Multiphase Motion Estimation in a Two Phase Flow

Examensarbete utfört i Medieteknik
vid Linköpings tekniska högskola
av

Gabriella Gustafsson

LITH-ISY-EX--05/3723--SE

Handledare: **Carl-Fredrik Linberg,**
ABB Corporate Research

Examinator: **Klas Nordberg**

Linköping 051011

Multiphase Motion Estimation in a Two Phase Flow

Gabriella Gustafsson

October 11, 2005

Abstract

To improve the control of a steel casting process ABB has developed an Electro Magnetic Brake (EMBR). This product is designed to improve steel quality, i.e. reduce non-metallic inclusions and blisters as well as risk of surface cracks. There is a demand of increasing the steel quality and in order to optimize the steel casting, simulations and experiments play an important role in achieving this. An advanced CFD simulation model has been created to carry out this task.

The validation of the simulation model is performed on a water model that has been built for this purpose. This water model also makes experiments possible. One step to the simulation model is to measure the velocity and motion pattern of the seeding particles and the air bubbles in the water model to see if it corresponds to the simulation results.

Since the water is transparent, seeding particles have been added to the liquid in order to observe the motion of the water. They have the same density as water. Hence the particles will follow the flow accurately. The motions of the air bubbles that are added into the water model need also to be observed since they influence the flow pattern.

An algorithm - "Transparent motions" - is thoroughly inspected and implemented. "Transparent motions" was originally designed to post process x-ray images. However in this thesis, it is investigated whether the algorithm might be applicable to the water model and the image sequences containing seeding particles and air bubbles that are going to be used for motion estimation.

The result show satisfying results for image sequences of particles only, however with a camera with a faster sampling interval, these results would improve. For image sequences with both bubbles and particles no results have been achieved.

Preface

This thesis has been carried out at ABB Corporate Research from February to July 2005. This work has been very exciting in view of fact that it has deepened my understanding in areas such as physics, mathematics and image processing.

I would like to thank the following people for helping me during this thesis.

Examiner Klas Nordberg
Supervisor Carl-Fredrik Lindberg
Alf Isaksson
Jan-Erik Eriksson
Göte Tallbäck
Ping Li
My family and friends

Contents

List of Figures

Chapter 1

Introduction

There are different methods of casting steel. The most commonly used technique in steel production is Continuous Casting. In recent years the demand for higher quality and larger quantities of steel has increased. This is what gives the Continuous Casting method an advantage compared to other existing methods, due to the fact that it is faster. Continuous Casting processes are the most efficient ways to solidify large volumes of metal.

1.1 Continuous Casting

Continuous casting is distinguished from other solidification processes by its steady state nature, relative to an outside observer in a laboratory frame of reference. The molten metal solidifies against the mold walls while it is simultaneously withdrawn from the bottom of the mold at a rate which maintains the solid/liquid interface at a constant position with time [1].

In the continuous casting process, pictured in figure 1.1, molten steel flows from a ladle, through a tundish into the mold. It should be protected from exposure to air by a slag cover over each vessel and by ceramic nozzles between vessels. Argon gas is added to prevent clogging. Once in the mold, the molten steel freezes against the water-cooled copper mold walls to form a solid shell. This shell contains the liquid as the shell is withdrawn continuously from the bottom of the mold.

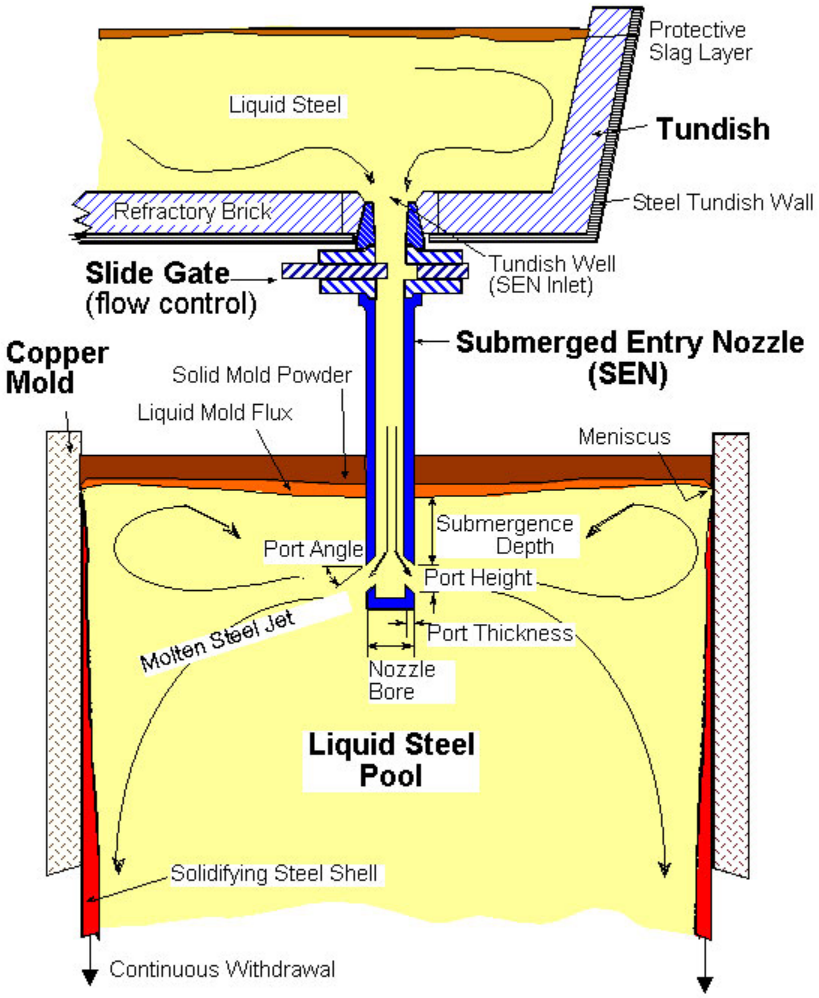

Figure 1.1: Continuous Casting process

1.2 Electro Magnetic Brakes

To improve the control of the steel casting process ABB has developed an Electro Magnetic Brake (EMBR). This product is designed to improve steel quality, i.e. reduce non-metallic inclusions and blisters as well as risk of surface cracks. EMBR brakes the flow of steel, which helps to remove impurities and prevents mold powder from being dragged down into the mold see figure 1.2.

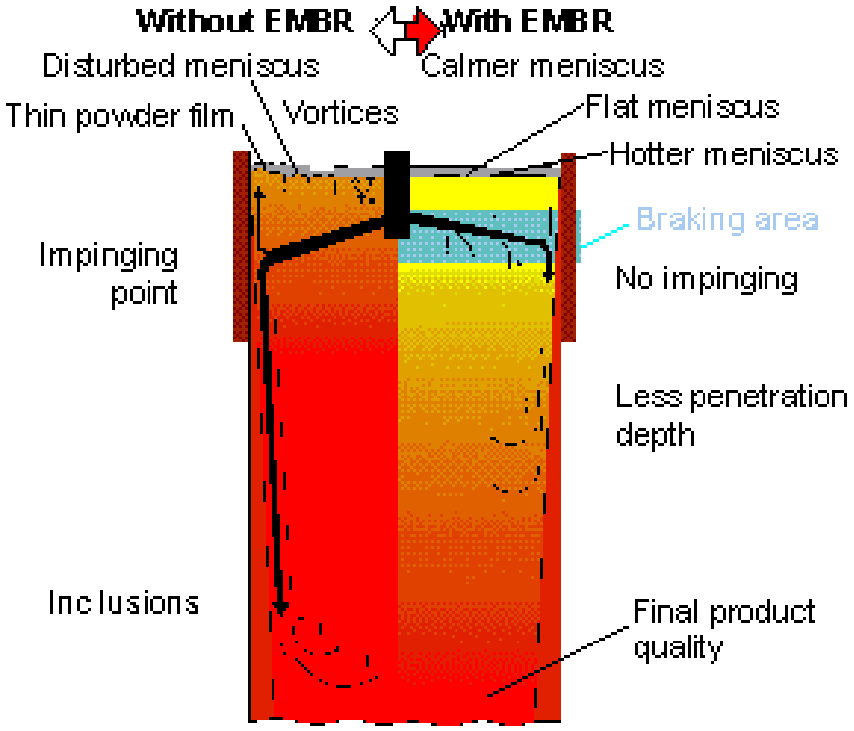

Figure 1.2: EMBR

To achieve high steel quality, it is important to use the correct braking power [8].

In order to optimize the steel casting in such a way that it will produce steel of as high quality as possible, it is crucial to perform simulations and experiments. Therefore, a simulation model software based on advanced Computational Fluid Dynamics (CFD) has been created. The half scale water model has been developed in order to be able to validate the simulation model and also to be able to perform experiments. The water

model consists of the parts in figure 1.2 and argon is substituted by air and the steel with water. When validating the model ABB wants to obtain similar flow topology and velocity vectors in the simulation as in measured velocities in the water model, both for the liquid and air bubbles.

1.3 Motion Estimation

1.3.1 Motion

Naturally one relates motion with changes in position. In image processing motion might also be associated with changes. That is variation in one image to another in an image sequence. Differences from one image to another in image processing are change of gray values. However a change of gray values does not necessarily say that there is any motion present. An example of this is an image sequence showing a room and a door. When the door is closed one illumination is present while in another image where the door is open a new illumination takes place which results in change of gray values. Nevertheless no motion is present. This observation results in that motion might result in spatial gray value changes, but it might also be due to the fact that the illumination has been altered [7]. In image processing it is possible to measure changes in pixel values over time, however this is only an indirect connection to movement. Even if something is moving it is not always possible to measure it. There is no simple way of measuring movement in images, but the optical flow method, described below, provides a way that might be good enough after some approximations and assumptions are made.

1.3.2 Optical Flow in general

Motion estimation can sometimes be challenging and there are numerous methods available. However, in the following section one method of approach - Optical Flow - will be described. Optical Flow has a chance of solving the motion estimation problem because of the fact that there are constant illumination in the scene and that the objects in a neighborhood move in a similar way.

When analyzing a series of images over time - a sequence of images - containing moving objects, it is possible to retrieve useful information as a result of the movement of the objects from image to image.

An example of this is an image sequence of a bicycle rider cycling on the street. With the help of optical flow it is possible to calculate the motion of the bicyclist. Another area of application is that by determining the motions it is also achievable to separate the background from the foreground.

The function optical flow is able to estimate the number of objects there are in the scene, the direction they are moving, how fast they are moving, and the type of motion.

Using the optical flow technique, one two dimensional vector $u(x, y)$ is computed for each pixel in the image sequence. The vector describes the direction and how fast the content in that pixel is moving [4].

1.3.3 Transparent Motions

The algorithm - Transparent Motions - is an unexplored way to estimate multiple motions in two phase flows [9]. It was originally designed for post processing x-ray images. It is based on an Optical Flow algorithm. It estimates two motions in every pixel. Hence, this method has a good chance of estimating motion in the water model.

1.4 Purpose

The goal of this report is to describe the algorithm "Transparent motions" in detail and investigate whether it is possible to use it for the purpose of measuring motion of particles and bubbles in a water model developed by ABB Corporate Research. Hopefully, this report will show that this algorithm is useful in such a way that it provides reliable vector maps for the bubbles and particles separately.

1.4.1 Problem description

One way of estimating multiple motions is to segment the two phases and estimate the motions separately. Here, the term two

phase flow refers to a system containing gas and liquid - air and water.

This is not possible with the equipment available today. This includes one high speed camera and one video camera. The solution to this problem is to find a method that does not require segmentation of the two phases. Hence, "Transparent motions" is investigated. The theory behind transparent motions needs to be inspected thoroughly in order to be able to implement an algorithm. The image sequences that are going to be used for motion estimation will contain seeding particles and air bubbles.

1.5 Tasks

This section describes the tasks that are carried out in the thesis.

1. The first task is to collect background information in order to understand the steel casting process and the water model.
2. The largest part of the thesis is to understand the theory behind "Transparent motions".
3. Implement the method.
4. Test the method using visual means.

1.6 Thesis Outline

- The Background chapter offers an introduction to the water model, the Particle Image Velocimetry system and the camera.
- The third chapter - Transparent Motions provides a thorough walk through of the article of which the thesis is based on.
- The fourth chapter shows the estimation results.
- The fifth chapter provides a discussion.

Chapter 2

Background

2.1 The Particle Image Velocimetry System

The Particle Image Velocimetry system (PIV) is provided from Dantec Dynamics. It is specifically set up to provide instantaneous velocity vector measurements in a cross-section of the mold (see figure 2.1 for velocity vector map). This system is an alternate method to estimate motions. However in this thesis it is used for the purpose of image acquisition only.

2.1.1 Principles

"In PIV, the velocity vectors are derived from sub sections of the target area of the particle seeded flow by measuring the movement of particles between two light pulses.

The flow is illuminated in the target area with a light sheet shown in figure 2.2. The camera lens images the target area onto the CCD array of a digital camera. The CCD is able to capture each light pulse in separate image frames.

Once a sequence of two light pulses is recorded, the images are divided into small subsections called interrogation areas. The interrogation areas from each image frame, I1 and I2, are cross correlated with each other, pixel by pixel."

"A velocity vector map over the whole target area is obtained by repeating the cross correlation for each interrogation area over the two image frames captured by the CCD camera" [6].

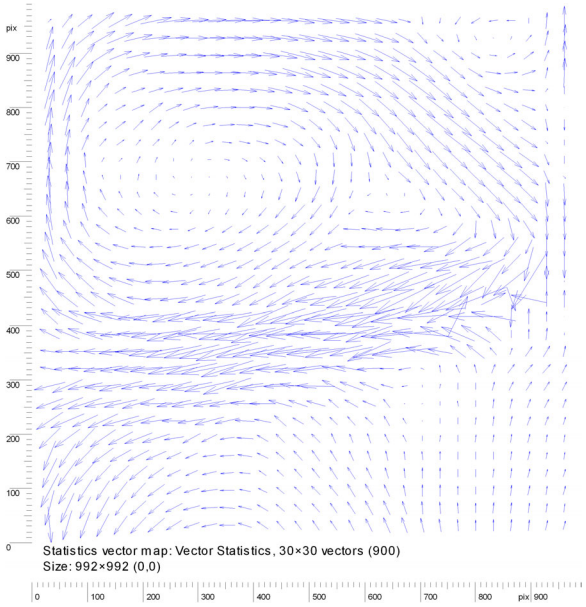

Figure 2.1: An example of a velocity vector map.

2.1.2 The Camera

The available camera is part of the PIV system from Dantech Dynamics. It is supported in the FlowMap software which is used for post processing the images. This is how Dantec Dynamics describes the camera.

"The cross-correlation cameras supported in the FlowMap PIV system use high-performance progressive-scan-interline CCD chips. Such chips includes 1018 x 1008 light-sensitive cells and an equal number of storage cells. These are shown schematically below in figure 2.3. The latter are not exposed to light. The first laser pulse is timed to expose the first frame, which is transferred from the light sensitive cells to the storage cells immediately after the laser pulse. The second laser pulse is then fired to expose the second frame. The storage cells now contain the first camera frame of the pair with information about the initial positions of seeding particles. The light sensitive cells contain the second camera frame. These two frames are then transferred sequentially to the camera outputs for acquisition and cross correlation processing by the FlowMap PIV Processor."

This camera can produce images with an interval of 37ms at most, which means less than 30 images per second. The

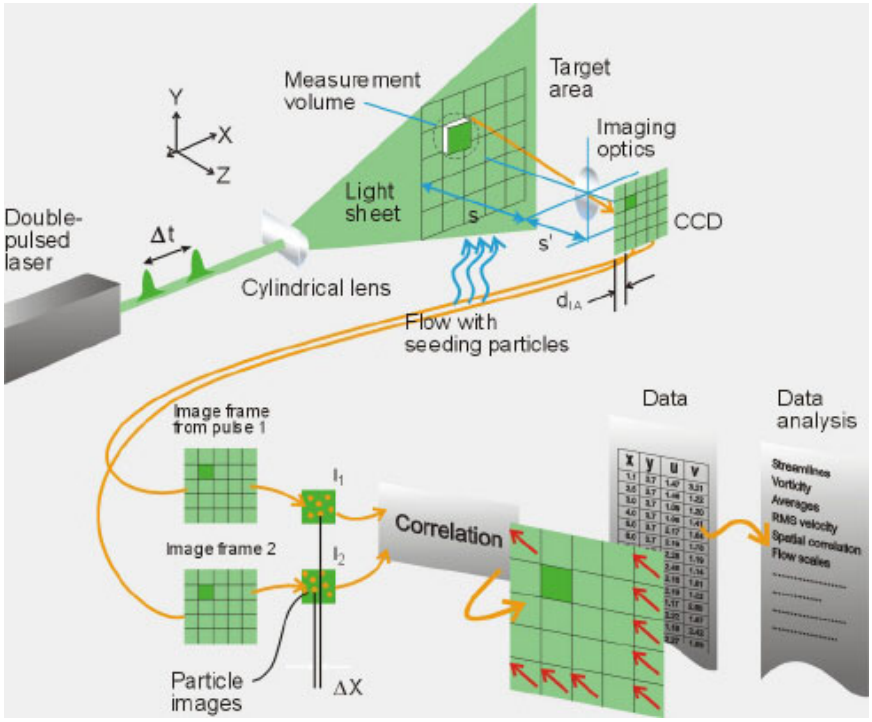

Figure 2.2: Overview of the principles of the PIV system.

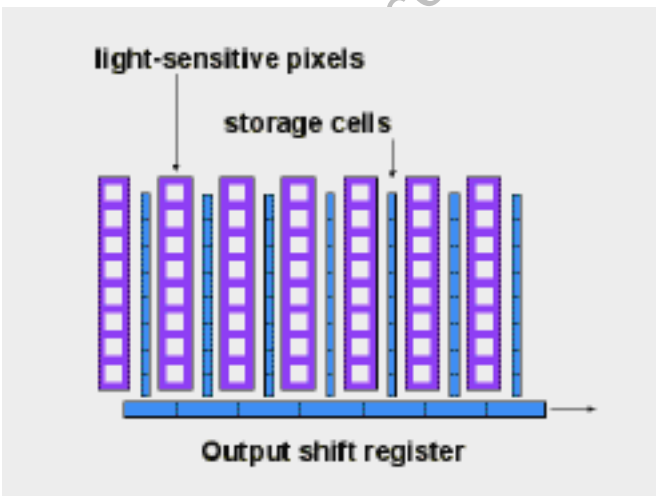

Figure 2.3: Schematic illustration of light-sensitive pixels and storage cell layout of cross correlation PIV cameras.

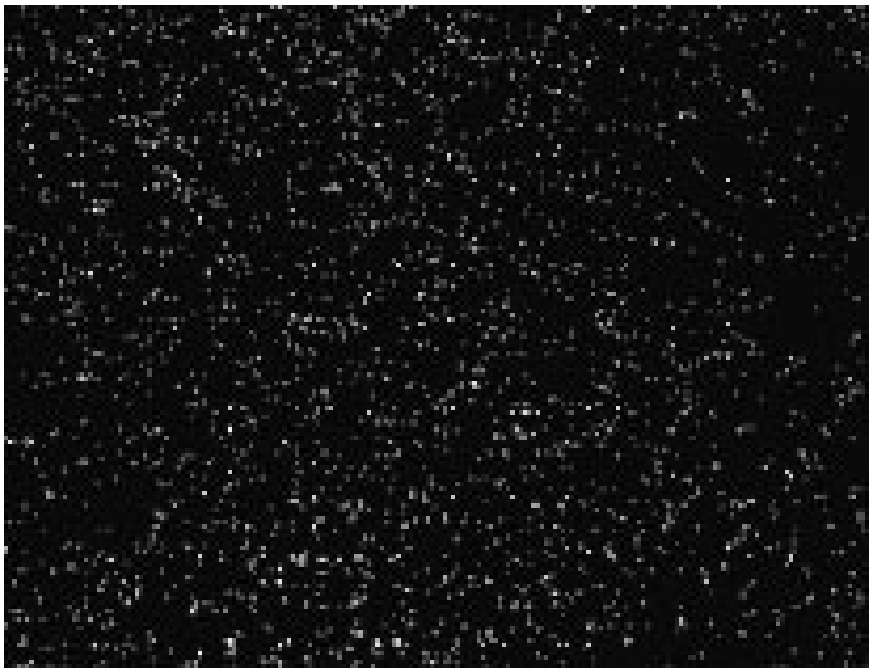

Figure 2.4: An image of particles acquired by the PIV system.

resolution of the images is 1018x1008 pixels (see figure 2.4 for an example of an image).

2.2 The Water Model

ABB is developing Electromagnetic Brakes (EMBR). The central reason of developing this product is to ensure that the quality of the steel in the casting process will be as high as possible. To improve the product even more an on-line control system, EM Control, is being developed as well. It controls the surface flow rate and reduces flow and jet oscillations. The water model is used for carrying out experiments and for validating a simulation model of a steel casting process.

The water model looks like figure 2.5. The smaller white dots are seeding particles and the larger ones are air bubbles. The bubbles are representing the argon gas which is used to reduce clogging in the pipe and lift impurities to the surface in a real process.

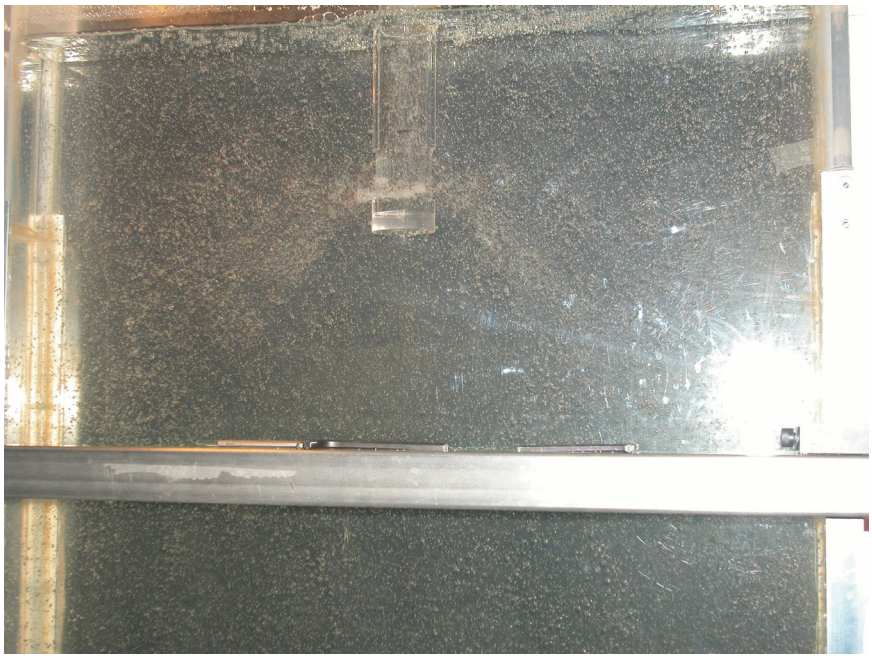

Figure 2.5: With seeding particles and bubbles

2.2.1 Motion estimation in the water model

One step to the simulation model is to measure the velocity and motion pattern of the seeding particles and the air bubbles in the water model to see if it corresponds to the simulation results. Seeding particles are added to the water in order to observe the motion of the transparent water. They have a size of $50\mu m$, and have the same density as water. Hence they will follow the flow of the water accurately. The other motion that is interesting is the air bubbles. They influence the flow pattern of the water. Hence, it is necessary to perform motion estimation of particles with the air bubbles present. The motion estimation is achieved by taking multiple images of the flow of the water. With the help of the laser, a thin sheet is lit up. A camera is synchronized with the laser and acquires one image each time the laser is turned on. This results in image sequences showing the particles and the bubbles. The particles have moved from one image to another. The change of gray values indicates that there is motion present since the particles have constant illumination. These images are then used in "Transparent motions" in order to be able to determine the velocities.

Chapter 3

Transparent Motions

The following chapter describes the algorithm "Transparent Motions" in the article [9]. It is a framework for estimating multiple motions that does not require segmentation of the two phases.

It is based on the thought that the images are composed of two layers. In our case with two motions, the images have two layers. One layer consists of the particles that represent the water flow and the other layer contains the air bubbles. The outcome from this technique is that you obtain a pair of vectors from every pixel in the image. The drawback with this routine is that it does not give any information about which layer the motion belongs to. However, it might be possible to estimate which layer a velocity belongs to by assuming that the velocities should not vary too much in a neighborhood.

This method uses optical flow to detect the motions. The optical flow is defined as the flow of gray values at the image plane, which we can observe. Optical flow and motions field are equal if the object brightness does not change. For notational clarity it is shown in figure 3.1 that the origin is in the upper left corner. Throughout this report x is used to describe the rows and y the columns in the image. These coordinates are always discrete integers. The first image occurs when t = 0 and the following at t = 1 to t = N, where N is the number of images in the sequence.

3.1 Optical flow

The optical flow is derived from the following equation.

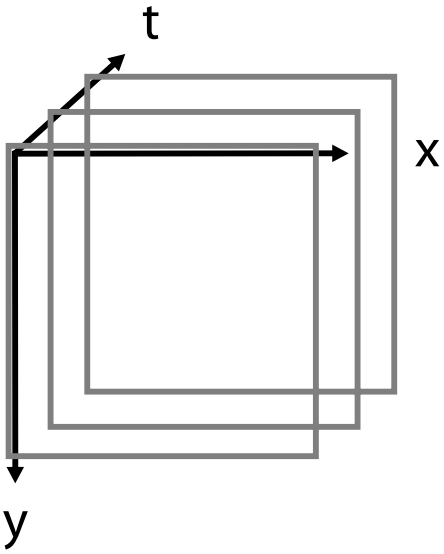

Figure 3.1: Coordinate system. t = [frames] y = [pixels] x = [pixels].

$$v_x f_x + v_y f_y + f_t = 0 \tag{3.1}$$

This equation is based on the assumptions that the object brightness does not change, and that the neighboring pixels move in a similar way. Where does this equation come from and how could it be true?

The Optical Flow can be illustrated in the following way. If any point is considered in the first image and if it is moved

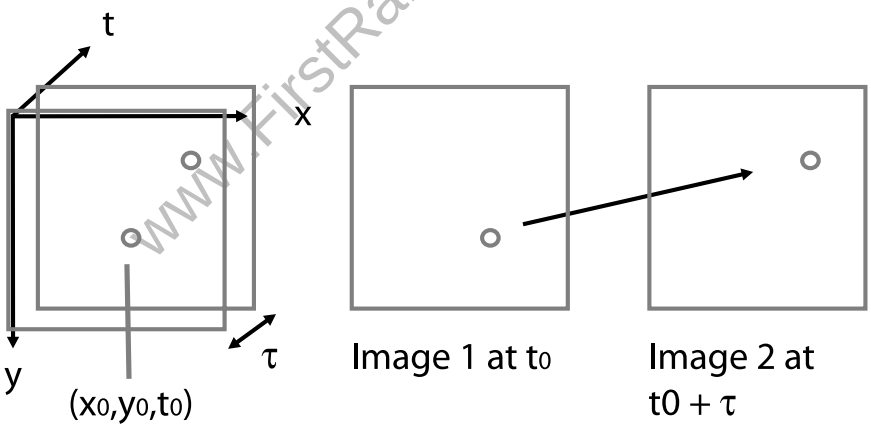

Figure 3.2: Visualization of Optical Flow

a certain distance in the next image, $v_x t$ in the x-direction and $v_y t$ in the y-direction, the following equation is acquired,

$$f(x_0, y_0, t_0) = f(x_0 + v_x\tau, y_0 + v_y\tau, t_0 + \tau) \tag{3.2}$$

where f is an intensity function and x_0, y_0, t_0, are the coordinates of the pixel that is being moved. This is illustrated in figure 3.2. Eq.(3.2) says that if the point is moved a certain distance with a constant velocity, this point will maintain its intensity. Both sides in Eq.(3.2) is differentiated with respect to τ, the left hand side becomes zero and the right hand side an expression of partial derivatives Eq.(3.3). These are calculated by using the chain rule.

$$0 = \frac{\partial f}{\partial x}\frac{\partial x}{\partial \tau} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial \tau} + \frac{\partial f}{\partial t}\frac{\partial t}{\partial \tau} \tag{3.3}$$

$$\frac{\partial x}{\partial \tau} = v_x, \frac{\partial y}{\partial \tau} = v_y, \frac{\partial t}{\partial \tau} = 1 \tag{3.4}$$

$$0 = \frac{\partial f}{\partial x}v_x + \frac{\partial f}{\partial y}v_y + \frac{\partial f}{\partial t} \tag{3.5}$$

Normally these partial derivatives, $\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial y}$, and $\frac{\partial f}{\partial t}$ are written in a shorter form, as f_x, f_y, and f_t. The intensity derivatives in the x-, y- and t-directions are simply measures of how fast the intensities in the image are changing. And if the notation mentioned above is used it is possible to see that Eq.(3.5) is the same as the constant brightness constraint Eq.(3.1). v_x and v_y are the optical flow components in the x- and y-directions. By rewriting Eq.(3.3) it is possible to see that this is a plane with a normal vector.

$$[f_x f_y f_t] \cdot [v_x v_y 1] = 0 \tag{3.6}$$

The intensity derivatives are estimated, and the velocities are unknown and should be determined. With one equation and two unknowns, this is an under determined system. The velocity vector is a normal vector to the plane that contains the intensity derivatives. This is due to the fact that the dot product between the derivatives and the velocities are zero. The problem with this is to find this vector.

The main task is to estimate the velocity vector in each pixel. Looking at Eq(3.6) it is possible to say that in order to extract

useful information from the image, it is necessary to observe how the neighboring pixels behave. The plane of intensity derivatives represent the neighboring pixels.

3.2 Single Motion Estimation

Single motion estimation will follow the schematic illustration in figure 3.3 and use the box called mixed motion coefficients as the last step. However, for single motion estimation the coefficients will not be mixed. In fact, the motion coefficients will contain the estimated velocities. The last three steps are introduced in section 3.3. The following section will describe the procedure of estimating one motion.

3.2.1 The Derivative Operator for one motion

The derivative operator shows in what way the images need to be differentiated. The differentiation is performed with the derivative operator and a Gaussian function. This is then filtrated with the images. From this result it is possible to say something about the motion.

The derivative operator looks like this:

$$\alpha(v) = v_x\frac{\partial}{\partial x} + v_y\frac{\partial}{\partial y} + \frac{\partial}{\partial t} \tag{3.7}$$

This operator will differentiate a function in all directions in the case with one motion. v is a vector that contains the velocities in the x- and y-directions, v_x and v_y. The derivative operator weights the derivatives with the velocity components and thus it depends on v. Using the derivative operator on the image generates,

$$\alpha(v)f = 0 \tag{3.8}$$

This is true for one motion.
The next step is to use the derivative operator.

From the definition of α in Eq.(3.7) it is possible to rewrite Eq.(3.1) as Eq.(3.8). The velocities v_x and v_y are unknowns

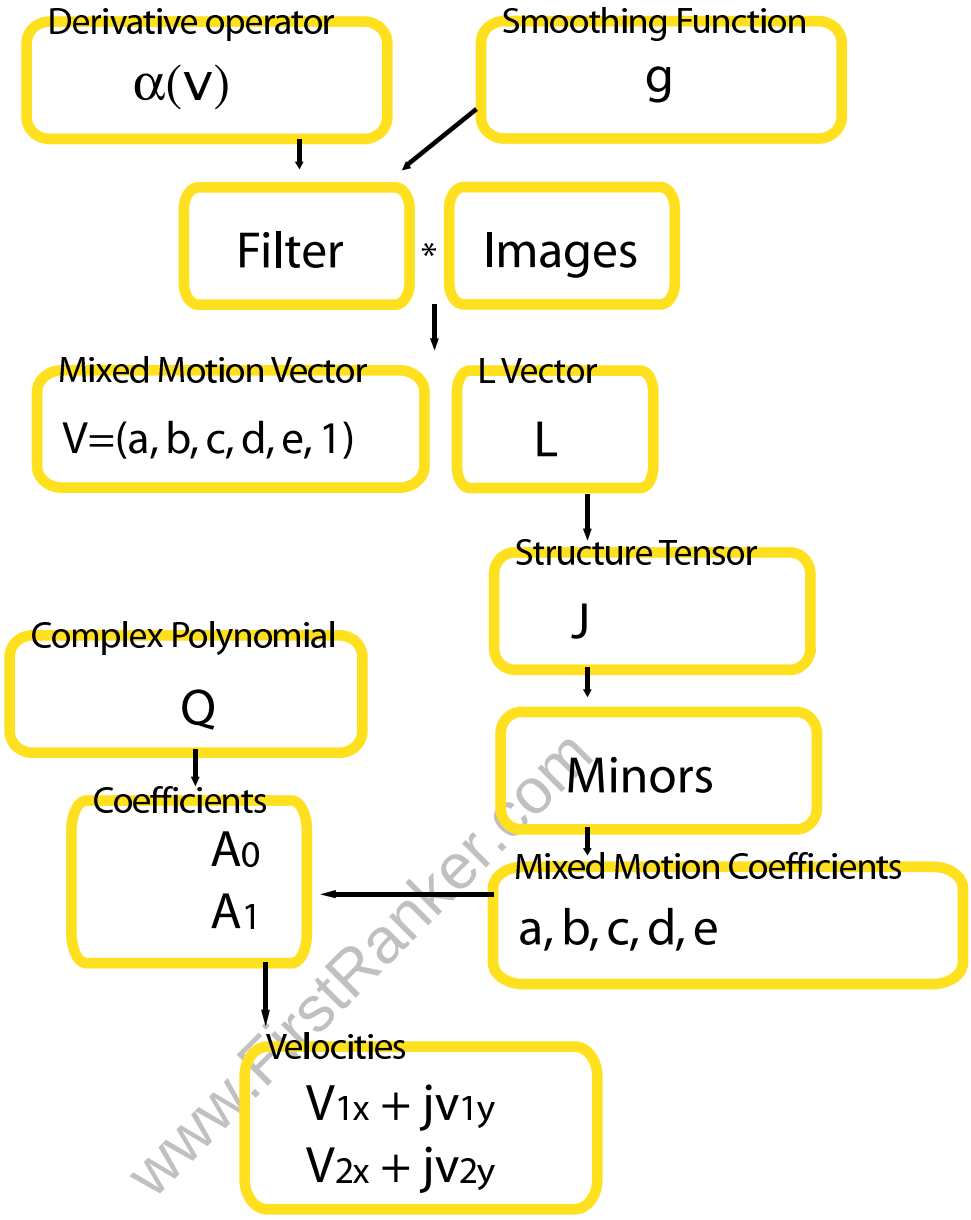

Figure 3.3: Schematic view of Transparent motions. Each box represents a step in the algorithm.

and need to be determined. The velocities are put in a vector in order to be able to handle the derivatives separately. This vector comes from the constant brightness constraint and is more obvious when looking at the rewritten constraint in Eq(3.6).

$$V = \begin{pmatrix} v_{1x} \\ v_{1y} \\ 1 \end{pmatrix} \tag{3.9}$$

3.2.2 The smoothing function

The smoothing function is a way to differentiate the images using a filter. This is shown in the equations below. If Eq.(3.2) is assumed to be true it follows that Eq.(3.10) is true (see Appendix A.6 for proof) and that $g * f$ complies with the constant brightness constraint in Eq.(3.11) under the assumption that the velocity is relatively constant in a neighborhood of the same size as the function g.

$$(g * f)(x_0, y_0, t_0) = (g * f)(x_0 + v_x\tau, y_0 + v_y\tau, t_0 + \tau) \tag{3.10}$$

$$v_x(g * f)_x + v_y(g * f)_y + (g * f)_t = 0 \tag{3.11}$$

The equation below shows that applying the derivative operator on the images convolved with the smoothing function is the equivalent of applying the derivative operator on the smoothing function and then convolving this result with the images (see Appendix A.5 for proof).

$$\alpha(v)(g * f) = \alpha(v)(g) * f \tag{3.12}$$

In order to estimate the velocity components v_x and v_y according to Eq.(3.1), gradients from f are needed. Since f is a signal that has been sampled it is only possible to approximately calculate the gradients. One way to approximate the gradients is to use Eq.(3.12) that says that gradients from $(g * f)$ can be obtained by convoluting gradients of g with f, which is simple to implement if g is chosen as a Gaussian function since gradients of g becomes filters with a satisfying size if sigma is chosen accordingly. With the aid of a smoothing filter, irrelevant details are reduced in the images. It can reduce details that are

small with respect to the size of the filter mask. An example of a smoothing function, is a Gaussian function

$$g(x, y, t) = e^{\left(-\frac{1}{2}\frac{x^2}{\sigma_x^2}-\frac{1}{2}\frac{y^2}{\sigma_y^2}-\frac{1}{2}\frac{t^2}{\sigma_t^2}\right)} \tag{3.13}$$

but any function that is local in both the spatial- and the temporal domains will do.

The smoothing function is then differentiated using the derivative operator. The Gaussian function provides a way to soften the images by reducing the noise.

Three derivatives are obtained. The Gaussian function helps to observe the neighboring pixels. Pixels close to the pixel of concern will be weighted with a high value. Pixels further away will be of less importance.

$$\begin{aligned} &\frac{\partial}{\partial x}g(x, y, t) \\ &\frac{\partial}{\partial y}g(x, y, t) \\ &\frac{\partial}{\partial t}g(x, y, t) \end{aligned} \tag{3.14}$$

This filters are separable here since g is chosen as a Gaussian function, which allows filtration of the images with the derivatives separately. The filters are convolved with the images, and by rewriting $\alpha(v_1)g * f = 0$ as,

$$LV = 0 \tag{3.15}$$

it is possible to observe that the motions in V are handled separately. V comes from Eq.(3.9), and L is a row vector and consists of the filter convolved with the images:

$$L = \left(\frac{\partial}{\partial x}g(x, y, t), \frac{\partial}{\partial y}g(x, y, t), \frac{\partial}{\partial t}g(x, y, t)\right) * f \tag{3.16}$$

3.2.3 The Structure Tensor

A structure tensor can describe the structure or the orientation of a signal. It is obtained by estimating the gradient, usually by using a Gaussian derivative filter, and then by calculating the outer product in every pixel of the gradient vector by itself. This

means that one structure tensor is obtained from every pixel in the image [5].

Eq.(3.15) is always possible to solve, but it has no unique solution. It is necessary to look in the surroundings in order to estimate V uniquely. A minimization of the left hand side Eq.(3.15) solves this problem. Eq.(3.17) describes ϵ which involves the left-hand side of Eq.(3.15) and thus it is dependent on V. If $LV = 0$ or if it is small then $|\epsilon| = 0$ if it is assumed that the velocity is constant. The objective is to find the V that minimizes ϵ. Hence, a least square minimization is used. ω is a convolution kernel, which may be, for example Gaussian.

$$\epsilon = \int \omega(x, y, t)(L(x, y, t)V)^2 dxdydt \tag{3.17}$$

In order to solve the minimization of ϵ, a constraint is needed because of the fact that the solution $V = 0$ is not wanted. What type of solution that is desired needs to be specified. The constraint is chosen in such a way that it will make the implementation as simple as possible in practice. The desired type of solution and thus the constraint is to find V that has the norm one: $|V| = 1$. The gradient of ϵ, which is a function of the elements in V, is proportional to the gradient of the constraint for those V that minimizes ϵ at the same time as they comply with the constraint:

$$\int L(x, y, t)^T L(x, y, t)V\omega(x, y, t)dxdydt = \lambda V \tag{3.18}$$

This method is a commonly used technique for finding the minimum points or maximum points of multi-variable functions under some constraints. Supposing that the motion vectors are locally constant, V is taken out of the integral.

$$J = \int L(x, y, t)^T L(x, y, t)\omega(x, y, t)dxdydt \tag{3.19}$$

$$JV = \lambda V \tag{3.20}$$

J is a structure tensor. A structure tensor describes the structure and orientation in the images. According to Eq.(3.20) the V that minimizes ϵ and complies with the constraint should be an eigenvector to J. It is possible to show that V needs to be chosen as an eigenvector with the smallest eigenvalue to generate the smallest ϵ.

3.2.4 Vectors from minors

"A minor M_{ij} is the reduced determinant of a determinant expansion that is formed by omitting the i-th row and j-th column of a matrix. So, for example, the minor of the above matrix in figure 3.4 is given by figure 3.5 " [3].

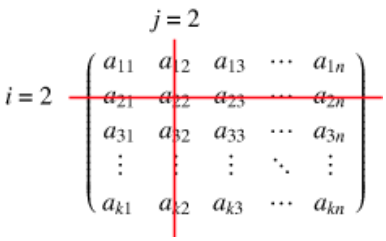

Figure 3.4: A matrix

The minors provide a useful way to extract the vector containing the motions from the tensor. Looking at Eq.(3.20) it is possible to rewrite this as Eq.(3.21).

$$\begin{pmatrix} J_{11} & J_{12} & J_{13} \\ J_{21} & J_{22} & J_{23} \\ J_{31} & J_{32} & J_{33} \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{3.21}$$

$$\begin{aligned} J_{11}v_x + J_{12}v_y + J_{13} &= 0 \\ J_{21}v_x + J_{22}v_y + J_{23} &= 0 \\ J_{31}v_x + J_{32}v_y + J_{33} &= 0 \end{aligned} \tag{3.22}$$

The rows are linearly dependent if Eq.(3.21) is true and thus

$$det(J) = 0 \tag{3.23}$$

One way of calculating the determinant is by using the minors of the matrix.

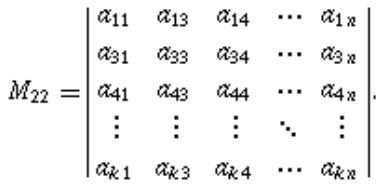

Figure 3.5: The minor

$$\begin{pmatrix} \boxed{J_{11}} & J_{12} & J_{13} \\ J_{21} & \boxed{J_{22} & J_{23} \\ J_{31} & J_{32} & J_{33}} \end{pmatrix} \tag{3.24}$$

$$det(J) = \left\{ \begin{array}{c} \boxed{J_{11}M_{11}} + J_{12}M_{12} + J_{13}M_{13} = 0 \\ J_{21}M_{21} + J_{22}M_{22} + J_{23}M_{23} = 0 \\ J_{31}M_{31} + J_{32}M_{32} + J_{33}M_{33} = 0 \end{array} \right\}$$

It is obvious that if this is compared to Eq.(3.23), the minor vectors and the velocity vector are parallel and thus proportional.

$$V = \begin{pmatrix} v_x \\ v_y \\ 1 \end{pmatrix} \propto \begin{pmatrix} M_{11} \\ M_{12} \\ M_{13} \end{pmatrix} \propto \begin{pmatrix} M_{21} \\ M_{22} \\ M_{23} \end{pmatrix} \propto \begin{pmatrix} M_{31} \\ M_{32} \\ M_{33} \end{pmatrix} \tag{3.25}$$

Another way of writing this is

$$V_i = (M_{i3}, -M_{i2}, M_{i1}) \tag{3.26}$$

if and only if the nullity(J) has the dimension one. However, the minor vectors may be of different lengths and hence, weighting is needed.

It is possible to calculate J and the object is to find V that is the eigenvector of J and that has the smallest eigenvalue. If it is assumed that the value of the smallest eigenvalue is approximately zero, V can be estimated because of the fact that it is proportional to minor-vectors that have been described. These are calculated from J and the remaining part is to weight them in some way into the vector V. This weighting is described in the next section.

3.2.5 Finding the velocities

It would be enough to estimate the vector with the largest norm. However if the vectors are weighted instead, a more reliable result would be obtained. The weighting of these vectors is performed with the aid of a weighting factor according to [9]:

$$\beta_i = \frac{M_{1i}}{M_{11}^2 + M_{12}^2 + M_{13}^2} \tag{3.27}$$

This weighting factor gives the most weight to the vector with the largest norm. It is used in the following way:

$$V^T = \beta_1 V_1^T + \beta_2 V_2^T + \beta_3 V_3^T \tag{3.28}$$

V contains the calculated velocities:

$$V \propto \begin{pmatrix} v_x \\ v_y \\ 1 \end{pmatrix} \tag{3.29}$$

To sum up single motion estimation one can say that after some assumptions has been made, it is possible to estimate the velocity from the structure tensor J and V_1, $V2$, and V_3, weight them into V and normalize. The normalization scales the vector in Eq.(3.28) in such a way that the third component gets the value = 1.

3.3 Two motions

The following section will describe the procedure of estimating two motions.

3.3.1 The Derivative Operator for two motions

In the case with two motions, the intensity function f is composed of two layers, $f = f_1 + f_2$, which generally moves with two different velocities - (v_{1x}, v_{1y}) and (v_{2x}, v_{2y}). Looking at Eq.(3.1) and extending that for two motions results in

$$f(x_0, y_0, t_0) = f_1(x_0 + v_{1x}\tau, x_0 + v_{1y}\tau, t+\tau) + f_2(x_0 + v_{2x}\tau, x_0 + v_{2y}\tau, t+\tau) \tag{3.30}$$

The next step is to use the derivative operator on two motions. Two motions mean, according to Eq.(3.31) and Eq.(3.32), two derivative operators.

$$\alpha(v_1) = v_{1x}\frac{\partial}{\partial x} + v_{1y}\frac{\partial}{\partial y} + \frac{\partial}{\partial t} \tag{3.31}$$

$$\alpha(v_2) = v_{2x}\frac{\partial}{\partial x} + v_{2y}\frac{\partial}{\partial y} + \frac{\partial}{\partial t} \tag{3.32}$$

Expanding these terms generates

$$\begin{aligned} \alpha(v_1)\alpha(v_2) &= \\ &= \left(v_{1x}\frac{\partial}{\partial x} + v_{1y}\frac{\partial}{\partial y} + \frac{\partial}{\partial t}\right)\left(v_{2x}\frac{\partial}{\partial x} + v_{2y}\frac{\partial}{\partial y} + \frac{\partial}{\partial t}\right) \end{aligned} \tag{3.33}$$

In Eq.(A.4) it is possible to see that second derivatives appear.
Using the derivative operator on the image generates, for two motions.

$$\alpha(v_1)f_1 = 0, \alpha(v_2)f_2 = 0 \tag{3.34}$$

The operators are commutative (see Appendix A.4). Due to this fact, the equation above can be rewritten:

$$\alpha(v_1)\alpha(v_2)f = 0 \tag{3.35}$$

By expanding Eq.(3.35) it is possible to see that it is zero. Since the operators are commutative it is possible to switch the order of the derivative operators and by some calculation it becomes zero.

$$\begin{aligned} \alpha(v_1)\alpha(v_2)f &= \alpha(v_1)\alpha(v_2)(f_1 + f_2) \\ &= \alpha(v_2)\alpha(v_1)f_1 + \alpha(v_1)\alpha(v_2)f_2 \\ &= \alpha(v_2)0 + \alpha(v_1)0 \\ &= 0 \end{aligned} \tag{3.36}$$

Applying the derivative operator to an image generates:

$$\begin{aligned} v_{1x}v_{2x}f_{xx} + v_{1y}v_{2y}f_{yy} + (v_{1x}v_{2y} + v_{1y}v_{2x})f_{xy} + \\ (v_{1x} + v_{2x})f_{xt} + (v_{1y} + v_{2y})f_{yt} + f_{tt} = 0 \end{aligned} \tag{3.37}$$

Say that the mixed motion coefficients that contain different combinations of the two motions are named:

$$a = v_{1x}v_{2x}, b = v_{1y}v_{2y}, c = v_{1x}v_{2y} + v_{1y}v_{2x}, \quad (3.38)$$
$$d = v_{1x} + v_{2x}, e = v_{1y} + v_{2y}$$

And then replace them in Eq.(3.37) to get

$$af_{xx} + bf_{yy} + cf_{xy} + df_{xt} + ef_{yt} + f_{tt} = 0 \quad (3.39)$$

This is the equivalent of Eq.(3.1), in the case with two motions. This represents a normal vector and a hyper plane. A hyper plane is a planar surface in high dimensional space and thus hard to visualize. The coefficients a, b, c, d and e are unknowns. The coefficients are put in a vector in order to be able to handle the derivatives separately.

$$V = \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ 1 \end{pmatrix} \quad (3.40)$$

3.3.2 The smoothing function

The smoothing function has the same purpose as for single motion estimation in section 3.2.2 - to differentiate the images using a filter. Hence, the smoothing function in Eq.(3.13) is differentiated according to Eq.(3.39), but without the mixed motion coefficients.

$$\alpha(v_1)\alpha(v_2)g \quad (3.41)$$

Six derivatives are obtained.

$$\frac{\partial^2}{\partial x^2}g(x,y,t) \quad (3.42)$$
$$\frac{\partial^2}{\partial y^2}g(x,y,t)$$
$$\frac{\partial^2}{\partial x \partial y}g(x,y,t)$$

$$\frac{\partial^2}{\partial x \partial t} g(x, y, t)$$
$$\frac{\partial^2}{\partial y \partial t} g(x, y, t)$$
$$\frac{\partial^2}{\partial t^2} g(x, y, t)$$

This filter is separable (see section 3.2.2). The filters are convolved with the images, and by rewriting $\alpha(v_1)\alpha(v_2)g * f = 0$ as,

$$LV = 0 \tag{3.43}$$

it is possible to observe that the mixed motion coefficients are handled separately. V comes from Eq.(3.40), and L consists of the filter convolved with the images:

$$L = (g_{xx}, g_{yy}, g_{xy}, g_{xt}, g_{yt}, g_{tt}) * f \tag{3.44}$$

3.3.3 The Structure Tensor

ϵ (Eq 3.45) needs to be minimized in the same manner as in the case with one motion:

$$\epsilon = \int \omega(x, y, t)(L(x, y, t)V)^2 dxdydt \tag{3.45}$$

ϵ is as small as possible when its derivatives are zero. Following the same procedure using minimization and a constraint as in section 3.2.3 generates:

$$\int L(x, y, t)^T L(x, y, t) V \omega(x, y, t) dxdydt = \lambda V \tag{3.46}$$

Supposing that the motion vectors are locally constant, V is taken out of the integral.

$$J = \int L(x, y, t)^T L(x, y, t) \omega(x, y, t) dxdydt \tag{3.47}$$

$$JV = \lambda V \tag{3.48}$$

J is a structure tensor. Six vectors are then calculated from the structure tensor with the help of its minors.

3.3.4 Vectors from minors

The structure tensor for two motions looks like this

$$\begin{pmatrix} J_{11} & J_{12} & J_{13} & J_{14} & J_{15} & J_{16} \\ J_{21} & J_{22} & J_{23} & J_{24} & J_{25} & J_{26} \\ J_{31} & J_{32} & J_{33} & J_{34} & J_{35} & J_{36} \\ J_{41} & J_{42} & J_{43} & J_{44} & J_{45} & J_{46} \\ J_{51} & J_{52} & J_{53} & J_{54} & J_{55} & J_{56} \\ J_{61} & J_{62} & J_{63} & J_{64} & J_{65} & J_{66} \end{pmatrix}$$

It corresponds to Eq.(3.24) in the single motion case. Following the same discussion as in section 3.2.4 and extending Eq.(3.26) to the case with two motions V is retrieved from the minors matrix:

$$V_i^T = (M_{i6}, -M_{i5}, M_{i4}, -M_{i3}, M_{i2}, -M_{i1}) \tag{3.50}$$

V is retrieved if and only if the nullity(J) has the dimension one and consists of mixed motions. These vectors V_i consist of mixed motion coefficients.

3.3.5 Finding the velocities

So far V has been found by solving an eigenvalue problem which in turn has been solved by the "vectors from minors"-method. This V is now depending on the velocity components that need to be estimated -(v_{1x}, v_{1y}) and (v_{2x}, v_{2y}). In this chapter, the velocity components will be found given the elements in V.

The weighting of these vectors is performed in the same manner as in section 3.2.5 with the aid of a weighting factor

$$\beta_i = \frac{M_{1i}}{M_{11}^2 + M_{12}^2 + M_{13}^2 + M_{14}^2 + M_{15}^2 + M_{16}^2} \tag{3.51}$$

This weighting factor is used in the following way:

$$V^T = \beta_1 V_1^T + \beta_2 V_2^T + \beta_3 V_3^T + \beta_4 V_4^T + \beta_5 V_5^T + \beta_6 V_6^T$$

(3.52)

If a second degree polynom Q

$$Q = Z^2 - A_1 Z + A_0 \tag{3.53}$$

that has the roots:

$$\begin{aligned} v_{1x} + jv_{1y} \\ v_{2x} + jv_{2y} \end{aligned} \tag{3.54}$$

is defined, it follows that the coefficients in Q which comes from Eq.(3.40)

$$\begin{aligned} A_0 = a - b + jc \\ A_1 = d + je \end{aligned} \tag{3.55}$$

will be given by simple expressions of the elements in V (which are possible to estimate) according to:

$$\begin{aligned} Z^2 - (v_{1x} + v_{2x} - j(v_{1y} + v_{2y}))Z + v_{1x}v_{2x} - v_{1y}v_{2y} + \\ j(v_{1x}v_{2y} + v_{1y}v_{2x}) = \\ = Z^2 - (d - je)Z + a - b + jc \end{aligned} \tag{3.56}$$

3.4 Confidence measures

A number of assumptions are made in this framework. It is important to quantify these assumptions. Thus, confidence measures provide a useful way to analyze this.

If no motion is present in the image, no further calculations are needed. One way of seeing if there are motion in the image, is to observe the derivatives in the x- and y-directions. If the sum of the lengths of the gradients is zero, no motion is present.

$$H = |f_x| + |f_y| = 0 \tag{3.57}$$

Another confidence measure involves looking at the eigenvalues of the structure tensor. Eq.(3.21) is true if and only if the nullity(J) is one (see Appendix A.3). This means that J needs to have one eigenvalue equal to zero. One way of checking this

is to use a definition of the determinant that says that to obtain the determinant one can multiply all eigenvalues. J has the dimension $n \times n$.

$$K = det(J) = \lambda_1 \lambda_2 ... \lambda_n = 0 \tag{3.58}$$

If there is one eigenvalue equal to zero, this determinant will also be zero. However, there's also a possibility that there are more than one eigenvalue that is equal to zero. And if this is the case, the calculations are not reliable. To check this it is possible to use the following equation:

$$S = \frac{\lambda_1 \lambda_2 ... \lambda_n}{\lambda_1} + \frac{\lambda_1 \lambda_2 ... \lambda_n}{\lambda_2} + ... + \frac{\lambda_1 \lambda_2 ... \lambda_n}{\lambda_n} \tag{3.59}$$

One value is skipped in each step. If this equation is zero there are more than one eigenvalue equal to zero.

In practice, Eq.(3.58) is rarely exactly zero. To work around this it is possible to say that K needs to be as small as possible, and S need to be larger than K. To be able to compare these two confidence measures they need to be normalized.

$$K^{1/n} << S^{1/(n-1)} \tag{3.60}$$

The determination of how these numbers scale to each other might be different in different applications. The trial and error method is applied in this thesis.

Chapter 4

Estimation results on water model images

4.1 Image acquisition details

The images are acquired under the best conditions possible. The first image sequence is acquired with particles only, and then bubbles are added. The images are sub-sampled one time and thereby reducing the motion. The simulation results will only show simulations of particles since no satisfying result of both bubbles and particles have been achieved.

4.2 A walk through of the algorithm

This chapter describes the implementation of the algorithm which is implemented in Matlab.

1. The algorithm reads a sequence of approximately 20 images and transforms them into images with decimal pixel values (see Appendix B.1). The format of the images may be of any image format available in Matlab.

   The images are cropped in order to be able to determine the area of interest (see Appendix B.2), and then sub-sampled with the function subSample (see Appendix B.1). The sub-sampling reduces the image by half in each direction and is done by filtering the image sequence with a

Gaussian filter kernel retrieved from the function fspecial and then by resizing it with the Matlab function resize with the bilinear option. The filtering is an anti-aliasing procedure which reduces the alias distortion in the resulting image. The sub-sampling reduces the pixel motion by half in each direction. All images are added into one image in order to visualize in what way the seeding particles and bubbles move. This image will show the motion pattern of the object in the sequence if the time between the frames is short enough. It is also sub sampled for the purpose of having the total image and the vector map of the same size.

2. The main algorithm filters the images with a differentiated Gaussian filter kernel (Gaussian is optional, any noise reducing filter will do) in order to get the first order partial derivatives (see Appendix B.5). The differentiated Gaussian filter kernel is made by differentiating a Gaussian function and then making it discrete by sampling it into a vector (see Appendix B.4). The σ in Eq(3.13) is the same in all directions. By some testing $\sigma_x = \sigma_y = \sigma_t = 1$ seemed like a good choice due to the fact that the particles are so small. However, this value may vary with area of application.
   This is followed by filtering the first order partial derivatives by the same filter kernel as above in order to get the second order derivatives.

3. The structure tensor for one motion is then calculated according to Eq.(3.47) by convoluting the outer product between the first order partial derivatives by itself and a filter kernel, which is Gaussian in this case (see Appendix B.7).

4. The minors matrix of the structure tensor may then be calculated from the definition in section 3.2.4. When all this is done it is possible to get the confidence measures for every pixel in the image.

5. The first confidence - K - computes the determinant according to Eq.(3.58) of the structure tensor. The second one - H - computes the trace of the structure tensor, which

means adding the derivatives in the x- and y-direction. The third - S - adds the products of all possible combinations of eigenvalues where one eigenvalue is removed from each product, consistent with Eq.(3.59).

6. The subsequent loop (see Appendix B.6) begins with checking if H is larger than ϵ (see Appendix B.9 for a detailed description of epsilon and the confidence measures). If it is not, no motion is present.
   If there is motion present K is compared to S to see if it is smaller. When the second confidence measure is smaller, the vector containing the mixed motion coefficients is calculated from the minors according to Eq.(3.3.5) and Eq.(3.3.5).

7. The first two elements in this vector contain the velocities in the x- and y-directions. They are transformed into a complex value in line with Eq.(3.55) and put in the current coordinates of the first motion field matrix.
   If K is in fact larger than S, the same procedure as above is repeated with the exception that the second order partial derivatives are used and that the roots of Eq.(3.56) need to be calculated from the coefficients of the mixed motion vector.
   The first root is the first complex velocity and the second root is the second complex velocity: These are saved in the first and second motion field matrices.

8. With this loop finished one will end up with two motion field matrices with the same size as the images minus two times the size of the derivative filter since the derivatives are not correct there. The velocities are plotted with the help of the Matlab function Quiver and with all added images as background.

4.2.1 Confidence measures

Tuning of the confidence measure is of great importance in order to achieve satisfying results. The first confidence measure checks if there are any motion present. This is done by checking if the sum of the derivatives in the x- and y-directions is zero. In practice however, it is rarely exactly zero. Therefore ϵ (see

Appendix B.6) is used to set a value where one can say that sums lower than this indicates that there are no motion present. The second confidence measure - K - looks at the possibility that one eigenvalue might be zero (see Appendix 3.4 for details), and the third - S - investigates whether there might be more than one eigenvalue equal to zero. For this to be applicable in practice, one can say that K needs to be smaller than S. The question is how much smaller? In order to be able to compare the values of K and S they need to be normalized in some way. This is achieved by taking the n-th root of K and S, where n is the number of factors. C_1 and C_2 are simply values that are multiplied by S in order to say how much smaller K needs to be than S. It is by no means clear how to determine the value of C_1 and C_2. Here, the algorithm is run through a few times in order to see what values that might be acceptable. This is achieved by making the images binary and investigating if K is smaller than S only where the objects are. If so, the chosen value of C_1 is kept. However, there has been no success in finding a reasonable value for C_2. The algorithm takes about 20 minutes to run on an ordinary PC. The confidence measures used in the simulations below are the following:

$\epsilon = 1$

$C_1 = 1$

$C_2 = 1$

4.3 Estimation results

Estimation results are presented in the following figures. Two images are shown from each selected session. The first two images, figure 4.1 and figure 4.2 are images from a synthetic sequence. The first image of the two shows a white square moving right and a texture moving downward, and the second image is the resulting vector map. The yellow lines are arrows pointing downwards, and the magenta arrows are pointing to the right. This shows that the algorithm works for two motions.
This is followed by image sequences of particles and bubbles. The first image of the two is all original images added together and the second image shows the resulting vector map. These images shows one motion due to the fact that the bubbles are

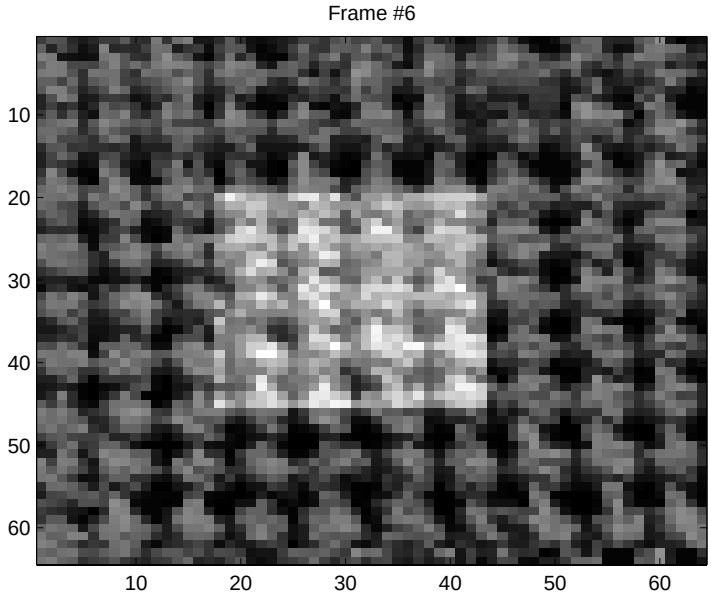

Figure 4.1: A synthetic image

moving too fast. The next sequence, figure 4.3 and figure 4.4 shows a satisfying result. One can observe the motion pattern in figure 4.3 and see that the resulting vector map corresponds to that pattern. It seems to be true to the real flow pattern. The pair of figures 4.5 and 4.6 and the last pair of figures 4.7 and 4.8 are also successful due to the same facts as above. One failed session is shown in figures 4.9 and 4.10. There are no obvious flow pattern in figure 4.9. The resulting vectormap is not very reliable.

4.4 Conclusions

The results show that with a very slow speed - slower than 0.1 m/s (the average speed is approximately 0.3m/s) - it is possible to get results of how the particles are moving. In fact, the accuracy of the resulting vector maps seems to be very high since they follow the real flow pattern of the liquid. The results of the images with both particles and bubbles show no motion. Since the algorithm works for two phase synthetic images, and one phase slowly moving particles on real images, it is likely that

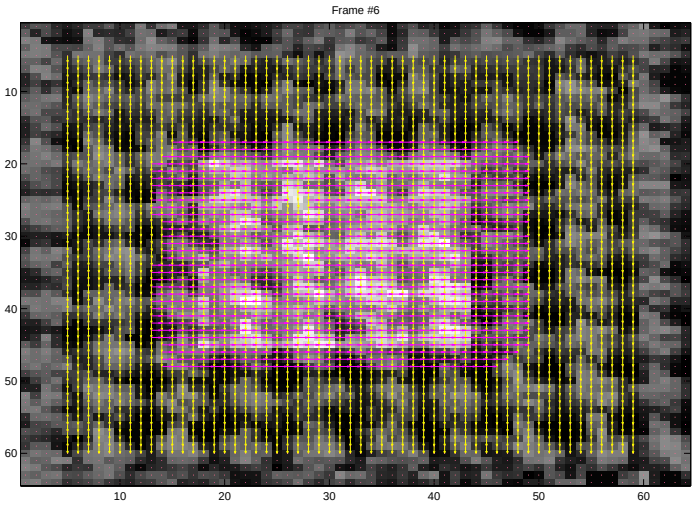

Figure 4.2: Resulting vector map from the synthetic sequence

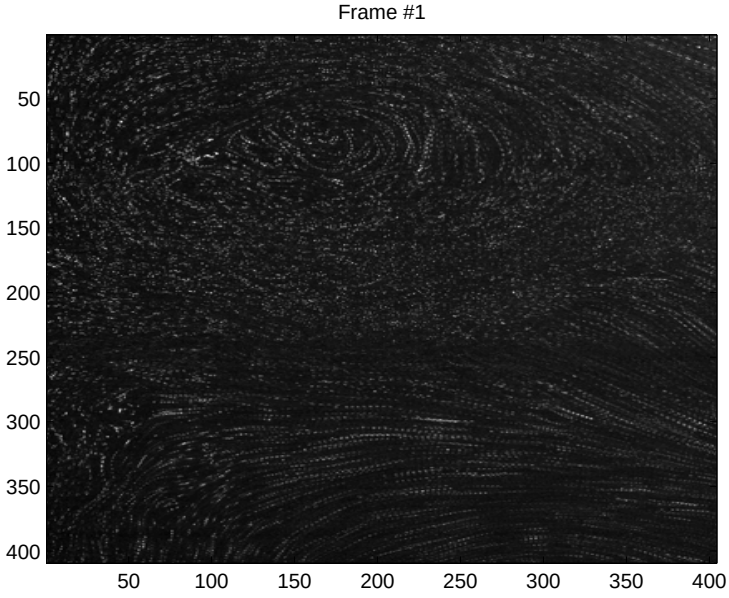

Figure 4.3: Image from water model of particles only

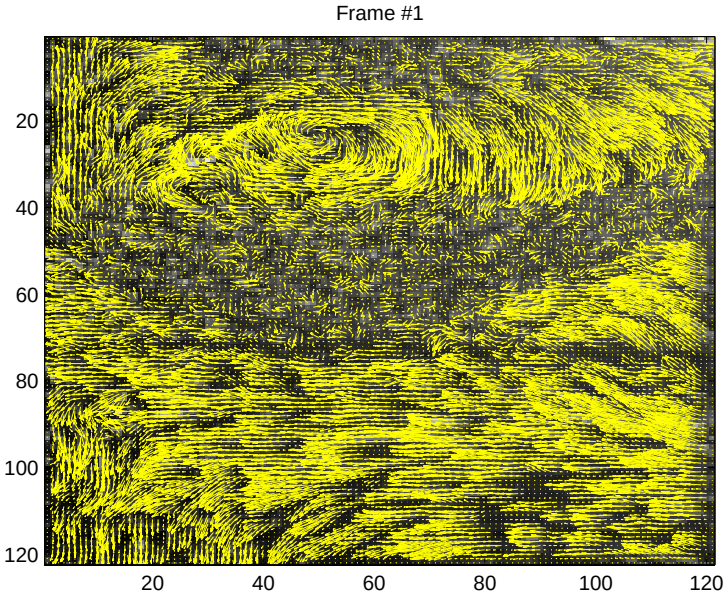

Figure 4.4: Resulting vector map from water model images of particles only

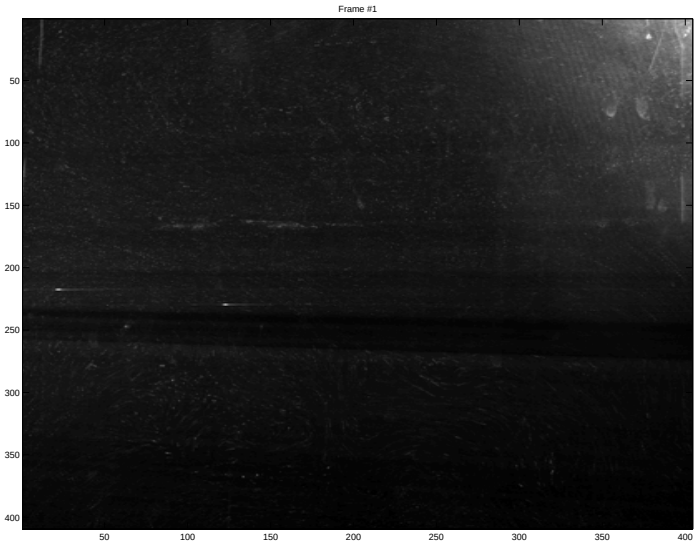

Figure 4.5: Added images of particles with the bubbles filtered out

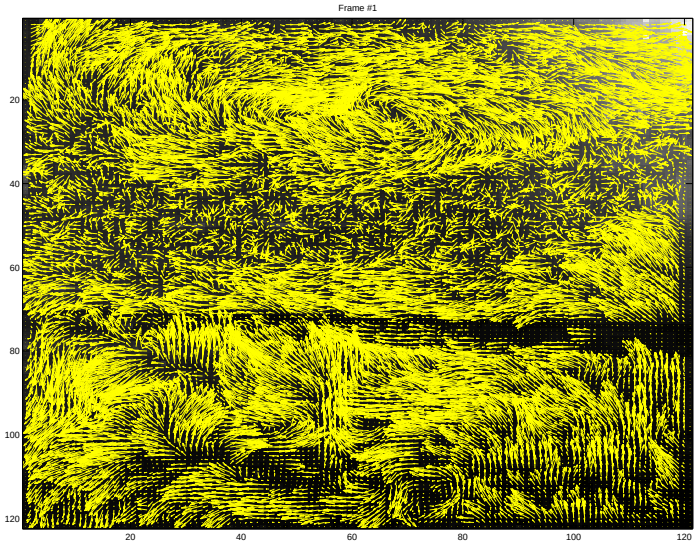

Figure 4.6: Resulting vector map of particles with the bubbles filtered out

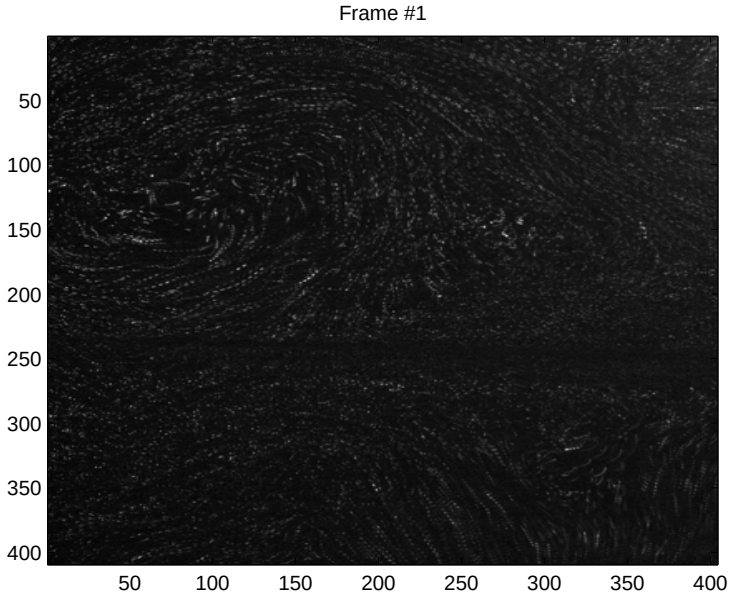

Figure 4.7: Added images of particles with the bubbles filtered out

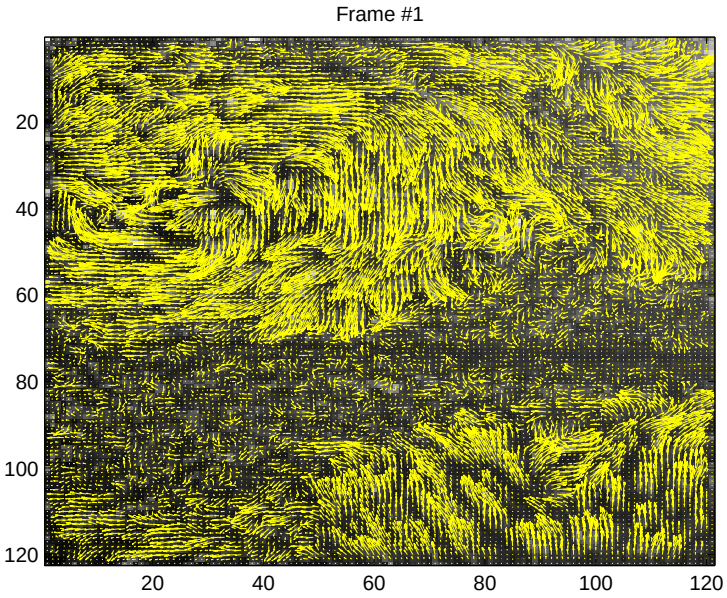

Figure 4.8: Resulting vector map of particles with the bubbles filtered out

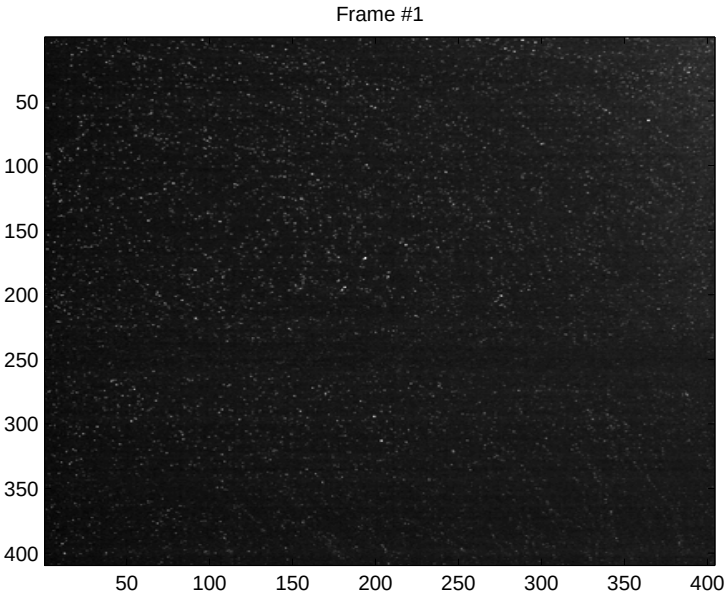

Figure 4.9: Added images of particles with the bubbles filtered out

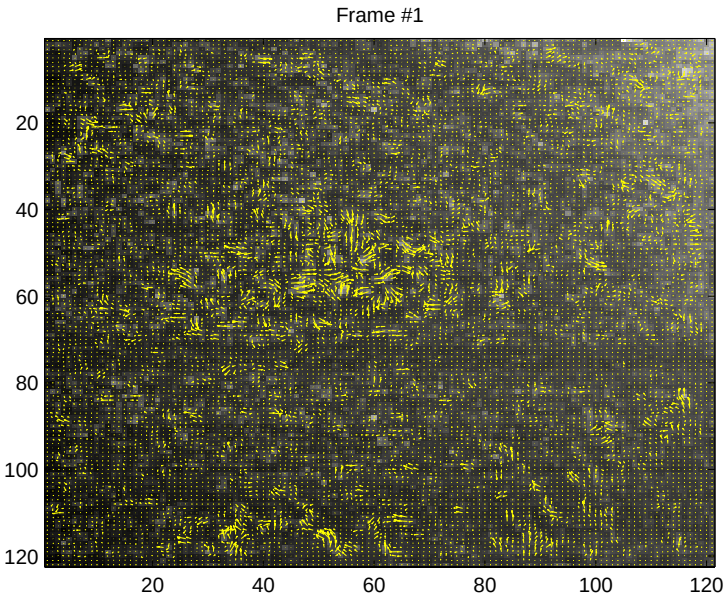

Figure 4.10: Resulting vector map of particles with the bubbles filtered out

the velocity of the faster moving bubbles could be estimated if it was possible to sample faster with the camera. That would allow for faster casting speed and thus give more flexibility in choosing casting speed.

The qualities of the images are decided visually. The original images that are added together gives an indication of what the resulting vector map should look like. As a result of the fact that the flow is not measurable it is difficult to say that each image acquisition session has the same conditions as the next. However it seems as though the quality of the image sequences is quite consistent. Looking at the estimation results for one motion only, three out of four sessions gave a very good result.

Due to all these simulation results above one can say that the algorithm gives quite satisfying results for one phase in flows that are slow.

The overall conclusion is that with the available equipment, "Transparent motions" is a suitable method for estimating one phase slowly moving particles in the water model. This algorithm has a good forecast of estimating two phase motions with new equipment.

Chapter 5

Discussion

The images that are considered in this thesis have a quite high resolution of 1018x1008 pixels and the image sequence may consist of up to 30 images. The high resolution is of great importance because of the small size of the particles and the large image acquisition area, and the large number of images in the sequence is needed in order to achieve as accurate motion estimation as possible. This requires quite a bit of memory in the computer to be able to handle such large image sequences. At the moment, the water flow needs to be incredibly slow for the algorithm to work - much slower than 0.1m/s. This is due to the fact that the particle cannot move very fast on the basis of the fact that the objects in the image cannot move out of the window where the differentiation is performed.

The fastest velocities are required to move at most 10 pixels per frame if the method is going to work. If we have this velocity it is possible to sub-sample the images one time. More sub sampling makes the images too blurry to get any reliable result. This generates images with a slower velocity.

The speed of the water obviously influences the flow pattern. With a very low speed of the water, it is not representative to how the steel flows in the steel casting model. With this in mind the water must have at least a certain speed. The bubbles also influence the flow pattern. If the bubbles are too few, a reverse flow appears which is not representative either. This means that to accomplish consistent results, the algorithm need to be accurate for measuring bubble and water motion under representative conditions. A camera with a shorter sampling interval is needed in order to carry out this task.

5.1 Limitations of the camera

The limitation of the camera is obviously the image sampling interval capacity. It is able to acquire images with a 37ms interval, which means less than 30 images per second. With the water floating with an average speed of 0.3 m/s the average movement will be approximately 25 pixels long. With this average speed and the available resolution, the number of frames per second need to be at least 70, which gives a movement of approximately 10 pixels per frame. It is then possible to sub-sample the image one time in order to make the speed half as fast.

5.2 Limitations of the algorithm

The algorithm has some limitations in the way of how the derivative filters work and the size of the structure tensor. The derivatives are calculated in a local neighborhood. This means that the objects cannot move out of this neighborhood from one frame to another in order to be able to compute the correct velocity. Another limitation that may be considered is the fact that there may be more motions than two present due to the fact that the water is rarely drained. With this in mind there might be small objects in the water that contributes to the motion estimation. The particle motion and the bubble motion are not distinguishable from one and other which means that the only way to tell which motion belongs to which layer is purely visual and based on guesses.

5.3 Fulfillment of goals

The goals that were setup for this thesis are fulfilled. The theory of "Transparent motions" has been investigated, an algorithm was developed and some testing was performed.

Appendix A

Theoretical Review

To understand this thesis it might be useful with some theoretical review. The following sections involve facts that are used in the report.

A.1 Intensity

Intensity is synonymous with the term gray level. The gray level of an image is the intensity of a monochrome image. A mono chromic image is void of color.

A.2 Sub Sampling

Sub sampling is a geometric transformation of the image. When the image is scaled into a smaller size, the movement in the images decreases.

The reduction of the image is performed by either replacing original pixels in a neighborhood by the value of one arbitrarily chosen pixel in the image, or by interpolation. There are different ways to interpolate. The most commonly used are linear, bilinear and bi cubic interpolation [2].

A.3 Nullity and rank

The Rank of a matrix is defined as the number of eigenvalues not equal to zero.

The nullity of a matrix is all eigenvectors that satisfy the equation

$$Jx = 0 \tag{A.1}$$

A.4 Commutative

For the interested reader it is shown below together with Eq.(3.34) that $\alpha(v_1)\alpha(v_2)$ is commutative.

$$\begin{aligned} \alpha(v_2)\alpha(v_1) &= \\ &= \left(v_{2x}\frac{\partial}{\partial x} + v_{2y}\frac{\partial}{\partial y} + \frac{\partial}{\partial t}\right)\left(v_{1x}\frac{\partial}{\partial x} + v_{1y}\frac{\partial}{\partial y} + \frac{\partial}{\partial t}\right) \end{aligned} \tag{A.2}$$

$$\begin{aligned} v_{2x}v_{1x}\left(\frac{\partial^2}{\partial x^2}\right) + v_{2y}v_{1y}\left(\frac{\partial^2}{\partial y^2}\right) + (v_{2x}v_{1y} + v_{2y}v_{1x})\left(\frac{\partial^2}{\partial x \partial y}\right) \\ +(v_{2x} + v_{1x})\left(\frac{\partial^2}{\partial x \partial t}\right) + (v_{2y} + v_{1y})\left(\frac{\partial^2}{\partial y \partial t}\right) + \left(\frac{\partial^2}{\partial t^2}\right) \end{aligned} \tag{A.3}$$

$$\begin{aligned} v_{2x}v_{1x}f_{xx} + v_{2y}v_{1y}f_{yy} + (v_{2x}v_{1y} + v_{2y}v_{1x})f_{xy} + \\ (v_{2x} + v_{1x})f_{xt} + (v_{2y} + v_{1y})f_{yt} + f_{tt} \end{aligned} \tag{A.4}$$

Which is the same as Eq(3.37)

$$\begin{aligned} v_{1x}v_{2x}f_{xx} + v_{1y}v_{2y}f_{yy} + (v_{1x}v_{2y} + v_{1y}v_{2x})f_{xy} + \\ (v_{1x} + v_{2x})f_{xt} + (v_{1y} + v_{2y})f_{yt} + f_{tt} \end{aligned} \tag{A.5}$$

A.5 Proofing Eq.(3.12)

$$\alpha(v)(g * f) = \alpha(v)\left(\int g(x', y', t')f(x - x', y - y', t - t')dx'dy'dt'\right) \tag{A.6}$$

$\alpha(v)$ is a derivative operator and in general, it is allowed to switch the order of differentiation and integration. If this is

performed on the right hand side of the equation above, the following is obtained:

$$= \left(\int \alpha(v) g(x', y', t') f(x - x', y - y', t - t') dx' dy' dt' \right) \tag{A.7}$$

$$\begin{aligned} (\alpha(v)g) * f &= \left(\int \alpha(v) g(x', y', t') f(x - x', y - y', t - t') dx' dy' dt' \right) \\ &= \alpha(v)(g * f) \end{aligned} \tag{A.8}$$

A.6 Proofing Eq.(3.10)

$$f(x_0, y_0, t_0) = f(x_0 + v_x\tau, y_0 + v_y\tau, t_0 + v_t\tau)$$

$$(g * f)(x_0, y_0, t_0) = \int g(\lambda_x, \lambda_y, \lambda_t) f(x_0 - \lambda_x, y_0 - \lambda_y, t_0 - \lambda_t) d\lambda_x d\lambda_y d\lambda_t$$

$$(g * f)(x_0 + v_x\tau, y_0 + v_y\tau, t_0 + v_t\tau) = \int AB d\lambda_x d\lambda_y d\lambda_t$$

$$\begin{aligned} A &= g(\lambda_x, \lambda_y, \lambda_t) \\ B &= f(x_0 + v_x\tau - \lambda_x, y_0 + v_y\tau - \lambda_y, t_0 + v_t\tau - \lambda_t) = f(x_0 - \lambda_x, y_0 - \lambda_y, t_0 - \lambda_t) \end{aligned}$$

The right hand side of the expression above is obtained by using the expression on the first row.

$$\begin{aligned} \int AB d\lambda_x d\lambda_y d\lambda_t &= \int g(\lambda_x, \lambda_y, \lambda_t) f(x_0 - \lambda_x, y_0 - \lambda_y, t_0 - \lambda_t) d\lambda_x d\lambda_y d\lambda_t \\ &= (g * f)(x_0, y_0, t_0) \end{aligned}$$

Appendix B

The Code for Transparent Motions

B.1 Main

```
function Main()
  %% Single and transparent motion estimation
  %%
  %%******     Synthetic Sequence   ********************************
%   %% load two images

  f9  = double(rgb2gray(imread('Fabric.0009.tif')));
  f11 = double(rgb2gray(imread('Fabric.0011.tif')));

  %% subsample one image

  filter=fspecial('gaussian',[3,3]);
  f11=convn(f11,filter,'same');
  f11=f11(1:2:end,1:2:end);

  %% select some image parts

  f9 = f9(1:64,1:64);
  f11=f11(20:45,20:45);

  %% construct new image
```

```
Im3 = zeros(64);
Im3(20:45,20:45) = f11;

%% generate test sequence

S1(:,:,8:-1:1)=MoveSeq(f9,[0,-1],8);
S1(:,:,8:16)=MoveSeq(f9,[0,1],9);

S2(:,:,8:-1:1)=MoveSeq(Im3,[-1,0],8);
S2(:,:,8:16)=MoveSeq(Im3,[1,0],9);

%% Add both sequences

imageSequence=S1+S2;

%% Show image sequence

disp('Click mouse button to scroll through the image sequence');

DisplaySequence(imageSequence);

%%******      Real Sequence   ***********************************

filename = 'E:\exjobb - Lella\bilder\050711\File0';
```

```
cropTop = 200;
cropBottom = 200;
cropRight = 200;
cropLeft = 200;
total = zeros(1018-(cropTop+cropBottom)+1,1008-(cropRight+cropLeft)+1);
indexet = 1;

%% --------- read images ----------------------------
for i = 12:12
    for j = 2:9
        %%% ************************************** IMAGE A *****
        %%% ************************************** IMAGE A *****
        %%% ************************************** IMAGE A *****

        image = imread([filename num2str(i) num2str(j) '_a.bmp']);
        image = double(image);
        imCropped = cropImages(image,cropTop,cropBottom,cropLeft,cropRight);
        grayImage = imCropped/max(imCropped(:));
        imageSequence(:,:,indexet) = imCropped;
        total(:,:)=total(:,:)+grayImage;
        indexet = indexet+1;

        %%% ************************************** IMAGE B *****
        %%% ************************************** IMAGE B *****
```

```
%%% ************************************************

    image = imread([filename num2str(i) num2str(j) '_b.bmp']);
    image = double(image);
    imCropped = cropImages(image,cropTop,cropBottom,cropLeft,cropRight);
    grayImage = imCropped/max(imCropped(:));
    imageSequence(:,:,indexet) = imCropped;
    total(:,:)=total(:,:)+grayImage;
    indexet = indexet+1;
  end
end
pack
clear image imCropped;

imageSequence = subSample(imageSequence);
total = imresize(total,0.5,'bilinear');
% DisplaySequence(imageSequence);

%%*****************************************************

[height width depth] = size(imageSequence);
  epsilon      = 1;
c1             = 1;      %for synthetic sequence, a value between 0 and 1
c2             = 1;      %for synthetic sequence
```

```
Start_x     = 5;          % Do not compute motions at the image border
End_x       = width-5;      % and the first and last frames, because
Start_y     = 5;          % the derivatives are not correct there!!!
End_y       = height-5;
Start_t     = 3;
End_t       = depth-3;

[Utot,Vtot]=TransparentMotion(imageSequence,epsilon,c1,c2, Start_x,End_x, Start_y,End_y,...
    Start_t, End_t);

DisplaySequence(imageSequence,Utot,Vtot);

    Utotabs = abs(Utot);
%  Vtotabs = abs(Vtot);

for t = 1:depth
   for i = 1:height
      for j =1:width
         if abs(Utotabs(i,j,t))>1
            Utot(i,j,t)= 0;
         end
      end
   end
end
```

```
%
% ************************UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU

averageFilter = fspecial('average');
averageImagesUtot = imfilter(Utot, averageFilter, 'conv');
t1 = toc/60
DisplaySequence(total,sum(averageImagesUtot,3));
averageImagesSmallUtot = imresize(averageImagesUtot,0.3,'bilinear');
total = imresize(total,0.3,'bilinear');
DisplaySequence(total,sum(averageImagesSmallUtot,3));


return
%%-----------------------------------------------------------------
function [subimageSequence]=subSample(imageSequence)

kernel = fspecial('gaussian');
subimageSequence = convn(imageSequence,Make_3D_Filter(kernel(:,1),kernel(:,2),kernel(:,3)),'same');
subimageSequence = imresize(imageSequence,0.5,'bilinear');

return
%%-----------------------------------------------------------------
```

B.2 cropImages

```
function b = cropImages(theImage, crop_left, crop_right, crop_top, crop_bottom)
[row col depth]= size(theImage);

if depth ==3
theImageR = theImage(crop_left:(row-crop_right), crop_top:(col-crop_bottom),1);
theImageG = theImage(crop_left:(row-crop_right), crop_top:(col-crop_bottom),2);
theImageB = theImage(crop_left:(row-crop_right), crop_top:(col-crop_bottom),3);

b(:,:,1) = theImageR;
b(:,:,2) = theImageG;
b(:,:,3) = theImageB;

else
    b = theImage(crop_left:(row-crop_right),crop_top:(col-crop_bottom));
end

return
```

B.3 TransparentMotion

```
%% This program is corresponds hierarchical algorithm proposed in the
%% paper:
%%
%% C. Mota, I. Stuke, and E. Barth,
%% "Analytic Solutions for Multiple Motions",
%% In Proceedings of the International Conference on Image Processing,
%% pages 917-920, 2001.


function [Utot,Vtot]=TransparentMotion(imageSequence,epsilon,c1,c2,Start_x,End_x,Start_y,End_y,...
    Start_t,End_t)
disp('<<TransparentMotions');

[height width depth] = size(imageSequence);

%calculate gaussian derivatives and convert into filters
[g1Diff1 g1Diff2 g1Diff3] = calcGaussianDerivativeFilter(3,3,3);

% calculate first and second order partial derivatives
[Lx Ly Lt] = calculateL(g1Diff1, g1Diff2, g1Diff3, imageSequence, 1);
[Lxx Lxy Lxt Lyy Lyt Ltt] = calculateL(g1Diff1, g1Diff2, g1Diff3, ...
                                imageSequence,2,...
                                    Lx,Ly,Lt);
```

```
%calculate velocities
[Utot Vtot] = calculateVelocities(Lx, Ly, Lt,...
                                  Lxx, Lxy, Lxt, Lyy, Lyt, Ltt,...
                                  epsilon,c1,c2,...
                                  Start_x,End_x,Start_y,End_y,Start_t,End_t , ...
                                  imageSequence);

return
```

B.4 calcGaussianDerivativeFilter

```
function [varargout]= ...
calcGaussianDerivativeFilter(filterSizeX, filterSizeY,filterSizeT)
%calculates the derivatives of a gaussian function
disp('<< calcGaussianDerivativeFilter');


syms x y t;
syms sigmaX sigmaY sigmaT;

sigma = 1;

filterSizeX = (filterSizeX-1)/2;
```

```
filterSizeY = (filterSizeY-1)/2;
filterSizeT = (filterSizeT-1)/2;

x = -filterSizeX:filterSizeX;
y(1:3,:) = -filterSizeY:filterSizeY;
t(:,:,1:3) = -filterSizeT:filterSizeT;

g1Diff1 = -x./sigma^2.*exp(-x.^2/(2*sigma^2));
g1Diff2 = -y./sigma^2.*exp(-y.^2/(2*sigma^2));
g1Diff3 = -t./sigma^2.*exp(-t.^2/(2*sigma^2));
g1Diff1=g1Diff1/sum(abs(g1Diff1));
g1Diff2=g1Diff2/sum(abs(g1Diff2));
g1Diff3=g1Diff3/sum(abs(g1Diff3));

varargout = {g1Diff1, g1Diff2, g1Diff3};

return
```

B.5 calculateL

```
function [varargout]= calculateL(g1Diff1, g1Diff2, g1Diff3,imGray, NoOfMotions, Lx, Ly, Lt)
```

```
disp('<< calculateL');

[height width depth] = size(imGray);

if NoOfMotions == 1
    L11 = zeros(height, width, depth);
    L12 = zeros(height, width, depth);
    L13 = zeros(height, width, depth);

    L11 = convn(imGray, g1Diff1,'same');
    L12 = convn(imGray, g1Diff2,'same');
    L13 = convn(imGray, g1Diff3, 'same');
    varargout ={L11, L12, L13};
else
    L21 = zeros(height, width, depth);
    L22 = zeros(height, width, depth);
    L23 = zeros(height, width, depth);
    L24 = zeros(height, width, depth);
    L25 = zeros(height, width, depth);
    L26 = zeros(height, width, depth);

    L21 = convn(Lx, g1Diff1, 'same');
```

```
    L22 = convn(Ly, g1Diff1, 'same');
    L23 = convn(Lt, g1Diff1, 'same');
    L24 = convn(Ly, g1Diff2, 'same');
    L25 = convn(Lt, g1Diff2, 'same');
    L26 = convn(Lt, g1Diff3, 'same');

    varargout ={L21, L22, L23, L24, L25, L26};
end
return
```

B.6 calculateVelocities

```
function [Utot Vtot]=calculateVelocities(Lx, Ly, Lt,...
                                         Lxx, Lxy, Lxt, Lyy, Lyt, Ltt,...
                                         epsilon,c1,c2,...
                                         Start_x,End_x,Start_y,End_y,Start_t,End_t , ...
                                         imageSequence);
disp('<<calculateVelocities');

J1 = zeros(3,3);
J2 = zeros(6,6);
theMinors1 = zeros(3,3);
theMinors2 = zeros(6,6);
```

```
Utot    = complex(zeros(size(imageSequence)),zeros(size(imageSequence)));
Vtot    = Utot;
[height width depth] = size(imageSequence);
disp(['from' Start_t ' to ' End_t]);

[J1tot J2tot]= calculateTensors(Lx, Ly, Lt, Lxx, Lyy, Lxy, Lxt, Lyt, Ltt);

%% Main Loop
%%------------------------------
%% Computes the motions vectors for all pixels
%%
for t =Start_t:End_t,
  t
  for y =Start_y:End_y,
    for x =Start_x:End_x,

      J1 = [J1tot{1,1}(y,x,t) J1tot{1,2}(y,x,t) J1tot{1,3}(y,x,t); ...
            J1tot{2,1}(y,x,t) J1tot{2,2}(y,x,t) J1tot{2,3}(y,x,t); ...
            J1tot{3,1}(y,x,t) J1tot{3,2}(y,x,t) J1tot{3,3}(y,x,t)];

            [H1 K1 S1] = confidence(J1);

            if(H1 > epsilon)
```

```
if(K1^(1/3) < c1*sqrt(S1))

    theMinors1 = minorsMatrix1(J1);

    V1 = [theMinors1(1,3) -theMinors1(1,2) theMinors1(1,1)];
    V2 = [theMinors1(2,3) -theMinors1(2,2) theMinors1(2,1)];
    V3 = [theMinors1(3,3) -theMinors1(3,2) theMinors1(3,1)];

    denominator = theMinors1(1,1)^2+theMinors1(1,2)^2+theMinors1(1,3)^2;

    if(denominator == 0)
      denominator = 0.001;
    end

    alfa1 = theMinors1(1,1)/denominator;
    alfa2 = theMinors1(1,2)/denominator;
    alfa3 = theMinors1(1,3)/denominator;

    v = alfa1*V1+alfa2*V2+alfa3*V3;

    Utot(y,x,t)=complex(v(1),v(2));

    else
```

```
    %% Compute J2
J2 = ...
[J2tot{1,1}(y,x,t) J2tot{1,2}(y,x,t) J2tot{1,3}(y,x,t) J2tot{1,4}(y,x,t) J2tot{1,5}(y,x,t) J2tot{1,6}(y,x,t);..
 J2tot{2,1}(y,x,t) J2tot{2,2}(y,x,t) J2tot{2,3}(y,x,t) J2tot{2,4}(y,x,t) J2tot{2,5}(y,x,t) J2tot{2,6}(y,x,t);...
 J2tot{3,1}(y,x,t) J2tot{3,2}(y,x,t) J2tot{3,3}(y,x,t) J2tot{3,4}(y,x,t) J2tot{3,5}(y,x,t) J2tot{3,6}(y,x,t);...
 J2tot{4,1}(y,x,t) J2tot{4,2}(y,x,t) J2tot{4,3}(y,x,t) J2tot{4,4}(y,x,t) J2tot{4,5}(y,x,t) J2tot{4,6}(y,x,t);...
 J2tot{5,1}(y,x,t) J2tot{5,2}(y,x,t) J2tot{5,3}(y,x,t) J2tot{5,4}(y,x,t) J2tot{5,5}(y,x,t) J2tot{5,6}(y,x,t);...
 J2tot{6,1}(y,x,t) J2tot{6,2}(y,x,t) J2tot{6,3}(y,x,t) J2tot{6,4}(y,x,t) J2tot{6,5}(y,x,t) J2tot{6,6}(y,x,t)];

theMinors2 =minorsMatrix2(J2);
[K2 S2] = confidence(J2);

if ( K2^(1/6) < c2*S2^(1/5))

    alfa  =  theMinors2(1,1:6)';
    v  =  theMinors2*alfa;

    %% Solve the complex polynomial

    a  =  v(6)/v(1);
    b  =  v(5)/v(1);
    c  =  v(4)/v(1);
    d  =  v(3)/v(1);
```

```
        e  =  v(2)/v(1);
        A1  = d+j*e;
        A0  = a -b +j*c;
         x1 = 1/2*(A1-sqrt(-4*A0+A1^2));
         x2 = 1/2*(A1+sqrt(-4*A0+A1^2));

                Utot(y,x,t)=x1;
                 Vtot(y,x,t)=x2;
                 warning off last
              end
           end
        end
     end
  end
end
return
```

B.7 calculateTensors

```
function [J1tot J2tot]= ...
calculateTensors(Lx, Ly, Lt, Lxx, Lyy, Lxy, Lxt, Lyt, Ltt)

% integration filter for the tensors
omega = fspecial('gaussian',[3 1],1);
omega = Make_3D_Filter(omega,omega,omega);

% calculate structure tensor for ...
%one motion using first degree partial
% derivatives
J1tot=   ...
{convn(Lx.*Lx,omega,'same') , convn(Lx.*Ly,omega,'same') , convn(Lx.*Lt,omega,'same');...
convn(Ly.*Lx,omega,'same') , convn(Ly.*Ly,omega,'same') , convn(Ly.*Lt,omega,'same');...
convn(Lt.*Lx,omega,'same') , convn(Lt.*Ly,omega,'same') , convn(Lt.*Lt,omega,'same')};

% calculate structure tensor for two motions using second degree partial
% derivatives
%
```

J2tot = convn(Lxx.*Lxx,omega,'same'), convn(Lxx.*Lyy,omega,'same'), convn(Lxx.*Lxy,omega,'same'),... convn(Lxx.*Lxt,omega,'san
convn(Lxx.*Lyt,omega,'same'), convn(Lxx.*Ltt,omega,'same');... convn(Lxx.*Lyy,omega,'same'), convn(Lyy.*Lyy,omega,'same'),
convn(Lyy.*Lxy,omega,'same'),... convn(Lyy.*Lxt,omega,'same'), convn(Lyy.*Lyt,omega,'same'), convn(Lyy.*Ltt,omega,'same');...

convn(Lxx.*Lxy,omega,'same'), convn(Lxy.*Lyy,omega,'same'), convn(Lxy.*Lxy,omega,'same'),... convn(Lxy.*Lxt,omega,'same'),
convn(Lxy.*Lyt,omega,'same'), convn(Lxy.*Ltt,omega,'same');... convn(Lxx.*Lxt,omega,'same'), convn(Lxt.*Lyy,omega,'same'),
convn(Lxt.*Lxy,omega,'same'),... convn(Lxt.*Lxt,omega,'same'), convn(Lxt.*Lyt,omega,'same'), convn(Lxt.*Ltt,omega,'same');...
convn(Lxx.*Lyt,omega,'same'), convn(Lyt.*Lyy,omega,'same'), convn(Lyt.*Lxy,omega,'same'),... convn(Lyt.*Lxt,omega,'same'),
convn(Lyt.*Lyt,omega,'same'), convn(Lyt.*Ltt,omega,'same');... convn(Lxx.*Ltt,omega,'same'), convn(Ltt.*Lyy,omega,'same'),
convn(Ltt.*Lxy,omega,'same'),... convn(Ltt.*Lxt,omega,'same'), convn(Ltt.*Lyt,omega,'same'), convn(Ltt.*Ltt,omega,'same');
return

B.8 Make 3D Filter

```
function f=Make_3D_Filter(f1,f2,f3)

%% This function generates a 3D Filter form three 1D filters.

for i = 1:length(f1),
    for j = 1:length(f2),
        for k = 1:length(f3),
            f(i,j,k)=f1(i)*f2(j)*f3(k);
        end
    end
end
```

```
return
```

B.9 confidence

```
function [varargout]= confidence(J)

[m,n] = size(J);

K = det(J);

if m == 3
    H = J(1,1)+J(2,2);
    S = (1/(m-1))*(Minor(J,1,1)+Minor(J,2,2)+Minor(J,3,3));
    varargout={H,K,S};
else
    S = (1/(m-1))*(Minor(J,1,1)+Minor(J,2,2)+Minor(J,3,3)+Minor(J,4,4)+Minor(J,5,5)+Minor(J,6,6));
    varargout={K,S};
end

return
```

B.10 minorsMatrix1

```
 function theMatrix = minorsMatrix1(A)
%MINOR
%Gives submatrix and minors of an element of a matrix.
%Calling format: minors of A.

[p,q] = size(A);
matrisen = zeros(p,q);

for ipoint = 1:p
    for jpoint = 1:q
        B = A;
        B(p+1-ipoint,:)=[];
        B(:,q+1-jpoint)=[];
        matrisen(ipoint,jpoint)= det(B);%(-1)^(ipoint+jpoint)**det(B);
    end
end

theMatrix = matrisen;
```

```
return
```

Bibliography

[1] http://ccc.me.uiuc.edu/.

[2] http://homepages.inf.ed.ac.uk/rbf/hipr2/scale.htm.

[3] http://mathworld.wolfram.com/minor.html.

[4] http://www.cs.cf.ac.uk/dave/vision$_{l}ecture/node45.html$.

[5] http://kogs-www.informatik.uni-hamburg.de/ koethe/papers/structuretensor.pdf, 07 2005.

[6] http://www.dantecdynamics.com, 07 2005.

[7] Berndt Jähne. Digital image processing. 2002.

[8] P Löfgren. and S Kollberg. Hastighetsbegränsning i stränggjutningen? *Bergsmannen*, 2004.

[9] C. Mota, I. Stuke, and E. Barth. Analytic solutions for multiple motions. In *International Conference on Image Processing*, 2001.

På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/