University of Groningen **Faculty of Mathematics and Natural Sciences**

Department of Mathematics and Computing Science

Radar Image Fusion and Target Tracking

Dinne Bosman

June, 2002

RuG

Radar Image Fusion and Target Tracking

Dinne Bosman

Advisors:

ir. C.P. Valens
SODENA Company
Crach, France

dr. J.B.T.M. Roerdink
Department of Mathematics and Computing Science
University of Groningen

June 2002

Radar image fusion and target tracking

Dinne Bosman

June 2002

Contents

Acknowledgments

First of all I would like to thank all those people who supported me during my research.

I would like to thank the company Sodena and in particular Clemens Valens. Sodena presented my thesis subject, which proved to be very exciting, and offered me an interesting visit to France. I thank Clemens Valens for all his time spent in answering my questions. His and his wife's hospitality were really great during my (unfortunately short) stay in France.

While working on any thesis, planning is very important, as I have learned the hard way. Thanks go to my supervisor Roerdink at the RUG for the help he gave when things went less smooth.

Thanks go to Iwein Fuld. I very much appreciated the discussions about the physical nature of radar. I am a computer science student, and his insights in physics (his field) where invaluable.

Lastly I thank the people at Insource who where willing to loan me the 'movable' computer for my presentation and my visit to France.

Groningen,
May 2002

Chapter 1

Abstract

The sea traffic surrounding many large ports is becoming increasingly crowded. Traffic regulation is needed to allow even higher traffic densities and to avoid dangerous situations. The traffic regulation is managed from Vessel Traffic Surveillance (VTS) centers. In these centers radar images from radar posts in the vicinity of the port are analyzed. When a ship enters the surveillance zone, the new ship is tagged with an ID with which the ship can be identified during observation. Because each radar post in the VTS system will only cover a small area it is possible for the ship to move in regions that are being observed by different radars. When this happens the ship's ID must remain the same. Furthermore, when tracking ships with radar, highly robust tracking algorithms are needed, because the radar signal is often contaminated with a lot of noise. This can lead to a failure of ship detection or a wrong ID-tag to ship association. To cope with these problems a multiple hypothesis tracking (MHT) algorithm using Kalman filtering has been developed. An extensive radar-image simulation environment is also created to generate test scenarios for the tracking algorithm.

Chapter 2

Introduction: traffic management

The sea traffic surrounding many large ports is becoming increasingly crowded. Traffic regulation is needed to allow even higher traffic densities and to avoid dangerous situations. The traffic regulation is managed from Vessel Traffic Surveillance (VTS) centers. A VTS center acts just like the traffic regulation center on an airport. Several video cameras and radar stations are used to detect the positions of the ships and radio is used to communicate with them. There is one big difference: airplanes are equipped with modern active radar transponders, most ships are not. When an airplane is detected by radar, the airplane navigation system will actively send special information about the plane's identity to the traffic center. In a VTS center this identity information can be obtained from a database, but it is still a problem to associate the identity (ID) information with the correct radar detection. Furthermore each radar post in the VTS system will only cover a small area and it is possible for the ship to move in regions that are being observed by different radars. When this happens the ship's ID must remain the same. Ship tracking with radar, requires highly robust tracking algorithms, because the received radar echoes can be very hard to classify. For instance when the coverage regions of two radar posts overlap the tracking algorithm will have to deal with the fact that:

- there is no guarantee that the echo of the ship is actually visible in one (or both) of the images.
- when an echo is displayed, there is a possibility that the echo actually represents noise.
- it is possible that the echoes of the same ship are displayed at different positions in each of the images.
- the radars are not synchronized, thus the ship echo in the first image is already old when the ship is detected in the second image.

If the tracking algorithm fails to deal correctly with the situations mentioned above, this can lead to a failure of ship detection or a wrong ID-tag to ship association.

The most important drawback of most VTS systems is that such systems are very expensive, because of the used hardware. As a result VTS systems are only used in rich countries and in important areas. This thesis subject was inspired from a project of the company Sodena. Sodena is a French company specialized in navigation soft and hardware for the shipping industry. Sodena has developed a system

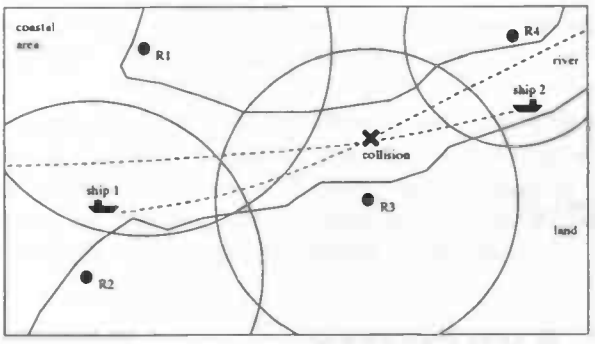


Figure 2.1: *Vessel Traffic Services (VTS) system sketch: R1, R2, R3 and R4 are radar observation posts. Ship 1 is observed by R1 and R2, ship 2 is observed by R4. After course extrapolation the system determines that the two ships will collide at the given point.*

with which radar images can be viewed with a normal PC. With this technology a VTS system can be implemented a lot cheaper. Such a VTS system can be used for example by smaller ports or by ports in third world countries witch have less money. At the start of the project Sodena already had the technology for a one-radar VTS system.

The assignment was to develop a robust fusion algorithm that fuses several (possibly overlapping) radar streams into one big radar stream. A tracking algorithm must be built which will have to extract the correct tracks from the fused stream.

We will start with an introduction on the operation of radar in general in chapter 3. This chapter introduces basic formulas that describe the physical laws which allow target detection with radar. Next the specific properties of each radar system component is described. The chapter concludes with information on some more exotic radar systems and by giving an example of a typical radar station that is used in a VTS system. In Chapter 4 a radar simulation software package is designed. The simulated radar images which we obtain with this package are used to test the fusion and tracking algorithm. The first part of the chapter derives the simulation formulas which in detail describe the physical nature of a radar system. Furthermore the computational algorithms are introduced which implement these formulas efficiently. The second part of the chapter will focus on the implementation of these algorithms. In Chapter 5 a robust fusion and tracking algorithm is developed. The tracking algorithm is a multiple hypotheses tracking (MHT) algorithm which uses Kalman filtering for target position prediction. To test such an algorithm simulated radar images are used. Implementation issues are discussed next. The simulation software is expanded with the fusion and tracking algorithm. At this point all algorithms have been implemented. A global framework is constructed in chapter 6 which separates the graphical user interface from the actual developed algorithms. The last chapter 7 gives a discussion on the problems which remain an issue in the simulation algorithm as well as in the fusion and tracking algorithms.

Chapter 3

Radar systems

At one point during the development of a VTS system, we will need to specify the radar type which is to be used. A VTS system for instance will have very different radar requirements when compared to other important radar applications like weather measurement and aircraft guidance systems. If there are also financial constraints on the system then tradeoffs will have to be made. In all cases the choices will have to be made very carefully. In order to make sensible decisions we have to know how radar works and which choices we have in adjusting the radar performance. In this chapter we will discuss important radar principles and the parameters that define the performance of the radar.

3.1 Target position measurement

In a VTS system we will use radar to detect targets like ships and buoys. A target is detected by the radar through the detection of echoes. The radar sends a powerful burst of radio energy, some of the energy will be reflected by the target and this reflected energy is measured by the radar receiver. The time between sending a receiving gives a measurement of the range of the target. So the range is a function of signal propagation delay:

$$R = \frac{cT}{2} \tag{3.1}$$

In this formula R is the range of the target, c is the speed of radio waves and T the time between sending and reception. The constant 2 is used because the radio wave has to make a round trip from radar to target. The angle θ between the target and the aerial is given by the direction of the aerial. Thus the coordinates of a target are measured by a tuple (R, θ) in polar coordinate form. In a so-called scanning radar targets are detected in 360 degrees by rotating the areal. The accuracy of the position measurements depends heavily on the transmitter, aerial and the receiver.

3.2 Target characteristics

The radar *cross-section* (RCS) is a characteristic of radar targets, which create an echo by scattering (reflecting) the radar EM wave. The RCS of a target is the equivalent area intercepting that amount of power, which when scattered isotropically produces at the receiver a density, which is equal to that scattered by the target itself:

$$\sigma = \lim_{R \to \infty} \inf \left[4\pi R^2 \frac{W_s}{W_i} \right]$$

where W_s and W_i are respectively the incident power density and scattered power density in W/m^2. The RCS value itself is in m^2.

It should be noted that the RCS has little in common with any of the cross-sections of the actual scatterer. However, it is very representative of the reflection properties of the target. It very much depends on the angle of incidence, on the angle of observation, on the shape of the scatterer, on the EM properties of the matter that it is built of, and on the wavelength of the radar.

3.3 Transmitter

3.3.1 Pulse repetition rate and pulse length

The transmitter produces pulses of radar energy at a regular interval. The number of radar pulses sent in one second is referred to as the *pulse repetition frequency* (PRF) or *pulse repetition rate* (PRR). Each of the radar pulses itself has a duration known as the *pulse length.*

3.3.2 Detection range

The minimum and maximum range at which the radar can detect targets are theoretically bound by the PRF and the pulse length.

Minimum range: The power of the pulse sent by a radar is vastly higher than the power of a received echo. The sensitive receive electronics would be damaged if they were turned on during the sending of a pulse, consequently the radar cannot receive echoes while it is sending a pulse, so if a target is close enough its echo could be lost because the radar is still sending. This gives rise to the theoretical minimum target range:

$$R_{min} = \frac{c\tau_{pulse}}{2} \tag{3.2}$$

In this formula τ_{pulse} denotes the duration of a single radar pulse.

Maximum unambiguous range The PRF defines a maximum unambiguous range:

$$R_{max} = \frac{c}{2PRF} \tag{3.3}$$

If we choose R_{max} inappropriate small (or the PRF consequently very high) in comparison to the sensitivity of the receiver, false echoes might appear. A target beyond R_{max} then reflects an echo. When the echo arrives at the receiver one or more new pulses will already have been sent. The range of the false echo depends on the range of the real target but is measured as a value anywhere between R_{min} and R_{max}

Maximum range The real detection range of course depends on the target, radar energy etc. The following equation is called 'the radar range equation':

$$R_{max} = \left(\frac{G^2\sigma_s\lambda^2 E_t}{(4\pi)^3 FkT}\right)^{\frac{1}{4}} \tag{3.4}$$

In this equation E_t is the transmitted energy, F is the receiver noise figure, k is the Boltzmann constant, T the temperature in Kelvin, G the antenna gain, λ the radar wavelength, and σ_s the scattering cross section of the target. The formula gives the range from which a target with an equivalent scattering cross section σ_s will supply a radar with a signal equal to the thermal noise energy in a time-resolution cell.

Range resolution There is a minimum range between two targets if they are to be detected by radar as two separate objects. This resolution is given by:

$$\Delta R = \frac{c\tau_{pulse}}{2} \tag{3.5}$$

This formula will only be exact if the radar waveform would be perfect square pulse. See section 3.3.4 for details.

Note:

- In addition a pulse is defined by its power and shape. If more power is used to send the wave more radar energy will be reflected by a target. In most cases the received noise energy will increase less.
- Long pulse lengths increase the probability of detection. Most amplifiers are also more effective in amplifying long pulses than short pulses. There is thus a trade off between pulse length and R_{min}.
- If targets of considerable range are to be detected, a low PRF and long pulse length apply. Nearby targets can be detected with a high PRF and short pulse length.
- Short pulses decrease detection probability, but a high PRF will cause the radar to send more pulses to the target and thereby increases the detection probability. It depends on the statistical detection properties of a target whether a high PRF or a short pulse length is to be used.

3.3.3 Radar frequency

There are two kinds of generally used frequency bands for radar. The X-band which contains the frequencies between 9300 and 9500 MHz (wavelength about 3 cm) and the S-band which contains frequencies between 2900 and 3100 Mhz (wavelength about 10 cm). The frequency chosen depends on the size of the aerial. As shown in section 3.4 long wavelengths require large aerials. Short wavelengths alow better resolution, long wavelengths are less affected by rain and increase maximum range detection. At very short wavelengths however the energy of the radio waves is absorbed by the atmosphere. In many cases the properties of X-band and S-band radar are complementary and both X- and S-band systems are used simultaneously.

3.3.4 Waveform

The waveform determines the resolution of the radar in range. The shorter the duration of the pulse the better the range resolution. If the pulse has a large length then the endpoint of the pulse reflection from one target can overlap with the start of the reflection of another target.

At a first glance it seems that obtaining both high range resolution and using a radar system with long pulses is impossible. In many modern radar systems the sent pulse has a special shape to alleviate this restriction. This pulse shapening scheme is called pulse compression. Pulse Compression allows a high range resolution while keeping reasonable pulse lengths which in turn increase detection. First the transmitted pulse is modulated time dependently. When an echo returns it is matched to the transmitted signal (at the time of the sent pulse). The difference is then used compute the range. The range resolution ΔR now depends on the resolution possible within the difference value. If the difference value is independent of the pulse duration τ_{pulse} then we can choose the pulse length according to other needs. Many of these pulse compression schemes like Barker codes, FM chirps and Huffman codes are described in [4].

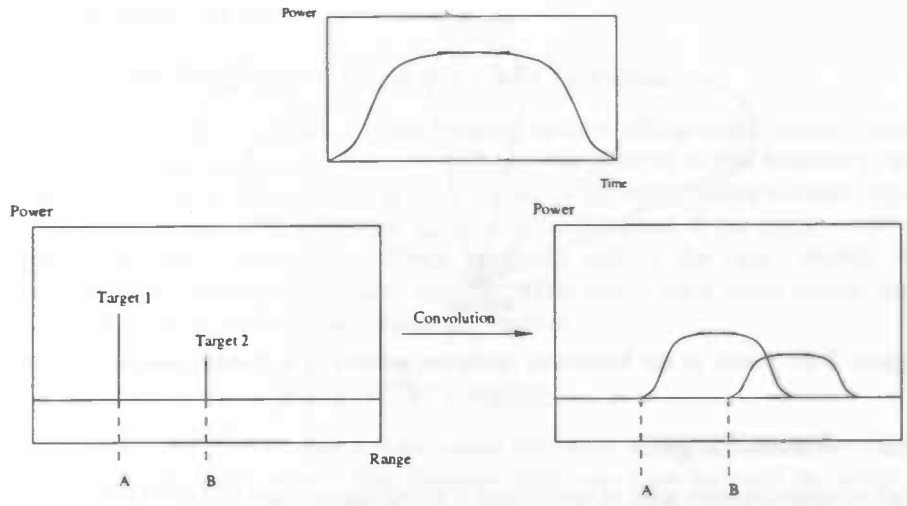


Figure 3.1: *Pulse duration determines range resolution.*

3.4 Aerial

The aerial is a major component of a radar system. For each radar application an aerial can be modelled according to the needs of the application. If the costs of a system are restricted, then an aerial like the slotted waveguide antenna, which is available in standard packages, is practical. The slotted waveguide antenna has a rectangular aperture, which has certain properties which are discussed below.

3.4.1 Radiation intensity

The aperture of the aerial determines the shape of the radar beam. Horizontally the beam must be narrow to ensure accurate angular measurement. Vertically, the beam should be wide enough to maximise detection probability, but small enough to avoid detecting low flying airplanes. Usually aerials have a rectangular aperture from which the radar waves exit in phase (e.g. parabola and slotted waveguide). In this case the radiation pattern will contain a main lobe and side lobes. Side lobes result from the fact that at some places the radar waves will arrive in anti-phase and interference will cause them to be zero. Ideally the side lobes should not exist because they will degrade angular measurement: A target which exists in a side lobe will contribute to the received signal even though the antenna is not positioned to the target. Vertically an interference pattern will also exist, because the main lobe will be partially reflected by sea or land. Together these reflected waves will form an interference pattern.

The angular width of the main lobe is called the *half power horizontal beam width* (HBW), which can be calculated by:

$$HBW = 70\frac{\lambda}{w} \tag{3.6}$$

where λ is the transmitted wavelength and w the aperture width. As there are limits on the maximum width of an aperture due to constructional constraints, there are also limits on the maximum wavelength if a specific HBW is chosen.

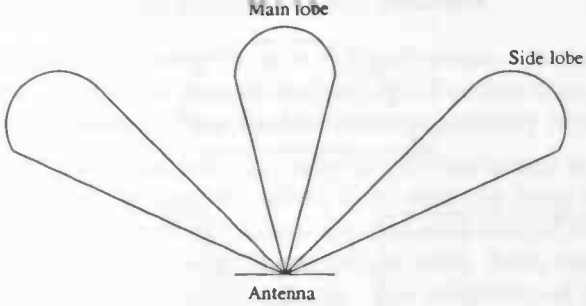


Figure 3.2: *Sketch of the horizontal radiation pattern of a slotted waveguide antenna*

3.4.2 Antenna gain

A high antenna receive gain is important if small targets are to be detected at large ranges. The formula for antenna gain is:

$$G_r = A_r \frac{4\pi}{\lambda^2} \tag{3.7}$$

where G_r is the gain of the receive aperture and A_r the receive aperture area.

3.4.3 Rotational rate

The radar antenna will scan 360 degrees. The time in which a complete scan must be finished determines the rotational speed of the antenna. The radar image has to be updated at a specific rate, a higher update rate will cause the image to be more responsive to changes. The update rate however must not be too high because then each target is shorter illuminated by the radar beam, and detection probability degrades. If N is the rotation speed of the aerial then the number of pulses S that strike a point target is:

$$S = PRF \frac{HBW}{6N} \tag{3.8}$$

The factor 6 is just a conversion factor, HBW is given in degrees and N in rpm. Note that not every strike will actually lead to detection.

3.5 Receiver

The received echoes have very little power and will have to be amplified. However, amplification at high frequencies as those from radar is not efficient. In order to make amplification possible the frequency of the received signal is transformed down to *intermediate frequency* (IF) by a component known as the *mixer*. The IF output of the mixer is then led through a *multi-stage amplifier*. There are two kinds of amplifiers, older systems use linear amplification while newer systems use logarithmic amplification. A shortcoming of linear amplification is that weak target echoes are easily masked by noise, the factor between the power of a target echo signal and strong noise could be 10000. In logarithmic amplification the output will be proportional to the logarithm of the input, this will reduce the dynamic range of the amplifier output, e.g. $\log(10000) = 4$.

3.6 Radar types

3.6.1 Moving target indicator (MTI) radar

MTI radars are used to detect moving targets against background clutter (noise). The basic principle used is that clutter will only be present in the frequency spectrum at a relatively narrow and low frequency band. By using filters at these regions and using the Doppler shift a moving target can be detected, if the target's velocity is higher than that of the clutter which generally will be the case. Mostly MTI radars will use a normal pulsed radar system. After acquiring a radar image, signal processing is used to remove the stationary clutter.

3.6.2 Continuous wave (CW) radar

In *continuous wave* (CW) radar a continuous sine wave is sent. The received signal is compared to the sent wave. The Doppler shift can then be used to determine the velocity of a target. Because of the continuous wave the CW radar will have no range discrimination. In order to measure the range of a target, the sent radar wave is modified by introducing frequency modulation.

3.6.3 Pulse Doppler radar

A pulse-Doppler radar combines advantages of CW and pulse radars. Instead of using a continuous wave a pulse train is sent. The Doppler shift will introduce a widening or shortening effect on the interval between pulses in the pulse train. Because pulse trains are used extra precautions have to be taken in order not to receive echoes from beyond R_{max} (see 'maximum range' in 3.3.2). Pulse-Doppler systems are mainly used in military applications.

3.7 Specification

There are a number of issues when choosing the radar sensors which are of a less technical nature, these globally define the radar sensors that can be used by the system.

Cost Radar sensors can be designed with particular properties which are useful for VTS systems. If there are budget constraints then customizing the sensors is usually only partially possible. Besides the number of sensors and the type of radar used are bound by cost constraints.

Legislation Collision avoidance is one of the tasks of the VTS. To cover liability issues legislation has been developed to ensure a certain performance standard for VTS systems. The standard gives the bounds of some of the radar parameters (like the rotational rate, minimum range, maximum range) which can be used.

VTS requirements There are different types of VTS systems, namely: coastal VTS, port VTS and river VTS systems. The terrain topology differences in each type of VTS system define the usable radar type and the usable radar parameters.

Sensors The fusion of two radar images is achieved by merging their overlapping sections in space. As will be shown in ** the gain in accuracy in the fused image is highest when each of the images alone have moderate accuracy. In addition much better results are possible when the two images are complementary (e.g. the use of both X-band and S-band images).

Fusion The goal of the image fusion has to be defined.

- Image fusion is needed because the performance of each sensor alone is too low. Thus more sensors are used which return data of the same area. Signal processing is then used to increase accuracy in the fused image.
- Fusion is needed because the range of only one sensor is not high enough. More sensors are used to collect data over the total surveillance area. There is only a small overlap in the detection area of the sensors. Signal processing is used to align the sensors exactly. After this step objects are extracted from the complete image. The trajectory of these these object are then estimated by a tracking algorithm.

Cost A pulse radar is used. A cost effective radar will make use of the so called slotted waveguide aerial. These systems can be delivered 'off the shelf'.

Legislation The legislation requires a bandwidth of at least 2 degrees, a maximum update time of 2 sec., and a minimum detection range of 50 m.

VTS requirements A river VTS system is targeted. It will need high resolution, small target detection and water or weather clutter removal.

Sensors First, the system will be built with one type of sensors. After assessment of the system performance other types of sensors can be taken into account.

Fusion Both fusion goals will require correct alignment of the sensors. First a river VTS system with just enough sensors for coverage is considered.

3.8 Example

In this section we will make a simple design of a typical radar system which could be used to track ships at a river or sea. The designed system will have good performance in targets close to the radar. It has a small horizontal beam width which will result in higher resolution and less clutter. Because of the short pulse length the radar will be less suited for detection of distant targets.

- The radar will transmit at X-band frequency: 9500 Mhz.
- The horizontal beam width HBW should be about 0.1 degree: $HBW = 0.1°$.
- The radar will use 1 strike : $S = PRF\frac{HBW}{6N} = 1$.
- Each radar image is updated in two seconds. The rotational rate: $N = 24$ rpm.
- The values for HBW and S result in a PRF of: $S\frac{6N}{HBW} \approx 1500$ Hz.
- The PRF value results in a maximum unambiguous range: $R_{max} = \frac{c}{2PRF} =$ 100 km.
- The minimum range R_{min} at which targets have to be detected is 30 m.
- A minimum range of 30 m gives a τ_{pulse} value : $2\frac{R_{min}}{c} = \tau_{pulse} = 200\,nsec.$.
- The minimum distance of two separately detected targets depends on τ_{pulse} : $\Delta R = \frac{c\tau_{pulse}}{2} = 30$ m.
- The antenna aperture width can be about 3 m.

- The antenna gain can be looked up in a table, the two way gain of the antenna will be about 20 dB.
- The maximum range at which we want to detect targets is 12 km.
- The noise figure F is specific for each receiver. A typical value is 5 dB.
- The power of a low cost VTS radar will be typically about 25 kW.
- A typical value for the gain of the matched receiver filter is about 21 dB.

Chapter 4

Radar simulation

In order to test the fusion algorithm once it is developed, we will need realistic radar images. We could record images directly from real radar receivers. This will take a lot of time and we will have to make careful preparations to ensure that the right test scenarios are obtained. A more flexible approach is to build a simulation model. With such a model we can set up different test scenarios in only a fraction of the time needed to take real measurements.

4.1 Requirements

Literature gives us two kinds of radar simulation models. One model is directly based on calculation of the electro-magnetic (EM) wave propagation using Maxwell's equations. This model can give very realistic results like Doppler effects, wave extinction and target shape effects, because the physical behaviour is approximated. Unfortunately the simulation space needs a very fine grid resolution which is about 1/20 of the wavelength used. In marine radar simulation the simulation space is several square kilometers in size and the application of this model would require prohibitively large amounts of memory. Another simulation model uses the radar range equation. In this model targets and clutter are modelled by their radar cross-section reflectivity. Statistical distributions are used to model reflectivity changes over time. The result of this simulation model can be realistic if the statistical properties of the phenomena to be simulated have been described accurately. In this chapter simulation algorithms are developed and implemented based on this model. The mathematical simulation model is from [17]. This source is however a bit dated (1976). As result most of the described algorithms have been designed with severe computation and memory constraints. In this chapter new algorithms which are designed with modern computation capabilities in thought are developed. A last possibility is to use a 'quick and dirty' method to generate eye-pleasing radar images which are not really based on physics. This method is of course computationally less demanding than the other two methods. When implementing such a method the programmer should be aware of all anomalies that can occur in real radar images as these anomalies will not be generated by the model itself.

The next data flow diagram shows which components are needed in the simulation. The sea chaff and fog components are not simulated in this project because we did not have enough time to focus on them. These phenomena seem however to be simulated in much the same way as rain. The target component will be simplified somewhat (see section 4.2.2).

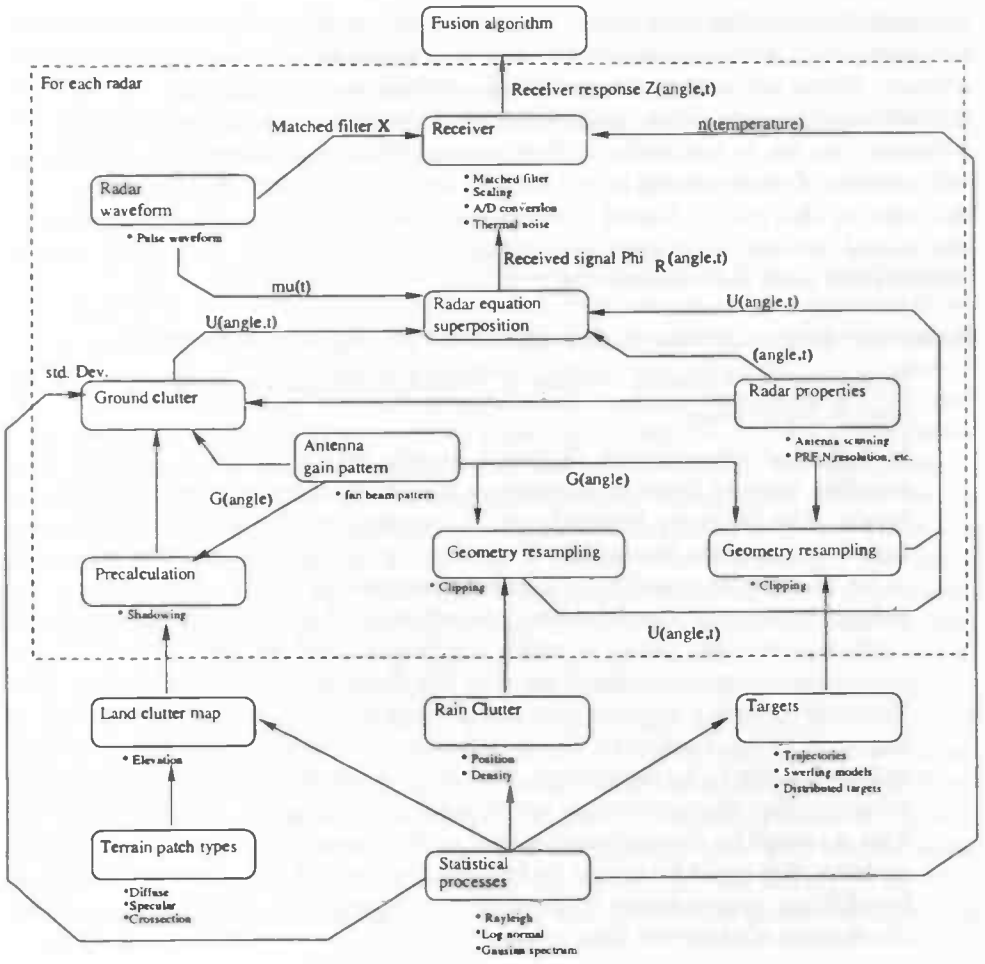


Figure 4.1: *The model used to simulate several radar stations: the labels next to the arrows refer to mathematic formulae in the next section.*

First the components that interact with the radar are described in sections 4.2.1 (ground clutter), 4.2.2 (targets) and 4.2.3 (rain clutter). Next the sections 4.2.4 (range equation), 4.2.5 (antenna pattern), 4.2.6 (radar waveform) and 4.2.7 (receiver) will model the components of the radar itself. The last section 4.2.8 of this chapter completes the simulation model by 'glueing' the different components together.

4.2 Mathematical modelling

4.2.1 Ground clutter

Ground clutter is the result of reception of the radar waves which are reflected by the terrain. If we have a elevation map of the terrain, the ground clutter signal can be simulated. Bitmaps, which are stored in a Cartesian coordinate system, cannot be used efficiently because, as most radar stations will cover large areas, this would cost large amounts of memory. E.g. if we have a radar range swath of $50\,km$, the total area is about $7800\,km^2$, a pixel area of 5 by 5 m. would create a map of 312 Mb, still this map would only contain byte values! A solution for this problem is to create a map which only contains data in important areas. The sea for instance

is mostly flat and this area might be defined by less data. In the areas between the given points which are thus not evenly distributed we will need an interpolation scheme. There are several interpolation schemes, but the different methods are divided into two categories: global and local methods. Global methods use the complete data set to interpolate a query point. With local interpolation methods, only a subset of observational points located near the new point are used to estimate the value at this point. Global methods are computationally expensive, which is why a local method seems more appropriate in this simulation. Among the various methods two seem more interesting:

Delaunay Triangulation In this scheme the set of points is triangulated.

When the the 3 nearest vertices of a triangle containing the query point are used to interpolate.

In a general triangulation there are usually several different triangulations possible, some of these triangulations introduce artifacts when interpolating height. The Delaunay triangulation is a method which obtains triangles that are as equi-angular as possible. The value for an interpolated point is ensured to be as close as possible to a known observation point. Furthermore this Delaunay property simplifies the interpolation: if we find the triangle which contains the query point at which the height must be interpolated we can simply interpolate the height between the three vertices of the triangle. The Delaunay property ensures that the 3 vertices of the triangle are closest to the query point. Lastly the triangulation is not affected by the order of observational points to be considered. The main disadvantage of the triangulation scheme is that the surfaces are not smooth and may give a jagged appearance. This is caused by discontinuous slopes at the triangle edges and data points, a problem that could be solved by introducing shading (e.g. Gouraud shading). In addition, triangulation is generally not suitable for extrapolation beyond the domain of observed data points.

Inverse distance interpolation In this interpolation method, observational points are weighted during interpolation such that the influence of one point relative to another declines with distance from the new point. The weight is calculated by taking the reciprocal of the distance to some power. As this power increases, the nearest point to the query point becomes more dominant. The simplicity of the underlying principle, the speed in calculation, the ease of programming, and reasonable results for many types of data are some of the advantages associated with inverse distance interpolation. The disadvantages are: choice of weighting function may introduce ambiguity, especially when the characteristics of the underlying surface are not known; the interpolation can easily be affected by uneven distribution of observational data points since an equal weight will be assigned to each of the data points even if it is in a cluster; maxima and minima in the interpolated surface can only occur at data points since inverse distance weighted interpolation is a smoothing technique by definition.

There are two software packages LEDA and CGAL, which solve geometrical problems, both offer implementations for the Delaunay triangulation and an efficient nearest neighbour search algorithm. As we looked further into CGAL, we came to the conclusion that it would take a lot of time to understand the basic usage of the library and to adapt it for this particular interpolation application. After some research on the inverse distance method it seemed that it would be easier to implement than the Delaunay triangulation and that it probably would be faster too. As it turned out the opposite was the case. Our Delaunay interpolation

implementation is much faster than the inverse distance interpolation implementation. Although inverse distance interpolation is still supported in the simulation software, we will use the Delaunay triangulation method from this point on. The Delaunay interpolation method will be described in this section, the inverse distance interpolation scheme is described in appendix D.

Delaunay triangulation

There are several ways to construct a Delaunay triangulation from a set of points. Our algorithm, however, supports incremental insertions as well as deletions to allow interactive editing. When a point is inserted the triangle is searched which contains the point. Then edges are added to update the triangulation. If some of the new edges violate the Delaunay triangulation condition then edges are swapped to restore the Delaunay property. If the point lies outside the current triangulation then the bounding triangle is stretched until the point lies inside the triangulation. During a deletion a polygon is created of all edges that surround the deleted vertex, this polygon is then triangulated while preserving the Delaunay property. When the height of an arbitrary location in the map is needed, first the triangle containing the query point is searched. The found triangle consists of three vertices which, because of the Delaunay property, are nearest to the query point. Gouraud shading is used to interpolate the three height values at the location of the query point. Although the Delaunay triangulation will work fine most of the time, the algorithm is not geometrically robust. In special vertex configurations floating point roundoff error result which can cause the algorithm to fail. In the implementation we found, the algorithm used a special floating point library that starts using of precise floating point arithmetic when extra accuracy is needed. This library however 'hacked' directly into the floating point processor and would only work with specific operating systems. This is why we have chosen not to include this library. The other option is to always use precise arithmetic but this would slow down the algorithm considerably.

Terrain cross-section calculation

Now that we can use the elevation and terrain type map in the radar coordinate system, the map can be used to calculate the radar return signal. At discrete points in range and azimuth the radar cross-section σ of a terrain patch is calculated. We start by calculating the area of a discretized terrain patch.
High peaks on the map can obscure lower lying terrain which lies after the peak. Terrain features can even obscure targets. The cross-section is lowered when the terrain patch is shadowed. To approximate the terrain patch area, it is assumed to be an area on a circle with a range interval $[r_s..r_e]$, a height interval $[h_s..h_e]$ and an arc angle $\Delta\alpha$. The area is then calculated via a parametrized surface. The area A of a parametrized surface $(x(u,v), y(u,v), z(u,v))$ is:

$$A = \int\int_D \|T_u \times T_v\| \, du \, dv \tag{4.1}$$

where D is the region, and T_u and T_v are:

$$T_u = \frac{\partial x}{\partial u}(u,v)i + \frac{\partial y}{\partial u}(u,v)j + \frac{\partial z}{\partial u}(u,v)k \tag{4.2}$$

$$T_v = \frac{\partial x}{\partial v}(u,v)i + \frac{\partial y}{\partial v}(u,v)j + \frac{\partial z}{\partial u}(u,v)k \tag{4.3}$$

Our parametrized surface is a part of a dice with elevation values:

$$x(u,v) = u \tag{4.4}$$

$$y(u,v) = v \tag{4.5}$$

$$z(u,v) = (\sqrt{u^2+v^2} - r_s) \cdot \frac{(h_e - h_s)}{(r_e - r_s)} + h_s \tag{4.6}$$

Now (4.1) becomes:

$$A = \int_{r_s}^{r_e} \int_{0}^{\Delta\alpha} \|T_r \times T_v\| \, dr \, dv \tag{4.7}$$

This evaluates to:

$$A = \frac{1}{2}\Delta\alpha({r_e}^2 - {r_s}^2)\sqrt{\frac{(h_e - h_s)^2 + (r_e - r_s)^2}{(r_e - r_s)^2}} \tag{4.8}$$

If we take $h_s = h_e = 1$, $r_s = 0$ and $\Delta\alpha = 2\pi$ (the complete circle rotation), then A winds up to be just the area of a circle $\pi {r_e}^2$.
Shadow is easily incorporated on a terrain patch if the shadow range r_d and shadow height h_d are known. The area patch witch is not shadowed has an area A with $h_s = h_d$ and $r_s = r_d$

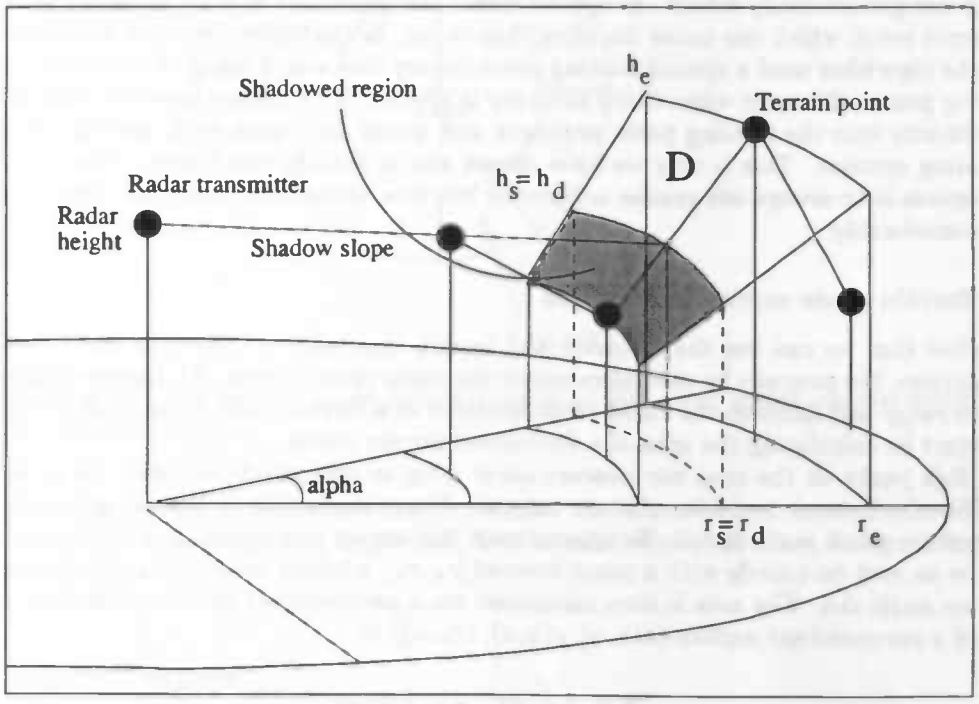


Figure 4.2: *D is the terrain region that is not shadowed. We can calculate the area of this region by using (4.8). The dots depict terrain height values which are sampled in polar coordinates.*

Another factor besides shadow and area that influences the radar cross-section of a terrain patch is the terrain type. For each terrain type a reflection function can be defined which depends on the grazing angle g. This angle between the radar beam and a terrain patch is calculated by taking the inner product of the terrain direction vector and the direction of the beam:

$$\cos g = \frac{s_{beam} \cdot s_{terrain}}{\|s_{beam}\| \, \|s_{terrain}\|} \tag{4.9}$$

If the grazing angle g is small then the terrain surface is almost perpendicular to the beam and much energy will be reflected back in the direction of the radar. When g is almost zero most of the energy is reflected away from the radar.

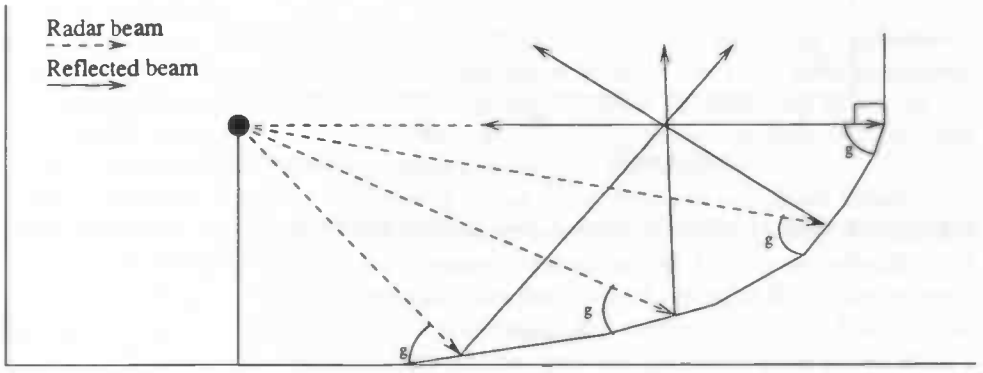


Figure 4.3: *When the grazing angle g is small, the beam is reflected away from the radar. When g is about 90° the beam is reflected back to the radar receiver.*

Rough terrain surfaces will reflect the radar signal in a diffuse manner. Smooth surfaces can reflect radar waves specularly. The total reflection thus consists of two components:

$$\sigma_{reflect}(\theta) = \sigma_{diff}(\theta) + \sigma_{spec}(\theta) \tag{4.10}$$

In this formula $\sigma_{diff}(\theta)$ is defined as:

$$\sigma_{diff}(\theta) = \sigma_d \sin(\theta) \tag{4.11}$$

and $\sigma_{spec}(\theta)$ is defined as:

$$\sigma_{spec}(\theta) = \sigma_s e^{\frac{(\frac{1}{2}\pi - \theta)^2}{\phi_{spec}^2}} \tag{4.12}$$

In these functions σ_d, σ_s and ϕ_{spec} are constants which describe respectively the diffuse reflectivity, specular reflectivity and the angle width of the specular reflection.

The total reflection $\sigma_{reflect}$ represents the mean factor with which the radar signal is reflected. By multiplying this with the terrain patch area calculated from (4.8) we get the ground clutter mean cross-section value. According to [17] the ground clutter cross-section is distributed by the log-normal distribution. But the user can select another statistical distribution like the Weibull distribution which is also used often. The only restriction is that the selected distribution must have a mean of 0. Each time a reflection factor is needed, the mean cross-section value is scaled with a sample from the selected distribution.

When the terrain RCS values have been calculated. The antenna pattern must be applied. This is done through a circular convolution between antenna pattern and terrain RCS values.

Different scenarios were designed to test the simulation. The most interesting was the simulation of rocky coast. A terrain map was designed in which the coast had a very steep descent into the ocean. Such coasts exist for instance in Norway. The idea is that radar waves will reflect very well from the steep terrain but much

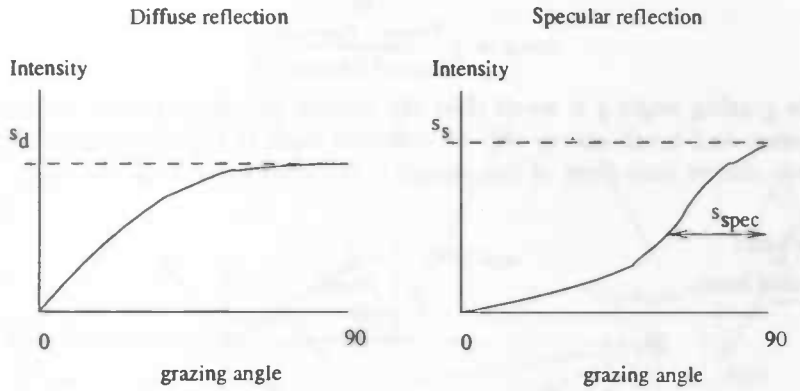


Figure 4.4: *Diffuse reflection ($\sigma_d = s_d$)and specular reflection ($\sigma_s = s_s$ and $\phi_{spec} = s_{spec}$)*

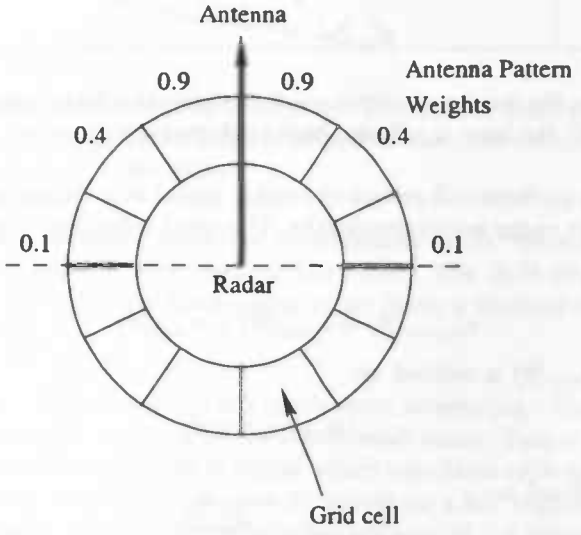


Figure 4.5: *The antenna weight pattern is convolved with a ring of grid cells that contain the terrain RCS values.*

less from the flat terrain which lies just behind the coast. Indeed the coast showed up on the radar image as a bright line. The terrain behind the coast was much less visible.

4.2.2 Targets

Trajectory design

When the target tracking algorithm is finished we will need to check its accuracy. If the algorithm indicates a target at a certain position with a certain velocity and acceleration we must be able to check these values against the real position of the target. Thus a trajectory definition is needed with which we can calculate positional information at a given time. A trajectory is best designed by specifying a number of way points. Each way point specifies a number of target property constraints like time, position, velocity etc. Now the problem arises that between given way points

the constraints need to be interpolated. First the constraints are defined for a way point P:

$$P = (t, p_x, p_y, v_x, v_y) \tag{4.13}$$

The target will arrive at way point P on time t with position (p_x, p_y) and velocity (v_x, v_y). Instead of giving the velocity as a vector we could also just specify the velocity magnitude or speed s and let the algorithm determine the vector components v_x and v_y.
In a first attempt linear interpolation could be used, however this is very unrealistic. The derivative of position is velocity, and when straight lines are used to connect way points there will be discontinuities in the velocity. At each way point the velocity would suddenly change to a new value, causing infinite acceleration at these points, realistic targets of course have a maximum acceleration.
A nice approach would be to define a spline $S_p(d)$ through the given target positions, where d is the distance covered with respect to the starting position. Another spline $S_v(t)$ would interpolate the distance covered (in effect the given velocities) at time t. At a time t we could then calculate the covered distance $S_v(t)$, the covered distance in turn is used to find the position of the target $S_p(S_v(t))$. Finding the spline $S_p(d)$ however is a difficult problem because the spline has to be parametrized according to distance. Popular methods that use polynomials as interpolators (e.g. cubic splines) cannot be analytically parametrized, their distance parametrization will have to be approximated numerically (see [12]), which is too cumbersome for our application.
With some simplifications Hermite spline interpolation (see [6] for details)is a good solution. At the beginning and end point of each spline section we have two way point constraints P_1 and P_2. Between P_1, P_2 a Hermite spline $S_p(t)$ is defined, this spline is parametrized according to time, at $t = 0$ the spline is at P_1 and at $t = 1$ the spline gives the position P_2 . For the definition of a Hermite spline, the derivatives at the end points are needed as well. The derivative of position along a spline, which is parametrized to time, represents velocity, and we can specify the given target velocity constraints at each way point by setting the derivative parameters of the spline. If only the speed is given at one of the way points P_1, P_2, cardinal spline interpolation is used to find the derivative components at P_1 and P_2, the given speed is then used to scale the found derivatives (velocity) so that their magnitude matches the speed.

Clipping

Each target is partitioned into sample points, and at each sample point the radar cross-section is calculated. Complicated targets can require lengthy calculation of many samples. To speed up calculation it will be necessary to exclude those targets from the simulation which are currently not in the line of sight of the radar. Normally such a clipping region could be a box, but because most calculations are done in polar coordinates, an efficient clipping region consists of two angles rather than box coordinates. To determine whether a target should be calculated the following test are performed in order:

1. The target is included if the target's azimuth lies between the clipping angles.
2. The target is included if the radius of its bounding circle is larger the range of the target.
3. If these tests fail the cosine rule is used to calculate the angular width of the bounding circle. If the azimuth of the target plus its angular width overlaps with the clipping region the target is included.

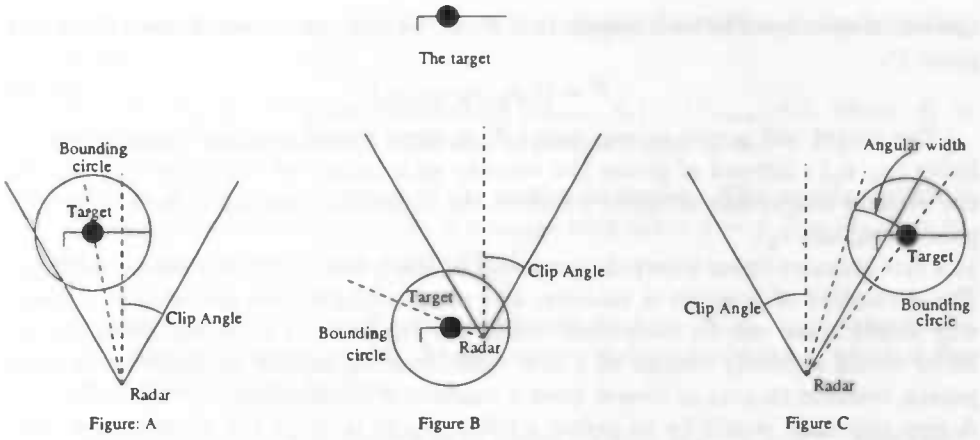


Figure 4.6: *Target clipping: Figure A represents test 1, figure B represents test 2, figure C represents test 3.*

Cross-section model

Each target consists of a number of scatterers. The current simulation supports line scatterers and point scatterers. Each scatterer is sampled according to the range and azimuth resolution of the radar.

Point scatterers A point scatterer is simulated easily, because the scattering is invariant with respect to the angle of the radar. The RCS of a point scatterer is computed by multiplying a fluctuating value (from a statistical process) with a given area constant.

Line scatterers Line scatterers are much more interesting. The return of a line segment depends on the angle of the incident radar beam. Just as with the terrain type definition, a reflection model is added which calculates reflectivity with respect to the radar grazing angle. Furthermore all points on a line segment with the same range from the radar will have to be included because of the antenna pattern. The antenna will receive echos even from those targets which are not directly in the line sight of the radar. On a line we will have to add at each range at most 2 samples, however there are many cases when only one point is added. The basic idea is as follows: first find one point on the line of which we know that it is needed in the calculation. From this point iteration starts in a positive direction until stop criteria are met. The next iteration starts in a negative direction until stop criteria are met. Other simpler approaches which just iterate from the beginning of the line segment all the way through the end of the segment cannot be modified to include the use of the clip angles. This is the resulting algorithm:

1. First the minimum distance R_{min} from the radar to the line is calculated, by a simple point-line distance calculation.
2. The line is parametrized by u. Define u_{min} as the point at which the range is R_{min}.
3. Calculate the maximum range R_{max} and u_{max}.
4. The next step is to calculate the intersection between the line segment and the antenna direction line. This results in a point u_{int}.

5. If u_{int} does not lie on the line segment, the point u_{start} is chosen as the point on the line segment which lies closest to u_{int}. If u_{int} does lie on the line segment we choose $u_{start} = u_{int}$.

6. Define the range at u_{start} as R_{start}.

7. Phase 1: Let r iterate from R_{start} to R_{min} with a step of the range resolution Δr and calculate u between u_{min} and u_{start} accordingly. At each point, u is used to calculate the azimuth position of that point . The cross-section is calculated by multiplying the area of a sample with a value from a probability distribution and the reflection amount which is obtained from the reflection model. Another point u_{sym} is identified which lies symmetrically around u_{min}, the cross-section calculation for this point uses the same values (except possibly the statistical fluctuation value). Iteration is ended when R_{min} is reached or when both symmetrical points lie outside the line segment or clipping region.

8. Phase 2: Let r iterate from R_{start} to R_{max}, the cross-section at each point is calculated as in the previous step.

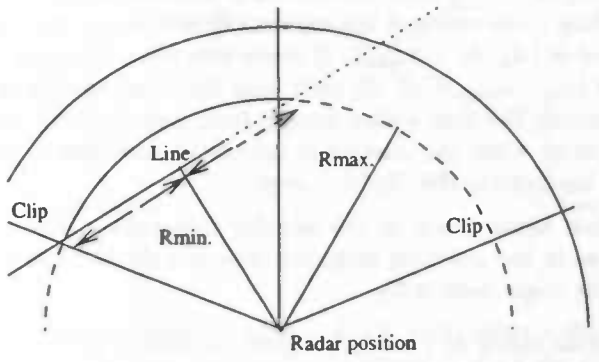


Figure 4.7: *Sampling a line segment. Sampling starts at the endpoint of the line and then progresses until the point of minimum range (Rmin) is reached. During this phase at each sample point two samples are calculated, which lie symmetrically around the point of minimum range. In the next phase sampling again starts at the endpoint, but now the sampling progresses to the maximum ranges (Rmax). In this example only one sample is added at each sample point, because of the fact that the other sample lies outside the line segment (dashed arrow).*

4.2.3 Volumetric clutter

Clutter like rain and fog can affect radar performance enormously. First the precipitation itself generates many echoes in which target echoes can 'drown'. Furthermore the clutter will attenuate the radar signal which results in lower detection probability of targets which lie behind the clutter. The algorithms used to calculate volumetric clutter resemble those used in target cross-section simulation. A difference is that an extra step is needed to calculate the attenuation. In our model the clutter extends over the area of a circle. The bounding circle coincides with this area. Thus if the bounding circle overlaps with the clipping region we also know that some clutter values are within the region. The algorithm works as follows:

1. The maximum and minimum range of the clutter are determined: $R_{min} = R_c - f$ and $R_{max} = R_c + f$, where R_c is the range to the center of the clutter

and f is the radius of the clutter (which equals the radius of the bounding circle).

2. Identify the line in polar coordinates $(R_{min}, A_c) - (R_{max}, A_c)$, where A_c is the clutter center azimuth. This line is parametrized by u.

3. At $u = \frac{1}{2}$ the angular width of the clutter is largest. The angular width is calculated with the cosine rule, a triangle with sides $A = f, B = R_c, C = R_c$ has an angle $\alpha = A_{max} = \cos^{-1}(\frac{2R_c^2 - f^2}{2R_c^2})$.

4. Iteration starts at $r = R_c$ and progresses to $r = R_{min}$, u is determined accordingly between 0 and 1. At each step r is decreased by Δr, which is the range resolution of the radar.

   (a) Calculate the angular width at range r which is $a_{max} = A_{max} \cos(\frac{1}{2} u\pi)$.

   (b) Calculate the volume of a resolution cell: $P = \Delta a\, \pi((R_c + \Delta r)^2 - R_c^2)D$, where Δa is the radar azimuth resolution and D is the depth of the clutter.

   (c) Iteration starts in the angular dimension. If A_c lies between the clipping angles C_1, C_2 then the iteration interval is $[A_c, A_c + a_{max}]$. If the bounding circle overlaps the antenna direction line with azimuth A_a, the interval is $[A_a, A_c + a_{max}]$. If these two tests fail then the interval becomes $[A_c - a_{max}, C_1]$. At each step the cross-section is determined by multiplying the area with a sample from a probability distribution. Iteration stops when the interval is completely sampled or when the sample point lies outside the clipping angle.

   (d) Iteration again starts in the angular dimension. The procedure is the same as in the previous step, but now the direction is negative and the clipping angle used is C_2.

5. Iteration again starts at $r = R_c$ but now progresses to $r = R_{max}$. Each step is performed as in the previous step.

Attenuation

The user can specify an attenuation constant, e.g. attenuation of 3 dB/m, means that every meter of clutter will decrease the signal power by $\frac{1}{2}$. To determine which part of a scan line has to be attenuated first the two intersection ranges of the clutter signal with the scan line have to be found. The signal samples which lie before the first intersection range are not attenuated. The samples between the first intersection range and the last intersection range are multiplied by an attenuation factor, which decreases with the distance from the center to the border of the clutter. The samples after the last intersection range are multiplied by the last calculated attenuation factor.

Rain clutter

Signal attenuation in dB/km caused by rain clutter is calculated by:

$$\alpha = aR^b \tag{4.14}$$

with α the attenuation in dB/km, R the rain rate in mm/hour and a, b coefficients which depend on radar frequency and temperature. Moderate rain will have $R = 4$ mm/hour and heavy rain has an R of 16 mm/hour. At a radar frequency of 9400

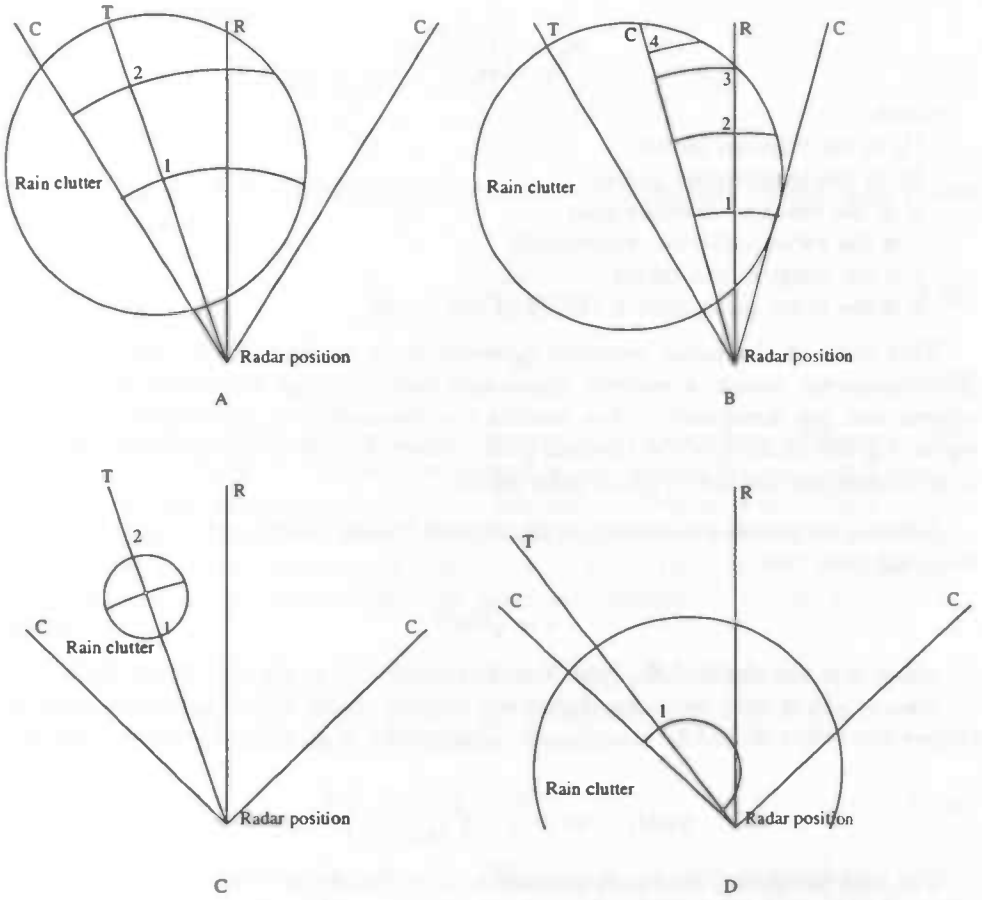


Figure 4.8: *Sampling volumetric clutter. A: At each range r the angular interval is* $[A_c, A_c + a_{max}]$ *and* $[A_c, A_c - a_{max}]$. *B: At ranges 1, 2, 3 the angular interval is* $[A_a, A_c + a_{max}]$ *and* $[A_a, A_c - a_{max}]$, *at range 4 the interval is* $[A_a - a_{max}, C_1]$. *C: special case where the clutter completely resides within the clipping region. D: Clutter overlaps with the radar position, the angular width of the clutter is* π.

Mhz. a and b have values of 1, 6 and 0.64 respectively. This leads to an attenuation of 3, 9 dB/km for moderate rain and 9, 4 dB/km for heavy rain. The mean cross-section of rain at 9500 Mhz is about -40 dB per meter for moderate rain and -35 dB per meter for heavy rain.
Tests showed that the echo from a ship behind a cloud of rain was correctly much weaker than the echo that was received when the rain was removed. There is still a problem in the sampling of the rain cloud shape. The rain cloud is modelled as a circular area, all points within the circle are affected by rain. Due to some of the simplifications which were needed to implement the sampling of the circle area in polar coordinates efficiently, artifacts result if the rain cloud is very close to the radar origin. These artifacts distort the shape of the circle.

4.2.4 Radar Range Equation

The equation which we will have to implement is the radar range equation, which describes the received power from a target:

$$P_R = \frac{P_T G^2 \lambda^2}{(4\pi)^3 r^4} \sigma \tag{4.15}$$

where
P_R is the received power
P_T is the transmitted power
G is the one-way antenna gain
λ is the radar radiation wavelength
r is the range of the target
σ is the radar cross-section (RCS) of the target

This form of the radar equation however lacks detail. We are dealing with electromagnetic waves, a realistic simulation requires that amplitude and phase information are simulated. This enables the simulation of interference between waves, e.g. the forming of the antenna gain pattern. Complex calculus is introduced to accommodate the use of phase information.

When a target reflects energy, a phase shift is also introduced. Instead of just σ we will now use:

$$\gamma = \sqrt{\sigma} e^{j\phi} \tag{4.16}$$

where ϕ is the phase shift. Note that the power $|\gamma|^2$ is just the target RCS σ.

Also (4.15) is only valid for stationary targets. If we want to include moving targets then time should be introduced. This results in an adapted radar equation:

$$\psi_R(t) = \psi_T(t - \tau) \left(\frac{G^2 \lambda^2}{(4\pi)^3 r^4} \right)^{\frac{1}{2}} \gamma \tag{4.17}$$

The next problem is that each measurement will take some time. In the equation above G, τ, r and γ have been considered to be constant during this time. This is actually only true in certain circumstances. If we take this into the formula becomes:

$$\psi_R(t) = \psi_T(t - \tau(t)) \left(\frac{\lambda^2}{(4\pi)^3 r^4(t)} \right)^{\frac{1}{2}} G(t)\gamma(t) \tag{4.18}$$

$\tau(t)$ and $r(t)$ are time dependent if the target is moving. As the antenna is scanning it will rotate during the measurement, thus the antenna gain G depends on time as well. A moving target will present different aspects to the radar, which is why $\gamma(t)$ is not constant. If we want to perform an efficient simulation then we must take G, r, τ and γ constant over some time interval, which is only possible if they are functions that vary slowly enough. If this is the case, then the radar range equation can be reduced to:

$$\psi_R(t) = \psi_T(t - \tau) e^{j2\pi vt} \frac{\lambda}{(4\pi)^{3/2} r^2} G\gamma \tag{4.19}$$

This equation gives us received RF energy. $\psi_T(t)$ is the transmitted RF energy, which is actually a complex modulation functions centered at the radar carrier frequency:

$$\psi_T(t) = \mu_T(t) e^{j2\pi f_c t} \tag{4.20}$$

where $\mu_T(t)$ is the complex modulation function or the transmitted waveform and f_c is the radar carrier frequency. If we rewrite $\psi_T(t)$ in the radar equation, the

radar equation becomes:

$$\psi_R(t) = \mu_T(t-\tau)e^{j2\pi\upsilon+f_c t}\frac{\lambda}{(4\pi)^{3/2}r^2}G\gamma \tag{4.21}$$

The variables υ (Doppler coefficient) and τ are dependent on the range r and the range rate r':

$$\tau = \frac{2r}{c} \tag{4.22}$$

$$\upsilon = \frac{-2r'}{\lambda} \tag{4.23}$$

The Doppler coefficient is important because it will introduce a frequency shift in the received signal if a target moves away or approaches the radar at a specific velocity. In pulse Doppler radar systems this shift in frequency is actually used to detect targets in stationary clutter. In our simulation where we use only pulse radar systems this Doppler effect is not used. If υ, τ, G, γ and r are to be considered constant over the measured time then there are restrictions on the speed of the target:

First we have the assumption of constant τ. As seen earlier, the radar's range resolution is given by $\Delta r = c\tau_{pulse}/2$. If the measurement time is T then the target will move $r'T$, this value should be smaller than Δr:

$$r' \ll \frac{c\tau_{pulse}}{2T} \tag{4.24}$$

The resolution of radar in Doppler (the difference in velocity at which two targets can be set apart) is $\Delta\upsilon = 1/T$. If a target accelerates then its change in Doppler $2r''/\lambda$ follows from (4.23). This value should be smaller than $\Delta\upsilon$:

$$r'' \ll \frac{\lambda}{2T^2} \tag{4.25}$$

The gain G can be considered constant if the antenna rotation rate is not too high. Just like τ and υ the rotation rate θ' should be smaller than the antenna resolution $\Delta\theta$:

$$\theta' \ll \frac{\Delta\theta}{T} \tag{4.26}$$

Furthermore the complex reflection coefficient γ should be constant. If γ changes rapidly this is usually due to a rotating target that will present its different aspects to the radar. In most cases such a target can be broken up into two or more separate targets which all have constant γ. Lastly a varying range r contributes to an amplitude modulation of the received signal, however this effect is very small in comparison to the other effects which is why r can be considered constant too. The conditions stated in (4.24), (4.25) and (4.26) can be considered satisfied if their difference is a factor of about 4 (see [17] p. 13).

Now that we have obtained an equation for one scatterer we can calculate the complete signal received by the receiver just by applying (4.21) for all scatterers and adding the results.

4.2.5 Antenna pattern

The ideal radiation pattern that the antenna should emit is the so-called pencil beam. In realistic antennas a radiation pattern is created which has some properties of the pencil beam, however the differences should be taken into account as well. Usually in a non-ideal pattern one main beam or main lobe exists, which looks like the pencil beam and around it there are other smaller beams which are called side lobes. At some points the radiation pattern could even be zero due to phase interference. Targets and clutter in the side lobe region of the antenna pattern will interfere with the targets in the main lobe. This effect should be simulated and we will derive an equation for the antenna pattern of slotted waveguide antennas. We will consider the two-dimensional pattern only, as most naval radar systems have a very wide vertical detection angle. The antenna is also oriented vertically, other orientations can be obtained simply by performing rotations.

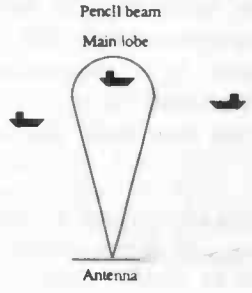


Figure 4.9: *The ideal antenna beam is pencil shaped. Only one ship will be illuminated by the radar waves that are transmitted via this antenna.*

Antenna pattern for a rectangular aperture The radiation intensity will be calculated at (u, v). $2a$ is the length of the aperture. x is a point on the (one dimensional) aperture from $[-a.a]$. λ is the radar wavelength. K is a constant which denotes the radiation intensity. The radiation in (u, v) is calculated by integrating all radiation received as we move the point x along the aperture $[-a.a]$. In effect all the phase differences, which are caused by the difference in distance between (u, v) and $(0, x)$, are added. If we denote the distance difference as $\Delta d(x)$ then we obtain:

$$G(\theta) = \int_{-a}^{a} K e^{2\pi i \Delta d(x)} dx \tag{4.27}$$

Direct calculation of $\Delta d(x)$ leads to taking the difference of the two distances between $(0, 0)..(u, v)$ and $(0, x)..(u, v)$ which are $\sqrt{u^2 + v^2} - \sqrt{u^2 + (v - x)^2}$. Unfortunately this expression for $\Delta d(x)$ would lead to an analytically unsolvable equation 4.27, which would then have to be solved numerically. Another option is to approximate $\Delta d(x)$ by $x \sin(\theta)$ where θ is the angle of the vector $\vec{p} = (u, v)$ with the x-axis. This approximation is valid as long as the distance of (u, v) is large enough with respect to the aperture size $[-a.a]$. After applying the approximation, (4.27) becomes:

$$G(\theta) = \frac{\lambda \csc(\theta) \sin(\frac{2a\pi \sin(\theta)}{\lambda})}{\pi} = G(\theta) \tag{4.28}$$

If we look at the graph 4.10 we indeed see a large main lobe in the center with smaller side lobes around it. Now recall the formula for the horizontal beam width which is $HBW = C\frac{\lambda}{2a}$ (in degree), where C is a constant between 51 and 70. The

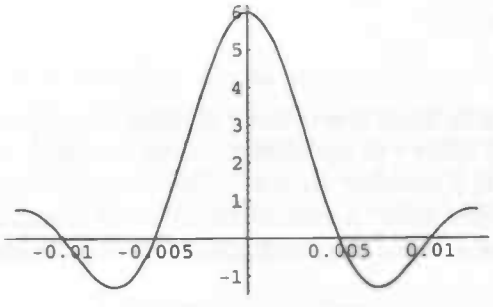


Figure 4.10: *Example antenna pattern*

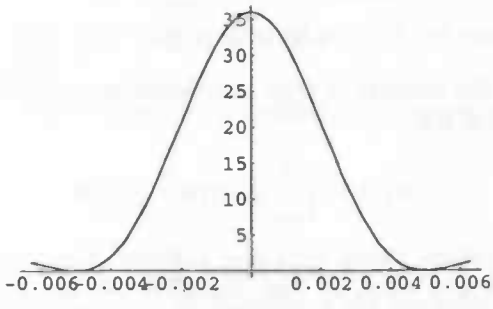


Figure 4.11: $G(\theta)^2$ *with* $2a = 6$ *m and* $\lambda = 0.03$ *m.*

HBW is defined as the angular interval between the $3dB$ (half power) points of $G(\theta)^2$. If we take $\lambda = 0.03$ m, $2a = 6.0$ m. and $C = 68$ then the formula gives us a HBW value of 0.006 rad.
If we take a look at the power graph 4.11 of $G(\theta)^2$ then we see that for $\lambda = 0.03$ m and $2a = 6.0$ m, the HBW would be the interval between the half power points of the graph. The maximum of $G(\theta)^2$ is 36 and the half power points thus have $G(\theta)^2 = 18$. This restricts the value of θ between -0.003 and 0.003 rad. Thus the HBW value is 0.006 rad, which is also the result from the HBW formula.

Strangely enough, during simulation we found that the pattern described above is not completely correct. It consists of too many side lobes. In the literature [9] (chapter 3.2 p. 5) and [19] (chapter 6 p. 27), however (4.28) is clearly mentioned. This will have to be studied further.

4.2.6 Radar waveform

As discussed in the introduction our simulated radars will use simple short pulses as their waveform. An important aspect of short pulse waveform is that it has good time resolution but no Doppler resolution. This means that the position of targets can be measured accurately, however the velocity of the targets are unknown. This lack of Doppler resolution in the radar also means that moving targets in clutter are hard to detect.

The fact that the radars will not have resolution in Doppler simplifies the radar equation 4.21. We can take $\upsilon = 0$, resulting in a modified radar equation:

$$C = \frac{\lambda}{(4\pi)^{3/2}\tau^2} G\gamma \tag{4.29}$$

$$\phi_R(t) = C\mu_T(t-\tau)e^{j2\pi f_c t} \tag{4.30}$$

4.2.7 Receiver

Filtering

In the receiver the radar input is processed to make target detection possible. First the received signal is filtered to optimize the noise to signal ratio. The filter which we use for this task is a modified version of the transmitted pulse μ_t. This kind of filter is called a matched filter. A matched filter is the original signal flipped about the origin on the time axis and then conjugated. Its impulse response is constructed by:

$$H(f) = M_t^*(f) \tag{4.31}$$

$$h(t) = \mu_t^*(-t) \tag{4.32}$$

The response of the receiver is then calculated by a convolution of the filter h and the received signal ϕ_R.

$$Z(\tau) = \int_{-\infty}^{\infty} \phi_R(t)h(\tau - t)\,dt \tag{4.33}$$

Convolving the return signal with the matched filter basically amplifies the bursts in the clutter and noise so that the SNR (signal to noise ratio) is large. If we use the definition (4.30) for ϕ_R and the matched filter definition (eq. 4.32) for h the formula can be rewritten in a very functional form:

$$Z(\tau) = C\,\chi(\tau - \tau_{\mathtt{scatterer}}) \tag{4.34}$$

$$\chi(\tau) = \int_{-\infty}^{\infty} \mu_t(t)h(\tau - t)\,dt \tag{4.35}$$

In this formula C is a scaling factor as in (4.29), χ is called the thumbtack ambiguity function. The function χ characterizes the range and Doppler resolution of the transmitted waveform. When there are more scatterers, the receiver responses for each scatterer are added. Each scatterer will result in a superposition of the (scaled) ambiguity function χ at the scatterer delay coordinate τ.

Scaling

After filtering, scaling is needed because the received signal will have a very large dynamic range. From the radar equation (eq. 4.29) it is seen that the received power from a target is proportional to $1/r^4$. To decrease the dynamical range the starting samples of the signal where r^4 is very large are attenuated:

$$\mu_{\mathtt{scaled}}(t_k) = (1 - e^{-(r/sc)^2})\mu_{\mathtt{rec}}(t_k) \tag{4.36}$$

In this formula $\mu_{\mathtt{rec}}$ is the original unscaled received signal and r the range at which we scale. sc is a scaling constant which determines the range after which the influence of the scaling quickly diminishes.

Quantization

All computer applications involving digital radar image processing will require a digital signal. Furthermore both amplitude and phase information are processed in a coherent receiver, and each sample will have an amplitude and phase component. The analog to digital domain conversion is simulated in two steps. Firstly clipping will be introduced. If we have a maximum signal power P then the clipped signal $\mu_{\mathtt{clip}}$ is:

$$\mu_{\mathtt{clip}}(t_k) = \begin{cases} \mu_{\mathtt{scaled}}(t_k) & , |\mu_{\mathtt{scaled}}(t_k)| \leq \sqrt{P} \\ \frac{\sqrt{P} * \mu_{\mathtt{scaled}}(t_k)}{|\mu_{\mathtt{scaled}}(t_k)|} & , |\mu_{\mathtt{scaled}}(t_k)| > \sqrt{P} \end{cases} \tag{4.37}$$

This clipping function only affects the real part of the signal, the phase information is left unchanged.

After clipping the D/A conversion is applied. In most modern radars 8 bits are used for the conversion.

Receiver noise

Due to imperfections in both receiver and antenna noise is generated. There are different noise sources but thermal noise is one of the more dominant factors. Thermal noise is relatively simple to simulate because it is considered to be white noise, and thus has a flat spectrum. When we have generated the noise samples they have to be injected in the system at the appropriate point. Receiver noise is affected by the matched filter. So the flat noise spectrum is not flat after the filtering. The bandwidth of the filter however is about the reciprocal of the pulse length τ and thus the bandwidth of the noise is also $\frac{1}{\tau}$. The power of the noise can be calculated as $P_n = kFTG_f\frac{1}{\tau}$, in which k is the Boltzmann constant, F the receiver noise figure, T the temperature and G_f the receiver gain. The actual noise samples are calculated from a Gaussian process with the variance equal to the noise power above. The noise samples are added just after the calculation of the receiver response and the filtering, but before scaling and quantization.

4.2.8 Putting it all together

Now that the building blocks of the simulation have been developed they will have to be put together to generate the final simulated radar signal. This is done as follows:

Initialization and precalculation

1. Calculate the shadowed areas.
2. Compute the radar cross-section of each terrain patch by calculating the visible area and the reflection function of the terrain.
3. Apply the antenna gain pattern.
4. Precalculate the thumbtack ambiguity function (eq. 4.35), by correlating the radar waveform with the matched filter of the receiver.
5. Initialize 'Receiver Noise' generator.
6. Calculate bounding radius of each target.
7. Reset antenna position.

Simulation

In each step a single azimuth beam is calculated and written to disk as follows.

1. Get the precalculated terrain clutter radar cross-section values. A fluctuating value computed from the statistical properties of the terrain type is added. The fluctuating RCS value is added according to the radar equation 4.30.
2. The targets are positioned and sampled. Each sample that is not shadowed is added through the radar equation.
3. The rain clutter is positioned and sampled. Each sample is added through the radar equation.
4. The receiver response is calculated by convolving the ambiguity function (eq. 4.35) $Z(t)$ with our summed RCS scan line.
5. Receiver noise is added to the receiver output.
6. The received signal is scaled by equation 4.36.
7. The output is quantized.
8. The resulting digital radar signal is written to the buffer.
9. The buffer is visualized and written to disk.

4.3 Implementation

4.3.1 Ground clutter

First a base class `TTerrain` has been created. Two classes `TDelaunayTerrain` and `TKDTreeTerrain` have been derived from this class which represent terrain maps stored in respectively a Delaunay triangulation and a kd-tree. The `TKDTreeTerrain` was implemented first, unfortunately the KD tree algorithm is much slower than the Delaunay triangulation algorithm. Thanks to the object oriented design the kd-tree algorithm however can still be used in the simulation. Inverse distance interpolation has been implemented on top of the kd-tree algorithm. See appendix D for backgrounds on the inverse distance interpolation algorithm. The `TDelaunayTerrain` class stores a `TDelaunayTriangulation` object which implements the actual triangulation algorithm. The triangulation is built up as a list of triangles. The triangles contain edges, and these edges in turn contain vertices:

```
TVertexPtr = ^TVertex;
TEdgePtr= ^TEdge;
TTrianglePtr = ^TTriangle;

TMapPoint = record
    x, y, z: double;
    tt: TTerrainType;
end;

TVertex = record
    point: TMapPoint;
    // linked list of edges
    edges: TLinkElementPtr;
end;
```

```
TEdge = record
    v1, v2: TVertexPtr;
    // linked list of triangles
    triangles: TLinkElementPtr;
end;

TTriangle = record
    e1, e2, e3: TEdgePtr;
end;
```

When a triangle is destroyed, edges that no longer are used in other triangles are destroyed as well, vertices which are not used in any edge are destroyed too. The triangle search algorithm allows a 'guess' triangle to be given. The search is then started at the guessed triangle and with a proper guess the search is sped up. This 'guess' parameter is automatically set to the result of the previous triangle query, and should especially speed up screen drawing. The `TDelaunayTriangulation` supports methods to search objects in different ways within the triangulation:

```
// Finds bounding box of all triangles
procedure calcBoundingBox(var r: Tregion);
// finds triangle containing p
function VertexLocate (
      const p: TVertexPtr;
      guess: TTrianglePtr): TTrianglePtr;

// finds triangles which overlap with region r
// result: a linked list res of triangles
procedure TrianglesInRegion(
  const r: TRegion;
  var res: TLinkElementPtr;
  var count: integer);

// finds vertices in a window r
// returns a list l of vertices
procedure getVerticesInRegion(
  const buildList: boolean;
  const r: TRegion;
  var l: TList;
  var count: integer);
```

The search algorithms used in these methods make optimal use of the Delaunay structure and are very efficient. Vertices can of course be inserted and deleted. The Delaunay algorithm used in `TDelaunayTriangulation` is incremental, so deletions and insertions are efficient. The definitions for the insert and remove methods are:

```
// remove a vertex v
procedure RemoveVertex(
  v: TVertexPtr);

// add a vertex v
function AddVertex (
      const v: TVertexPtr;
      const guess: TTrianglePtr): TVertexPtr;
```

Further the `TTerrain` is persistent and can be stored to disk. When this is done a `TerrainTypeFixupTable` object is stored also. This table object allows the terrain types which are stored as pointers within the `TTerrainmap` object to be resolved when the `TTerrain` is read back from disk.

Shadow calculation

One of the effects of ground clutter is shadowing. High obstacles in the clutter can obscure targets which lie in the shadow of these obstacles. At each point in the clutter map it is determined whether the point is shadowed or not. It is easy to calculate the shadowed points if the clutter map is processed in range, azimuth order while the map is transformed from Cartesian coordinates to polar coordinates. The following pseudo-code will process one range array R of elevation values and will calculate the shadowed elements:

```
dx:= R[0].range/2
dy:= R[0].elevation - radar.elevation
rc:= dy/dx

for n = 0 to R.length do begin
  shadowHeight:=R[n].range*rc+radar.elevation
  if R[n].elevation < shadowHeight then
    R[n].shadow:= true
  else
    R[n].shadow:= false
    dx:= R[n].range
    dy:= R[n].elevation - radar.elevation
    rc:= dy/dx
   end if
end
```

`rc` is the slope of the line which constitutes the border of the shadowed region. If terrain points lie below this line they are shadowed. If they lie above the line they will not be shadowed, and in this case a new `rc` is calculated.

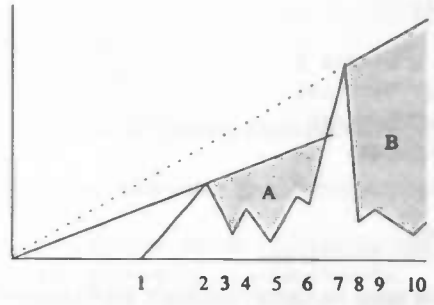


Figure 4.12: *The shaded areas are shadowed if the radar is positioned in the origin. The first shadow line, which is obtained after processing of point* 2, *shadows the terrain segments between points* $2, 3, 4, 5, 6$ *completely and the terrain between* 6 *and* 7 *is partially. The second shadow line, obtained after processing of point* 7, *shadows the terrain segments between* $7, 8, 9$ *and* 10.

A direct implementation of this pseudo code did not give correct results however. Floating point roundoff errors in the slope as well as small errors in the sample range caused many small gaps of shadow in an actually unshadowed region. In the implemented algorithm the `rc` is only updated when the following two rules apply:

- `R[n].elevation >= shadowHeight`: The current point is shadowed.
- The updated `rc` is calculated. If this `rc` would cause the next point to be shadowed then this `rc` is used, otherwise the unmodified `rc` is used. A special version of the algorithm is implemented in section 4.14.

Ground clutter during simulation

During calculation of the return of one radar beam, the first step is to apply the ground clutter. The `fTerrainRCSScanline` and `fTerrainTypesScanline` objects have been precalculated. The `fTerrainRCSScanline` signal gives the mean RCS of each terrain sample, fluctuations are added according to the statistical definitions of the terrain type. The antenna pattern was used during calculation of `fTerrainRCSScanline`, but is not used when adding the fluctuations. Because of the caching algorithm which loads blocks of precalculated terrain information (see section 4.3.2) we will generally not know which terrain types surround the current terrain sample. Actually exclusion of the antenna side lobes during calculation of the terrain fluctuation will not make much difference because we are dealing with a random quantity anyway. The ground clutter is added in the `setGroundClutter` method of the `TSimSearchRadar` object, see figure 4.13 for its implementation.

4.3.2 Precalculation

The ground clutter cross-sections and shadow information are constant throughout the entire simulation. This suggests that precalculation could improve performance. Before the simulation is started a special `TPrecalculation` object is created. First the object determines whether previous precalculated data exists and whether this data is still valid. If not then precalculation is started. Terrain cross-section calculation demands a range-azimuth order traversal (during application of the antenna pattern); the outer loop iterates over range while the inner loop iterates over 2π degrees. A radar picture however is generated in azimuth-range order; each scan line consists of the value from zero range to maximum range, and the antenna scans the 2π degrees. By precalculation we can store the values in a file in the correct order so that at any time during simulation only a fraction of the complete precalculated clutter values have to be present in memory. At any time during the first range-azimuth phase the algorithm will deal with rings of samples that lie at a specific range. If the number of samples in each ring would be kept constant while iterating over range, the inner circles would have high resolution while the outer rings would have a very coarse resolution. Thus the number of samples must increase when the range increases. Each simulated radar has a 'Maximum Delta Arc Length' (`TSimradar.maxSubMapArcLength`) parameter. The physical length of the ring of samples at a specific range $2\pi r$ is divided by this parameter to calculate the number of samples needed. The obtained high resolution rings are then used to:

1. calculate shadowed samples
2. calculate radar cross-section of each ring sample
3. apply the antenna pattern through a convolution.

Afterwards the resulting ring of samples is downsampled with a quadratic interpolation scheme.

- Application of the antenna pattern is relatively easy because the data is already in the correct order: a circular convolution is used to convolve the sampled antenna pattern ring with the ring of terrain cross-section samples. The only problem is that the number of samples in each ring has to be a power of 2, because the convolution is implemented through the use of FFTs. Normally zero padding could be used to fill the signals to the nearest power of 2, but unfortunately we need a circular convolution. So both rings are interpolated to a length equalling the nearest power of 2.

```
procedure TSimSearchRadar.setGroundClutter;
var
    rangeIndex: integer;
    RCS, rangeU, rangeClutterIndex: double;
    V: TComplex;
    tt: TTerrainType;
begin
  // PRE: clutter signal has already been convolved with ant. gain pattern
  // during precalculation
  for rangeIndex:=0 to fSizeU-1 do begin
    // calc. range
    convertDiscreteToContinue(
      rangeIndex,
      fDiscreteRangeSubCellWidth,
      fSubRangeSwathStart,
      rangeU);

    if rangeU<discreteRangeSwathStart then begin
      // some start samples for the discrete convolution
      fZ.ComplexPtr[rangeIndex]^:=Complex(0, 0);

    end else begin
      convertContinueToDiscrete(
        RangeU,
        DiscreteCellRangeWidth,
        DiscreteRangeSwathStart,
        rangeClutterIndex);
      RCS:=fTerrainRCSScanline.getInterpolatedReal(rangeClutterIndex, 1);
      tt:=TTerrainType(
        fTerrainTypesScanline.component[trunc(rangeClutterIndex)]);

      // calc the random amplitude which fluctuates the terrain RCS
      RCS:=RCS*tt.Statistic.generator.generateSample;
      V:=calculateComplexReflectionCoeff(rangeU, RCS);

      fZ.ComplexPtr[rangeIndex]^:=V;
    end;
  end;
end;
```

Figure 4.13: *The 'setGroundClutter' method.*

- The shadow problem is harder because the physical nature of shadow calculation is range ordered: for each azimuth value we would like to traverse through all the range values. However only rings of height values at a specific range are available. In the first ring (which has the smallest range) a ring of shadow slope values is formed. In each next step a new ring of shadow slope values is formed from the previous shadow slope ring. The new range will contain more samples than the previous ring, so a direct copy again is not possible. Interpolation is used to stretch the previous shadow slope ring. In the next step the shadow slopes are modified through the shadow algorithm. Figure 4.14 shows the implementation of the shadow algorithm, which now operates on rings of samples.

```
nextRange:=range+fSimRadar.DiscreteCellRangeWidth;

for subAzimuthIndex:=0 to subAzimuthBinCount-1 do begin
  // calculate the height position in the map
  z:=fCurElevationRing.elevation[subAzimuthIndex];

  slope:=ShadowRing.real[subAzimuthIndex];
  shadowHeight:=slope*range+fSimRadar.radar.height;

  if z>=shadowHeight then begin
     // not shadowed
     // new shadow slope
     newslope:=(z-fSimRadar.radar.height)/range;
     nextshadowHeight:=(newSlope*nextRange)+fSimRadar.radar.height;

     nextz:=fNextElevationRing.elevation[subAzimuthIndex];
     if nextz<nextshadowHeight then begin
       // next point shadowed
       ShadowRing.real[subAzimuthIndex]:=newslope;
     end;
  end;
end;
```

Figure 4.14: *Shadow algorithm.*

The updated shadow slope values are converted to shadow height values which are eventually written to disk in the needed range-ordered way.

After the precalculation has been finished, a number of temporary files will have been created. Each of the files will have a corresponding `TAzimuthInterval` record in memory which stores the azimuth values for which the precalculated file is valid. When a specific scan line of shadow height values is needed, the following two `TPrecalculation` methods can be used:

```
procedure getShadowMapHeightBlock(
  const azimuth: double;
  var curInterval: TAzimuthIntervalPtr;
  var shadowMapBlock: TObjectList);

procedure getShadowMapScanLine(
  const azimuth: double;
  const curInterval: TAzimuthIntervalPtr;
  const shadowMapBlock: TObjectList;
  var scanLine: TObject; var owned: boolean);
```

Before using getShadowMapScanLine to get the actual scan line, the block containing the scan line will have to be read from disk. The getShadowMapHeightBlock method will check if azimuth is contained in the shadowMapBlock block via the curInterval record. If not, the shadowMapBlock is freed and the appropriate azimuth interval record (the interval in which azimuth falls) is searched and returned in curInterval. The index of the azimuth interval record in the fshadowMapFileIntervalList determines which block file will be read, the block of scan lines is returned in shadowMapBlock. The getShadowMapHeightBlock method is very efficient because it works like a cache. Most of the time the searched azimuth value will fall in the curInterval interval, in which case no file operations are done at all.

4.3.3 Targets

All moving targets have been implemented in the TMovingTarget object.
A TMovingTarget object consists of a list of TTrajectory objects which define the target's movements. Two subclasses exist: the target class TTarget and the volumetric clutter class TVolumetricClutter. The TTarget object further contains a TTargetGeometry object which defines the shape of the target. Because the trajectory and the target shape are represented by different objects it is possible to reuse the geometry object for different targets with different trajectories.

Trajectories

The TTrajectory object contains a list of TWaypoint records. Way points are defined as:

```
TWayPoint = record
  Duration: double; \\ Duration of next spline segment in seconds
  StartTime: double;
  x, y: double; // startposition

  Tension: double; //spline tension

  PositionGiven: boolean; // position or velocity given
  VelocityGiven: tVelocGiven;
  // velocityGiven == tMagnitudeGiven : (vx, vy) valid
  // velocityGiven == tVectorGiven    : speed valid
  // velocityGiven == tNoneGiven                : Positiongiven == TRUE

  Speed: double;  // speed magnitude or
  vx, vy: double;  // velocity vector

  ax, bx, cx, dx: double;   // spline coeff.
  ay, by, cy, dy: double;
end;
```

Each way point can define the position (PositionGiven==TRUE) of a spline segment point. Or the velocity can be given in a vector (vx, vy) Lastly the velocity can be given as a magnitude in speed.
All way points are stored in a list which is sorted with respect to the way point StartTime. This way, when the position of a target at time t is needed, binary searching can be used to quickly find the way point before t. Then the spline coefficients can be used to calculate the actual position, which is done in the TTrajectory .getPosition method.

An extra boolean option `rotateTarget` was added which, if true, rotates the target according to the angle of the velocity vector after position calculation. This allows the target's bough to be correctly oriented according to its course.

Geometry

Each target consists of line and point scatterers. The point scatterers are very easy to sample, and are implemented by the class `TPointScatterer`. The line scatterers which are implemented in `TLineScatterer` however are a totally different story when it comes to implementation. The biggest difficulty arises during the sampling of the line segment. The position of each sample has to be calculated, after which the sample is added via the radar equation. Lines are easy to parametrize in Cartesian coordinates, but are hard to parametrize efficiently (without the use of trigonometric functions) in polar coordinates. We have implemented a method which iterates along the range cells. In each step the range is increased with the current range resolution Δr of the radar. After this step the angular displacement $\Delta\alpha$ between two samples is calculated. If $\Delta\alpha$ is larger than the radar's angular resolution, $\Delta\alpha$ is set to the angular resolution. In this case the range is recalculated, and a smaller $\Delta\alpha$ is taken. This method ensures that a line is rasterized in polar coordinates.

4.3.4 Radar range equation

The simplified radar equation 4.30 is used to add the individual RCS values of different scatterers. First the RCS σ must be transformed to a complex quantity γ (eq. 4.16):

$$\gamma = \sqrt{\sigma} e^{j\phi} \tag{4.38}$$

The phase of γ is generally unknown and the best we can do is to give it a random value. We can use the radar equation to calculate the amplitude of the sample. We then multiply the amplitude with a random phasor of unit power to calculate the value of C (eq. 4.29).

$$C = \frac{\lambda}{(4\pi)^{3/2} r^2} G(\theta)\gamma \tag{4.39}$$

In the program C is calculated in the `calculateComplexReflectionCoeff` method:

```
function TSimSearchRadar.calculateComplexReflectionCoeff(
  const az, r: double; const RCS: double): TComplex;
var
    b: double;
    rangePowerCoeff, amplitude: double;
begin
    b:=Sqr(fradar.GetAntennaGainPatternFarField(
      SmallestDeltaAngle(az, fAntennaAzimuth))*fradar.antennaGain);

    rangePowerCoeff:=(b/Power(r, 4))*RCS;

    amplitude:=Sqrt(rangePowerCoeff)*fRangePowerCoeffConstantSqrt;
    // calc. phasor with unit power
    fPhasorGenerator.generateRandomPhasor(result);

    result:=multComplex(result, Complex(amplitude, 0));
end;
```

In the simulation the range r will have been discretized into cells. Consequently a calculated RCS value from a target at a specific (continuous) range will rarely fall exactly into one range cell. A simple method would be to add the RCS value to the nearest neighbouring cell, but this results in large errors. A much better method identifies both neighbouring cells. Both cells share the energy of the scatterer according to their distance to the actual range. For instance if each range cell would be 1 meter wide, a sample value x at a range 2.3 would be added to range cell 2 and range cell 3. Range cell 2 would get 70 percent of the value and range cell 3 would get 30 percent. The following code implements this algorithm:

```
convertContinueToDiscrete(
  range,
  fDiscreteRangeSubCellWidth,
  RangeStart,
  rangeIndex);

// rangeIndex is a float value
ri:=floor(rangeIndex);
fr:=rangeIndex-ri;

c:=calculateComplexReflectionCoeff(
  azimuth,
  range,
  RCS*(1-fr));

d:=fZ.complexPtr[ri];
d^:=AddComplex(d^, c);

c:=calculateComplexReflectionCoeff(
  azimuth,
  range,
  RCS*fr);

// fZ contains the summed RCS samples
d:=fZ.complexPtr[ri+1];
d^:=AddComplex(d^, c);
```

4.3.5 Radar waveform and receiver

During the precalculation phase the thumbtack ambiguity (4.35) is calculated by correlation between the matched receiver filter waveform and the sent radar waveform. The sample rate at which the waveform is sampled must be high enough to treat the so-called frequency side lobes of the waveform. Internally the simulation will use a higher sample rate during calculation of the receiver response. Afterwards the signal is again downsampled before output processing is applied. The amount of oversampling is determined by the `fWaveformOversampling` parameter. The receiver response is calculated by:

4.3.6 Putting it all together

The `calculateBeam` method is called each time a beam is calculated. This method will simulate each component of the radar by calling the methods `setGroundClutter` (method 4.13), `addTargets`, `addVolumetricClutter` and `generateReceiverResponse` (method 4.15). The final result is obtained after execution of the `outputProcessing` method. The collaboration diagram 4.16 shows the execution flow during simulation:

```
procedure TSimSearchRadar.generateReceiverResponse;
begin
        // clear the samples that are not used
        // because roundoff errors accumulate in these
        // unused samples.
  fZ.clearComplex(fSizeU, fZ.ComplexWidth);

  // convolve X (the ambiguity function) with the summed RCS values in fZ
  // the result is returned in fZ

  // Instead of X we use the precalculated
  // fft of the ambiguity function X
  ComplexConvolutionKernelFFT(fZ, fXFFT);
end;
```

Figure 4.15: *The 'generateReceiverResponse' method.*

The collaboration diagram has been simplified somewhat from the actual source code, but the main sequence of events is the same:

1 Precalculation is started before the simulation executes.

- **1.1** The positions of the targets are written to a file. This file can be used to compare the real target positions to measured positions.
- **1.2** Each `SimRadar` object will have to do some precalculation.
  - **1.2.1** The ambiguity function (4.35) is calculated.
  - **1.2.2** A special object `TPrecalculate` is created which will build the shadow and terrain clutter files.
    - **1.2.2.1** `buildMapFile()` will build the shadow and terrain clutter file in blocks.
    - **1.2.2.2** `buildLandMask()` generates a file which can be used to mask all land pixels on a radar image. This land mask is used by the fusion algorithm.

2 `simulate()` starts the actual simulation.

- **2.1** First each `SimRadar` object is initialized: buffers are created, random number generators are created, and the antenna position is initialized.
  - **2.1.1** All probability distributions in a project have a seed-number. If the seed-number is not reset then each simulation run will produce a different output.
  - **2.1.2** The bounding circle radius of each target is calculated. The radius value is used in the target clipping algorithm.
- **2.2** Simulation starts. A loop will call `calculateBeam()` until the specified simulation time interval is exceeded.
  - **2.2.1** Each time `calculateBeam()` is called the antenna position of the radar is advanced.
  - **2.2.2** The antenna position determines which precalculated shadow and terrain clutter blocks are cached.
  - **2.2.3** Terrain clutter is added to the output signal.
  - **2.2.4** Target returns are added.
  - **2.2.5** Rain clutter values are calculated.

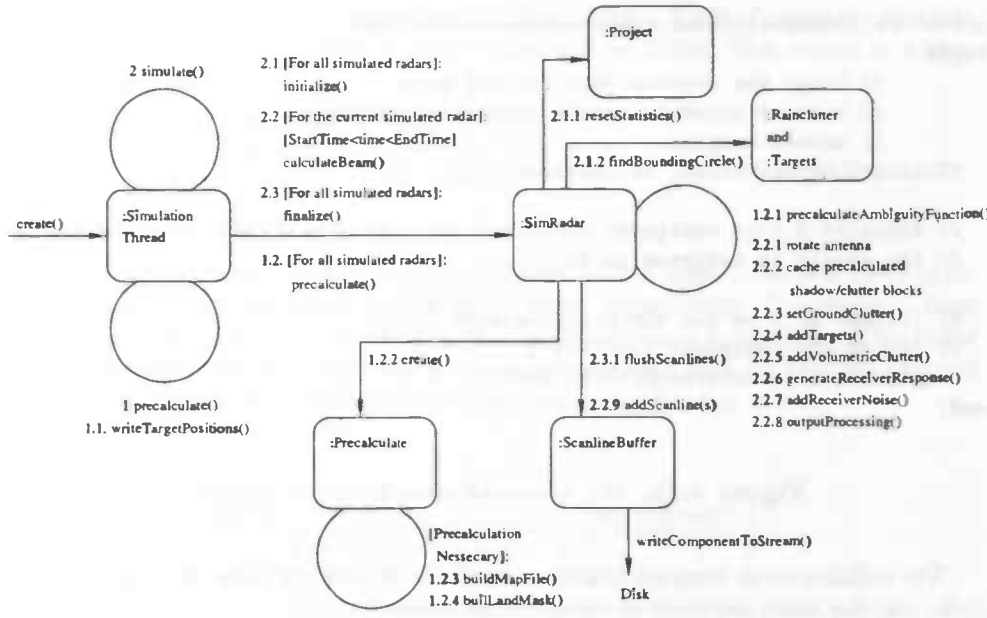


Figure 4.16: *Simulation collaboration diagram*

2.2.6 The receiver response is calculated from the output signal.

2.2.7 Receiver noise is added.

2.2.8 The signal is scaled and quantized, this results in the final radar scan line.

2.2.9 The scan line is added to a buffer. The buffer is observed by the visualization algorithm which will update the screen. When the scan line in no longer used it will be written to disk.

2.3 Before closing the simulation all buffers need to be written to disk. Allocated memory is freed up.

There are some details which have not been mentioned yet in this chapter. One of the hardest problems was to get a proper physical result. After each step it was clear that the operation in general was correct, however this is not enough in a physical simulation. There were a lot of problems to get the energy in the signals right. For instance, the standard quadratic interpolation scheme for complex signals does not behave correctly if we take a look at the power of the interpolated samples. A special `TSignal` method `stretchFromComplexInterpolateAmp` had to be developed to resolve this problem. In most other cases where the energy of the signal was incorrect, the energy was scaled afterwards by the `setComplexEnergy` method of the `TSignal` object. It would be better of course to modify the equations to obtain a signal with the correct energy directly, rather than to scale the signal afterwards.

Chapter 5

Image fusion and Tracking

From the point of view of tracking, we must do the opposite of simulation. Now we have (simulated) data and from this data we have to reconstruct the targets and eliminate the noise. Unfortunately it is very hard to completely remove the noise while keeping small targets. Thus it is necessary to build a very robust tracking algorithm, which does not fail if there is some noise in its input.

5.1 Mathematical modelling

5.1.1 Time discretization

The fusion application will receive data streams from different radar stream files which contain precalculated simulated data or real recorded data. When the fusion application is used in real time radar stations will supply the data at different rates. This synchronization problem will have to be simulated when stream files are used. One radar will deliver a complete azimuth beam each $1/PRF$ seconds. As mentioned earlier the antenna rotation is negligible during this period. Thus we could take a time step of $1/PRF$. In each step a new azimuth-range beam is calculated after which the antenna position is updated. Problems will arise when two or more radars have to be simulated simultaneously, for example:

We have two radars A and B, and use the time step $1/PRF_a = 0.25$ sec:

Radar Name	PRF (Hz.)	Time step (sec.)
A	4	0.25
B	3	0.33..

When we run the simulation radar A is calculated each time step, however calculation of radar B however will be subject to timing errors:

Time (sec.)	timing error radar A (sec.)	timing error radar B (sec.)
0	0	0
0.25	0	0.08
0.5	0	0.17
0.75	0	0.08
1.0	0	0

A solution to this problem can be found if we take the time step of a radar X to be the least common multiple (LCM) of the PRFs of the involved radars divided by the PRF of the radar X:

$$\Delta i_x = \frac{LCM}{PRF_x} \tag{5.1}$$

If for instance we have three radars A, B, C with a PRF of respectively 1500, 2000 and 2400 Hertz, the LCM of these values will be 12000. This results in a time step for each radar of:

Radar Name	PRF (Hz.)	Δi_x
A	1500	8
B	2000	6
C	2400	5

In this example each time step is 1/12000 sec, each 8 steps radar A is calculated, radar B and C are calculated each 6 and 5 steps, respectively. The required time step quickly becomes very small, but this is actually not a problem. The radar calculation times can be sorted into a queue. During each step we take the value of the head of the queue. The step value then increases with this value.

Step	Calculation	Sorted list	Step advance
		A=0, B=0, C=0	0
0	A, B, C	C=5, B=6, A=8	5
5	C	B=6, A=8, C=10	1
6	B	A=8, C=10, B=12	2
8	A	C=10, B=12, A=16	2

The LCM algorithm works well when the PRFs of the used radars are not relatively prime. When they are the LCM value will get very large, and possibly an integer overflow could occur.

5.1.2 Target extraction

The tracking algorithm to be described in section 5.1.4 requires a stream of target detections as its input. In this section a segmentation algorithm is developed which extracts targets and passes these detections to the tracking algorithm. Each radar will have its own instance of the segmentation algorithm. The segmentation algorithm works relatively simple. First each new received scan line of range samples is thresholded. The threshold level must be set high enough to eliminate as much noise as possible, but should also be set low enough to keep the smallest target. If a terrain map is available it can be used as a mask to set all terrain samples to zero. After thresholding the scan line is passed to a modified region growing algorithm, which segments all disjoint objects into targets. The original segmentation algorithm called 'TargetExtractor' was supplied by the company Sodena, but was not really efficiently coded (see implementation). The main difference between a standard region growing algorithm is that the 'TargetExtractor' algorithm is scan line oriented:

- The growing of a region is stopped if the region's angular width becomes larger then 2π.
- With each region its number of samples and the total of all sample values are stored. When two regions are merged these values are added. After completion of the region, these stored values can be readily used to calculate the mean sample value of the region.
- The actual sample positions are also stored in the region, to allow drawing of the target in a visualization.

When a target has been extracted some simple heuristics can be used to decide whether to discard the target or to pass the target on to the MHT algorithm.

1. If the target consists of only one sample, the target is discarded.

2. If the targets angular width is larger then π, the target is discarded.

The rules above will remove a large number of false targets. The first rule however is also very dangerous, because in reality there will be ships that are only detected in one sample. The rule was mainly added because without it far too many false targets would be passed on to the MHT algorithm, which would lead to very slow execution. In an actual application the MHT algorithm and segmentation would have to work in real time! Clearly a better noise filtering technique is needed.

5.1.3 Tracking of a single target without false alarms

By restricting the number of targets to only one and removing false alarms the tracking problem is greatly simplified. The association problem of determining which target belongs to which track completely disappears. We still have the problem however that in a number of consecutive radar scans the target could be missing in some of these scans (e.g. by shadowing). During detection misses the track of the target will have to be extrapolated to allow monitoring of the ship. For this task Kalman filtering is used. There are several advantages to Kalman filtering that make tracking easier:

1. A Kalman filter accounts for measurement errors: a new measurement will update the filter with a certain weight.

2. The Kalman filter not only extrapolates position but also calculates a measurement of accuracy for the extrapolation: if for some time there has not been a filter update the uncertainty of the extrapolation increases. The accuracy measurement is used to decide how much weight is applied to a new detection during a filter update. When the accuracy of the estimate falls below a certain threshold (e.g. after several minutes of missed detections) we can decide to delete the track.

3. The target dynamics (or equations of motion) are included in the filter: if the a ship's maximum velocity is known we can determine a so-called detection gate where, physically, this ship can exist.

An alternative to Kalman filtering would be to hold a large number of previous detections in memory and then fit some sort of spline or polynomial through these detections. This could be a better solution because in most tracking hardware memory is more abundant then free CPU time. However polynomials for instance do not extrapolate very well (large oscillations can occur outside the source domain). Also splines would not give an accuracy measurement and without such a measurement the data association problem (see the next section) becomes harder to solve.

The Kalman filter which is used in this application is a very simple one. The x and y coordinates of a detection are separated and a Kalman filter is used for each coordinate. A better Kalman filter design would directly use the polar coordinates of a target (because the angle measurement is often less accurate than the range measurement) but then we would have to use a nonlinear Kalman filter; this will have to be studied. The following formulas describe the Kalman filter for extrapolation of the x-coordinate of the target. The Kalman filter for the y-coordinate is identical, but with y substituted for x, a complete definition of the Kalman filter can be found in appendix C. In our filter the state vector X measures position as well as speed and is defined as:

$$X_n = \begin{bmatrix} x_n \\ v_n^x \end{bmatrix} \tag{5.2}$$

The targets dynamic model is defined as:

$$x_{n+1} = x_n + T v_n^x \tag{5.3}$$
$$v_{n+1}^x = v_n^x + u_n \tag{5.4}$$

These equations define the state transition matrix Φ and the dynamic model driving noise vector U:

$$\Phi = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \tag{5.5}$$
$$U_n = \begin{bmatrix} 0 \\ u_n \end{bmatrix} \tag{5.6}$$
$$X_{n+1} = \Phi X_n + U_n \tag{5.7}$$

In these formulas x_{n+1}, v_{n+1}^x are the position and velocity respectively at time $n+1$, u_n is a random noise process to allow for a non constant velocity of the target. The radar will measure only the position of the target, thus the measurement matrix Y_n becomes:

$$Y_n = M X_n + N_n \tag{5.8}$$
$$M = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{5.9}$$
$$N_n = \begin{bmatrix} r_n \end{bmatrix} \tag{5.10}$$
$$Y_n = \begin{bmatrix} y_n \end{bmatrix} \tag{5.11}$$
$$\Rightarrow y_n = x_n + r_n \tag{5.12}$$

with M the observation matrix and N_n the observation error. The last two matrices needed to define the Kalman filter are R_n which is the covariance matrix of N_n and Q_n which is the covariance matrix of the driving noise vector U_n:

$$Q_n = \begin{bmatrix} 0 & 0 \\ 0 & \overline{u_n^2} \end{bmatrix} \tag{5.13}$$
$$R_n = \begin{bmatrix} \overline{r_n^2} \end{bmatrix} = \begin{bmatrix} \sigma_x^2 \end{bmatrix} \tag{5.14}$$

Q gives the magnitude of the target trajectory uncertainty. In the above definition of Q, however the trajectory uncertainty will not increase with time. A small adaptation of Q, i.e. letting it depend on T, solves this problem:

$$Q_n = \overline{u_n^2} \begin{bmatrix} \frac{1}{3}T^3 & \frac{1}{2}T^2 \\ \frac{1}{2}T^2 & T \end{bmatrix} \tag{5.15}$$

This particular form of the matrix Q comes from the definition of the Singer Kalman filter, which is described in [22] (p. 88).

When the Kalman filter is initialized we will need a start estimate X^* of state vector X. This can only be done after the first two observations because an estimate of the velocity becomes available only then.

5.1.4 Tracking of multiple targets

Now that the tracking of a single target is possible, some enhancements are needed before we can track multiple targets. In particular we have the problem of data association: if an echo is detected how do we know to which track it belongs?

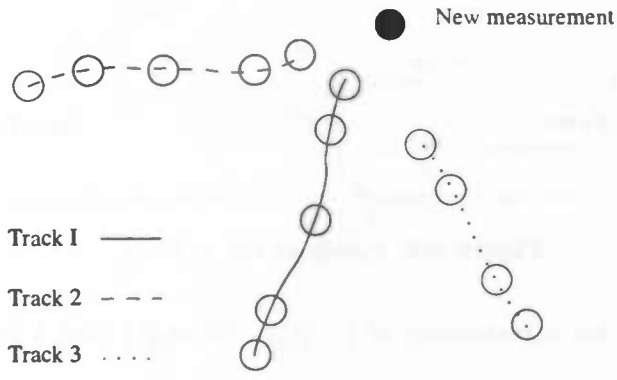


Figure 5.1: *Data association problem: Does the new measurement belong to track 1, track 2 or track 3?*

A very simple solution to the association problem is called the 'Nearest Neighbour Association' (NNA) method. A new measurement is associated to the track of the nearest measurement. Although the algorithm for NNA is very simple and fast, the algorithm will perform very poorly if there are many targets and tracks packed closely together as is the case in many large ports. Even a human operator, looking at figure 5.1, will not be completely sure whether the new measurement belongs to track 1, track 2 or track 3. He will however be able to indicate that the new measurement is more likely to belong to track 1 or track 2 than to track 3. Only when additional new measurements have been received it should be clear how to associate the new measurement. Possible combinations are kept in memory, and when one of the combinations clearly becomes the right one, the other combinations are no longer needed. It is this idea that is implemented in the multi hypothesis tracking (MHT) algorithm. All hypotheses are stored in a tree and unlikely branches in the tree are pruned. Although MHT is not a new algorithm and has been used in military applications, few commercial trackers use the algorithm because of the computational load. The framework of the MHT algorithm is as follows:

1. When at $T = 0$ the first measurement M_0 is received, two hypotheses are formed. $H_{T,0}$: the measurement belongs to new track A, or $H_{T,1}$ the measurement is a false alarm (noise). If the false alarm probability $P(FA)$ is 0.2 then $P(H_{0,0}) = 0.8$ and $P(H_{0,1}) = 0.2$.

$$\begin{aligned} H_{0,0} &= \{(A, (M_0))\} \\ H_{0,1} &= \{(FA, (M_0))\} \end{aligned}$$

$$\begin{aligned} P(H_{0,0}) &= 0.8 \\ P(H_{0,1}) &= 0.2 \end{aligned}$$

   These first two hypotheses form the two roots of the hypotheses tree which is expanded in each step. The initial tree is presented in figure 5.2.

2. After the second measurement M_1 has been received on $T = 1$, the existing hypotheses are expanded: The new measurement can be a false alarm, a detection belonging to track A or the start point of a new track B. If the $P(FA)$ is again 0.2, the new track probability $P(N)$ is 0.3 and the association

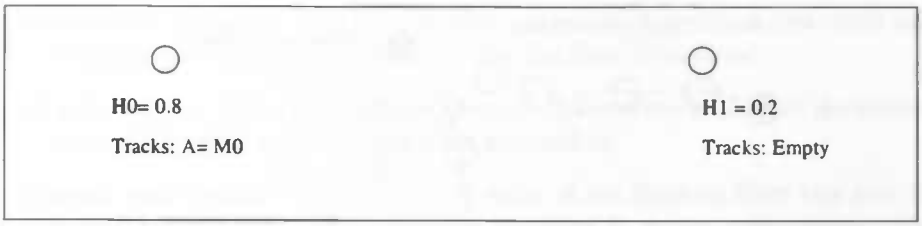


Figure 5.2: *Hypothesis tree at* $T = 0$.

to track A has a probability of $1 - 0.2 - 0.3 = 0.5$ then 5 hypotheses are formed:

$$\begin{aligned}
H_{0,0} = \{(A,(M_0))\} &\Rightarrow H_{1,0} = \{(A,(M_0,M_1))\} \\
&\Rightarrow H_{1,1} = \{(A,(M_0)),(B,(M_1))\} \\
&\Rightarrow H_{1,2} = \{(A,(M_0)),(FA,(M_1))\} \\
H_{0,1} = \{(FA,(M_0))\} &\Rightarrow H_{1,3} = \{(FA,(M_0,M_1))\} \\
&\Rightarrow H_{1,4} = \{(FA,(M_0)),(B,(M_1))\}
\end{aligned}$$

$$\begin{aligned}
P(H_{1,0}) &= P(H_{0,0}) \cdot 0.5 = 0.4 \\
P(H_{1,1}) &= P(H_{0,0}) \cdot 0.3 = 0.24 \\
P(H_{1,2}) &= P(H_{0,0}) \cdot 0.2 = 0.16 \\
P(H_{1,3}) &= P(H_{0,1}) \cdot 0.2 = 0.04 \\
P(H_{1,4}) &= P(H_{0,1}) \cdot 0.8 = 0.16
\end{aligned}$$

After the tree has been updated with these hypotheses it will look like figure 5.3.

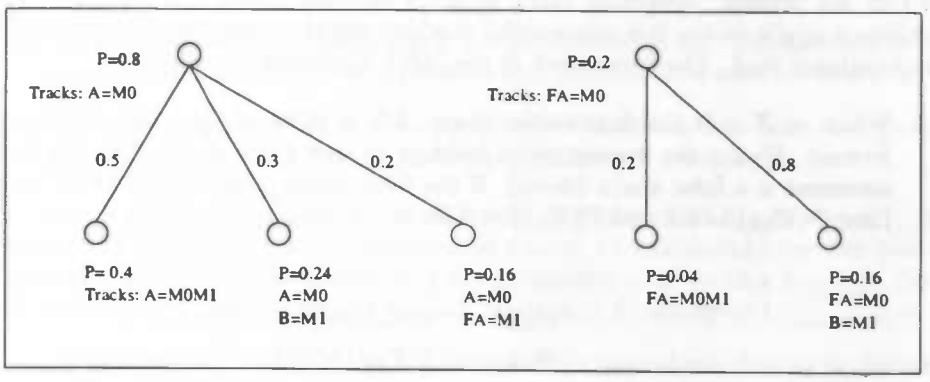


Figure 5.3: *Hypothesis tree at* $T = 1$.

3. It is clear that the hypothesis tree grows exponentially when even more detections are added. Full expansion of the tree in the third step yields 15 hypotheses.

To keep the size of the tree within limits pruning techniques are needed:

- The first technique is to use the accuracy and position estimate of the Kalman filter of each track to form a gate. Only detections that fall within the gate will be associated with the track.

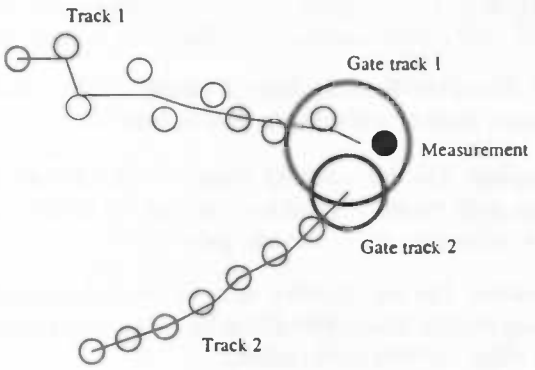


Figure 5.4: *Gates used to simplify association: The new measurement can only be associated with track 1.*

- All hypotheses whose probabilities fall below a threshold are deleted.
- All tracks whose probabilities fall below a threshold are deleted. The probability of a single track is the sum of probabilities of all hypotheses that contain the track.
- As the radar scans each track will only be updated in time steps of about the rotation period of the radar. A minimum track update time interval ensures that the track is only updated after a new scan.
- A maximum track update time interval is stored. When a track is not updated every x seconds all hypotheses that contain the track incur a miss penalty. This causes wrong or dead tracks to be eliminated.
- A maximum number of hypotheses can be set. By sorting the hypothesis list on descending probability all less likely hypotheses are deleted each time the number of hypotheses gets too large.

Although these pruning techniques already greatly decrease the number of possible hypotheses, the current MHT algorithm is still too slow. Even more pruning is achieved when methods described in the literature [2] and [3] will be implemented:

Clustering Clusters are groups of tracks which have overlapping prediction gates. If clusters are formed the data association problem is simplified. When a new detection is added, the cluster, in which the new detection falls, is identified. Now the detection can only be associated with the tracks in the cluster. In effect the MHT algorithm is parallelized, each cluster could be handled separately.

N Observation Pruning The basic idea of N-observation pruning is to use tracks in the current most likely hypothesis to delete other tracks based upon conflict over observations received N observations back in time.

In spite of the current algorithm's computational load, the algorithm is very memory efficient. The data structures are described in 5.2.2. The current MHT algorithm has several parameters that influence tracking behaviour:

Dynamic model noise U This Kalman filter parameter determines how fast the uncertainty of a track increases with time, if the track is not updated.

Measurement variance This Kalman filter parameter determines how much value is attached to a new measurement when the filter is updated.

Start gate radius When the Kalman filter is initialized the initialized uncertainty value will be used scaled to the start gate radius.

Maximum gate radius The uncertainty value of the Kalman filter can give rise to a very large gate radius, possibly resulting in wrong associations. This parameter sets maximum value for the gate radius.

Minimum gate radius The uncertainty value of the Kalman filter can give rise to a very small gate radius, quick loss of tracking can be a result. This parameter sets minimum value for the gate radius.

New track probability factor This parameter relates to the chance that the new measurement will create a new track. It is however not the probability. See point 3 of the probability calculations below for the definition of this parameter.

False alarm probability P_{FA} The probability that the measurement is noise. If follows that the chance of detection satisfies: $P_D = 1 - P_{FA}$.

Maximum track update time When a track has not been updated at least once in a complete radar scan, the track will have missed a detection. If the time since the previous update of the track is longer than the 'maximum track update time', all the hypotheses containing this track will incur a miss penalty.

Miss penalty See the previous parameter.

Minimum track update time A track cannot be updated continuously (in theory it could, but the target would have to have a very high speed and special trajectory). A measurement can only be associated to a track if the time since the previous track update is larger than the 'minimum track update time'.

Minimum hypothesis probability If the probability of the hypothesis falls below the threshold then the hypothesis and possibly its tracks are deleted.

Minimum track probability If the probability of the track falls below the threshold, then the track and all hypotheses containing it are deleted.

When a parent hypothesis H^p is expanded into new child hypotheses H_i^c, the probabilities of each of the new hypotheses i must be calculated from the parent hypothesis. This is done by multiplying the probability of H^p with a factor f_i. Note that if there are N new hypotheses formed, the factors f_i sum up to 1: $\sum_{i=0}^{N-1} f_i = 1$.

1. *F*alse alarm expansion: If the measurement is a false alarm then its probability is P_{FA} and the new hypothesis will have probability $P(H_0^c) = P(H^p) \cdot P_{FA}$.

2. *T*rack addition: Now we have $\sum_{i=1}^{N-1} f_i = 1 - P_{FA} = P_D$. The new measurement will lie in the gate of the updated track i. If the gate's radius is R_i^g and the measurement's distance to the center of the gate is R_i^m, then we know $0 \leq R_i^m < R_i^g$. If R_i^m is closer to zero then the chance of the measurement belonging to the track is higher. If R_i^m is near R_i^g then the chance that the measurement belongs to the track is almost zero. A so called hit factor h_i can be defined for each track addition i which is R^m scaled between 1 (when $R_i^m = 0$) and 0 (when $R_i^m = R_i^g$). The sum of all hit factors is stored in h : $h = \sum_{i=1}^{N-2} h_i$.

3. *T*rack creation: We could define the track creation probability to be constant, however when there are many child hypotheses with track additions possible it is desirable to lower the track creation probability. Presumably the chance that the detection belongs to a track is higher than the chance that the detection belongs to new track. This problems is solved by introducing a 'new track probability factor' t. The value t is added to the sum of the hit factors h. In this way t will get relatively smaller when h increases.

4. Now we are able to normalize the hit factors and t to probabilities:

$$\begin{array}{l} i = 0 \\ 0 < i < N-2 \\ i = N-1 \end{array} \left\{ \begin{array}{l} P(H_i^c) = P(H^p) \cdot P_{FA} \\ P(H_i^c) = P(H^p) \cdot \frac{h_i P_D}{h+t} \\ P(H_i^c) = P(H^p) \cdot \frac{t P_D}{h+t} \end{array} \right.$$

5.1.5 Fusion

At the start of the project it seemed that tracking would be easier once all the radar streams are fused into one big stream. During the development of the MHT algorithm, it became evident that the tracking algorithm itself can be used to fuse the radar streams. The segmentation algorithm just supplies the MHT algorithm with detected target positions. In this way it does not matter if there are targets that are detected by different radars. Even the synchronization problem vanishes because each target that is extracted will be given its own time tag by the target extractor algorithm. When two radar posts observe the same target this should result in more frequent updates of the correct hypothesis. A small problem is that the radars will have to be calibrated to establish their relative positions. If the segmentation algorithm is implemented embedded at the radar station then a global timing reference is also needed to calculate the correct time-tag value for the extracted targets.

5.2 Implementation

5.2.1 Kalman filter

The Kalman filter has been implemented through a base class `TKalmanFilter`. Derived classes can implement specific Kalman filters. The most important parts of the definition of the class are:

```
TKalmanFilter = class
protected
  ...
  procedure updateTrans; virtual; abstract;
  procedure updateQ; virtual; abstract;
  procedure updateR; virtual; abstract;
  procedure updateM; virtual; abstract;
public
      ...
  procedure predict(const aTime: double); virtual;
  procedure correct(const aTime: double); virtual;
  procedure TimeStep; virtual;

            procedure copyStateFrom(k: TKalmanFilter); override;

  property Sstart: TDynamicMatrix;
  property Xstart: TDynamicMatrix;;
```

```
    property Y: TDynamicMatrix read fY;
    property XPredict: TDynamicMatrix;
    property SPredict: TDynamicMatrix;
  end;
```

Spredict, XPredict After a call to predict the Kalman predictions equations (C.3) and (C.6) are used to calculate `XPredict` and `SPredict` .

Correct The measurement must have been placed in `Y` before `correct` can be called. `Correct` will use Kalman filter update equations C.4 and C.9 to update the Kalman filter state. These equations a include matrix inversion. In the general base class we do not know what the dimensions of these matrix are so a general matrix inversion method (Gaussian elimination with pivoting) algorithm is used. In a simple filter like the `TProgressFilter` such a method is of course overkill.

TimeStep Resets the base time, from which a prediction is made, to the time on which the correction was made.

CopyStateFrom Is used to make a copy of the Kalman filter. When in the MHT algorithm tracks are expanded with new detections, these track will first have to be copied including their Kalman filters.

Sstart The start covariance matrix, see appendix C.

Xstart The start state matrix, see appendix C.

updateM, updateQ, updateR, updateTrans Must be implemented in derived classes to define the Kalman filter.

A Kalman filter `TProgressFilter` has been implemented which linearly predicts a coordinate and a velocity, only coordinate measurements are needed. The `TTrackingFilter` class uses two `TProgressFilter` to predict x and y. Another Kalman filter called the Singer Kalman filter has been implemented but not tested. This filter also includes acceleration in the state vector. In theory this filter should be able to predict curved trajectories much better then the simple `TProgresFilter`.

5.2.2 MHT algorithm

The MHT algorithm uses three main data entities hypotheses, tracks and detections. A detection stores the position of a measurement, a track stores a list of detections and a hypothesis stores a list of tracks.

It would be very memory inefficient to implement the track as an actual list of detections, since a lot of the different tracks in the MHT algorithm will resemble each other. A solution is to use a track tree. A track tree does not store the tracks themselves but rather stores all different sequences of detections.

E.g. if we have 20 detections $D_0, .., D_{19}$ then a track is only identified by its base index and a pointer to a node in the track tree which is its last track link:

Track	base index	last track link
(D_0, D_1, D_2)	0	C
(D_0, D_2, D_4)	0	M
(D_1, D_2, D_3)	1	C
(D_{18})	18	A
(D_8, D_6, D_5, D_4)	4	K

When a track is expanded with a new detection, the track tree must be updated. This however is a very simple procedure:

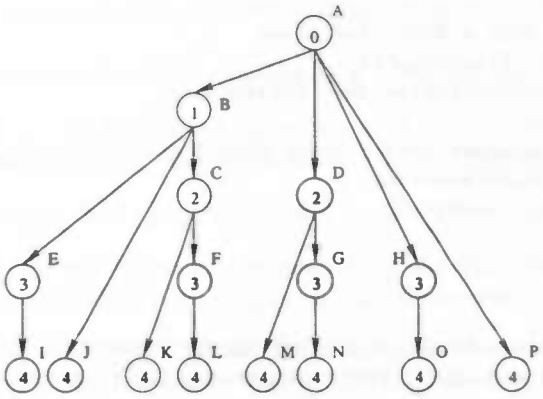


Figure 5.5: *Tracktree: The number in each node is an offset index, which together with a track base index results in the index in the detection list. The letter above each node is an identification of the node.*

1. A new detection with index D_i is added to the track.
2. The track's old tree node (called last track link) T_p is identified. The track has a base index b.
3. (a) If a child T_c of T_p already contains the requested offset $i - b$ then T_c becomes the new last track link.

   (b) If T_p has no child with an offset $i - b$ the child T_c is created with this offset and becomes the new last track link.

Additionally each node (or track link) in the tree stores the number of ending tracks at that node. This counter is used a reference counter. If the counter is zero and the node has no children it can be deleted. The track tree is implemented by a data structure TTrackLinkRec:

```
TTrackLinkRec = record
  startingTrackCount: integer;
  previous: TTrackLinkRecPtr;
  successors: TLinkElementPtr;
  detectionOffsetIndex: integer;
end;
```

The tracks themselves are defined by the TTrackRec record:

```
TTrackRec = record
  // global linked list element
  TrackListElement: TLinkElementPtr;

  refcount: integer;
  length: integer;
  trackingFilter: TTrackingFilter;
  // sequence of detections
  last: TTrackLinkRecPtr;
  lastUpdateTime: double;
  DetectionBaseIndex: integer;
  lastGateRadius: double;
  // total track probabilty among all hypotheses
  prob: double;
  // used to normalize probability
  lastHitFactor: double;
```

```
    // this track with a detection added
    detectionadded: TTrackRecPtr;
    // just for identification when drawing
    color: integer;
    // list of hypotheses that contain this track
    hypothesis: TLinkElementPtr;
    hypothesisCount: integer;

    tmp: pointer;
  end;
```

The detectionadded field of a track record stores a link to the same track but now expanded with the current measurement. When the hypothesis tree is expanded this field is used to update the tracks in the hypothesis:

1. If the detectionadded field of a track is not yet defined (nil) then the track is copied and a new measurement is added to the track and a reference to the copied and updated track is stored in the detectionadded field of the old track.

2. If the detectionadded field of a track is already set then the stored reference is used. No duplicate track is created.

Hypothesis initialization starts with the creation of two hypotheses:

```
procedure TMHTracking.initTracks(const falseAlarmProb: double);
var
  newTrackLink; TTrackLinkRecPtr;
  newHyp: THypothesisRecPtr;
  newTrack: TTrackRecPtr;
begin
       // First TrackLink points to first detection
  newTrackLink:=createTrackLink(0);

  fFirstTrackLink:=newTrackLink;

  // Hypothesis: first detection is a new track
  newHyp:=createHyp(1-falseAlarmProb);
  newHyp^.updateCount:=1;

  // new track with one detection (offset 0)
  newTrack:=createTrack(1-falseAlarmProb, newHyp, newTrackLink, 0);
  newTrack^.prob:=newHyp^.prob;

  // Hypothesis: false alarm
  newHyp:=createHyp(falseAlarmProb);
  newHyp^.updateCount:=1;

  // Total probability
  fTotalHypothesisWeight:=1;
  fOldTotalHypothesisWeight:=1;

  addNewhypotheses;
end;
```

When a new hypothesis is created it is added to the new hypothesis list, which is sorted on probability. When addNewhypotheses is called this sorted list is merged

with the original hypothesis list. The sorting and merging has been implemented efficiently.
After a new measurement becomes available, the following steps are taken:

1. ExpandTracks. All tracks are expanded with the new measurement. The detectionadded field of each original track will contain the updated track.

2. Iteration through the hypothesis list starts.

3. If the probability of a hypothesis is larger than the fMinimumHypothesisProb then the hypothesis is expanded into at least three new hypotheses.

   (a) New hypotheses are created for each track in the original hypothesis (hyp) that has its detectionadded field assigned.

   ```
   // adds hyp. with existing track additions
   continueExistingTracks(Hyp, newTrackProb, falseAlarmProb);
   ```

   (b) A new hypothesis is created from the original hypothesis (hyp) in which the current detection starts a new track. The newly created track is returned in the var-parameter newTrack. When the next hypothesis is expanded with 'a new track' assignment, the now assigned newTrack will be used and thus only one new track will be created during the addition of each new measurement.

   ```
   // adds hyp. with new track
   constructNewTracks(Hyp, newTrack, newTrackProb);
   ```

   (c) Lastly, the new detection is treated as a false alarm. Because this is the last possibility, the current hypothesis hyp is not needed anymore and can be modified.

   ```
   // updates hyp to contain a false alarm
   addFalseAlarm(Hyp, falseAlarmProb);
   ```

4. If the hypothesis probability is smaller than fMinimumHypothesisProb it is deleted. Tracks which are not used in other hypotheses are deleted too.

5.2.3 Visualization of one radar stream

Computer images are mostly built up on a Cartesian grid, radar images however use polar coordinates. During the drawing of a complete radar image, r plays the role of range and τ the role of azimuth or bearing. As the antenna rotates τ is increased.
The image on the screen has to be drawn in the same manner as the radar image, thus range line after range line. A simple algorithm just parametrizes the boundary of the maximum range circle. After each line the angle is increased with a very small value $d\tau$.

```
dDrawAngle:=tan(1.0/maxRadius);
DrawAngle:=0
while DrawAngle<2*Pi do begin
  px:=round(sin(StartdrawAngle)*maxRadius)+centerx;
  py:=round(-cos(StartdrawAngle)*maxRadius)+centery;

  drawLine(centerx, centery, px, py);
  DrawAngle:=DrawAngle+DdrawAngle;
end;
```

If the $d\tau$ is not small enough gaps will be visible between the lines. In fact even with very small $d\tau$ a pattern of gaps is still visible. An efficient method works without the use of floating-point arithmetic by using the midpoint circle algorithm and Bresenham's line drawing algorithm. The boundary of the maximum range circle is rasterized with the circle midpoint algorithm and stored in an array. When an image is rasterized an index iterates along the boundary pixel positions in the array. The scan line can then be drawn from the center of the circle to the stored boundary point. An extra precalculation step is needed because not every scan line will have the same length in pixels, in spite of the fact that all the scan lines do cover the same range. A horizontal scan line will consist of more pixels then a diagonal scan line. During precalculation, the length in pixels of each scan line is stored together with the boundary position information. Because the boundary positions of the circle are always adjacent, the scan lines to center of the circle will also be drawn adjacent. The algorithm in pseudo code looks like this:

```
VAR
  C: array of (x, y, angle)
  L: array of (x, y)

INITIALIZE
  C(0..m).(x, y, angle):=MidpointCircle(0, 0, r)
  PreviousAngle:=0
  N:=0
  L:=BresenhamLine(0, 0, C(N))
```

First all the coordinates of the pixels on the boundary of the circle are stored in C. The positions of the pixels that lie on the line from the center of the circle to the first point in C are stored in L. This initialization step is only needed once in an application.

```
PROCEDURE Plot(PreviousAngle, N, EndAngle)
  WHILE PreviousAngle<EndAngle
    DrawRangeLine(L, C(N).(x, y))
    N:=N+1
    PreviousAngle:=C(N).angle
  END
END
```

As long as the PreviousAngle variable does not pass the given value EndAngle, lines are drawn from the center of the circle to points on the boundary.

```
PROCEDURE DrawRangeLine(L, (ex, ey))
  (x, y)=(centerx, centery)
  M:=0
  WHILE NOT bresenham_done
     DrawPixel((x, y), data)
     FillPixelAndGaps(M, L(M).(x, y), (x, y))

     (x, y)=NextPixelOnBresenhamLine((0, 0), (ex, ey))
     M:=M+1
  END
END
```

The while loop in the pseudo code above will iterate through all the points on the line from the center of the circle and the boundary. At some points there will still be gaps between two consecutive scan lines, however the gaps are only one pixel wide and can be eliminated with a few simple tests. The positions of the previous drawn scan line are stored in the array `L`, and when a new scan line is drawn the new and the old positions are compared. If the difference is more then one pixel a gap exists. These pixels can then be filled in by the `FillPixelAndGaps` procedure. To find the position of the gaps first `dx` and `dy` are calculated from the current pixel position and the position of the pixel in the previous scan line at the same range (In the pseudo code above this is `dx:=L(M).x-x, dy:=L(M).y-y`). When the image is built counterclockwise (the angle decreases) these tests are:

Image octant	test	when TRUE gap w.r.t (x, y)
1	(dx<-1) or (dy<-1)	Gap to the top left
2	(dx<-1) and (dy=0)	Gap to the left
3	(dx<-1) or (dy>1)	Gap to the bottom left
4	(dx=0) and (dy>1)	Gap to the bottom
5	(dx>1) or (dy>1)	Gap to the bottom right
6	(dx>1) and (dy=0)	Gap to the right
7	(dx>1) or (dy<-1)	Gap to the top right
8	(dx=0) and (dy<-1)	Gap to the top

If for instance test 1 is true then a gap exists one pixel to the top left pixel from the current (x, y) coordinate. This pixel will then be filled by the `FillPixelAndGaps` procedure.

Chapter 6

Global software model

The software is built in Delphi object PASCAL. Object PASCAL combines the advantages of object oriented design with clear readable PASCAL code. The software model consists of two parts. There is one part which manages the data, while the other part consists of the different data types which implement a specific task (in this case the simulation). It took a lot of effort to keep the two parts separated and generalized, but we will see that modifications of the simulation implementation will now automatically be reflected in the GUI.

6.1 Model-View architecture

In previous application development, often the same problems arise involving GUI (graphical user interface) development and data management. In those cases the solution was to merely program a specific solution to each problem. The common problems during GUI development were:

- A useful GUI should include options to save, load, close and create new items. If a new item has been modified by the user, the program should ask whether to save the item. If the item has not been saved before a `saveAs` action has to be executed.
- The GUI menu system should be context sensitive. If the user selects a different opened item (e.g. simulation project, terrain, radar definition) a close action should close that selected item. Also the 'new item' option must only include items that can be created as children of the current selected item (e.g. a simulation environment can only be created if there is an active project).
- Each item will have its own set of editors. If for example a radar item is selected, the system has to update its menu structure to include options to edit the selected radar.
- It is very important to keep the GUI synchronized with the actual data it displays. If for example a radar item is deleted, it should also be removed from the terrain view and the simulation environment. If the filename of an item is changed all editors which display the item should have their displayed titles updated.

Common data management problems are:

- It should be possible to keep all objects persistent, in other words to read and write the objects (including referenced objects!) to a file.
- If one object is destroyed all objects witch rely on this object (e.g. a 'simulated radar' object depends on a 'radar definition' object) have to be destroyed as well.
- The functionality objects (in this case the objects that implement the simulation) should not rely on the graphical user interface. The GUI code must be well separated from the actual problem solving code.

A good solution to some of the problems mentioned above is to design the software according to the 'model-view' design pattern which is described in among others [7]. This OOP method clearly divides software design into the GUI (view) and the independent application objects (model). The separation is accomplished as follows:

A base class `TTransceiver` has been developed. This class implements methods to send and receive messages, also different `TTransceiver` objects can be connected. If one `TTransceiver` receives a message it will send the message to all connected transceivers. User objects can register event methods for specific messages on each `TTransceiver` object. If a `TTransceiver` object receives a message for which there is event handling code the user code is executed. User objects, as well as the `TTransceiver` objects themselves can generate messages via the `setChanged` method. The idea is that the GUI (view) objects register event handlers to look for modifications on the model objects which generate messages. This method ensures that the model code does not need to know anything about the user interface code. The messages that can be sent consist of objects which are derived of the class `TTransceiverMessage`, this base class includes a `sender` and `destination` field. If the destination is left empty (nil) then a broadcast is done. Another message class is `TDestroyMessage` which is sent when an object is destroyed, the message includes a field `reference` to indicate that all references to `reference` have become invalid.
The class `TTransceiverComponent` acts as a base class for all model objects. The base class includes a `TTransceiver` object and automatically registers an abstract event handler `invalidateReferences` that handles `TDestroyMessage`, the user object should override `invalidateReferences` this method.
A `TTransceiverComponent` which is destroyed will send a `TDestroyMessage`. When the `invalidateReferences` method is called with a `TDestroyMessage`, the user code should ensure that every reference to `TDestroyMessage.reference` is removed, if the conclusion is that the object can no longer sustain itself (for instance if its owner object has been destroyed) it sets the
`TTransceiver.invalidatedBy` property, and the object will be destroyed by its transceiver object.

At a first glance this looks like a nice system, there are however some very subtle problems:

- If after a call to `setChanged(msg)` the new message would be executed immediately, each sent message will execute event handlers in the GUI at once. During sub-procedure calls many messages can be sent, and many needless GUI updates are a result. A better approach is to use the 'Lock/Unlock' design pattern. At the beginning of a procedure we lock the transceiver, and at the end we unlock the transceiver. If, during nested sub-procedure calls, a `setChanged` is executed no update is generated, because the transceiver has been locked. The locking mechanism has been implemented through the use of timers: after all procedure calls are finished the application will have

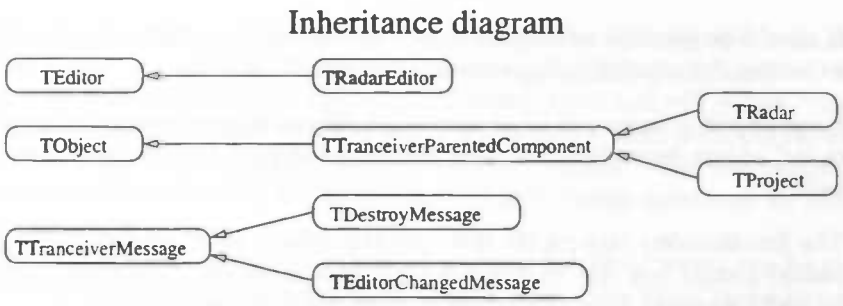


Figure 6.1: *Inheritance diagram, in which some of the different model and view classes are shown.*

time to respond to GUI events and timer events, during the procedure calls all messages are queued in a buffer and identical messages are removed. If a timer in a transceiver object generates its `onTimerEvent` the transceiver will go through its messages list and take actions accordingly.

- `TDestroyMessages` represent a special message class. If they would be handled as normal messages, some very nasty errors are created.
  `TDestroyMessages` have to be executed at once, their execution will not wait on the next timer event. Each `invalidateReference` method will be called immediately. In this method we must be very careful not to use objects which are invalid. If for example the `invalidateReference` method of an editor determines that the editor form needs to be updated, it is better to send a message which indicates this to oneself.

To illustrate the behaviour of the transceiver system, the next example defines a graphical user interface 'GUI', a simulation project 'project1', two radars definition 'radar1' and 'radar2', and a terrain map 'terrain1'. Each of the components has a transceiver 'A', 'B', 'C', 'D' and 'E' which handles messages. Note that the first picture denotes a 'uses' relations, while the second picture contains a run-time connected TTransceiver graph.

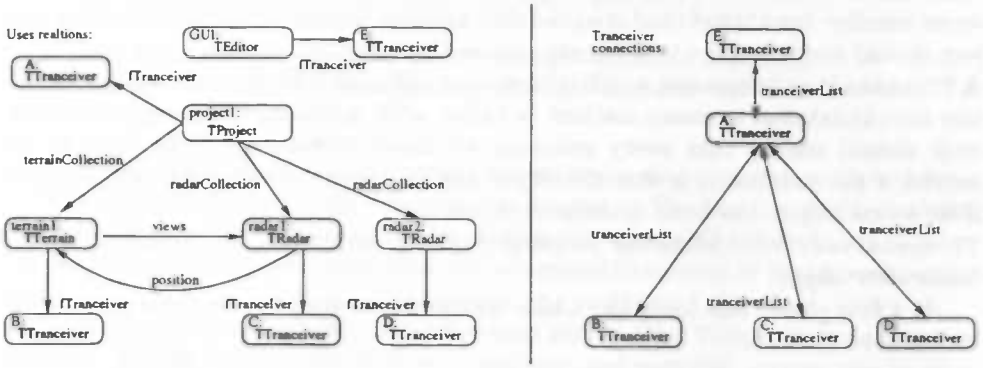


Figure 6.2: *Example: uses relation diagram and transceiver graph*

Now if the component 'radar1' would be deleted, references to 'radar1' have to be deleted from 'project1' and 'terrain1'. Furthermore the GUI will have to be updated. The following sequence diagram shows how this is accomplished.

The described model relieves the programmer of much GUI update code. Further different views of the same model object can opened, without the problem of maintaining synchronization. The chance of memory leaks will also decrease. The

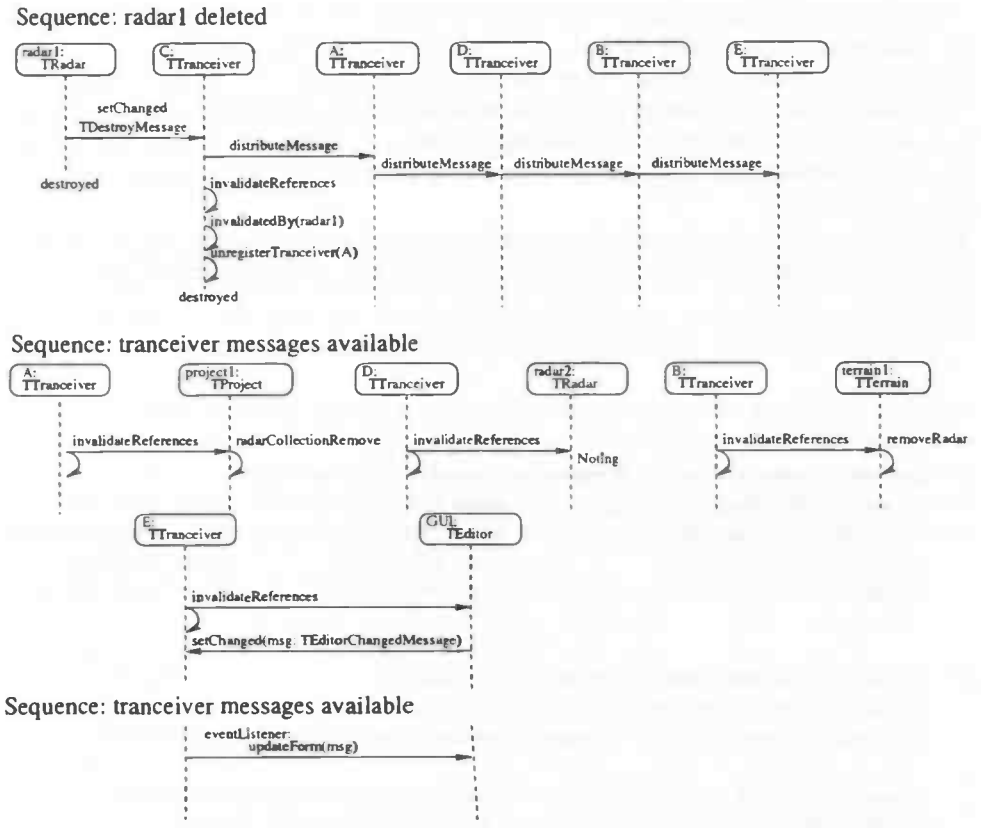


Figure 6.3: *Example: sequence of actions performed by deleting 'radar1'*

next step was to make the GUI context sensitive, a view class with a general interface is needed for this task. The class `TEditor` has been introduced which simplifies editing of model data. This class introduces static abstract functions which are overridden by user-designed editor descendants, these functions allow the editor to be automatically used by the GUI in the appropriate context:

```
TEditor = class (TForm)
  private
  ...
  public
  ...
        class function
          multipeInstancesAllowed: boolean; virtual; abstract;
  class function
    EditorName: string; virtual; abstract;
  class function
    canEdit(c: TTransceiverComponentClass): boolean; virtual; abstract;
  class function
    autoShowFor(cl: TTransceiverComponentClass): boolean; virtual; abstract;
  ...
  property SelectedSubData: TTransceiverComponent
    read getSelectedSubData;
  property EditedData: TTransceiverComponent
    read fEditedData
    write SetEditedData;
end;
```

The class function canEdit determines whether the new editor can be invoked on data which is currently edited. For example, if a TProject 'terrain1' is chosen in a TDataManagerEditor then the EditedData method will return 'terrain1'. The canEdit of editor class TTerrainEditor will result TRUE for a terrain1 class and so opening this editor will become a menu option. The EditedData and SelectedSubData also help the file menu to display which items can be saved via the project menu.
The TTransceiverComponent class also has static functions which can be checked by the GUI to display the correct 'new', 'save', 'open' and 'close' menu items:

```
TTransceiverComponent = class(...)
  private
  ...
  public
  ...
    property transceiver: TTransceiver read Ftransceiver;
    property FileName: string         read FFileName       write SetFileName;
    property FileNameChosen: boolean  read FFileNameChosen write SetFileNameChosen;
    property Path: string             read FPath           write SetPath;
    property Modified: boolean        read FModified       write SetModified;


    procedure saveAs(fname: string); virtual;
    procedure save; virtual;
    procedure insert(child: TTransceiverComponent); virtual; abstract;

    class procedure setDefaultFilePath(s: string); virtual; abstract;
    class function DataTitle: string; virtual; abstract;
    class function canBeChildOf(cl: TTransceiverComponentClass): boolean;
      virtual; abstract;
    class function open(
      aowner: TComponent; fname: string): TTransceiverComponent; virtual;

    class function canLoad: Boolean; virtual; abstract;
    class function canNew:  Boolean; virtual; abstract;
    class function canSave: Boolean; virtual; abstract;

    class function getDefaultFileExt: string; virtual; abstract;
    class function getDefaultFilePath: string; virtual; abstract;
  end;
```

The setDefaultPath function writes the current path for a class to the configuration which is saved when the user exits the program. The function canBeChildOf is checked by the GUI when it fills the 'new' menu item with a list of TTransceiverComponent classes that can be created in context of the current edited data.
By letting each TTransceiverComponent class inherit from TComponent persistence is added. Delphi handles this in a very nice way, and uses this system also to save for example form designs. TComponent classes have methods to write to and read them from a stream. When a component is written its published properties are used to write the data, references to other objects are also stored if the written object is owned by one of written parent components, e.g. :

```
TFigureSpace = class(TComponent)
published
  property figure1: TBox read fBox write setBox;
end;
```

```
TBox = class(TComponent)
published
  property color: TColor          read fColor      write setColor;
  property volume: double         read fVolume     write setVolume;
  property contents: TCollection read fCollection write setCollection;
end;
```

If a TFigureSpace object is saved, its published propertyfigure1 is written as well, at least if the TBox data it contains is owned by the TFigureSpace object. The TBox object will save its colour and volume data, and also the contents of collection contents if contents is owned either by TBox or by TFigureSpace.

It took some time to figure out how this system actually worked, because it was poorly documented and there are some strange problems, but on the whole it will save the programmer a lot of time writing saveTo and readFrom routines. There is also a very handy function objectBinaryToText which allows a textual representation to be made from the objects written to stream, the generated text is almost as easy to read as normal source code:

```
object TProject50: TProject
  RadarList = <>
  TerrainTypeList = <
    item
      TerrainType = TTerrainType51
    end>
  object TTerrainType51: TTerrainType
    color = clBlack
    HasUpperElevationBound = False
    id = 0
    ReflectDiffuse = 1
    ReflectSpecAngle = 0.087
    ReflectSpecular = 1
    title = 'Water'
  end
end
```

6.2 Graphical user interface

The following editors have been added:

TDataManager Displays all the data in memory. This editor acts as a manager from which items can be edited.

TTerrainEditor Displays a particular TTerrain object. It will also show radar positions and target trajectories.

TTerrainTypeEditor Displays the terrain patch types (TTerrainType objects) for a project.

TRadarEditor Allows the definition of radar post (TRadar objects).

TSimulationEditor This editor allows the main simulation settings (start, stop time etc.) to be adjusted for a TSimulation object. Furthermore if TSimRadar objects are present in the simulation these can be edited.

TRadarVisualizationEditor This editor displays the actual received signal for a radar in one as well as two dimensions. Several precalculated data types can be viewed as well.

TFusionDisplay Plots all radar streams within a fusion project in the same window.

TStatisticDistributionEditor Edits parameters of statistical distributions such as `TExpStatDist` (exponential distribution), `TLogNormalStatDistr` (lognormal distribution) etc. The editor can edit any class that is derived from `TStatisticDistribution`.

TTrajectoryEditor This editor allows the definition of way points for a trajectory. The designed trajectory will be drawn onto the current map of the project.

TTargetGeometryEditor Allows you to design the geometry of a target (`TTargetGeometry` object).

TTargetEditor Defines targets (`TTarget` objects) as a combination of trajectories and a target geometry.

TPointScattererEditor Displays the parameters of a point scatterer (`TPointScatterer` object) in a target geometry.

TLineScattererEditor Displays the parameters of a line scatterer (`TLineScatterer` object) in a target geometry.

TFusionProjectEditor This editor allows the main fusion settings (start, stop time etc.) to be adjusted for a `TFusionProject` object. Furthermore, if `TFusionStream` objects are present in the fusion project their properties can be edited too.

TSegmentationEditor Is a test editor for the segmentation algorithm. The user can draw line in a polar coordinate display. The drawn regions can then be segmented.

TKalmanFilterEditor In this editor a test scenario can be designed for the MHT algorithm. Each test scenario consists of a number of echo observations. These observations can be fed through the MHT algorithm after which the user can verify if the observations have been associated to the correct tracks. The management of the list of observations is implemented in the `TMHTTest` object.

TMHTEditor Allows modification of the MHT algorithm (`TMHTracking` object) parameters.

Chapter 7

Conclusion & possible improvements

7.1 Simulation

The simulation part of the project is extensive and already the results are promising. Many components are involved, and it was hard to decide where more detail was required and on the other hand to decide where less detail was needed. As a result some components are a bit overblown, while others need more work. The software has been set up in very modular way, and in many cases it will not be a problem to add detail to the individual components. There are some ideas to use the simulator as a real-time radar generator. As it is now, the simulation cannot be performed in real-time. Unfortunately the software design is not really fit to operate in real-time, because the design has been tailored to generate an accurate image, which costs a lot of computation time.
The simulation could be speed up greatly, however if all unnecessary 'debug' checks are removed. In some cases inefficient programming was required to maintain clear source code. The slowest part of the simulation is the convolution of the radar waveform with the summed RCS values. E.g. when the number of range bins is 1025, the waveform convolution (implemented via FFTs) must operate on a signal of 2048 bins. This results in a significant slowdown. Furthermore the required signal copying between signals of different sizes is a big time spender, even more so when the copied signals are interpolated. Lastly, when targets are added and an antenna pattern with high side lobes is used this requires a wide target clip angle to maintain accuracy. With a large clip angle more targets fall within the clip angle. Consequently these targets are included in the simulation and more computation time will be spent on the targets. In some places the accuracy of the algorithms do not weigh up the computational complexity of the algorithms. In these algorithms there is room left for simplification and consequently speed up.

Component	Possible improvements
Ground clutter	• The current use of precalculation assumes a static positioned radar. If the radar moves the terrain clutter cannot be precalculated. • The reflection model has to be adapted. In many cases the angle of incidence of the radar beam with respect to the terrain slope is very small. The result is that in too many cases only the beginning of the diffuse part of the reflection curve is used. The specular peak of the curve currently affects the radar return too little.
Targets	Each line scatterer in the target geometry is discretely sampled in polar coordinates. The mathematical problems described in section 4.2.2 are easy, but a computational efficient algorithm was very hard to implement. Floating point roundoff errors causing 'division by zero' errors, avoiding the use of trigonometric functions and working with radians (all calculations are $mod 2\pi$) are some of the problems. The current algorithm is efficient, but the source code is too complicated.
Rain clutter	The rain clutter is modelled as a sampled circle area. Again the mathematical problem is easy, but the efficient implementation is too complex. Furthermore because some simplifications had to be made to avoid the use of arctan, sin, cos functions, the shape of the rain clutter gets distorted if the rain clutter approaches the radar station very closely.
Antenna pattern	The antenna pattern is an approximation of the far-field antenna pattern. It is not accurate at very short distances. Furthermore the problem as discussed in section 4.2.5 has to be solved.
Radar waveform	Internal to the simulation complex signals are used. The radar equation also requires a 'complex modulation function' $\mu_T(t)$ in eq. (4.21). It is not completely clear what the shape is of the imaginary part of a pulse waveform.
Receiver	A noise generator in the receiver adds simulated receiver noise. The power of the noise is proportional to the bandwidth of the filter. The filter's bandwidth is about the reciprocal of the pulse length. It is not clear how accurate this bandwidth calculation is.

7.2 Image fusion and Tracking

At a first glance, the fusion/tracking algorithm which is presented in chapter 5 is a central fusion algorithm. All raw radar streams are sent to a central processor. In such a system the tracker has access to all the data. In an inexpensive VTS system however, the raw data streams cannot be delivered to the VTS center in real-time, because the data rates which are supported by the communication lines between radar and VTS center are a bottle-neck. If we assume that the radar stations themselves do some data processing, the system can work. In our system this would mean that the segmentation algorithm is placed embedded at the radar station. Only the positions of the extracted targets (and information on their brightness, size etc.) are sent to the VTS center where the information is processed by the MHT algorithm.

The current MHT algorithm itself is flexible and works fairly well. There are still some problems, as described next.

Component	Possible improvements
Segmentation algorithm	Before the scan lines are fed into the segmentation algorithm, more noise will have to be removed, and as a result less segmented regions are formed. On the one hand this will relieve the data channel to the VTS center and on the other hand, there will be less processing needed in the VTS center itself.
MHT algorithm	• The MHT algorithm still generates too many hypotheses. The number of hypotheses can be lowered by implementing 'clustering' and the 'N observation back' method (both described in section 5.1.4) to the MHT algorithm. • The Kalman filter, which is used to design the tracking gate, must be enhanced. Currently the track is extrapolated linearly. By taking acceleration into account in the prediction model, manoeuvring targets can be tracked much better. Further the maximum turning circle radius of a ship can be used to further restrict the position of the tracking gate. • A penalty should be given to the probabilities of those tracks that are not smooth. This point and the above proposed enhancement will decrease the probability that noise will be tracked. In the current implementation the MHT algorithm could track noise if in each step a noise sample exists in the tracking gate. This track, which consists entirely of noise samples, is usually not smooth or will violate the target's dynamic model (maximum speed, turning circle, etc.). If this happens the track and the hypotheses that contain the track should become less probable.

Appendix A

Symbols

c	The speed of radio waves.
$csc(\theta)$	$csc(\theta) = \frac{1}{\sin(\theta)}$.
E	Energy.
EM	Electro-magnetic.
f_c	Radar carrier frequency.
F	Receiver noise figure.
G	Antenna Gain.
HBW	Horizontal Beam Width.
k	The Boltzmann constant.
LCM	Least Common multiple.
N	The rotational speed of the radar antenna in rpm.
MHT	Multiple hypothesis tracking.
NNA	Nearest Neighbour Association.
P_{FA}	False alarm probability.
P_D	Detection probability.
PRF	Pulse Repetition frequency.
R	Range.
RCS	see σ.
r'	Range rate. The range component of the velocity of a target in polar form.
S	Target strike count: The number of pulses that are used for each sampled antenna position.
T	1. Time between sending and reception of a radar pulse. 2. Temperature in Kelvin.
w	Aperture width.
λ	Wavelength.
$\psi(t)$	Radar signal at RF frequency.
σ	Radar cross-section (RCS).
$\mu(t)$	Radar signal after translating the carrier frequency down to 0.
υ	Doppler coefficient.
τ_{pulse}	Duration of a single radar pulse.

Appendix B

Visit to the Delfzijl VTS installation

The research that I have done will partly be used to develop a complete vessel traffic surveillance (VTS) system. Although there are several companies which already offer highly complex VTS systems, these systems are expensive. Smaller companies like Sodena are developing their own (smaller) systems. A less expensive VTS is an excellent solution for smaller ports, and a good option for ports in developing countries. Although I had read some information about installed VTS'es, I still did not really have an idea of the scales involved and the practical problems which occur in real systems. To get a better impression I contacted Groningen Seaports. Groningen Seaports has a VTS facility to support the harbour 'Eemshaven' and the port of Delfzijl. They kindly agreed upon a tour of the facility. The tour was very helpful for my understanding, but most technical information (and source-code!) was kept confidential, because the company STN-Atlas obviously does not want to help its competitors. In the following report I will summarize the talk that I had with mr. W. Tepper (port VTMS technical expert) and give an impression of the functioning of the system.

The VTS system is situated in the port of Delfzijl. It consists of three radars which are situated along the coastline from the Eemshaven to Delfzijl. The total covered area of the system is about 40 by 50 kilometers. There are also a number of real time controlled video cameras which can be used. In every year the system will have to follow more than 40, 000 ships. The resolution of the system is about 5 meters, which is quite high and thus puts serious limitations on the pulse length of the radar. One of the radar outposts has an areal which is 6 meters in diameter, also a remarkable figure. As we begin at each radar post first the incoming radar signal is digitized, which results in an average data stream of 20 Megabytes per second for each radar. The data is compressed and the resulting stream is sent to the VTS facility. For the two radars in Delfzijl the data is compressed in a 2 megabit stream which is sent through high-speed glass fibre cable. The radar in the Eemshaven however is connected via a telephone line which limits the data transfer speed to about 40 kilobit per second. In the nearby future this connection will be updated to glass fibre. The output of video cameras is digitized as well. The picture from the Eemshaven is only updated every 2 or 3 seconds, while the pictures from the Delfzijl cameras are shown in real-time. When the data arrives at the facility it is further processed to lower the data rate. Blanking is performed to zero out all signals that are coming from land. Next the signals are filtered by means of a threshold to remove noise. In the following step each radar picture is segmented and individual targets are detected. The next step involves the fusion algorithm

which combines the pictures of the three radars.

1. If a segmented object is matched to a known object on a previous picture, tracking information will be available and the position of the object is updated to the current time.

2. After synchronizing all pictures, the objects are matched once more. Now all the objects on the input pictures will have been matched.

3. If an object is detected on all three radars it will probably have different positions according to each radar. A new position is calculated which is a weighted average of the three positions. The radar with the smallest range to the object will get the largest weight.

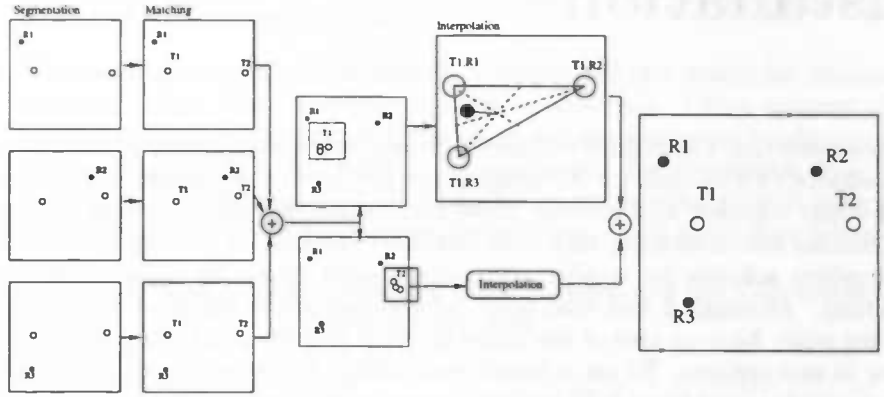


Figure B.1: *Steps in the fusion algorithm: The first column shows the input of three radars, the second column shows how the targets T1, T2 are matched. In the third column the three positions of each target are merged.*

The output of the fusion algorithm is a list of objects each with an identification tag, a position and a video shape. The position and identification tag are stored on disk, so that in the event of e.g. a collision, the actions leading to that collision can be looked up. The detected targets are then fed through a tracking algorithm that assigns or updates the velocity vector for each target. An object has to be detected at least 7 times before it is considered as a target. The tracking algorithm works well as long as the target does not make sudden course changes. A collision detection algorithm can be used for extra safety. The collision detection algorithm is disabled however, because of the number of false alarms which it made. The last step is the generation of three synthetic radar pictures which are shown on the operators desk. The synthetic pictures show additional data like sea-depth and land. Each target is shown with its original radar echo shape and a vector indicating its direction and velocity.

After a technical discussion I was shown the control room where a trained operator had three monitors at his disposal. Each of the monitors showed largely the area of its corresponding radar. The zoom factor is very easily adjusted, for the Eemshaven radar however there is a lag of some 5 seconds before changes take effect. As I observed one of the monitors, it displayed a number of ships, Although most targets were displayed correctly, there were also some anomalies. After a closer inspection it was possible to explain most of these errors:

Two targets belonging to the same object: It could happen that because of the shape of a vessel a target is broken and detected as two separate targets.

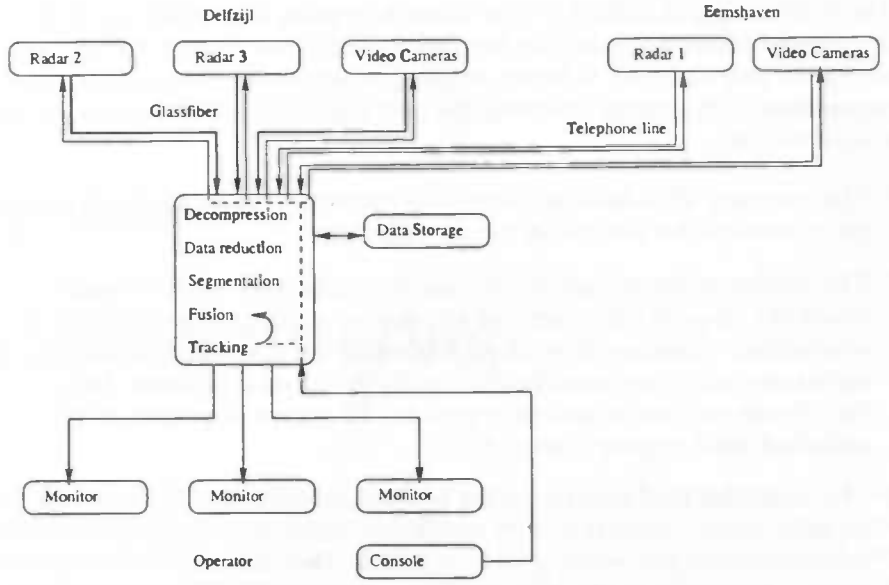


Figure B.2: *Dataflow model of the Groningen Seaport VTS system.*

If a ship is oriented properly towards the radar it will only consist of a large bridge and bow which both reflect echoes well. The rest of the ship however will not reflect echoes well and thus the ship is shown as two targets: bow and bridge. The tracking algorithm will track two targets, only after a few more detections it will be clear that the targets belong to one larger target, and at this point the targets are merged by the algorithm. The merging can always be done manually.

Tracking problems: The tracking algorithm prefers the ships to move in straight lines. The algorithm is insensitive to sudden changes and this will prevent velocity or bearing errors because of noise. If a ship makes a fast turn the tracking algorithm will not be able to follow the target, and the track is lost. Somewhat later the target is again detected but it is classified as a new target instead of reassigned to the original tag. This is done to avoid the possibility of wrong reassignment. The old track which has lost its target is just updated according to the last known velocity and bearing until it moves off the screen or until the operator intervenes.

False targets: There are some objects which will show up on the radar screen but which are not ships or buoys. Mostly this is land clutter which is almost but not completely removed by blanking. E.g. I could see some poles sticking out of the water. Now and then they were classified as targets and were given a track, but after a few more detections it became obvious that they were not ships and the track was removed again. Also a pier was shown in very elongated way, which is due to backscattering of the radar echoes. The Eemshaven radar showed the result of sea clutter. The source of sea clutter is the reflection of the radar wave by the sea waves. Fortunately sea clutter only appears near the radar (at high grazing angles).

Although I did not actually had the chance to see the effect of rain, I was told that it was a serious problem. During rain the tracking algorithm can easily be overwhelmed by false targets. This is prevented by setting the threshold higher, however this will also make it harder to detect smaller vessels.

The VTS system in Delfzijl is now about four years old. Although there were some technical problems during the first years (which had to do with salty sea-air corroding the chip sockets!) it is now working satisfactorily. Although there are a few competitor VTS systems available, the port chose this system because of a few important factors:

1. The company STN-Atlas is situated in Bremen. If there are large problems the system can be serviced in two to three hours.

2. The display of the system is only partly synthesized. Other systems try to detect the shape of the target and will display an image of a ship in its correct orientation. However, it is often impossible to detect the shape of a ship accurately and poor classification results in a wrong synthetic image. E.g. mr. Tepper told me he had once seen a VTS system displaying a ship which undocked while moving sideward!

3. The maximum hardware resolution is 5 meters and this is also the maximum zooming range. Thus it is only possible to zoom out. Other systems allow software zooming-in which goes further than their maximum hardware resolution. The result is that while the operator gets the impression that he is looking very carefully at his high resolution image, he is actually looking at a software interpolated low resolution image which could lack information.

Finally I will summarize the most important properties of the VTS system:

Power	$P = 25$ kW (of one radar)
Pulse length	$\tau = +/- 50$ ns
Resolution range	$\Delta r = 5$ m
Aperture length	$l = 6$ m (for one antenna)
Gain	$G = 35$ dB (for the 6 m antenna)
Wavelength	$\lambda = 9370$ Mhz
Area	$a = 200$ m^2
Rotation speed	$N = 12 - 20$ rpm
Trackable targets	2500

Appendix C

Kalman filtering

In 1960, R.E. Kalman published his famous paper describing a recursive solution to the discrete-data linear filtering problem. Since that time, due in large part to advances in digital computing, the Kalman filter has been the subject of extensive research and application, particularly in the area of autonomous or assisted navigation. The Kalman filter is a set of mathematical equations that provides an efficient computational (recursive) solution of the least-squares method. The filter is very powerful in several aspects: it supports estimations of past, present, and even future states, and it can do so even when the precise nature of the modelled system is unknown. The purpose of this appendix is to provide an introduction to the discrete Kalman filter. The information presented here has been summed up from [22].

The definition of a Kalman filter starts by defining the variables which we want to predict (or track). These values define a state vector X_n at time n, e.g. the state vector could contain position and velocity. In reality we cannot know the state exactly because the measurements we take are always contaminated with noise. Each measurement is defined by a measurement vector Y_n, which does not have to contain the same variables as the state vector. E.g. we could have measurements in polar coordinates but predict in Cartesian coordinates. Now we define an estimate of the state as X_n^*. The algorithm itself of course will always work with an estimate X_n^* of the real state X_n. Furthermore the notation of X_n^* must reflect how the estimate was obtained. For this purpose an extra subscript is added to the notation: $X_{n,m}^*$ is the estimate of X_n based on measurements at m and before. The Kalman filter will combine a measurement Y_n and an estimated prediction of the state $X_{n,n-1}^*$ into a new prediction $X_{n,n}^*$ in such a way that that the new estimate $X_{n,n}^*$ has a minimum variance, that is the highest accuracy. The behaviour of the state vector X_n is modelled by the target dynamics. This target dynamics model is represented by the matrix Φ. Because in reality our model of the target is inaccurate we will have to add a noise vector called the 'dynamic model driving noise' U_n:

$$X_{n+1} = \Phi X_n + U_n \tag{C.1}$$

The measurement Y_n is related to the actual state X_n with an added observation error N_n. Because X_n and Y_n do not have necessarily have one to one correspondence an observation matrix M is needed to map the state X_n to the measurement:

$$Y_n = MX_n + N_n \tag{C.2}$$

Now we can write the **prediction equation** of the Kalman filter:

$$X_{n+1,n}^* = \Phi X_{n,n}^* \tag{C.3}$$

The measurement Y_n is incorporated in the estimate $X^*_{n,n-1}$ through the **Kalman filtering equation**:

$$X^*_{n,n} = X^*_{n,n-1} + H_n(Y_n - MX^*_{n,n-1}) \tag{C.4}$$

The matrix H_n contains the tracking filter constants and is given by the **weight equation**:

$$H_n = S^*_{n,n-1}M^T\left[R_n + MS^*_{n,n-1}M^T\right]^{-1} \tag{C.5}$$

The matrix $S^*_{n,n-1}$ is an estimate of the accuracy in predicting the target's position at time n based on measurements of time $n-1$ and before. $S^*_{n,n-1}$ is given by the **predictor equation**:

$$S^*_{n,n-1} = \Phi S^*_{n-1,n-1}\Phi^T + Q_n \tag{C.6}$$

Q_n is the dynamic model noise covariance $COV(U_n)$ and R_n in equation C.5 is the observation noise covariance $COV(N_n$.

$$Q_n = COV(U_n) \tag{C.7}$$

$$R_n = COV(N_n) \tag{C.8}$$

Lastly the estimated accuracy is corrected by a measurement through **the corrector equation**:

$$S^*_{n,n} = [I - H_nM]\,S^*_{n,n-1} \tag{C.9}$$

The Kalman filter can then be used as follows:

1. **Initialization:** Q_0, R_0, S_0 and X_0 must be given.
2. **Prediction:** Use the **prediction equation** and the **predictor equation** to predict position and to predict accuracy to the time of the new measurement.
3. **Correction:** Use the **weight equation** to calculate H_n and then the **Kalman filtering equation** to update the estimate $X^*_{n,n}$. The accuracy estimate $S^*_{n,n}$ is updated through **the corrector equation**.
4. **Iterate:** Go to step 2.

Appendix D

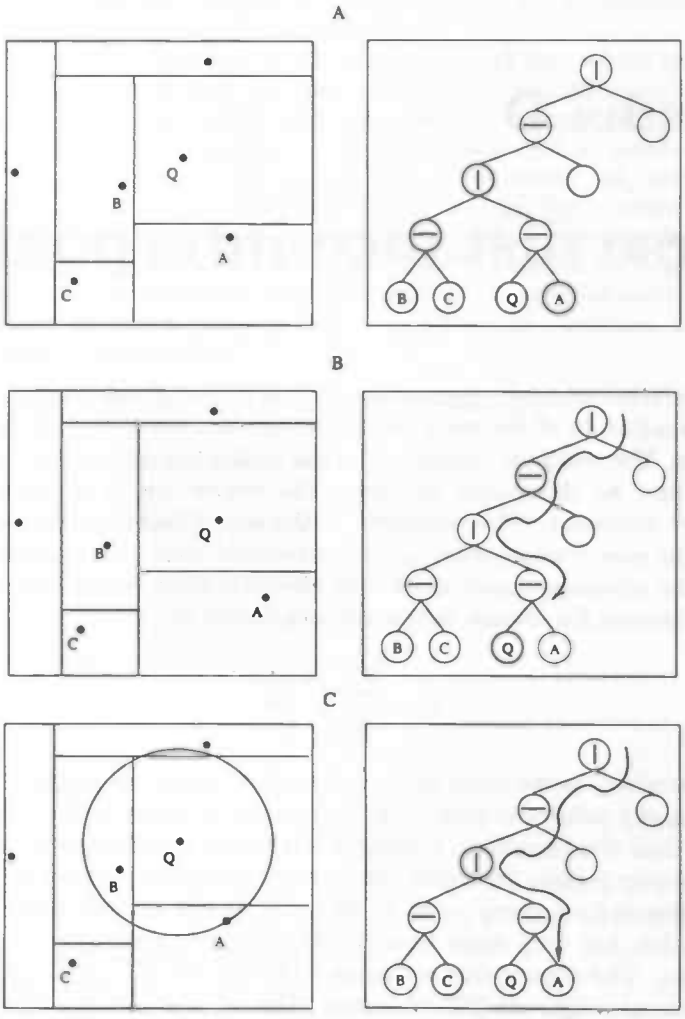
Inverse distance interpolation

In this interpolation method, observational points are weighted during interpolation such that the influence of one point relative to another declines with distance from the new point. The weight is calculated by the taking the reciprocal of the distance to some power. As this power increases, the nearest point to the query point becomes more dominant. The simplicity of the underlying principle, the speed in calculation, the ease of programming, and reasonable results for many types of data are some of the advantages associated with inverse distance weighted interpolation. The general formula for inverse distance interpolation is:

$$Q = \frac{\sum_{i=1}^{N} V_i \frac{1}{d_i^p}}{\sum_{i=1}^{N} \frac{1}{d_i^p}} \tag{D.1}$$

In this formula Q is the value of the interpolated query point, d_i is the distance between the query point and point i, V_i is the value of point i, N is the number of points taken into consideration. Finally, p is a factor which adjusts the weighting strength of nearby points. To obtain the interpolated value we have to know the N nearest neighbours for a query point q. We could go through the whole data set for each query point, but with more then 10, 000 points to search this is will take far too much time. The observation set needs to be placed in a data structure which allows fast nearest neighbour (NN) searches. The kd-tree data structure (see [5] for a definition), with some extensions, is very useful for this task.
In [5] it is explained how kd-trees can be built and used to do a points-within-rectangular-window search. The data set is copied in two lists. The first list is ordered by X-coordinate, the second by Y-coordinate. The kd-tree is then built by dividing the lists in two at each level, by either X or Y median. This ensures that the tree is filled evenly. With about 100,000 points in the observation set it is not feasible to do a complete kd-tree composition as it would contain about 2^{18-1} leafs and thus $2^{18-1} - 1 \approx 200,000$ internal nodes. To circumvent this, bins (lists at the tree leafs) are constructed when the current set of not yet inserted points contains less than 200 points. Now the tree structure will have $2^{18-1-7} - 1 \approx 1000$ nodes. There is another optimization possible which uses an associated data structure at the leafs of the tree. This search structure is especially fast when used with small data sets of about 200 points, which is exactly what we have.
The NN search algorithm consists of a tree search algorithm and a search algorithm on the associated data structures at the bins. The tree algorithm determines at each node recursively at which side of the splitting line its nearest neighbours are most likely to be found and it then traverses this tree branch. If nearest neighbours are found, a global value D_{max} for the maximum distance of the found nearest neighbours is updated. When a recursive call returns, the algorithm then decides

on the basis of the value D_{max} whether the subtree of the other unexplored branch could still contain nearest neighbours. The next picture D.1 shows how this works.



The optimization with the associated data structure at the bins works by using the triangle inequality formula which is described in [1]. Each bin is sorted by distance to a statically chosen point, which never changes. Then the distance between the query point Q and this point P_{ref} is calculated. Now we find the index i of a point P_i with minimum distance to P_{ref} via a binary search. The index of this point is also stored within the data structure. From the index i start iterating to the left until some stop criteria holds. The stop criteria and the sorting of the bin ensures that no NN lie to the left. The same principle is used to iterate to the right of i. If the global variable for the maximum NN distance is initially guessed then the search is speeded up a even more. During the search the total number of points that are inspected have been count, the next table show some results for a random point set:

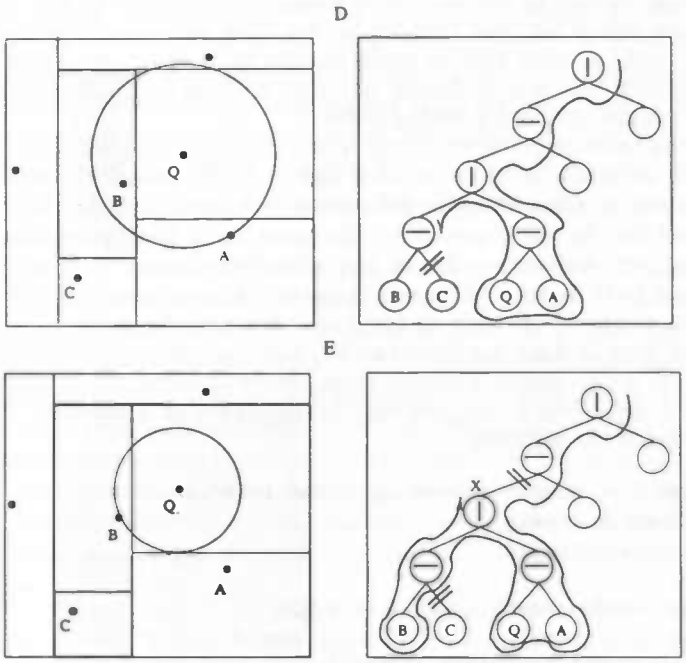


Figure D.1: *Nearest neighbour search in a kd-tree: (A): The data set consists of the points A, B, C and the query point Q. (B): The parent of the query point is searched, the nearest neighbour search starts here. (C): Point A is the first neighbour found. A bounding circle is created. All future NN candidates must lie within the bounding circle. (D): The NN search backtracks. The leaf with point C is not searched because the complete segment of point C does not overlap with the bounding circle. Point B's segment is searched because it does overlap. (E): Point B is a better nearest neighbour candidate then point A, thus it becomes the NN candidate and the radius of the bounding circle is updated. The search is finished because the shaded segment represented by node X completely overlaps the bounding circle.*

# inserted points	Number of NN	bin size	NN guess	Points inspected
100, 000	1	200	no	≈ 50
100, 000	1	200	yes	≈ 50
100, 000	10	200	no	≈ 80
100, 000	10	200	yes	≈ 77
10, 000	1	1	no	≈ 20
10, 000	1	1	yes	≈ 20
10, 000	10	1	no	≈ 65
10, 000	10	1	yes	≈ 55

Tree construction

The object `T2dTree` implements the tree structure and has methods for inserting, deleting, searching etc. Although general kd-tree algorithms are described in [5] there were some very nasty details which were very hard to implement. Most of the problems were encountered during tree construction. During construction of the tree, the x and y median of the data set is used split the data set in two. To find the median of a data set two lists are constructed, one contains all the data points sorted with respect to their x coordinates, the other list contains the same points, but sorted with respect to their y coordinates. Now it is easy to find the x or y median: if there are n points, just look in the appropriate list at index

$\lfloor \frac{n}{2} \rfloor$. If splitting occurs to the x-median then it is thus easy to find the x-median in x-coordinate sorted list and this list is then split in two. Unfortunately the y-coordinate sorted list must also be updated and split, but now with respect to the x-median, which is not easy to find in this list, and the list is certainly not easy to split in two, maintaining the y-sorted order.
The solution is to store pointers in the x and y sorted lists which contain a data set point and an extra boolean variable `left`. If the two lists contain the same pointers (but due to the sorting in different order) we can set `left` for some point in the x-sorted list. In the y-sorted list the same point (but at another index) will then have its `left` also true. So we can split the x-sorted list, and for the left part set all the `left` to true. Afterwards iterate through the y-sorted list and add the points which have `left` true to the `leftY` list and the rest to the `rightY` list. When the splitting is done we have two left lists, one sorted by x coordinate, the other sorted by y coordinate and two right lists which are also sorted by x and y coordinate. In these lists it is again easy to find the x or y median. The following code accomplishes the splitting:

```
ALGORITHM: split a dataset according to the x-median
  sortX:=TPointList.create;
  sortY:=TPointList.create;

  for n:=0 to DataSetPoints.count-1 do begin
    new(pxy);
    pxy^.point := DataSetPoints.items[n];
    pxy^.left := false;
    sortX.add(pxy);
    sortY.add(pxy);
  end;

  sortedListX.SortX;
  sortedListY.SortY;

  // example: splitting to the x-coordinate

  sortedListX.split(medianX, leftX, rightX);

  for n:=0 to leftX.count-1 do begin
    pxy:=leftX.items[n];
    pxy^.left:=true;
  end;

  for n:=0 to rightX.count-1 do begin
    pxy:=rightX.items[n];
    pxy^.left:=false;
  end;

  // now its easy to split the y-sorted list
  leftY:=dataSetPointList.create;
  rightY:=dataSetPointList.create;

  for n:=0 to sortY.count-1 do begin
    pxy:=sortY.items[n];
    if pxy^.left then
      leftY.add(curOther)
    else
      rightY.add(curOther);
  end;
```

The current implementation of the kd-tree loads the complete data set in mem-

ory and each leaf of the tree consists of a list of data points. In the case of very large data sets an enhancement could be to only load leafs in memory if they are needed during a search. Furthermore the insert and delete routines are not very efficient. A list of data points is constructed from the tree, in this list a point is inserted or a point is removed from the list, whereafter the tree is reconstructed. In this way the insert and delete routines keep the tree balanced, efficient insert and delete algorithms for kd-trees (which keep the tree balanced) are yet another research project!
Although the nearest neighbour interpolation scheme works correctly, its still not very fast when the interpolation is used to draw the map on the screen. Also the interpolation generates artifacts in some situations. There are two parameters that affect these artifacts. First the `minControlPoints` parameter which states that at least `minControlPoints` number of elevation points are used to calculate the elevation of the interpolated point. Secondly, only elevation points within `maxSampleRadius` parameter radius are used to calculate the height value of the interpolated point. `minControlPoints` is used to be sure that at least some values are used for interpolation. Decreasing `maxSampleRadius` speeds up the NN search because it prunes the search tree (see figure D.1). Artifacts will appear in the following cases:

- If a point A with a large height value lies just outside the `maxSampleRadius` radius of a query point Q, it will not be incorporated into the interpolation. If the points in `maxSampleRadius` radius of Q all have small height value there will be a discontinuity in the interpolation: We can choose the position of a query point P such that A falls just in the `maxSampleRadius` radius, or we can move P a little so that A is not included. Because the height value A is large A will have a significant effect on the interpolated value.

- If the given data points are distributed poorly, it could happen that a query point Q will have only few neighbours in its `maxSampleRadius` radius. The `minControlPoints` parameter is used in this case to add more points.

On the whole it can be concluded that the complexity of nearest neighbour interpolation is too large to justify its use over an interpolation using Delaunay triangulation.

Bibliography

[1] M. Greenspan, G. Godin, J. Talbot, *Acceleration of Binning Nearest Neighbour Methods*, Visual Information Technology Group, University of Toronto, Institute for Information Technology and MD Robotics Ltd., 1997.

[2] S.S. Lim, D. F. Liang, M. Blanchette, *Air Defence Radar Surveillance System Tracking Assessment*, Defence Research Establishment, Ottawa.

[3] S. S. Blackman, R. J. Dempster, T. S. Nichols, *Application of Multiple Hypothesis Tracking to Multie-Radar Air Defence Systems*, Hughes Aircraft Company, El Segundo, USA.

[4] B.L. Lewis, F.F. Kertschmer, Jr., Wesley W. Shelton, *Aspects of Radar Signal Processing*, The Artech House Radar Library, 1986.

[5] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, *Computational Geometry, Algorithms and Applications*, Springer, 1997.

[6] D. Hearn, M. P. Baker, *Computational Graphics*, second edition, Prentice Hall, 1997.

[7] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns, Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.

[8] C. Aulbert, *Effizient Absorbierende Randbedingungen in Numerischen Bodenradarsimulationen mittels Generalized Perfectly Matched Layers*, TU Braunschweig, institut für Geophysic und Meteorology, 1986.

[9] M. A. Richards, *Fundamentals of Radar Signal Processing*, School of Electrical and Computer Engineering, Georgia Institute of Technology, 2000.

[10] J. Hu, *Generating Surfaces In Environmental GIS Applications*, User Conference Proceedings, Environmental Systems Research Institute, Inc., 1995.

[11] D.L. Hall, *Mathematical Techniques in Multisensor Data Fusion*, Artech House Inc., 1992.

[12] J. Gil, D. Keren, *New Approach to the Arc Length Parametrization Problem*, The Technion - Israel institute of technology, Ilaifa Universiy, 1997.

[13] S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, W. H. Press, *Numerical Recipes in C: The Art of Scientific Computing*, Camebridge University press, 1997.

[14] M.H. Carpenter, *Principles of Modern Radar Systems*, Artech House inc., London, 1988.

[15] A.G. Bole, W.O. Dineley, *Radar and ARPA Manual*, Butterworth-Heinemann Ltd., 1990.

[16] J.C. Toomay, *Radar Principles for the Non-specialist*, Van Nostran Reinhold, New York, 1989.

[17] R. L. Mitchell, *Radar Signal Simulation*, Artech House inc., 1976.

[18] Dr. E. F. Carter Jr., *Random Number Generation*, `http://www.taygeta.com/random/`, Taygeta Scientific Inc.

[19] G. Brooker, *Sensors*, Australian Centre for Field Robotics, the University of Sydney, 2001.

[20] T. Brown, R. Aitmehdi, *Shore Based Radar Antennas*, Easat Antennas Ltd., 1992.

[21] P. V. Coates, *The Fusion of an IR Search and Track With Existing Sensors To Provide a Tracking System for Low Observability Targets*, Pilkington Thorn Optronics, Hayes, England.

[22] E. Brookner, *Tracking and Kalman Filtering Made Easy*, John Whily & sons, inc., 1998.