Advanced subsurface imaging by space-time deconvolution for array-based landmine detection GPR

N.T. van Tol
TU Delft, Faculty of EEMCS

June 16, 2008

Master of Science Thesis Report
IRCTR - A - 016 - 008

Contents

Chapter 1

Introduction

This chapter will start with a small introduction into the landmine problem. After that the array-based GPR of IRCTR is briefly described which is followed by some necessary processing steps needed for all GPR data. The need for imaging algorithms will also be made clear in this section. The approach of my thesis work is outlined in the final section.

1.1 Landmine problem

The UN states that around 80-110 million landmines are laid throughout the world. Landmines can remain active for decades, so they continue to be a threat even after the conflict has long been resolved. Since mines can not distinguish between soldier and civilians, they cause a high number of civilian casualties. Annually thousands of people are killed or maimed by landmines. Not only casualties are a negative aspect of landmines, the infected ground can not be used for agricultural of infrastructural use. This will inevitable inhibit the economic growth potential as well. For these reasons it's important that the mine fields are cleared. The UN states that a mine field is considered safe when 99.6% of the mines are cleared [1].

Roughly two types of landmines can be distinguished; anti-personnel mine and anti-tank mine. The names are quite self-explanatory. The first is aimed to incapacitate personnel, it has a diameter smaller than 15 cm and will explode at the pressure of around 10-15 kilos. The second is used to take out vehicles, it's diameter is larger than 25 cm and needs a pressure higher than 100 kilos to detonate. Obviously the anti-personnel mines are more difficult to detect.

Currently the most effective detection techniques involve dogs and carefully interrogating the soil by using a metal prod. Various detection techniques currently being developed are detection by acoustics, nuclear quadrupole resonance and ground penetrating radar. The first methods sends sounds waves into the ground and looks at the reflection, while the second detects the nitrogen isotope N-14 which is abundant in high-explosives. This thesis deals with landmine detection using GPR.

1.2 Array-based GPR for landmine detection

The data used has been obtained with an ultra-wideband array-based time-domain radar sensor [2]. This radar array is vehicle mounted and it's important to note that it works in the near-field when imaging burried landmines. Other key characteristics of this configuration will also be discussed in this section.

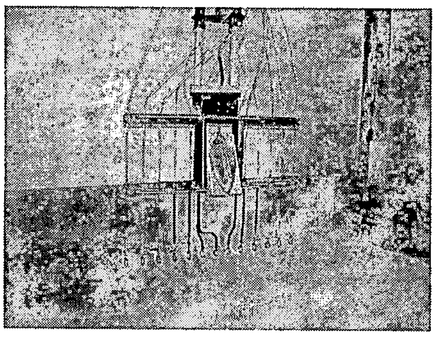


Figure 1.1: Radar array

Figure 1.1 shows the used array consisting of one transmitting antenna which is a dielectric wedge antenna with a 84 cm diameter footprint (at -10 dB level) and an array of 13 receiving loops. The basic element chosen for the receiving array is the shielded loop antenna. The receiving array is located 27 cm under the transmitter to reduce the direct coupling of the antennas. The pulse generator fires a double-exponential pulse with a pulse duration of 41 ps (half-amplitude level) and a magnitude of 40 V. This pulse-type has been chosen to have a large spectral density at frequencies below 1 GHz, which are not radiated effectively by the transmitting antenna due to the relatively small size of its physical aperture. The resulting pulse has a bandwidth from 0.5 GHz to 3.3 GHz (-3 dB level). The receiver chain consists of a switch, a seven-channel signal conditioner and an eight-channel sampling converter, which allows for simultaneous receiving of the scattered field. The entire chain has an analog bandwidth from 200 MHz to 6 GHz. The sampling rate of the converter is 525 kHz, which is more than sufficient to apply averaging to reduce the noise floor. The overall system bandwidth is 3.56 GHz and starts at 240 MHz.

The total length of the aperture is 84 cm, the 13 loops are placed at 7 cm intervals. Having a receiving array instead of one antenna removes the necesity of scanning in two dimensions. In this case an area with a width of

84 cm can be scanned one-dimensionally.

1.3 Processing of GPR data

All data acquired by the array needs processing to compensate for some unwanted influences. It is described here because the steps that need to be taken are the same regardless of what imaging algorithm is used. First the data is pre-processed to remove among others the ground reflection and direct coupling. This is further described in the first subsection. In the next subsection the need for an imaging algorithm is discussed and there the goal of this thesis work will be made explicit. The final part of this section is used to discuss the post-processing of the imaged result. For each imaging technique and result the same method is used which makes the performance comparison of the techniques possible.

1.3.1 Pre-processing

Before imaging several steps are taken to pre-process the data acquired by the array. This includes low-pass filtering (cut-off frequency 6 GHz) to suppress uncorrelated noise, alignment of the direct coupling in every B-scan to compensate for time drift, and background subtraction (by moving average subraction) to remove direct coupling and ground surface reflection [2].

The antenna footprint introduces a non-uniform amplitude distribution to received signals over the loops, consequently this results in weaker reflections when objects are located at the side of the array. This is compensated by scaling all the B-scans with linear weights as introduced in [3]. The weights are determined by comparing the antenna coupling of all seperate B-scans. The antenna coupling of each loop is measured to quantify the non-uniformity of the footprint, and to provide weighting factors accordingly. For every B-scan the peak of the antenna coupling is taken and that peak relative to the maximum peak determines the factor the B-scan needs to be compensated with. Figures 1.2 and 1.3 [3] show the normalised peak antenna coupling and the corresponding weights, being the inverse of the normalised peak.

The final step is time-gating the raw data. This basically removes all the information up till a certain point in time and this very effectively removes all the direct coupling and ground reflection. When the ground reflection is also removed then information from surface-layed targets is also discarded. Normally that would be inacceptable, but the entire landmine system comprises of multiple sensors with one especially focused at detection of surface-layed and / or shallowly buried landmines. The GPR's task is to interrogate deeper parts of the soil and therefore removing information from surface-layed objects is not objectionable. These steps are taken

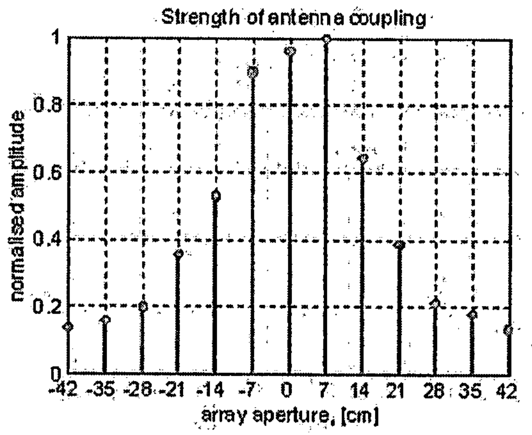


Figure 1.2: Normalised peak antenna coupling for each B-scan

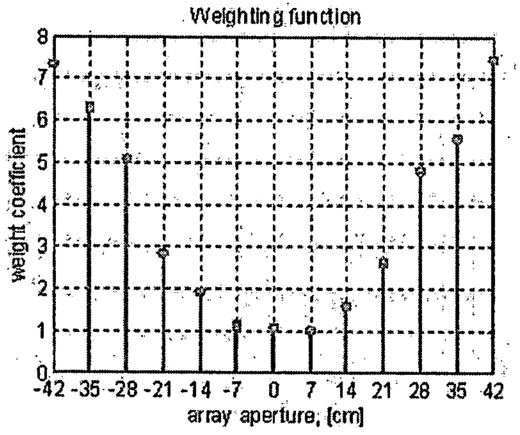


Figure 1.3: Weighting factors for each B-scan

sequentially and Figure 1.4 shows this process in a flowchart. It should be noted that it is absolutely vital that the time-drift compensation is done before time-gating is applied.

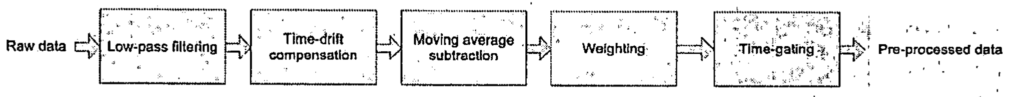


Figure 1.4: Pre-processing flowchart

The result of these pre-processing steps is shown in Figures 1.5 and 1.6.

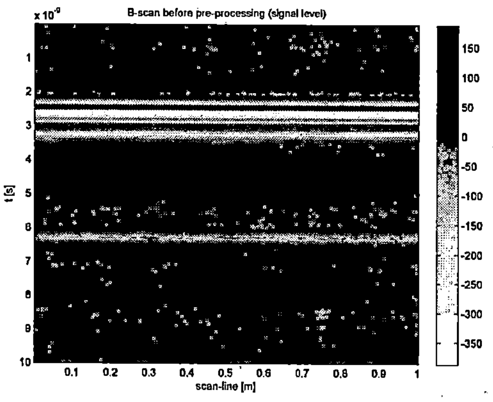


Figure 1.5: Raw B-scan (magnitude in [mV])

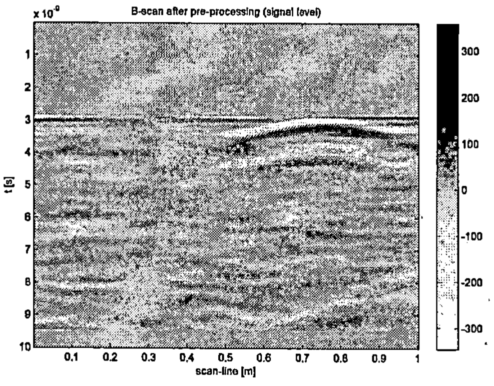


Figure 1.6: Pre-processed B-scan (magnitude in [mV])

1.3.2 Migration

Migration is the process of focussing target reflections in the recored data back into their true position and physical shape [4]. Figure 1.6 shows the pre-processed reflection of a sub-surface target. First the array is placed at 0 cm and there the received signal is recorded, this results in one A-scan for every loop. A series of A-scans aquired over a scan-line forms the shown B-scan. Due to the finite beamwidth of the transmitting and receiving antenna the reflection of the point scatterer is perceived as a hyperbola. Figure 1.7 schematically shows what happens.

The array is shown at two positions on the scan-line, a point-scatterer (denoted in red) is present in the center and the resulting hyperbola can be seen. From the first position the transmitter fires the pulse and because of the finite beamwidth of the antennas the reflection of the small sphere

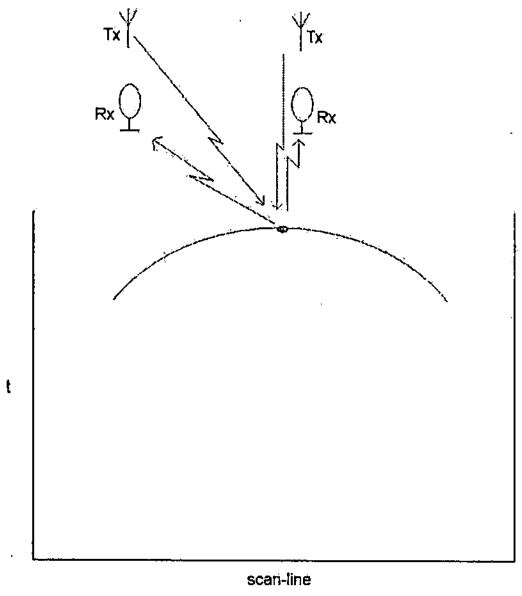


Figure 1.7: Hyperbola created by point scatterer

is already noticeable in the A-scan. The same is done when the array is placed directly above the target. Since the distance from antenna to object is larger at the first position the reflection occurs later in time. As can be seen in Figure 1.7 this results in the hyperbola.

This distribution of energy over a B- or C-scan does not provide good localisation and resolution of buried targets and therefore needs to be reversed. The process of focussing this energy back to the object's actual position is called migration.

1.3.3 2D confidence map of 3D data

Two main ways of mine detection exist in 3D GPR data, the first being slice by slice detection and the second being by constructing a confidence map. The former is more suited when the data is being examined by a human operator, the latter is more suited for automated mine detection and since the final goal is to have an automated system our results will be presented in that manner. A confidence map is created by means of energy projection, which is a method that projects the energy of some kind of feature of a 3D migrated dataset onto a 2D surface. Three main projection techniques are available that will be discussed in this subsection [5].

Energy projection

This method determines the total energy for every point with fixed coordinates (x,y) by integrating the signal power over the time-axes by

$$EP_{ij} = \sum_{z=1}^{M} C_{z,ij}^{2} \tag{1.1}$$

EP_{ij} - Confidence value at (i,j)
$C_{z,ij}$ - Migrated A-scan

This method is easy to implement but the result can have substantial artefacts.

Windowed energy projection

Windowed energy projection is a more advanced version of Energy projection. It uses the knowledge that migrated images of mines are quite short (3 to 6 cm) and that no objects of interest were placed directly underneath of each other in the test scenarios [5]. Taking this shortness into account when calculating the energy of the A-scan we can deduce that only the small part, in which the mine is present, contributes to the mine image, while all the rest contributes to clutter. This leads to replacing Equation 1.1 to:

$$EW_{ij} = max[\sum_{z=k-L}^{k+L} C_{ij}^{2}(z)] \tag{1.2}$$

EW_{ij} - Confidence value at (i,j)
$C_{ij}(z)$ - Migrated A-scan

Equation 1.2 represents the maximum energy contained in a window with width L of each A-scan of the migrated C-san. The window length should be large enough to be able to contain the entire length of the mine, but as small as possible to suppress the levels of the mine-free scans [5].

This method is slightly more computationally intensive but reduces the clutter levels heavily in comparison to windowless Energy projection.

Alternating sign windowed energy projection

Alternating sign windowed energy projection (ASWEP) uses even more a-priori knowledge of the images of mines in migrated C-scans. Not only mine height tends to remain unchanged, a mine image seems to consist of a negative and positive bulb [5]. This makes it possible to further elaborate 1.2 to:

$$EA_{ij} = max[\sum_{z=k-L}^{k+L} C_{ij}^2(z) * Q(z)]^2 \qquad (1.3)$$

The ASWEP given by Equation 1.3 uses the mask $Q(z)$ to discriminate between negative and positive parts and maximise the expected output for mines. The following triple valued mask:

$$Q(z) = \begin{cases} -1, z \in \left\{Z_i^N\right\}_i \\ 1, z \in \left\{Z_j^P\right\}_j \\ 0, z \notin \left(\left\{Z_j^P\right\}_j \bigcup z \in \left\{Z_i^N\right\}_i\right) \end{cases} \qquad (1.4)$$

Z^N - Collection of negative depth portion
Z^P - Collection of positive depth portion

$Q(z)$ is the mask that's representative for the changing signs of mine images. This technique offers the best result but with a little more computational effort as compared to Windowed energy projection.

1.4 Goal of thesis

The field of my thesis is the migration of GPR data. Many methods that exist now are adapted versions of seismic migration methods that use geometric approaches or back-projection. These methods only take some radar characteristics and only the relative electrical permittiviy ϵ_r of the soil into account and therefore a new method of migration by deconvolution has been proposed in [6]. Migration by deconvolution does use system characteristics and also soil characteristics (f.e. loss-tangent and ground impulse response) and by doing so can offer an improvement of the results of the migration. A second drawback of the currently available methods is that they require considerable computational effort and as such they are mostly used in offline processing. These drawbacks determine the goal of this thesis work: to implement a fast migration technique by extending the deconvolution approach to the case of the array-based GPR of IRCTR.

1.5 Thesis approach

The research started by familiarising with the problems and challenges in landmine-detection by GPR. This entails the studying of the GPR array used by IRCTR and the migration problem in particular. The results of this preliminary study are discussed in chapter 1. Before developing the new algorithm I implemented a commonly used algorithm (diffraction stack) for the GPR array. The goal of this was two-fold: First a good reference was needed to compare the results of the to be developed new migration algorithm, second the implementing of this algorithm would give me more insight and experience in the field of migration in general. The implemenation and results of the diffraction stack are discussed in chapter 2. When this was completed the work was started on developing the method proposed in [6] for our GPR array. First a 2D version of the algorithm was developed, which performance was compared to that of the diffraction stack algorithm. After that the 3D version was developed and it soon became clear that additional processing steps were needed. The road to the final developed algorithm and it's performance are discussed in chapter 3. Chapter 4 deals with more GPR data to validate the functionality of the algorithm and to validate approximations that were made. The final chapter summarises the thesis work and draws conclusions after which some recommendations for further research are given.

Chapter 2

Diffraction stack algorithm

This migration algorithm is widely used in seismic and GPR processing and the reason for describing is two-fold. For my research a good reference method was needed to compare the results of the new imaging algorithm, but developing an implementation for the GPR array would also benefit my personal understanding of the migration problem. Diffraction stack is a time-inversion technique is based on the travel times of the signal from the transmitter to the object and back again to the receiving array. At first a grid is created which will be the focused volume, then for each grid point the travel time from transmitter to receiving loop is calculated. From the B-scan the corresponding value is taken and this is added to the value of that grid point. This process is repeated for each grid point for each individual loop. The algorithm can be described by Equation 2.1 [2]. All data used in this chapter is pre-processed and weighted according to the steps described in the previous chapter.

$$s(x_l, y_m, z_n) = \sum_{k=1}^{L}\sum_{j=1}^{M} s_k(t_{kj}, x_l, y_j, z_n) \tag{2.1}$$

t_{kj} - travel time to (x_l, y_j, z_n) for k-th loop
$s_k(t_{kj}, x_l, y_l, z_n)$ - Pre-processed weighted signal amplitude

This chapter is divided in three main parts. The first being free space data imaged and the second being sub-surface data, followed by the conclusions discussing the performance of the algorithm.

2.1 Focusing of free space data

The focusing of free space data is easier because the travel time calculations are very straightforward. To illustrate the performance of the algorithm we used a dataset acquired with the array-based GPR for a surface-laid sphere in IRCTR. The sphere had a 2 cm diameter and was located directly under the transmitter at a distance of 42 cm from the transmitting antenna and at the middle of the scan-line (30 cm). This measurement set-up is shown in Figure 2.1. The original received data for one loop is shown in Figure 2.2, with Figure 2.3 showing the focused image. The result has dimensions [126 x 61] and needed 52 seconds to focus.

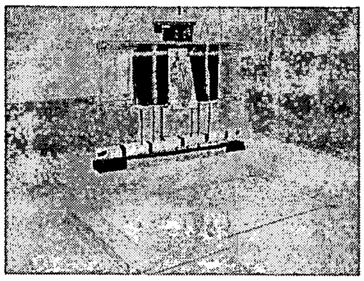


Figure 2.1: Measurement set-up of free space scenario

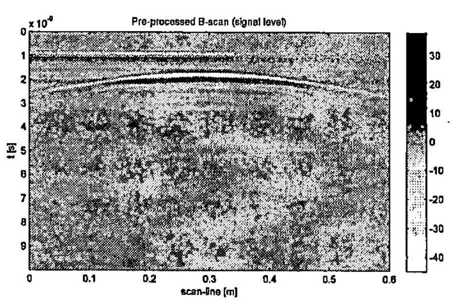


Figure 2.2: Raw B-scan

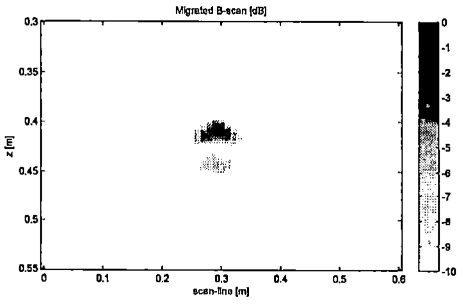


Figure 2.3: Focused B-scan

In Figure 2.3 the sphere is clearly visible.

2.2 Focusing of subsurface data

The ability of focusing subsurface data is more relevant to detecting landmines. As an example, we used a dataset acquired by IRCTR with two mine-like objects in the ground which was again pre-processed and weighted. The transmitting antenna was located 52.5 cm above the ground and the ϵ_r of the soil was 3.03. Because of the different dielectric properties of free space and soil the travel time calculation is less straightforward. The shortest distance (from the wave's point of view) is not a straight line from target to antenna, but is through a refraction point which is located on the border of the two dielectrics (the ground). This is further illustrated in Figure 2.4. The location of this refraction point is needed for calculating the travel times. Two ways are discussed to calculate these travel times; both depending on the (chosen) location of the refraction point.

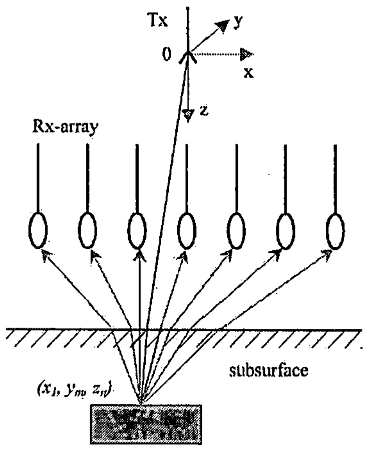


Figure 2.4: Imaging geometry

2.2.1 Travel time calculation with approximated refraction point

The first method approximates the location of the refraction point by using a point which has the same x and y coordinates as the imaged point (directly above the target). Figure 2.5 illustrates this more clearly [7].

$$Traveltime_{total} = Traveltime_{freespace} + Traveltime_{subsurface} \qquad (2.2)$$

$$Traveltime_{fs} = \frac{\sqrt{(xt-xr)^2+(yt-yr)^2+(zr)^2}+\sqrt{(xa-xr)^2+(ya-yr)^2+(za-zr)^2}}{C_0} \qquad (2.3)$$

$$Traveltime_{ss} = \frac{2|zo - zr|\sqrt{\epsilon}}{C_0} \tag{2.4}$$

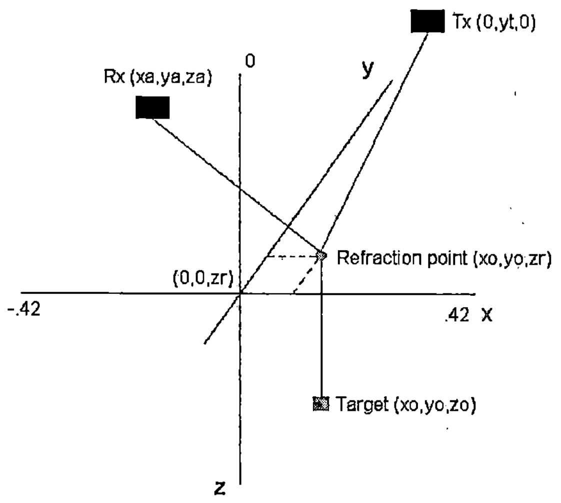


Figure 2.5: Approximate refraction point

The method has the advantage of being simple. The lower complexity comes at a cost of accuracy. Both the ASWEP result and a slice are shown in Figure 2.6.

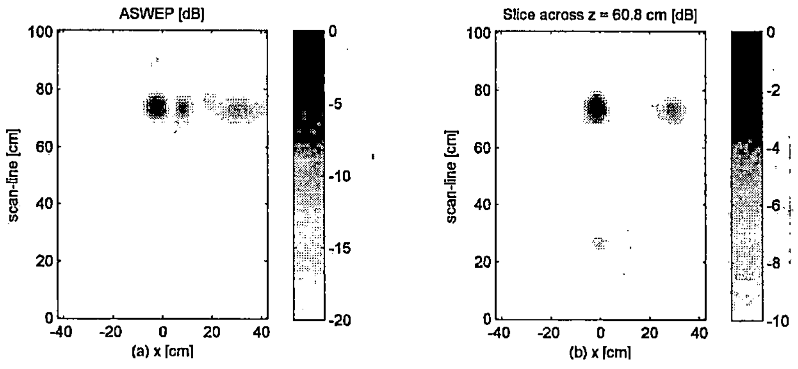


Figure 2.6: Result of approximation (a) ASWEP, (b) Slice

Both results are normalised, but have different dynamic ranges. The 10 dB range that's used for the slice result is acceptable and increasing it would only add clutter. The ASWEP uses more information, because it projects the entire 3D dataset on a 2D surface. Therefore it's logical that a higher dynamic range can be expected and this is a clear asset of the

ASWEP. Although this image shows both targets it's clear that the side target, even after weighting, is considerably weaker. Furthermore a very large artefact is present between the two targets. This approach is useful when conditions involve dry soil and shallow targets, because these both conditions minimise the error of the approximation of the refraction point. Last it should be noted that this algorithm (even being an approximation) is very computationally intensive. The algorithm needed 6 hours to migrate a matrix with dimensions [85 x 101 x 27].

2.2.2 Travel time calculation with exact refraction point

It's also possible to find the exact location of the refraction point. This requires more computational power, but will give more accurate results. The travelpath is illustrated in Figure 2.7 [7].

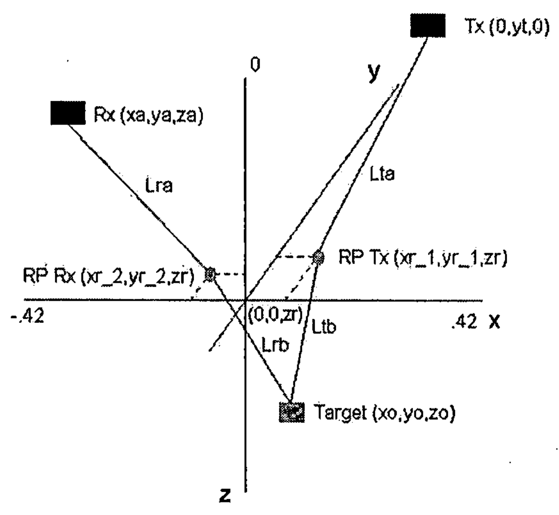


Figure 2.7: Exact refraction point

The refraction point lies on the path of the signal with the shortest travel time from transmitter to target. For the total travel time the path from target to the receiver needs to be calculated as well. This path has another refraction point so the process needs to be repeated. This will not be treated here, because it's basically the same process.
First an equation for the travel time (from transmitter to target) is needed.

$$L_{ta} = \sqrt{(xt - xr_1)^2 + (yt - yr_1)^2 + (zr)^2} \tag{2.5}$$

$$L_{tb} = \sqrt{(xo - xr_1)^2 + (yo - yr_1)^2 + (zo - zr)^2} \tag{2.6}$$

$$Traveltime = \frac{Lta}{C_0} + \frac{Ltb\sqrt{\epsilon}}{C_0} \tag{2.7}$$

The objective is to find the refraction point which will generate the shortest travel time. Equation 2.7 must have only one variable, namely the x or y coordinate of the refraction point. The missing coordinate can be calculated from the other. The find the minimum travel time the following equation is solved

$$\frac{dTraveltime}{dxa} = 0 \tag{2.8}$$

The coordinate-system as given in Figure 2.7 concurs with the coordinate system that was used. Our coordinate-system has its' origin at the transmitting antenna and the beginning of the scan-line. So the aperture starts at x = -42 cm untill x = 42 cm, z starts at the antenna (0) and increases as we go deeper.

Solving Equation 2.8

Equation 2.8 is a fourth order polynomial and several polynomials are needed to calculate all the exact travel times in a given grid. This is due to the fact that when antenna and object are in the same plane the refraction point calculation differs, which results in three scenarios: Antenna and object at same x, antenna and object at same y, antenna and object in different planes. As said earlier the polynomials between Tx and target are also different from the polynomials between the receiving loops and the target. Therefore to calculate all refraction points accurately six scenarios (and therefore six fourth order polynomials are needed). First the polynomials between Tx and target will be discussed, followed by the other three. As with any fourth order polynomial four solutions will be produced and only one is the refraction point coordinate that is needed. The logic used to select the correct solution from the set of four will be discussed after considering the polynomials.

- **Polynomial from Tx to target, scenario 1**
  Antenna and object are in the same x plane. This scenario is illustrated in Figure 2.8.

  Here the X and Z coordinate of refraction point are known, we can

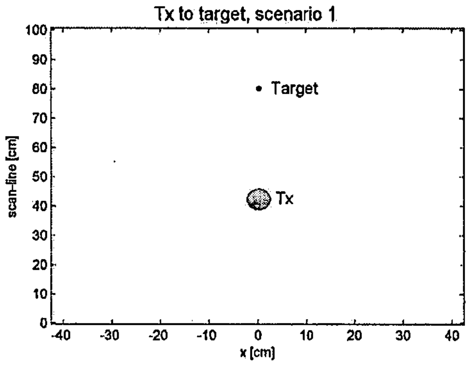


Figure 2.8: Tx and object in same x plane

find the y coordinate by solving

$$\begin{aligned}
&A \cdot yr^4 + B \cdot yr^3 + C \cdot yr^2 + D \cdot yr + E = 0 \\
&A = \epsilon - 1 \\
&B = 2yo + 2yt - 2\epsilon \cdot yt - 2\epsilon \cdot yo \\
&C = -zo^2 + 2zr \cdot zo - zr^2 - yo^2 - 4yt \cdot yo - yt^2 + \epsilon \cdot (zr^2 + yt^2) + 4\epsilon \cdot yo \cdot yt + \epsilon \cdot yo^2 \\
&D = 2(zo^2 - 2zr \cdot zo + zr^2 + yo^2)yt + 2yt^2 \cdot yo - 2\epsilon \cdot yo(zr^2 + yt^2) - 2\epsilon \cdot yo^2 \cdot yt \\
&E = \epsilon \cdot yo^2(zr^2 + yt^2) - (zo^2 - 2zr \cdot zo + zr^2 + yo^2)yt^2
\end{aligned} \tag{2.9}$$

where:
yo, zo y-,z-coordinates of the location of the grid-point
yt y-coordinate of the transmitting antenna
yr, zr y-,z-coordinates of the refraction point
ϵ permittivity in the ground

- **Polynomial from Tx to target, scenario 2**
  Antenna and object are in the same y plane. This scenario is illustrated in Figure 2.9.

  Y and Z coordinate of refraction point are known, we can find the x coordinate by solving

$$\begin{aligned}
&A \cdot xr^4 + B \cdot xr^3 + C \cdot xr^2 + D \cdot xr + E = 0 \\
&A = \epsilon - 1 \\
&B = 2xo + 2xt - 2\epsilon \cdot xt - 2\epsilon \cdot xo \\
&C = -zo^2 + 2zr \cdot zo - zr^2 - xo^2 - 4xt \cdot xo - xt^2 + \epsilon(zr^2 + xt^2) + 4\epsilon \cdot xo \cdot xt + \epsilon \cdot xo^2 \\
&D = 2(zo^2 - 2zr \cdot zo + zr^2 + xo^2)xt + 2xt^2 \cdot xo - 2\epsilon \cdot xo(zr^2 + xt^2) - 2\epsilon \cdot xo^2 \cdot xt \\
&E = \epsilon \cdot xo^2(zr^2 + xt^2) - (zo^2 - 2zr \cdot zo + zr^2 + xo^2)xt^2
\end{aligned} \tag{2.10}$$

  where:
  xo, zo x-,z-coordinates of the location of the grid-point

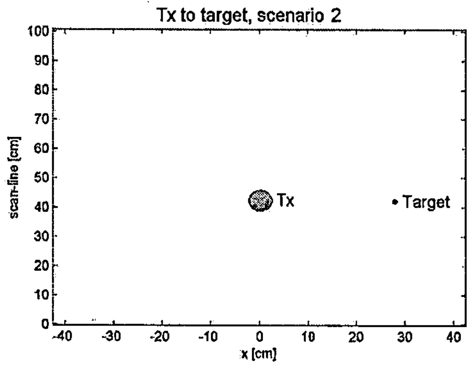


Figure 2.9: Tx and object in same y plane

xt x-coordinate of the transmitting antenna (zero in our set-up)
xr, zr x-,z-coordinates of the refraction point
ϵ permittivity in the ground

- **Polynomial from Tx to target, scenario 3**
  Antenna and object are in different x and y planes. This scenario is illustrated in Figure 2.10.

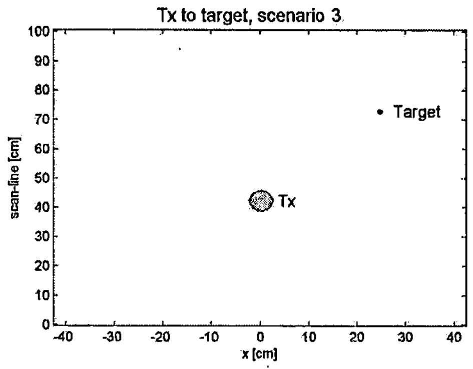


Figure 2.10: Tx and object in different x and y planes

Only the Z coordinate of the refraction point is known, but the relation between the Y and X coordinate can be derived.

$$
\begin{aligned}
& A\cdot xr^4 + B\cdot xr^3 + C\cdot xr^2 + D\cdot xr + E = 0 \\
& yr = -(yt-yo)\cdot xr/xo + yt \\
& A = (1+(yt-yo)^2/xo^2)^3 - (1+(yt-yo)^2/xo^2)^3\epsilon \\
& B = -2(1+(yt-yo)^2/xo^2)^2\epsilon(-xo-(yt-yo)^2/xo) \\
& \cdots + (1+(yt-yo)^2/xo^2)^2(-2xo-2(yt-yo)^2/xo) \\
& C = -zr^2\epsilon(1+(yt-yo)^2/xo^2)^2 - (1+(yt-yo)^2/xo^2)\epsilon(-xo-(yt-yo)^2/xo)^2 \\
& \cdots + (1+(yt-yo)^2/xo^2)^2((zr-zo)^2+xo^2+(yt-yo)^2) \\
& D = -2zr^2\cdot\epsilon(-xo-(yt-yo)^2/xo)(1+(yt-yo)^2/xo^2) \\
& E = -zr^2\epsilon(-xo-(yt-yo)^2/xo)^2
\end{aligned}
\tag{2.11}
$$

where:
xo, yo, zo x-,y-,z-coordinates of the location of the grid-point
xt, yt x-,y-coordinates of the transmitting antenna
xr, yr, zr x-,y-,z-coordinates of the refraction point
ϵ permittivity in the ground

- **Polynomial from Rx to target, scenario 1**
  The scenario, antenna and object are in the same x plane, is illustrated in Figure 2.11.

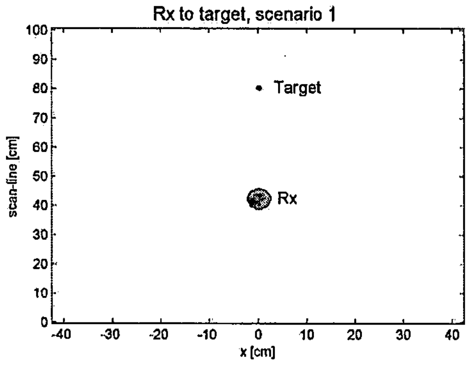


Figure 2.11: Tx and object in same x plane

The X and Z coordinate of refraction point are known, we can find the y coordinate by solving

$$
\begin{aligned}
& A\cdot yr^4 + B\cdot yr^3 + C\cdot yr^2 + D\cdot yr + E = 0 \\
& A = 1-\epsilon \\
& B = 2\epsilon\cdot yo + 2\epsilon\cdot ya - 2ya - 2yo \\
& C = -\epsilon\cdot yo^2 - 4\epsilon\cdot yo\cdot ya - \epsilon((za)^2+ya^2) + ya^2 + 4ya\cdot yo + (zo-zr)^2 + yo^2 \\
& D = 2\epsilon\cdot yo^2 ya + 2\epsilon\cdot yo((za)^2+ya^2) - 2ya^2\cdot yo - 2ya((zo-zr)^2+yo^2) \\
& E = ya^2\cdot((zo-zr)^2+yo^2) - \epsilon\cdot yo^2((za)^2+ya^2)
\end{aligned}
\tag{2.12}
$$

where:
yo, zo y-,z-coordinates of the location of the grid-point
ya, za y,z-coordinates of the receiving loop
yr, zr y-,z-coordinates of the refraction point
ϵ permittivity in the ground

- **Polynomial from Rx to target, scenario 2**
  Antenna and object are in the same y plane. This scenario is illustrated in Figure 2.12.

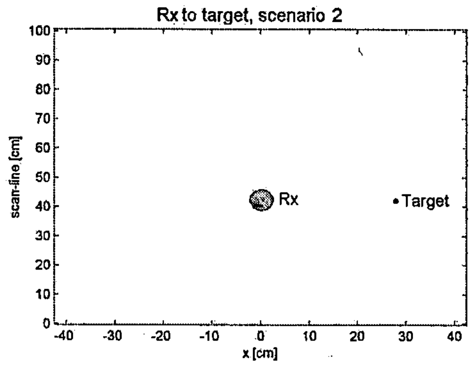


Figure 2.12: Tx and object in same y plane

The Y and Z coordinates of refraction point are known, we can find the x coordinate by solving

$$\begin{aligned}
&A \cdot xr^4 + B \cdot xr^3 + C \cdot xr^2 + D \cdot xr + E = 0\\
&A = 1 - \epsilon\\
&B = 2\epsilon \cdot xo + 2\epsilon \cdot xa - 2xa - 2xo\\
&C = -\epsilon \cdot xo^2 - 4\epsilon \cdot xo \cdot xa - \epsilon((za)^2 + xa^2) + xa^2 + 4xa \cdot xo + (zo - zr)^2 + xo^2\\
&D = 2\epsilon \cdot xo^2 xa + 2\epsilon \cdot xo((za)^2 + xa^2) - 2xa^2 \cdot xo - 2xa((zo - zr)^2 + xo^2)\\
&E = xa^2((zo - zr)^2 + xo^2) - \epsilon \cdot xo^2((za)^2 + xa^2)
\end{aligned} \tag{2.13}$$

where:
xo, zo x-,z-coordinates of the location of the grid-point
xa, za x-,z-coordinates of the receiving loop
xr, zr x-,z-coordinates of the refraction point
ϵ permittivity in the ground

- **Polynomial from Rx to target, scenario 3**
  Antenna and object in different x and y planes. This scenario is illustrated in Figure 2.13.

  Only Z coordinate of the refraction point is known, but the relation between the Y and X coordinate can be derived.

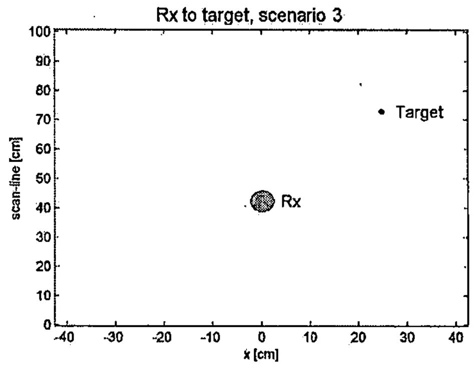


Figure 2.13: Tx and object in different x and y planes

$$
\begin{array}{l}
A \cdot xr^4 + B \cdot xr^3 + C \cdot xr^2 + D \cdot xr + E = 0 \\
yr = (ya - yo)(xr_R x - xa)/(xa - xo) + ya \\
A = (1 + (ya - yo)^2/(xa - xo)^2)^3 - (1 + (ya - yo)^2/(xa - xo)^2)^3 \epsilon \\
B = -(-2xa - 2(ya - yo)^2 \cdot xa/(xa - xo)^2)\epsilon(1 + (ya - yo)^2/(xa - xo)^2)^2 \\
\cdots - 2(1 + (ya - yo)^2/(xa - xo)^2)^2 \cdot \epsilon \\
\cdots * (-xo + (-(ya - yo)xa/(xa - xo) + ya - yo)(ya - yo)/(xa - xo)) \\
\cdots + 2(-xa - (ya - yo)^2 xa/(xa - xo)^2)(1 + (ya - yo)^2/(xa - xo)^2)^2 \\
\cdots + (1 + (ya - yo)^2/(xa - xo)^2)^2 \\
\cdots * (-2xo + 2(-(ya - yo)xa/(xa - xo) + ya - yo)(ya - yo)/(xa - xo)) \\
C = -(xa^2 + za^2 + (ya - yo)^2 \cdot xa^2/(xa - xo)^2)\epsilon(1 + (ya - yo)^2/(xa - xo)^2)^2 \\
\cdots - 2(-2xa - 2(ya - yo)^2 \cdot xa/(xa - xo)^2) \\
\cdots * \epsilon(-xo + (-(ya - yo)xa/(xa - xo) + ya - yo)(ya - yo)/(xa - xo)) \\
\cdots * (1 + (ya - yo)^2/(xa - xo)^2) \\
\cdots - (1 + (ya - yo)^2/(xa - xo)^2)\epsilon(-xo + (-(ya - yo)xa/(xa - xo) + ya - yo) \\
\cdots * (ya - yo)/(xa - xo))^2 \\
\cdots + (-xa - (ya - yo)^2 \cdot xa/(xa - xo)^2)^2(1 + (ya - yo)^2/(xa - xo)^2) \\
\cdots + 2(-xa - (ya - yo)^2 \cdot xa/(xa - xo)^2) \\
\cdots * (1 + (ya - yo)^2/(xa - xo)^2) \\
\cdots * (-2xo + 2(-(ya - yo)xa/(xa - xo) + ya - yo)(ya - yo)/(xa - xo)) + \\
\cdots * (1 + (ya - yo)^2/(xa - xo)^2)^2 \cdot (xo^2 + (zr - zo)^2 \\
\cdots + (-(ya - yo)xa/(xa - xo) + ya - yo)^2) \\
D = -2(xa^2 + za^2 + (ya - yo)^2 \cdot xa^2/(xa - xo)^2) \\
\cdots \epsilon(-xo + (-(ya - yo)xa/(xa - xo) + ya - yo)(ya - yo)/(xa - xo)) \\
\cdots * (1 + (ya - yo)^2/(xa - xo)^2) - \\
\cdots * (-2xa - 2(ya - yo)^2 \cdot xa/(xa - xo)^2) \\
\cdots \epsilon(-xo + (-(ya - yo)xa/(xa - xo) + ya - yo)(ya - yo)/(xa - xo))^2 \\
\cdots + (-xa - (ya - yo)^2 \cdot xa/(xa - xo)^2)^2 \\
\cdots * (-2xo + 2(-(ya - yo)xa/(xa - xo) + ya - yo)(ya - yo)/(xa - xo)) \\
\cdots + 2(-xa - (ya - yo)^2 \cdot xa/(xa - xo)^2) \\
\cdots * (1 + (ya - yo)^2/(xa - xo)^2)(xo^2 + (zr - zo)^2 \\
\cdots + (-(ya - yo)xa/(xa - xo) + ya - yo)^2) \\
E = (-xa - (ya - yo)^2 \cdot xa/(xa - xo)^2)^2 \\
\cdots * (xo^2 + (zr - zo)^2 + (-(ya - yo)xa/(xa - xo) + ya - yo)^2) \\
\cdots - (xa^2 + za^2 + (ya - yo)^2 \cdot xa^2/(xa - xo)^2) \\
\cdots \epsilon(-xo + (-(ya - yo)xa/(xa - xo) + ya - yo)(ya - yo)/(xa - xo))^2
\end{array}
\tag{2.14}
$$

where:
xo, yo, zo x-,y-,z-coordinates of the location of the grid-point
xa, ya, za x-,y-,z-coordinates of the transmitting antenna
xr, yr, zr x-,y-,z-coordinates of the refraction point
ϵ permittivity in the ground

- **Selecting the correct root**
  The polynomial is solved by computing the eigenvalues of the companion matrix. This results in four roots of which one concurs with the coordinate of the refraction point. Two imaginairy solutions can be discarded immediately since our solution is a real point in our coordinate system. The choice between the last two solutions is made based on geometry. We know that the refraction point has to be between the grid-point of the focused volume and the antenna; this will only be the case for one root.

Now the coordinates of the refraction point are known and this information can be introduced to Equation 2.7 to calculate the exact travel time. With the travel time calculated the implementation of the algorithm is straightforward (and as such identical to the free space case). The results in Figure 2.14 show that the exact method shows the two targets clearly, but as with the approximate method (as shown in Figure 2.6) the side target is weaker and a large artefact is present. The increased complexity is noticeable in the computation time; the same matrix [85 x 101 x 27] needed 21 to 22 hours to focus.

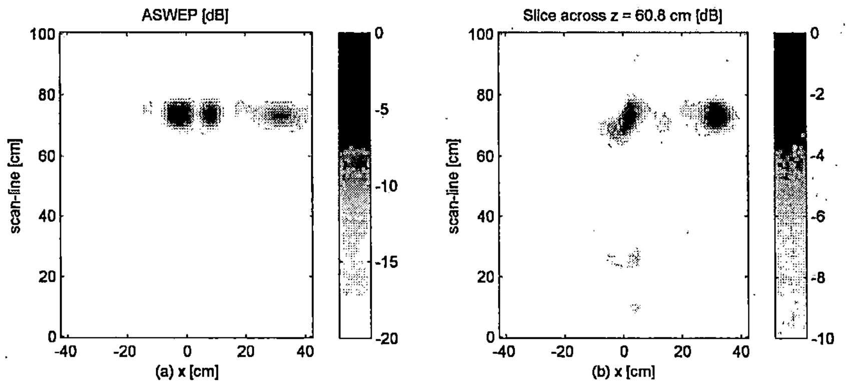


Figure 2.14: Result of exact method (a) ASWEP, (b) Slice

2.3 Conclusions

We described a commonly used focusing technique based on geometrical optics, namely the diffraction stack algorithm. This technique has been extended to the array-based GPR. For the sub-surface case two methods of travel time calculation were implemented, the first being an approximation and the second being the analytically obtained solution. For the geometrical optics paths equations were derived for finding the exact location of refraction points in two-layer media. These equations gave slightly better results than the approximation at the expense of a three- to fourfold increase in computational time. For both implementations it's clear that the diffraction stack method can only be used in off-line applications since the focussing of an area of interest of 1 m^2 takes at least six hours.

Chapter 3

Advanced radar imaging by parametric space-time deconvolution algorithm

The previously discussed algorithm only uses the data as input for the migration. In migration by deconvolution the system characteristics (of radar and ground) are also taken into account and this is expected to give better results. The basic concept of the algorithm will be discussed first and there the importance of the time-domain model of a reflection by a point-scatterer will be clarified. In sections 3.2 and 3.3 a 2D implementation of the algorithm is discussed and the performance is compared to that of the diffraction stack algorithm. The implementation of the 3D version needed some additional steps which are described in section 3.4. The results of the 3D-implementation are compared to the diffraction stack method in section 3.5 after which this chapter is ended with the conclusions.

3.1 Basic concept

The key assumption of this algorithm is that the aquired signal from any given object can be approximated by combining reflections from a collection of point-scatterers; basically through modeling each object by a set of small independent isotropic point-scatterers [4]. If this operation is assumed linear their combining is a convolution in space which leads to the received signal being described by

$$c(x, y, t) = w(x, y, z_0, t) \otimes_{x,y,t} \Lambda_{zo}(x, y, t) \tag{3.1}$$

where:
$\Lambda_{zo}(x, y, t)$ collection of point-scatterers
$c(x, y, t)$ recorded data

$\otimes_{x,y,t}$ convolution operator
$w(x, y, z_o, t)$ reflection of an individual isotropic point-scatterer

In Equation 3.1 it is shown that the acquired data is represented as a convolution of a collection of point-scatterers and the reflection of one of those scatterers. Equation 3.1 introduces two terms that will play a vital role in understanding and implementing the algorithm: $w(x, y, z_o, t)$ being the point-spread function and $\Lambda_{zo}(x, y, t)$ being the target scattering matrix.

The target scattering matrix is the desired result in this problem because that is the collection of point-scatterers giving rise to the acquired signal. This matrix contains the objects located in the interrogated section and therefore is the migrated dataset. In the following subsections the method of determining the reflection of an individual point-scatterer is discussed and after that the filtering of the point-spread function out of the acquired data is treated.

3.1.1 Time-domain model of reflection by point scatterer

The time-domain model of the reflection of an isotropic point scatterer is obtained by considering the system as a cascade of linear responses. A time-domain model is intuitively more fitting for the used time-domain radar, it has the advantage that no assumptions on frequency dependent terms have to be made [4]. This cascade is illustrated in Figure 3.1 where each block denotes a linear response. A pulse is fired and after propagating through air it arrives at the air-ground interface. It then propagates through the ground and reflects at the target point scatterer, the wave returns through the ground-air interface to finally arrive at the receiving antenna.

The combination of these linear responses will result in a convolution in the time-domain

$$b(\vec{r_a}, t) = \frac{T_{a-g}T_{g-a}}{8\pi^2 R_t R_r c} g_d(t) \otimes h_{N,Tx}(\vec{a_i}, t) \otimes \Lambda_0(t) \otimes h_{N,Rx}(-\vec{a_s}, t) \otimes \frac{dV_s(t-t_d)}{dt} \tag{3.2}$$

where:
$V_s(t)$ excitation voltage applied to the transmitting antenna
$h_{N,Tx}(\vec{a_i}, t)$ normalised impulse response (IR) of the transmitting antenna
$h_{N,Rx}(-\vec{a_s}, t)$ normalised IR of the receiving antenna
$g_d(t)$ IR of the ground
$\Lambda_0(t)$ IR of the target
R_t total path length from the transmitting antenna to the target
R_r total path length from the target to the receiving antenna
T_{g-a},T_{a-g} transmission coefficient at the ground-air/air-ground interface
t_d exact time-delay (of the wave between the transmitter and receiver)

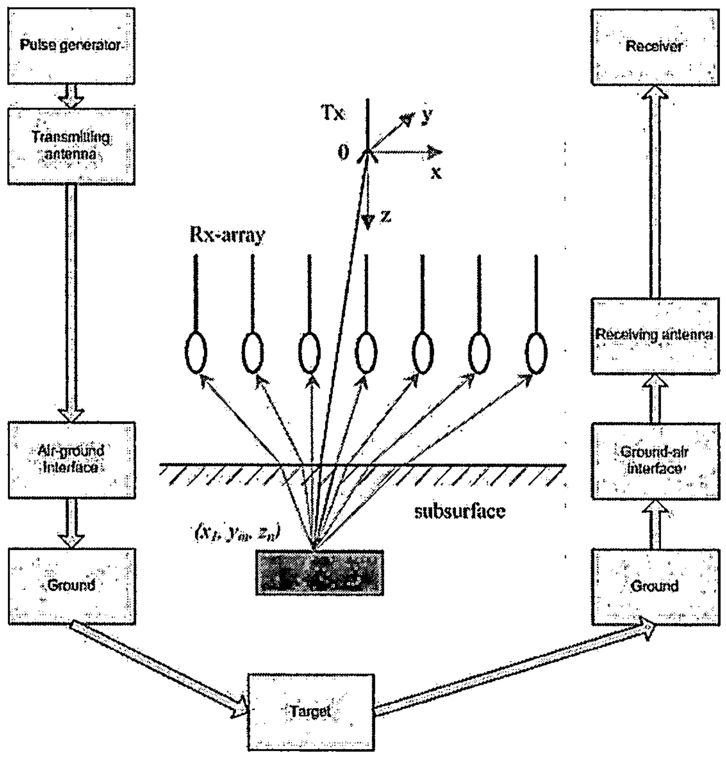


Figure 3.1: Cascade of linear responses

All the blocks in Figure 3.1 are accounted for in Equation 3.2. $V_s(t)$ is the excitation pulse, $h_{N,Tx}(\vec{a_i}, t)$ accounts for the transmit antenna and its' value depend on the direction of the transmitted wave. Both the air to ground and ground to air interfaces are implemented by T_{a-g} and T_{g-a} where $g_d(t)$ deals with the ground influences. Propagation losses and time-delay are introduced by $8\pi^2 R_t R_r c$ and t_d. The IR of the target and the receiving antenna at last are accounted for by $\Lambda_0(t)$ and $h_{N,Rx}(-\vec{a_s}, t)$, respectively.

This model is used to calculate a A-scan, $b(\vec{r_a}, t)$ which is a vital part in forming the synthetic B- or C-scan, i.e. the point-spread function (PSF). How this is done will be explained further in the sections on PSF formation.

3.1.2 Inverse Wiener filter with quality criteria

In Equation 3.1 the relation between the migrated image, $\widehat{\Lambda}(x, y, t)$, the point-spread function, $w(x, y, zo, t)$, and the acquired data, $c(x, y, t)$ is denoted. This subsection treats on how to retrieve the point-spread function out of the aquired dataset to give us the migrated image. To obtain this scattering matrix the point-spread function needs to be deconvolved out of the acquired data. This leads to

$$\Lambda_{zo}(x,y,t) = c(x,y,t) \otimes^{-1}_{x,y,t} w(x,y,z_o,t) \tag{3.3}$$

where:
$\Lambda_{zo}(x,y,t)$ migrated image
$c(x,y,t)$ acquired data
$\otimes^{-1}_{x,y,t}$ deconvolution
$w(x,y,z_o,t)$ point-spread function

A first issue of the point-spread function is that it depends on the depth of the point-scatterer, which prevents the possibility of calculating $\Lambda_{zo}(x,y,t)$ in one step. It can be shown that the depth-dependency of the point-spread function is not very strong and therefore an approximation is made [4], which is validated more thoroughly in Chapter 4. The depth-dependency is omitted and a fixed depth is chosen for the point-scatterer. The chosen depth-value depends on the application but a frequently used value in landmine detection applications is $z_0 = 6$ cm [4]. By fixing the depth of the scatterer Equation 3.3 can be calculated in one step.

The second issue is that the deconvolution operation is very computationally intensive, so this operation is performed in the frequency-wavenumber domain. But due to noise and the finite bandwidth equation 3.3 is ill-posed, so a straightforward true inverse filter can not be used. Therefore the deconvolution needs to be regularised and this is done by means of a inverse Wiener filter [6]. This filter minimises the variance of the error between the restored input data (obtained by convolving the result with the point-spread function) and the original input data, and allows for regularisation by means of a single parameter. Since the filtering is done in the frequency-wavenumber domain the acquired dataset and the point-spread function are Fourier transformed with the following transform

$$X(k_x,k_y,\omega) = \int\int X(x,y,t)e^{ik_xx+ik_yy-i\omega t}dxdyd\omega \tag{3.4}$$

The deconvolution is done by the inverse Wiener filter, which is described as

$$\widehat{\Lambda}(k_x,k_y,\omega) = \frac{C(k_x,k_y,\omega)W^*(k_x,k_y,\omega)}{W(k_x,k_y,\omega)W^*(k_x,k_y,\omega)+\beta} \tag{3.5}$$

where:
$C(k_x,k_y,\omega)$ Fourier transformed acquired data-set
$W(k_x,k_y,\omega)$ Fourier transformed point-spread function
β regularisation parameter

The regularisation parameter is actually the inverse of the SNR, which is difficult to be determined beforehand because the signal is not known. To resolve this a procedure is introduced that determines the performance of the filter based on two quality criteria: the Energy ratio and the Error [8].

- Energy ratio
  The energy ratio is the ratio of the energy of raw data and the energy of the deconvolution result.

$$\gamma = \frac{||\widehat{\Lambda}(x,y,t)||_2}{||c(x,y,t)||_2} 100\% \tag{3.6}$$

- Error
  The error is the measure of difference between the original signal and deconvolution result convolved with the point-spread function. Ideally this convolution should result in exactly the recorded B-scan.

$$\epsilon = \frac{||c(x,y,t) - \widehat{\Lambda}(x,y,t) \otimes w(x,y,t)||_2}{||c(x,y,t)||_2 + ||\widehat{\Lambda}(x,y,t) \otimes w(x,y,t)||_2} 100\% \tag{3.7}$$

  Since the convolution is a computationally intensive operation $\widehat{\Lambda}(y,t) \otimes w(y,t)$ is performed in the frequency domain and calculated as

$$\widehat{\Lambda}(x,y,t) \otimes w(x,y,t) = ifft2[\widehat{\Lambda}(k_x,k_y,\omega) \cdot w(k_x,k_y,\omega)] \tag{3.8}$$

Ideally the error would be zero, so the natural approach is to vary the regularisation parameter to minimise the error. The maximum energy ratio functions as a constraint for the parameter, since not only the error matters but the quality of the resulting image as well. Also the maximum energy ratio needs to be less than 100% to avoid ringing. This maximum energy ratio depends on the scenario (2D- or 3D-data) and good rule-of-thumb values are discussed in the sections that deal with real 2D- or 3D-data. Basically with the error the accuracy can be controlled, while energy ratio controls the stability of the result.

After this $\widehat{\Lambda}(k_x,k_y,\omega)$ is transferred back to time-space domain with the inverse Fourier transformation as described by

$$\widehat{\Lambda}(x,y,t) = \frac{1}{2\pi} \int\int \widehat{\Lambda}(k_x,k_y,\omega) e^{-i(k_x x + k_y y - \omega t)} dk_x dk_y d\omega \tag{3.9}$$

Equation 3.9 results in the migrated dataset.

3.2 2D PSF calculation

This section deals with how to create the point-spread function that is used as an input for the inverse Wiener filter in a 2D scenario. In the first subsection all terms in Equation 3.2 are treated and an implementation of them is giving. The second subsection shows how to construct the 2D PSF (B-scan) from all known data.

3.2.1 Estimation of PSF terms

The propagation losses and time-delay depend on the distance between the object and the array. This requires the location of the refraction points, but since these calculations are the same as with the "exact method" [7] discussed in the previous chapter they will not be discussed again here.

Transmission coefficients

The transmission coefficients are dimensionless values and they are a function of the relative permittivities, permeabilities and the angles of incidence and refraction [6]. The angle of refraction depends on the relative permittivities and the angle of incidence, which in geometrical optics is described by Snell's law

$$sin(\phi_2) = \sqrt{\frac{\epsilon_1}{\epsilon_2}} sin(\phi_1) \tag{3.10}$$

where:
ϵ_1 permittivity in first medium
ϵ_2 permittivity in second medium
ϕ_1 incident angle
ϕ_2 refraction angle

The permeability of the ground and the air are both 1. Geological media in general are considered conductive, this implies that the reflection and transmission coefficients are complex. But it has been shown that at radar frequencies, the case can be accurately approximated by calculating the coefficients as if the media were loss-less [6]. This simplifies the case, because now only the real part of permittivity needs to be used. The angles of incidence and refraction not only depend on the permittivities, but as well on the antenna and target position. The exact method [7] for refraction point calculation gives all the necesary information to accurately calculate the incident and refraction angles. These angles are then used to calculate the transmission coefficients by

$$T_{a-g} = \frac{2\sqrt{\epsilon_0}cos(\phi_1)}{\sqrt{\epsilon_0}cos(\phi_1) + \sqrt{\epsilon'}cos(\phi_2)} \qquad T_{g-a} = \frac{2\sqrt{\epsilon'}cos(\phi_1)}{\sqrt{\epsilon'}cos(\phi_1) + \sqrt{\epsilon_0}cos(\phi_2)} \tag{3.11}$$

where:
ϵ_0 permittivity in free space
ϵ' real part of the permittivity in the ground
ϕ_1 incident angle
ϕ_2 refraction angle

Equations 3.11 imply that the boundary interface is flat. This is correct in our experimental set-up, but this may not be realistic for actual field conditions.

In Figure 3.2 the point-scatterer is positioned at the center (50 cm) of the scan-line at a depth of 6 cm. The transmitting and receiving antenna were located at respectively 52.5 cm and 26 cm above a ground with $\epsilon_r = 3.03$.

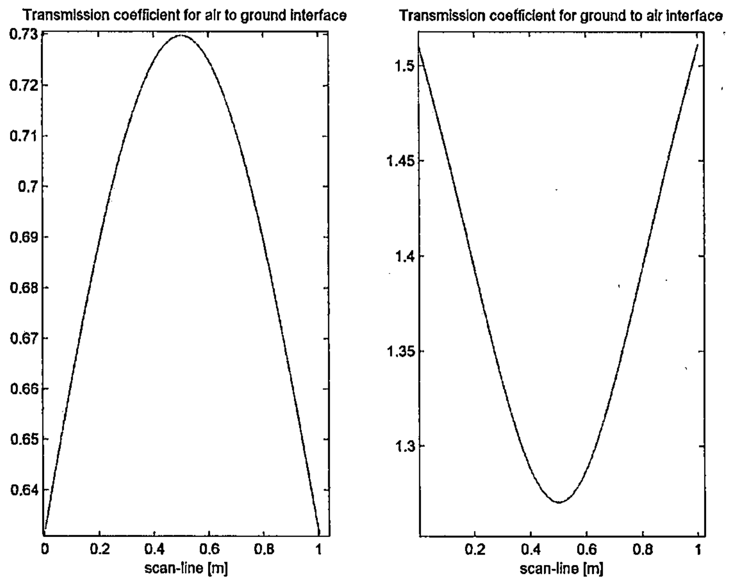


Figure 3.2: Transmission coefficients as a function of antenna position

Ground impulse response

The effects of the ground on the phase and magnitude fired of the fired pulse are accounted for by modelling the ground as a low-pass filter, which parameters depend on the soil characteristics (permeability, permittivity, losses) and the two-way path length (from the transmitter to the target back to the receiver) [6]. The filter is expressed as

$$G_d(\omega) = e^{-(\alpha+j\beta)d} \tag{3.12}$$

where:
$\alpha = \omega\sqrt{\mu_0\epsilon'}\sqrt{\frac{1}{2}(\sqrt{(1+\tan^2(\delta))}-1)}$ attenuation constant
$\beta = \omega\sqrt{\mu_0\epsilon'}\sqrt{\frac{1}{2}(\sqrt{(1+\tan^2(\delta))}+1)}$ phase constant
d two-way travel path length in the soil

The loss-tangent $\tan(\delta)$ and permittivity ϵ' are in fact frequency dependent, but determining these characteristics over a large frequency band can be quite difficult and therefore an approximation is made that assumes them constant over the band of interest. Now the desired constants can be determined with one measurement in the middle of the frequency band. The attenuation and phase constant can now be determined from

$$\alpha = \omega\sqrt{\mu\epsilon'}\frac{\tan(\delta)}{2} \tag{3.13}$$

$$\beta = \omega\sqrt{\mu\epsilon'} \tag{3.14}$$

where:
ϵ' permittivity in the medium
μ permeability in the medium
$\tan(\delta)$ loss-tangent of the soil

Equation 3.12 can be written as

$$G_d(\omega) = e^{-j\omega d\sqrt{\mu\epsilon'}}e^{-\omega d\sqrt{|\mu|\epsilon'}\tan(\delta)/2} \tag{3.15}$$

So far we've been speaking about a filter in the frequency-domain. In order to get the needed impulse response (in the time-domain) Equation 3.15 is transformed to the time-domain by means of an inverse Fourier transform (Equation 3.16).

$$g_d(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} G_d(\omega)e^{j\omega t}d\omega \tag{3.16}$$

The impulse response of the ground can now be expressed as

$$g_d(t) = \frac{d\sqrt{\mu\epsilon'}(\tan(\delta)/2}{\pi[(t - d\sqrt{\mu\epsilon'})^2 + (d\sqrt{\mu\epsilon'}\tan\delta/2)^2]} \quad (3.17)$$

where:
d distance (in meters) propagated in the ground
ϵ_0 permittivity in free space
ϵ' real part of the permittivity in the ground
$\tan(\delta)$ loss-tangent of the soil

An example of such an impulse response is shown in Figure 3.3. The soil characteristics in this example are $tan(\delta) = 0.08$ and $\epsilon_r = 3.03$. The distance to the point scatterer depends on the antenna position, and is therefore different for every A-scan. So it follows that for every A-scan the ground impulse response needs to be recalculated. Figure 3.3 shows the impulse response when the antenna and point scatterer are directly above eachother. It can be shown that the peak concurs with a distance of 12 cm. Since the depth of the scatterer is 6 cm this is in accordance with a two way travel path of 12 cm.

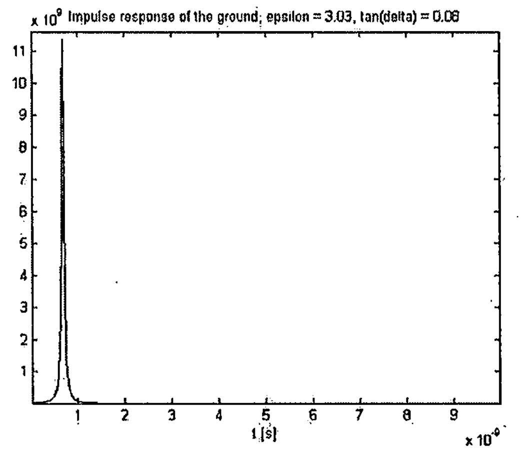


Figure 3.3: IR of ground for parameters $\epsilon_r = 3.03$ and $tan(\delta) = 0.08$

Excitation voltage and IRs of the antennas

The system parameters $h_{N,Tx}(\vec{a_i}, t), h_{N,Rx}(-\vec{a}_s, t), \frac{dV_s(t-t_d)}{dt}$ are combined into one vector acquired by one measurement. The angle-dependency of the IRs of the antennas is dismissed here for ease of implementation, so for every A-scan the same system vector is used. The convolution of the impulse responses and the excitation voltage is the received reflected signal after firing the pulse from the transmitter towards a metal plate (that acts as a mirror). The measurement set-up can be seen in Figure 3.4.

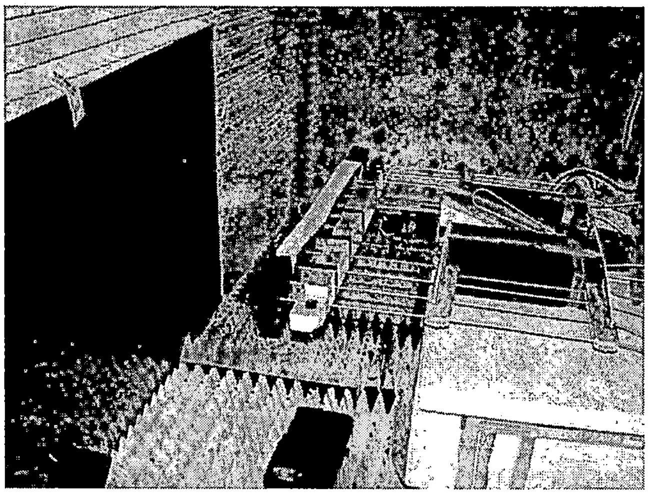


Figure 3.4: Measurement set-up for determining the convolution of excitation voltage with the IRs of antennas

Impulse response of the point-like target

Equation 3.2 also denotes the influence of a point-like target. It has been decided not to implement this by convolving with the IR of a point-like target, but to implement this differently. Small point-like targets have the property of time-differentiating incoming waves upon reflection [9]. This influence on the A-scan is implemented by simply differentiating in the time dimension as the final step of the PSF formation.

3.2.2 Formation of 2D PSF

All terms in Equation 3.2 have been described in the previous chapter and what remains is the implementation to actually create the 2D PSF. The point-scatterer is placed at the center of the mechanical scanning direction at a depth of 6 cm. The processing is summarised in the following step program.

- Step 1.
  Start as if the array were at the beginning of the mechanical scan direction.

- Step 2.
  Calculate the first term $\frac{T_{a-g}T_{g-a}}{8\pi^2 R_t R_r c} g_d(t)$ of Equation 3.2.

- Step 3.
  Determine the time-delay t_d. The time-delay depends solely on the distance travelled, i.e. the free space path length and the sub-surface path length. Both are divided by the respective wave velocities to give the travel times in the two media.

- Step 4.
  As said before the system parameters $h_{N,Tx}(\vec{a_i}, t)$, $h_{N,Rx}(-\vec{a}_s, t)$, $\frac{dV_s(t-t_d)}{dt}$ are combined into one vector, from here on called the system vector, which is the same for all calculated A-scans. This system vector is shifted with the time-delay calculated in the previous step. This shift is implemented by padding zeros at the beginning of the vector. Since the point-spread function has the same dimensions as the raw data it's possible that when many zeros are padded the last time-part of the system vector is cut off. This will reduce the information content in the point-spread function, so to prevent this cut-off the minimum number of padded zeros is subtracted from every padded zero sequence. In other words, the minimum time-delay of the entire B-scan is calculated and this is subtracted from all the time-delays calculated. This reduces the zero-padding (and so the cut-off) and therefore maximises the information content in the point-spread function, without removing the important relative time-delay information between the A-scans. Basically, the point-spread function is shifted earlier in time so that it can contain a larger part of the system vector.

- Step 5.
  The convolution in time is performed with the first term and the time-shifted system vector as described by

  $$f \otimes g = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{3.18}$$

  The result of this convolution is the synthetic A-scan.

- Step 6.
  Take one step further on the scan-line and start over from step 2 until all A-scans are computed.

- Step 7.
  Form the B-scan from all A-scans.

- Step 8.
  Take the 2-norm of the B-scan (to normalise the energy).

An advantage of this approach is that, when the configuration (radar and soil) conditions remain the same, the PSF only needs to be calculated once. This PSF can then be used to migrate all datasets. Figure 3.5 shows the

PSF where the transmitting and receiving antenna are located at 52.5 cm and 26 cm above the ground, depth in the soil of the point-like scatterer is 6 cm and the soil has $tan(\delta) = 0.08$ and $\epsilon_r = 3.03$. The hyperbola can clearly be seen.

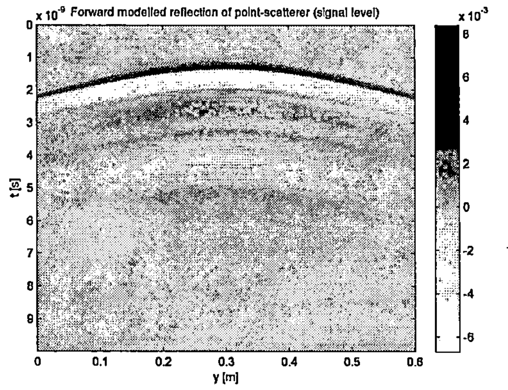


Figure 3.5: An example of a point-spread function (magnitude is dimensionless)

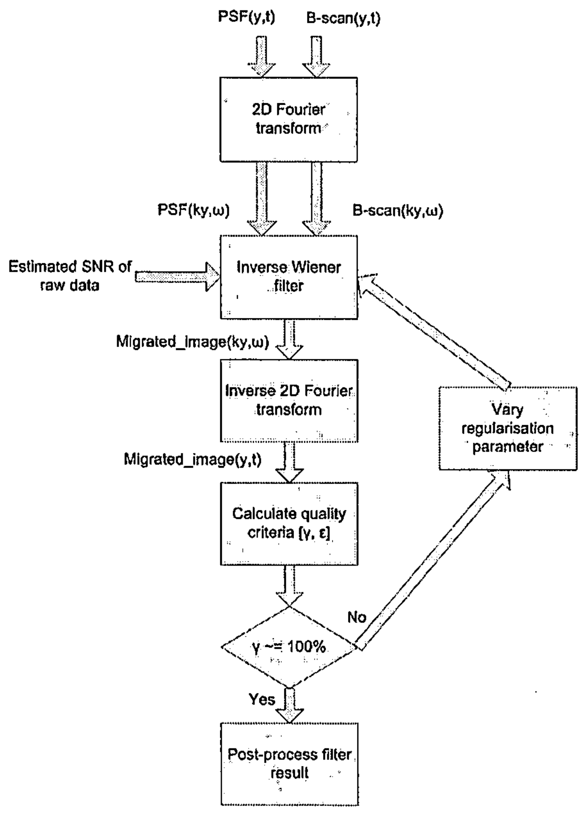
3.3 2D space-time deconvolution

In the first subsection the actual inverse filtering operation will be discussed. This filter operation introduces a shift in every dimension of the result which needs to be compensated. This compensation is discussed in the second subsection after which the results and performance of the algorithm are compared to those of the diffraction stack algorithm.

3.3.1 Inverse filtering operation

The inverse Wiener filter and quality criteria are implemented as described by Equations 3.5, 3.6 and 3.7. The filter operation will only work if both the PSF and the raw data have matching dimensions and resolutions. By experience it's determined that good 2D deconvolution results have a maximum energy ratio of 100% so to obtain this the loop illustrated in Figure 3.6 is used.

The error is calculated as well, but as shown by the loop it does not play a role in determining the optimal regularisation parameter. It's only used as a performance indicator that gives some information on the migrated result.



PSF(y,t)
B-scan(y,t)
2D Fourier transform
PSF(ky,ω)
B-scan(ky,ω)
Estimated SNR of raw data
Inverse Wiener filter
Migrated_image(ky,ω)
Inverse 2D Fourier transform
Migrated_image(y,t)
Vary regularisation parameter
Calculate quality criteria [γ, ε]
No
γ ~= 100%
Yes
Post-process filter result

Figure 3.6: Flowchart of the deconvolution filter operation

3.3.2 Shift compensation

The deconvolution introduces a shift in both dimensions of the result. The point-spread function is transformed to the frequency-wavenumber domain and so becomes complex. The complex PSF is introduced to the filter and as is known phase information in the frequency domain concurs with a time-shift in the time domain. The same happens in the space-dimension because that dimension of the transformed PSF is also complex. These shifts only depend on the PSF and are therefore constant regardless which dataset is migrated. The result needs to be compensated in space and time in order to get the objects at their proper location. The goal is to determine a fixed shift for all dimensions and apply that to the deconvolution result as being a part of the calibration process.

Space shift

The shift introduced by the filter along the scan-line direction will always be half the length of the scan-line, regardless of how long the mechanical scan direction is. In the dataset shown in Figure 3.7 one object is located around 80 cm and the effect of the filtering and compensation can be seen. Before compensation the target is imaged at 30 cm and after compensation the object is imaged at the correct position.

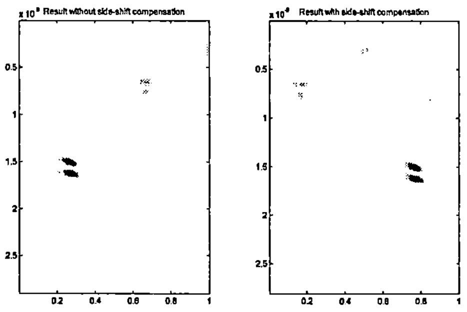


Figure 3.7: Space-shift introduced by filtering and the compensation

Time shift

As said earlier the shift that occurs in time is constant and needs to be compensated. Determining the necessary shift is based on the intuitive relation between the locations of the strongest reflection in the raw data and in the migrated dataset. Even though the energy in the raw data is spread over a large area by the hyperbola the strongest reflection will appear at its' apex, in other words the object location. The location of this reflection in the recorded B-scan should concur with the strongest value of the migrated image. The procedure is as follows: First the strongest reflection is located in the recorded B-scan, after which the strongest signal level in the deconvolution result is located. The difference denotes the time shift that is introduced by the filter. The final step involves the deconvolution result being shifted circularly. This is further illustrated in Figure 3.8.

Figure 3.8 shows the reflection of a landmine after clutter removal. The original A-scan is shown in blue and the deconvolution result in red. The maxima are clearly not in the same position and so they can accurately give the time-shift that will be introduced when using that particular PSF.

3.3.3 Migration results

First the deconvolution results of the same free space data focused with the diffraction stack will be given and discussed. After that results of the 2D version of the sub-surface scenario treated in Chapter 2 are shown.

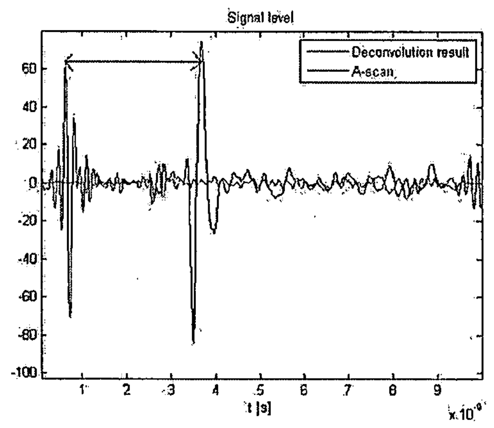


Figure 3.8: Time-shift compensation based on maximum signal levels

Free space data

This scenario involves a sphere at 42 cm from the transmitting antenna. Equation 3.2 deals with sub-surface scenarios; therefore it has to be adapted to be suited for the free space case. This means that the transmission coefficients T_{a-g}, T_{g-a} are set to 1 and the ground IR $g_d(t)$ is $\delta(t)$. The assumed depth of the scatterer of the PSF has been set at 42 cm. The pre-processed B-scan is shown in Figure 1.7.

The result of the deconvolution is in the space-time domain so the time-axes is converted to the space-domain (in order to give accurate depth information). Since this is free space data the conversion is straightforward. The first .9 ns are cut off, this corresponds to the time that the wave travels from the transmitter to the receiver (26.5 cm). The rest of the time-information corresponds to a two-way travel path, so what remains is the time-information which has a linear relation with the depth. In Figure 3.9 the results of the deconvolution algorithm are shown. In Figure 2.3 the results of the diffraction stack algorithm can be seen.
The sphere can be seen clearly at the correct location in the deconvolution result (center of the image). The dimensions of the object in the image concur with the actual dimensions (diameter of 2 cm). The result is better than with the diffraction stack algorithm, which clearly has a lower resolution. Table 3.1 shows the quality parameters and the calculation times.
We can conclude by saying that the new migration algorithm gives higher resolution free space images at much lower computational expense.

Sub-surface data

The original B-scan is shown in Figure 3.10. The data used is the same as the sub-surface data used with the diffraction stack algorithm, but only the

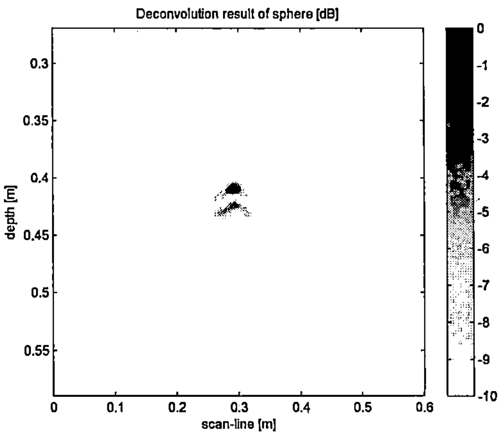


Figure 3.9: Result of deconvolution

	Calculation time	Image size
Diffraction stack	55 s	126 x 61
Deconvolution	< 5 s	2048 x 667
Error	23%	
Energy ratio	100%	
Assumed SNR	41 dB	

Table 3.1: Quality criteria and performance of free space result

B-scan of the middle loop of the receiving array is used in this section. A hyperbola can clearly be seen at the right side and at the left some ground residu remains. The object in the ground is a PNM-2 landmine model at a depth of around 6 cm, where the transmitter is positioned 52.5 cm above the ground. The soil characteristics are assumed to be $tan(\delta) = 0.08$ and $\epsilon_r = 3.03$. The results of the migration by deconvolution and diffraction stack can be seen in Figure 3.11, respectively.
The deconvolution and diffraction stack yield comparable results; both show the PMN-2 at the right and some ground reflection at the left. The deconvolution result does have a small artefact above the target, which can be attributed to the low-pass filter property of the inverse filter. The performance figures are shown in Table 3.2.
We can conclude by saying that the new migration algorithm gives comparable sub-surface images at a higher resolution at much lower computational expense.

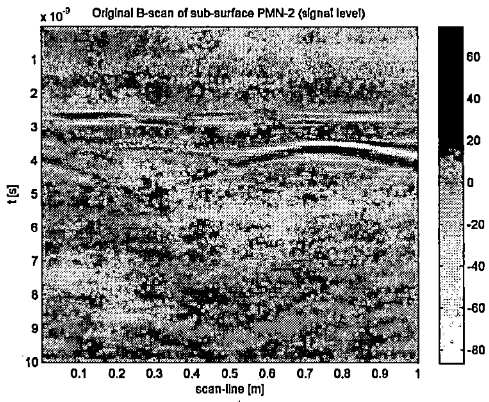


Figure 3.10: Raw B-scan

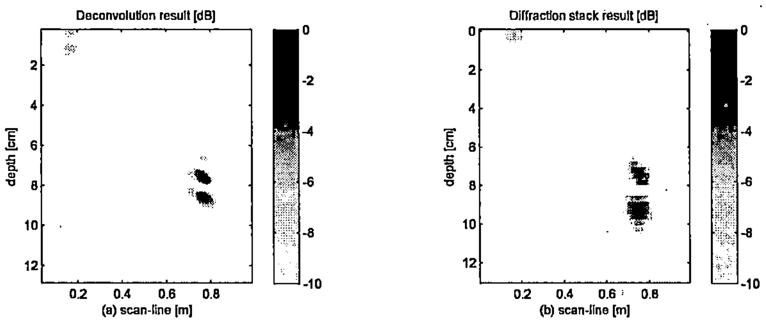


Figure 3.11: Sub-surface migration result (a) Deconvolution, (b) Diffraction stack

	Calculation time	Image size
Diffraction stack	71 s	44 x 101
Deconvolution	< 5 s	1024 x 410
Error	11%	
Energy ratio	100%	
Assumed SNR	38 dB	

Table 3.2: Quality criteria and performance of sub-surface result

3.4 3D array PSF formation and 3D space-time deconvolution

The eventual goal of the migration algorithm is to generate 3D results. The formation of the 3D PSF is described in this section after which interpolation to increase the resolution is explained. The actual deconvolution operation has many similarities with the 2D scenario and so will only be briefly discussed in section 3.4.3. For accurate imaging all targets multiple PSFs are needed, which is first confirmed with modeled data and then followed by actual GPR data. In the conclusions we reflect on the performance of the developed algorithm.

3.4.1 3D PSF for central array channel

In this stage the algorithm is extended to be able to deal with 3D data, which of course needs a 3D PSF as input for the inverse Wiener filter. In this subsection the PSF is constructed as if the array consisted of 13 identical receiving loops, only the system vector of the central channel was used. The point-scatterer was positioned at the middle of the scan-line as (with the 2D scenario) but now also at the center of the aperture (directly underneath the central channel). The formation of a 3D PSF is an extension of the method presented in section 3.2. All terms in Equation 3.2 are calculated similarly remembering that the path lengths of the pulse are now in 3D. The following procedure is followed to arrive at a 3D PSF for the central array channel.

- Step 1
  Start at the beginning of the aperture (left most loop).
- Step 2
  Calculate 2D PSF as described in section 3.2 for a point-scatterer located at the center of the aperture and scan-line at a depth of 6 cm.
- Step 3
  Go to the next loop and repeat step 2 untill all 13 B-scans are calculated.
- Step 4
  Combine all B-scans to form C-scan.
- Step 5
  Normalise the C-scan by dividing it by its' total energy

The process is illustrated with a flowchart in Figure 3.12. This results in a C-scan consisting of 13 B-scans, which in turn yields a resolution of 7 cm over the aperture dimension. This is insufficient for landmine detection applications and therefore interpolation is applied.

3.4.2 Interpolation of array data and PSF

To achieve a higher resolution interpolation is applied over the aperture dimension. Implementing the interpolation on a standard desktop machine quite quickly led to memory problems. Through trial and error the resolution was set at 1 cm over the aperture and the scan-line, which is deemed

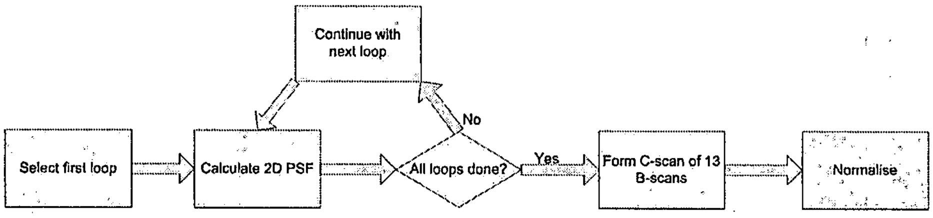


Figure 3.12: Flowchart of 3D PSF formation for central array channel

acceptable for landmine detection purposes. The memory problems also led to chosing linear interpolation for the method, because it is one of the simpler ones. The linear method is a standard method that's illustrated in Figure 3.13.

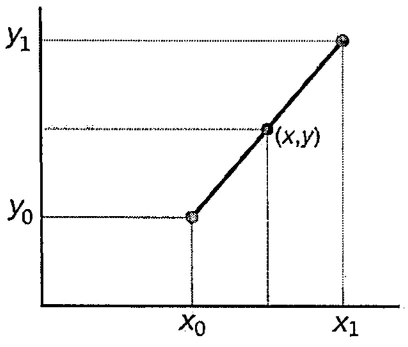


Figure 3.13: Example of linear interpolation

Further research should definitely also look at other interpolation methods (such as spline) because an easy quality improvement can be expected from this. The result of the interpolation is shown in Figure 3.14. It shows that the resolution has increased but that the sides of the PSF aren't completely smooth.

3.4.3 3D space-time deconvolution

The first section deals with the actual filtering operation, the second describes the determination and compensation of the shifts introduced by the filter while the last section shows the results.

Filtering operation

The 3D implementation is virtually the same as the 2D implementation described in section 3.2, only now the 2D raw data and PSF are replaced by their 3D counterparts. The loop used to determine the optimum regularisation parameter is the same as shown in Figure 3.6 only that the (inverse)

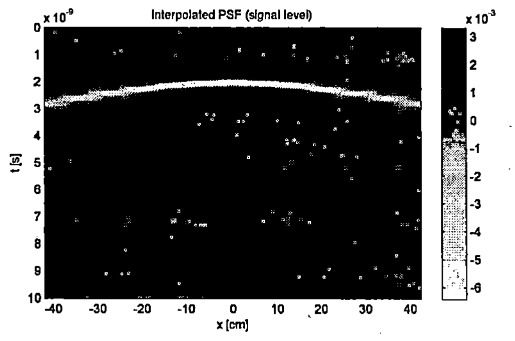


Figure 3.14: Interpolated PSF over aperture dimension

Fourier transforms are now 3D and that the maximum energy ratio now is determined to give optimal images when set at 5%.

Shift compensation

In the 3D scenario, next to the shifts in time and over the scan-line, also a shift over the aperture dimension is introduced. Just like with the shift regarding the scan-line this shift is half of the length of the aperture. Since the other shifts are the same as with the 2D case all shifts can be easily compensated.

Migration results

Only the subsurface scenario with two PMN-2 mines will be discussed in this section. Both of them are located at around 80 cm of the scan-line at a depth of 6 cm. The 3D results are shown in Figure 3.15 projected using ASWEP as discussed in Chapter 1.

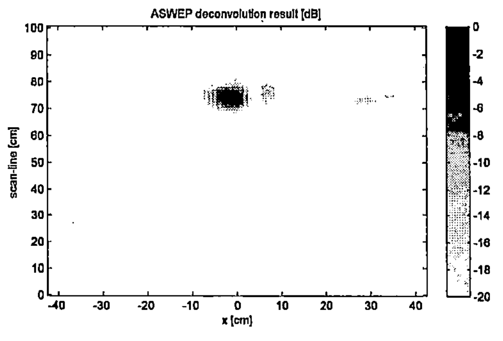


Figure 3.15: Result of 3D deconvolution using one PSF

Directly it's clear that only the center target can be seen well. The side target is very weak and therefore this result is just unacceptable. One reason for this result is that only the system vector of the center loop was used for the implemenation. The other reason is that the used PSF is calculated with the point-scatterer located at the center of the aperture and therefore it's logical that objects located at the center are imaged stronger. When the object is located at the center the hyperbola will have the strongest correlation with the PSF and so have a stronger migrated result. In a single-target scenario this will not give problems and single targets at the side will still be imaged. In a multi-target scenario however the targets located at the center will be imaged stronger than targets at the side of the array and it's possible that side targets become more vague.

3.4.4 Modeling result

A target that's located at the same location as the point-scatterer will always have a stronger migrated result than a target at a different location. Therefore an algorithm that uses multiple PSFs (where in every PSF the scatterer's location over the aperture dimension was different) is proposed to deal with this issue. Synthetically generated datasets were used to investigate the necessity and usefulness of multiple PSFs. Synthetic data has the advantage of being able to accurately determine the influence of target positions and multiple PSFs, since the input is exactly known and noise is by definition zero. One scenario in particular is of interest to us, namely the two target scenario with one center target and one side target. To analyse this scenario a zero matrix was generated with the same size as the other datasets. Two "objects" were placed in this matrix by giving two sets of elements a non-zero value. The result of this is illustrated in Figure 3.16.

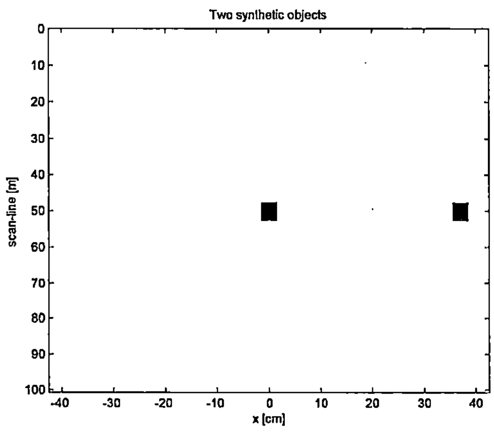


Figure 3.16: Two synthetic targets in the matrix

To create the "raw" C-scans both targets are convolved separately with point-spread functions that have point-scatterer at the same location as the targets, i.e. the center target is convolved with a PSF where the scatterer is located at the center where as the side target is convolved with a PSF with a scatterer at the side. Both convolution results are then summed and that gives the C-scan that for our filter will function as the raw dataset. Both targets are located at the center of the scan-line; Figure 3.17 shows the raw data at that position. Both the hyperbolae can clearly be seen.

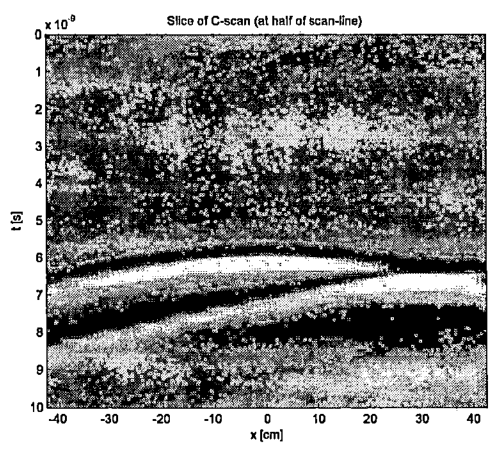


Figure 3.17: Two hyperbolae in the 'raw data'

Figure 3.18 shows the result when the middle PSF is used, the side PSF is used and the combination of these results.

Figure 3.18 very clearly illustrates the need for multiple PSFs and the combining of their results. When separated they're not able to image both targets with equal strenghts, but by combining the results will image both targets properly. Concludingly we can say that the use of multiple PSFs is necessary to image all targets correctly over the aperture dimension.

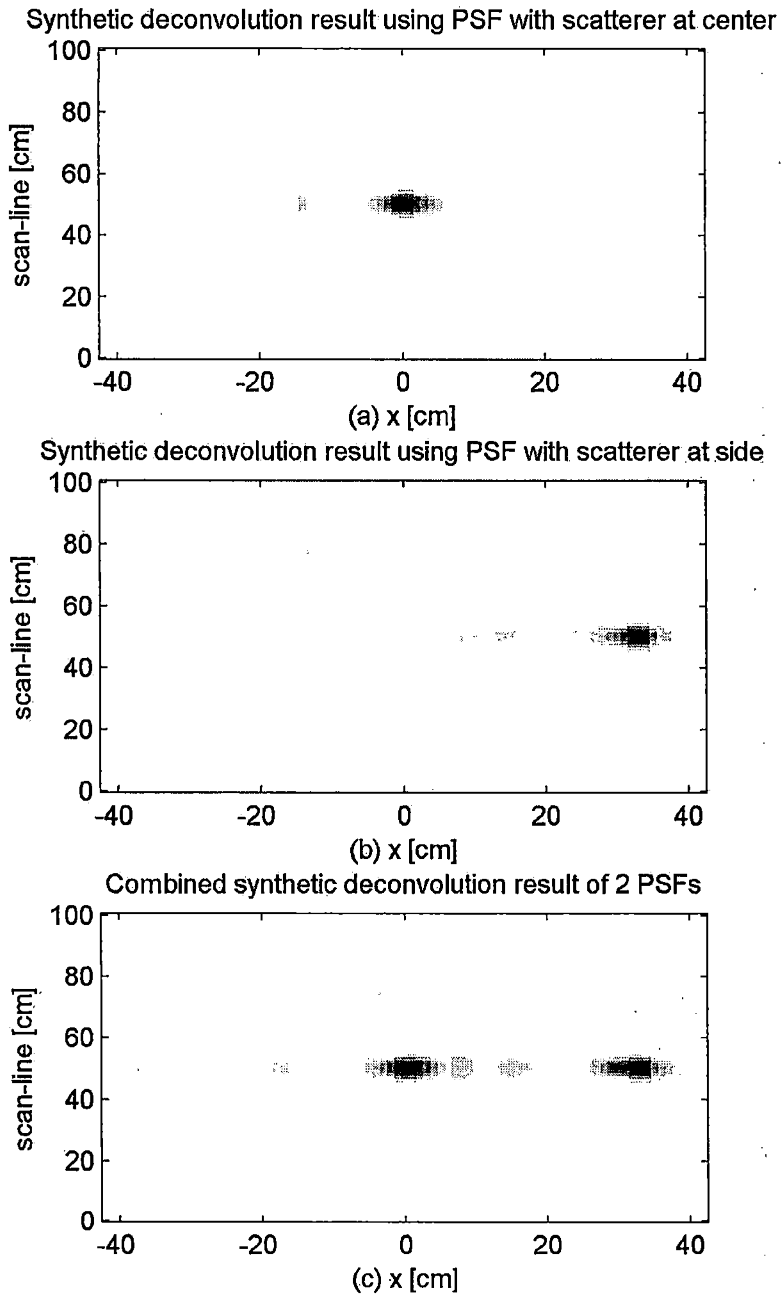


Figure 3.18: Modeling result (a) only PSF with center scatterer, (b) only PSF with side scatterer, (c) Combined result

3.4.5 Array PSF with amplitude correction

Multiple PSFs will give multiple deconvolution results which need to be combined to one. How this is done will be discussed first after which the choice for the number of PSFs will be explained. The B-scans of the PSFs calculated in this chapter use the corresponding system vector of that loop to give extra accuracy. After implementing an algorithm with multiple PSFs the side-targets were still imaged weaker which was resolved by an extra compensation which is described in the last section.

Combining deconvolution results

Using more PSFs also generates more migrated matrices which obviously need to be combined to one. Three main possibilities remain for the combination: Add all results, Combining with overlap, Combining without overlap. The first method is the simplest one but shows many artefacts, so soon it became clear that this was not a good method. The second and third method use only a part of each deconvolution result to form the final migrated matrix. It can be argued that a particular PSF with it's scatterer at f.e. x = 35 cm will have the best result for all targets located at x = 35 cm, therefore when combining the results this PSF result should be used for x = 35 cm. This is illustrated more clearly in Figure 3.19.

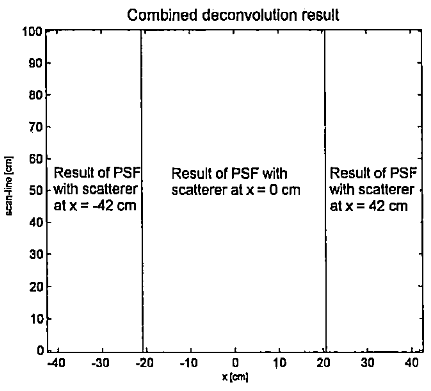

Combined deconvolution result
Result of PSF with scatterer at x = -42 cm
Result of PSF with scatterer at x = 0 cm
Result of PSF with scatterer at x = 42 cm
scan-line [cm]
x [cm]

Figure 3.19: Combining deconvolution results when using 3 PSFs

In this case the result is divided into three parts and the corresponding sections of the migrated results are used to create the result. Using overlap usually smoothes out the result but also introduces more artefacts than combining the result without any overlap. Therefore it was chosen to combine without overlap (regardless of the number of PSFs used) and smoothing out the final result by applying 2D median filtering. This is a commonly used tool in image processing to remove single spikes or speckles. 2D median filtering is a spatial filtering operation of which each value of the matrix is compared to its nearby neighbours to decide whether or not it is representative. When the window is of equal size of that of a target its' edges will be smoothed as well. In our case a window of 6 cm by 6 cm was used and the median of that window was set to be the new value of the filtered matrix entry. This in combination with the ASWEP as described in Chapter 1 gave the best results, so it follows that all shown 3D deconvolution results are formed this way.

Selecting the number PSFs

Choosing the number of PSFs and their distribution over the aperture was not straightforward, since using more PSFs would treat more objects fairer but will also give an increase in the computational effort (because for every PSF the filtering operation has to be performed). At first a 12 PSF implementation was introduced that placed the scatterers between the loops. This implementation showed both targets better, but the result was still unacceptable. Part of this depended on how the point-spread-function was calculated, because first the 13 B-scans were formed after which they were linearly interpolated. Figure 3.20 shows the problem when the scatterer is located between loops 8 and 9. The hyperbola occurs at the same point in time at the loops 8 and 9, so when this result is interpolated (as indicated in Figure 3.20 with red) no clear apex at the exact location of the point scatterer is formed. This can be resolved by placing the scatterers directly underneath the loop, this operation will place the apex on one of the B-scans after which interpolation has less negative influence. Since hyperbolea of targets naturally have a clear apex it was chosen from here on to only place the point-scatterers directly under the loops.

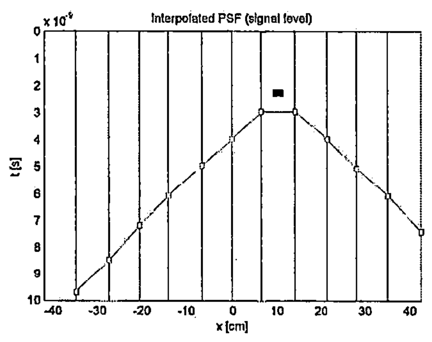


Figure 3.20: Schematic PSF of scatterer placed at 10.5 cm

Several configurations were implemented using 3, 5 or 13 PSFs and there the initial assumption was validated that more PSFs give a better result. So there was opted for an implementation using 13 PSFs but this still showed side targets imaged weaker, as illustrated in Figure 3.21.

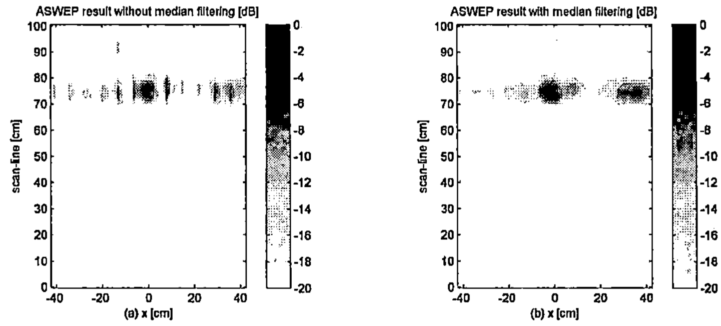


Figure 3.21: Combined result (no overlap) of 13 PSFs (a) without 2D median filtering, (b) with 2D median filtering

The side target is clearly visible but still weaker than the center target. It should also be noted that using different PSFs also means compensating for different shifts. Over the aperture dimension it was found that this shift was always the summation of half the aperture length and the scatterer location. For example, for the PSF with the scatterer at -42 cm the shift was (-42 + 42) = 0 cm.

Amplitude correction

Looking for reasons to account for this difference in strength we went back to the original acquired data. The GPR array has a footprint which non-uniformity is compensated by weighting as discussed in Chapter 1. But when we look at the hyperbolae of targets we see that they require substantial length in both dimensions. We assume that the majority of the energy of a hyperbola of a PMN-2 mine is spread over a surface of f.e. 1 m^2 (so we need about 1 m^2 of hyperbola as input for the migrating algorithm to give a correct result). This is no problem when the target is laid at the center of the aperture, but when a target is laid at the side of the array a large part of the hyperbola (and therefore the energy) is "missed". Simply stated, the array will acquire less reflection energy of side-targets than center-targets.

Having less energy in the raw data also means less energy in the migrated result. This is partly caused by the normalising of all PSFs separately. Normalising is common practice when using an inverse Wiener filter [8] but there's another reason why we've opted for this approach because it has the major advantage of being able to compare all filtering operations by selecting one value for the energy ratio which then will give comparable results for each used PSF. Normalising the PSF does have the disadvantage of removing it's energy information, which is important in relation to the other PSFs. In the un-normalised PSFs already a difference in energy can be seen between the energy of a scatterer which is placed in the center and a scatterer placed at the side. Basically the value which is used to normalise each PSF gives us the relation between the total energies of all PSFs and therefore can be used to compensate the results. In Figure 3.22 the normalisation factor, $\sqrt{\sum_{all_x}\sum_{all_y}\sum_{all_t} C^2_{x,y,t}}$, of each of the thirteen PSFs and the resulting weighting factors for the migrated matrix are shown.

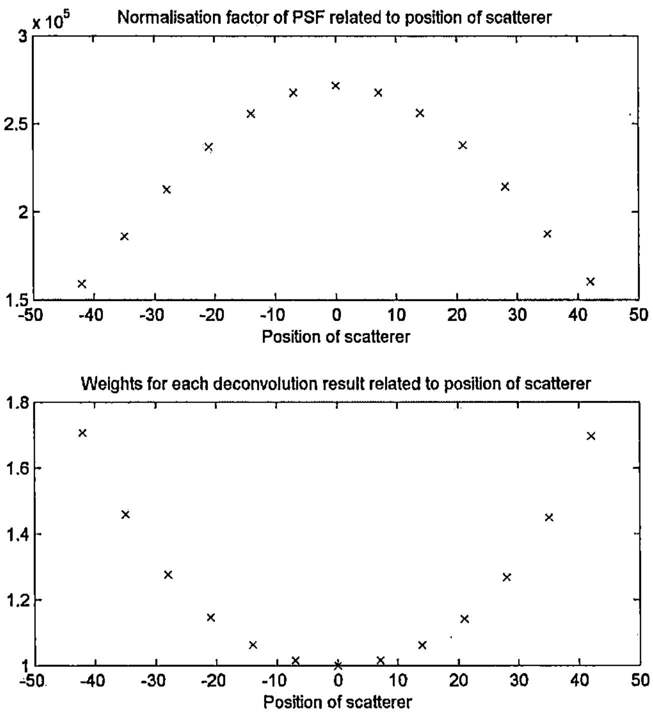


Figure 3.22: The relation between the PSF total energies and their weights

3.5 Reference data set result

This section deals with the result concerning the two PMN-2 mine scenario. This result is obtained by combining the result of deconvolution of 13 different PSFs (each PSF having the point-scatterer under a different loop), subsequent amplitude correction and 2D median filtering. The ASWEP as described in Chapter 1 is used for creating the confidence map. Figure 3.23 shows the results.

The two targets can clearly be seen in the image, but some clutter at the

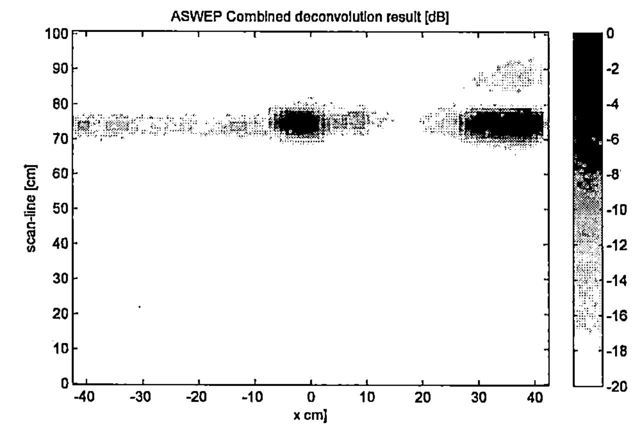


Figure 3.23: Deconvolution result

left and top side remains. The diffraction stack result (Figure 2.14) shows a large artifact right next to the center target, but has less clutter. Even though the deconvolution result has more clutter the two targets are clearly visible, however the quality of the diffraction stack image is higher. The performance figures are given in Table 3.3.

	Calculation time	**Size of migrated matrix**
Diffraction stack	22-23 hrs	27 x 101 x 85
Deconvolution	8 mins	512 x 101 x 85
Avg. Error	26%	
Avg. Energy ratio	5%	
Avg. Assumed SNR	-32 dB	

Table 3.3: Quality criteria and performance of 3D result

It's clear that the proposed method is considerably faster than it's diffraction stack counterpart while still having a higher resolution (in the time-dimension). The high error and very low assumed SNR are related to the maximum energy constraint. It's also logical that the SNR in the 3D case is lower because a larger area gives more noise and does not necessarily add more signal. The 5% was chosen as the maximum energy ratio because this value gives good images for different datasets.

3.6 Conclusions

An extended version of the method proposed in [6] has been developed for the GPR-array and is able to migrate real GPR-datasets at a fraction of the time needed by the diffraction stack algorithm. Varying the regularisation parameter to get the quality criteria (as introduced in [8]) within certain boundaries has proven to give good and consistent results for 2D and 3D datasets. Interpolating the 3D version of the deconvolution algorithm showed to be a good way to increase the resolution over the aperture. Implementing the algorithm with multiple filtering operations (each using a PSF with a different location on the aperture of the point-scatterer) and combining their result is a novelty and is needed to image all targets over the aperture dimension. To image the side-targets as strong as the center-targets another novelty being a new kind of amplitude correction was developed based on the total energy of the corresponding PSFs. All novelties contribute at imaging a real GPR dataset many times faster than diffraction stack, while still giving good results.

Chapter 4

Validation of developed imaging algorithm

This chapter will further validate several approximations and assumptions made in the previous chapter on the developed algorithm. Up till now only one dataset, containing one PNM-2 mine at the center and one PMN-2 mine at the side, has been migrated. The first section will give the migrated results of another dataset to show the performance and capabilities of the algorithm. When developing the algorithm the assumption was made that the depth of the point-scatterer of the PSF had little influence on the result and was therefore given the fixed value of 6 cm [6]. In the second section this will be validated by migrating some datasets with deeper targets and also by using PSFs with varying point-scatterer depth. Finally, the antenna height has little influence on the implementation of the algorithm, since only the PSFs need to be recalculated for a different antenna height. However it is possible that varying the antenna height does have an influence on the quality of the result and therefore in this last subsection two similar datasets with different antenna heights are migrated.

4.1 Different positions of targets

Figure 4.1 shows with what targets this dataset is acquired. At the upper-left corner a PMN-2 mine is laid, to the right a stone is positioned. The bottom-left corner contains a M-14 mine and a piece of barbed wire is located to the side.

To arrive at the migrated result the exact same method was used as with the two target scenario, only this dataset was acquired with the transmitting antenna positioned at 46.5 cm above the ground. Figure 4.2 shows the migrated result for both algorithms and Table 4.1 shows the performances for this 4 target scenario.

Some clutter and artifacts remain and not all targets are visible in the

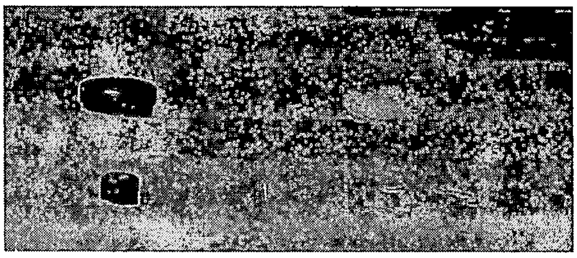
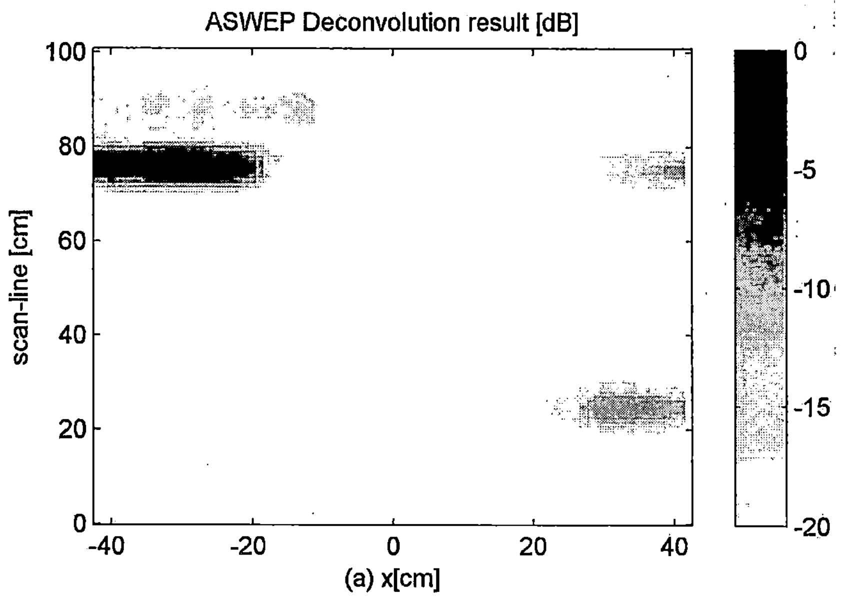


Figure 4.1: Target scenario

	Calculation time	**Size of migrated matrix**
Diffraction stack	22-23 hrs	27 x 101 x 85
Deconvolution	7:11 mins	512 x 101 x 85
Avg. Error	22%	
Avg. Energy ratio	5%	
Avg. Assumed SNR	-30 dB	

Table 4.1: Quality criteria and performance of 3D result

deconvolution result. The PMN-2 mine has definitely the clearest reflection but as can be seen in Figure 4.1 it's also one of the bigger targets. The stone has similar dimensions but the difference in electrical permittivity between the soil and stone is small which is the reason for the weaker reflection. The barbed wire is visible, but the M-14 mine can not be seen (this is contributed to the radar's resolution which is too course to detect this small mine properly). The diffraction stack result has nicer shapes of the targets and offers better positioning, but does have more clutter and artifacts. Again the deconvolution algorithm gives good results at a fraction of the computation time needed for the diffraction stack algorithm.



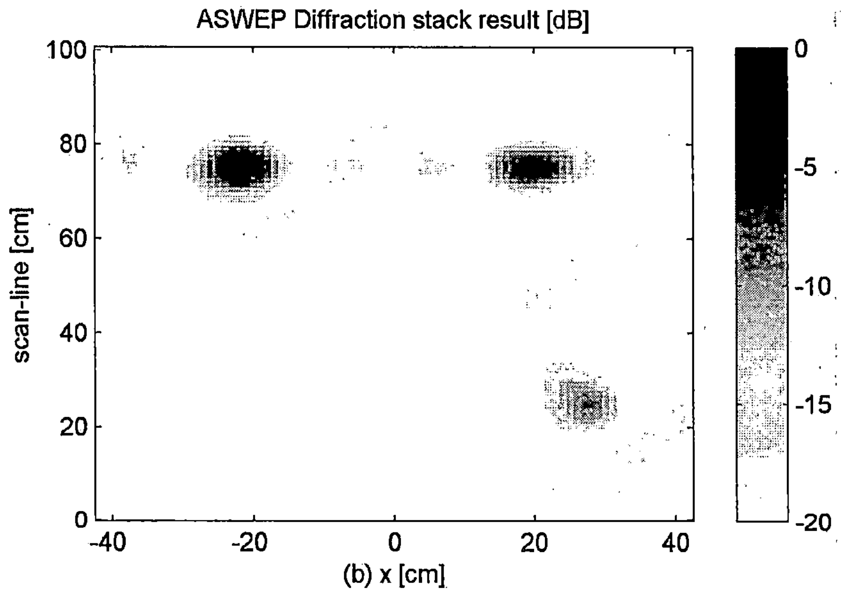


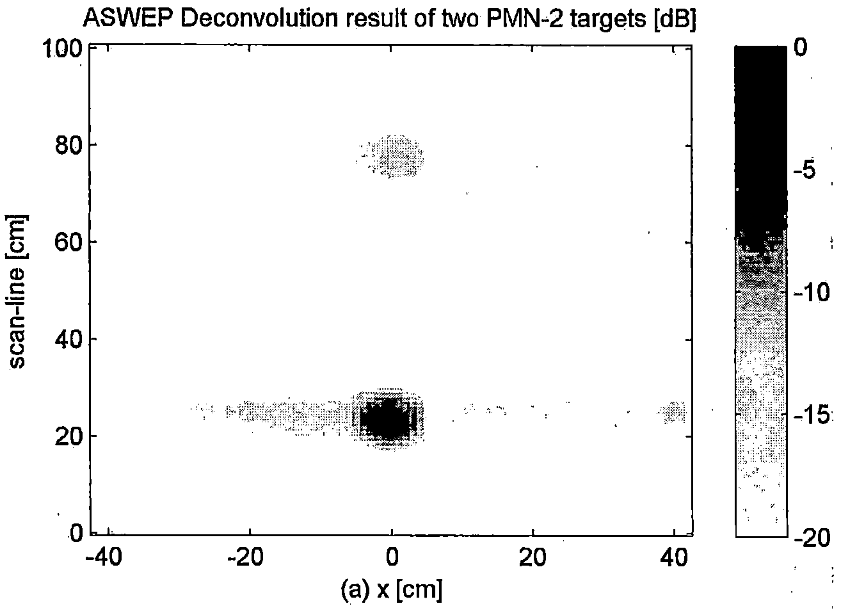
Figure 4.2: Migrated result with antenna height 46.5 cm (a) Deconvolution result, (b) Diffraction stack

4.2 Different depths of targets

This section validates the approximation that the depth chosen of the PSF scatterer and the depth of the actual target does not have a big influence. For this comparison a stretch of 1 m is chosen containing two PMN-2 mines, one at a depth of 5 cm and the other at 10 cm. First in Figure 4.3 the result is shown when processed by the algorithm (PSF depth at 6 cm) then a PSF depth of 10 cm is used and the data is processed again.

The PMN-2 target at 10 cm depth is located at the top and clearly has a weaker reflection, but since this is the case in both results this is due to propagation losses. Since both results are so similar it validates the low influence of the PSF scatterer depth. Now only the deeper target is migrated using the two different PSFs, the results are shown in Figure 4.4.

Again the results are very similar in shape and energy. Since the pulse has a penetration depth of around 20 cm, this applies over the entire depth range of the radar. All results contribute on validating the approximation of using a fixed object depth for the point-scatterer. It is shown that this depth has little influence on the deconvolution result.



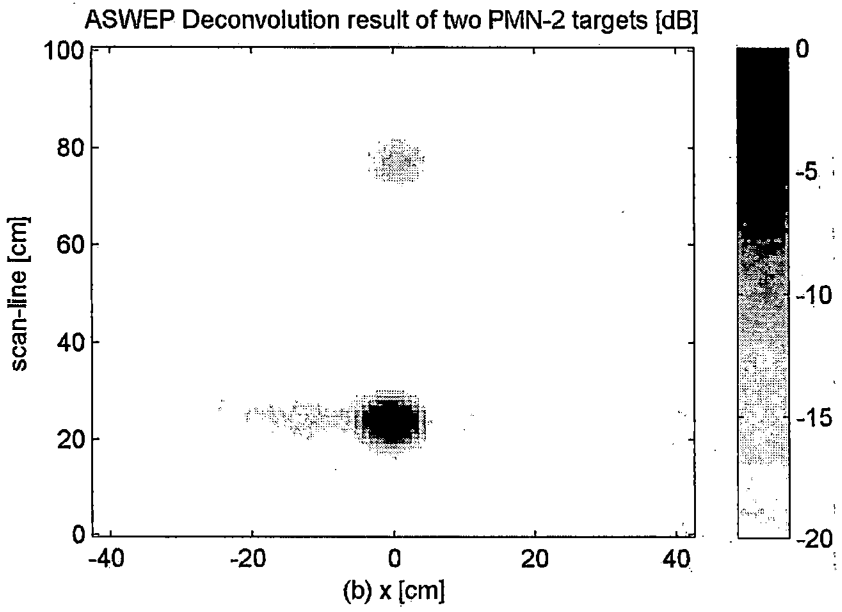
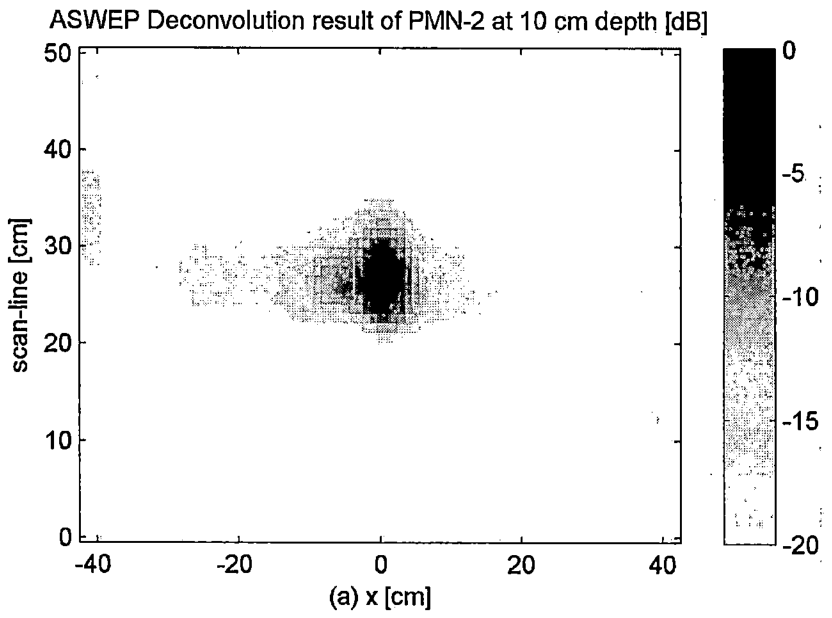


Figure 4.3: Deconvolution result using a PSF with depth (a) 6 cm, (b) 10 cm



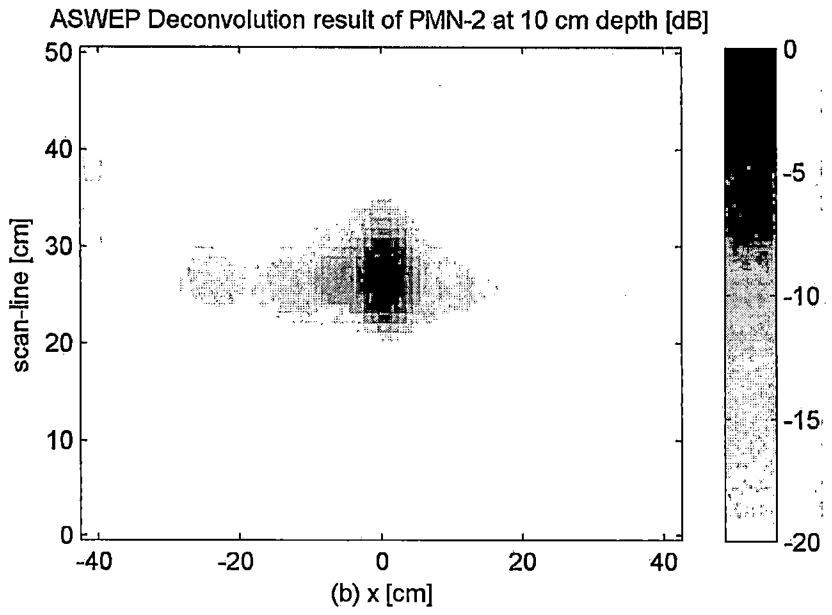


Figure 4.4: Migrated PMN-2 mine with PSF depth (a) 6 cm, (b) 10 cm

4.3 Different elevation of the antenna system

This section will deal with the influence of the height of the antenna system. We have the choice between 46.5 cm and 52.5 cm for the height of the transmitter above the ground. The results shown earlier in this chapter (Figure 4.2) are obtained with an antenna height of 46.5 cm. The result shown in Figure 4.5 is derived from the same target scenario but the data was acquired with an antenna height of 52.5 cm.

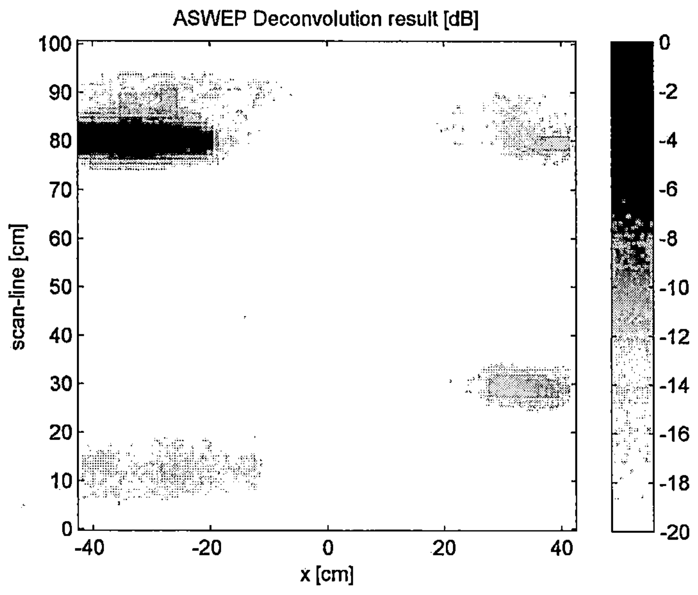


Figure 4.5: Deconvolution result for antenna height of 52.5 cm

The result shown in Figure 4.5 shows all 4 targets, but has definitely more clutter than the result obtained with an antenna height of 46.5 cm. The latter only shows 3 targets but they are far more condensed, so both results have their pros and cons. A slight preference to results obtained at 46.5 cm exist because of its' low clutter and condensed images.

Chapter 5

Summary and conclusions

5.1 Summary

Landmines continue to pose a large threat to people, while clearing them is a dangerous and laborious process. Several detection methods are currently being developend, where GPR has shown to be promising sensor. A novel array-based GPR has been developed in IRCTR for vehicular landmine detection. The GPR uses a single transmit antenna and a linear array of 13 loop antennas receiving the scattered EM field simultaneously. Near-field imaging of buried landmines is being done by focusing the acquired data along the direction of mechanical scanning (SAR) and along the array aperture [2].

The widely used diffraction stack algorithm is a time-inversion technique that migrates data by calculating signal travel times. This algorithm was extended for the array-based GPR and equations were derived to analytically solve the location of the refraction point. The algorithm was tested with real GPR data and showed to image subsurface datasets at the expense of large computational effort.

The currently available methods only take some radar properties and some soil characteristics (in general only ϵ_r) into account. The method proposed in [6] models the reflections from arbitrarily shaped targets as the convolution of the reflection of one point-scatterer (point-spread function) with the collection of point-scatterers (target scattering matrix). This is described by

$$c(x,y,t) = w(x,y,z_0,t) \otimes_{x,y,t} \Lambda_{zo}(x,y,t) \tag{5.1}$$

The $\Lambda_{zo}(x,y,t)$ is the desired result in Equation 5.1 because that basically is the migrated dataset; the objects in the ground are modeled by collections of point-scatterers. The point-spread function ($w(x,y,z_0,t)$) is obtained by forward modeling as described in [6], where the radar characteristics are implemented by using one measurement (reflection of the pulse on a metal plate). The point-spread function is then filtered out of the acquired signal ($c(x, y, t)$ by means of an inverse Wiener filter which results in the migrated result.

First a 2D version of the migration by deconvolution method was developed according to [6] but the first novelty here was to use two quality criteria, Energy ratio & Error [8], to determine the performance of the filtering operation and automatically select a good regularisation parameter for the inverse Wiener filter. The second novelty was implementing the influence of the point-scatterer in the 2D PSF not by it's impulse response, but by differentiating the entire PSF in the time dimension. The developed method could migrate 2D data considerably faster and with a higher resolution than the diffraction stack method, without compromising on the quality of the result.

Extending the 3D version of the algorithm to the array-based GPR was done with several extra steps. The third novelty was forming the 3D PSF by calculating 13 B-scans (one for each loop) and then use interpolation. The raw data was also interpolated to increase the resolution. The filter and quality criteria were extended to a 3D version where it soon became clear that the position of the point-scatterer in the PSF influences the strength of imaged targets. Similar targets at the same position as the point-scatterer are imaged stronger than those targets at other positions. To resolve this the use of multiple PSFs was developed, which is the fourth main novelty of the 3D implementation. This was first confirmed with synthetically generated data where it proved to image targets better over the entire aperture dimension. When this was applied to actual GPR data this still did not image identical targets at different locations over the aperture dimension with equal strengths. The fifth novelty was to correct the deconvolution result per PSF. Targets at the side have less hyperbola energy in the acquired data because of the finite dimensions of the array; a larger part of the energy of a hyperbola of a center target is recorded than that of a side target. Less energy in the raw data leads to weaker reflections in the migrated result. A method was developed that weighted the filtered result with a factor that's inversely proportional to the square root of the total energy of that particular PSF (before normalisation). This weighting is done after filtering so not to interfere with the quality criteria and subsequent regularisation parameter calculation.

The novelties of this thesis work are

1. Introducing the quality criteria to control the accuracy and stability of the inverse Wiener filter.

2. Differentiating the PSF to account for the differentiating property of a point scatterer.

3. Applying interpolation to increase the resolution over the aperture dimension.

4. Using multiple PSFs with scatterers at different positions and combining their results.

5. Weighting the deconvolution result with a factor depending on the energy of the corresponding PSF.

5.2 Conclusions

An advanced imaging algorithm based on migration by parametric, regularised deconvolution has been developed, where the method introduced in [6] was extended to an array-based GPR. The algorithm was tested on real GPR datasets and it is shown that it produces high quality subsurface images, regardless of their position over the aperture dimension. In a 2D and 3D scenario this has shown to give higher resolution results than the diffraction stack algorithm, at a minor fraction of the computational time. It's straightforward that the calculation time increases with scan length, but the increase is much higher with the diffraction stack method. The performance difference and the dependency on scan-length of one particular dataset are illustrated in Table 5.1.

Scan length	1 m	2.5 m
Diffraction stack	22-23 hrs	6.4 days
Deconvolution	4 mins	5:50 mins

Table 5.1: Calculation time dependency on scan-line length

A small degradation in quality is present with regard to the diffraction stack result, mainly because the target images are less condensed and their spacing is less precise. It seems that the images of center targets have nicer shapes and better spacing than their counterparts located at the side, a reason for this can be the omitting of the angle-of-incidence dependency of the antennas.

Furthermore we can conclude that the approximations made in [6] on target depth were valid. It should be noted that the radar can only image to a soil depth of 20 cm. No large differences in the results were noted when using PSFs with scatterer depths of 6 and 10 cms, so this approximation can be made over the entire depth range of the radar.

The influence of the height of antenna system is visible in the deconvolution result and it is preferred to use a height of 46.5 cm, because this in comparison to an antenna height of 52.5 cm results in more condensed images.

5.3 Recommendations for further research

(1) Interpolation technique

The method of interpolation deserves more attention. Because of memory limitations there was opted for linear interpolation, but other methods should definitely be implemented for they can give easy image quality improvements.

(2) Optimal scan-length

The optimal scan-length should be determined. Although an increase of scan-length from 1 m to 2.5 m only takes a couple minutes of processing more, they have the disadvantage that weak targets can be missed because large and strong targets overpower their reflections.

(3) Angle-of-incidence dependency of antennas

The angle-of-incidence dependency of the impulse responses of the antennas have been ignored. When forming one PSF B-scan the same measurement data was used with every A-scan. Further research should investigate if this has a large influence on the result; a performance increase can be expected from this.

(4) Programming efficiency

The programming efficiency of the algorithm should be further investigated. The currently developed software can be made faster and more efficient, f.e. by reducing the resolution in the time-dimension. More attention should also be given to the start value of the regularisation parameter. With the current implementation it starts for every PSF deconvolution operation with the same parameter, an implementation where the final parameter is used as a first input for the next operation could already give a performance improvement.

(5) Position of point-scatterer on scan-line

With all 3D PSFs formed the point-scatterer was positioned at the middle of the scan-line. This does not gave any problems (or biased results) on the available datasets used, but it can be argued that the same problems that were present over the aperture dimension can also occur over the scan-line.

Bibliography

[1] C. Bruschini and B. Gros. A survey of research on sensor technology for landmine detection. *The Journal of Humanitarian Demining*, vol 2.1, 1998.

[2] A. G. Yarovoy, T. G. Savelyev, P. J. Aubry, P. E. Lys, and L. P. Ligthart. Uwb array-based sensor for near-field imaging. *IEEE Transactions on Microwave Theory and Techniques.*, vol. 55: pages 1288–1295, 2007.

[3] T. G. Savelyev, A. G. Yarovoy, and L. P. Ligthart. Weighted near-field focusing in an array gpr for landmine detection. *URSI Radio Science (accepted)*, 2008.

[4] B. Scheers and M. Acheroy. *Ground Penetrating Radar, 2^{nd} edition*, chapter "Migration technique based on deconvolution", pages 283–293. D. J. Daniels ed., IEE Radar, Sonar, Navigation and Avionics Series 15, 2004.

[5] V. Kovalenko. *Advanced GPR Data Processing Algorithms for Detection of Anti-personnel Landmines.* PhD thesis, Delft University of Technology, Netherlands, 2006.

[6] B. Scheers. *Ultra-wideband ground penetrating radar with applications to the detection of anti-personnel landmines.* PhD thesis, Universite Catholique de Louvain and Royal Military Academy, Belgium, 2001.

[7] R. Tanaka. Report on sar imaging. Technical report, Delft University of Technology, Netherlands, 2003.

[8] T. G. Savelyev, L. van Kempen, and H. Sahli. *Ground Penetrating Radar, 2^{nd} edition*, chapter "Deconvolution techniques", pages 298–310. D. J. Daniels ed., IEE Radar, Sonar, Navigation and Avionics Series 15, 2004.

[9] D.B. Davidson. *Computational Electromagnetics for RF and Microwave Engineering*, chapter "The method of moments for surface modeling", pages 189–195. Cambridge University Press, 2004.

Appendix A

Description of developed software

A.1 Diffraction stack

This section deals with the software that migrates 3D subsurface data. It's comprised of two parts, the first does some additional steps of pre-processing and creates the focused box (migrated volume). For every position on the aperture dimension (x) thirteen migrated B-scans (one for each loop) are calculated and summed. The calculation of the B-scans is done by the second file.

FocusedBoxExact.m

First the data is weighted and the time-drift is compensated after which nulling is applied. Then the focused box is formed where the depth is basically the only variable, because the aperture dimension is fixed and scan-line depends on the acquired data. Before migration several properties of the array (heights) and the soil (ϵ_r) need to be specified. The loop is initiated starting at the left side of the aperture and then the 13 B-scans are calculated by initiating another loop.

Exact tt2.m

This function migrates the B-scan of a specific loop. For this it needs the following information: B-scan and it's time- and scan-information, loop-number, position on aperture, dimension of grid, antenna heights and electrical permittivity. Now a large nested loop is started; for each depth the antenna's are placed at all positions on the scan-line. For each antenna position separately the object position is varied over the entire scan-line. For each object position the travel time can be now calculated. First the travel time from transmitter to object is calculated (according to the three scenarios described in Chapter 2) and then the travel time from object to receiver. The total travel time now corresponds to a value in the original B-scan which is added to the grid point where the object is located.

A.2 Deconvolution

This section deals with the developed software for the subsurface 2D and 3D migration by deconvolution algorithm. The 2D and 3D implementations are treated in different sections, but they're built up in the same way. A very important part of this is the point-spread function. It only has to be calculated once for a certain set of parameters, including antenna and soil characteristics. The first subsection describes the formation of the PSF, the second subsection deals with the actual deconvolution operation. Some post-processing of the multiple results for the 3D case and their combination are treated in the final subsection.

A.2.1 2D implementation

The main script is deconv shift.m, which calls to psf exact.m. The PSF is recalculated every time, strictly speaking this is not necessary but does hardly add any significant computation time.

deconv shift.m

The raw data and system vector information are loaded after which the latter is compensated. Some pre-processing is done after which the PSF is calculated. The filter operation (loop) is performed and the criteria are calculated. The final step compensates the time and space shift, resulting in the migrated B-scan.

psf exact.m

The function is called with the parameters regarding the time- and mechanical scan direction, the system vector, epsilon, object depth and the loss-tangent. To further clarify: yo, zo refer to the object. yt, zt refer to the transmitter. yr, zr refer to the receiver. yp, zp refer to the refraction point.

First the refraction points are calculated, which are needed to calculate the terms of the PSF. The PSF is built up A-scan per A-scan eventually giving the required B-scan with the same dimensions as the raw data. After normalising the PSF is returned as the result of the function.

A.2.2 3D implementation

In the 3D implementation the PSFs are calculated beforehand, because the 3D versions do take significant time to calculate. Psf 3D.m calculates the

PSFs. The main algorithm which loads the PSFs and raw data is comb deconv.m. Deconv perpsf.m performs the filtering operation and quality criteria calculation.

psf 3D.m

First the raw data is loaded so the dimensions for the PSF can be determined, after that the system vectors are loaded and aligned. This is followed by setting some radar properties (as antenna height of transmitter, z ground, and receiver, z Rx), some soil properties (ϵ_r and loss-tangent $tan(\delta)$) and finally the location of the point scatterer. This last one is determined by setting xo, yo and depth. To further clarify: xt, yt, zt refer to the transmitter. xa, ya, za refer to the receiver. xr, yr, zr refer to the refraction point. When the parameters are all set correctly the script can be run. Per A-scan it will calculate the refraction points and all coefficients of the first term of Equation 3.2. The system vector is shifted with the time-delay by using zero-padding (the minimum number of padded zeros of the entire C-scan is subtracted from this to maximise the information content). The two vectors are now convolved with eachother resulting in the final A-scan. The following step combines all the formed A-scans to a C-scan, which is then differentiated in the time-dimension. The resulting C-scan is interpolated with the function interp 3D.m after which it's normalised. The result is the interpolated PSFs with its' point-scatterer at location xo. The normalisation factor (manual norm) should also be saved as it's needed for weighting the deconvolution results.

deconv perpsf.m

This function performs the deconvolution operation and has the migrated C-scan as result. It uses the raw C-scan, the PSF and the desired Energy Ratio for inputs. First the raw data and the PSF are transformed to the frequency wavenumber domain using fft3.m (the inverse operation is performed with ifft3.m). After that a first guess SNR is used to determine the start value of the regularisation parameter. Then the data is filtered and the Energy Ratio is calculated, after which the regularisation parameter is altered until the actual Energy Ratio is close to the desired one. The final step is calculating the error.

comb deconv.m

First some pre-processing steps are taken such as weighting, compensation for time-drift and nulling. The desired Energy Ratio is then set after which the 13 deconvolution operations are performed. A complicated clear, load and save structure is needed to avoid memory problems. After the deconvolution the results are weighted with the manual norm of the corresponding

PSF. The shifts of the results (induced by the filter) are compensated with one command. The shifts over the scan-line and aperture are implemented as described in Chapter 3, the time-shift has been manually determined. The results are then combined (without overlap) over the aperture dimension and ASWEP is performed. Median filtering is the final step before the transition to the power-scale is done. The maximum of the result is determined so the give the final image a dynamic range of 20 dB.

interp 3D.m

This function performs the linear interpolation. The C-scan needs to be entered with the coordinate system [x,t,y]. The function results in the interpolation of the x-dimension and sampling of the t- and y-dimension.

Appendix B

MATLAB code of developed software

B.1 Diffraction stack

FocusedBoxExact.m

```
%this script will focus the data in a 3d BOX. The
dimensions are 1.0m
%(resolution 1cm) by 42 cm (resolution 1 cm) and 5 cm
depth (.55 cm to .60 m, resolution 0.3 cm)
clear all;
load lane1_pro_Tx465
t_orig = dt.*(1:length(Cpro(:,1,1)));
t = t_orig;
dy =dx;
y_orig = dy.*(1:length(Cpro(1,:,1)));
y=y_orig;

% Channel amplitude weights
coeffs=[0.1363 0.1584 0.1969 0.3528 0.5261 0.8970 0.9621 1
0.6389 0.3853 0.2089 0.1798 0.1340]
for p=1:13
Cpro(:,:,p) = Cpro(:,:,p)*1/coeffs(p);
end

%Offset vector
Tzero=[0.57 0.57 0.53 0.56 0.17 0.55 0.39 0.43 0.6 0.62
0.62 0.61 0.62]*1e-9;

%Compensate offset CHECKED RAW DATA HAS CORRECT TIME-SCALE
Tzero=[0.57 0.57 0.53 0.56 0.17 0.55 0.39 0.43 0.6 0.62
0.62 0.61 0.62]*1e-9;
for loopcounter=1:length(Cpro(1,1,:))
    B = squeeze(Cpro(:,:,loopcounter));
    offset = Tzero(loopcounter);  %offset in B-scan
    offset_index = round(offset/t(end)*length(t));
    Cpro(:,:,loopcounter) =[B(offset_index:end,:) ;
zeros(offset_index-1,length(y))];%Keep same size
end

%Null first part until 270
Cpro(1:270,:,:) = zeros(270,1025,13);

%Create the volume in where the data will be focused
z_start = .50;
z_end = .57;
dz=(z_start:0.003:z_end);
%dz = 0.52; %Extra image for Timofey
y_start = 0;
y_end = dy*length(Cpro(1,:,1));
dy=(y_start:0.01:y_end);

x_start = -.42;
x_end = .42;
dx=(x_start:0.01:x_end);

z_Rx = .265;
z_ground = .465
epsilon = 3.03;

Cmigr_exact = zeros(length(dz),length(dy),length(dx));

%Go through the box in the x-direction
for k=1:length(dx)
    dx(k)
    %Focus B-scans (one for each loop, and add them)
    Bmigr_final = zeros(length(dz),length(dy));
    for loop=1:length(Tzero)
         Bmigr_final = Bmigr_final +
Exact_tt2(Cpro(:,:,loop),dz,dy,dx(k),loop,Tzero(loop),t_or
ig,y_orig,z_Rx,z_ground,epsilon);
    end
    Cmigr_exact(:,:,k) = Bmigr_final;
end
```

Exact tt2.m

```
%This program determines the traveltimes to subsurface objects. It
%calculates the exact position of the refraction point on the ground.
Then
%it searches the corresponding element in the B-scan and migrates
it.

function Bmigr =
Exact_tt2(B,dz,dy,x_pos,loop,offset,t,y,z_Rx,z_ground,epsilon)

%'New x'
% dz = [.625];
% dy = [0:0.01:1];
% dx = [-.42];
% loop =7;
%offset = 1*10^-9;
xo = x_pos;
xt = 0;
zr = z_ground;
za = z_Rx;
if (loop == 7)
    xa = 0;
else
    xa = -0.42 + (loop-1)*0.07;
end

specialcounter1 = 0;
specialcounter2 = 0;
specialcounter3 = 0;

Bmigr = zeros(length(dz),length(dy));

    for z=1:length(dz) %all z
        zo = dz(z);
        for a=1:length(dy) %all antenna positions
            yt = dy(a);
            ya = dy(a);
              for l=1:length(dy)%all y
                  yo = dy(l);

                 %First the refraction point between Tx and the object is
                  %calculated
```

```
                    coeff = zeros(1,5);
                    if (xo == xt) % Object and transmitter in same x_plane
                        xr = xt;
                        coeff(1) = epsilon-1;
                        coeff(2) = 2*yo+2*yt-2*epsilon*yt-2*epsilon*yo;
                        coeff(3) = -zo^2+2*z_ground*zo-z_ground^2-yo^2-
4*yt*yo-
yt^2+epsilon*(z_ground^2+yt^2)+4*epsilon*yo*yt+epsilon*yo^2;
                        coeff(4) = 2*(zo^2-
2*z_ground*zo+z_ground^2+yo^2)*yt+2*yt^2*yo-
2*epsilon*yo*(z_ground^2+yt^2)-2*epsilon*yo^2*yt;
                        coeff(5) = epsilon*yo^2*(z_ground^2+yt^2)-(zo^2-
2*z_ground*zo+z_ground^2+yo^2)*yt^2;
                        solutions = roots(coeff);

                        %select the proper solution
                        for p=1:length(solutions)
                            if (abs(imag(solutions(p))) <= 0.001)
                                if (yo == yt)
                                    if((real(solutions(p))-yo) < 0.001)
                                        yr = yo;
                                    end
                                elseif (yo < yt)
                                    if(real(solutions(p)) -yo >= -.00001 &&
real(solutions(p)) -yt <= 0.0001)
                                        yr = real(solutions(p));
                                    end
                                else
                                    if(real(solutions(p)) -yt >= -.00001 &&
real(solutions(p)) -yo <= 0.0001)
                                        yr = real(solutions(p));
                                    end
                                end
                            end
                        end
                        refr_point = [xr yr];
                        Lta = (z_ground^2 + (yt -yr)^2)^.5;
                        Ltb = ((z_ground - zo)^2 + (yr - yo)^2)^.5;

                        traveltime_Tx = Lta/(3*10^8) +
Ltb*sqrt(epsilon)/(3*10^8);

                        %Object and Tx are in the same y-plane
                    elseif (yt == yo)
```

```
                  yr=yo;
                      coeff(1) = epsilon-1;
                      coeff(2) = 2*xo+2*xt-2*epsilon*xt-2*epsilon*xo;
                      coeff(3) = -zo^2+2*z_ground*zo-z_ground^2-xo^2-
4*xt*xo-
xt^2+epsilon*(z_ground^2+xt^2)+4*epsilon*xo*xt+epsilon*xo^2;
                      coeff(4) = 2*(zo^2-
2*z_ground*zo+z_ground^2+xo^2)*xt+2*xt^2*xo-
2*epsilon*xo*(z_ground^2+xt^2)-2*epsilon*xo^2*xt;
                      coeff(5) = epsilon*xo^2*(z_ground^2+xt^2)-(zo^2-
2*z_ground*zo+z_ground^2+xo^2)*xt^2;
                      solutions = roots(coeff);

                      %select the proper solution
                      for p=1:length(solutions)
                          if (abs(imag(solutions(p))) <= 0.001)
                                if (xo == xt)
                                    if((real(solutions(p))-yo) < 0.001)
                                        xr = xo;
                                    end
                                elseif (xo < xt)
                                    if(real(solutions(p)) -xo >= -.00001 &&
real(solutions(p)) -xt <= 0.0001)
                                        xr = real(solutions(p));
                                    end
                                else
                                    if(real(solutions(p)) -xt >= -.00001 &&
real(solutions(p)) - xo <= 0.0001)
                                        xr = real(solutions(p));
                                    end
                                end
                          end
                      end
                      refr_point = [xr yr];
                      Lta = (z_ground^2 + (xt -xr)^2)^.5;
                      Ltb = ((z_ground - zo)^2 + (xr - xo)^2)^.5;

                      traveltime_Tx = Lta/(3*10^8) +
Ltb*sqrt(epsilon)/(3*10^8);

                  else %Most general case target not in same x-plane or y-
plane
                      % yr = -(yt-yo)*xr/xo+yt is used to let the polynomial
                      % have only one variable (xr)
```

```
                    zr = z_ground;

                    coeff(1) = (1+(yt-yo)^2/xo^2)^3-(1+(yt-
yo)^2/xo^2)^3*epsilon;
                    coeff(2) = -2*(1+(yt-yo)^2/xo^2)^2*epsilon*(-xo-(yt-
yo)^2/xo)+(1+(yt-yo)^2/xo^2)^2*(-2*xo-2*(yt-yo)^2/xo);
                    coeff(3) = -zr^2*epsilon*(1+(yt-yo)^2/xo^2)^2-(1+(yt-
yo)^2/xo^2)*epsilon*(-xo-(yt-yo)^2/xo)^2+(1+(yt-
yo)^2/xo^2)^2*((zr-zo)^2+xo^2+(yt-yo)^2);
                    coeff(4) = -2*zr^2*epsilon*(-xo-(yt-yo)^2/xo)*(1+(yt-
yo)^2/xo^2);
                    coeff(5) = -zr^2*epsilon*(-xo-(yt-yo)^2/xo)^2;
                    solutions = roots(coeff);

                    %select the proper solution
                    for p=1:length(solutions)
                        if (abs(imag(solutions(p))) <= 0.001)
                            if (xo == xt)
                                if((real(solutions(p))-yo) < 0.001)
                                    xr = xo;
                                end
                            elseif (xo < xt)
                                if(real(solutions(p)) -xo >= -.00001 &&
real(solutions(p)) -xt <= 0.0001)
                                    xr = real(solutions(p));
                                end
                            else
                                if(real(solutions(p)) -xt >= -.00001 &&
real(solutions(p)) - xo <= 0.0001)
                                    xr = real(solutions(p));
                                end
                            end
                        end
                    end
                    yr = -(yt-yo)*xr/xo+yt;
                    refr_point = [xr yr];
                    Lta = sqrt(zr^2 + xr^2 + (yt-yr)^2);
                    Ltb = sqrt((zr-zo)^2 + (xr-xo)^2 + (yr-yo)^2);

                    traveltime_Tx = Lta/(3*10^8) +
Ltb*sqrt(epsilon)/(3*10^8);
                end
                %Now the traveltime from the object to the receiving
loop
                coeff_Rx = zeros(1,5);

               xa = round(100*xa); % This is done to eliminate a
rounding error.
               xo = round(100*xo);
               xa = xa/100;
               xo = xo/100;

                if (xo == xa) %Object in same x-plane as receiving loop
(Different implementation than in notebook!)
                    xr = xa;

                   coeff_Rx(1) = 1-epsilon;
                   coeff_Rx(2) = 2*epsilon*yo+2*epsilon*ya-2*ya-2*yo;
                   coeff_Rx(3) = -epsilon*yo^2-4*epsilon*yo*ya-
epsilon*((za)^2+ya^2)+ya^2+4*ya*yo+(zo-zr)^2+yo^2;
                   coeff_Rx(4) =
2*epsilon*yo^2*ya+2*epsilon*yo*((za)^2+ya^2)-2*ya^2*yo-
2*ya*((zo-zr)^2+yo^2);
                   coeff_Rx(5) = ya^2*((zo-zr)^2+yo^2)-
epsilon*yo^2*((za)^2+ya^2);
                   solutions_Rx = roots(coeff_Rx);

                    %select the proper solution
                    for p=1:length(solutions_Rx)
                        if (abs(imag(solutions_Rx(p))) <= 0.001)
                              if (yo == ya)
                                  if((real(solutions_Rx(p))-yo) < 0.001)
                                      yr_Rx = yo;
                                  end
                              elseif (yo < ya)
                                  if(real(solutions_Rx(p)) -yo >= -.00001 &&
real(solutions_Rx(p)) -ya <= 0.0001)
                                      yr_Rx = real(solutions_Rx(p));
                                  end
                              else
```

```
                                        if(real(solutions_Rx(p)) -ya >= -.00001 &&
real(solutions_Rx(p)) -yo <= 0.0001)
                                           yr_Rx = real(solutions_Rx(p));
                                        end
                                    end
                              end
                          end
%                           xr
%                           yr_Rx
                          specialcounter1 = specialcounter1 + 1;
                          Lra = sqrt((zr-za)^2 + (yr_Rx - ya)^2);
                          Lrb = sqrt((zr-zo)^2 + (yr_Rx - yo)^2);
                          traveltime_Rx = Lra/(3*10^8) +
Lrb*sqrt(epsilon)/(3*10^8);

                      elseif (yo == ya)  %Object in same y-plane as receiving
loop
                        yr = ya;
                          %Different structure in these equations the height from
                          %the antenna to the ground is expressed with za

                          coeff_Rx(1) = 1-epsilon;
                          coeff_Rx(2) = 2*epsilon*xo+2*epsilon*xa-2*xa-2*xo;
                          coeff_Rx(3) = -epsilon*xo^2-4*epsilon*xo*xa-
epsilon*((za)^2+xa^2)+xa^2+4*xa*xo+(zo-zr)^2+xo^2;
                          coeff_Rx(4) =
2*epsilon*xo^2*xa+2*epsilon*xo*((za)^2+xa^2)-2*xa^2*xo-
2*xa*((zo-zr)^2+xo^2);
                          coeff_Rx(5) = xa^2*((zo-zr)^2+xo^2)-
epsilon*xo^2*((za)^2+xa^2);
                          solutions_Rx = roots(coeff_Rx);

                             %select the proper solution
                          for p=1:length(solutions_Rx)
                              if (abs(imag(solutions_Rx(p))) <= 0.001)
                                    if (xo == xa)
                                        if((real(solutions_Rx(p))-xo) < 0.001)
                                            xr_Rx = xo;
                                        end
                                    elseif (xo < xa)
                                        if(real(solutions_Rx(p)) -xo >= -.00001 &&
real(solutions_Rx(p)) -xa <= 0.0001)
                                            xr_Rx = real(solutions_Rx(p));
```

```
                                      end
                                  else

                                      if(real(solutions_Rx(p)) - xa >= -.00001 &&
real(solutions_Rx(p)) - xo <= 0.0001)
                                         xr_Rx = real(solutions_Rx(p));
                                      end
                                  end
                            end
                        end
%                         xr_Rx
%                         yr

                        specialcounter2 = specialcounter2 + 1;
                        Lra = sqrt((zr-za)^2 + (xr_Rx - xa)^2);
                        Lrb = sqrt((zr-zo)^2 + (xr_Rx - xo)^2);
                        traveltime_Rx = Lra/(3*10^8) +
Lrb*sqrt(epsilon)/(3*10^8);

                    else %most general case. Object and receiving loop not in
the same plane.

                        coeff_Rx(1) = (1+(ya-yo)^2/(xa-xo)^2)^3-(1+(ya-
yo)^2/(xa-xo)^2)^3*epsilon;
                        coeff_Rx(2) = -(-2*xa-2*(ya-yo)^2*xa/(xa-
xo)^2)*epsilon*(1+(ya-yo)^2/(xa-xo)^2)^2-2*(1+(ya-yo)^2/(xa-
xo)^2)^2*epsilon*(-xo+(-(ya-yo)*xa/(xa-xo)+ya-yo)*(ya-yo)/(xa-
xo))+2*(-xa-(ya-yo)^2*xa/(xa-xo)^2)*(1+(ya-yo)^2/(xa-
xo)^2)^2+(1+(ya-yo)^2/(xa-xo)^2)^2*(-2*xo+2*(-(ya-yo)*xa/(xa-
xo)+ya-yo)*(ya-yo)/(xa-xo));
                        coeff_Rx(3) = -(xa^2+za^2+(ya-yo)^2*xa^2/(xa-
xo)^2)*epsilon*(1+(ya-yo)^2/(xa-xo)^2)^2-2*(-2*xa-2*(ya-
yo)^2*xa/(xa-xo)^2)*epsilon*(-xo+(-(ya-yo)*xa/(xa-xo)+ya-
yo)*(ya-yo)/(xa-xo))*(1+(ya-yo)^2/(xa-xo)^2)-(1+(ya-yo)^2/(xa-
xo)^2)*epsilon*(-xo+(-(ya-yo)*xa/(xa-xo)+ya-yo)*(ya-yo)/(xa-
xo))^2+(-xa-(ya-yo)^2*xa/(xa-xo)^2)^2*(1+(ya-yo)^2/(xa-
xo)^2)+2*(-xa-(ya-yo)^2*xa/(xa-xo)^2)*(1+(ya-yo)^2/(xa-xo)^2)*(-
2*xo+2*(-(ya-yo)*xa/(xa-xo)+ya-yo)*(ya-yo)/(xa-xo))+(1+(ya-
yo)^2/(xa-xo)^2)^2*(xo^2+(zr-zo)^2+(-(ya-yo)*xa/(xa-xo)+ya-
yo)^2);
                        coeff_Rx(4) = -2*(xa^2+za^2+(ya-yo)^2*xa^2/(xa-
xo)^2)*epsilon*(-xo+(-(ya-yo)*xa/(xa-xo)+ya-yo)*(ya-yo)/(xa-
xo))*(1+(ya-yo)^2/(xa-xo)^2)-(-2*xa-2*(ya-yo)^2*xa/(xa-
xo)^2)*epsilon*(-xo+(-(ya-yo)*xa/(xa-xo)+ya-yo)*(ya-yo)/(xa-
xo))^2+(-xa-(ya-yo)^2*xa/(xa-xo)^2)^2*(-2*xo+2*(-(ya-yo)*xa/(xa-
xo)+ya-yo)*(ya-yo)/(xa-xo))+2*(-xa-(ya-yo)^2*xa/(xa-
xo)^2)*(1+(ya-yo)^2/(xa-xo)^2)*(xo^2+(zr-zo)^2+(-(ya-yo)*xa/(xa-
xo)+ya-yo)^2);
                            coeff_Rx(5) = (-xa-(ya-yo)^2*xa/(xa-
xo)^2)^2*(xo^2+(zr-zo)^2+(-(ya-yo)*xa/(xa-xo)+ya-yo)^2)-
(xa^2+za^2+(ya-yo)^2*xa^2/(xa-xo)^2)*epsilon*(-xo+(-(ya-
yo)*xa/(xa-xo)+ya-yo)*(ya-yo)/(xa-xo))^2;

                            solutions_Rx = roots(coeff_Rx);
                            %select the proper solution
                            for p=1:length(solutions_Rx)
                                if (abs(imag(solutions_Rx(p))) <= 0.001)
                                      if (xo == xa)
                                          if((real(solutions_Rx(p))-xo) < 0.001)
                                              xr_Rx = xo;
                                          end
                                      elseif (xo < xa)
                                          if(real(solutions_Rx(p)) -xo >= -.00001 &&
real(solutions_Rx(p)) -xa <= 0.0001)
                                              xr_Rx = real(solutions_Rx(p));
                                          end
                                      else

                                          if(real(solutions_Rx(p)) - xa >= -.00001 &&
real(solutions_Rx(p)) - xo <= 0.0001)
                                             xr_Rx = real(solutions_Rx(p));
                                          end
                                      end
                                end
                            end

                            yr = (ya-yo)*(xr_Rx-xa)/(xa-xo)+ya;

                            specialcounter3 = specialcounter3 + 1;
                            Lra = sqrt((zr-za)^2 + (ya - yr)^2 + (xr_Rx - xa)^2);
                            Lrb = sqrt((zr-zo)^2 + (yr - yo)^2 + (xr_Rx - xo)^2);
                            traveltime_Rx = Lra/(3*10^8) +
Lrb*sqrt(epsilon)/(3*10^8);
                       end

                       traveltime_total = traveltime_Tx + traveltime_Rx;% +
offset; %Offset is already compensated
```

```
                %determine to which element in B this corresponds
                diff_t = t-traveltime_total;
                [k,i] = min(abs(diff_t));
                t_index = i;

                diff_y = y-dy(1);
                [k,i] = min(abs(diff_y));
                y_index = i;

                Bmigr(z,a) = Bmigr(z,a) + B(t_index,y_index);
            end
        end
    end
```

B.2 Deconvolution

B.2.1 2D implementation

deconv shift.m

```
%This script creates the point spread function for a sub-surface
scenario
%with the following geometry.

%z_Tx=0 - height of Transmitter (origin of coordinate system)
%z_Rx=0.27 m - position of Receiver with respect to Tx
%y - mechanical scan direction, [0, 60] cm

%z_target - .608 m m with respect to Tx
%Time of B-scan = 9.7752e-012s per step. 1024 steps gives
1.0010e-008 total
%time
%Mechanical scan direction 0.0024 m per step. 410 steps gives
.9840 m

clear all;
load MPdata50cm;
load test_data_subsurf;

loopnumber = 7;
B = squeeze(C(:,:,loopnumber));
epsilon
loss_tangent = 0.08 %loss tangent of soil
depth = 0.06; %depth of scatterer


t = dt*(1:length(B(:,1)));
t_orig = t;
y=dy*(1:length(B(1,:)));

%Compensate offset CHECKED RAW DATA HAS
CORRECT TIME-SCALE
offset = Tzero(loopnumber);  %offset in B-scan
offset_index = round(offset/t(end)*length(t));
B=[B(offset_index:end,:); zeros(offset_index-
1,length(y))];%Keep same size


h1 = channels(:,loopnumber); %don't normalize because
```

```
excitation voltage information is in there as well
h_short = h1(1:2:length(h1)); %adapt channel information to
new situation
ht = [h_short(350:end)];



 %THE ALGORITH WORKS AS %WELL WITH REALLY
LONG B-SCANS. ALL OBJECTS ARE IMAGED
PROPERLY. THE MIDDLE
%ARTEFACT DECREASES WITH B-SCAN LENGT, BUT
IS ALMOST ALWAYS TO BE SEEN

point_spread_function =
psf_exact(t,y,z_ground,ht,epsilon,z_Rx,loss_tangent,depth);
point_spread_function = [zeros(1,410);
diff(point_spread_function)];


%fft of raw data and psf,
f_psf = fft2(point_spread_function);
f_raw = fft2(B);

SNR_db = 36;
Energy_ratio = 0;
while (Energy_ratio < 100) %This loop will change the
regularization parameter until the energy ratio is too big.

    SNR = 10^(SNR_db/20);
    %wiener filter
    wif = conj(f_psf)./((conj(f_psf).*f_psf)+ 1/SNR);
    f3= wif.*f_raw;
    %result back to time domain
    s1=real(ifft2(f3));

    %Calculate 'Quality parameters'
    Energy_ratio = norm(s1,2)/norm(B,2)*100;
    % Recontstructing the data for the deconvolved result
    rData=real(ifft2(f3.*f_psf));
    % Difference between the original and reconstructed data as
error
    err=norm(B - rData,'fro')/(norm(B,'fro') +
norm(rData,'fro'))*100;
    SNR_db = SNR_db + 1;
```

```
end
SNR_db
Energy_ratio
err

orig_result =s1;
%Compensate the side-way shift introduced by the
deconvolution. It's fixed
%(always half of the mechanical scan direction)
temp = s1(:,1:round(length(y)/2));
temp2= s1(:,round(length(y)/2)+1:end);
s1 = [temp2 temp];

%use automatic shift compensation with circular shift!! Find
time-index of maximum in raw
%data. Shift maximum of deconvolution result to that time-
index. The maxima
%should be on the same location

raw_max = max(max(abs(B))); %location of maximum of raw
data
[q,max_raw_index_y] = min(min(abs(abs(B)-raw_max))); %to
find y index
[q,max_raw_index_t] = min(min(abs(abs(B)-
raw_max),[],2));%to find t index
column_with_maximum = max_raw_index_y;

raw_res = max(max(abs(s1(:,max_raw_index_y)))); %location
of maximum of deconvolved result,
%search area confined to column where maximum of raw data
is located.
%This is to ensure that the same objects are compared.
[q,max_index_res] = min(min(abs(abs(s1)-raw_res),[],2));

diff_index = max_raw_index_t - max_index_res %the shift is
now known

if diff_index >= 0 %Perform circular shift
    s2 = [s1(end-diff_index:end,:); s1(1:end-diff_index-1,:)];
else
    diff_index_comp = length(t) + diff_index %make the
negative shift positive by using the circularity
    s2 = [s1(end-diff_index_comp:end,:); s1(1:end-
diff_index_comp-1,:)];
```

```
end
close all;
figure;
imagesc(y,t,B),title(strcat(['B-scan with compensation for '
num2str(offset) ' s offset'])),ylabel('t [s]'),xlabel('y
[m]'),colorbar

figure;
imagesc(y,t,point_spread_function),title('Point spread
function'),ylabel('t [s]'),xlabel('y [m]'),colorbar

orig_result_db = 20*log10(abs(orig_result)+.001);
s1_db = 20*log10(abs(s1)+.001);
max_orig = max(max(orig_result_db));
max_s1 = max(max(s1_db));

figure;
subplot(1,2,1);imagesc(y,t,orig_result_db,[max_orig-10
max_orig]),title('Result without side-shift compensation')
subplot(1,2,2);imagesc(y,t,s1_db,[max_s1-10
max_s1]),title('Result with side-shift compensation')


%For the depth axis we start counting from 2.6 ns (.785 m),
after that only
%subsurface reflections will be measured and the traveltime
and depth will have a
%linear dependency. The first 2.6 ns of the deconvolved image
is cut off and
%a proper depth axes is constructed
%
  time_shift = (z_ground + (z_ground-z_Rx))/3e8;
  shift_index = round((time_shift/t(end))*length(t));

  s3 = [s2(shift_index:end,:)];

  t_compensated = ((t(end))/length(t))*(0:length(s3(:,1))-1);

  max_s3 = max(max(20*log10(abs(s3))));
  depth_of_image =
(t_compensated(end))*3e8/(sqrt(epsilon)*2);
  depth_axes = (0:depth_of_image/(length(s3(:,1))-
1):depth_of_image);
```

```
 figure;
imagesc(y,100*depth_axes,20*log10(abs(s3)+.0001),[max_s3-
10 max_s3]),ylabel('z [cm]'),xlabel('y
[m]'),title(strcat(['Deconvolved image (only subsurface) in
[dB]'])),colorbar
```

psf exact.m

```
%This function creates the point-spread function. One scatterer is
placed
%at the center. For the refraction point calculation the
approximation
%method is used. Then the left and right side are shifted. R
coordinates
%are of refraction point. O coordinates are of object in soil
(relative to ground). R are for
%the receiving loop. P are for the refraction point. T for the
%transmitter
function B_scan =
psf_exact(t,y,z_ground,ht,epsilon,zr,loss_tangent,depth)

c = 3e8;
mu = 4*pi*10^-7;
e_0 = 1/(c^2*mu);


zt = 0; %the coordinate system starts at the tranmitter.
zp = z_ground;
zo = z_ground + depth; %estimated depth of object (subsurface)
yo = y(end)/2; %scatterer in center

%calculate minimum number of added zeros (see compensation
for meaningless
%zeros further on)
min_td = (z_ground + (z_ground-zr))/c + 2*(zo-
z_ground)*sqrt(epsilon)/c;
%the minimum number of added zeros is when the antenna and
object are directly above eachother.
min_zeros = round(min_td/(t(end))*length(t));


%make the psf the same size as raw data
B_scan = zeros(length(t),length(y));
res = zeros(2,length(y));
trans = zeros(2,length(y));

for k=1:length(y) %all the antenna positions. At every one
position one A scan is calculated.
               clear yp yp2;
                yt = y(k);
                %Calculate refraction point with the exact method
                %First the refraction point between transmitter and
target

                if yt == yo
                    yp = yt;
                    yp2 = yt;
                else
                     coeff = zeros(1,5);
                     coeff(1) = 1-epsilon;
                     coeff(2) = 2*yt*epsilon+2*epsilon*yo-2*yt-2*yo;
                     coeff(3) = -(yt^2+zp^2)*epsilon-4*yt*epsilon*yo-
epsilon*yo^2+yt^2+4*yt*yo+yo^2+(zo-zp)^2;
                     coeff(4) =
(2*(yt^2+zp^2))*epsilon*yo+2*yt*epsilon*yo^2-2*yt^2*yo-
2*yt*(yo^2+(zo-zp)^2);
                     coeff(5) = yt^2*(yo^2+(zo-zp)^2)-
(yt^2+zp^2)*epsilon*yo^2;

                     solutions = roots(coeff);
                     for p=1:length(solutions)
                          if (abs(imag(solutions(p))) <= 0.01)
                             if(real(solutions(p)) -yo >= -.00001 &&
real(solutions(p)) -yt <= 0.0001)
                                 yp = real(solutions(p));
                             end
                             if(real(solutions(p)) -yt >= -.00001 &&
real(solutions(p)) -yo <= 0.0001)
                                 yp = real(solutions(p));
                             end
                          end
                     end

                     %Now the second refraction point is calculated
                     coeff2 = zeros(1,5);
                     coeff2(1) = 1-epsilon;
                     coeff2(2) = 2*yt*epsilon+2*epsilon*yo-2*yt-2*yo;
                     coeff2(3) = -epsilon*yo^2-4*yt*epsilon*yo-
epsilon*(yt^2+(zp-zr)^2)+yt^2+4*yt*yo+yo^2+(zo-zp)^2;
                     coeff2(4) =
2*yt*epsilon*yo^2+2*epsilon*yo*(yt^2+(zp-zr)^2)-2*yt^2*yo-
2*yt*(yo^2+(zo-zp)^2);
                     coeff2(5) = yt^2*(yo^2+(zo-zp)^2)-
```

```
epsilon*yo^2*(yt^2+(zp-zr)^2);

                    solutions2 =roots(coeff2);
                    for l=1:length(solutions2)
                         if (abs(imag(solutions2(l))) <= 0.01)
                            if(real(solutions2(l)) -yo >= -.00001 &&
real(solutions2(l)) -yt <= 0.0001)
                                yp2 = real(solutions2(l));
                            end
                            if(real(solutions2(l)) -yt >= -.00001 &&
real(solutions2(l)) -yo <= 0.0001)
                                yp2 = real(solutions2(l));
                            end
                         end
                    end
               end

               phi1tx = atan((yt-yp)/zp);
               phi2tx = atan((yo-yp)/(zo-zp));
               res(1,k) = (sin(phi1tx)/sin(phi2tx))^2; %FOLLOWS
SNELL'S LAW
               %RESULT = EPSILON

               phi1rx = atan((yt-yp2)/(zp-zr));
               phi2rx = atan((yo-yp2)/(zo-zp));
               res(2,k) = (sin(phi1rx)/sin(phi2rx))^2; %FOLLOWS
SNELL'S LAW
               %RESULT = EPSILON
               %Calculate first term of 7.57 / 7.59
               %Calculate refraction point with the approximation
presented
               %by Bart Scheers in his doctoral thesis.
%                y1 = (yt - yo)/(zo + z_ground)*zo + yo;
%                y_rp = yo + (1/sqrt(epsilon))*(y1 - yo)

               Rt_fs = sqrt(z_ground^2 + (yp -yt)^2);
               Rr_fs = sqrt((z_ground-zr)^2 + (yp2 -yt)^2);
               Rt_ss = sqrt((zo-zp)^2 + (yo-yp)^2);
               Rr_ss = sqrt((zo-zp)^2 + (yo-yp2)^2);
               Rt = Rt_fs + Rt_ss;
               Rr = Rr_fs + Rr_ss;

               %compensate for time shift [1024 steps = 1.01e-8 s ]
               td = ((Rt_fs + Rr_fs)/c) + ((Rt_ss +
Rr_ss)*sqrt(epsilon)/c); %time-delay in seconds
               td_index = round(td/(t(end))*length(t)); %time-delay in
index

               %we delete as many zeros as possible. This will
increase the
               %information in the psf (since the size of the psf is
fixed
               td_index = td_index - min_zeros;

               h_comp = [zeros(td_index,1) ; ht]; %the 300 shifts the
function up to t = 0 and improves the image

               %create impulse response of the ground
               d = Rt_ss + Rr_ss;
               T_ag = 2*cos(phi1tx)/(cos(phi1tx) +
sqrt(epsilon)*cos(phi2tx));
               T_ga = 2*sqrt(epsilon)*cos(phi2rx)/(cos(phi1rx) +
sqrt(epsilon)*cos(phi2rx));
               trans(1,k)= T_ag;
               trans(2,k)= T_ga;

               gd = (d*sqrt(mu*epsilon*e_0)*loss_tangent/2)./(pi*((t-
d*sqrt(mu*epsilon*e_0)).^2 +
(d*sqrt(mu*epsilon*e_0)*loss_tangent/2)^2));
               %gd = gd/max(gd);%normalize
               %Shift gd to actual position
               %fs_shift = round(((Rt_fs +
Rr_fs)/c)/(t(end))*length(t));
               %THE SHIFT WORKS FINE, BUT IT DECREASED
THE RESULT. THE
               %ERROR IS LESS, BUT THAT IS BECAUSE THE
LARGER PART OF THE
               %INFORMATION OF THE PSF IS NOW LOST. THE
IMAGE DETERIORATES.
               %gd = [zeros(1,fs_shift)  gd(1:end-fs_shift)];

%                 if k == length(y)/2
%                     figure;
%                     plot(t,gd),title(['Impulse response of the ground,
epsilon = ' num2str(epsilon) ', tan(delta) = '
num2str(loss_tangent)]),xlabel('t [s]')
%                 end
               First_term =
T_ag*T_ga*gd/(8*pi^2*Rt*Rr*(c/sqrt(epsilon)));

               if length(h_comp) > length(t)
                   h_comp = h_comp(1:length(t));
               else
                   h_comp = [h_comp ; zeros(length(t)-
length(h_comp),1)];
               end
               A_scan =conv(First_term,h_comp); %calculate A-scan
               A_scan = A_scan(1:length(t));
               B_scan(1:length(A_scan),k) =
B_scan(1:length(A_scan),k) + A_scan; %form B-scan
end
%res
%trans;
%    figure;
%    subplot(1,2,1), plot(y,trans(1,:)),xlabel('scan-line
[m]'),title('Transmission coefficient for air to ground interface')
%    subplot(1,2,2), plot(y,trans(2,:)),xlabel('scan-line
[m]'),title('Transmission coefficient for ground to air interface')

%shift the point spread function (reverse the left and right side)
% temp = B_scan(:,1:round(length(y)/2));
% temp2= B_scan(:,round(length(y)/2)+1:end);
% B_scan = [temp2 temp];
%point
%Take frobenius norm of matrix
B_scan = B_scan/norm(B_scan,2);
```

B.2.2 3D implementation

psf 3D.m

```
%function C_scan = psf_3D(z_ground,zr,)
clear all
load MPdata50cm
load 4targets_prepro_Tx525mm

C = Cpro;
dy =dx;
z_ground =.525;
z_Rx = .265
epsilon =3.03

loss_tangent = 0.08;
c = 3e8;
mu = 4*pi*10^-7;
e_0 = 1/(c^2*mu);

B = squeeze(C(:,:,7));
y=dy*(1:length(B(1,:)));
t = dt*(1:length(B(:,1)));

phis=zeros(2);
eps = [];
trans = zeros(2,length(y));
res = zeros(2,length(y));
%
% figure;
% plot(channels)
%get all the maxima of the system vector at the same position,
because the
%time-delay is already compensated for. The maximum of the
middle loop channel reponse is chosen to be
%point on which all the maxima should occur.
[max_center index_m_center] = max(channels(:,7));
for r=1:length(channels(1,:))
    [max_loop index_m_loop] = max(channels(:,r));
    diff_max = index_m_center - index_m_loop;

    if diff_max >= 0 %Perform circular shift
        channels(:,r) = [channels(end-diff_max:end,r);
channels(1:end-diff_max-1,r)];
```

```
    else
        diff_max_comp = length(channels(:,r)) + diff_max; %make
the negative shift positive by using the circularity
        channels(:,r) = [channels(end-diff_max_comp:end,r);
channels(1:end-diff_max_comp-1,r)];
    end
end
% figure;
% plot(channels)




%make the psf the same size as raw data
B_scan = zeros(length(t),length(y));
C_scan = zeros(13,length(t),length(y));

%DATA FOR CALCULATION REFRECTION POINT
zr = .525; %refraction point location is on ground
%object data (scatterer put in center of grid at depth of 6 cm.)
depth = 0.06;
yo = y(end)/2;
xo = 0
zo = z_ground + depth;
%Tx data
xt = 0;
zt = 0;
%Rx data
za = zr-.265; %z_Rx = .265

%calculate minimum number of added zeros (see compensation
for meaningless
%zeros further on)
min_td = (z_ground + (z_ground-z_Rx))/c +
2*(depth)*sqrt(epsilon)/c;
%the minimum number of added zeros is when the antenna and
object are
%directly above eachother (y and x are the same). This is true for
the
%middle loop when the antenna is at half the mechanical scan
direction
min_zeros = round(min_td/(t(end))*length(t));
norm_of_all = [];
```

```
%Here we start the nested loop for creating the A-scans. First
loop number
%1 etc.
for k=1:length(channels(1,:))%all loops
    %Rx data varies per loop
    if (k == 7)
        xa = 0;
    else
        xa = -0.42 + (k-1)*0.07;
    end
    %the system vector varies per loop.
    h1 = channels(:,k); %don't normalize because excitation
voltage information is in there as well
    h_short = h1(1:2:length(h1)); %adapt channel information to
new situation where sample rate is halved.
    ht = [h_short(350:end)]; %CORRECTION IS CONSTANT
AND PEAKS ALIGN.

    %Here per loop the mechanical scan direction is walked
through.
    %Resulting in all the A-scans.
    for l=1:length(y)
        ya = y(l);
        yt = y(l);

                %First the refraction point between Tx and the object
is
                %calculated
                'Refraction point between Tx and Object';
                coeff = zeros(1,5);
                if (xo == xt)
                    xr = xt;
                    coeff(1) = epsilon-1;
                    coeff(2) = 2*yo+2*yt-2*epsilon*yt-2*epsilon*yo;
                    coeff(3) = -zo^2+2*z_ground*zo-z_ground^2-
yo^2-4*yt*yo-
yt^2+epsilon*(z_ground^2+yt^2)+4*epsilon*yo*yt+epsilon*yo^
2;
                    coeff(4) = 2*(zo^2-
2*z_ground*zo+z_ground^2+yo^2)*yt+2*yt^2*yo-
2*epsilon*yo*(z_ground^2+yt^2)-2*epsilon*yo^2*yt;
                    coeff(5) = epsilon*yo^2*(z_ground^2+yt^2)-(zo^2-
2*z_ground*zo+z_ground^2+yo^2)*yt^2;
                    solutions = roots(coeff);
```

```
                         %select the proper solution
                         for p=1:length(solutions)
                             if (abs(imag(solutions(p))) <= 0.001)
                                 if (yo == yt)
                                     if((real(solutions(p))-yo) < 0.001)
                                         yr = yo;
                                     end
                                 elseif (yo < yt)
                                     if(real(solutions(p)) -yo >= -.00001 &&
real(solutions(p)) -yt <= 0.0001)
                                         yr = real(solutions(p));
                                     end
                                 else
                                     if(real(solutions(p)) -yt >= -.00001 &&
real(solutions(p)) -yo <= 0.0001)
                                         yr = real(solutions(p));
                                     end
                                 end
                             end
                         end
%                           'antenna position'
%                          [xt yt zt]
%                          'object position'
%                          [xo yo zo]
                         refr_point_Tx = [xr yr zr];


                         %Object and Tx are in the same y-plane
                     elseif (yt == yo)
                         yr=yo;
                         coeff(1) = epsilon-1;
                         coeff(2) = 2*xo+2*xt-2*epsilon*xt-2*epsilon*xo;
                         coeff(3) = -zo^2+2*z_ground*zo-z_ground^2-
xo^2-4*xt*xo-
xt^2+epsilon*(z_ground^2+xt^2)+4*epsilon*xo*xt+epsilon*xo^
2;
                         coeff(4) = 2*(zo^2-
2*z_ground*zo+z_ground^2+xo^2)*xt+2*xt^2*xo-
2*epsilon*xo*(z_ground^2+xt^2)-2*epsilon*xo^2*xt;
                         coeff(5) = epsilon*xo^2*(z_ground^2+xt^2)-(zo^2-
2*z_ground*zo+z_ground^2+xo^2)*xt^2;
                         solutions = roots(coeff);
```

```
                        %select the proper solution
                        for p=1:length(solutions)
                            if (abs(imag(solutions(p))) <= 0.001)
                                  if (xo == xt)
                                      if((real(solutions(p))-yo) < 0.001)
                                          xr = xo;
                                      end
                                  elseif (xo < xt)
                                      if(real(solutions(p)) -xo >= -.00001 &&
real(solutions(p)) -xt <= 0.0001)
                                          xr = real(solutions(p));
                                      end
                                  else
                                      if(real(solutions(p)) -xt >= -.00001 &&
real(solutions(p)) - xo <= 0.0001)
                                          xr = real(solutions(p));
                                      end
                                  end
                            end
                        end
%                         'antenna position'
%                         [xt yt zt]
%                         'object position'
%                         [xo yo zo]
                        refr_point_Tx = [xr yr zr];


                    else %Most general case target not in same x-plane or
y-plane
                        % yr = -(yt-yo)*xr/xo+yt is used to let the
polynomial
                        % have only one variable (xr)
                        %zr = z_ground;

                        coeff(1) = (1+(yt-yo)^2/xo^2)^3-(1+(yt-
yo)^2/xo^2)^3*epsilon;
                        coeff(2) = -2*(1+(yt-yo)^2/xo^2)^2*epsilon*(-xo-
(yt-yo)^2/xo)+(1+(yt-yo)^2/xo^2)^2*(-2*xo-2*(yt-yo)^2/xo);
                        coeff(3) = -zr^2*epsilon*(1+(yt-yo)^2/xo^2)^2-
(1+(yt-yo)^2/xo^2)*epsilon*(-xo-(yt-yo)^2/xo)^2+(1+(yt-
yo)^2/xo^2)^2*((zr-zo)^2+xo^2+(yt-yo)^2);
                        coeff(4) = -2*zr^2*epsilon*(-xo-(yt-
yo)^2/xo)*(1+(yt-yo)^2/xo^2);
                        coeff(5) = -zr^2*epsilon*(-xo-(yt-yo)^2/xo)^2;
                    solutions = roots(coeff);

                    %select the proper solution
                    for p=1:length(solutions)
                        if (abs(imag(solutions(p))) <= 0.001)
                              if (xo == xt)
                                  if((real(solutions(p))-yo) < 0.001)
                                      xr = xo;
                                  end
                              elseif (xo < xt)
                                  if(real(solutions(p)) -xo >= -.00001 &&
real(solutions(p)) -xt <= 0.0001)
                                      xr = real(solutions(p));
                                  end
                              else
                                  if(real(solutions(p)) -xt >= -.00001 &&
real(solutions(p)) - xo <= 0.0001)
                                      xr = real(solutions(p));
                                  end
                              end
                        end
                    end
                    yr = -(yt-yo)*xr/xo+yt;
%                     'antenna position'
%                     [xt yt zt]
%                     'object position'
%                     [xo yo zo]
                    refr_point_Tx = [xr yr zr];

                end


                %Now the refraction point from the object to the
receiving loop
                coeff_Rx = zeros(1,5);
                'Refraction point between Object and Rx';

               xa = round(100*xa); % This is done to eliminate a
rounding error.
               xo = round(100*xo);
               xa = xa/100;
               xo = xo/100;

                if (xa == xo) %Object in same x-plane as receiving
```

```
loop (Different implementation than in notebook!)
                     xr = xa;

                     coeff_Rx(1) = 1-epsilon;
                     coeff_Rx(2) = 2*epsilon*yo+2*epsilon*ya-2*ya-
2*yo;
                     coeff_Rx(3) = -epsilon*yo^2-4*epsilon*yo*ya-
epsilon*((za)^2+ya^2)+ya^2+4*ya*yo+(zo-zr)^2+yo^2;
                     coeff_Rx(4) =
2*epsilon*yo^2*ya+2*epsilon*yo*((za)^2+ya^2)-2*ya^2*yo-
2*ya*((zo-zr)^2+yo^2);
                     coeff_Rx(5) = ya^2*((zo-zr)^2+yo^2)-
epsilon*yo^2*((za)^2+ya^2);
                     solutions_Rx = roots(coeff_Rx);

                      %select the proper solution
                      for p=1:length(solutions_Rx)
                          if (abs(imag(solutions_Rx(p))) <= 0.001)
                                if (yo == ya)
                                    if((real(solutions_Rx(p))-yo) < 0.001)
                                        yr_Rx = yo;
                                    end
                                elseif (yo < ya)
                                    if(real(solutions_Rx(p)) -yo >= -.00001
&& real(solutions_Rx(p)) -ya <= 0.0001)
                                        yr_Rx = real(solutions_Rx(p));
                                    end
                                else

                                    if(real(solutions_Rx(p)) -ya >= -.00001
&& real(solutions_Rx(p)) -yo <= 0.0001)
                                       yr_Rx = real(solutions_Rx(p));
                                    end
                                end
                          end
                      end
%                       xr
%                       yr_Rx
%                       'antenna position'
%                       [xa ya .265]
%                       'object position'
%                       [xo yo zo]

                      refr_point_Rx = [xr yr_Rx zr];
```

```
                  elseif (yo == ya)  %Object in same y-plane as
receiving loop
                      yr = ya;
                      %Different structure in these equations the height
from
                      %the antenna to the ground is expressed

                      coeff_Rx(1) = 1-epsilon;
                      coeff_Rx(2) = 2*epsilon*xo+2*epsilon*xa-2*xa-
2*xo;
                      coeff_Rx(3) = -epsilon*xo^2-4*epsilon*xo*xa-
epsilon*((za)^2+xa^2)+xa^2+4*xa*xo+(zo-zr)^2+xo^2;
                      coeff_Rx(4) =
2*epsilon*xo^2*xa+2*epsilon*xo*((za)^2+xa^2)-2*xa^2*xo-
2*xa*((zo-zr)^2+xo^2);
                      coeff_Rx(5) = xa^2*((zo-zr)^2+xo^2)-
epsilon*xo^2*((za)^2+xa^2);
                      solutions_Rx = roots(coeff_Rx);

                         %select the proper solution
                      for p=1:length(solutions_Rx)
                          if (abs(imag(solutions_Rx(p))) <= 0.001)
                                if (xo == xa)
                                    if((real(solutions_Rx(p))-xo) < 0.001)
                                        xr_Rx = xo;
                                    end
                                elseif (xo < xa)
                                    if(real(solutions_Rx(p)) -xo >= -.00001
&& real(solutions_Rx(p)) -xa <= 0.0001)
                                        xr_Rx = real(solutions_Rx(p));
                                    end
                                else

                                    if(real(solutions_Rx(p)) - xa >= -.00001
&& real(solutions_Rx(p)) - xo <= 0.0001)
                                       xr_Rx = real(solutions_Rx(p));
                                    end
                                end
                          end
                      end
%                       xr_Rx
%                       yr
%                       'antenna position'
%                              [xa ya .265]
%                              'object position'
%                              [xo yo zo]
                            refr_point_Rx = [xr_Rx yr zr];


                       else %most general case. Object and receiving loop
not in the same plane.

                            coeff_Rx(1) = (1+(ya-yo)^2/(xa-xo)^2)^3-(1+(ya-
yo)^2/(xa-xo)^2)^3*epsilon;
                            coeff_Rx(2) = -(-2*xa-2*(ya-yo)^2*xa/(xa-
xo)^2)*epsilon*(1+(ya-yo)^2/(xa-xo)^2)^2-2*(1+(ya-yo)^2/(xa-
xo)^2)^2*epsilon*(-xo+(-(ya-yo)*xa/(xa-xo)+ya-yo)*(ya-
yo)/(xa-xo))+2*(-xa-(ya-yo)^2*xa/(xa-xo)^2)*(1+(ya-yo)^2/(xa-
xo)^2)^2+(1+(ya-yo)^2/(xa-xo)^2)^2*(-2*xo+2*(-(ya-
yo)*xa/(xa-xo)+ya-yo)*(ya-yo)/(xa-xo));
                            coeff_Rx(3) = -(xa^2+za^2+(ya-yo)^2*xa^2/(xa-
xo)^2)*epsilon*(1+(ya-yo)^2/(xa-xo)^2)^2-2*(-2*xa-2*(ya-
yo)^2*xa/(xa-xo)^2)*epsilon*(-xo+(-(ya-yo)*xa/(xa-xo)+ya-
yo)*(ya-yo)/(xa-xo))*(1+(ya-yo)^2/(xa-xo)^2)-(1+(ya-yo)^2/(xa-
xo)^2)*epsilon*(-xo+(-(ya-yo)*xa/(xa-xo)+ya-yo)*(ya-yo)/(xa-
xo))^2+(-xa-(ya-yo)^2*xa/(xa-xo)^2)^2*(1+(ya-yo)^2/(xa-
xo)^2)+2*(-xa-(ya-yo)^2*xa/(xa-xo)^2)*(1+(ya-yo)^2/(xa-
xo)^2)*(-2*xo+2*(-(ya-yo)*xa/(xa-xo)+ya-yo)*(ya-yo)/(xa-
xo))+(1+(ya-yo)^2/(xa-xo)^2)^2*(xo^2+(zr-zo)^2+(-(ya-
yo)*xa/(xa-xo)+ya-yo)^2);
                            coeff_Rx(4) = -2*(xa^2+za^2+(ya-yo)^2*xa^2/(xa-
xo)^2)*epsilon*(-xo+(-(ya-yo)*xa/(xa-xo)+ya-yo)*(ya-yo)/(xa-
xo))*(1+(ya-yo)^2/(xa-xo)^2)-(-2*xa-2*(ya-yo)^2*xa/(xa-
xo)^2)*epsilon*(-xo+(-(ya-yo)*xa/(xa-xo)+ya-yo)*(ya-yo)/(xa-
xo))^2+(-xa-(ya-yo)^2*xa/(xa-xo)^2)^2*(-2*xo+2*(-(ya-
yo)*xa/(xa-xo)+ya-yo)*(ya-yo)/(xa-xo))+2*(-xa-(ya-
yo)^2*xa/(xa-xo)^2)*(1+(ya-yo)^2/(xa-xo)^2)*(xo^2+(zr-
zo)^2+(-(ya-yo)*xa/(xa-xo)+ya-yo)^2);
                            coeff_Rx(5) = (-xa-(ya-yo)^2*xa/(xa-
xo)^2)^2*(xo^2+(zr-zo)^2+(-(ya-yo)*xa/(xa-xo)+ya-yo)^2)-
(xa^2+za^2+(ya-yo)^2*xa^2/(xa-xo)^2)*epsilon*(-xo+(-(ya-
yo)*xa/(xa-xo)+ya-yo)*(ya-yo)/(xa-xo))^2;

                            solutions_Rx = roots(coeff_Rx);
                            %select the proper solution
                            for p=1:length(solutions_Rx)
                                if (abs(imag(solutions_Rx(p))) <= 0.001)
                                if (xo == xa)
                                    if((real(solutions_Rx(p))-xo) < 0.001)
                                        xr_Rx = xo;
                                    end
                                elseif (xo < xa)
                                    if(real(solutions_Rx(p)) -xo >= -.00001
&& real(solutions_Rx(p)) -xa <= 0.0001)
                                        xr_Rx = real(solutions_Rx(p));
                                    end
                                else

                                    if(real(solutions_Rx(p)) - xa >= -.00001
&& real(solutions_Rx(p)) - xo <= 0.0001)
                                        xr_Rx = real(solutions_Rx(p));
                                    end
                                end
                          end
                    end

                    yr = (ya-yo)*(xr_Rx-xa)/(xa-xo)+ya;

%                     'antenna position'
%                     [xa ya .265]
%                     'object position'
%                     [xo yo zo]

                    refr_point_Rx = [xr_Rx yr zr];
                end

                %Calculation of snells law of refraction point Tx
                dist_Tx_object = sqrt((xo-xt)^2 + (yo-yt)^2); %on the
ground
                dist_Rp_object = sqrt((xo-refr_point_Tx(1))^2 +
(refr_point_Tx(2) - yo)^2);
                dist_Tx_Rp = dist_Tx_object - dist_Rp_object;
                phis(1,1) = atan(dist_Tx_Rp/zr);
                phis(1,2) = atan(dist_Rp_object/(zo-zr));

                if (sin(phis(1,1)) == 0 && sin(phis(1,2)) == 0)
                    'Epsilon from 1st refraction point can not be
determined from angles, because phi1 = phi2 = 0 degrees';
                else
                    eps = [eps (sin(phis(1,1))/sin(phis(1,2)))^2];
                end
```

```
                  %calculation of snells law of refraction point Rx
                  dist_Rx_object = sqrt((xo-xa)^2 + (yo-ya)^2); %on
the ground
                  dist_Rp_object2 = sqrt((xo-refr_point_Rx(1))^2 +
(refr_point_Rx(2) - yo)^2);
                  dist_Rx_Rp = dist_Rx_object - dist_Rp_object2;
                  phis(2,1) = atan(dist_Rx_Rp/(za));
                  phis(2,2) = atan(dist_Rp_object2/(zo-zr));

                  if (sin(phis(2,1)) == 0 && sin(phis(2,2)) == 0)
                      'Epsilon from 2nd refraction point can not be
determined from angles, because phi1 = phi2 = 0 degrees';
                  else
                      eps = [eps (sin(phis(2,1))/sin(phis(2,2)))^2];
                  end

                  %Now the refraction points have been calculated and
                  %verified
                  xp1 = refr_point_Tx(1);
                  xp2 = refr_point_Rx(1);
                  yp1 = refr_point_Tx(2);
                  yp2 = refr_point_Rx(2);

                  %calculate distances (again). This is also done
                  %previously but this is done for easy of programming
rest
                  %of psf
                  Rt_fs = sqrt((xt-xp1)^2 + (yt-yp1)^2 + (z_ground)^2);
                  Rr_fs = sqrt((xa-xp2)^2 + (ya-yp2)^2 + (za)^2);
                  Rt_ss = sqrt((xo-xp1)^2 + (yo-yp1)^2 + (depth)^2);
                  Rr_ss = sqrt((xo-xp2)^2 + (yo-yp2)^2 + (depth)^2);
                  Rt = Rt_fs + Rt_ss;
                  Rr = Rr_fs + Rr_ss;

                  phi1tx = phis(1,1);
                  phi2tx = phis(1,2);
                  phi1rx = phis(2,1);
                  phi2rx = phis(2,2);

                  %compensate for time shift [1024 steps = 1.01e-8 s ]
                  td = ((Rt_fs + Rr_fs)/c) + ((Rt_ss +
Rr_ss)*sqrt(epsilon)/c); %time-delay in seconds
                  td_index = round(td/(t(end))*length(t)); %time-delay
```

```
in index

                  %we delete as many zeros as possible. This will
increase the
                  %information in the psf (since the size of the psf is
fixed
                  td_index = td_index - min_zeros;
                  h_comp = [zeros(td_index,1) ; ht];

                  %calculate transmission coefficients and impulse
response of the ground
                  d = Rt_ss + Rr_ss;
                  T_ag = 2*cos(phi1tx)/(cos(phi1tx) +
sqrt(epsilon)*cos(phi2tx));
                  T_ga = 2*sqrt(epsilon)*cos(phi2rx)/(cos(phi1rx) +
sqrt(epsilon)*cos(phi2rx));
                  trans(1,1)= T_ag;
                  trans(2,1)= T_ga;


                  gd =
(d*sqrt(mu*epsilon*e_0)*loss_tangent/2)./(pi*((t-
d*sqrt(mu*epsilon*e_0)).^2 +
(d*sqrt(mu*epsilon*e_0)*loss_tangent/2)^2));

                  First_term =
T_ag*T_ga*gd/(8*pi^2*Rt*Rr*(c/sqrt(epsilon)));


                  if length(h_comp) > length(t)
                      h_comp = h_comp(1:length(t));
                  else
                      h_comp = [h_comp ; zeros(length(t)-
length(h_comp),1)];
                  end
                  A_scan =conv(First_term,h_comp); %calculate A-
scan
                  A_scan = A_scan(1:length(t));
                  B_scan(1:length(A_scan),1) = A_scan; %form B-scan
    end
    %norm_of_all = [norm_of_all  norm(B_scan,'fro')];
    %B_scan = B_scan/norm(B_scan,'fro');
    C_scan(k,:,:) = B_scan;
end
```

```
C_scan_diff = zeros(size(C_scan));
C_scan_diff(:,2:end,:) = diff(C_scan,1,2);
C_scan = C_scan_diff;



 %Now all the B-scans are made. The interpolation needs to be
done, with a
x_vector = -42:42; %the array has a width of 84 cm and a
resolution of 1 cm is required.
'start of interpolation'
save 13_loops C_scan;
clear all;
load 13_loops

tic;
interp_psf = inter_3D(C_scan);
toc;

manual_norm = sqrt(sum(sum(sum(interp_psf.^2))));
interp_psf = interp_psf/manual_norm;
```

deconv perpsf.m

```
%This script creates the point spread function for a sub-surface
scenario
%with the following geometry.

%z_Tx=0 - height of Transmitter (origin of coordinate system)
%z_Rx=0.27 m - position of Receiver with respect to Tx
%y - mechanical scan direction, [0, 60] cm

%z_target - .608 m m with respect to Tx
%Time of B-scan = 9.7752e-012s per step. 1024 steps gives
1.0010e-008 total
%time
%Mechanical scan direction 0.0024 m per step. 410 steps gives
.9840 m
%THIS FUNCTION CALCULATES THE FILTER RESULT
AND SHIFTS IT ACCORDINGLY IN
%THE T AND Y DIMENSION.
function s1 =
deconv_perpsf(C_int,point_spread_function,ER_threshold)


point_spread_function = shiftdim(point_spread_function,1);
%propabably because of implementation of (i)fft3
C_int = shiftdim(C_int,1);

%TO FREQUENCY DOMAIN
f_psf = fft3(point_spread_function);
f_raw = fft3(C_int);

%INITIATE ENERGY RATIOS
Energy_ratio = 0;

%The energy of the raw data is fixed
norm_raw = sqrt(sum(sum(sum(C_int.^2))));

%START AT MIDDLE, THIS WILL MOST LIKELY HAVE A
RESULT AND THEREFORE A GOOD
%ESTIMATE FOR THE REGULARIZATION PARAMETER

SNR_db = -27;

'start of WIENER filter'
```

```
% tic;
%wiener filter
SNR = 10^(SNR_db/20);
wif = conj(f_psf)./((conj(f_psf).*f_psf)+ 1/SNR);
f3= wif.*f_raw;
%result back to time domain
s1=real(ifft3(f3));

%Calculate 'Quality parameters' first Energy_ratio

norm_res = sqrt(sum(sum(sum(s1.^2))));
norm_psf = sqrt(sum(sum(sum(point_spread_function.^2))));

Energy_ratio = norm_res/norm_raw*100;

if (Energy_ratio < ER_threshold)

    while (Energy_ratio < ER_threshold)
        SNR_db = SNR_db + 1;
        SNR = 10^(SNR_db/20);
        wif = conj(f_psf)./((conj(f_psf).*f_psf)+ 1/SNR);
        f3= wif.*f_raw;

        %result back to time domain
        s1=real(ifft3(f3));

        %Calculate 'Quality parameters' first Energy_ratio
        norm_res = sqrt(sum(sum(sum(s1.^2))));
        norm_psf =
sqrt(sum(sum(sum(point_spread_function.^2))));

        Energy_ratio = norm_res/norm_raw*100;
    end
else
    while (Energy_ratio > ER_threshold)
        SNR_db = SNR_db - 1;
        SNR = 10^(SNR_db/20);
        wif = conj(f_psf)./((conj(f_psf).*f_psf)+ 1/SNR);
        f3= wif.*f_raw;

        %result back to time domain
        s1=real(ifft3(f3));

        %Calculate 'Quality parameters' first Energy_ratio
        norm_res = sqrt(sum(sum(sum(s1.^2))));
        norm_psf =
sqrt(sum(sum(sum(point_spread_function.^2))));

        Energy_ratio = norm_res/norm_raw*100;
    end
end
%Now the error. First reconstruct the raw data from the result
and the
%psf.
allData = real(ifft3(f3.*f_psf));
% Difference between the original and reconstructed data as
error,
% calculated for every x
norm_of_difference = sqrt(sum(sum(sum((C_int - allData).^2))));
norm_allData = sqrt(sum(sum(sum(allData.^2))));

error = 100*norm_of_difference/(norm_raw + norm_allData);
% toc;

SNR_db
Energy_ratio
error
```

comb deconv.m

```
%Superscript that calls calculates all 13 filter results and
combines them.

clear all;
load test_data_subsurf;

t = dt*(1:length(C(:,1,1)));
t_orig = t;
y=dy*(1:length(C(1,:,1)));
x=(-42:42);

%Compensate offset CHECKED RAW DATA HAS CORRECT
TIME-SCALE
for loopcounter=1:length(C(1,1,:))
    B = squeeze(C(:,:,loopcounter));
    offset = Tzero(loopcounter);  %offset in B-scan
    offset_index = round(offset/t(end)*length(t));
    C(:,:,loopcounter) =[B(offset_index:end,:) ;
zeros(offset_index-1,length(y))];%Keep same size
end

%Null the first part
C(1:350,:,:) = zeros(350,410,13);

%interpolate the C-scans
C = shiftdim(shiftdim(C,1),1); %the same orientation as the
point-spread function
C_int = inter_3D(C);
save raw C_int;

ER_threshold = 5;
save ER ER_threshold;

%Here the 13 deconvolutions are performed.
%This complicated load, save and clear structure is needed to
avoid Memory
%problems.
'Filter 1/13'
load raw;
load 3Dpsf_min042;
load ER;
s01= deconv_perpsf(C_int,interp_psf_min042,ER_threshold);
```

```
save result1 s01;
clear all
'Filter 2/13'
load raw;
load 3Dpsf_min035;
load ER;
s02= deconv_perpsf(C_int,interp_psf_min035,ER_threshold);
save result2 s02;
clear all
'Filter 3/13'
load raw;
load 3Dpsf_min028;
load ER;
s03= deconv_perpsf(C_int,interp_psf_min028,ER_threshold);
save result3 s03;
clear all
'Filter 4/13'
load raw;
load 3Dpsf_min021;
load ER;
s04= deconv_perpsf(C_int,interp_psf_min021,ER_threshold);
save result4 s04;
clear all
'Filter 5/13'
load raw;
load 3Dpsf_min014;
load ER;
s05= deconv_perpsf(C_int,interp_psf_min014,ER_threshold);
save result5 s05;
clear all
'Filter 6/13'
load raw;
load 3Dpsf_min007;
load ER;
s06= deconv_perpsf(C_int,interp_psf_min007,ER_threshold);
save result6 s06;
clear all

'Filter 7/13'
load raw;
load 3Dpsf_0;
load ER;
s07= deconv_perpsf(C_int,interp_psf_0,ER_threshold);
save result7 s07;
```

```
clear all


'Filter 8/13'
load raw;
load 3Dpsf_plus007;
load ER;
s08= deconv_perpsf(C_int,interp_psf_plus007,ER_threshold);
save result8 s08;
clear all
'Filter 9/13'
load raw;
load 3Dpsf_plus014;
load ER;
s09= deconv_perpsf(C_int,interp_psf_plus014,ER_threshold);
save result9 s09;
clear all
'Filter 10/13'
load raw;
load 3Dpsf_plus021;
load ER;
s10= deconv_perpsf(C_int,interp_psf_plus021,ER_threshold);
save result10 s10;
clear all
'Filter 11/13'
load raw;
load 3Dpsf_plus028;
load ER;
s11= deconv_perpsf(C_int,interp_psf_plus028,ER_threshold);
save result11 s11;
clear all
'Filter 12/13'
load raw;
load 3Dpsf_plus035;
load ER;
s12= deconv_perpsf(C_int,interp_psf_plus035,ER_threshold);
save result12 s12;
clear all
'Filter 13/13'
load raw;
load 3Dpsf_plus042;
load ER;
s13 = deconv_perpsf(C_int,interp_psf_plus042,ER_threshold);
save result13 s13;
```

```
clear all

load result1;
load result2;
load result3;
load result4;
load result5;
load result6;
load result7;
load result8;
load result9;
load result10;
load result11;
load result12;
load result13;

%WEIGHTING FACTORS. The results (in signal level) are
multiplied with the
%highest norm and divided by their own norm. The weaker
norms will
%therefore be compensated.
load all_norms;
min_norms = [manual_norm_min042 manual_norm_min035
manual_norm_min028 manual_norm_min021
manual_norm_min014 manual_norm_min007 manual_norm_0];
plus_norms = [manual_norm_plus042 manual_norm_plus035
manual_norm_plus028 manual_norm_plus021
manual_norm_plus014 manual_norm_plus007];
max_of_norms = max([min_norms plus_norms]);

s01 = -s01*max_of_norms/manual_norm_min042;
s02 = -s02*max_of_norms/manual_norm_min035;
s03 = -s03*max_of_norms/manual_norm_min028;
s04 = -s04*max_of_norms/manual_norm_min021;
s05 = -s05*max_of_norms/manual_norm_min014;
s06 = -s06*max_of_norms/manual_norm_min007;

s07 = -s07*max_of_norms/manual_norm_0;

s08 = -s08*max_of_norms/manual_norm_plus007;
s09 = -s09*max_of_norms/manual_norm_plus014;
s10 = -s10*max_of_norms/manual_norm_plus021;
s11 = -s11*max_of_norms/manual_norm_plus028;
s12 = -s12*max_of_norms/manual_norm_plus035;
```

```
s13 = -s13*max_of_norms/manual_norm_plus042;

%Perform all necessary shifts. Y_SHIFT = Half the mechanical
scan direction. The
%calculated x_shift and the determined t_shift
%X_SHIFT
%calculate shift in x-direction. It seems it has always this
%form 42 + xo (object location in psf) + 3.5
x_vec = -42:7:42;
x_circ_shift = 42 + x_vec;
%T_SHIFT
shift_01 = 189-65;%189-66; %189 is maximum in C, 66 is 2nd
maximum in result s01 & s12. The shifts are symmetrical
shift_02 = 189-73;%189-74;
shift_03 = 189-78;%189-79;
shift_04 = 189-83;%189-84;
shift_05 = 189-86;%189-87;
shift_06 = 189-88;%189-89;
shift_07 = 189-89;
shift_08 = 189-88;
shift_09 = 189-86;
shift_10 = 189-82;
shift_11 = 189-77;
shift_12 = 189-70;
shift_13 = 189-62;

%perform the shifts
s01 = circshift(s01,[shift_01 round(length(s01(1,:,1))/2)
x_circ_shift(1)+1]);
s02 = circshift(s02,[shift_02 round(length(s01(1,:,1))/2)
x_circ_shift(2)+1]);
s03 = circshift(s03,[shift_03 round(length(s01(1,:,1))/2)
x_circ_shift(3)+1]);
s04 = circshift(s04,[shift_04 round(length(s01(1,:,1))/2)
x_circ_shift(4)+1]);
s05 = circshift(s05,[shift_05 round(length(s01(1,:,1))/2)
x_circ_shift(5)+1]);
s06 = circshift(s06,[shift_06 round(length(s01(1,:,1))/2)
x_circ_shift(6)+1]);
s07 = circshift(s07,[shift_07 round(length(s01(1,:,1))/2)
x_circ_shift(7)+1]);
s08 = circshift(s08,[shift_08 round(length(s01(1,:,1))/2)
x_circ_shift(8)+1]);
s09 = circshift(s09,[shift_09 round(length(s01(1,:,1))/2)
x_circ_shift(9)+1]);
s10 = circshift(s10,[shift_10 round(length(s01(1,:,1))/2)
x_circ_shift(10)+1]);
s11 = circshift(s11,[shift_11 round(length(s01(1,:,1))/2)
x_circ_shift(11)+1]);
s12 = circshift(s12,[shift_12 round(length(s01(1,:,1))/2)
x_circ_shift(12)+1]);
s13 = circshift(s13,[shift_13 round(length(s01(1,:,1))/2)
x_circ_shift(13)+1]);
% NOW ALL THE RESULTS ARE PROPERLY WEIGHTED
AND SHIFTED. ALL THAT NEEDS TO
% BE DONE IS TO COMBINE THE 12 RESULTS TO 1
RESULT.
%
%
% combine all relevant parts of the filter results to one C-scan
without
% overlap.
s1_comb = zeros(size(s01));

s1_comb(:,:,1:4)   = s01(:,:,1:4);
s1_comb(:,:,5:11)  = s02(:,:,5:11);
s1_comb(:,:,12:18) = s03(:,:,12:18);
s1_comb(:,:,19:25) = s04(:,:,19:25);
s1_comb(:,:,26:32) = s05(:,:,26:32);
s1_comb(:,:,33:39) = s06(:,:,33:39);
s1_comb(:,:,40:46) = s07(:,:,40:46);
s1_comb(:,:,47:53) = s08(:,:,47:53);
s1_comb(:,:,54:60) = s09(:,:,54:60);
s1_comb(:,:,61:67) = s10(:,:,61:67);
s1_comb(:,:,68:74) = s11(:,:,68:74);
s1_comb(:,:,75:81) = s12(:,:,75:81);
s1_comb(:,:,82:85) = s13(:,:,82:85);


% WITH VARIABEL OVERLAP, I=0 GIVES PRACTICALLY
NO OVERLAP
s1_comb_over = zeros(size(s01));
i=7;
s1_comb_over(:,:,1:1+i-1)     = s1_comb_over(:,:,1:1+i-1)     +
s01(:,:,1:1+i-1);
s1_comb_over(:,:,8-i:8+i-1)  = s1_comb_over(:,:,8-i:8+i-1)  +
s02(:,:,8-i:8+i-1) ;
s1_comb_over(:,:,15-i:15+i-1) = s1_comb_over(:,:,15-i:15+i-1) +
s03(:,:,15-i:15+i-1);
s1_comb_over(:,:,22-i:22+i-1) = s1_comb_over(:,:,22-i:22+i-1) +
s04(:,:,22-i:22+i-1);
s1_comb_over(:,:,29-i:29+i-1) = s1_comb_over(:,:,29-i:29+i-1) +
s05(:,:,29-i:29+i-1);
s1_comb_over(:,:,36-i:36+i-1) = s1_comb_over(:,:,36-i:36+i-1) +
s06(:,:,36-i:36+i-1);
s1_comb_over(:,:,43-i:43+i-1) = s1_comb_over(:,:,43-i:43+i-1) +
s07(:,:,43-i:43+i-1);
s1_comb_over(:,:,50-i:50+i-1) = s1_comb_over(:,:,50-i:50+i-1) +
s08(:,:,50-i:50+i-1);
s1_comb_over(:,:,57-i:57+i-1) = s1_comb_over(:,:,57-i:57+i-1) +
s09(:,:,57-i:57+i-1);
s1_comb_over(:,:,64-i:64+i-1) = s1_comb_over(:,:,64-i:64+i-1) +
s10(:,:,64-i:64+i-1);
s1_comb_over(:,:,71-i:71+i-1) = s1_comb_over(:,:,71-i:71+i-1) +
s11(:,:,71-i:71+i-1);
s1_comb_over(:,:,78-i:78+i-1)   = s1_comb_over(:,:,78-i:78+i-1)
+ s12(:,:,78-i:78+i-1);
s1_comb_over(:,:,85-i:85)   = s1_comb_over(:,:,85-i:85)  +
s13(:,:,85-i:85);

%full all three
s1_full = s01 + s02 + s03 +s04 + s05 + s06 + s07 + s08 + s09 +
s10 + s11 + s12 + s13;

%window projection. window size = 50 %FULL gives no result,
%comb_withoverlap gives two targets, side weaker. No overlap
gives 2
%targets, but side-target has bad shape.
sum_window = zeros(size(squeeze(s1_full(1,:,:))));
sum_window_over = zeros(size(squeeze(s1_full(1,:,:))));
sum_window_full = zeros(size(squeeze(s1_full(1,:,:))));
window_size = 23
%WEP
% 'wep'
% for p=1:103
%     for q=1:85
%         wind_result = [];
%         wind_result_overlap = [];
%         wind_result_full = [];
%         for slide = window_size+1:512
%             wind_result = [wind_result; sum(s1_comb((slide-
window_size):slide,p,q).^2)];
```

```
%               wind_result_overlap = [wind_result_overlap;
sum(s1_comb_over((slide-window_size):slide,p,q).^2)];
%               wind_result_full = [wind_result_full;
sum(s1_full((slide-window_size):slide,p,q).^2)];
%           end;
%           sum_window(p,q) = max(wind_result);
%           sum_window_over(p,q) = max(wind_result_overlap);
%           sum_window_full(p,q) = max(wind_result_full);
%       end;
% end;

%ASWEP, first is negative, rest is positive
'aswep'
q_z = ones(1,window_size);
q_z(1:12) = -1*q_z(1:12);
for p=1:103
    for q=1:85
        wind_result = [];
        wind_result_overlap = [];
        wind_result_full = [];
        for slide = window_size+1:512
            %wind_result = [wind_result; (q_z*s1_comb((slide-
window_size):slide-1,p,q)).^2];
            wind_result = [wind_result; (q_z*s1_comb((slide-
window_size):slide-1,p,q)).^2];
            wind_result_overlap = [wind_result_overlap;
(q_z*s1_comb_over((slide-window_size):slide-1,p,q)).^2];
%              wind_result_full =
[wind_result_full;(q_z*s1_full((slide-window_size):slide-
1,p,q)).^2 ];
        end;
        sum_window(p,q) = max(wind_result);
        sum_window_over(p,q) = max(wind_result_overlap);
%          sum_window_full(p,q) = max(wind_result_full);
    end;
end;

sum_window2 = sum_window;
sum_window2 = medfilt2(sum_window,[6 6]); %not bigger than
6, not smaller than 5
sum_window_over = medfilt2(sum_window_over,[6 6]);

%convert to log
 s1_full = 20*log10(abs(s1_full)+0.001);
 sum_window_db = 20*log10(sum_window2+0.001);
 sum_window_over_db = 20*log10(sum_window_over+0.001);
% sum_window_over = 20*log10(sum_window_over+0.001);
% sum_window_full = 20*log10(sum_window_full+0.001);

%maxima
max_sum_window = max(max(sum_window_db));
max_sum_window_over = max(max(sum_window_over_db));
% max_sum_window_full = max(max(sum_window_full));

%toc;
% load data again just to get nice axes
load test_data_subsurf;
t = dt*(1:length(C(:,1,1)));
t_orig = t;
y=dy*(1:length(C(1,:,1)));
x=(-42:42);

% close all; DO NOT USE OVERLAP, USE WINDOW IS 23
WITH ASWEP AND MEDIAN
% FILTERING WITH 6
figure;imagesc(x,100*y,sum_window_db-max_sum_window,[-
20 0]),colorbar,title('ASWEP Deconvolution result [dB]')
%figure;imagesc(x,100*y,sum_window_over_db,[max_sum_win
dow_over-20 max_sum_window_over]),colorbar,title('Top down
projected deconvolution result')
%figure;imagesc(sum_window_over_db,[max_sum_window_ov
er-10 max_sum_window_over]),colorbar
%
figure;imagesc(x,100*y,sum_window_full,[max_sum_window_f
ull-10 max_sum_window_full]),colorbar

figure;imagesc(s1_sum_no_overlap,[s1_sum_no_overlap_max-
10 s1_sum_no_overlap_max]),colorbar
% % figure;imagesc(s1_sum_comb,[s1_sum_comb_max-10
s1_sum_comb_max]),colorbar
% % figure;imagesc(s1_sum_full,[s1_sum_full_max-10
s1_sum_full_max]),colorbar
% figure;imagesc(sum_window,[max_sum_window-10
max_sum_window]),colorbar
% figure;imagesc(sum_window_over,[max_sum_window_over-
10 max_sum_window_over]),colorbar
% figure;imagesc(sum_window_full,[max_sum_window_full-10
max_sum_window_full]),colorbar
```

```
% % for e=1:30 %021 good starting value
% %
figure;imagesc(squeeze(s1_comb_over(175+e,:,:)),[max_s1_com
b_over-10 max_s1_comb_over]),xlabel('x [cm]'),ylabel('y
[cm]'),title('Deconvolution result seperated psfs [dB]'),colorbar;
% % end;
```

inter 3D.m

```
%this function interpolates the 13 (synthetic) B-scans of all the
loops

function C_scan_int = inter_3D(All_B)

%For memory reasons the C_scans resolution is reduced.
All_B =
All_B(:,1:2:length(All_B(1,:,1)),1:4:length(All_B(1,1,:)));

Delta_x = 0.01

[xi,yi,zi] =
meshgrid(1:length(All_B(1,:,1)),1:1/7:13,1:length(All_B(1,1,:)));
%
%we give a resolution of 1/7 between the loops. Corresponding
to 1 cm.

C_scan_int = interp3(All_B,xi,yi,zi,'linear'); %the interpolation
works
```