^2)^2/d^2);
```

```
if (R0 > R1)
    xt=abs(xt);
else
    xt=-abs(xt);
end
yt=abs(yt);
xt
yt

xvec0n=[];
yvec0n=[];
xvec0=[];
yvec0=[];
xvec0p=[];
yvec0p=[];

xvec1n=[];
yvec1n=[];
xvec1=[];
yvec1=[];
xvec1p=[];
yvec1p=[];

for (theta_deg=0:0.01:180)
    theta=theta_deg*pi/180;

    xvec0n=[xvec0n abs(R0-Nsigma*sigma_r)*cos(theta)-d/2];
    yvec0n=[yvec0n abs(R0-Nsigma*sigma_r)*sin(theta)];
    xvec0=[xvec0 R0*cos(theta)-d/2];
    yvec0=[yvec0 R0*sin(theta)];
    xvec0p=[xvec0p abs(R0+Nsigma*sigma_r)*cos(theta)-d/2];
    yvec0p=[yvec0p abs(R0+Nsigma*sigma_r)*sin(theta)];

    xvec1n=[xvec1n abs(R1-Nsigma*sigma_r)*cos(theta)+d/2];
    yvec1n=[yvec1n abs(R1-Nsigma*sigma_r)*sin(theta)];
    xvec1=[xvec1 R1*cos(theta)+d/2];
    yvec1=[yvec1 R1*sin(theta)];
    xvec1p=[xvec1p abs(R1+Nsigma*sigma_r)*cos(theta)+d/2];
    yvec1p=[yvec1p abs(R1+Nsigma*sigma_r)*sin(theta)];
end

figure(1);
plot(xvec0n,yvec0n,'b--',xvec0,yvec0,'b',xvec0p,yvec0p,'b--',xvec1n,yvec1n,'r--',xvec1,yvec
grid on;
axis([-2 2 0 5]);

figure(2);
```

```
plot(xvec0n,yvec0n,'b',xvec0p,yvec0p,'b',xvec1n,yvec1n,'r',xvec1p,yvec1p,'r',xt,yt,'g*','Ma
grid on;
axis([-4 4 0 7]);
title('Triangulation uncertainty area for 2 radars');
xlabel('x [m]','fontsize',20);
ylabel('y [m]','fontsize',20);
set(gca,'fontsize', 20);
```

Bibliography

[1] Heinrich Hertz. http://www.famousscientists.org/heinrich-hertz/. [Online; accessed Feb-2015].

[2] Guglielmo Marconi. http://www.history.com/topics/inventions/guglielmo-marconi. [Online; accessed Feb-2015].

[3] Christian Huelsmeyer. http://www.radarworld.org/huelsmeyer.html. [Online; accessed Feb-2015].

[4] Chain Home Defense. http://www.radarpages.co.uk/mob/ch/chainhome.htm. [Online; accessed Feb-2015].

[5] S. Blackman and R. Popoli. Design and analysis of modern tracking systems. pages 157–167, 1999.

[6] J. Elfring, R. Janssen, and R. Molengraft. Data association and tracking: A litterature study. *RoboEarth*, pages 6–17, April 2010.

[7] X. R. Li and V. P. Jilkov. A survey of maneuvering target tracking: Dynamic models. *SPIE CONFERENCE ON SIGNAL AND DATA PROCESSING OF SMALL TARGETS*, pages 1–7, April 2000.

[8] P. H. Olsen. Multisensor tracking of two flying targets in formation. pages 1–130, 2010.

[9] X. R. Li and V. P. Jilkov. A survey of maneuvering target tracking: Measurement models. *SPIE CONFERENCE ON SIGNAL AND DATA PROCESSING OF SMALL TARGETS*, pages 1–6, July-August 2001.

[10] J. Wang, P. He, and T. Long. Use of radial velocity measurement in target tracking. *IEEE TRANSACTIONON AEROSPACE AND ELECTRONIC SYSTEMS*, 39(2): 401–413, April 2003.

[11] Supervisor at Novelda AS. [Conversation with supervisor Jan Roar Pleym].