Visual Motion Estimation based on Motion Blur Interpretation

Ioannis Rekleitis
School of Computer Science
McGill University, Montreal

A Thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfilment of the requirements for the degree of M.Sc. in Computer Science.

Abstract

When the relative velocity between the different objects in a scene and the camera is relative large – compared with the camera's exposure time – in the resulting image we have a distortion called motion blur. In the past, a lot of algorithms have been proposed for estimating the relative velocity from one or, most of the time, more images. The motion blur is generally considered an extra source of noise and is eliminated, or is assumed nonexistent. Unlike most of these approaches, it is feasible to estimate the Optical Flow map using only the information *encoded* in the motion blur. This thesis presents an algorithm that estimates the velocity vector of an image patch using the motion blur only, in two steps. The information used for the estimation of the velocity vectors is extracted from the frequency domain, and the most computationally expensive operation is the Fast Fourier Transform that transforms the image from the spatial to the frequency domain. Consequently, the complexity of the algorithm is bound by this operation into O($nlog(n)$). The first step consists of using the response of a family of steerable filters applied on the *log* of the Power Spectrum in order to calculate the orientation of the velocity vector. The second step uses a technique called *Cepstral Analysis*. More precisely, the *log* power spectrum is treated as another signal and we examine the Inverse Fourier Transform of it in order to estimate the magnitude of the velocity vector. Experiments have been conducted on artificially blurred images and with real world data, and an error analysis on these results is also presented.

Résumé

Lorsque la vitesse relative entre plusieurs objets dans une scène et la caméra est élevée – en comparaison avec le temps de pose – nous retrouvons dans l'image une distorsion communément appelée "Flou de Mouvement". Plusieurs algorithmes ont été créés afin d'estimer la vitesse relative à partir d'une ou de plusieurs images. Généralement le "Flou de Mouvement" est considéré comme une source de bruit que nous devons éliminer ou ignorer. Il est possible d' estimer le carte du Flot optique en utilisant seulement l'information contenue dans le "Flou de Mouvement". Cette thèse présente un algorithme qui détermine le vecteur vitesse d'une partie d'image en utilisant le "Flou de Mouvement" en deux étapes. L'information fréquentielle est obtenue par une transformée de Fourier rapide. Ceci limite la complexité de l'algorithme à O($nlog(n)$). La première étape consiste à utiliser le résultat d'une famille de filtres adaptatifs sur le logarithme du spectre de puissance afin de calculer l'orientation du vecteur de vitesse. La deuxième étape utilise une technique nommée analyse "cepstrale". Dans ce cas le logarithme du spectre de puissance est considéré comme un signal que nous analysons par le biais d'une transformée de Fourier inverse pour déterminer l'amplitude du vecteur de vitesse. Des expériences ont été réalisées sur des images synthétiques et sur des images réelles. Une analyse de l'erreur des résultats obtenus est présentée.

Acknowledgements

I would like to thank everyone who helped me during the development of my thesis. First of all, my supervisor David Jones for the useful insights in computational vision he provided during his supervision, along with valuable pointers where to search for information. I would like also to thank my co-supervisor Godfried Toussaint who through his courses enhanced my geometry knowledge and show me to always search for the strange counterexamples for my algorithm. Finally I would like to thank my fellow graduate students and especially David Lamb for his valuable advice in signal processing and the Fourier Transform intricacies.

I am thankful to the people at the school of Computer Science for their contribution in my education and their help during my adaptation in Montreal. Among them a special thanks goes to the graduate secretary L. Harper for her advice to cope with McGill bureaucracy and its countless deadlines, and to my fellow graduate student and friend Kostas Kontogiannis for his support and advice through my staying in Canada. I owe a lot to L. Solomon for her help over the English part of my thesis.

Most of all I am grateful to my family that supported me any time I was in need, always standing by me, guiding me into searching for answers and striving for a better tomorrow.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

One of the fundamental problems in early Computer Vision is the measurement of motion in an image, frequently called optical flow. In many cases when a scene is observed by a camera there exists motion, created either by the movement of the camera or by the independent movement of objects in the scene. In both cases, the goal is to assign a 3D velocity vector to each visible point in the scene; such an assignment is called the *velocity map*. In general it is impossible to infer from one view the 3D velocity map; however, most motion estimation algorithms calculate the projection of the velocity map onto the imaging surface. A large number of different algorithms have been developed in order to solve this problem.

The problem of estimating the optical flow has received much attention because of its many different applications. Tasks such as passive scene interpretation, image segmentation, surface structure reconstruction, inference of egomotion, and active navigation, all use optical flow as input information.

Until now, most motion estimation algorithms considered optical flow with displacements of only a few pixels per frame. This approach limits the applications to slower motions and fails to seriously address the issue of motion blur; moreover, it works on images that are considered to be taken with infinitely small exposure time, more or less in a "stop and shoot" approach, which limits the real time applications. Also, most of these algorithms work on a series of images by calculating the displacement of every pixel from image to image, ignoring any information about motion that exists within each single image.

In this thesis we have developed and evaluated a new approach to the problem of visual motion estimation. The algorithm we have developed is based on interpreting the cue of *motion blur* to estimate the optical flow field in a *single image*. A key observation is that motion blur introduces a certain structure, a ripple, in the Fourier transform that can be detected and quantified using a modified form of *cepstral analysis*. Unlike classical approaches to visual motion analysis that rely upon operators tuned to specific spatial and temporal frequencies at specific orientations, our new approach making use of all the information that can be gathered from a patch of the image and is thus quite robust.

The first step in our motion blur analysis is to compute the log power spectrum of a local image patch. Motion blur leads to a tell-tale ripple, centred at the origin, with orientation perpendicular to the orientation of the velocity vector. This orientation can be reliably determined, even in the presence of noise, using a steerable second Gaussian derivative filter. The magnitude of the velocity, which is related to the period of the ripple, can then be determined by first collapsing the log spectrum data into a 1-D vector, and then performing a second Fourier transform, to yield the *cepstrum*, in which the magnitude of the velocity is clearly identified by a negative peak. The computational complexity of this algorithm is bounded by the Fast Fourier Transform operation, which is $O(n \log n)$, where n is the number of pixels in the image patch. Applying this analysis throughout the image provides an estimation of the complete optical flow field.

The structure of this thesis is as follows. In Chapter 2 we describe the problem of motion estimation in general; review the work that has already been done, along with a brief description of the major existing algorithms; then we analyse the problem as it exists with the appearance of motion blur. The solution to the optical flow estimation problem under the occurrence of blur is described in Chapter 3. In addition to the basic algorithm we analyse how the technique of *zero padding* can provide more detailed information, and how we can eliminate the *ringing effect* by masking the original image with a Gaussian window. In Chapter 4 we demonstrate results from both artificially simulated motion and from real images, and we evaluate the robustness of the algorithm. Finally, in Chapter 5, conclusions and suggestions for future developments are presented.

More concisely, the objective of this thesis is to develop a new algorithm that produces the Optical Flow map of the scene using only the information that exist in the motion blur of the image.

Chapter 2

Background: Optical Flow and Motion Blur

When a visual observer moves through an environment, or when objects move in front of a stationary observer, the visual image of the scene changes over time. Analysis of the movement or flow of image structure on the image plane provides a cue to allow the inference of observer or object motion.

This optical flow problem is described below, along with the traditional approaches to solving it. The importance of motion *blur* is also introduced. Although it has been treated as noise by most optical flow algorithms, it in fact carries information that can be exploited.

2.1 Optical flow

When Heraclitus said, 2600 years ago, that: "Everything flows, everything moves, and nothing stays"[1] , he made an observation on the tendency for change in nature. That tendency reflects also in the visual domain – if a visual system observes a scene for a

[1] $\Pi\alpha\nu\tau\alpha\ P\varepsilon\iota$, $\Pi\alpha\nu\tau\alpha\ X\omega\rho\varepsilon\iota$, $\kappa\alpha\iota\ O\upsilon\delta\varepsilon\nu\ M\varepsilon\nu\varepsilon\iota$

long enough period of time, there are going to be notable changes. In most biological visual systems, the analysis of motion is critical; interesting experiments have been made with the visual system of the pigeon, rabbit, frog, fly, and more. Many insights for machine vision have came out of these experiments. The psychophysical aspects of motion information has been demonstrated by Ullman [21] and Marr [17], and the use of this information has been demonstrated in computer vision by Horn and Schunck [13]. A lot of work has been done and different approaches have been taken in order to extract this information.

2.1.1 Definition of the problem

First we are going to set the framework for the study of motion in a visual system. When there exists relative motion between the camera and objects in the scene, there appears corresponding changes in the received image.

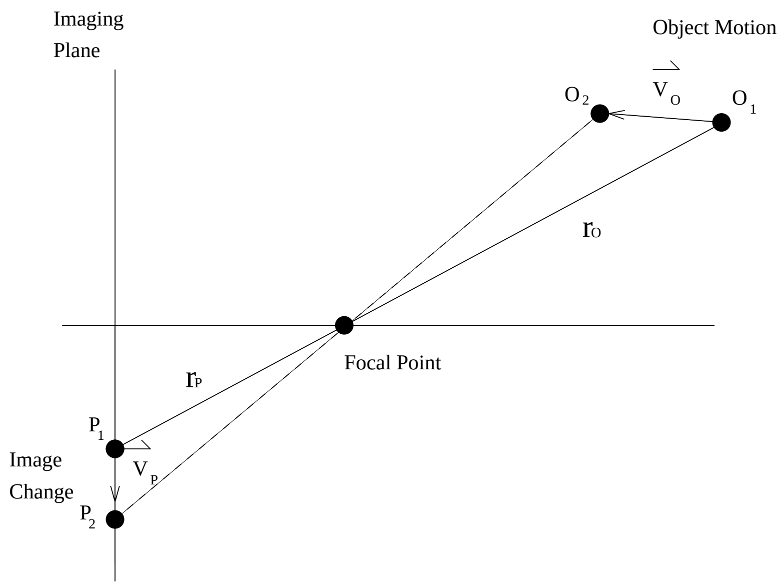


Figure 2.1: Motion correspondence – An object at point O_1 moves with velocity V_O to point O_2. The corresponding image point P_1 moves on the image plane with velocity V_P to point P_2.

The **Motion Field** is defined by assigning a 2D vector to every point in the image, corresponding to the projection of the equivalent velocity vector in the scene [14]. If a point O_1 in the scene moves with a velocity $\vec{V_O}$ (see figure 2.1) in time δt is moving to the position O_2. The equivalent image point P_1 is moving with a proportional velocity $\vec{V_P}$ to the point P_2. That means that we can have an image I_1 taken at time t_0 and a different image I_2 taken at $t_0 + \delta t$, the motion field of the image consists of the velocity vectors $\vec{V_{P_i}}$ that exist for every point P_i of the image. Now the relation between these factors are given in Horn [14] in equation 2.1 where $\vec{V_p} = \frac{dr_p}{dt}$ and $\vec{V_o} = \frac{dr_o}{dt}$ and f' is the distance between the focal point and the image plane.

$$\frac{1}{f'} \times r_p = \frac{1}{r_o \times \mathbf{z}} \times r_o \tag{2.1}$$

During a period of time, the brightness of a specific pixel $P_{i,j}$ could change, the most obvious reason is the relative motion between the camera and the scene; although changes in the shadows and in lighting could also be responsible. As **Optical Flow** we define that variation of the brightness patterns in the image [14]. The problem of estimating the relative motion between the camera and the objects in the scene is generally complex. The first step of reconstructing the 3D velocity vectors is to derive the Motion Field from the Optical Flow. Note that there exist other cases, as mentioned earlier, when the change in the image brightness is not due to the relative motion, but due to other causes such as the situation when a light source is moving changing the shadows in the image and the reflectance from different surfaces. In such a case of course the Optical flow is quite different from the Motion field (see for example figure 2.2).

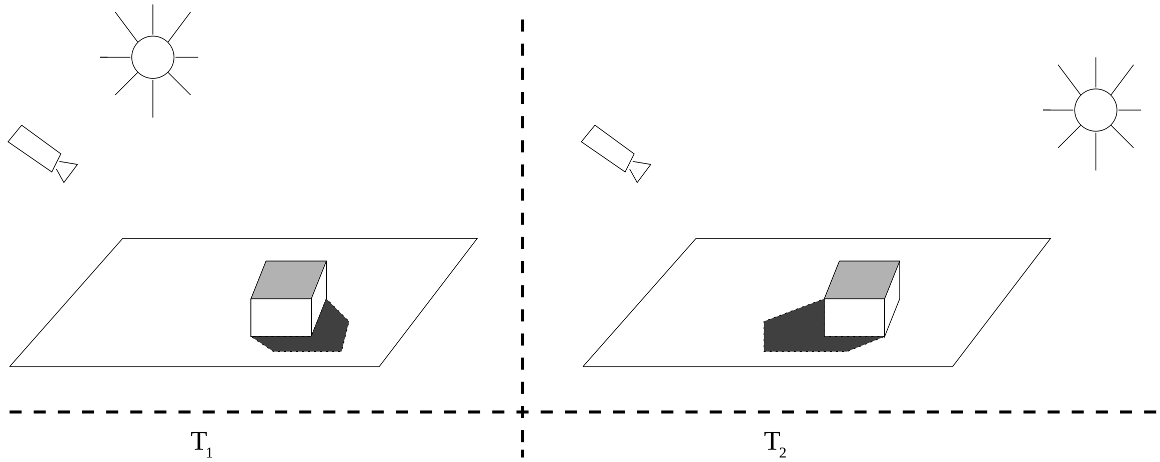


Figure 2.2: Optical Flow due to motion of the light source.

2.1.2 Previous work on traditional algorithms

Many algorithms have been developed since 1980 when Horn and Schunck published their well known paper [13]. The different algorithms can be divided into different groups according the principles of the method they use, the results they seek to get, and the available input data. In this section I am going to give a brief overview for the most commonly used algorithms.

Among the first papers on machine motion estimation is the paper of Horn and Schunck in 1980 [13]. The algorithm in this paper can be defined as a differential method; it assumes that the brightness in any particular point in the scene is constant. That shows in equation 2.2, where **I** is the image intensity.

$$\frac{d\mathbf{I}}{d\mathbf{t}} = 0 \tag{2.2}$$

By taking the first order differentiation of equation 2.2 we have equation 2.3 where $v_x = \frac{dx}{dt}$ and $v_y = \frac{dy}{dt}$

$$\frac{\partial I}{\partial x} v_x + \frac{\partial I}{\partial y} v_y + \frac{\partial I}{\partial t} = 0 \tag{2.3}$$

From this equation is clear that we have two unknowns v_x, v_y and only one constraint. In order to solve the problem, we need one more equation, which we get by making one more assumption - the smoothness constraint - which is actually used in most of the algorithms with some variations. If every pixel in the scene was moving with its own velocity, the problem could prove almost unsolvable, but, as most of the motion happens among rigid objects, there exists a set of pixels (belonging to the same object) that have a smoothly varying velocity. Therefore, one additional constraint can be found by minimising the differences among the velocities in a small patch of the image. In Horn and Schunck's paper this is done by minimising the sum of the squares of the Laplacian of v_x, v_y as given in equation 2.4.

$$\nabla^2 v_x = \frac{\partial^2 v_x}{\partial^2 x} + \frac{\partial^2 v_x}{\partial^2 y} \quad , \quad \nabla^2 v_y = \frac{\partial^2 v_y}{\partial^2 x} + \frac{\partial^2 v_y}{\partial^2 y} \tag{2.4}$$

The quantity that we have to minimise is given in 2.5.

$$Er = \sum(\nabla^2 v_x + \nabla^2 v_y) \tag{2.5}$$

This paper started a whole category of algorithms named differential algorithms, which are based on the concept of taking the derivative (first or second order) of the image intensity, and use one more constraint. Next I am going to present a few survey papers that group together different approaches to the optical flow problem.

In 1988, Aggarwal and Nandhakumar [1] present a review paper on the calculation of the motion. In this paper they divide the methods used to solve the problem, into two different categories: the feature based methods and the optical flow methods.

The feature based methods compute the velocities in the scene only in some areas of the image where features (lines, points, edges) have already been found. Although this kind of method doesn't give a continuous field of the velocities in the scene it is faster and can define the velocity of an object by extrapolating from the velocities at its boundary. In general, this approach assumes that that all the objects in the scene are rigid and their movement consists of a translation and a rotation. The algorithms in this class try to define the 3D motion that exist in the scene based on a set of features, therefore they use a set of lines and/or points that match during the series of images and calculate the 3D velocities. Variations exist considering the number of features, and the number of consecutive images used. Usually, by solving the velocity problem, this approach also computes the 3D structure in the scene. There is also an extension on this a class of algorithms that work with a series of binocular images.

Optical flow methods deal with velocities over the whole image. Many existing methods are differential in nature, based on the work of Horn and Schunck. These approaches usually have one of the following constraints: the smoothness constraint (see earlier), the restricted motion constraint (the change in brightness is a result of a constrained motion), or a homogeneity constraint (all the pixels in a specific region, belonging in the same object, move with the same velocity). Some algorithms use the second order derivatives of the image, and others use iterative methods that moves from a coarser estimation of the optical flow to a finer one. There also exist algorithms that use binocular image series in order to extract the 3D structure and the 3D velocity field of the scene, but they assume that the correspondence problem among every stereo pair in the sequence is solved. In the absence of binocular correspondence, other constraints can be used in order to compute the 3D structure and velocity field of a scene from the optical flow field.

The main differences between these two types of approaches is that, the feature based methods require the existence of a match of features among consecutive images before the algorithm is applied – up to now, most of the algorithms have only give partial solutions to this problem – while, the optical flow methods don't need any feature correspondence to be established. Another difference is that optical flow techniques are very sensitive to noise and this make their application to the real world situation difficult.

The same division can be made in both biological and computer visual systems;

experiments on biological systems have been shown by Ullman [22]. Again the methods are divided into intensity based, or otherwise called Optical Flow methods, and feature based, or otherwise called Token Matching methods. For the optical flow methods two different kinds of approaches have been proposed: Correlation schemes and Gradient schemes. In the first case, the input of the two consecutive images is compared after the first image have been translated by $d = vdt$; different variations of this method have been proposed. The second case, called Gradient scheme, has been found implemented in the retina. In that case, research from Hartline, Barlow and Kuffler have shown that the retina cells behave like the difference of two Gaussians. In other words the input image is convolved by the Laplacian of a Gaussian and the response to that operation point out zero-crossings that corresponds to sharp changes. This Gaussian behaves like a smoothing filter, and controls the size of the operation. Consequently, at the position where an edge exists, the values of the convolution increase according to the direction of the movement of the edge. The output of a second biological filter that provides the time derivative on the results of that convolution is going to provide also the direction and magnitude of the motion [22]. Feature based approaches have been also proposed. From the experiments up to now it seems that these approaches are valid and most probably coexist in biological visual systems.

Vega-Riveros and Jabbour in 1989 [23] take a similar approach into dividing the algorithms. They consider two general categories, the first one that calculates the optical flow based on a differential kind of algorithm, and the second which is more general than the feature based approach and is based on pattern matching.

The differential methods are essentially based on the same idea as Horn and Schunck's paper [13]; the optical field is considered smooth and the same is assumed for the motion, and the calculations are done by differentiating the images spatially or according to time. There exists variations mainly in the constraints that are used in order to solve the problem. In this category there exist also algorithms that are using the second derivative; this is done by extending the basic equation 2.6 into a Taylor series up to the second order terms. This equation assures that a pixel $I_{x,y,t}$ at time t and position x, y after time dt in its new position have the same intensity.

$$I(x, y, t) = I(x + v_x dt, y + v_y dt, t + dt) \tag{2.6}$$

The feature identification methods can be subdivided into more categories, mainly according to what kind of features we are using and how we match them from frame to frame. Methods that are using cross correlation are quite common in the literature - this approach is using cross-correlation between two consecutive images in order to find the best match that gives the movement of a certain patch of the image. Other algorithms detect the match that occurs for the biggest moving object ignoring the motion of smaller objects. Another interesting but specialised kind of method is when there exist a mathematical function that describes some aspects of the object and then the algorithm tries to match that function into different images and calculate the displacement. In the category of feature identification methods exists of course the feature correspondence methods that appear in the Aggarwal and Nandhakumar [1] paper.

In 1992, Barron, Fleet and Beauchemin [4] made a quantitative analysis of the different common algorithms that exist for solving the optical flow problem. There are four different categories according to this analysis: one is the differential methods, which starts with the Horn and Schunck algorithm, and continues with the Lucas and Kanade algorithm and then the Uras, Girosi, Verri and Torre (which is a second order derivative method). The other category is the region based method where a correlation type algorithm of Anandan is used which is iterative and calculates the optical flow from a coarser to a finer result. The third category is the energy-based approach, which is using the output of special velocity tuned filters, usually the calculations being transformed in the frequency space by the Fourier transform;

these methods are also called frequency based – for example the algorithm of Heeger. The last type of algorithm is based on the phase-based method, which calculates the velocity by the behaviour of the phase of band-pass filter outputs, like the algorithm presented by Fleet and Jepson.

In the next part I am going to present different papers that explain more a certain approach, or give better results than a previous algorithm due to a new idea.

In the differential framework, there is an approach that follows the same method for solving a series of problems in vision [24].This approach has been already used in stereopsis and texture and is now applied into the optical flow problem. The series of images is convolved with a set of linear, separable, spatiotemporal filters similar to those used in the previous vision problems, then the usual brightness constancy constraint is applied and we get an over-determined system of equations from where we can estimate the optical flow using a robust total least square method. The advantages of this approach are: firstly the ability to use the same set of filters (applied only once) and solve a series of problems – approach that seems compatible with what happening in the biological visual systems; secondly the fact that the application of the filters can be done in parallel and, therefore, have a fast solution.

In the differential approach the smoothness constraint presents a problem at the boundary areas. This is due to the assumption that every pixel has a velocity similar to its neighbours, assumption that holds only if all the pixels belong to the same object. Also, the assumption that every point maintains the same brightness can generate problems. Therefore, a study has been done by Black and Anandan [5] especially in order to deal with this outliers. In pursuing this goal they use a robust statistical method, where different kind of estimators are used in order to minimise the error that outliers introduce.

Sometimes we need to calculate the optical flow in a specific direction; in this case a faster approach can be taken in order to calculate 1D optical flow. Although the results are qualitative, this can be enough for primitive tasks as a time to crash detector [2]. One method is to use a correlation scheme in one direction only. Consequently, if the optical flow is estimated along the horizontal and vertical axis, then an approximation of the real motion can be extracted.

2.2 Motion Blur

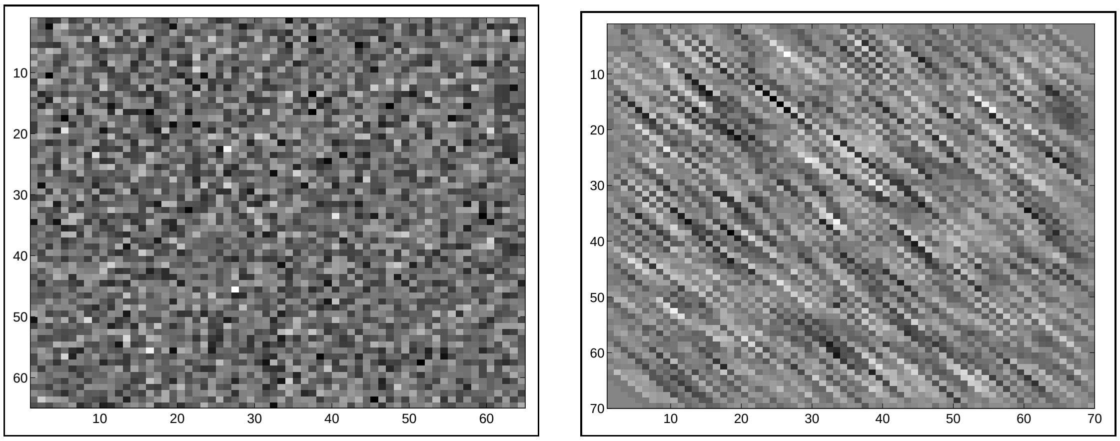
In all the previously described approaches, a set of conditions have been assumed to be true. Although statistical methods have been used in order to minimise the error that is caused when these conditions fail, the algorithms are based on the assumption that in general these conditions hold. Among these conditions are the assumption that the pixels keep their brightness from one frame to the other having changed their position only, also they consider every pixel to refer to a unique point in the scene. In addition, the previously mentioned methods work on a series of consecutive images (at least two) in order to calculate the optical flow. In the next section I am going to analyse what happens when the motion is faster than a pixel per frame, and what has already been done in using the motion blur.

2.2.1 Motion blur definition

When a changing scene is observed by a camera, all the classical algorithms assume that it is possible to take pictures every δt instantly, that means that every picture is taken with a $dt \approx 0$ exposure time. If that is not the case, then the exposure time ($dt = T$) is large enough that different points in the scene are moving far enough and consequently their corresponding projections on the image plane travel several pixels. Therefore, during the capture of an image, at any single image point, a certain number of scene points is projected during the exposure time, each one contributing to the final brightness of the image point; this effect is clearly demonstrated in figure 2.3. More formally, during the exposure time T in front of the pixel $P_{i,j}$ we could assume that they pass k scene points with brightness $(C_1 \ldots C_k)$ respectively, then the resulting brightness value for pixel $P_{i,j}$ is given in equation 2.7, in the case of continues movement the summation is replaced by integration. This holds in general for every pixel that can *see* moving points in the scene. It is clear that the blurring of the image exists only across the direction of the motion, this one dimensional blur is called *Motion Blur*.

$$P_{i,j} = \frac{1}{k} \sum_{l=1}^{k} C_l \tag{2.7}$$

The result of motion blur is more obvious in the figure 2.3 where an image consistent of random value pixels is shown in figure 2.3a and then the blurred image is shown in figure 2.3b.



(a) (b)

Figure 2.3: Random noise image, and the same image blurred due to motion.

The motion blur can be described mathematically as the result of a linear filter $\mathbf{b}(x, y) = \mathbf{i}(x, y) * \mathbf{h}(x, y)$ where $\mathbf{i}$ is the theoretical image taken with an exposure time $T_e = 0$, $\mathbf{b}$ the real blurred image and $\mathbf{h}$ the point spread function (PSF). Given an angle= α and the length $d = V_o \times T_e$, which is the number of scene points that affect a specific pixel, the point spread function of motion blur is given in equation 2.8.

$$h(x, y) = \begin{cases} \frac{1}{d}, & 0 \leq |x| \leq d \star cos(\alpha) \quad y = sin(\alpha) \star d \\ 0, & otherwise \end{cases} \tag{2.8}$$

The focus of this thesis research is to formulate and evaluate methods for recovering and interpreting motion blur. In practical terms, this mean computing accurate estimates for the two parameters of the motion blur PSF, namely the length, d, and the angle, $\hat{\alpha}$. From these quantities, the relative velocity at this point can be easily recovered knowing the exposure time. Moreover, in a lot of applications we simply need this qualitative measure and not its exact value when, for example, we want to deblur the image, or infer the egomotion.

2.2.2 Interpretation of the motion blur and previous work

Up to now, blurring due to motion was considered an additional source of noise. Usually the traditional algorithms for motion estimation tried to ignore it, or recover from it. Also, in many applications the blur in an image is a source of noise, and techniques have been developed in order to remove it [11], [16]. But, in the other side, the motion blur is a structured noise and contains information that can be used. Psychophysical experiments have been done in order to analyse the use of motion blur by the human visual system and some approaches have been taken in order to use it in machine vision systems. Moreover, by using the motion blur we can estimate the optical flow using only one image. In general, the use of motion blur belongs to a group of methods that try to extract information from blurred images in order to estimate the 3D structure of the scene from out of focus blur or the optical flow from motion blur.

The experiments that have been done for the human visual system in order to determine the influence of motion blur in human perception conclude that a deblurring mechanism must exist in order to distinguish features in a specific image [3]. The human visual system can identify motions that vary from less than one to more than 100,000 minutes of arc per second; apparently, it has been demonstrated that not the same mechanism is used for all this broad spectrum of motions. For slower motions, where the shapes barely move from cone to cone the model of Bonnet [6], "Displacement Analysing System" is used; as the motion become faster Bonnet's "Movingness Analyzing System" is stimulated. Finally at high speed motions where the motion blur is more obvious, a third mechanism is used in parallel with the other two. In high speed flights for example, where jet pilots flew just above the ground, the motion blur is forming patterns that could be analysed in order to produce useful information; in such cases a pattern recognition mechanism is activated. Experiments have been done [12] that estimate the importance of different parameters of motion blur patterns in identifying the motion and aspects of the 3D structure of the viewing surface. The parameters that were used in the experiments were: blur pattern divergence, where the observers have to use the divergence in the blur lines in order to extract the tilt of the viewed surface, and blur pattern curvature which appears when there is a change in direction of move. The other parameter that is important in the motion blur patterns is the "blur pattern divergence change" which appears when the observer change his velocity of climb or descent. The last parameter that have been studied was "blur pattern curvature change".

The issue of estimating the blur parameters has also been studied by the machine vision community; usually, there exist two kinds of blur, the out of focus blur and the motion blur. More specifically, the motion blur identification, and consequently the extraction of the motion blur parameters, has been studied mainly in order to deblur the images for a series of applications. Also, most of the image restoration algorithms of motion blurred images assume that the parameters of the PSF are already known, and therefore there is no need for estimating them. Usually, in addition to the motion blur, there are also other kinds of noise present in the image, so a more robust estimation of the motion blur parameters is needed. One approach, [8] which is working for both motion and out-of-focus blur, is to proceed in two stages. First, the degraded image is processed in order to improve the SNR and then the algorithm that extracts the blur parameters is applied. This approach assumes a model for the degraded image as given in equation 2.9 where $g(i,j)$ is the degraded image, $f(i,j)$ is the *ideal* image, $h(i,j)$ the PSF, and $n(i,j)$ additional noise.

$$g(i,j) = f(i,j) * h(i,j) + n(i,j) \tag{2.9}$$

The $h(i,j)$ for the case of motion blur created by uniform motion across the X axis is given in equation 2.10. The algorithm in [8] is developed only for this specific case and therefore it is clear that equation 2.10 is just a sub-case of equation 2.8 that we analyse at the definition of motion blur.

$$h(i,j) = \left\{ \begin{array}{ll} \frac{1}{d}, & -d/2 \leq i \leq d/2; \quad j = 0 \\ 0, & otherwise \end{array} \right| \tag{2.10}$$

At this point we have to define two tools that are essential for the further analysis of the algorithms. The Fourier Transform [2] (FT) $F(u,v) = \mathcal{F}\{f(x,y)\}$ of a function $f(x,y)$ is defined in equation 2.11 together with the Inverse Fourier Transform (IFT) $\mathcal{F}^{-1}\{F(u,v)\} = f(x,y)$ (see [11], [16]). The Fourier transform of $h(i,j)$ from equation 2.10 is shown in equation 2.12.

$$\begin{aligned} F(u,v) = \mathcal{F}\{f(x,y)\} &= \int_{\infty}^{\infty} \int_{\infty}^{\infty} f(x,y) e^{-\imath(ux+vy)} dx dy \\ f(x,y) = \mathcal{F}^{-1}\{F(u,v)\} &= \frac{1}{4\pi^2} \int_{\infty}^{\infty} \int_{\infty}^{\infty} F(u,v) e^{\imath(ux+vy)} du dv \end{aligned} \tag{2.11}$$

$$H(u,v) = \frac{\sin(\pi du)}{\pi du} = sinc(\pi du) \tag{2.12}$$

[2] For a more detailed analysis see section 3.2

Another transform that can be used in analysing one image is the *Cepstrum*[3]. The definition is given by equation 2.13, where $\mathcal{F}^{-1}$ is the Inverse Fourier Transform (usually using the fast version of $IFFT$), and $F(u,v) = \mathcal{F}\{f(x,y)\}$ is the Fourier Transform of $f(x,y)$ (as in equation 2.11). The Cepstrum is the Fourier transformation of the *log* spectrum of an image; it is therefore a tool for analysing the frequency domain of an image.

$$C_f(p.q) = \mathcal{F}^{-1}\{\log|F(u,v)|\} \tag{2.13}$$

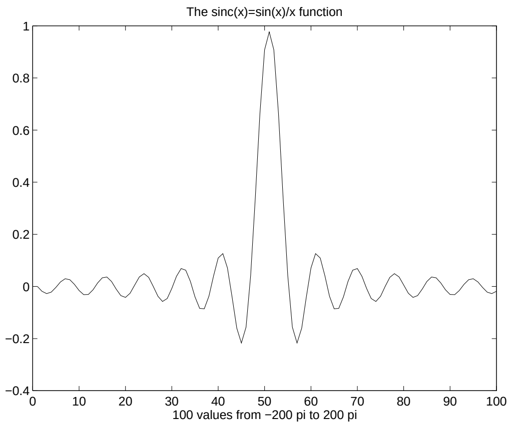


Figure 2.4: The Graphical representation of the sinc function

As from the Fourier Transform of the blur PSF $h(u,v)$ in equation 2.12 and its graphical representation in figure 2.4, it is clear that $H(u,v) = sinc(\pi du)$ is a periodic function with period $T = \frac{1}{d}$, therefore every $\frac{1}{d}$ there exist a zero crossing. The convolution operation in the frequency domain is transformed into the multiplication of the two matrices, as a result the periodic function that is the Power Spectrum of the blur PSF appears as a ripple in the Power Spectrum of the blurred image, this ripple can be identified by a negative peak in the Cepstrum domain. For a more in-depth explanation refer to Chapter 3.

[3] For a more detailed analysis see section 3.5

Most recent work dealing with blur in images focuses on the problem of extracting the blur parameters from a noisy image [8]. The stage of noise reduction is accomplished with a technique called Spectral Subtraction, which can be used complementary to the divide and averaging technique. The main technique is to take an estimation of the Fourier Transform of the noise and subtract it from the Fourier Transform of the blurred image; sometimes different estimations can exist for different parts of the image. As these algorithms deal only with uniform motion across the X-axis only the line $C_b(p, 0)$ is used, where C_b is the Cepstrum of the enhanced image. In order to improve the robustness of the algorithm one more stage of filtering is used to the 1D signal $C_b(p, 0)$. As only the negative candidates count, and they are repeated periodically, a comb like filter is employed. This approach divides every negative pulse with the root mean square (*RMS*) of all the negative terms except the ones that are in multiples of the index of this pulse.

The frequency domain also is used in another method [7]. In that case the bispectrum is used in order to find the parameters of the blur *PSF*. Like in the previous case, uniform motion across the X-axis is assumed and thus the problem is restrained in the one dimension. In an other approach the Discrete Cosine Transform DCT is used [25]. In this case the same kind of movement is assumed and the use of DCT instead of FT is preferred because of the assumption the DCT makes that the signal is assumed to be at the boundaries even symmetric, instead of periodic as in Fourier transform.

Chapter 3

Algorithm for Analysis of Motion Blur

In this chapter a new algorithm for extracting the parameters of motion blur in an image is presented and analysed. The method that is developed here calculates the optical flow from independent relative motion between the camera and different objects at the scene. For example, a situation as in figure 3.1, where three objects A, B, D move with different velocities $\vec{V}_A, \vec{V}_B, \vec{V}_D$ and a camera C moves with a velocity $\vec{V}_C$, is handled by assigning different velocities in different parts of the image.

In section 3.1 a brief outline of the algorithm is given. In the next section (3.2) the application of the Fast Fourier Transform and different techniques to improve the results is going to be analysed. Consequently the role of the Steerable Filters in feature extraction from the spatial frequency domain is discussed in section 3.3. The next section 3.4 deals with the transform of the 2D signal to 1D with the proper normalisation. In section 3.5 the use of Cepstrum and the extraction of the length of the velocity vector are demonstrated. Finally in section 3.6 a complexity analysis of the algorithm is done.

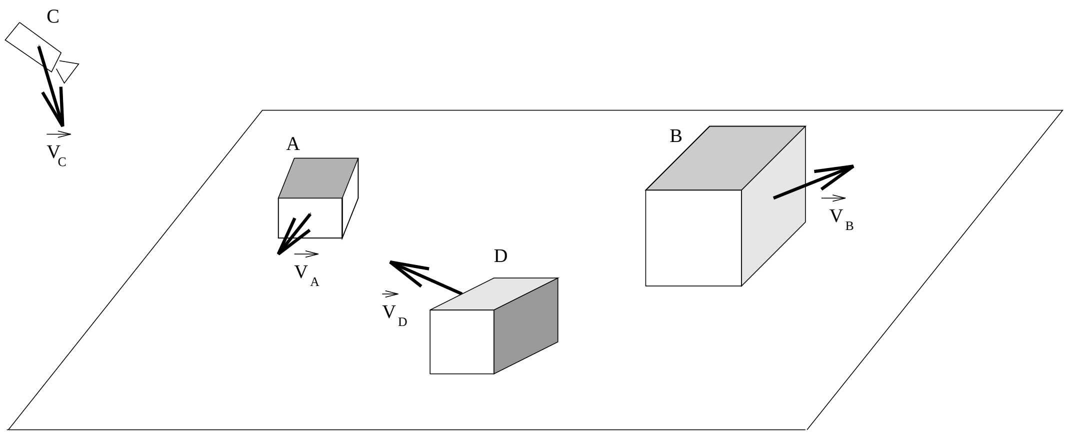


Figure 3.1: Independent motion between the camera and the objects in the scene

3.1 Outline of the algorithm

In order to calculate the optical flow for a certain point we make use of an area around it – this method needs only one frame taken with an exposure time δt where the motion blur spans for more than a couple of pixels, as is the situation in a series of applications. Therefore, in order to calculate the optical flow of the whole image we run the following described algorithm for a series of overlapping image segments. The algorithm can be divided in two stages: first there is the extraction of the orientation of the velocity vector, and second the calculation of the magnitude of it.

In the first stage there exists an optional step of preprocessing in order to have better results with the initial Fourier Transform. Two methods can be used in this step either separately or at the same time - zero padding, and masking with a Gaussian window. The second step is the extraction of the orientation of the velocity vector; this is done from the power spectrum of the image (taken by the Fourier Transform) by finding the maximum response in a set of Steerable Filters.

The second stage has also two steps: a preprocessing step where the 2D Power Spectrum of the image is *collapsed* in 1D (at that point also a normalisation is performed in order for the collapsed 1D signal to have the format of a Power Spectrum),

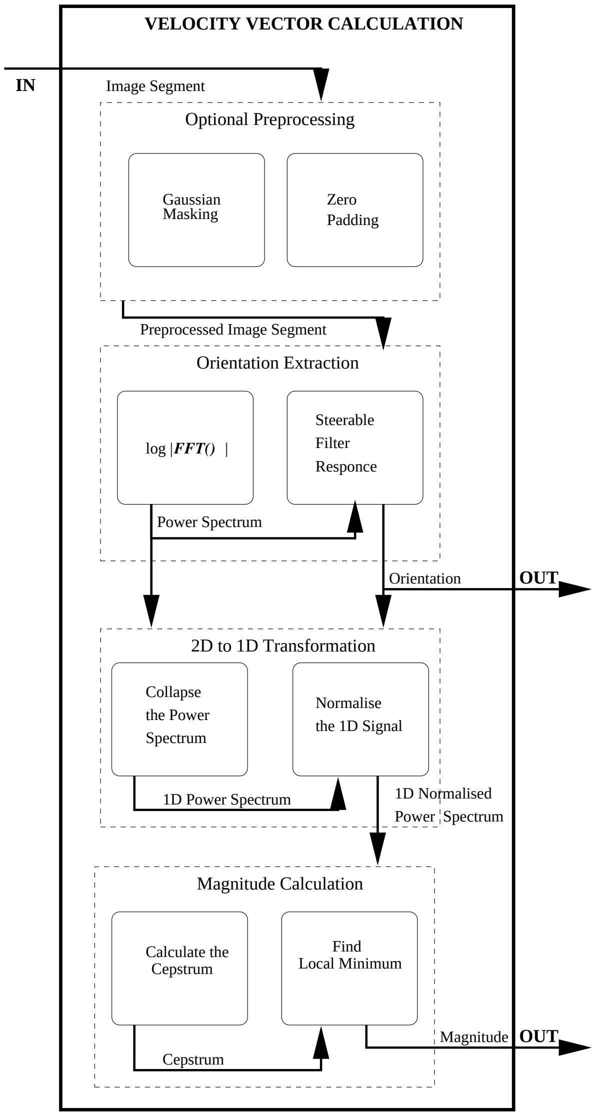


Figure 3.2: The outline of the algorithm for calculating the Velocity Vector of a image segment

and the second step where the application of the Cepstrum provides us with the magnitude of the velocity vector.

The algorithm in figure 3.2 calculates the velocity vector for the pixel that is at the middle of *Image Segment*. It could be run in parallel in order to calculate the optical flow in the whole image.

3.2 Fourier Transform

The identification of the direction of the motion blur is calculated in the frequency domain. The first step is the transformation of the image from the spatial to the frequency domain through the Fourier Transform. In order to have lower computation time the Fast Fourier Transform algorithm is applied, and for enhancing the features the logarithm of the *Power Spectrum* is used.

3.2.1 *FT* definition and properties

One of the most used common transforms in Computer Vision and Image Processing is the Fourier Transform (*FT*). It is a well defined and a popular tool, as it has a lot of useful properties, and is relatively quick to compute (see [11], [16], [18]). In equations 3.1 to 3.4 we have the continuous 2D *FT*, the Discrete 2D *DFT* and their Inverses *IFT*, *IDFT*. The $f(x, y)$ represents a function in the Spatial domain (an image), and the Fourier Transform ($\mathcal{F}$) transfer it to the Spatial Frequency domain.

$$F(u,v) = \mathcal{F}\{f(x,y)\} = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(x,y)e^{-\imath(ux+vy)}dxdy \tag{3.1}$$

$$F(h,j) = \mathcal{F}\{f(k,l)\} = \frac{1}{n}\sum_{k=0}^{n-1}\sum_{l=0}^{n-1} f(k,l)e^{-\imath 2\pi(kh+lj)/n} \quad 0 \le h,j \le n-1 \tag{3.2}$$

$$f(x,y) = \mathcal{F}^{-1}\{F(u,v)\} = \frac{1}{4\pi^2}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} F(u,v)e^{\imath(ux+vy)}dudv \tag{3.3}$$

$$f(k,l) = \mathcal{F}^{-1}\{F(h,j)\} = \frac{1}{n}\sum_{h=0}^{n-1}\sum_{j=0}^{n-1} F(h,j)e^{\imath 2\pi(kh+lj)/n} \quad 0 \le k,l \le n-1 \tag{3.4}$$

As it is obvious from the equations, the *FT* is almost symmetrical with its inverse *IFT*. In order for the transformations to be possible a few conditions must apply: for the continues case, $f(x, y)$ must be a piecewise continue function of real variables

x, y, and having the left and right hand derivatives; in the discrete case, the main assumption is that $f(k, l)$ is periodic.[1] Although an image $f(x, y)$ is a real function, its transformation $F(u, v)$ is, in general, a complex one; consequently, we can define the Real and the Imaginary part of the FT as in equation 3.5. In a lot of situations it is useful to have the FT expressed in terms of an exponential as in equation 3.6 with the magnitude $|F|$ and the phase ϕ defined in equations 3.7, 3.8 with the help of the Real and Imaginary part. The magnitude $|F(\omega, v)|$ is commonly called *Fourier spectrum* and its square $P(\omega, v) = |F(\omega, v)|^2$ is called *Power spectrum* or *Spectral Density*. The $\phi(\omega, v)$ is called the *phase* function.

$$\mathcal{F}\{f(x, y)\} = F(\omega, v) = R(\omega, v) + \imath I(\omega, v) \tag{3.5}$$

$$F(\omega, v) = |F(\omega, v)| e^{\imath\phi(\omega, v)} \tag{3.6}$$

$$|F(\omega, v)| = \sqrt{R^2(\omega, v) + I^2(\omega, v)} \tag{3.7}$$

$$\phi(\omega, v) = \tan^{-1}\{\frac{I(\omega, v)}{R(\omega, v)}\} \tag{3.8}$$

In a number of applications the *Power Spectrum* is used in order to identify different features. In a lot of images though, the *Fourier Spectra* decreases rapidly and the features are not recognisable; therefore, another function is used in order to *amplify* the signal – the logarithm of the *Fourier Spectrum* plus one (see equation 3.9). This function have the property of keeping the zero values of the *Fourier Spectra* zero, and at the same time magnify small differences.

$$L(\omega, v) = \log(1 + |F(\omega, v)|) \tag{3.9}$$

One of the most common properties of FT is known as the convolution theorem (see equation 3.10). This theorem shows that the FT of a convolution of two functions

[1] This condition is responsible for the effects discussed in 3.2.3

is equal to the product of the FT of the two functions; therefore a lot of functions that are represented by a convolution in the spatial domain can be transformed to a simple product of their FT in the frequency domain. Also, another of the basic properties of FT that comes directly from its definition is linearity (see equation 3.11).[2] Linearity enables us to break down a complicated function into simple ones, with a well known FT.

$$\mathcal{F}\{f(x,y) * h(x,y)\} = \mathcal{F}(f(x,y))\mathcal{F}(h(x,y)) = F(\omega,v)H(\omega,v) \tag{3.10}$$

$$\mathcal{F}\{\alpha f(x,y) + \beta h(x,y)\} = \alpha\mathcal{F}(f(x,y)) + \beta\mathcal{F}(h(x,y)) = \alpha F(\omega,v) + \beta H(\omega,v) \tag{3.11}$$

The computational cost of the Fourier Transform or its inverse in the discrete case is $O(n^2)$. Taking advantage of the separability property – which states that in the 2D FT we can perform first the summation (or integration in the continues case) over the first variable and then over the other independently – an algorithm has been developed called Fast Fourier Transform FFT which calculates the FT (or it's inverse) in time $O(n\log_2 n)$ [11]. For the rest of thesis the FFT and its inverse $IFFT$ are used.[3]

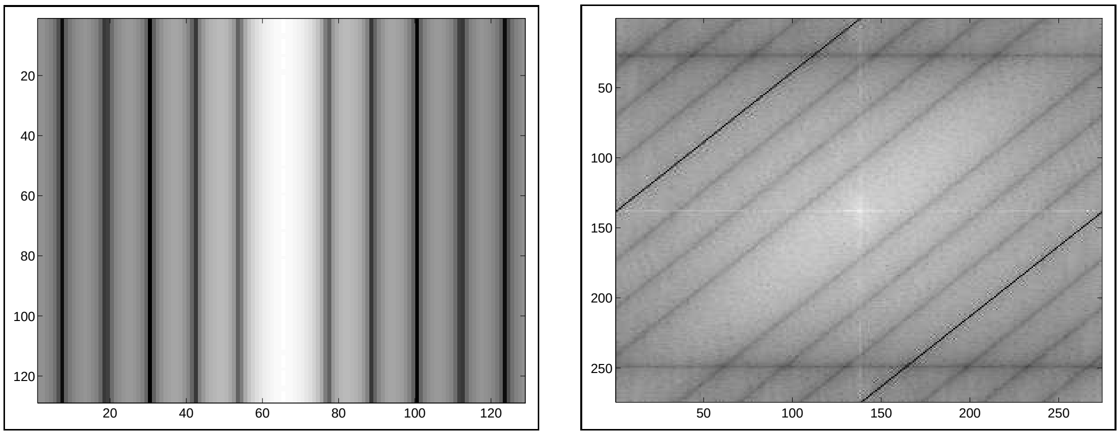
3.2.2 *FFT* of a Blurred Image

An image blurred due to motion is usually represented by a linear system of a convolution: $g(x,y) = f(x,y) * h(x,y)$ with $h(x,y)$ the convolution kernel that cause the blur. Already at [8] the FFT of the blur PSF is defined for uniformal motion across the $x - axis$ (see equation 2.12). In general, for an arbitrary direction of the motion

[2] For a proof of these two properties see [16].

[3] The presentation of the *Fourier Transform* and its properties here was rather simple, and mainly focused on the aspects that were used in this thesis. There exist a lot more properties of the *Fourier Transform*, for a more extended analysis see the references.

the *FFT* of the *PSF* is a ripple as shown in figure 3.3, clear in the case of horizontal or vertical motion (see figure 3.3a) or distorted slightly[4] – as is the case for a blur at the 45° angle (see figure 3.3b) where it is more the shape of an ellipse with the long axis perpendicular to the direction of motion. In any case the *Power Spectra* of the *PSF* of the motion blur is a ripple along the direction of the motion.



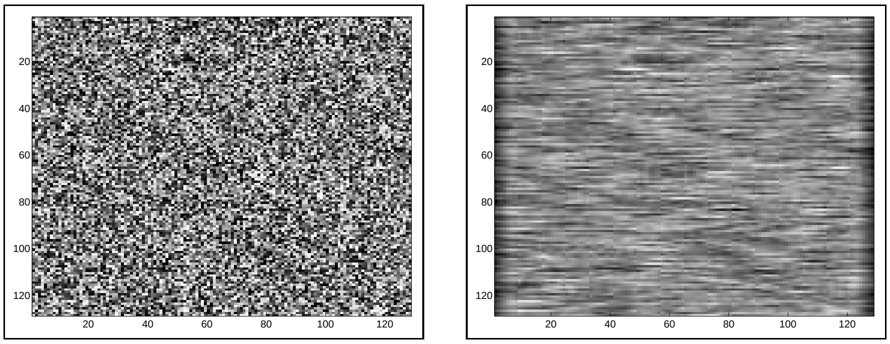
(a) (b)

Figure 3.3: The Power Spectrum of the *PSF* of horizontal (a) and at 45° angle (b) motion blur

In a motion blurred image, the *FFT* highlights some features in a way that makes the extraction of the direction of the motion easier. This is mainly accomplished because of the convolution theorem (see equation 3.10) which transform the convolution of the image with the *PSF* of the motion blur into a simple product of their *FT*. For example, if we have a random dot picture (see figure 3.4a) blurred by a horizontal motion (see figure 3.4b), then the transformation into the frequency domain is going

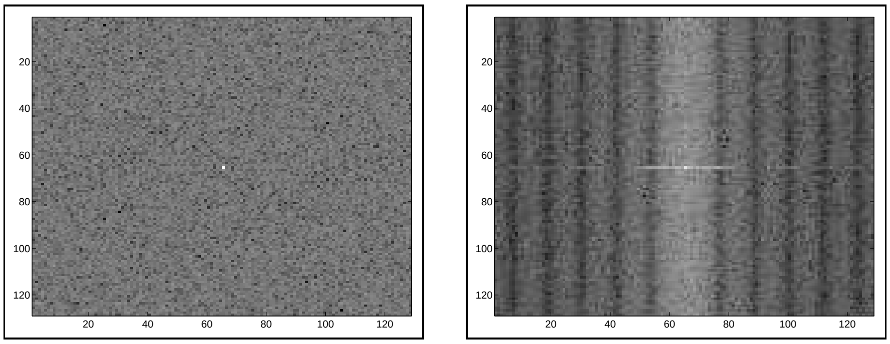
[4]Mainly because of numerical errors and the windowing effect. We have to take into account also the fact that *FT* is a complex transformation and therefore it exist an imaginary part that is not displayed here.

to enhance some features, shaping the result mainly according to the *FFT* of the motion blur *PSF*.



(a) (b)

Figure 3.4: Random Dot Image, and the same image blurred due to horizontal motion.



(a) (b)

Figure 3.5: Power Spectrum of the random image, and the *PS* of the blurred one.

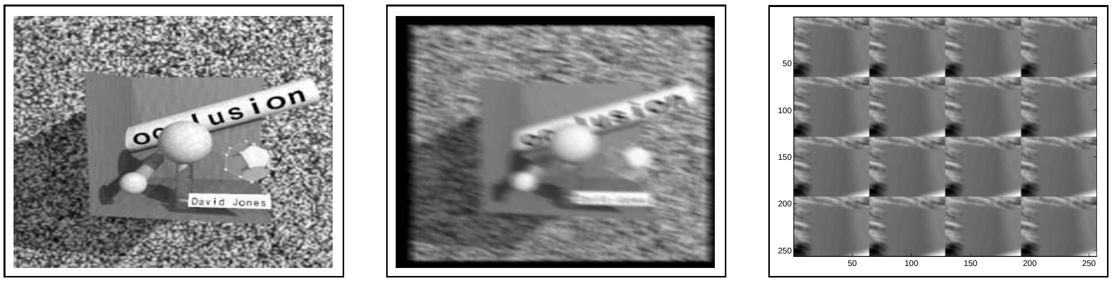
It is obvious from figure 3.5, that the logarithm of the Power Spectrum of the blurred image (see figure 3.5b) has an easily recognisable shape of a ripple located at the centre with its axis perpendicular to the direction of the blur. In the other side the *Power Spectra* of the random dot image is completely unstructured, as can be seen in figure 3.5a. One more source of information that exists in the ripple is the width of it. From the equation 2.12 the period of the ripple is equivalent to the length of the motion blur *PSF* which is equivalent to the velocity of the motion for a given exposure time.

3.2.3 Windowing effect

One of the properties of discrete Fourier Transform is that the signal is considered to extend periodically to infinity in each side. In the case where we have a finite signal the FT assumes an infinite periodic signal which consists of copies of the original signal shifted. Another property is that any sudden change in the spatial domain creates a response in the frequency domain. For example, assume that we have an image (figure 3.6a) which is artificially blurred due to motion at 70° angle with the y-axis (figure 3.6b), and we want to calculate the velocity at the point $P(32, 32)$ using a 64×64 window. In that case, we take the image patch around $P(32, 32)$ (figure 3.8b), and apply the FFT, which as is considering the signal to be periodic and infinite, and it is going to repeat the 64×64 patch one next to the other (in figure 3.6c we could see the repetition). But, by doing this it is going to cause a sudden change at the boundaries, which is going to appear in the FT later.

Another way to approach the problem is to consider what does it mean to take only a patch of the image; this is equivalent to taking the whole image and masking it with a window that has the value one at the 64×64 area of interest and zero everywhere else (see equation 3.12). As it is clear from the convolution theorem, such an operation is going to transform into the frequency domain and the result is going to be affected by the FT of the masking function.

$$m(x, y) = \begin{cases} 1, & 1 \leq x, y \leq 64 \\ 0, & otherwise \end{cases} \qquad (3.12)$$



(a) (b) (c)

Figure 3.6: An image, the result of the motion blur, and a 64×64 patch repeated periodically.

$$m(x,y) = \begin{cases} e^{(-x^2-y^2)}, & -32 \leq x, y \leq 32 \\ 0, & otherwise \end{cases} \tag{3.13}$$

In order to extract only one part of the image different masking functions can be used [20]. The more abrupt the change into the zero level, the more severe the artifacts that are going to appear in the frequency domain. Also by masking the original signal we want to keep it as much as possible unchanged. There exist a lot of research in signal processing for the best masking function. Among the most commonly used are the functions showing in table 3.14, the function $f(n)$ has the illustrated type in the space $[0, M-1]$, and 0 everywhere else (the functions are one dimensional but they are easily transferred into two dimensions, as have been done for the Gaussian in equation 3.13). The graphical representation of the masking functions and their Fourier Transform are shown in the graphs 3.7: the dense dots represents the Rectangular windowing function, the continues line the Gaussian function, the dot and dash is the graph of the Blackman function, the sparse dots display the Hamming one, and the dashed line the Hanning function.

$$f(n) = \left\{ \begin{array}{ll} 1 & Rectangular \\ e^{-(\frac{n-\frac{M-1}{2}}{(M-1)/4})^2}, & Gaussian \\ 0.42 - 0.5\cos(\frac{2\pi n}{M-1}) + 0.08\cos(\frac{4\pi n}{M-1}), & Blackman \\ 0.54 - 0.46\cos(\frac{2\pi n}{M-1}), & Hamming \\ \frac{1}{2}\left(1 - \cos(\frac{2\pi n}{M-1})\right), & Hanning \\ \quad Where \quad 0 \leq n \leq M-1 & \end{array} \right\} \tag{3.14}$$

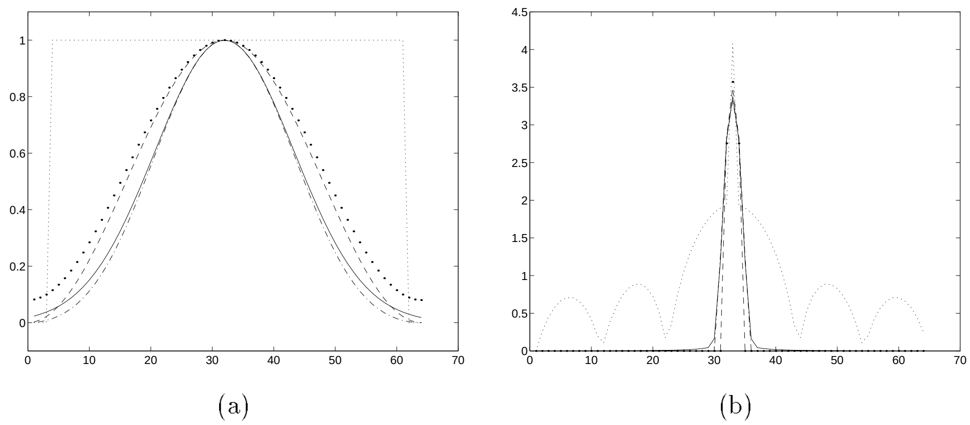
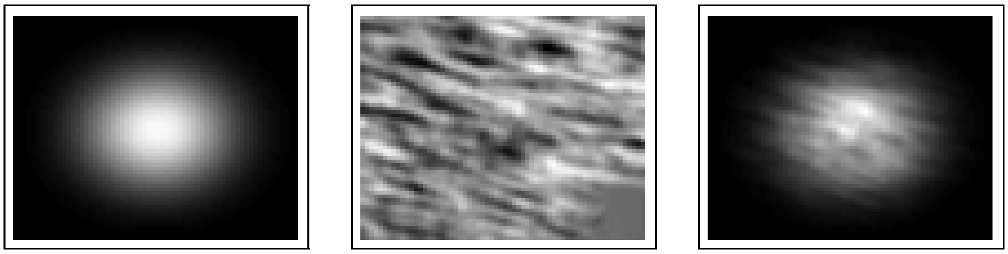


Figure 3.7: Different masking functions (a), and their Fourier Transform (b)

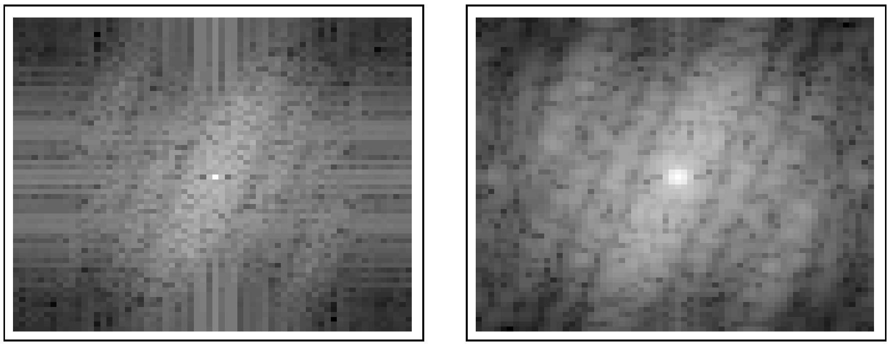
As is clear from figure 3.7b, the Rectangular function has a very strong artifact which, because of its shape is called *ringing effect*, where all the remaining windowing functions have approximately minimum ringing. In the algorithm that we use for calculating the FFT, the 2D Gaussian function is applied as can be seen in figure 3.8a. If we mask a 64×64 patch of the image (figure 3.8b) with the Gaussian window the result is shown in figure 3.8c. Using this Gaussian masked window we have the *Power Spectrum* as it appears in figure 3.9b. It is clear that when the *Power*

Spectrum of the image is taken by using the Gaussian instead of the Rectangular window masking (as in figure 3.9a) most of the artifacts disappear.



(a) (b) (c)

Figure 3.8: A Gaussian Window (a), a 64×64 patch of the blurred image (b), the same patch masked with the Gaussian function (c).



(a) (b)

Figure 3.9: The *Fourier Transform* of the image patch with the Square windowing function (a), and with the Gaussian one.

3.2.4 Zero padding

Another technique used to improve the efficiency of the *FFT* is called *Zero Padding* [20]. In the continues case, (see equation 3.1) the *FT* covers continuously all the

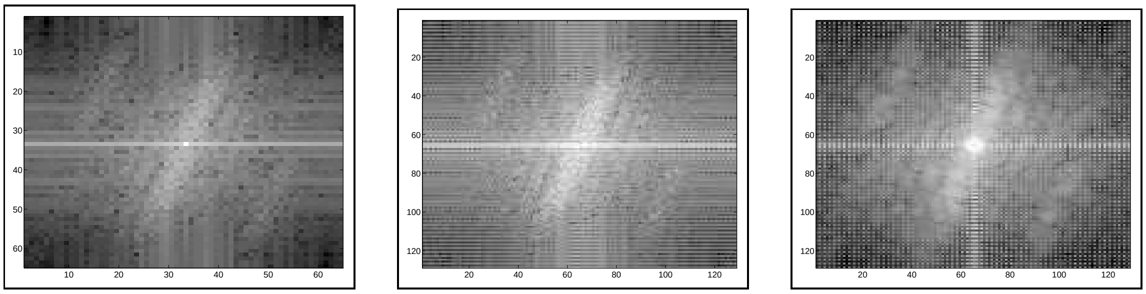
2D space; in the discrete case although, (see equation 3.2) the FT of a function is having as many samples as the function that it transforms. For example if we have an 128×128 image and we take its FT, then the result is going to be also 128×128. In order to get a more (optically) detailed frequency image, we could add zeros at the end of the signal, in both dimensions, (see equation 3.15), and take the Fourier Transform after. This increases the sampling rate of the FT; at the same time as the size of the signal increases the computation time is also increases, therefore although we could add as many zeros as we want at the end of the signal we always have to take into account the time constraint.

$$PaddedImg(x,y) = \begin{cases} Img(x,y), & 1 \leq x,y \leq 128 \\ 0, & 129 \leq x,y \leq 256 \end{cases} \qquad (3.15)$$

We have to mention that the addition of zeros at the end of the signal does not add any extra information and therefore the FFT of the zero padded image does not carry more information. But as there are more samples, the features in the frequency domain are more clear, and their interpolation is much easier. As in most of the natural images the DC response is much larger than the rest [9], this rises some additional problems in the application of the zero padding technique, because the DC value introduce a sinc like ripple quite strong that make the identification of the motion blur direction harder. One way to reduce that artifact is to zero the mean of the image. If the mean value of the image is zero we get much more presentable results, therefore in the algorithm we subtract each pixel by the mean value of the image and then we zero padd the image (see equation 3.16).

$$PaddedImg(x,y) = \begin{cases} Img(x,y) - Mean(Img), & 1 \leq x,y \leq 128 \\ 0, & 129 \leq x,y \leq 256 \end{cases} \qquad (3.16)$$

In the Figure 3.10 we have the Fourier Transforms of the same part of a blurred image with and without the zero padding. In figure 3.10a the FFT of a 64×64 part



(a) (b) (c)

Figure 3.10: The *Fourier Transform* of, (a) an image patch, (b) an Zero Padded image patch, (c) a Zero Padded, Gaussian Masked, image patch

of a blurred image is displayed; in 3.10a the same patch has been zero padded up to 128×128, and it is quite clear that a lot more details can be seen. Unfortunately, the ringing effect is also magnified by the zero padding. In order to get better results, first we mask the 64×64 part with a Gaussian window of the same size, and then we zero padd it into 128×128; the *FFT* of the Gaussian masked, zero padded image can be seen in image 3.10c. In order to compare it, the *FFT* of the Gaussian masked image patch is presented in figure 3.9b.

3.3 Steerable Filters

The next step in the algorithm is to extract the direction of the motion. As we have seen in the previous section, the Power Spectrum of the blurred image is characterised by a central ripple that goes across the direction of the motion. In order to extract this orientation a linear filter is applied; more specifically the second derivative of a two dimensional Gaussian is used. In figure 3.11a we could see the second derivative of the Gaussian along the x-axis ($G_2^0 = \frac{\partial^2 G}{\partial x^2}$)[5] if we filter the Power Spectrum of a blurred image with G_2^0 we are going to get maximum response when the ripple is across the x-axis as it is in figure 3.5b. Therefore, in order to extract the orientation of the ripple, we have to find the angle θ in which the filter of the second derivative of a Gaussian – oriented at that angle (G_2^θ) – is going to give the highest response.

The calculation of oriented filters has been a field of interest in Computer Vision and Image Processing research [10]. In a lot of cases it is necessary to know the orientation at which a filter is going to give maximum response, or to be able to construct a filter at a specific angle; it has been proven that there exist families of filters that are possible to be constructed based only on the responses of a minimum set of basic filters. In order to find the highest response of an oriented filter we could apply the same filter at different angles, changing the orientation by a $d\theta$ up until all possible angles are covered, unfortunately, this is going to be time consuming, because every time we apply a filter $n \times n$ it costs $O(n^2)$ computations. Another way is to construct the response at every possible angle based on the response of a small set of orientations. This can be done for certain types of filters by applying them in a few selected angles and then take the responses and interpolate among them in order to get the wanted angle. Filters that can be constructed as a linear combination of a few basis filters are called *steerable*. The functions that combine the basis filters are called *interpolation functions*.

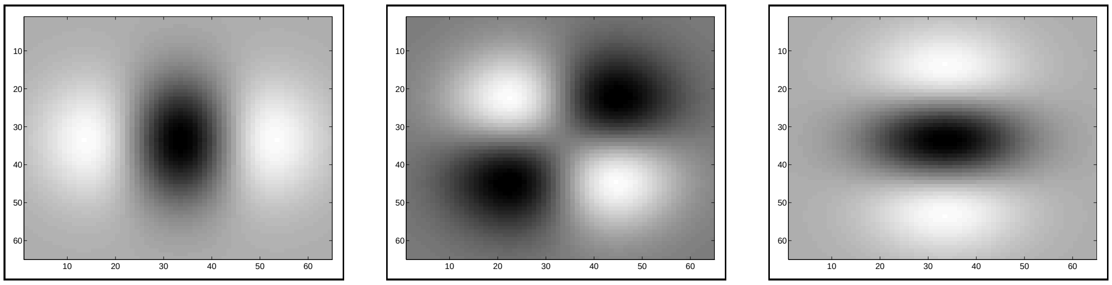
Fortunately the second derivative of the Gaussian G_2^θ belongs to such a family [10], and we can calculate its response at any angle θ based on the responses of the three *basis filters* as shown in the left column of the table 3.1, in equation 3.17 we could see the calculation; in table 3.1 at the right we could see the three *interpolation functions* that are used.

[5] The dark area has the highest value and the bright the lowest

$$RG_2^{\theta} = k_a(\theta)RG_{2a} + k_b(\theta)RG_{2b} + k_c(\theta)RG_{2c} \tag{3.17}$$

$G_{2a} = 0.9213(2x^2 - 1)e^{-(x^2+y^2)}$	$k_a(\theta) = \cos^2(\theta)$
$G_{2b} = 1.843xye^{-(x^2+y^2)}$	$k_b(\theta) = -2\cos(\theta)\sin(\theta)$
$G_{2c} = 0.9213(2y^2 - 1)e^{-(x^2+y^2)}$	$k_c(\theta) = \sin^2(\theta)$

Table 3.1: The three *basis filters* and their *interpolation functions*



(a) (b) (c)

Figure 3.11: The three masks used in the Steerable Filter calculation.

In figure 3.11 we have the pictures of the three *basis filters* as we use them in the algorithm for processing a 64×64 patch of the *Power Spectrum*. They are 64×64 windows centred at zero with values at $[-2, 2]$.

Assuming a 64×64 patch of the image, we take the logarithm of the *Fourier Spectrum* $(FImg)$[6]; then we calculate the response for each of the *basis filters* G_{2a}, G_{2b}, G_{2c}

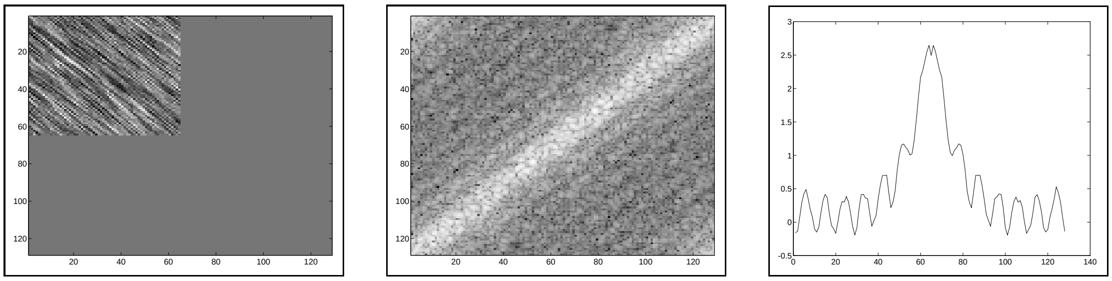
[6] If we have use zero padding then the size of the *Fourier Spectrum* is larger.

with the $FImg$ (see equations 3.18, 3.19, 3.20), and after that we use these responses $RG_{2a}, RG_{2b}, RG_{2c}$ to calculate the responses for all the different orientation in the space $[0°, 180°]$ with a step of $1°$ using the equation 3.17, finding the maximum that corresponds to the correct orientation. The computation cost in this stage for an $n \times n$ image patch is $\mathrm{O}(n^2)$.

$$RG_{2a} = \sum_{x,y=1}^{64} G_{2a}(x,y)FImg(x,y) \tag{3.18}$$

$$RG_{2a} = \sum_{x,y=1}^{64} G_{2b}(x,y)FImg(x,y) \tag{3.19}$$

$$RG_{2c} = \sum_{x,y=1}^{64} G_{2c}(x,y)FImg(x,y) \tag{3.20}$$



(a) (b) (c)

Figure 3.12: A zero padded image patch, its *Fourier Spectrum*, and then the *Fourier Spectrum* collapsed

To sum up, assume we have a random dot picture, artificially blurred at a 45° angle with a magnitude of 10 pixels; this is a quite simple situation and without the presence of any noise. If we take an image patch 64×64 and we try to calculate the optical flow vector we have seen up to now the following steps: first we zero padd the image (up to 128×128, figure 3.12a) then we take the logarithm of the *Fourier Spectrum* (figure 3.12b) on which we apply the steerable filters to extract the orientation of the blur, which turns to be 45°; in order to avoid distortions at the border values we take only the central 64×64 part of the *Fourier Spectrum*. In the next section we are going to see how and why we have to preprocess the signal in order to extract the magnitude of the optical flow vector.

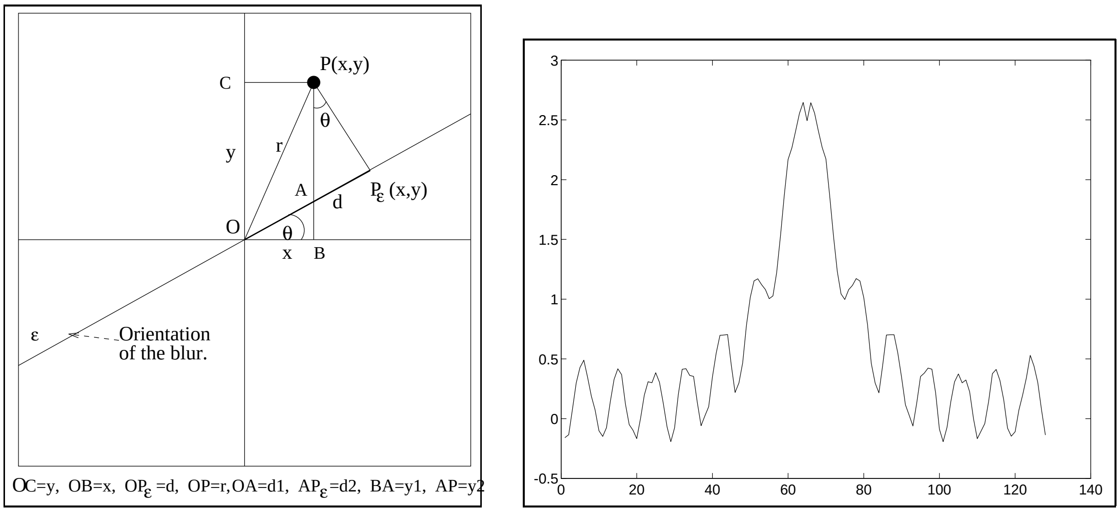
3.4 Transform the *Fourier Spectrum* into 1D

After we have calculated the orientation of the optical flow vector, the next step is to extract its magnitude. Unfortunately, the different artifacts that appear in the *Fourier Spectrum* of the blurred image make it really difficult to distinguish the size of the ripple and consequently the length of the blur. The artifacts are due to two reasons: one is the windowing effect, which exist even after the use of a Gaussian masking window, and the second is that the Fourier Transform of an unblurred image has already a certain structure that in a lot of cases changes the appearance of the motion blur ripple. Moreover, the magnitude is a scalar value, therefore it can be extracted by an 1D signal. The main idea is to create an 1D signal that is an approximation of the *Fourier Spectrum* transformed from 2D into 1D.

3.4.1 Collapse the *Fourier Spectrum*

In order to collapse the *Fourier Spectrum* we have to project every pixel into the line that passes through the origin with the same orientation as the motion blur. As can be seen in figure 3.13a a pixel P(x,y) in the image is going to be orthogonally projected into the line ϵ – that passes through the origin O at an angle θ with the x-axis – at the point $P_\epsilon(x, y)$ at distance d from the origin. The main task here is to calculate the distance d from the origin O. By applying the definition of the sin and cos in the figure 3.13a we have that the distance $d = x\cos(\theta) + y\sin(\theta)$.

$$d = x\cos(\theta) + y\sin(\theta) \tag{3.21}$$



(a) (b)

Figure 3.13: Collapsing the 2D data into 1D along the orientation of the blur, and the *Fourier Spectrum* of picture 3.12a

Since we work in the discrete space, the distance d has to be digitised. That means that every pixel $P(x, y)$ as it is mapped into a position $P_\epsilon(d)$ along the direction of the blur, it is going to affect in fact two values into the 1D signal $P_\epsilon(\lfloor d \rfloor)$, $P_\epsilon(\lfloor d \rfloor + 1)$. More specifically, assume that the pixel $P(x, y)$ is mapped at the distance $d = 5.6$ so it is going to contribute 40% of its value at $P_\epsilon(5)$ and 60% at $P_\epsilon(6)$. That way the mapping is much more accurate than by simply assigning the whole value of each pixel into one position in the 1D signal. Another issue that we have to take into account is the distribution of the pixels along the line. The *Fourier spectrum* is calculated in a square window, therefore the number of pixels that affect the central part is much larger than the number of pixels that affect the two ends. In order to have a uniform distribution, when we do the mapping of the pixels we assign in each position of the signal two values: the pixel value and a weight depending on the amount this pixel is contributing. In the end we normalise the 1D signal by dividing it by its accumulated weight. The collapse of the 128×128 *Fourier spectrum* of figure 3.12b is a 128, 1D signal and can be seen in figure 3.13b.

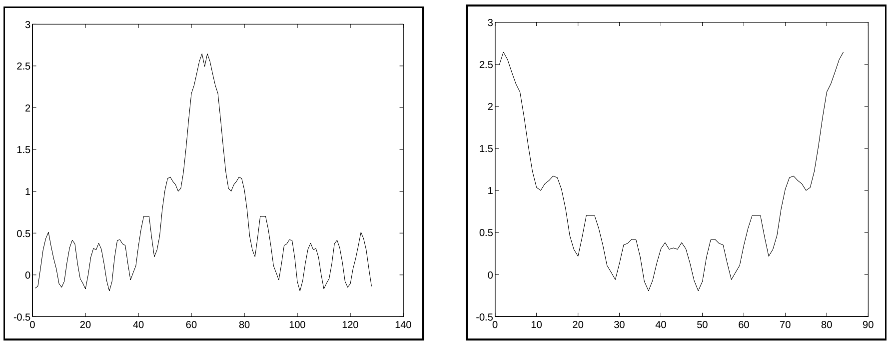
3.4.2 Normalisation of the data

One of the properties of Fourier Transform is symmetry; that means that a signal of n samples[7] is symmetrical around $\frac{n}{2}+1$. For example, for a signal P that has values at $[1 \ldots 128]$, the value $P(1)$ and $P(65)$ are unique and then $P(65-i) = P(65+i)$ for i in $[1 \ldots 63]$. Unfortunately when we collapse the 2D *Fourier spectrum* into 1D this condition does not hold. First of all small numerical and round off errors appear; second, the different artifacts that are due to the ringing effect and also due to the features of the *Fourier spectrum* of the unblurred image aren't symmetrical around the direction of the motion blur; third, the weighting process contributes some more round off errors into the newly created 1D signal. In order to achieve better results we average the values of the 1D signal with respect to the symmetry property. More specifically, the value $P_\epsilon(1)$ and $P_\epsilon(\frac{n}{2}+1)$ are kept the same, and then for the rest of the values the equation 3.22 is used.

$$
\begin{aligned}
P_\epsilon(\frac{n}{2}+1+i) &= P_\epsilon(\frac{n}{2}+1-i) \\
&= \frac{P_\epsilon(\frac{n}{2}+1+i)+P_\epsilon(\frac{n}{2}+1-i)}{2} \qquad 1 \leq i \leq \frac{n}{2}-1 \qquad (3.22)
\end{aligned}
$$

As can be seen in figure 3.14a the values at the borders are not as important as in the middle, therefore in order to achieve better results we keep only the central part of the signal. The next step is to shift in time in order to have the exact shape of the Fourier Transform. The result can been seen in figure 3.14b.

[7] n is usually a power of 2.



(a) (b)

Figure 3.14: The collapsed *Fourier Spectrum* normalised, and then shifted only the central part.

3.5 Cepstrum

In order to proceed to the next step, which is the calculation of the magnitude, a new tool needs to be defined. The power spectrum of an image is a signal by itself, therefore different features that appear in it can be extracted using classical signal processing techniques, such as edge extraction, Fourier Transform filtering, and so on. In order to identify the ripple that appears in the image a technique called *cepstral analysis* is used.

3.5.1 Definition of the Cepstrum

Different definitions have been given for the *Cepstrum* depending on the different application that was used. The most common definition of the *Cepstrum* [8] can be seen at equation 3.23, where $F(\omega, v)$ is the Fourier Transform of a function $f(x, y)$ [20],[19]. In other words, it is the *Inverse Fourier Transform* of the logarithm of the *Fourier Transform* of the signal. The *Cepstrum* calculated by equation 3.23 is a

[8] *Cepstrum* is a juxtaposition of letters for the word *Spectrum*

complex function; if we want to have only the real part then instead of the $F(\omega, v)$ we take its magnitude $|F(\omega, v)|$ (which is the case in this algorithm) as in equation 3.24. For the sake of simplicity, a more general definition is given in Eqn. 3.25 [20] where the Z-transform is used. In this case the *Cepstrum* of an 1D signal is the Z-transform of the natural logarithm of the Z-transform of the original 1D signal[9]

$$\mathcal{C}ep\{f(x,y)\} = \mathcal{F}^{-1}\{\log\left(F(\omega, v)\right)\} \tag{3.23}$$

$$\mathcal{C}ep\{f(x,y)\} = \mathcal{F}^{-1}\{\log\left(1 + |F(\omega, v)|\right)\} \tag{3.24}$$

$$c_x(n) = \frac{1}{2\pi j} \int lnX(z) z^{n-1} dz \tag{3.25}$$

The *cepstrum* has been found useful in a whole set of different applications. In one dimensional signal processing it has been used in speech recognition for echo detection; also early in image processing it has been used in nonlinear filtering for image enhancement [19], where the logarithm of the Fourier Transform is amplifying the information in the Frequency domain and the inverse Fourier Transform is used to filter certain features. Another application in which it has been used is passive Stereopsis [15]. More specifically, in Monocular Stereopsis we take a picture with a camera with two pinholes, creating therefore an *echo* in the image; as the echo is extended across the x-axis we could process each line as an 1D signal and detect the echo – which is equivalent to the disparity – using the *Cepstrum*. Finally another area where the *Cepstral* analysis has been used is in Optical Flow estimation when the motion is known to be uniform across the X-axis and the only unknown is the magnitude of the velocity vector [8]. As we have see in section 2.2.2 the Fourier transform of the motion blur PSF is of the form of a sinc ripple therefore it can be easily identified by the 1D Cepstrum.

[9] This definition can be also easily extended into 2D.

3.5.2 Calculation of the Cepstrum

As we see in the previous sections we have transformed the logarithm of the Power Spectrum of the blurred image into an 1D signal. This new signal has approximately the shape of a sinc ripple – distortions exist due to noise, windowing effect, and the process of collapsing the signal itself. The real part of the Cepstrum is used in order to estimate the length of the ripple, which is in fact the magnitude of the velocity vector. The signal we have is an artificial average signal of the logarithm of the Power Spectrum of the image. This has the advantage that the features in the Power Spectrum that were there due to the unblurred image have been cancelled out, leaving as a prominent characteristic the effect of the motion blur. As the 2D signal is collapsed across the direction of the motion it simulates a motion blur created by uniform movement across the x-axis and has the appearance of the sinc= $\frac{\sin x}{x}$, as can be seen easily by comparing the figures 3.14a – the collapsed signal, and figure 3.15 – the graphical representation of the sinc function.

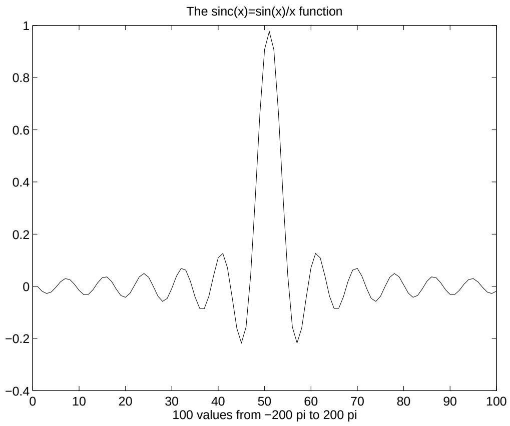


Figure 3.15: The Graphical representation of the sinc function

3.5.3 Information extraction from the Cepstrum

In this algorithm we assume that the velocities are bounded between 5 to 35 pixels per frame. Such an upper limit is logical as the ripple can not be identified if is larger, with the use of a window as small as 64×64. After the calculation of the 1D *Cepstrum* we search for a negative peak among the values in the interval [5...35].

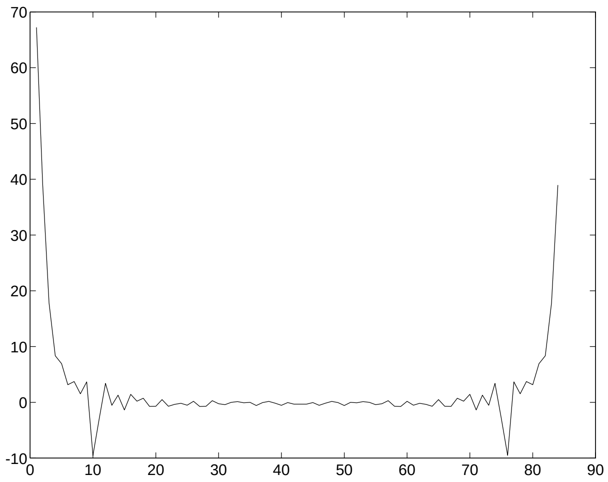


Figure 3.16: The Cepstrum of the image patch of image 3.12a

As can be seen from the plot in image 3.16, the first five values are heavily influenced by the DC value of the cepstrum and therefore unable to give us a robust answer; consequently this method works for exposure times that produce a blurred image. There is no noise reduction or validation at this point, although different techniques have been proposed in order to make more robust the magnitude extraction [8].

3.6 Complexity analysis of the algorithm

The previously described algorithm calculates the velocity vector for a point $I(x, y)$ making use of the information at the $n \times n$ image patch around it; usually, the size of the image segment n is a power of two, the most common sizes being $n = 64$

or $n = 128$. Therefore, for an $N = n \times n$ image segment the algorithm works in the stages: The *Gaussian Masking* which is $O(N)$, and the *Zero Padding*[10] which is $O(N)$, by just creating the new image patch. Then the most computationally expensive part is the *Fourier Transform*, which takes $O(N \log N)$[11]. The next step is the calculation of the maximum response for the steerable filters this is three times the application of the basis filters $O(N)$ and then $O(1)$ for calculating the correct angle. The collapsing of the Power Spectrum from 2D into 1D is linear, therefore the cost is again $O(N)$ and the normalisation is $O(n)$. The last step of the calculation of the magnitude of the velocity vector is the *Inverse Fourier Transform* of the 1D signal which has $O(n \log n)$ computational cost. Finding the negative peak in the space [5...35] is again constant.

As can be seen from the previous analysis for an image patch with a size $N = n \times n$ the computational cost is $O(N \log (N))$. In order to calculate the optical flow of a motion blurred image $m \times m$ in grid every δ pixels with a window $n \times n$ we have to apply the algorithm $M = \frac{m-n}{\delta} \times \frac{m-n}{\delta}$ times; therefore, the general cost is $O(M\{N \log (N)\})$. One of the advantages is that the algorithm could be run in parallel for every velocity vector as there is no need of sharing intermediate results. Also it is easy to develop a hardware implementation of the algorithm, as the most complicated step is the calculation of the *Fourier Transform*.

[10]usually doubles the size from $N = n \times n$ to $N' = 2n \times 2n$

[11]From now on we assume N to be the size of the zero padded segment.

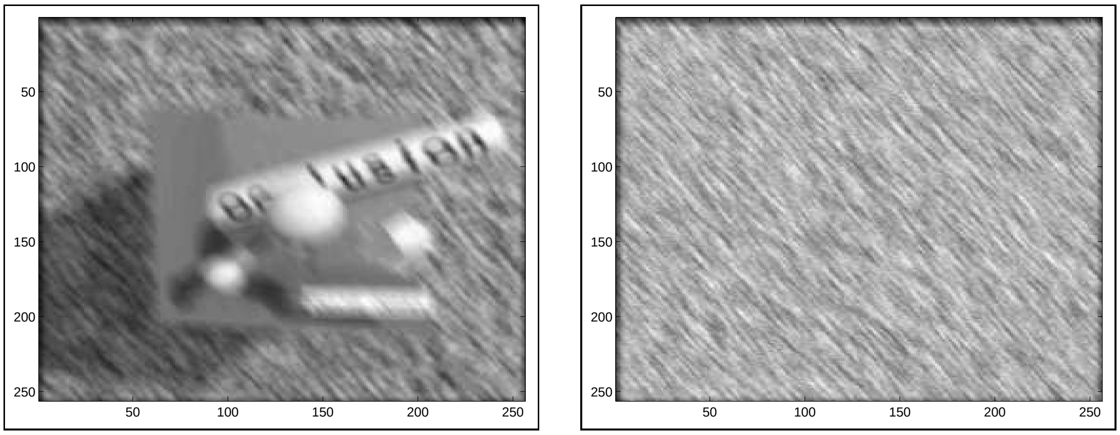
Chapter 4

Experimental Results

In this chapter, the new motion blur algorithm developed in Chapter 3 will be analysed by evaluating its performance on several experiments involving blurred images of various kinds. There exist two categories of input data that we are using. The first category consists of stationary images, natural or artificially created, that we artificially blur by simulating the results of motion blur; in the second category, are real images taken by a camera with the existence of relative motion between the camera and the scene. The data from the first category give us the ability to check the validity of our results and perform error measurements, while the images from the second category are ensuring that the algorithm is working on real world data. In the last part of this chapter an error analysis is performed, on the artificially blurred images for which the exact results are already known. For the images from the real world only a qualitative analysis is possible as we don't know the correct values before hand.

4.1 Artificial data

Two images have been used in this section, each one of them having different properties. The first one (figure 4.1a) is a real image taken by a stationary camera, which has a whole set of different features such as smooth surfaces, edges, and highly textured areas, with size 256×256 pixels. The second one (figure 4.1b) is a random noise picture, having the same size with the previous one. This image is rich in texture. As we discussed in the previous chapter the algorithm is more effective when there exist a lot of texture, this is quite obvious in the results we get, where there exist more incorrect estimations at the places where there are smooth surfaces.



(a) (b)

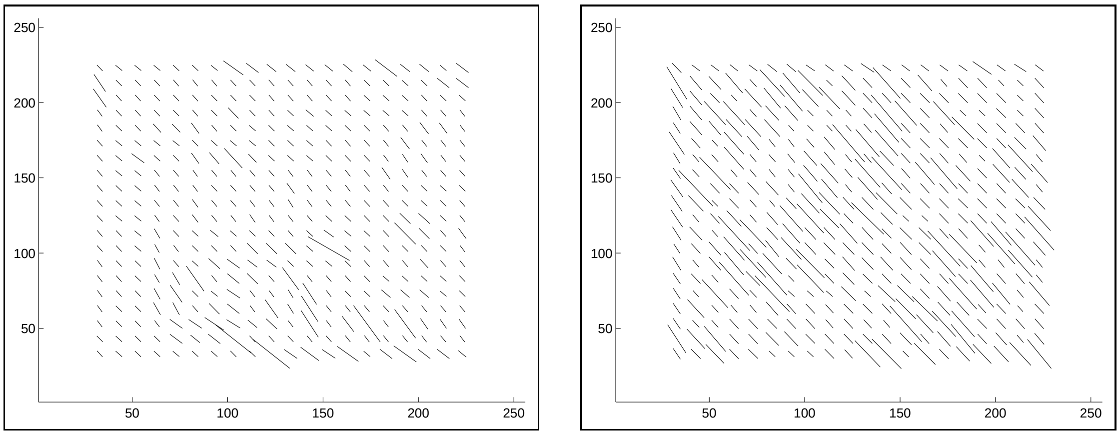
Figure 4.1: Two artificially blurred images (a) a natural image (b) a random noise image.

Both images have been blurred with the same kernel. The motion is assumed to be at a direction of $+125°$ angle with the x-axis (or $-55°$) and with a length of 13 pixels. In order to create the artificial blur we convolve the image with a kernel as in table 4.1. In real world the blur is created before the digitisation, therefore the points that contribute to the final value of the pixel exist in a straight line. When we try to reproduce the same results in the discrete space and we use an arbitrary angle, we have similar results to the aliasing. In other words, if we digitise the line into discrete steps with sudden changes, then we have a stair-case effect. In order to avoid that, the table is created by using the technique of antialiasing lines, as can be seen in table 4.1 where the pixels are weighted according to their distance to the "abstract" line.

0	0	0	0	0	0	0	0	0	0	0	0	0
0	**0.5**	**1**	**0.5**	0	0	0	0	0	0	0	0	0
0	0	**0.5**	**1**	**0.5**	0	0	0	0	0	0	0	0
0	0	**0.5**	**1**	**0.5**	0	0	0	0	0	0	0	0
0	0	0	**0.5**	**1**	**0.5**	0	0	0	0	0	0	0
0	0	0	0	**0.5**	**1**	**0.5**	0	0	0	0	0	0
0	0	0	0	0	**0.5**	**1**	**0.5**	0	0	0	0	0
0	0	0	0	0	**0.5**	**1**	**0.5**	0	0	0	0	0
0	0	0	0	0	0	**0.5**	**1**	**0.5**	0	0	0	0
0	0	0	0	0	0	0	**0.5**	**1**	**0.5**	0	0	0
0	0	0	0	0	0	0	**0.5**	**1**	**0.5**	0	0	0
0	0	0	0	0	0	0	0	**0.5**	**1**	**0.5**	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4.1: The convolution matrix for the motion blur, using antialising lines.

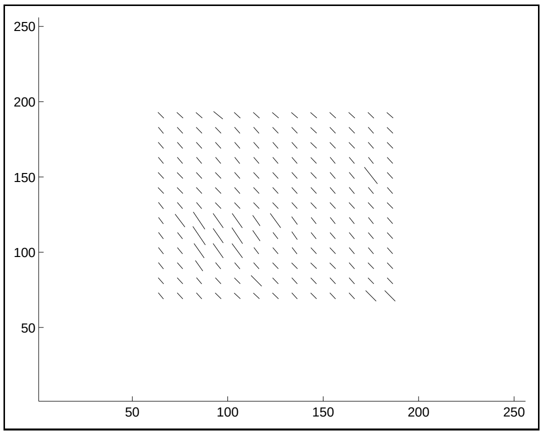
In the next pages we are going to present the Optical Flow maps for the previous two images (figure 4.1 a and b), using different configurations for the calculations; in the following set of images the left one (a) represents the natural image, and the right one (b) represents the random noise image. In the first set of images (figure 4.2) we use a 64×64 window for every velocity vector we calculate – leaving therefore a 32 pixel border around the image where we can not estimate the Optical Flow. We calculate the Optical Flow in a grid that has a density of ten pixels, and we zero pad every window up to 128×128. As can be seen in image 4.2a, the orientation of the velocity is calculated correctly in most of the places; there exist of set of incorrect estimations in the area of the lower rows especially between 80 to 200 at the x-axis, corresponding to the presence of the areas that are characterised as smooth surfaces. The magnitude of the velocity is more or less uniform. The Optical Flow of the random noise image (figure 4.2b) presents a uniform orientation estimation; there are not any areas where we have results that deviate from the correct direction. However, the magnitude estimation is not satisfactory, a fact which is mainly due to the size of the window, which is not big enough so we could have enough data at the frequency domain in order to calculate the cepstrum.



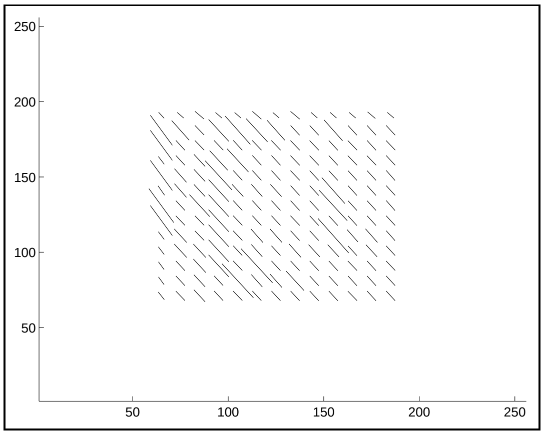
(a) (b)

Figure 4.2: The Optical Flow of the two artificially blurred images using a 64×64 window with a step of 10 pixels, only with zero padding

For the next pair of figures (4.3) a bigger window is used, in this case a 128×128 window. The same conditions as in the previous experiment were kept – a 10 pixel dense grid is used and the transformation to the frequency domain in done by using simple zero padding that transforms the window from 128×128 into 256×256. That way, although a big part of the image stays unfortunately without any velocity estimations, a more robust Optical Flow map is calculated. In the left map (figure 4.3a) that represents the Optical Flow of the natural image, the estimation of the orientation is mostly accurate, and there are only a few incorrect estimations for the magnitude; this is mainly due to the bigger size which gives us much more data to extrapolate the results. In the velocity map of the random noise image (figure 4.3b) we could see that the orientation is also correctly calculated although some incorrect results exists in the magnitude estimation.



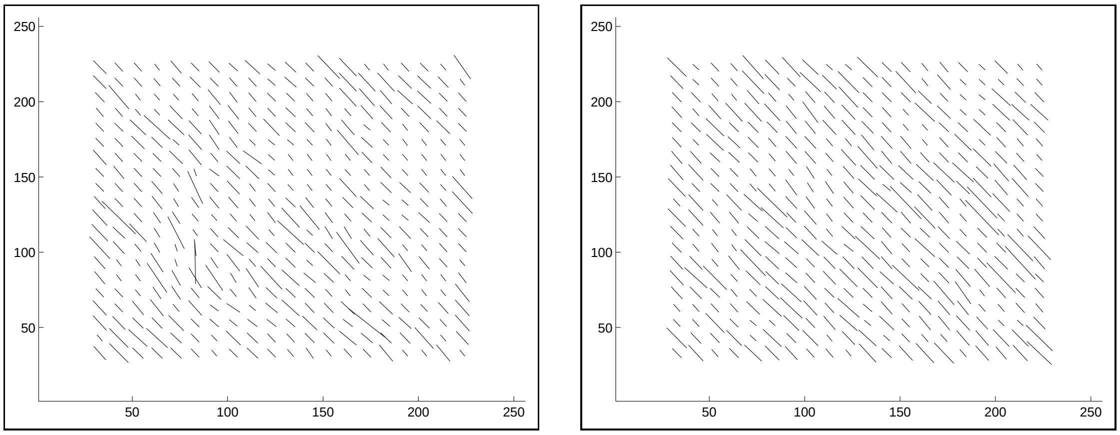
(a)



(b)

Figure 4.3: The Optical Flow of the two artificially blurred images using a 128×128 window with a step of 10 pixels, only with zero padding

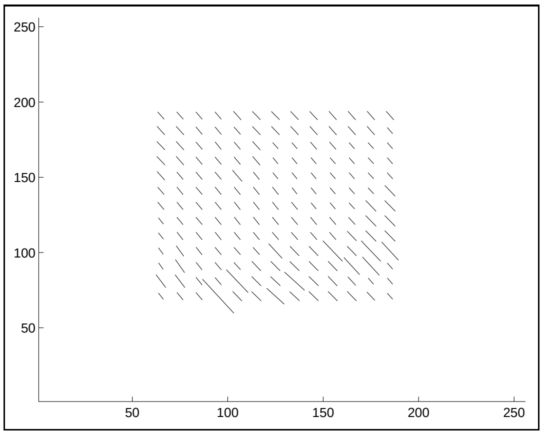
In order to get better results and to eliminate the ringing effect, a Gaussian window is used for masking before we proceed into the velocity vector estimation. The next figure (4.4) presents the Optical Flow maps created by using a 64×64 window which is masked with a 2D Gaussian window (of the same size) and then zero padded up to 128×128. As can be seen in figure 4.4a, the Gaussian masking causes an improvement into the orientation estimation by eliminating the ringing effect, but it interferes with the magnitude extraction from the cepstrum. Considering the random noise image, the Optical Flow estimation is robust as far as it concerns the orientation but in the magnitude estimation some incorrect results are still present. The estimations are improved considerably by using an even bigger window. As it is obvious from figure 4.5 the results are much more accurate. In the Optical Flow map of figure 4.5a the orientation is correct and there is only a small neighbourhood where there is a miscalculation of the magnitude. Almost the same is true for the random noise image which its Optical Flow map is presented in 4.5b.



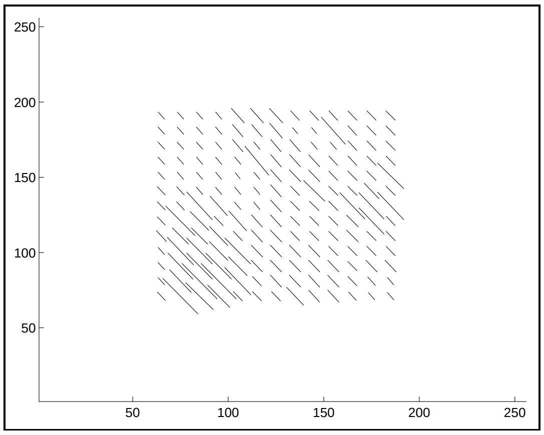
(a) (b)

Figure 4.4: The Optical Flow of the two artificially blurred images using a 64×64 window with a step of 10 pixels, with zero padding and Gaussian masking

To sum up the results from the simulated blurred images, we have to highlight some points. First of all the blur that was chosen was completely random; a blur at $55°$ with length 13 pixels has no regularity and therefore the construction of it creates some numerical errors. As it is going to be clear from Optical Flow maps of the natural images, the results are more robust. Secondly, different areas of the image respond better in different approaches, for example in certain areas the ringing effect is dominant compared to the blurred image signal, and in others simply the zero



(a)



(b)

Figure 4.5: The Optical Flow of the two artificially blurred images using a 128×128 window with a step of 10 pixels, with zero padding and Gaussian masking

pad is enough. In the next section we are going to demonstrate how the algorithm works on data taken with a camera from the real world. We are going to come back to artificially blurred images in the third section where we are going to do a quantitative error analysis of the results.

4.2 Natural data

The images in this case have been taken by a camera and immediately digitised into the computer. In order to have controlled motion between the camera and the scene the following setup was used: a camera was mounted on a base pointing downwards, and a plane (created by cardboard) with random dots on top of it was used as the main object in the scene. We moved the plane in different directions, sometimes having small objects on it, with a speed high enough to produce motion blur with the preset exposure time of the camera. The setup can be seen at the drawing in figure 4.6; in this case the plane is falling simply by its weight. During the different motions a frame grabber has been used to freeze the image into the computer; the speed was fast enough so that the picture is blurred and the Optical Flow map can be calculated. In the following part we are going to see blurred images created by different motions and their equivalent Optical Flow map. The format, for economy of space, consists of three different blurred images, labelled (A), (B), (C) in one figure, and their respectively Optical Flow maps in a second figure, following the same labelling. In all the experiments the same configurations have been used: we calculate the Optical Flow on a grid which was dense 10×10, using a 64×64 window. The patch of the blurred image was masked first with a Gaussian window (to avoid the ringing effect) and then zero padded up to 128×128.

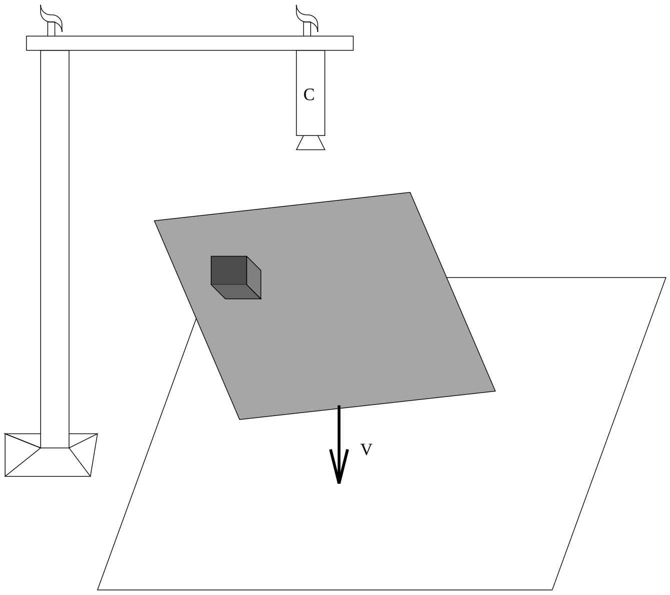


Figure 4.6: The camera setup with the plane falling downwards.

The first set of images is shown in figure 4.7. The first image 4.7A has been created by moving the plane in parallel with the y-axis with a steady and relative small velocity; the algorithm has correctly estimated the orientation of the velocity almost everywhere, as can be seen in the Optical Flow map in figure 4.8A. The accuracy of the magnitude estimation is not clear, although if we compare it with the next image some qualitative results can be drawn. The second image, 4.7B, is created again with a steady velocity parallel to the y-axis, this time at a higher speed, fact which is easily noticeable by the length of the blur. Again the Optical Flow map, in figure 4.8B, has an accurate estimation of the orientation and also gives an average bigger magnitude for the velocity vectors. By comparing these two cases it is obvious that the orientation estimation is correct and also the magnitude estimation shows the difference between different speeds. The third image 4.7C is created completely differently; the random-dot decorated plane is left to fall free under the camera and during that fall we take a snapshot. As can be seen from the blur lines the focus of expansion is at the middle of the left side, and indeed the algorithm gives the same results. In the Optical Flow map (figure 4.8C) we could see the velocity vectors pointing at the point of expansion and have a gradually decreasing magnitude as they reach that point.

The next set of images, created with free fall away from the camera, appears in figure 4.9. The first image 4.9A is taken very shortly after the fall begun, with the focus of expansion at the centre of the image; this is easily identified by the pixels at the centre where there is almost no blur at all and the size of the random dots relatively big compared with the dots in the other images. Accurately enough, the map of the Optical Flow in figure 4.10A shows the velocity vectors to converge in the middle where again their magnitude decreases. The middle image (4.9B) was also created by a falling plane, but this time the centre of expansion is at the left side in the lower part, and the blur is not big as the random dots are almost distinguishable. In fact, the velocity vectors in figure 4.10B are not very big and they correctly show the point of expansion. The third image is created differently; it involves again uniform motion across a line, although this time the angle with x-axis is almost at 45°. This image shows us that the algorithm can calculate the velocity vectors at an arbitrary angle. The Optical Flow map as presented in figure 4.10C describes correctly the orientation with correct magnitude in majority.

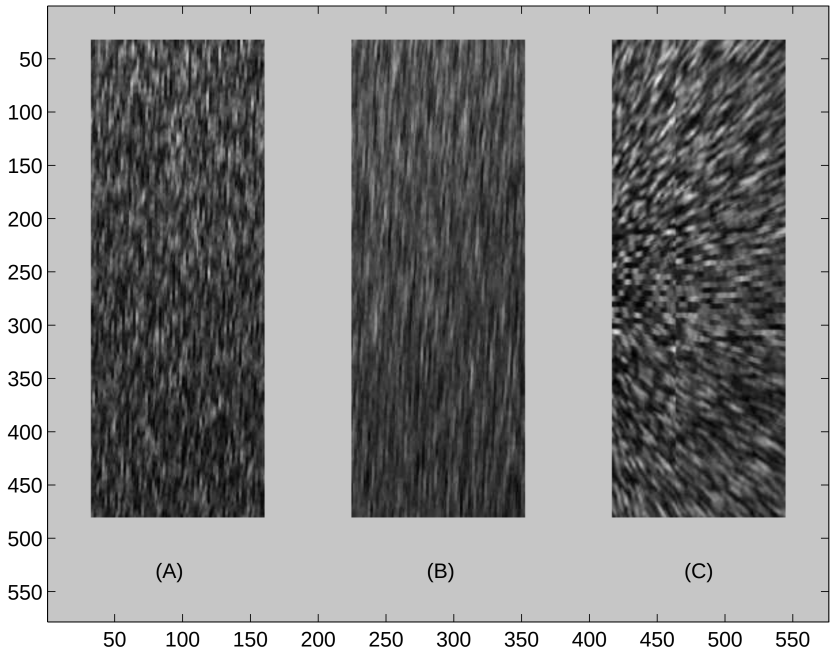


Figure 4.7: Three Images with motion blur

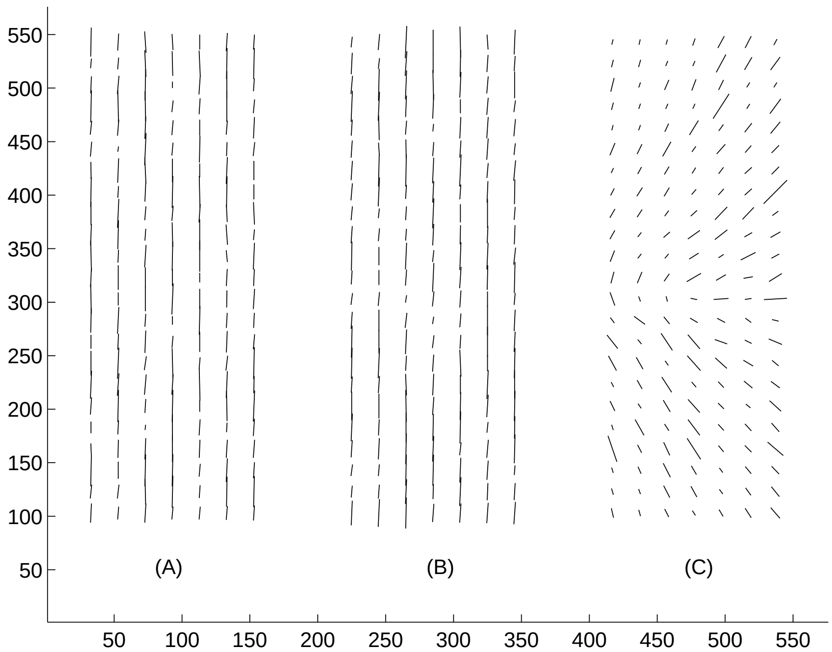


Figure 4.8: The Optical Flow map of the previous images using a 64×64 window with a step of 20 pixels, with zero padding and Gaussian masking

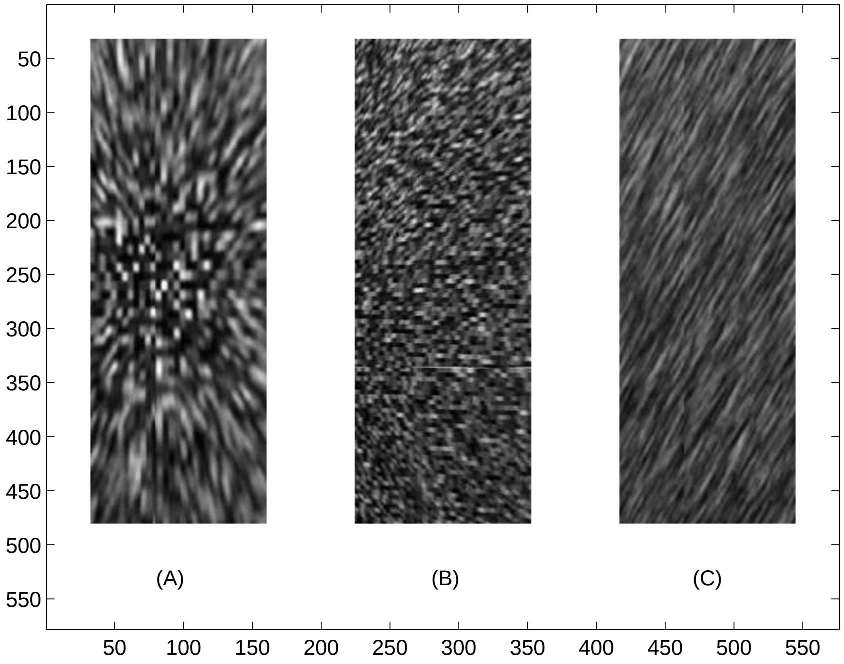


Figure 4.9: Three Images with motion blur

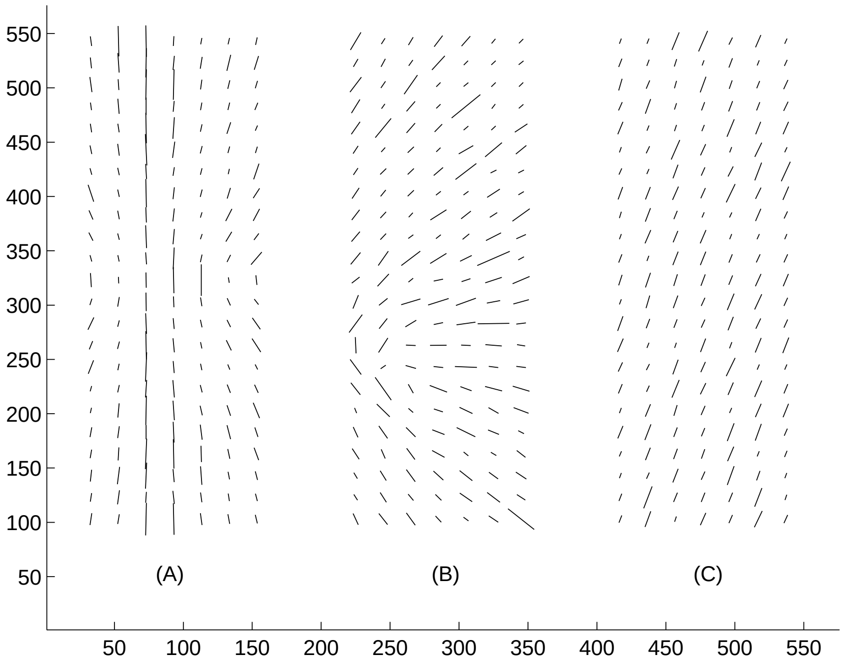


Figure 4.10: The Optical Flow map of the previous images using a 64×64 window with a step of 20 pixels, with zero padding and Gaussian masking

In the next set of images, some objects are placed on top of the random dot plane. Again uniform motion parallel to the y-axis was used, but this time four small objects were placed onto the plane, in order to see how they are going to affect the estimation of the velocity map. In figure 4.11A we could see the first image, where two circular pieces of cork appear in the upper part, the corner of a rectangle at the middle of the right side, and a small box in the middle of the lower part. In the Optical Flow map in figure 4.12A we could see the disturbance in the estimation that was caused by the lack of texture in these places, especially in the lower middle part. Exactly where the small box is in the blurred image, the velocity map shows a deviation at the orientation towards the right and the magnitude is incorrectly small; similar but smaller disturbance exist at the right side at the middle where the corner of the rectangle appears. The reason of this disturbance is also the lack of texture at these points. The same results appear also in the second image (figure 4.11B) where the snapshot was taken when the objects were translated towards the low left part of the image compared with the 4.11A. Again in the Optical Flow map (figure 4.12B) the velocity vectors are correctly calculated, except of the lower-left part of the image where the small box appears. The third image of this set presents a different kind of motion. In this case (figure 4.11C) the random dot plane was rotated bellow the camera creating a "galaxy" like pattern. The centre of rotation is located at the right side of the image slightly lower than the middle. The Optical Flow map (figure 4.12C) presents these results locating correctly the centre of rotation, assigning a very small magnitude to it, and arranging the velocity vectors circularly around it.

The third set of images have two more images created by rotational motion, but also an image created with a completely different setup. The first image (figure 4.13A) was created by moving the camera by hand horizontally across a self full of books and binders. The image is rotated by 90° due to the way *Matlab* is handling the images; taking that into account, the spiral binding of some of the books is quite obvious. Also the lighting of the scene was low and therefore some of the features didn't appear; in addition it is quite notable the lack of texture in a lot of the areas. In spite of these problems the velocity vectors in majority have the correct orientation and approximately the same magnitude, (figure 4.14A) results that agree with the blurred image. The last two images are created as before by rotating the random-dot plane under the camera. The middle image (figure 4.13B) is created by spinning the plane in high speed and that's how we get some almost continues blur lines; the centre of rotation is at the upper right part. In figure 4.14B we could see the velocity vectors having the proper orientation, and a rather big magnitude. The last instance, 4.13C, is taken with the plane considerably close to the camera and with a smaller rotation speed; the centre of rotation is at the upper left corner, and at that point the pixels are rather discrete. A smooth Optical Flow map is presented in figure 4.14C with the vectors having the correct orientation, circular around the upper left corner where the centre of rotation is, and with an almost constant magnitude.

There is a need to see the restrictions of this algorithm as well as its advantages. As we have already seen, the more information we have, e.g. more texture, or bigger window, the better the results; this introduces some constraints. First of all in a real world application, the texture we have in a certain image is given; this can change by examining the image at different scales, as for example, a carpet could be seen as a smooth gray surface, but if we zoom in we could have a very intricate pattern. In any case, if the texture is not enough we could not detect the motion; this is to be expected as any biological system has the same limitations. For example, if a uniform surface is moving and we could not detect any features then we could not infer the motion.

The second condition refers to the window size and has two side effects that ask for contradicting solutions. First, of all if the orientation of the velocity vector changes – as it is the case in rotation or free fall in the previously described examples, or in the simple case that inside the same window we have two or more objects moving to different directions – then there isn't one unique blur pattern to dominate the shape

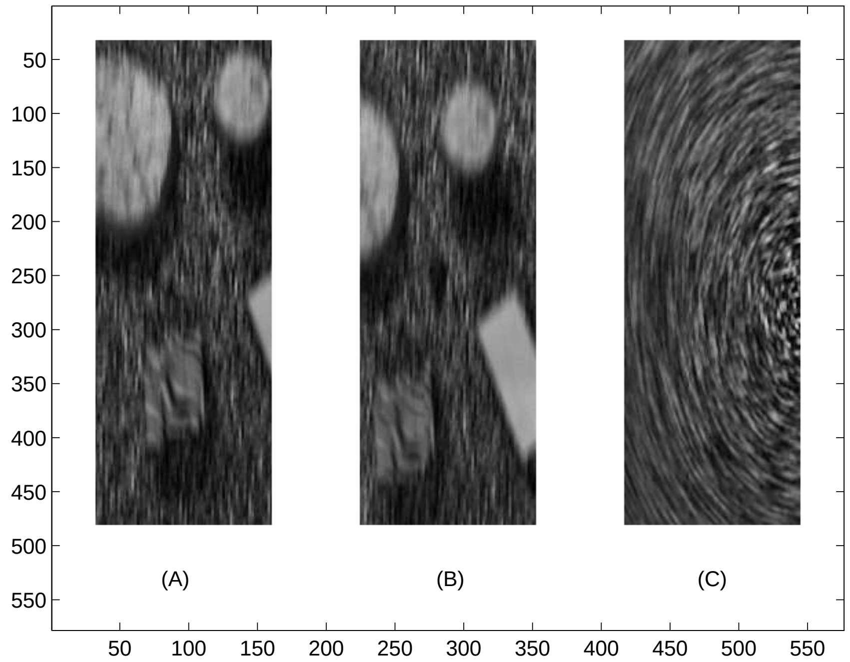


Figure 4.11: Three Images with motion blur

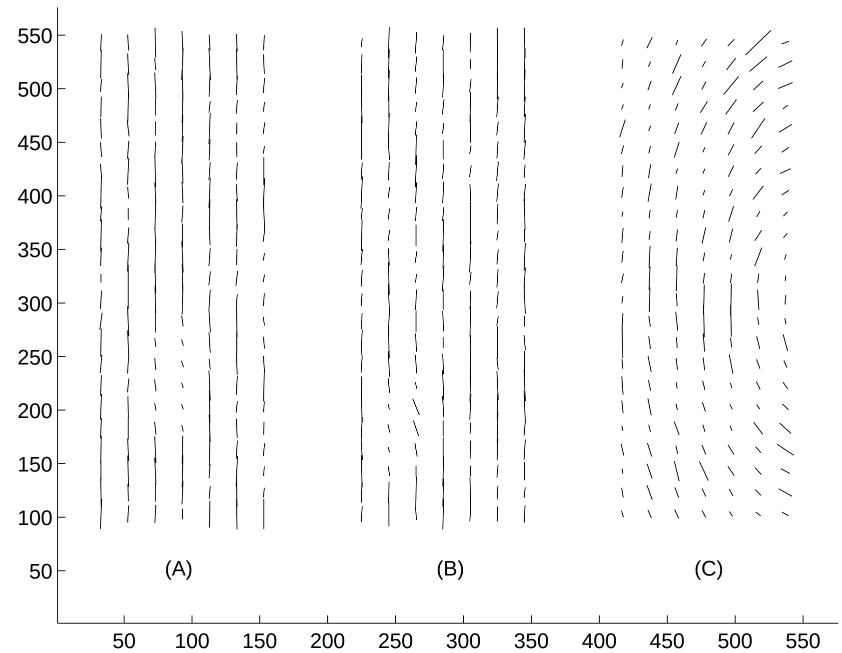


Figure 4.12: The Optical Flow map of the previous images using a 64×64 window with a step of 20 pixels, with zero padding and Gaussian masking

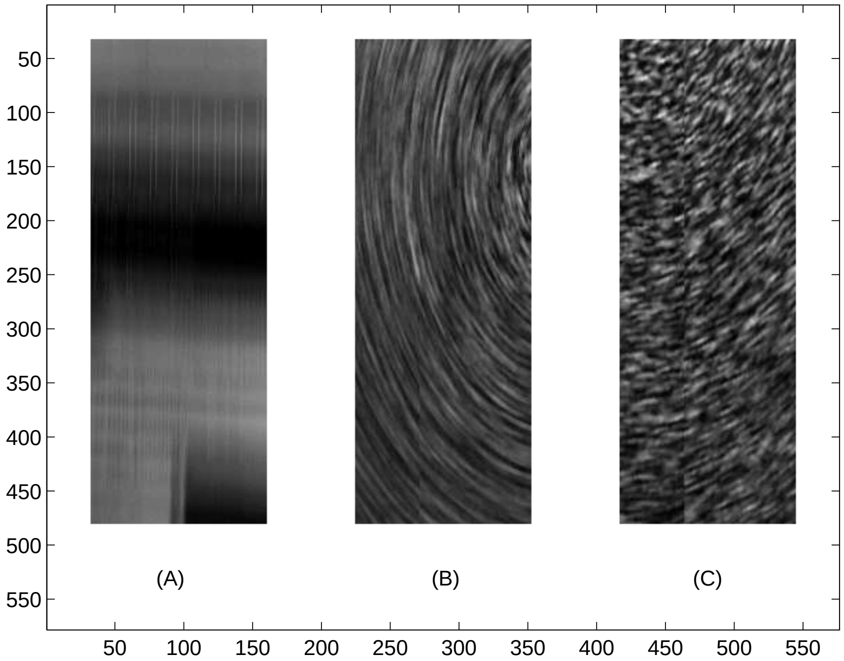


Figure 4.13: Three Images with motion blur

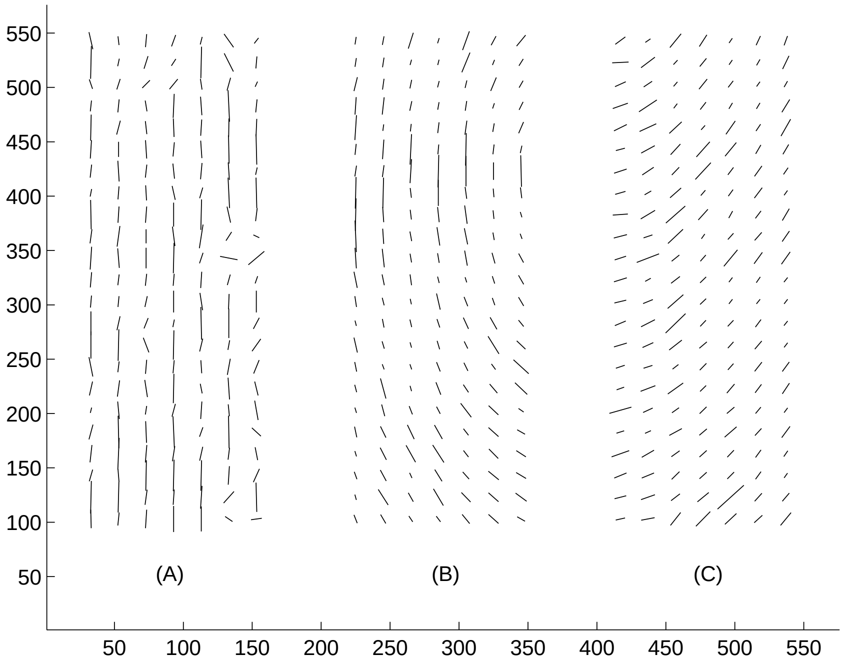


Figure 4.14: The Optical Flow map of the previous images using a 64×64 window with a step of 20 pixels, with zero padding and Gaussian masking

of the Power Spectrum. For such a case we have to use an appropriate window size, that assures approximately the same orientation of the motion blur for the entire window. In such a case the window size has an upper limit. This condition comes in contrast with the second side-effect. If the motion blur length is big enough then it is not possible to have an approximate representation of the motion blur ripple in the frequency domain, as we could see at most one or two periods of the ripple, and therefore it would not be possible to infer accurately the magnitude of the motion blur. The previously described problems show that the algorithm can not be used blindly. Also in a lot of cases, although an approximation of the Optical Flow map is obtained, we could not have accuracy; this makes the algorithm unsuitable for applications such as complete deblurring, which needs an accurate estimation of the motion blur *PSF* in order to restore the image, but it could still work on partial recovery of the image and also in a series of other applications such as inference of egomotion, time to crash estimation, moving obstacle detections and so on, where a sparser set of estimations is needed. In the next section we are going to use some artificially blurred images in order to measure the error in the estimated Optical Flow map.

4.3 Error analysis

Every velocity vector (in the Optical Flow map) consists of two numbers, the orientation which is given as the angle with the x-axis in degrees, and the magnitude measured in pixels. Therefore, the error analysis we are going to do, measures the errors created in these two estimations. The magnitude estimation follows the orientation estimation, using the calculated angle in order to compress the Power Spectrum from 2D into 1D, consequently if there exist an error in the orientation of the vector, this is going to propagate into the measurement of the magnitude; in order to avoid that, we also calculate the errors of the magnitude assuming a correct angle estimation. The error estimations that are presented in the following pages were created using the following methods: the Optical Flow maps were created on a grid with a density of ten pixels, and every velocity vector was created using two different sizes for the processing window, (so, we could see the improvement when we use more information –larger window size– for the calculations). The first column in every different method represents a window size of 64×64 pixels and the second 128×128 pixels. There were five different variations of the algorithm that were applied. The first variation (columns one and two) makes use of the complete algorithm with all the preprocessing stages, (Gaussian Masking and Zero Padding), and it is the most computationally expensive. The second variation (columns three and four) is using only Zero Padding, while the third one (columns five and six) is using only Gaussian Masking for the ringing effect. The fourth method (columns seven and eight) has no preprocessing at all, and all the calculations were applied at the raw data from the blurred image. Finally the last two columns present error estimations for the magnitude assuming correct orientation calculation. For every experiment (each variation with each size) there exist ten error measurements that are presented [1] in different rows. The first five error estimations measure deviations from the correct angle; the first is the mean value of the total number of errors in the angle estimation, which helps us to see how far from the correct orientation the general estimation points. As the mean value sum up the errors, negative and positive errors average into zero, therefore the mean of the absolute error is estimated in the second row, and gives a measure of the absolute error. The third row presents the Standard Deviation, and the fourth and fifth rows the maximum and minimum error respectively; as the error measurements are positive and negative these two estimations present the two larger errors in each direction (clockwise and counter-clockwise). All the angle error measurements are estimated in degrees. The following five rows have the same error

[1] The last variation with given orientation naturally doesn't have any angle error estimation.

estimations but this time for the magnitude of the velocity vectors measured in pixels.

The first two tables 4.2, 4.3 have the error estimations for the blurred images presented in figure 4.1a,b. In order to examine the influence of numerical errors inserted by the blurring processes (the use of an antialiasing line) the two images – the natural image and the random noise pattern – are blurred with a diagonal line (the 16×16 identity matrix) and an error analysis is presented in tables 4.4, 4.5.

ERROR	*Gaussian Padded*		*Zero Padding*		*Gaussian Masking*		*No Preprocessing*		*Known angle*	
Estimations	64p	128p	64p	128p	64p	128p	64p	128p	64p	128p
Mean angle	$0.6°$	$-0.1°$	$2.0°$	$0.2°$	$0.1°$	$-0.5°$	$1.7°$	$0.2°$	—	—
Mean ‖*angle*‖	$3.6°$	$1.6°$	$4.7°$	$2.4°$	$4.1°$	$2.1°$	$5.5°$	$2.6°$	—	—
S. Dev. angle	$2.1°$	$0.8°$	$1.2°$	$0.8°$	$2.1°$	$0.9°$	$1.4°$	$0.5°$	—	—
Max angle	$15°$	$6°$	$19°$	$9°$	$15°$	$6°$	$23°$	$7°$	—	—
Min angle	$-34°$	$-6°$	$-13°$	$-7°$	$-35°$	$-7°$	$-18°$	$-8°$	—	—
Mean length	-4.2	-5.9	-6.6	-7.3	-4.0	-5.2	-5.7	-6.8	-4.8	-6.2
Mean ‖*length*‖	5.2	6.4	7.2	7.3	5.5	5.9	6.8	7.4	5.6	6.4
S. Dev. length	1.1	1.5	1.6	1.2	1.3	1.4	1.2	2.2	1.1	1.2
Max length	15	15	15	1	17	12	15	13	12	9
Min length	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8

Table 4.2: Error Estimation for the blurred image of figure 4.1a, the vectors were estimated every 10 pixels. For a more detailed description of the table refer to the beginning of the section

The error estimations of the figure 4.1a are given in table 4.2. The qualitative observations that were made in section 4.1 can be verified here. As can be seen, in general the orientation is estimated much more robustly than the magnitude. In average, the error is as small as one tenth of a degree (not possible to detect) and at most two degrees, when a small window 64×64 with only zero padding was used; an improvement is also clearly detectable when we use a bigger window in every method. The same results hold when we take the average of the absolute error; this time, as we take into account every deviation from the correct orientation, the average error is higher but it still stays in acceptable levels having as maximum error 5.5 degrees, with no preprocessing and a relative small window. The standard deviation is small in almost all the cases. As far as it concerns extreme values of error, we have a positive deviation up to twenty three degrees (large number but rather rare as can be seen from the Optical flow maps in section 4.1; which is also appears in the worst situation of no preprocessing and with the use of a small window.) Rather interesting in the table is the decrease of extreme errors with the use of a larger window, for example the complete algorithm goes from $-34°$ for a 64×64 window to $-6°$ for a 128×128 window, and similar results hold for the rest of the variations. The magnitude errors are higher, but that is to be expected from the Optical Flow maps presented in the section 4.1. When we could not find a negative peak that signals the length of the ripple usually the algorithm picks the lowest value, which is the starting value (5 pixels in this configuration); that way the minimum length error is -8 pixels for all the cases. Although in average we don't have a major improvement when the orientation is given (this is due to the lowest possible estimation of -8 pixels that still exist) the maximum error decreases from 15 pixels to 9 pixels for a 128×128 window, and from 15 pixels to 12 pixels for a 64×64 window.

Another way to see the distribution of errors is presented in figure 4.15. We have created the error maps for different cases by displaying the error estimations that were used to construct table 4.2. The images 4.15a,b,c were created with a 64×64 window, where the images 4.15d,e,f were created with a 128×128 window. The first

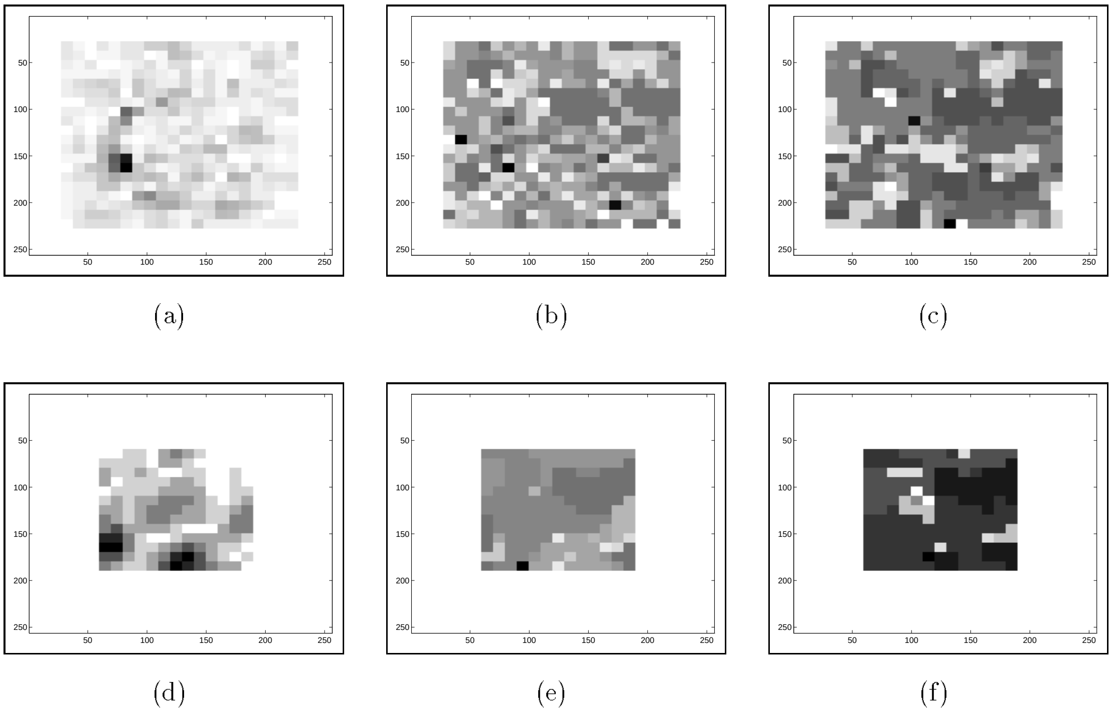


Figure 4.15: The Error Map for the image 4.1a for the orientation (a,d), the magnitude (b,e), and the magnitude with given the orientation (c,e); with a 64×64 window (a,b,c) and with a 128×128 window (d,e,f). Darker areas indicate larger relative error

column (4.15a,d) has the errors in the orientation calculated with using the complete algorithm, the middle images (4.15b,e) display the magnitude error estimated by the same algorithm; while the last column (4.15c,f) presents the magnitude errors when the orientation is given. The absolute value of the errors was used, with white for zero error, and black for the highest error value; also we have to mention that the black represents a different error value for each image and therefore, comparisons between different images based on the gray values can not be done. In order to analyse the error results we have to compare the areas in the error maps with the blurred images presented in 4.1a. In the first image we could see that the general error level is low with a high peak in two neighbouring areas, and by comparing these areas to the blurred image it is obvious that they are the places with minimum texture and therefore not enough information; the same holds for the error map of the larger window figure 4.15d. The magnitude error maps that are presented next have a more random distribution of errors, mainly gray (by checking the table with the average values,) due to the inability to estimate the magnitude in some positions and assigning the minimum value -5 to them; some peak results appear here also, and it is easily distinguishable where there are due to an orientation-estimation failure (by comparing with the error map in next column) and where there are not. An improvement is obvious from the small to the large window, where in the 128×128 window the bigger error is due to the error in the orientation estimation.

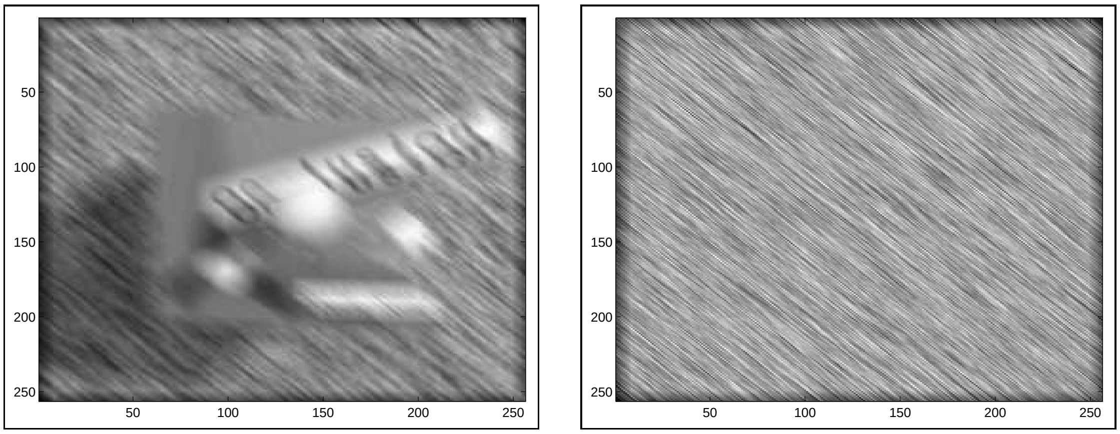
The error estimations for the random noise image (figure 4.1b) are given in table 4.3. Once again an analysis of the table confirms the observations from the Optical Flow maps in section 4.1 for the 4.1b blurred image. Also, a comparison between table 4.2 and table 4.3 highlights the importance of texture in the extraction of optical flow from the motion blur. In calculating the orientation of the motion blur the same improvement as before can be seen with the use of a bigger window, although, as the image is full of texture (random values), there is not much improvement with the use of different preprocessing techniques. In general the average error (absolute) holds about $2.5°$ to $3°$ for the small window and from $2°$ to $2.5°$ for the large; such error values are not optically detectable in the image. However, when it comes to the extreme values, we have a rather big improvement as they decrease from $23°$ for the image in figure 4.1a with no preprocessing and with the small window, down to $14°$ maximum when we use only zero padding and the 64×64 window; considerably more is the improvement for the negative differences, where from $-35°$ for the natural image as worst case we improve to $-10°$ for the random noise one. In general the extreme errors are radically reduced in the textured image because there exist no places with uniform surfaces and not enough information. In the calculation of the magnitude again we have some improvement, but not as much as in the orientation.

ERROR Estimations	*Gaussian Padded*		*Zero Padding*		*Gaussian Masking*		*No Preprocessing*		*Known angle*	
	64p	128p	64p	128p	64p	128p	64p	128p	64p	128p
Mean angle	2.4°	2.1°	1.8°	2.0°	2.0°	1.6°	1.6°	1.9°	—	—
Mean $\|\|angle\|\|$	3.0°	2.2°	2.8°	2.5°	3.0°	2.0°	2.5°	2.0°	—	—
S. Dev. angle	0.5°	0.4°	0.2°	0.2°	0.6°	0.5°	0.3°	0.2°	—	—
Max angle	10°	5°	14°	10°	11°	5°	12°	7°	—	—
Min angle	−8°	−2°	−9°	−5°	−10°	−3°	−7°	−2°	—	—
Mean length	-2.7	-1.5	-1.3	-0.2	-2.2	-3.1	-1.5	0.5	-3.2	-5.2
Mean $\|\|length\|\|$	4.1	5.7	4.7	6.6	5.0	6.0	5.6	6.3	4.6	6.2
S. Dev. length	0.8	1.1	1.0	1.7	1.2	1.4	1.0	1.4	1.1	2.1
Max length	16	14	17	17	17	12	13	17	15	15
Min length	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8

Table 4.3: Error Estimation for the blurred image of figure 4.1b, the vectors were estimated every 10 pixels. For a more detailed description of the table refer to the beginning of the section

This is most probably due to numerical errors like the ones that have been analysed in chapter three. In general the absolute error is smaller compared to the natural image in figure 4.1b. The results can be better if we use some methods to discard estimations that are not valid, which is here the major cause of error.

In order to analyse the error estimations of the algorithm without the complication of the numerical errors that occur during the simulation of the motion blur (as much as possible), and under good conditions, we create two more blurred images. This time the orientation of the blur is at a $-45°$ angle with the x-axis and it has a magnitude of 16 pixels; this way we don't use a simulation of antialiasing lines but the 16×16 identity matrix which is a matrix 16×16, with zero everywhere, except at the diagonal where it has the value $\frac{1}{16} = 0.0625$. In figure 4.16a, b, we present these two blurred images; they are produced by the same original images as the ones used for the images in figure 4.1a,b but this time we use a different blur kernel.



(a) (b)

Figure 4.16: Two artificially motion-blurred images (a) a natural image (b) a random noise image; with the motion blur diagonal and with a magnitude of 16 pixels.

For the first image 4.16a, the error estimations are presented in table 4.4. Doing

ERROR Estimations	*Gaussian Padded* 64p	*Gaussian Padded* 128p	*Zero Padding* 64p	*Zero Padding* 128p	*Gaussian Masking* 64p	*Gaussian Masking* 128p	*No Preprocessing* 64p	*No Preprocessing* 128p	*Known angle* 64p	*Known angle* 128p
Mean angle	-0.4°	-1.3°	-0.9°	-1.9°	-0.4°	-1.2°	-1.4°	-2.1°	—	—
Mean $\|\|angle\|\|$	2.5°	1.7°	3.7°	2.7°	2.9°	1.9°	4.8°	3.1°	—	—
S. Dev. angle	0.9°	0.6°	0.9°	0.5°	1.0°	0.6°	1.3°	0.5°	—	—
Max angle	10°	3°	11°	5°	10°	3°	17°	4°	—	—
Min angle	-15°	-5°	-13°	-9°	-16°	-6°	-19°	-10°	—	—
Mean length	-2.0	-0.5	-3.3	-2.6	2.6	-0.8	-0.9	-2.4	2.0	0
Mean $\|\|length\|\|$	3.2	0.9	4.8	3.2	8.8	1.3	8.9	3.7	2.0	0
S. Dev. length	1.1	1.4	0.9	0.7	0.6	1.0	0.8	1.3	2.2	0
Max length	13	13	14	6	14	11	14	11	11	0
Min length	-11	-7	-11	-11	-11	-8	-11	-11	0	0

Table 4.4: Error Estimation for the blurred image of figure 4.16a, the vectors were estimated every 10 pixels. For a more detailed description of the table refer to the beginning of the section

a crude comparison with the error estimations for the blurred image at figure 4.1a, some interesting results for the algorithm can be extracted. The estimation of the orientation does not present any major improvement mainly because the average error is rather small; but almost everywhere there exist a small improvement. There exists a noticeable improvement at the extreme error values especially for the use of the small window (64×64), where we have an improvement from $-34°$ down to $-15°$ for the use of the complete algorithm and from $-35°$ down to $-16°$ for Gaussian Masking only. It is clear that the Gaussian Masking is not as important as the zero padding for the calculation of the orientation for the small window. Overall, with no preprocessing at all, the estimation of the angle is worse than in any other case, but it is still at an acceptable error level. The magnitude estimation, which is much more sensitive to additive errors, presents a much more remarkable improvement - here the need for enough information and for the use of filtering in order to suppress the ringing effect are obvious. The average absolute error diminishes from 3.2 to 0.9 with the use of a 128×128 window instead of a 64×64. For the complete algorithm, if we use a small window and no zero padding, the results are completely wrong, with an average absolute error of 8.8 pixels. Extreme values are lower bounded by -11 when no negative peak is found, and it is remarkable that with the use of a large window and Gaussian Masking we don't reach this boundary and we get a minimum of -8. Finally when the orientation is known for a large window, we have an absolute success, which shows that the compression and the use of the cepstrum calculate the correct answer.

Comparing the error results for this image with the error estimation we got for the image 4.1a, help us to highlight some points. First, as noted earlier, the improvement is obvious from 6.4 pixels for a large window and the application of the complete algorithm we decreases to 0.9 pixels for the same setting, and also, even without that for the same setting. Also, even when we used zero padding only, the maximum absolute error we get is 4.8 for a 64×64 window. Second, the size of the window is important, because when there isn't enough information, even if we know exactly the direction on which we need to compress the Power Spectrum, it is rather difficult to get completely correct answers. Third, if the artifacts from the simulation are stronger than the blur ripple, even with given the orientation, the magnitude estimation is not error free. Finally, in order to improve the results, an iterative algorithm that estimates the value of the orientation by taking into account the value of the neighbours can be used, and if there exists a minimum error for the orientation an improvement to the magnitude estimation would be also achieved.

The last table, 4.5, contains the error estimations for the random noise image blurred across the diagonal, with a magnitude of 16 pixels as presented in figure 4.16b. The conditions in this case are rather good, as the image is full of texture and the artificial blur has minimum side effects. As can be seen from a simple comparison with the previous tables, we have the best results as far as it concerns the estimation of the orientation of the motion blur. The average absolute error in orientation is $2.3°$ at worst, which is as low as the best in every other case. Where the best result in average error is $0.9°$, which is unnoticeable. If we check the average error, which gives us an estimation on how well the resulting motion vectors approximate the general motion, we see an error of $0.5°$ which is simply unnoticeable. Moreover, when we check the extreme error values, they are also much lower than the previous ones; with an average $-6°$ (counterclockwise) and as small as $-2°$ for the complete algorithm and the use of a large window; for the positive values (clockwise) the smallest is $3°$ with at most a $10°$ for small window and no Gaussian masking to control the ringing effect. Overall, we could say that the full algorithm with a proper window size gives minimal errors when enough information is given. The improvement continues also in the magnitude estimation. With a 128×128 window size the average absolute error stays close to zero 0.1 - 0.2 pixels, while for a small window and zero padding (so enough information can be used) it is 2.3 pixels. The extreme values are accordingly small, -4, 4 pixels for the complete algorithm and a large window and similar values

ERROR Estimations	*Gaussian Padded*		*Zero Padding*		*Gaussian Masking*		*No Preprocessing*		*Known angle*	
	64p	128p	64p	128p	64p	128p	64p	128p	64p	128p
Mean angle	0.5°	0.5°	0.4°	0.5°	0.6°	0.5°	0.2°	0.4°	—	—
Mean $\|\|angle\|\|$	1.9°	0.9°	2.3°	1.3°	2.2°	1.1°	2.1°	1.0°	—	—
S. Dev. angle	0.5°	0.2°	0.2°	0.1°	0.5°	0.3°	0.4°	0.2°	—	—
Max angle	6°	3°	10°	6°	8°	4°	8°	5°	—	—
Min angle	-6°	-2°	-9°	-6°	-7°	-2°	-8°	-5°	—	—
Mean length	-1.9	0.0	-1.6	-0.7	4.1	-0.1	5.2	0.0	0	0
Mean $\|\|length\|\|$	2.3	0.1	2.3	0.8	8.9	0.2	9.1	0.2	0	0
S. Dev. length	0.6	0.5	0.5	1.6	0.7	0.7	0.7	0.4	0	0
Max length	9	4	14	9	14	2	14	6	0	0
Min length	-10	-4	-11	-10	-10	-5	-11	-3	0	0

Table 4.5: Error Estimation for the blurred image of figure 4.16b, the vectors were estimated every 10 pixels. For a more detailed description of the table refer to the beginning of the section

for the rest of the cases. It is worth noting that, given the correct orientation, even the 64×64 window gives completely correct results.

To sum up, the results from the error analysis demonstrate the major properties of the algorithm developed. When it is used with the appropriate data, the algorithm estimates the Optical Flow map quite accurately. To get meaningful results certain conditions have to be true: in the window that is used there must exist enough information in the blur in order to produce the characteristic ripple in the frequency domain, and the size of the window must be large enough so a few periods of the ripple appear and not just one, which would make the ripple undetectable.

Chapter 5

Conclusions

Whenever there is relative motion between a camera and objects in a visual scene, the camera's image of the scene is blurred. When the relative velocity is large enough, this *motion blur* can be quite significant.

Most visual motion estimation algorithms developed up to now treat motion blur as just one more source of noise. Most typically, these approaches either ignore motion blur, or assume a restricted situation in which camera and object velocities are relatively small.

In this thesis, a new approach to dealing with motion blur is formulated and evaluated experimentally. An algorithm is presented for computing the *optical flow* from a single motion-blurred image. The algorithm makes use of the information present in the structure imposed on the image by the motion blur.

The algorithm can be considered as operating in two steps. For each patch of the image, the direction of motion is first determined and then the speed in that direction is recovered. The algorithm operates in the frequency domain where it exploits the fact that motion blur introduces a characteristic *ripple* in the power spectrum. The orientation of these ripples in the 2D power spectrum is perpendicular to the direction of the motion blur. A key element of the algorithm developed in this thesis is the robust and efficient identification of the orientation of these ripples by making use of *steerable filters*. In experimental results, the orientation of motion blur is often recovered to within just a few degrees.

Once an accurate estimate of the oriention of the motion blur is known, the speed of motion, or the spatial extent of the blur, can be computed using a modified form of *cepstral analysis*. The first step in this procedure is to collapse the 2D log power spectrum into a 1D signal along the line indicating the direction of motion. The frequency of the ripple in the resulting 1D signal can be identified by taking a further Fourier Transform and locating a negative peak.

This algorithm has been implemented and evaluated experimentally using artificial and natural images. It has the advantage of exploiting information in a motion-blurred image that traditional motion analysis methods have tended to ignore. It has the added advantage of providing an optical flow map from a single image, instead of a sequence of images. The algorithm also lends itself easily to efficient parallel implementation.

There are some limitations for the applicability of this algorithm that are worth noting. Most importantly, the algorithm depends on the presence of texture in the image, since the blur in a region with homogenous brightness is undetectable. The magnitude of motion blur that can be detected is limited by the size of the image patch being analyzed. Also, if the motion blur is too small, on the order of just a few pixels, it becomes indistinguishable from other small-scale features, such as texture, noise, or out-of-focus blur.

In the following section, suggestions are made for application of this work, along with directions for future research in this area.

5.1 Future Goals

There are a number of directions that future developments could follow. Among the most obvious ones is the extension of the algorithm to deal with true colour pictures. Another is a parallel implementation of the algorithm which would improve its speed up to a point that it would be possible to run in almost real time for a whole image. That would make possible the use of the algorithm for extracting the Optical Flow from a moving camera on the fly, for navigational purposes.

More research on the windowing effect and the artifacts that it produces could lead into the application of knowledge intensive filters in order to reduce its side-effects. Also, the assumption that the image is noise free was made throughout this thesis, but future developments of the algorithm will have to take into account the noise factor and ensure the robustness of the algorithm in a noisy environment. One possible solution could be to prefilter the image in order to eliminate the noise. Another is to take into account the characteristic of the noise in the frequency domain and adapt the steerable filters and the cepstral analysis accordingly.

An open field of research is the adaptation of the algorithm according to the applications in which it could be used. Application specific issues should be addressed such as speed versus accuracy, acceptable error levels, and others. Incorporating the algorithm into a mobile robot architecture for self navigational purposes is one application; in that case it is important to update the optical flow field fast in order to get a general idea of the egomotion and also to identify moving objects that could present a threat for the robot, while the accuracy is not as important. Another application, with the opposite requirements, is the restoration of an image corrupted by motion blur. The image is taken with an inappropriate large exposure time, for example a security camera takes a shot of a speeding car, and the goal is to clean the picture. In that case there is no time constraint, but we need to find precisely the motion blur parameters in order to reconstruct the motion blur convolution matrix and perform a deconvolution to restore the image.

In conclusion, the algorithm developed could be useful in a wide range of applications, providing the Optical Flow map where traditional algorithms fail.

Bibliography

[1] J. K. Aggarwal and N. Nandhakumar. On the computation of motion from sequences of images-a review. *Proceedings of the IEEE*, 76(8):917–935, August 1988.

[2] Nicola Ancona and Tomaso Poggio. Optical flow from 1d correlation. pages 209–214. IEEE, 1993.

[3] Charles H. Anderson. Blur into focus. *Nature*, 343:419–420, February 1990.

[4] J. L. Barron, D. J. Fleet, and S. S. Beauchemin. Performance of optical flow techniques. *International Journal on Computer Vision*, 12(1):43–77, 1994.

[5] Michael J. Black and P. Anandan. A framework for the robust estimation of optical flow. pages 231–236. IEEE, 1993.

[6] C. Bonnet. Visual motion detection models:features and frequency. *Perception*, 6:491–500, 1977.

[7] Michael M. Chang, Murat A. Tekalp, and Tanju A. Erdem. Blur identification using the bispectrum. *IEEE Transactions on Signal Processing*, 39(10):2323–2325, Octomber 1991.

[8] R. Fabian and D. Malah. Robust identification of motion and out-of-focus blur parameters from blurred and noisy images. *CVGIP: Graphical Models and Image Processing*, 53(5):403–412, September 1991.

[9] D. J. Field. Relations between the statistics of natural images and the response properties oo cortical cells. *J. Optical Society of America*, A 4(12):2379–2394, 1987.

[10] William T. Freeman and Edward H. Adelson. The design and use of steerable filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(9):891–906, September 1991.

[11] R. C. Gonzalez and Wood. *Digital Image Processing*. Addison-Wesley, Reading, MA., 1992.

[12] Thomas L. Harrington and Marcia K. Harrington. Perception of motion using blur pattern information in the moderate and high-velocity domains of vision. *Acta Psychologica*, 48:227–237, 1981.

[13] Berthold K. P. Horn and Brian G. Schunck. Determining optical flow. Technical report, Massachusetts Institute of Technology, 1980.

[14] Berthold Klaus Paul Horn. *Robot Vision.* MIT Press, McGraw-Hill, 1986.

[15] David G. Lamb. Passive monocular range imaging with a multiple aperture camera. Master's thesis, Department of Electrical Engineering, McGill University, Montreal, August 1994.

[16] Jae S. Lim. *Two-Dimensional Signal and Image Processing.* Prentice-Hall, Englewood Cliffs, NJ, 1990.

[17] D. Marr. *Vision.* Freeman, New York, 1982.

[18] Wayne Niblack. *An Introduction to Digital Image Processing.* Prentice/Hall International, Englewood Cliffs, NJ, 1990.

[19] William K. Pratt. *Digital Image Processing.* John Wiley & Sons, Inc., 1978.

[20] John G. Proakis and Dimitris G. Manolakis. *Digital Signal Processing.* Macmillan Publishing Company, 866 Third Avenue, New York, New York 10022, second edition, 1992.

[21] Shimon Ullman. The interpetation of visual motion. Technical report, Massachusetts Institute of Technology, 1979.

[22] Shimon Ullman. Analysis of visual motion by biological and computer systems. In Firschein Oscar Fischler A. Martin, editor, *Readings in Computer Vision*, chapter Recovering Scene Geometry, pages 132–144. Morgan Kaufmann Publishers, Inc., 95 First Street, Los Altos, California 94022, 1987.

[23] J. F. Vega-Riveros and K. Jabbour. Review of motion analysis techniques. *IEE Proceedings*, 136(6):397–404, December 1989.

[24] Joseph Weber and Jitendra Malik. Robust computation of optical flow in a multiscale differential framework. In *International Conference in Computer Vision*, pages 12–20, 1993.

[25] Yasuo Yoshida, Kazuyochi Horiike, and Kazuhiro Fujita. Parameter estimation of uniform image blur using dct. *IEICE Trans. Fundamentals*, E76(7):1154–1157, July 1993.

List of Abbreviations

DCT: Discrete Cosine Transform

DFT: Discrete Fourier Transform

FT: Fourier Transform

FFT: Fast Fourier Transform

IFT: Inverse Fourier Transform

IFFT: Inverse Fast Fourier Transform

PSF: Point Spred Function

RMS: Root Mean Square

SNR: Signal to Noise Ratio