6-2006

Fast Video Stabilization Algorithms

Mohammed A. Alharbi

AFIT/GCS/ENG/06-02

Fast Video Stabilization Algorithms

THESIS

Mohammed A. Alharbi, Captain, RSAF

AFIT/GCS/ENG/06-02

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/06-02

AFIT/GCS/ENG/06-02

Fast Video Stabilization Algorithms

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

Mohammed A. Alharbi, B.S.Cp.E.

Captain, RSAF

June 2006

AFIT/GCS/ENG/06-02

Fast Video Stabilization Algorithms

Mohammed A. Alharbi, B.S.Cp.E.
Captain, RSAF

Approved:

/signed/	31 May 2006
Dr. Guna S. Seetharaman, Ph.D. (Chairman)	date
/signed/	31 May 2006
Lt. Col. Matthew E. Goda, Ph.D. (Member)	date
/signed/	31 May 2006
Maj. Scott R. Graham, Ph.D. (Member)	date

Abstract

A set of fast and robust electronic video stabilization algorithms are presented in this thesis. The first algorithm is based on a two-dimensional feature-based motion estimation technique. The method tracks a small set of features and estimates the movement of the camera between consecutive frames. An affine motion model is utilized to determine the parameters of translation and rotation between images. The determined affine transformation is then exploited to compensate for the abrupt temporal discontinuities of input image sequences. Also, a Frequency domain approach is developed to estimate translations between two consecutive frames in a video sequence. Finally, a jitter detection technique has been developed to isolate vibration affected subsequences from an image sequence. The experimental results of using both simulated and real images have revealed the applicability of the proposed techniques. In particular, the emphasis has been to develop real time implementable algorithms, suitable for unmanned vehicles with severe payload constraints.

Acknowledgement

I offer my sincere thanks and appreciation to my advisor, Dr. Guna S. Seetharaman, whose guidance and encouragement was extremely helpful during this effort. I would also like to extend appreciation to my committee members, Lt. Col. Matthew E. Goda and Maj. Scott R. Graham, for their insight and useful advice.

Mohammed A. Alharbi

Table of Contents

Table of Figures

List of Tables

Fast Video Stabilization Algorithms

I. Introduction

1.1 Problem Statement

Assume a camera rigidly mounted on a vehicle in motion. If the motion of the vehicle is smooth, so will be the corresponding image sequence taken from the camera. In the case of small unmanned aerial imaging system, and off road navigating ground vehicles, the onboard cameras experience sever jitter and vibration. Consequently, the video images acquired from these platforms have to be preprocessed to eliminate the jitter induced variations before human analysis. The task at hand is to detect the jitter and eliminate its effect. It is composed of two subtasks: First, to develop a reliable method to detect in real-time the subsequence affected by jitters. Second, to develop a strategy to interpolate the images, without sacrificing detail (dismount targets).

1.2 Research Goal

Motion in video images is caused by either the object motion or the camera movement. Digital (electronic) image stabilization (DIS/EIS) system endeavor to produce a compensated video sequence so that image motion due to the camera's undesirable vibration or juggles can be removed [1]. The goal of this research is to introduce a new approach to stabilize image sequence. The newly developed algorithm provides a fast and robust stabilization system, and alters real-time performance.

1.3 Applications of DIS

Modern (contemporary) light weight digital camera, camcorders, CCD sensing arrays, and next-generation mobile phone with visual display, etc., are principal candidates in need of automatic image stabilization. They are prone to inevitable and undesirable camera motion during the image capturing process. It would be worthwhile to have a digital image sequence stabilization scheme that can further stabilize the image sequence for improving the subjective quality of the video sequence obtained. Moreover, an image stabilization algorithm is reported to be beneficial to the coding efficiency of video signals [6]. It also has been used for the computation of egomotion [17, 18], detection and tracking of Independently Moving Objects (IMOs) [20, 21, 22], and video compression [19].

The developed algorithm is being implemented for Unmanned Arial Vehicle (UAV) surveillance applications.

1.4 Organization of the Study

This thesis is organized into five chapters. The first chapter presents the introduction, problem statement, the goal of the research, and finally it summarizes the three types of image stabilization methods. Chapter two reviews some related previous works. Chapter three presents fundamental concepts in the field of image processing which are necessary to understand the methodology used to solve the problem being studied. Chapter four explains the methodologies and the techniques used to implement the various algorithms. Chapter five documents the data resulted from the algorithms test. Chapter six summarizes the research, including limitations and areas of future work.

1.5 Image Stabilization Methods

There are three types of image stabilizers currently available [23]: Digital Image Stabilization (DIS), Optical Image Stabilization (OIS), and Mechanical Image Stabilization (MIS).

1.5.1 Digital Image Stabilization

Digital Image Stabilization (DIS) systems use electronic processing to control image stability. The DIS system starts working once the image hits the light-sensing chip, the Charge Coupled Device (CCD). If, through its sensors, the system detects what it thinks is camera vibration, it responds by slightly moving the image so that it remains in the same place on the CCD. For example, if the camera vibrations to the right, the image moves to the left to compensate, thus eliminating the vibration [23].

There are two ways DIS works to reduce the perceived movement of the image. One method increases the size of the image by digitally "Zooming" in on the image so that it is larger than the CCD. By making the image larger, the system can "pan and scan" within the image to counter the movement created by the vibration. Because this system must digitally zoom in on the image to slightly increase its size, it decreases the picture resolution somewhat. The other method of electronic stabilization uses an oversized CCD. The video image covers only about 90 percent of the chip's area, giving the system space in which to move the image. When the image is stable, the chip centers the image on the CCD. If the camera vibrates to the right, the image has the space to roam to the left to compensate for the vibration, keeping the subject of the image in exactly the same place on the CCD, thus eliminating the vibration.

Detecting the vibration is key to the effectiveness of the system. DIS systems use one of two ways to detect shaky video. Either they detect movement within the image as recorded on the CCD or they detect the actual movement of the camera. The first method of detection analyzes the changes between the fields in each image. A specially designed feature of the camera stores the odd and even fields of the video frame and look for changes between them. If parts of the image change in one field but not the other, it indicates that the subject in the field of view is moving but not the background. If however, the entire image changes from one field to the next, it most likely means there is camera vibration and the camera must correct the image. To correct the camera vibration, the camera's electronics detect the direction of the movement and shifts the active field so that it meshes with the memorized field. A major disadvantage of this system is that if there is a large object moving in the frame, it may be interpreted as camera vibration and the camera will attempt to stabilize the subject causing a blurring of the image and reduction in picture resolution. The camera can also use motion sensors to detect camera vibration. Because this method senses movement in the camera not the image, the movement of a subject in the image cannot fool it. However, it will sometimes react at the beginning of an intentional camera movement (such as a pan) and will take a short moment to realize that you are moving the camera on purpose. Instead of a smooth pan, the image will freeze and then leap into the pan suddenly [23].

1.5.2 Optical Image Stabilization

The Optical Image Stabilization (OIS) system, unlike the DIS system, manipulates the image before it gets to the CCD. When the lens moves, the light rays from the subject are bent relative to the optical axis, resulting in an unsteady image because the light rays are deflected. By shifting the IS lens group on a plane perpendicular to the optical axis to counter the degree of image vibration, the light rays reaching the image plane can be steadied [15].

Since image vibration occurs in both horizontal and vertical directions, two vibration-detecting sensors for yaw and pitch are used to detect the angle and speed of movement. Then the actuator moves the IS lens group horizontally and vertically thus counteracting the image vibration and maintaining the stable picture. The Shift-IS component is located within the lens groups and is most effective for lower frequency movements caused by platform vibration or wind effect without increasing the overall size and weight of the master lens. Figure 1-1 shows an illustration of this type of image stabilization.

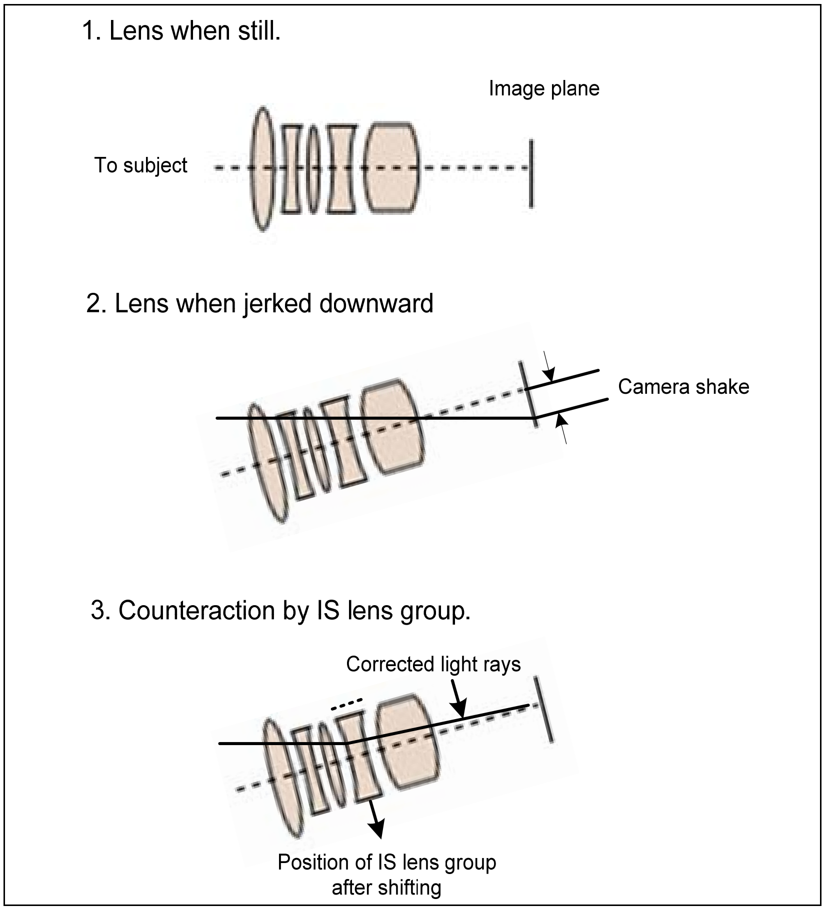


Figure 1-1: Optical Image Stabilization [15]

1.5.3 Mechanical Image Stabilization

Mechanical image stabilization involves stabilizing the entire camera, not just the image. This type of stabilization uses a device called “Gyros”. Gyros consist of a gyroscope with two perpendicular spinning wheels and a battery pack. Gyroscopes are motion sensors. When the gyroscopes sense movement, a signal is sent to the motors to move the wheels to maintain stability. The gyro attaches to the camera's tripod socket and acts like an "invisible tripod" [13].

Figure 1-2: Gyroscopic Stabilizer [13]

Figure 1-2 shows a picture of a gyroscopic stabilizer. The vibration gyro was improved by employing a tuning fork structure and a vibration amplitude feedback control [33]. They are heavy, consume more power, and are not suitable for energy sensitive and payload constrained imaging applications.

II. Literature Review

Many methods for video stabilization have been reported over the past few years. Most proposed methods compensate for all motion [2, 18, 20, 24, 25, 26], producing a sequence where the background remains motionless. Other techniques only subtract the 3D rotation of the camera [27, 28, 29] generating a de-rotated sequence. However, these methods can be distinguished by the models adopted to estimate the camera motion [9]. Several two-dimensional and three-dimensional stabilization schemes are described in [24]. For 2D models, in general all the estimated affine motion parameters are compensated for, i.e., gross motion is removed from the input sequence [20, 25, and 21]. Stabilization in 3D is achieved by re-rotating the frames, generating a translation-only sequence, or a sequence containing translation and low-frequency rotation. Yao *et al.* [29] Compensate for 3D rotation by tracking multiple visual cues, like distant points and horizon lines, using an extended Kalman filter for the estimation of the 3D motion parameters of interest. Both kinematics and kinetic models suitable for determining the smooth and oscillatory rotational motion components are considered, so that smoothed rotation can be also obtained. A vehicle model is also used in [27] to filter the high-frequency components of the rotational parameters. A flow-based motion estimator applied to points on the horizon (distant points) is used to estimate the rotational parameters, and the solution is recursively refined to obtain smoothed motion. Two-dimensional models are used by [17, 18, 19, and 22]. Another method in [19] seeks to use linear segments from the input images and align them with the absolute vertical direction, which can be provided by an inertial sensor, eliminating the need to estimate the rotation around the optical axis. Stabilization is achieved by compensating for 2D linear translation, which minimizes the disparity between two successive frames.

Fast implementations of 2D stabilization algorithms are presented in [25, 20, and 21]. Hansen *et al.* [25] describe the implementation of an image stabilization system based on a mosaic-based registration technique. Burt *et al*. [20] describe a system which uses a multi-resolution, iterative process that estimates affine motion parameters between levels of Laplacian pyramid images. From coarse to fine levels, the optical flow of local patches of the image is computed using a cross-correlation scheme. The motion parameters are then computed by fitting an affine motion model to the flow [9].

Some studies follow frequency domain algorithms to estimate motion between two images [30, 31, and 32]. The Fourier transform properties of relocated images are used to estimate rotation and translation. Frequency domain methods for estimating shifts in the image plane are based on the fact that a shift in spatial domain can be expressed as a phase shift in frequency domain. Two shifted images differ only by a linear phase difference [30, 31]. These methods can be extended to include (planar) rotation and scale using polar coordinates [32] with the advantage that shift, rotation and scale can be estimated separately. The main limitation of frequency domain methods is that they are restricted to global shifts and rotations in the image plane, and scale [11]. If the scene is composed of multiple, independently moving objects, then, the method will not provide adequate performance.

A fast and robust implementation of a digital image stabilization algorithm presented in this thesis is based on the 2D model described in [1].

The developed algorithm is similar to the other algorithms based on the 2D rigid motion model [29]. But instead of using extensive feature-tracking, our parametric motion model is obtained by tracking only a small set of features to characterize the underlying motion vectors and produce equally good performance.

The algorithm is applied to translational and rotational camera motion separately.

III. Background

This chapter presents basic ideas behind image stabilization, and introduce various analytical tools used in literature for building a simple vibration compensation systems. In particular, we investigated the problem using three approaches: (1) levelsets based shape analysis, (2) feature points based jitter detection and, (3) Fourier transform based approach.

3.1 Image Sampling

Before an image can be manipulated using various image processing techniques, it must be spatially sampled. The process of sampling an image is the process of applying a two-dimensional grid to a spatially continuous image to discretize it into a two-dimensional array of elements.

Figure 3-1 shows a sampled image containing a total of *NM* sampled elements using a rectangular grid. Any type of sampling grid can be used, but the rectangular grid is by far the most common because of its relationship to two-dimensional arrays. The fundamental unit of a sampled image is a picture element and is typically referred to as a pixel. The value of each pixel is equal to the average intensity of the continuous spatial image covered by that pixel.

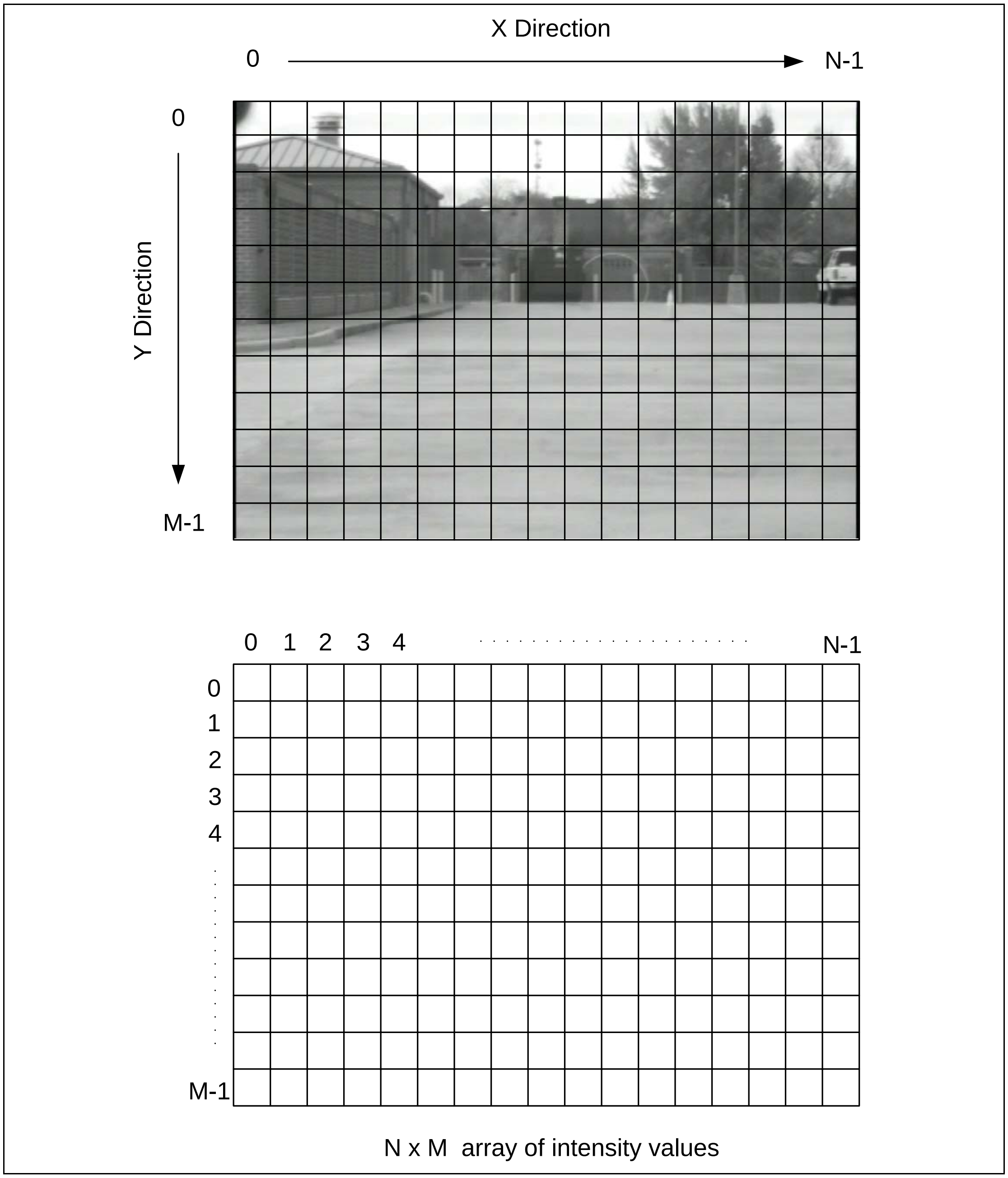


Figure 3-1: Spatially sampled image containing *N* x *M* picture elements

The result of sampling produces a two-dimensional array of numbers that are directly proportional to the intensity levels of the continuous spatial image. Real-time video data is usually digitized over a 320x240, 640x480, 768x525, or 1600x1200 grid according to the context. Many of these size-resolution combinations were chosen to be compatible with the spatial size of NTSC video and to meet the storage size requirements of digital memory. Image size that are powers of two exist because of the requirements for computing the Fast Fourier Transform (FFT), to be considered later.

3.2 Quantization

Besides spatial sampling, the intensity level at each pixel must also be digitized into a finite set of numbers. The process of digitization converts an analog intensity value into a set of digital numbers that represent the intensity levels in the image. The quantity of numbers used to represent the intensities in a continuous tone image determines the final quality of the digitization process. This set of numbers is referred as the gray levels or grayscales of an image.

Since an image is the spatial distribution of light energy, the numbers assigned to gray levels of a digitized image can take only positive values. Figure 3-2 (a) gives a 4x4 sub-image taken from an image. Figure 3-2 (b) gives the corresponding grayscale, with the value of 0 assigned to black and each grayscale value increasing in intensity until the value of 255 is reached, corresponding to white.

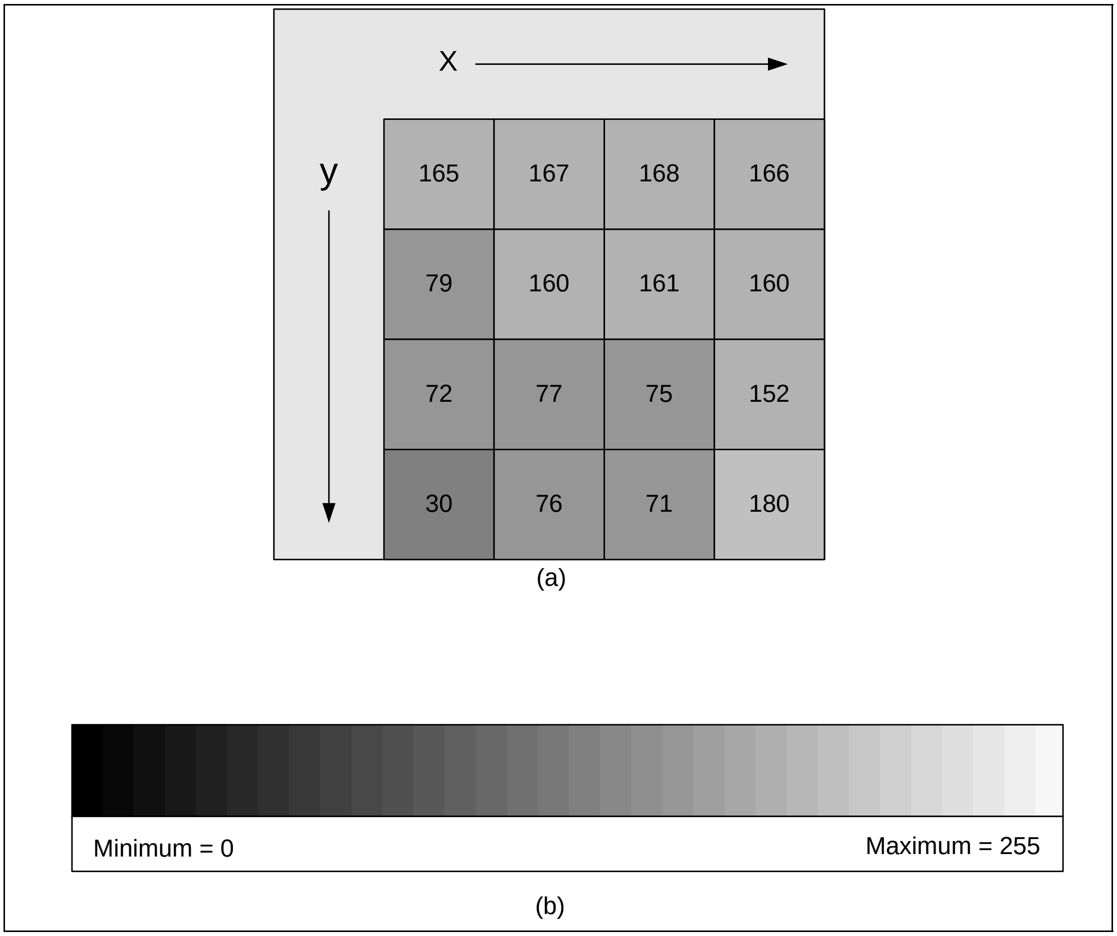


Figure 3-2: An example of (a) a sampled and digitized 4x4 sub-image and (b) its corresponding grayscale

3.3 Converting Gray-scale Images to Binary Image Using Thresholding

Thresholding is an image processing technique for converting a grayscale or color image to a binary image based upon a thresholding value. If a pixel in the image has an intensity value less than the threshold value k (i.e., $f(x,y)<k$), the corresponding pixel in the in the resulting image is set to 0 (black). Otherwise, if the pixel intensity value is greater or equal to the

threshold intensity k (i.e., $f(x,y) \geq k$), the resulting pixel is set to 255 (white). Thus, it is used to create a binary image, or an image with only 2 colors, black (0) and white (255). This can be formulated as follows:

$$f(x,y) = \begin{cases} 0 & f(x,y) < k \\ 255 & f(x,y) \geq k \end{cases} \tag{3-1}$$

The last equation can be generalized as follows:

$$f(x,y) = \begin{cases} G_a & f(x,y) < k \\ G_b & f(x,y) \geq k \end{cases} \tag{3-2}$$

where, G_a and G_b are the desired two gray levels in the threshold image.

The process of thresholding as described by equation 8 reduces a multilevel image to a two gray-level image containing gray levels G_a and G_b. Equation (3-2) can be expanded to include more than one threshold value as follows:

$$f(x,y) = \begin{cases} G_a & 0 \leq f(x,y) < k_1 \\ G_b & k_1 \leq f(x,y) < k_2 \\ G_c & k_2 \leq f(x,y) < G_{\max} \end{cases} \tag{3-3}$$

where, G_{max} is the maximum allowable gray level of the image f(x,y) (255 in case of 8-bit gray-scaled image). And k_1, and k_2 are threshold values.

3.4 Histogram

The brightness characteristic of an image can be concisely described with a tool known as the brightness histogram. The brightness histogram describes the frequency distribution of the gray levels of pixels within a digital image. It provides a graphical representation of how many pixels within an image fall into a given image.

A histogram appears as a graph with "brightness" on the horizontal axis from 0 to 255 (for an 8-bit gray scale) and "number of pixels" on the vertical axis. To find the number of pixels having a particular brightness within an image, we simply look up the brightness on the horizontal axis, follow the bar graph up, and read off the number of pixels on the vertical axis. Because all pixels must have some brightness value defining them, the number of pixels in each brightness column adds up to the total number of pixels in the image.

Let's assume that an image has been digitized and sampled into N pixels, each of which has been quantized into n levels in the range $d_0, d_1, \ldots, d_{n-1}$. Figure 3-3 shows the histogram of this image.

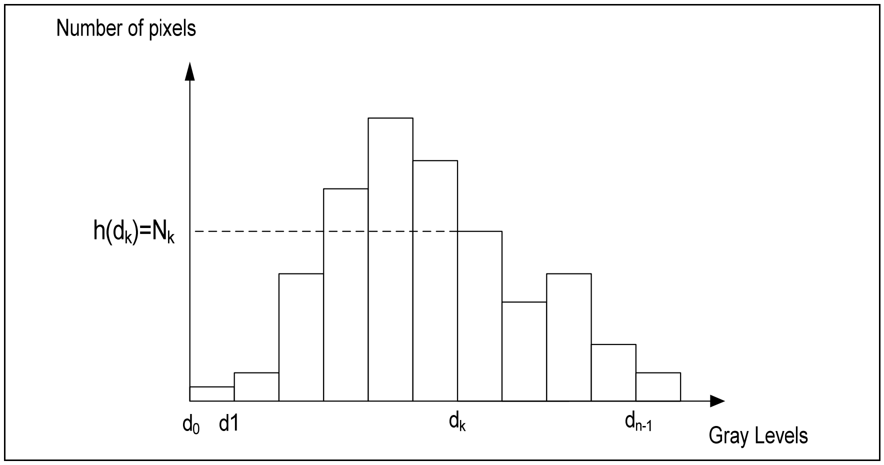


Figure 3-3: Image histogram

The function $h(d_k)$= The number of pixels with a gray level equals d_k and is written as :

$$h(d_k)=N_k \tag{3-4}$$

where, d_k is the gray level and N_k is the number of pixels with a gray level = d_k.

3.5 Cumulative Histogram

The cumulative histogram is another variation of the histogram in which the vertical axis gives not just the number of the pixels at that gray level, but rather gives the number of the pixels at that level plus the number of pixels with smaller values of gray level.

Using the same assumptions as in the last section, the cumulative histogram of the image is shown in Figure 3-4.

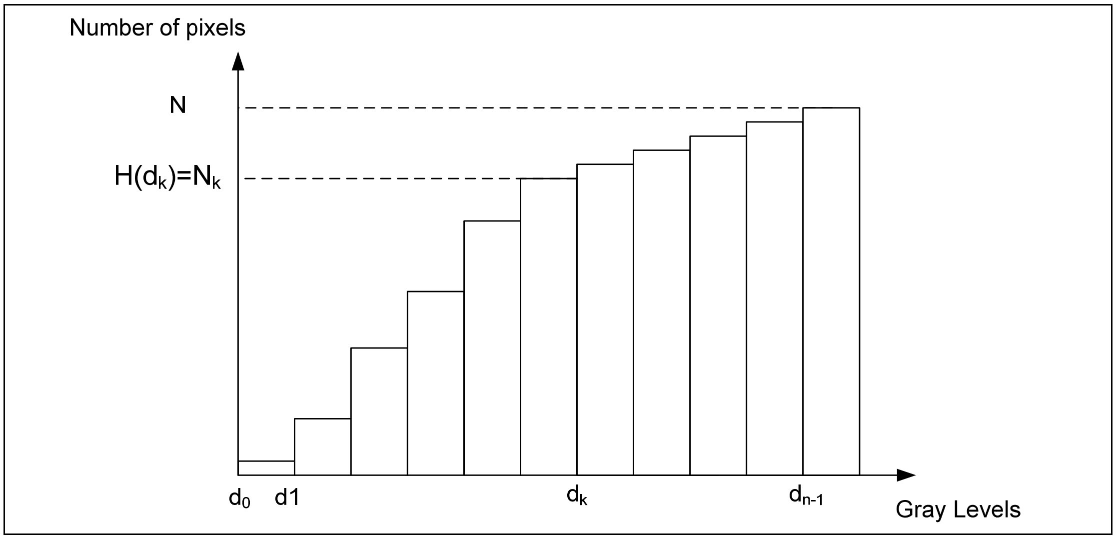


Figure 3-4: Cumulative histogram

The function $H(d_k)$ = The number of pixels with a gray level equal to or less than d_k. Hence,

$$H(d_k) = \sum_{i=0}^{k} h(d_i) = \sum_{i=0}^{k} N_i \qquad \textbf{(3-5)}$$

$$H(d_k) = \sum_{i=0}^{k} N_i \qquad \textbf{(3-6)}$$

Both histogram and cumulative histogram are step functions.

The cumulative histogram $H(d_k)$ increases from 0 to N, being the number of pixels in the image, since $\sum_{i=0}^{n-1} N_i = N$.

3.6 Invariant Moments

In general, the moments of a function are commonly used in probability theory. However, several desirable properties that can be derived from moments are also applicable to image analysis.

Definition: The set of moments of a bounded function *f(x,y)* of two variables is defined by:

$$M_{jk} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x^j y^k f(x,y)\, dxdy \qquad \textbf{(3-7)}$$

where, *j* and *k* take on all nonnegative integer values.

As *j* and *k* take on all nonnegative integer values, they generate an infinite set of moments. Furthermore, this set is sufficient to specify the function *f(x,y)* completely. In other words, the set $\{M_{jk}\}$ is unique for the function *f(x,y),* and only *f(x,y)* has that particular set of moments.

The parameter *j+k* is called the order of the moment. There is only one zero-order moment,

$$M_{00} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\, dxdy \qquad \textbf{(3-8)}$$

There are two first-order moments and correspondingly more moments of higher orders.

The coordinates of the center of gravity of an object are:

$$\bar{x} = \frac{M_{10}}{M_{00}}, \tag{3-9}$$

$$\bar{y} = \frac{M_{01}}{M_{00}} \tag{3-10}$$

where,

$$M_{00} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x, y)\, dxdy \tag{3-11}$$

$$M_{10} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x\, f(x, y)\, dxdy \tag{3-12}$$

$$M_{01} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} y\, f(x, y)\, dxdy \tag{3-13}$$

3.7 Spatial Moments of Binary Images and Level Sets

The spatial moments of an object in an image are statistical shape measures that give statistical measures related to an object's characteristics.

The zero-order spatial moment is computed as the sum of the pixel brightness values in an image. In the case of binary image, this is simply the number of pixels in the object, because every object pixel is equal to 1 (white). Therefore, the zero order spatial moment of a binary object is its area. For a gray-scaled image, an object's zero-order spatial moment is the sum of its pixel brightness.

The first order spatial moments of an object contain two independent components *x* and *y*. They are the *x* and *y* sums of the pixels brightness in the object, each multiplied by its respective *x* or *y* coordinate location in the image.

In the case of a binary image, the first-order *x* spatial moment is just the sum of the *x* coordinates of the object's pixels, because every object pixel is equal to 1 (white). Likewise, the y spatial moment is the sum of the y coordinate of the object's pixels. For a gray-scaled image, an object's first order spatial moments are as defined above. The first-order spatial moments of an object represent the object's mass and how it is spatially distributed.

The two most common image object measurements that use spatial moments are object area and center of mass (*a.k.a* centroid). As stated above, an object's area is computed as it's zero-order spatial moment. An object's center of mass can be computed as the first-order spatial moments (*x* and *y*) divided by the zero order moment, or the object area.

There are two forms of the center of mass, one that considers pixels to have uniform weight, as in a binary image, and one that weights pixels based on their brightness values. The second form considers pixels that are black to have a weight = 0, those that are white to have a weight = 255, and pixels with brightness in between to have a weight corresponding to their respective gray-levels.

The definitions for the center of mass measures are as follows:

Brightness-Weighted Center of Mass:

The balance point *(x,y)* of the object where there is equal brightness above, below, left, and right. If we think of the pixels in an object as having a weighted dependent upon their brightness, then the brightness weighted center of mass is the point where the object will perfectly balance on the tip of a point, as shown in Figure 3-5.

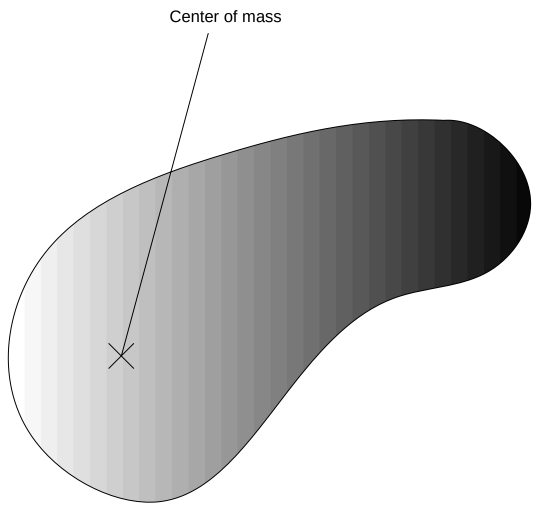


Figure 3-5: **Center of mass of a gray-scale image**

$$\text{Center of Mass}_x = \frac{\text{Sum of objects x-pixel coordinates} \times \text{pixel brightness}}{\text{Number of pixels in object}}$$

$$\text{Center of Mass}_y = \frac{\text{Sum of objects y-pixel coordinates} \times \text{pixel brightness}}{\text{Number of pixels in object}}$$

For a binary image, the pixel brightness will be equal to 1. So, for a binary image:

$$\text{Center of Mass}_x = \frac{\text{Sum of objects x-pixel coordinates}}{\text{Number of pixels in object}}$$

$$\text{Center of Mass}_y = \frac{\text{Sum of objects y-pixel coordinates}}{\text{Number of pixels in object}}$$

Figure 3-6 shows the center of mass for a binary object.

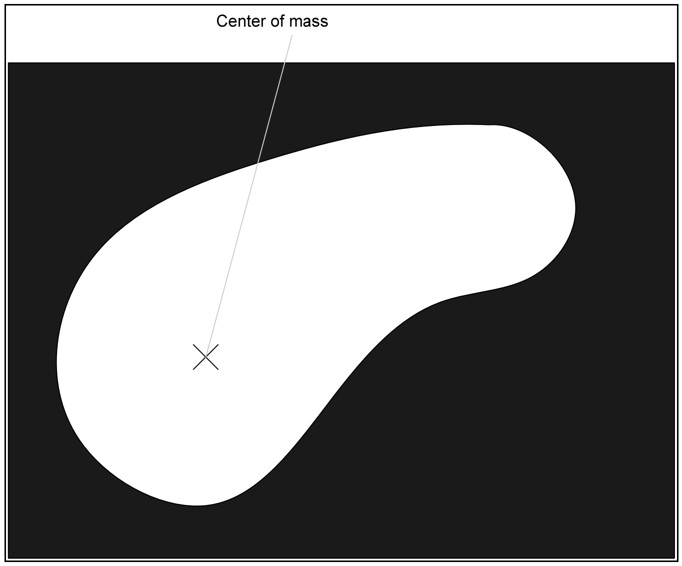


Figure 3-6: Center of mass of a binary image

For an *N*x*M* gray-scaled image, equation (3-7) can be changed to discrete version as follows:

$$M_{jk} = \sum_{y=0}^{M-1}\sum_{x=0}^{N-1} x^j y^k f(x,y) \qquad \textbf{(3-14)}$$

And for an NxM binary image, equation (3-7) reduces to:

$$M_{jk} = \sum_{y=0}^{M-1}\sum_{x=0}^{N-1} x^j y^k \delta(f(x,y)-1) \qquad \textbf{(3-15)}$$

Equation (3-8) can also be changed to the following:

$$M_{00} = \sum_{y=0}^{M-1}\sum_{x=0}^{N-1} f(x,y) \qquad \textbf{(3-16)}$$

Likewise, equation (3-9) can be changed as follows:

$$M_{10} = \sum_{y=0}^{M-1}\sum_{x=0}^{N-1} xf(x,y) \tag{3-17}$$

And, equation (3-10) can be changed as follows:

$$M_{01} = \sum_{y=0}^{M-1}\sum_{x=0}^{N-1} yf(x,y) \tag{3-18}$$

Finally, the center of gravity of an image will be:

$$\bar{x} = \frac{M_{10}}{M_{00}} = \frac{\sum_{y=0}^{M-1}\sum_{x=0}^{N-1} xf(x,y)}{\sum_{y=0}^{M-1}\sum_{x=0}^{N-1} f(x,y)} \tag{3-19}$$

$$\bar{y} = \frac{M_{01}}{M_{00}} = \frac{\sum_{y=0}^{M-1}\sum_{x=0}^{N-1} yf(x,y)}{\sum_{y=0}^{M-1}\sum_{x=0}^{N-1} f(x,y)} \tag{3-20}$$

We can also compute higher-order spatial moments. For instance, the second-order moments produce object orientation information. Spatial moments of an order that is greater than two produce abstract information that is difficult to tie specifically to physical object characteristics.

3.8 Motion Analysis

3.8.1 Image Translation

The basic model of disparity between two images is translation. Translation is used to move regions of an image intact to other locations within the image. Typically, it indicates that an object in the foreground has moved. If the translation operations moves a region outside the area defined by the original image, then a new image must be created that encompasses the original image plus the translated region. Image translation is defined as follows:

$$\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} x_{old} \\ y_{old} \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \qquad \textbf{(3-21)}$$

where, x_{old}, y_{old} are the pixel coordinates of an arbitrary point in the region to be translated; and, x_{new}, y_{new} are the coordinates of its location after of the translation is complete. The values Δx, and Δy define the amount of translation in the x and y directions, respectively. For each pixel within a region to be translated, Equation (3-21) is applied to produce a new set of translated coordinates. In translating a region, the original image is first copied to the output image and then the region to be translated is moved to its new position within the image using Equation (3-21). If the pixels within the original region to be translated are left unchanged, the translation process becomes equivalent to an image copy. If, on the other hand, the original region to be translated is filled with a constant gray level (erased), the translation operation becomes equivalent to a move operation. Figure 3-7 shows an example of a translation.

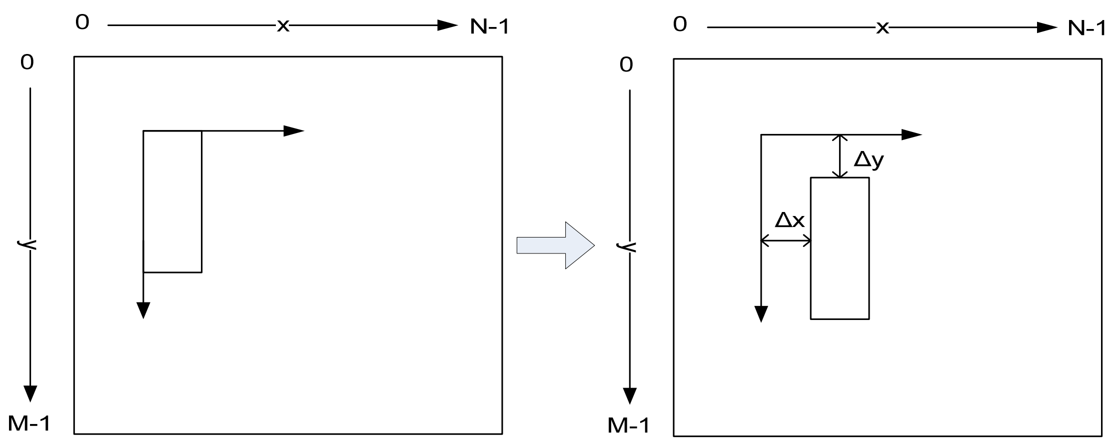


Figure 3-7: Translation example

Translation by integer pixel values is straight forward. However, translation by subpixels must be realized using bilinear interpolation.

3.8.2 Image Rotation

Rotation is one of the fundamental models of linear spatial transformations between two images. It is characterized by two parameters: center of rotation, and the rotation angle.

Consider a counter-clockwise rotation of the camera. The net effect is a clockwise rotation of all pixels to a new location.

$$\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \end{bmatrix} \tag{3-22}$$

where, θ is the angle of rotation.

Further analysis will indicate that:

$$\begin{bmatrix} \bar{x}_{new} \\ \bar{y}_{new} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \bar{x}_{old} \\ \bar{y}_{old} \end{bmatrix} \tag{3-23}$$

where the quantity $\bar{x}$ indicates an average value.

Then,

$$\begin{bmatrix} x_{new} - \bar{x}_{new} \\ y_{new} - \bar{y}_{new} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_{old} - \bar{x}_{old} \\ y_{old} - \bar{y}_{old} \end{bmatrix} \tag{3-24}$$

It is often convenient and more desirable to analyze and characterize the motion of individual objects in the scene, including their observed rotation(s). The expression (3-24) above facilitates such a mechanism.

From (3-22) and (3-2) we conclude that:

$$\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_{old} - \overline{x}_{old} \\ y_{old} - \overline{y}_{old} \end{bmatrix} + \begin{bmatrix} \overline{x}_{new} - \overline{x}_{old} \\ \overline{y}_{new} - \overline{y}_{old} \end{bmatrix} \tag{3-25}$$

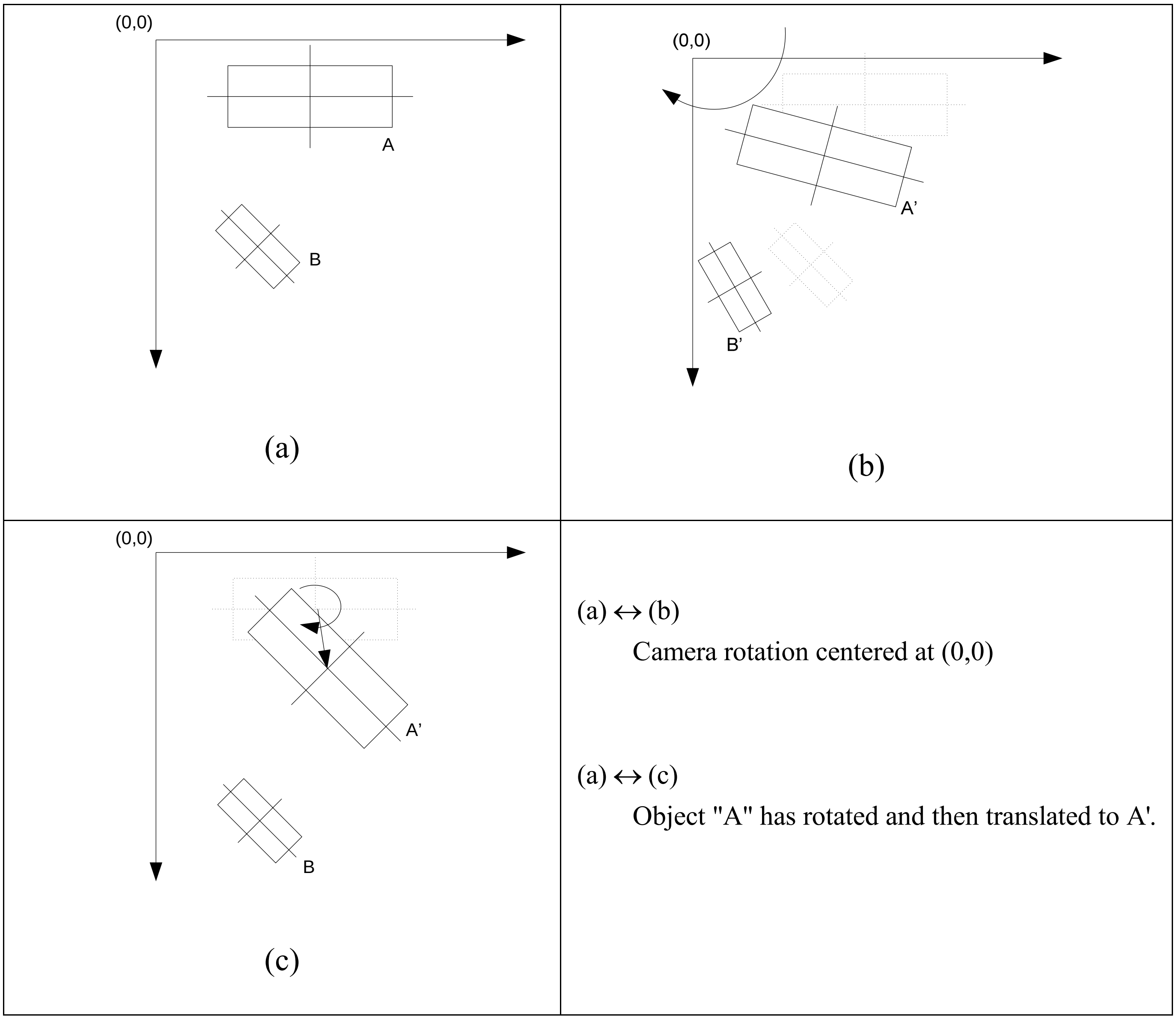


Figure 3-8 Rotation Example

Composite motion comprised of both geometrical translation and rotation of a region within an image about its geometrical center $M_{x,}$ M_y, is expressed as:

$$\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \left\{ \begin{bmatrix} x_{old} \\ y_{old} \end{bmatrix} - \begin{bmatrix} M_x \\ M_y \end{bmatrix}_{old} \right\} + \begin{bmatrix} M_x \\ M_y \end{bmatrix}_{new} \quad \textbf{(3-26)}$$

The geometrical center (centroid) of the region as we have seen before is given by:

$$M_x = \frac{1}{N}\sum_{i=1}^{N} x_i \quad \textbf{(3-27)}$$

and

$$M_y = \frac{1}{N}\sum_{i=1}^{N} y_i \quad \textbf{(3-28)}$$

where, x_i and y_i are the coordinates for each pixel in the region to be translated and the parameter N is defined as the number of pixels within the region being translated.

Equation (3-26) can also be used to rotate an entire image about the particular point x_o, y_o by setting $M_x = x_o$, $M_y = y_o$. Once the rotation is completed, the image is then translated back to its original position x_o, y_o.

3.8.3 Image Scaling

Another common type of geometrical operation is that of scaling. Scaling provides a means of reducing or enlarging the size of an image. Desired regions within an image can magnified to spatially enlarge features that would otherwise be difficult to observe. Geometrical image scaling is defined mathematically in equation (3-29)

$$\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \end{bmatrix} \qquad \textbf{(3-29)}$$

To scale a total image, x_{old}, y_{old} are defined over the coordinates of the entire image, and for region scaling x_{old}, y_{old} are defined by the pixels within the region to be scaled. For σ_x and $\sigma_y > 1$, the output image will be an enlarged version of the input image, while for σ_x and $\sigma_y < 1$ the scaled output image is a reduced version of the input image. For either σ_x or σ_y negative, the image is rotated about the axis of the negative scaling parameter. For example if σ_x=*-3,* σ_y=*1*, the image is increased by three and is flipped about the x axis. Geometric scaling in particular requires the use of interpolation prior to scaling an image. Interpolation will be discussed later in this chapter.

3.8.4 Image Skewing

The next basic model of shape change or disparity is skewing (deformation) or shear change. Figure 3-9 shows an image of a rectangle that has been skewed to the right in the x direction by an angle of α. Figure 3-10 shows the same image skewed to the lower direction of the y-axis by an angle of α.

The skewing geometrical transformation is defined by

$$\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} \cos\alpha & \sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \end{bmatrix} \tag{3-30}$$

where, α is the deformation angle.

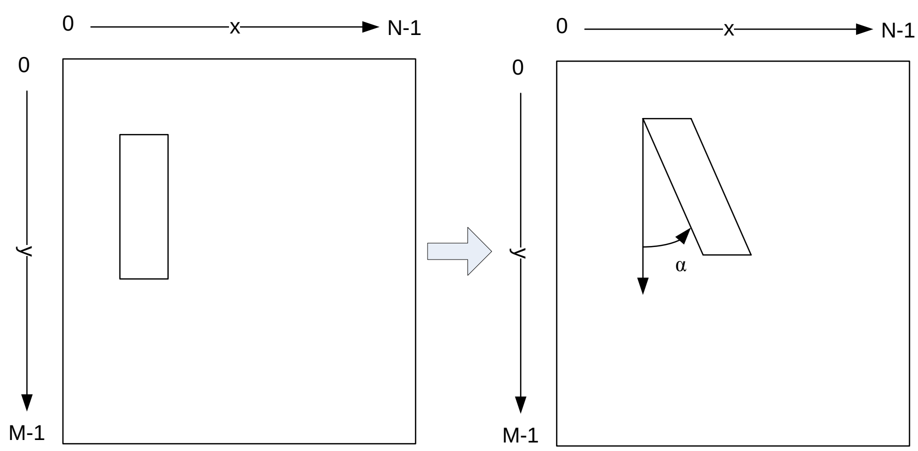


Figure 3-9: Image deformation to the right in the *x*-axis direction

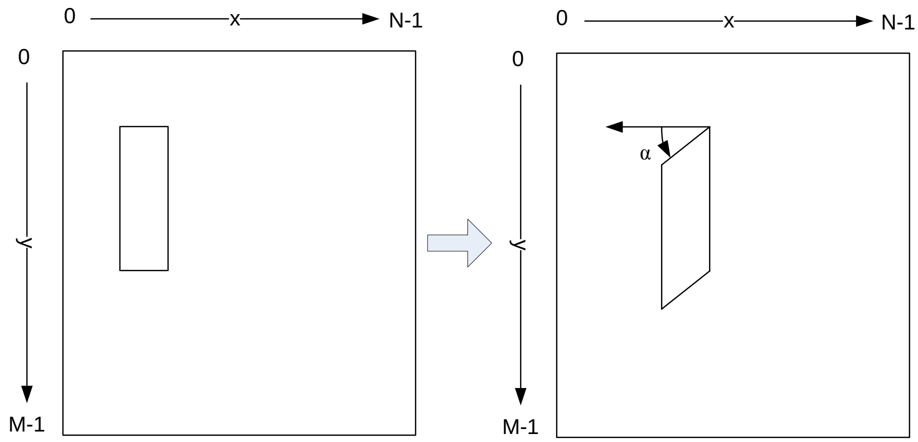


Figure 3-10: Image deformation to the lower direction of y-axis

Suppose an ellipse shaped disc were to be rotated by an axis parallel to its surface, whose orientation is not parallel to the major or minor axes, the resulting new contour will exhibit a shape conducive to be analyzed by this model.

3.9 *Image Interpolation*

A large number of geometric transformations, such as translation, rotation, and shearing will map pixels to a new position that is no longer an integer and so not on the original sampling grid. Figure 3-11 illustrates that a rotation of the image requires the evaluation of intensity at points that were not on the original grid.

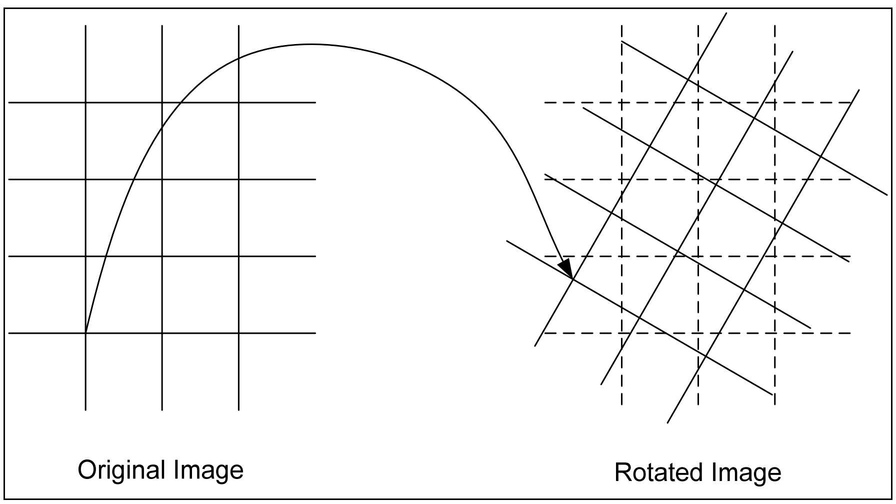


Figure 3-11: Illustration that a rotation of the image requires interpolation

Interpolation is a process of generating a value of a pixel based on its neighbors. Neighboring pixels contribute a certain weight to the value of the pixel being interpolated. This weight is often inversely proportional to the distance at which the neighbor is located.

There are several different types of interpolation methods. Nearest neighbor interpolation is the simplest method and basically makes the pixels bigger. The value of a pixel in the new image is the value of the nearest pixel of the original image. The other interpolation methods also include bilinear interpolation and bicubic interpolation. The interpolation method that is used in our DIS is bilinear interpolation. Bilinear interpolation determines the value of a new pixel based on a weighted average of the 4 pixels in the nearest 2x2 neighborhood of the pixel in the original image. Figure 3-12 shows four neighboring pixels surrounding the pixel *(x,y)* to be interpolated.

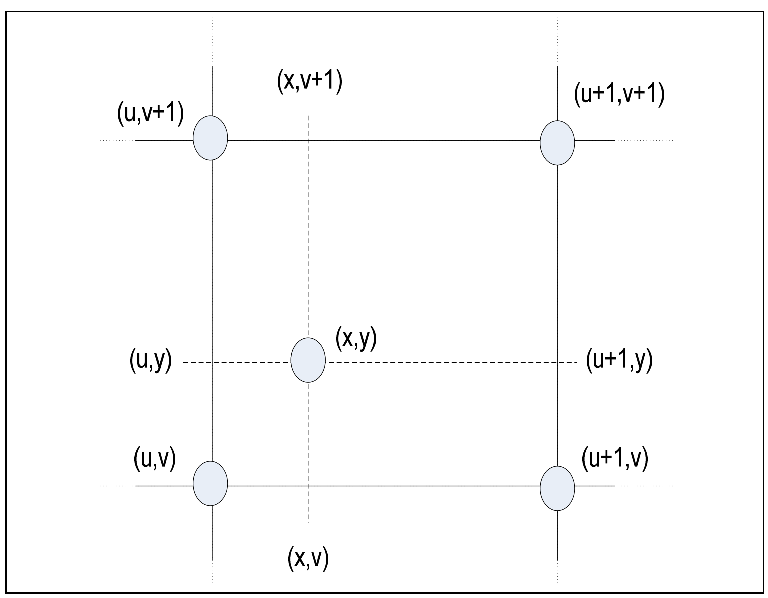


Figure 3-12: Bilinear Interpolation

In Figure 3-12, we assumed u and v are the integer parts of x and y, respectively, bilinear interpolation is defined by

$$f(x,y) = W_{u,v} f(u,v) + W_{u+1,v} f(u+1,v) + W_{u,v+1} f(u,v+1) + W_{u+1,v+1} f(u+1,v+1)$$

(3-31)

where,

$$W_{u,v} = (u+1-x)(v+1-y)$$

$$W_{u+1,v} = (x-u)(v+1-y)$$

$$W_{u,v+1} = (u+1-x)(y-v)$$

$$W_{u+1,v+1} = (x-u)(y-v)$$

The bilinear interpolation has an anti-aliasing effect and therefore produces relatively smooth edges.

IV. Methodology

A General method for DIS includes two modules: motion estimation module and motion compensation module. The motion estimation module calculates global motion vector of input frame relative to reference frame. Then, the motion compensation module processes input frame according to motion vector and stabilizes observed images. Figure 4-1 shows a block diagram of such a system.

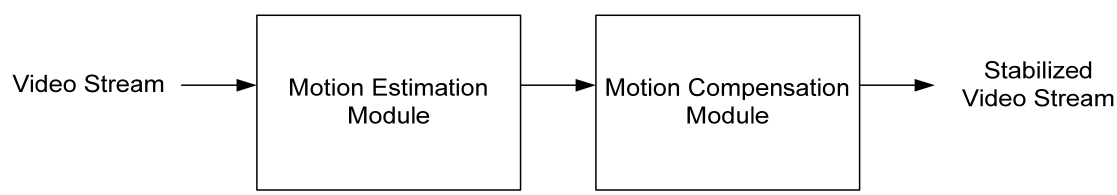


Figure 4-1: DIS Model

With the advantage of low energy consumption, light weight and compact size, DIS technique offers excellent performance in the case of low frequency and small amplitude system vibrations.

4.1 Motion Estimation Module

The DIS proposed in this thesis is based on the following assumptions that: each frame in the given image sequence is distinct, and the image instability is the result of translation, rotation, skewing and scaling between frames.

Through analyzing image frames, the motion vectors (including amounts of translation, rotation and scaling), which are the basis of compensation processing, can be calculated. Motion estimation between frames is usually based on a rigid motion model as follows:

$$\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{bmatrix} \begin{bmatrix} \cos\alpha & \sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \end{bmatrix} + \begin{bmatrix} \triangle x \\ \triangle y \end{bmatrix}$$

(4-1)

The above given model is explained in the following text. In the formula, x_{new}, x_{old} are horizontal coordinates of corresponding pixels in input frame and reference frame; y_{new}, y_{old} are vertical coordinates of corresponding pixels in input frame and reference frame; Δx, Δy are translation amounts between two frames; θ and α are the rotation and deformation angles between two frames respectively. The two factors σ_x, σ_y are the scaling factors.

Equation (4-1) can be rewritten as follows:

$$\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = A \begin{bmatrix} x_{old} \\ y_{old} \end{bmatrix} + \begin{bmatrix} \triangle y \\ \triangle y \end{bmatrix} \qquad \textbf{(4-2)}$$

where, A is a sequence of rotation, scaling and angular deformation. And it can be decomposed in the form:

$$\begin{aligned} A &= A_S A_D A_R \\ &= \begin{bmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{bmatrix} \begin{bmatrix} \cos\alpha & \sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \end{aligned}$$

Matrix A is a 4x4 matrix. So, it is in the form:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

Hence,

$$\begin{aligned} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} &= \begin{bmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{bmatrix} \begin{bmatrix} \cos\alpha & \sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \\ &= \begin{bmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{bmatrix} \begin{bmatrix} \cos(\alpha-\theta) & \sin(\alpha-\theta) \\ \sin(\alpha+\theta) & \cos(\alpha+\theta) \end{bmatrix} \\ &= \begin{bmatrix} \sigma_x \cos(\alpha-\theta) & \sigma_x \sin(\alpha-\theta) \\ \sigma_y \sin(\alpha+\theta) & \sigma_y \cos(\alpha+\theta) \end{bmatrix} \end{aligned} \tag{4-3}$$

By solving Equation (4-3),

$$\sigma_x = \sqrt{a^2{}_{11} + a^2{}_{12}} \tag{4-4}$$

$$\sigma_y = \sqrt{a^2{}_{21} + a^2{}_{22}} \tag{4-5}$$

$$(\alpha - \theta) = \operatorname{atan2}(a_{12}, a_{11}) \tag{4-6}$$

$$(\alpha + \theta) = \operatorname{atan2}(a_{21}, a_{22}) \tag{4-7}$$

To find the values of α and θ, we need to solve Equation (4-6) and Equation (4-7) simultaneously,

$$2\alpha = \mathrm{atan2}(a_{12}, a_{11}) + \mathrm{atan2}(a_{21}, a_{22})$$

$$\alpha = \frac{\mathrm{atan2}(a_{12}, a_{11}) + \mathrm{atan2}(a_{21}, a_{22})}{2} \tag{4-8}$$

By substituting the value of α into Equation (4-7),

$$\alpha = \frac{\mathrm{atan2}(a_{12}, a_{11}) - \mathrm{atan2}(a_{21}, a_{22})}{2} \tag{4-9}$$

Now, we have six variables to estimate, these values are show in Table 4-1.

Table 4-1: Motion vectors variables

Motion Vectors	Description
σ_x	The scaling factor in x axis
σ_y	The scaling factor in y axis
θ	Rotation angle
α	Deformation angle
Δx	Translation in x axis
Δy	Translation in y axis

Our aim is to estimate the elements of A and the translation vector $(\Delta x, \Delta y)$ from two given images. Since σ_x, σ_y, α, and θ are functions of the elements of A, then it is sufficient to find the value of A to get the values of

σ_x, σ_y, α, and θ. Because we have six unknowns, then we need six equations to be solved simultaneously.

Assume $\underline{x}_n$ is a feature point in an image at time=t where n is the image number. And assume $\underline{x}'_n$ is the same feature point in the same image at time=$t+1$ where n is the image number. We have agreed before in Equation (3-30) that:

$$\underline{x}'_n = A\underline{x}_n + C \tag{4-10}$$

where, $\underline{\mathrm{x}}'_{\mathrm{n}} = \begin{bmatrix} \mathrm{x}'_{\mathrm{n}} \\ \mathrm{y}'_{\mathrm{n}} \end{bmatrix}$, $\underline{\mathrm{x}}_{\mathrm{n}} = \begin{bmatrix} \mathrm{x}_{\mathrm{n}} \\ \mathrm{y}_{\mathrm{n}} \end{bmatrix}$ and $C = \begin{bmatrix} \triangle x \\ \triangle y \end{bmatrix}$

In order to estimate the value of A and C, we need 6 images; that is three images at time=t and the same 3 images but at time=$t+1$. This can be written mathematically as follows

$$\underline{x}'_1 = A\underline{x}_1 + C$$

$$\underline{x}'_2 = A\underline{x}_2 + C$$

$$\underline{x}'_3 = A\underline{x}_3 + C$$

These equations can also be expanded to the following equations:

$$x'_1 = a_{11}x_1 + a_{12}y_1 + \triangle x$$
$$y'_1 = a_{21}x_1 + a_{22}y_1 + \triangle y$$

$$x'_2 = a_{11}x_2 + a_{12}y_2 + \Delta x$$
$$y'_2 = a_{21}x_2 + a_{22}y_2 + \Delta y$$

$$x'_3 = a_{11}x_3 + a_{12}y_3 + \Delta x$$
$$y'_3 = a_{21}x_3 + a_{22}y_3 + \Delta y$$

These equations can be solved simultaneously to find the values of a_{11}, a_{12}, a_{21}, a_{22}, Δx and Δy. The above computation can be expressed in the form of matrix algebra as follows:

$$\begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \\ y'_1 \\ y'_2 \\ y'_3 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & & & \\ x_2 & y_2 & 1 & & & \\ x_3 & y_3 & 1 & & & \\ & & & x_1 & y_1 & 1 \\ & & & x_2 & y_2 & 1 \\ & & & x_3 & y_3 & 1 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ \Delta x \\ a_{21} \\ a_{22} \\ \Delta y \end{bmatrix} \quad \textbf{(4-11)}$$

Which is of the form:

$$\underline{p} = P\underline{a}$$

The above equation reveals several important facts. First, a minimum of three points must be known in each image. Second, these points should not be collinear. If they are collinear, then P can not be inverted. When more than three points are known in the images, then standard pseudo inverse computation computes an optimal estimate of $\underline{a}$ such that:

$$\underline{a} = (P^t P)^{-1} P^t p$$

4.1.1 Image Segmentation

Segmentation is the process of partitioning an image into regions or subimages. The region or the subimage here is defined as a group of pixels with similar properties. These properties include same graylevel or textures … *etc*. We will use graylevel as the property to distinguish between the subimages. The simplest representation of a segment is a binary valued image, where each pixel is assigned a value '1' if it is in the region, and a '0' otherwise.

Segmented images must satisfy the following two properties:

1. *Distinctness* :

   No pixel is shared by two regions. That is

   $$R_i \cap R_j = \varnothing \qquad \text{for } i, j = 1, \ldots, k; \quad i \neq j$$

   where, R is a subimage and k is the maximum number of subimages intended to create.

2. *Completeness*:

   All pixels in the image must be assigned to one of the k regions. That is

   $$R_1 \cup R_2 \ldots \cup R_k = I$$

where, I is the original image intended to be segmented.

The first property states that regions are disjoint sets and the second property states that the entire image I must be covered by the regions R_i , $i=1, 2, \ldots, k$.

One of the simplest methods to segment an image is to apply thresholding. Thresholding is a method for image segmentation. The cumulative histogram of the image is used to determine the proper value of the threshold. The general equation to create some binary images from a gray-scaled image can be written as follows:

$$B_n = \begin{cases} 1 & f(x,y) < k_n \\ 0 & f(x,y) \geq k_n \end{cases} \tag{4-12}$$

where, B_n is a binary image, k_n is the threshold value used in the segmentation to create this binary image and $f(x,y)$ is a gray-scaled image. One could iteratively try to determine the best threshold k_n by a systematic trial and error process. Also, well established decision techniques can be applied to estimate an optimal threshold k_n, when the parametric model of the underlying distribution (histogram) is known.

In our work, we have chosen a level-sets based approach to selecting up to six thresholds to divide image into six binary images. This approach is explained in the following text.

In order to determine the proper value of k_n , the image will be divided into regions according to the gray levels. The cumulative histogram is a useful tool to determine the threshold values needed in the segmentation. To find the best threshold values, the y-axis of the cumulative histogram which represents the number of pixels should be divided into the same number of subimages needed to create. In our DIS algorithm, the number of binary subimages is six. So, the y-axis should be divided into six portions.

Assume we have a cumulative histogram of an image plotted in Figure 4-2.

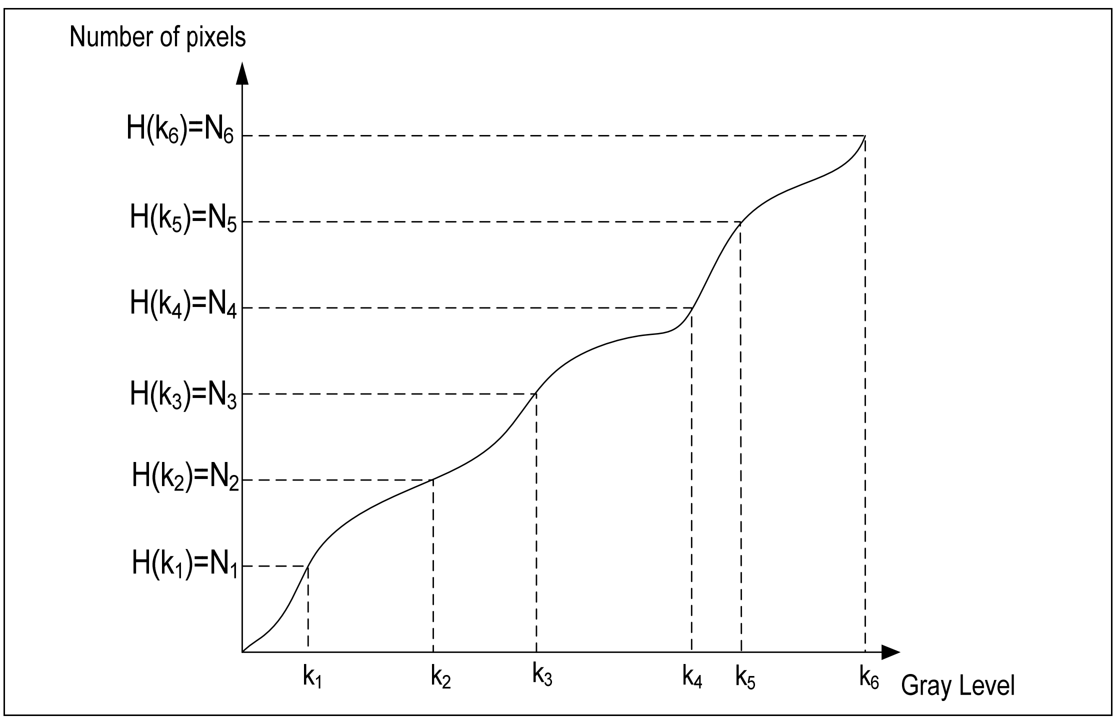


Figure 4-2: Cumulative Histogtam of an image

In this example, six binary images can be created by using a modified version of Equation (4-12) as follows:

$$B_n = \begin{cases} 1 & k_{n-1} < f(x, y) \le k_n \\ 0 & k_n < f(x, y) \le k_{n-1} \end{cases}$$

The threshold values ($k_1, k_2, k_3, k_4, k_5,$ and k_6) can be calculated as follows:

$$k_n = H^{-1}(D_n)$$

where,

$$D_n = \frac{n \times N}{m}$$

where, n is the threshold number and it can take values from 1 to the number of binary subimages, at least six in this context. N is the total number of pixels in the image. And m is the desired number of binary subimages.

The segmentation process discussed earlier will result in 6 binary subimages. All these subimages will be used to estimate the value of motion vectors ($\sigma_x, \sigma_y, \alpha, \theta, \Delta x,$ and Δy). Among the six subimages, each time we will use 3 subimages to estimate the motion vectors. So, the resultant number of motion vectors will be:

$$\binom{6}{3} = \frac{6!}{(6-3)! \times 3!} = 20$$

Hence, we will have 20 motion vectors. A question arises here, which one of these motion vectors should be used?

To determine the best motion vector, we are using a statistical tool called "clustering". To prepare our data to be clustered and then ready for analysis, the θ and α values should be plotted in x-axis and y-axis respectively. Then, a clustering technique will be used to analyze these data. The following section explains in detail the idea of "clustering" and why we need it here.

4.1.2 Data Clustering

Clustering is a classical topic in statistical data analysis and machine learning. There is much research work discussing clustering methods [5]. It is defined as the process of grouping a set of objects into classes of similar objects. We can show this with a simple graphical example as in Figure 4-3.

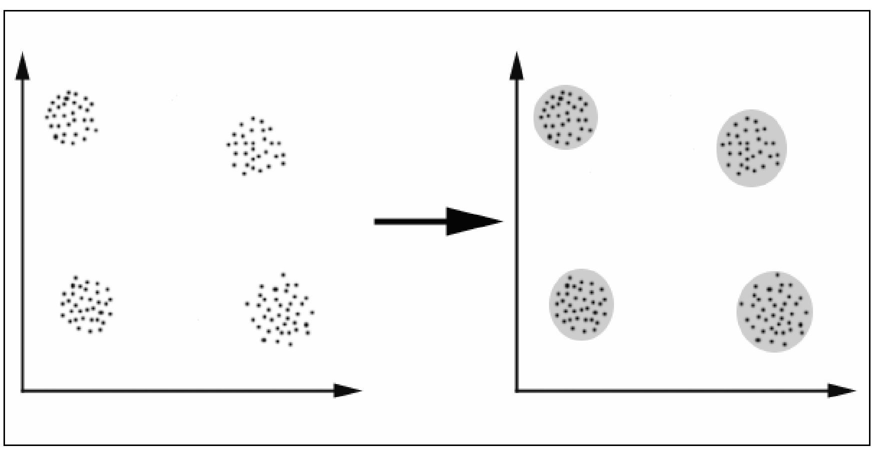


Figure 4-3: Data Clustering example [34].

The most well known similarity measures are based on distances, such as Euclidean distance and Manhattan distance. There are many algorithms can be used to implement data clustering. In this thesis, a graph theoretic algorithm will be used to do the job. Graph theoretic algorithm for clustering is a technique based on modified Kruskal's algorithm. The purpose is to take the advantage of the simplicity of tree structure, which can facilitate efficient implementations of much more sophisticated clustering algorithms. There are many variations in the family of graph theoretic algorithms, such as Minimal Spanning Tree (MST) based method, Cut algorithm, and Normalized Cut/Spectral methods. In general, the idea of graph theoretic algorithms is the following: firstly, it constructs a weighted graph upon the points in the high-dimensional space, with each point being a node, we will use (θ,α) as nodes and the distance value between any two nodes being the weight of the edge connecting the two nodes. Then, it decomposes the graph into connected components in some way, and calls those components as clusters or forests. We mainly focus on an MST-based clustering algorithm using Kruskal's algorithm. Kruskal's algorithm is used to create minimum spanning tree and it works as follows:

1. Consider the edges from shortest to longest.
2. Take the first (smallest) edge and then consider the next edge.
3. Take an edge if it does not make a cycle.

4. If you still have edges then go to step 1.

In our case here, we will not continue to create a full minimum spanning tree because this is not our goal. Instead, we will stop whenever we have an optimum cluster which satisfies our conditions. The conditions we set to be satisfied are two:

a. The cluster (forest) should contain all the subimages.
b. The distance between any two nodes in the cluster should not exceed 10% of the largest distance between the nodes.

Eventually, we will get a cluster with some nodes. Assume F is the chosen cluster, then F=$\{n_1, n_2, \ldots, n_n\}$, where, n_i is a node in (θ,α) graph. We know that every node here is calculated using 3 subimages as we have seen earlier.

Because every node is constructed by three subimages, then assume:

$n_1 = (p_1, q_1, r_1)$

$n_2 = (p_2, q_2, r_2)$

...

$n_n = (p_n, q_n, r_n)$

where, p_i, q_i *and* r_i, are subimages.

For every node n_i, the size of the subimage with the minimum size (w) will be used to calculate the final motion vectors as follows:

$$\hat{\alpha} = \frac{\sum_{i=1}^{n} (w_i . \alpha_i)}{\sum_{i=1}^{n} w_i} \tag{ 4-13}$$

$$\hat{\theta} = \frac{\sum_{i=1}^{n} (w_i . \theta_i)}{\sum_{i=1}^{n} w_i} \tag{4-14}$$

$$\hat{\sigma}_x = \frac{\sum_{i=1}^{n} (w_i . \sigma_{xi})}{\sum_{i=1}^{n} w_i} \tag{4-15}$$

$$\hat{\sigma}_y = \frac{\sum_{i=1}^{n} (w_i . \sigma_{yi})}{\sum_{i=1}^{n} w_i} \tag{4-16}$$

$$\Delta\hat{x} = \frac{\sum_{i=1}^{n} (w_i . \Delta x_i)}{\sum_{i=1}^{n} w_i} \tag{4-17}$$

$$\Delta\hat{y} = \frac{\sum_{i=1}^{n} (w_i . \Delta y_i)}{\sum_{i=1}^{n} w_i} \tag{4-18}$$

where,

$$w_i = \min_{i=1\ldots n}(size(p_i), size(q_i), size(r_i))$$

At the end of this phase of the DIS algorithm, we have the estimated motion vectors (σ_x, σ_y, α, θ, Δx, and Δy) which are necessary in the next phase which is "Motion Compensation".

4.2 Motion Compensation Module

The result of the motion estimation process described in the last section is capable of computing the motion vectors between two frames. The objective of motion compensation is to keep some kind of history of the motion estimates in order to create a stabilized sequence. We have seen that the DIS proposed is based on a hypothesis that the image instability in image sequence is the result of translation, rotation, skewing and scaling between frames. So, by knowing these motion vectors which are estimated in the last section, an image can be constructed.

An image can be constructed using the hypothesis in Equation (4-10):

$$\underline{x}_n' = A\underline{x}_n + C$$

where, A as calculated in the last section:

$$= \begin{bmatrix} \hat{\sigma}_x \cos(\hat{\alpha} - \hat{\theta}) & \hat{\sigma}_x \sin(\hat{\alpha} - \hat{\theta}) \\ \hat{\sigma}_y \sin(\hat{\alpha} + \hat{\theta}) & \hat{\sigma}_y \cos(\hat{\alpha} + \hat{\theta}) \end{bmatrix}$$

and

$$C = \begin{bmatrix} \triangle \hat{x} \\ \triangle \hat{y} \end{bmatrix}$$

As we know already, pixels of an image occupy integer coordinates. We can note from Equation (4-10) that the destination pixels may lie between the integer coordinates. So, in order to create an image from these pixels, destination pixels are interpolated at the integer coordinates.

4.3 Frequency Domain Approach To Estimate Image Translation

This section introduced the Fourier transform based approach to estimate image translation between two images.

4.3.1 Introduction

So far in this thesis, we have considered only one kind of image representation. For the most part, the images have recorded brightness as a function of position. In practice, there are many different ways in which image data can be presented. These changes of representation are useful to analyze certain characteristics of the images more efficiently. Some kinds of transformation produce representations which, although different from the original data, are completely equivalent to it in terms of the information contained. These so-called domain transforms, of which the Fourier transform is by far the most important, allow images to be treated in ways entirely different from those used on the original data.

4.3.2 Transforming Domain

Domain transforms provide alternative ways of describing an image. Instead of recording brightness as a function of position, we can choose a completely different presentation of the image.

In Fourier transform, the image is stored as a set of spatial frequency values together with their associated amplitudes and phase. The point is that instead of brightness as a function of position, the Fourier representation is a complex valued function of spatial frequency. In the frequency domain, changes in image position produce a noticeable changes in the phase of each spectral component. The phase change can be quantitively measured, and used to characterize the motion.

4.3.3 Fourier Transform of an Image

As we are only concerned with digital images, we will use the Discrete Fourier Transform (DFT). The DFT is the Fourier Transform variation used in digital processing.

The Fourier transform of a *M* x *N* image is shown mathematically as:

$$H(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} h(u,v) e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}, \qquad -\pi < \theta \leq \pi \tag{4-19}$$

where, *h(x,y)* is the image to be transformed and *H(x,y)* is the transformed one.

It is also possible to transform image from the frequency domain back to the spatial domain. This is done with an inverse Fourier transform as follows:

$$h(x,y) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} H(u,v)e^{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)} \qquad \textbf{(4-20)}$$

In the frequency domain, u represents the spatial frequency along the original images axis and v represents the spatial frequency along the y axis. In the center of the image u and v have their origin.

The Fourier transforms deals with complex numbers. The magnitude is expressed as:

$$|H(u,v)| = \sqrt{R^2(u,v)+I^2(u,v)} \qquad \textbf{(4-21)}$$

and phase as:

$$\theta(u,v) = \tan^{-1}\left[\frac{I(u,v)}{R(u,v)}\right] \qquad \textbf{(4-22)}$$

where, *R(u,v)* is the real part and *I(u,v)* is the imaginary.

The frequency is dependent on the pixel location in the transform. The further from the origin it is, the higher the spatial frequency it represents.

The computation of the Fourier transform stores the real and imaginary spectral components in arrays, starting with positive frequency values followed by negative frequency values. Figure 4-4 shows an example of the magnitude spectrum from a one-dimensional DFT, showing that the negative frequency components follow the positive frequency components.

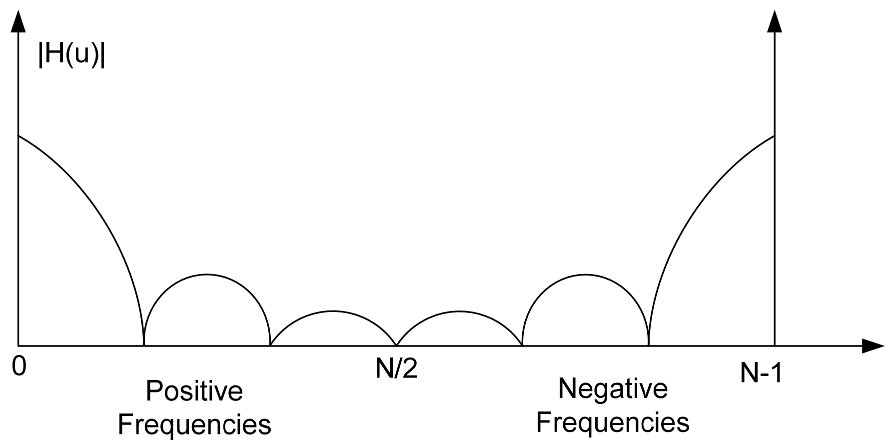


Figure 4-4: **Uncentered magnitude spectrum**

Normally when plotting spectral components using the Cartesian coordinate system, negative frequency components are plotted first followed by the positive frequency components. The first half and last half of the array of Fourier components must be swapped. This can be done by multiplying every pixels in the image by $(-1)^{x+y}$, that is:

$$f(x,y) \Rightarrow f(x,y)(-1)^{x+y} \qquad (4\text{-}23)$$

Equation (4-23) is the centering property of the DFT.

Figure 4-5 shows the output from the DFT after application of the centering property.

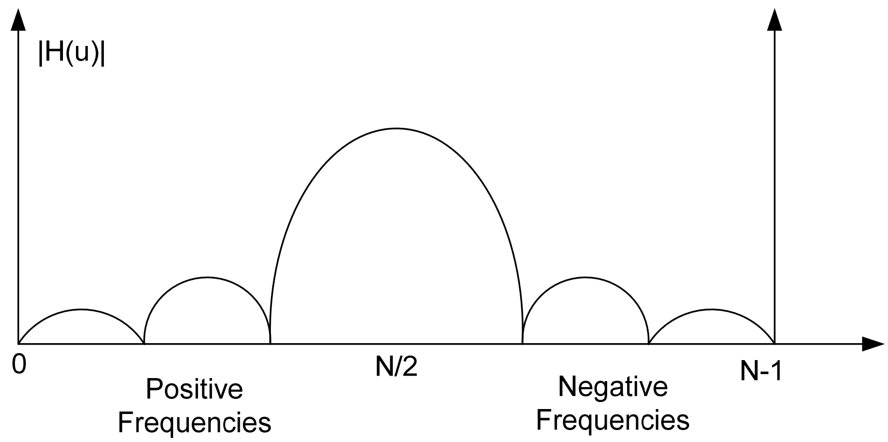


Figure 4-5: The Centered magnitude spectrum

Figure 4-6 shows the uncentered magnitude spectrum of an image containing a white object. Figure 4-7 shows the DFT spectrum of the same image after application of the centering property.

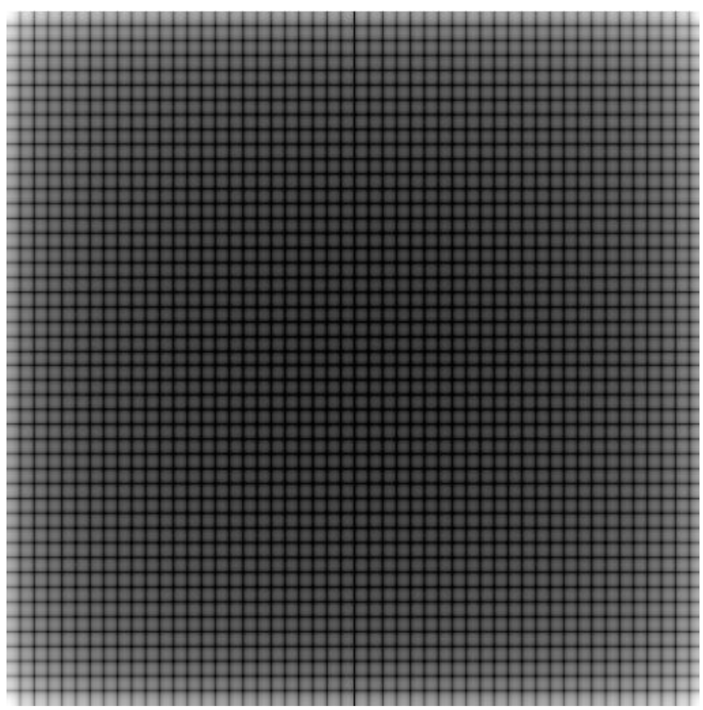


Figure 4-6: Uncentered spectrum of an image

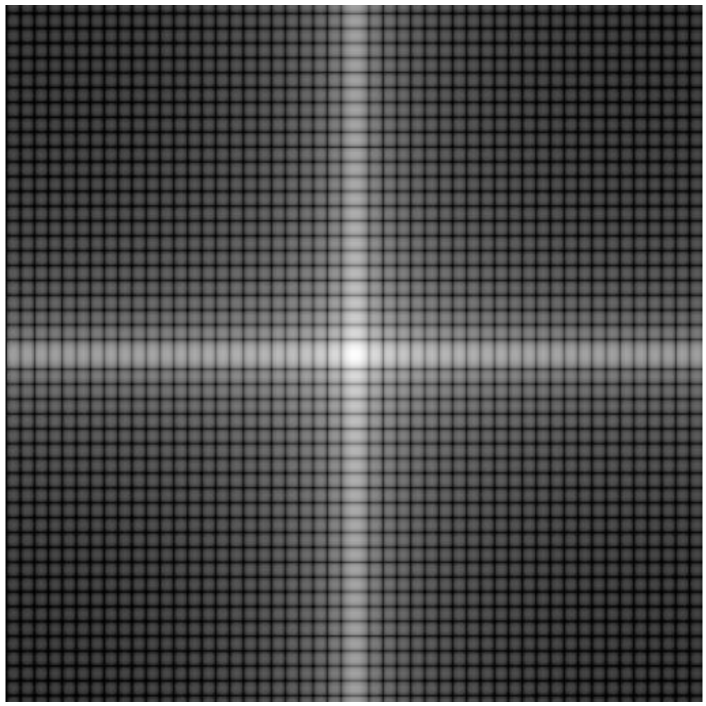


Figure 4-7: The centered spectrum after using centring property of DFT.

4.3.4 Translation Estimation

In this chapter, a frequency domain method is investigated for estimating the translation between two images.

The motion between a reference image and the second is assumed to be a pure translation. Considering such kind of displacement between two images the motion may be simply described by two parameters: horizontal and vertical shifts.

In the Fourier transform domain relation between two mutually shifted images can be expressed as follows:

$$f_2(x,y) = f_1(x-a, y-b) \leftrightarrow F_1(u,v)e^{j\frac{2\pi}{N}(au+bv)} \quad \textbf{(4-24)}$$

where, $f_1(x,y)$ is the reference image and $f_2(x,y)$ is the shifted one. $F_1(u,v)$ is the Fourier transform of $f_1(x,y)$. It is known from Fourier transform properties that a spatial shift results in multiplication.

What we want here is to find out the values of a and b in equation (4-24) because they represent the vertical and horizontal shifts.

We know from equation (4-24) that:

$$F_2(u,v) = F_1(u,v)e^{j\frac{2\pi}{N}(au+bv)} \quad \textbf{(4-25)}$$

Dividing Equation (4-25) by $F_1(u,v)$:

$$\frac{F_2(u,v)}{F_1(u,v)} = e^{j\frac{2\pi}{N}(au+bv)} \quad \textbf{(4-26)}$$

The right hand side term on this equation is a complex number and it can be split into two parts: the real part $R(u,v)$ and the imaginary part $S(u,v)$. The phase of this complex number can be calculated as follows:

$$\theta(u,v) = \mathrm{atan2}(S(u,v), R(u,v)) \quad \textbf{(4-27)}$$

The phase also can be found as follows:

$$\theta(u,v) = \frac{2\pi}{N}(au+bv) \quad \textbf{(4-28)}$$

So, for every point in the frequency domain, there is a phase as in Table 4-2.

Table 4-2: Phase of every point in u-v plane

u-v plane points	Phase
(u_1,v_1)	$\theta(u_1,v_1)$
(u_2,v_2)	$\theta(u_2,v_2)$
...	...
(u_N,v_N)	$\theta(u_N,v_N)$

By finding the phase of every point using Equation (4-28) and finding the square mean error of these phases, we get:

$$E = \sum_{i=1}^{N}\left[\frac{2\pi}{N}(au_i + bv_i) - \theta(u_i,v_i)\right]^2 \tag{4-29}$$

The error should be kept minimal. So,

$$E = \sum_{i=1}^{N}\left[\frac{2\pi}{N}(au_i + bv_i) - \theta(u_i,v_i)\right]^2 = \text{minimum} \tag{4-30}$$

$$\frac{\partial E}{\partial a} = 0, \qquad \frac{\partial E}{\partial b} = 0 \tag{4-31}$$

$$\frac{\partial E}{\partial a} = \sum_{i=1}^{N} 2\left[\frac{2\pi}{N}(au_i + bv_i) - \theta(u_i,v_i)\right].\frac{2\pi}{N}u_i = 0 \tag{4-32}$$

$$\frac{\partial E}{\partial b} = \sum_{i=1}^{N} 2\left[\frac{2\pi}{N}(au_i + bv_i) - \theta(u_i,v_i)\right].\frac{2\pi}{N}v_i = 0 \tag{4-33}$$

$$\sum_{i=1}^{N}\left(\frac{2\pi}{N}\right)^2 au_i^{\,2}+\sum_{i=1}^{N}\left(\frac{2\pi}{N}\right)^2 bu_iv_i^{\,2}=\sum_{i=1}^{N}\frac{2\pi}{N}u_i\theta_i \qquad \textbf{(4-34)}$$

$$\sum_{i=1}^{N}\left(\frac{2\pi}{N}\right)^2 u_iv_ia+\sum_{i=1}^{N}\left(\frac{2\pi}{N}\right)^2 bv_i^{\,2}=\sum_{i=1}^{N}\frac{2\pi}{N}v_i\theta_i \qquad \textbf{(4-35)}$$

$$\begin{bmatrix} \frac{2\pi}{N}\sum_{i=1}^{N}u_i^2 & \frac{2\pi}{N}\sum_{i=1}^{N}u_iv_i \\ \frac{2\pi}{N}\sum_{i=1}^{N}u_iv_i & \frac{2\pi}{N}\sum_{i=1}^{N}v_i^2 \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix}=\begin{bmatrix} \sum_{i=1}^{N}u_i\theta_i \\ \sum_{i=1}^{N}v_i\theta_i \end{bmatrix} \qquad \textbf{(4-36)}$$

The last equation gives the values of *a* and *b* which we are looking for.

4.3.5 Rotation and Scaling Estimation

The underlying computations to estimate rotation and scaling in Fourier domain are not as simple as in translation estimation. The complexities of the model poses a serious question about the efficiency of finding rotation and scale change in frequency domain vs. the same in spatial domain. Such challenges stem from the fact that:

Given, $f(x,y)\longleftrightarrow F(u,v)$, then:

$$f(x\cos\theta+y\sin\theta,-x\sin\theta+y\cos\theta)\longleftrightarrow F(u\cos\theta-v\sin\theta,u\sin\theta v\cos\theta)$$

So, the rotation in the spatial domain is also rotation in the frequency domain but in the opposite direction as shown in Figure 4-8.

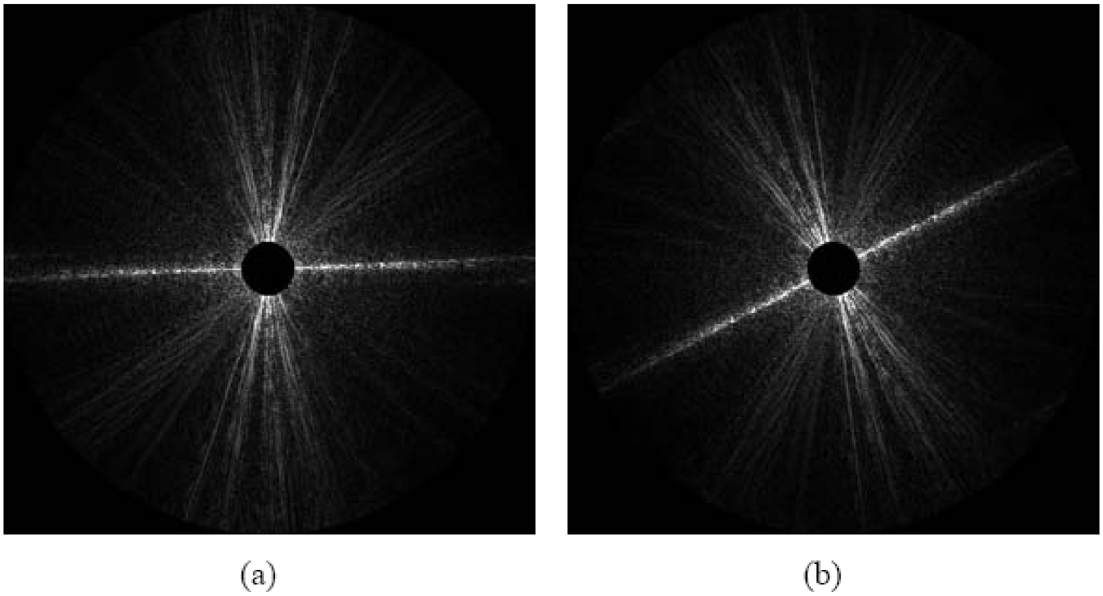


Figure 4-8: Rotation estimation. (a) Frequency values of a reference image. (b) Frequency values of the rotated image (θ=25 degrees) [37]

So, using this method will not simplify the rotation estimation. Same thing is applied in the case of scaling based on the fact that scale change in the spatial domain is also scale change in the frequency domain. This can be expressed as follows: Given $f(x) \longleftrightarrow F(w)$, then $f(ax) \longleftrightarrow \frac{1}{|a|} F\left(\frac{w}{a}\right)$.

4.4 Affine-Motion Inversion Scheme for Jitter Detection

In this section, a new approach to detect jitters in a sequence of video frames is introduced. The approach seeks to model the underlying changes a series of 2D affine transforms between consecutive video frames, without resorting to a three dimensional interpretation of the physical factors that give a rise to the changes.

The affine transformation is represented by a set of invariants to be estimated by weighted geometric moments of each observed image. In particular, the image-plane will be viewed as a collection of non-overlapping concentric regions of varying weights of interest. Thus, the moments will be calculated using a geometric-location weighted and intensity weighted computations.

The approach proposed is a simplified strategy to decide if the disparity between a video frame and its predecessor is due to a smooth motion or an erratic jitter. Six moments-based descriptors and the gray level histogram are used to arrive at that decision. Individual parameters used for such decision include: the change in direction in the apparent motion of the weighted center of gravity, the discontinuities in the angular velocity of the eigen-vectors of the scatter matrix (second order moments), and the dynamics of the focus-of-expansion of the observed ego-centric optical flow field. These computations are progressively complex. They will be implemented at different temporal sampling rate, i.e., the simplest method will be applied to every frame while the advanced method will be applied to every other frame, etc.

The basic method computes six numbers: M_{00}, M_{01}, M_{10}, M_{11}, M_{20}, *and* M_{02}. Which are defined as:

$$M_{ij} = \sum_{x,y} x^i y^j w(x,y) f(x,y) \tag{4-37}$$

The image *f(x,y)* and the weights *w(x,y)* are expressed on a 640x480 grid to be indexed as $-320 \le x < 320$, and $-240 \le y < 240$, reflecting a zero-centered image plane. A standard Gaussian function is used for *w(x,y)* with an arbitrary chosen half-power radius of 128. Then, *w* is computed as:

$$w(x,y) = e^{\left(-\frac{x^2+y^2}{2\sigma^2}\right)}, \qquad \sigma = 128 \tag{4-38}$$

The centroid of an image is generally the simplest descriptor of the image, which is depicted by:

$$\left(\mu_x, \mu_y\right) = \left(\tilde{M}_{10}, \tilde{M}_{01}\right), \tag{4-39}$$

Or

$$\left(r_\mu, \theta_\mu\right) = \left(\sqrt{\mu_x^2 + \mu_y^2}, \operatorname{arctan2}(\mu_y, \mu_x)\right); -\pi < \theta_\mu \le \pi. \tag{4-40}$$

The gravity adjusted second order moments, $\tilde{M}_{20}, \tilde{M}_{02}$ and $\tilde{M}_{11}$ are defined as:

$$\tilde{M}_{ij} = \frac{\sum_{x,y} (x-\mu_x)^i (y-\mu_y)^j w(x,y) f(x,y)}{\sum_{x,y} w(x,y) f(x,y)} \tag{4-41}$$

It is useful to represent them in the form:

$$\begin{pmatrix} \tilde{M}_{20} & \tilde{M}_{11} \\ \tilde{M}_{11} & \tilde{M}_{22} \end{pmatrix} = \begin{pmatrix} \cos\theta_1 & \cos\theta_2 \\ \sin\theta_1 & \sin\theta_2 \end{pmatrix} \begin{pmatrix} \lambda_1^2 & 0 \\ 0 & \lambda_2^2 \end{pmatrix} \begin{pmatrix} \cos\theta_1 & \sin\theta_1 \\ \cos\theta_2 & \sin\theta_2 \end{pmatrix}; -\pi < \theta_1, \theta_2 \le \pi;\ \text{and,}\ \lambda_1 \ge \lambda_2.$$

(4-42)

The factorization given above represents the scatter composition of the spatial data. The eigen-values and the eigen-vectors describe the underlying shape more succinctly. Thus, each frame $f(x,y;t)$ represented by a vector, $\Phi(t) = (r_\mu, \theta_\mu, \lambda_1, \lambda_2, \theta_1, \theta_2; t)^T$, in addition to the standard and weighted histograms: $h[g]$, and $h_w[g]$, respectively.

In general a smoothly varying image sequence must entail smooth variations in these parameters. For example, a constant motion of the aircraft above an otherwise static landscape would entail gradual variation in (r_μ, θ_μ) indicating a constant velocity or constant acceleration. This can be verified by computing the first and second derivatives (with respect to time) of these spatio-temporal parameters, $\theta_\mu, r_\mu, \ldots$, etc. Jitters are indicated by sudden discontinuities in velocities due to impulsive or transient forces. The simplest approach to detecting abrupt discontinuities in a function is to apply a derivative operator, and decide positive if the output magnitude is above a certain threshold. The threshold may have to be determined adaptively.

Presence of high frequency signals and noisy data pose serious challenge to this approach. Thus, it is useful to preprocess the data to cancel the effect of noise. Then, we run the risk of de-emphasizing a valid edge data, due to the low pass filtering nature of such preprocessing steps. We have chosen to apply a robust multiresolution technique, called Laplacian of Gaussian, also know as $\nabla^2 * G$.

It is essentially a finite impulse response digital filter, whose continuous (analog) impulse response is of the form:

$$h(x,\sigma) = \frac{\partial^2}{\partial x^2} * e^{-\frac{x^2}{2\sigma^2}} \tag{4-43}$$

The function looks like an inverted Mexican hat as shown in Figure 4-9.

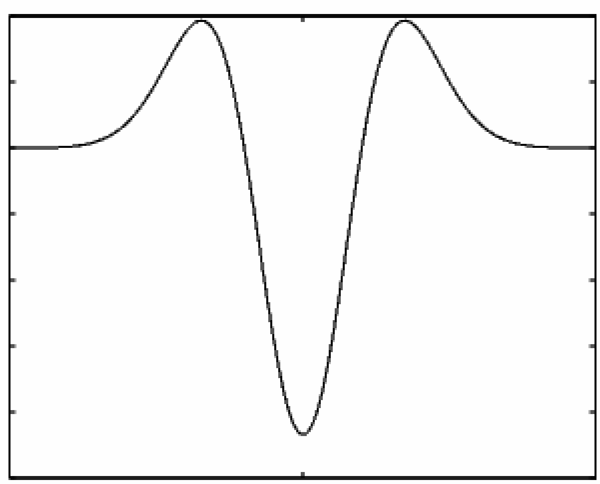


Figure 4-9: Inverted Mexican hat signal

We compute:

$$g(n) = h(n) \oplus f(n)$$

$$S(n) = sign(g(n))$$

$$E(n) = S(n)\ XOR\ S(n-1)$$

where, *g(n)* is the generated signal used for detection, *h(n)* is the digital filter whose equation is (4-43), *S(n)* is a sign function which can be expressed as follows:

$$sign(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases} \tag{4-44}$$

E(n)=1, whenever there is an edge. The procedure must be repeated at least for six σ, i.e., $h(x, \sigma_0)$, $h(x, \sigma_0 r)$, $h(x, \sigma_0 r^2)$, *...etc.* where r=1.1. The reason why we have at least six σ*'s,* is to accommodate wide range of variations in the imaging conditions. The typical values of *h(x)* for various scales and resolutions are shown in Table 4-3 and plotted in Figure 4-10.

Table 4-3: The digital filter used in the jitter detection process

Mask	Mask values							
h_1	-0.8094	-0.10289	0.3861	0.112726	0.008549	0.000209	0	0
h_2	-0.6849	-0.32967	0.24046	0.292063	0.114655	0.02239	0.002402	0.000147
h_3	-0.72272	-0.27137	0.317401	0.246125	0.06186	0.006962	0.000382	0
h_4	-0.64948	-0.37242	0.150245	0.306644	0.175519	0.05371	0.009892	0.001147
h_5	-0.74284	-0.23582	0.347275	0.214785	0.04177	0.003404	0	0
h_6	-0.68486	-0.32973	0.240357	0.292101	0.114727	0.022418	0.002406	0.000148

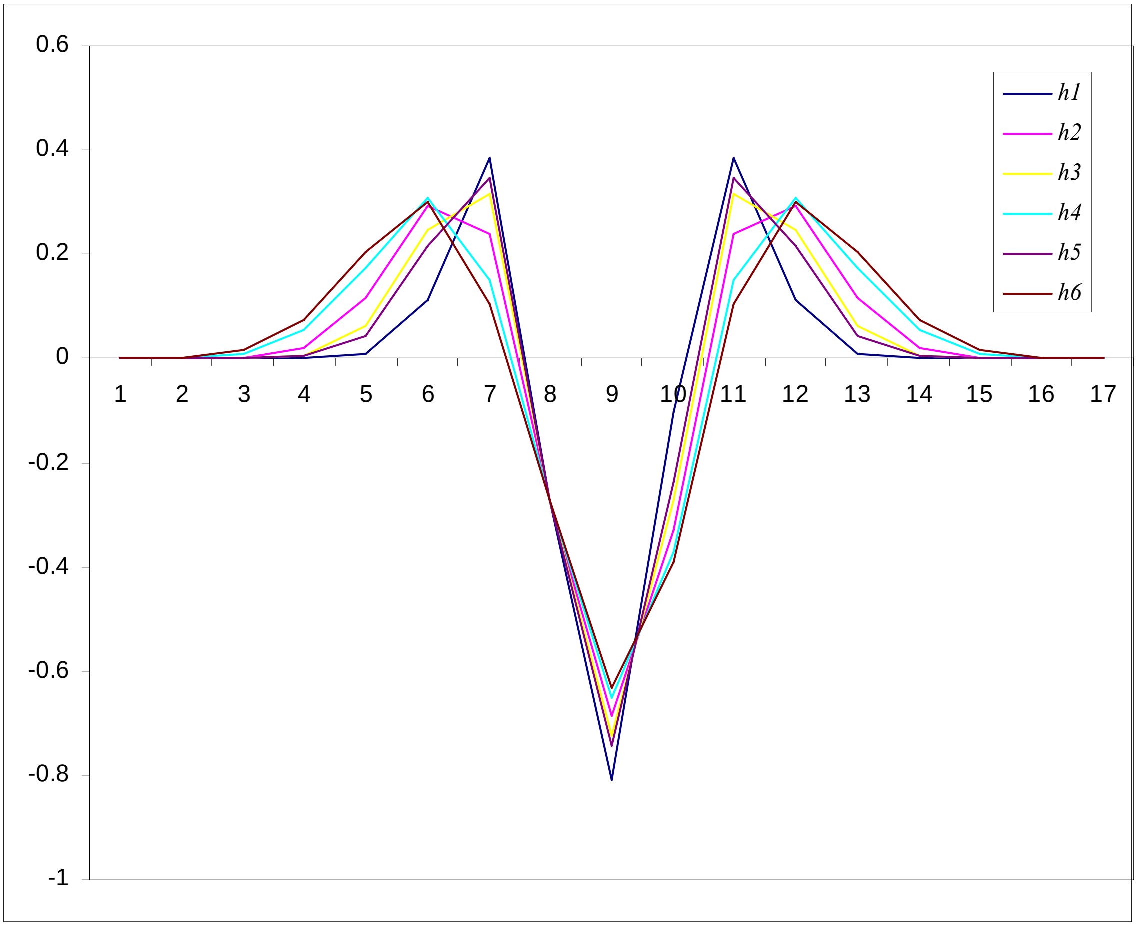


Figure 4-10: The digital filter plot used in the jitter detection process

If a zero-crossing is found in at least three *σ's,* this gives an indication of existence of jitter. The next zero-crossing of these *σ's* indicates the end of the jitter. All frames between these two consecutive zero-crossing instances are to be dropped and replaced by a suitable postprocessed images. We choose a simple method which take the first and the last images in the sequence, to generate a smoothly varying image sequence. This image sequence is computed by the FFT estimator described in section 4.3.

V. Results and Analysis

This chapter describes the experimental setup and the results obtained in testing all algorithms developed in this thesis.

The results of the affine model fitting algorithm to estimate motion vectors will be discussed first. The second section describes the test results for the frequency domain method to estimate the inter-frame translation in an image sequence. Finally, the effectiveness of jitter detection technique will be demonstrated.

5.1 Affine Based Approach for Motion Estimation

This approach was tested in a simulation setup and a practical experiment. The goal of the simulation was to evaluate the performance in a controlled environment with exact knowledge about the shift and rotation values between two images in an image sequence. This enables us to evaluate the performance and sensitivity of the algorithm. We have also tested the algorithm on a real world imagery without any modifications

5.1.1 Simulation

The scene we used in the simulation was the same scene used in the experimental section. However, the principale difference stems from the way we preprocessed the image frames. For the simulated testing, each image was padded with a black (all zeros) pixels background. In contrast, the realistic testing considers the image data without adding a well defined boundary conditions.

For the simulation, we started from an image shown in Figure 5-1. It is of size 700 x 400 pixels with a zero-padded background making the total size of image 720 x 480 pixels.

Figure 5-1: Reference image

Two images were constructed by shifting and rotating the original image. These two images were then used as inputs to the algorithm to estimate the motion vectors between each image and the original image. The first image shown in Figure 5-2 is constructed by shifting the original image by 19 pixels in the positive x-axis direction.

Figure 5-2: The constructed image by shifting the original image by 19 pixels

When only shifts are applied, the motion vectors estimation produces perfect results. Table 5-1 shows the estimated vectors.

Table 5-1: Estimated Motion Parameters for Pure Image Translation

Parameter	Value	
	Actual	Estimated
$\hat{\sigma}_x$	1	1
$\hat{\sigma}_y$	1	1
$\hat{\theta}$	0	0
$\hat{\alpha}$	0	0
$\Delta\hat{x}$	19	19
$\Delta\hat{y}$	0	0

The second image shown in Figure 5-3 is constructed by rotating the original image by $\theta = 0.0524$ radians. The rotation was centered at the center of the image grid.

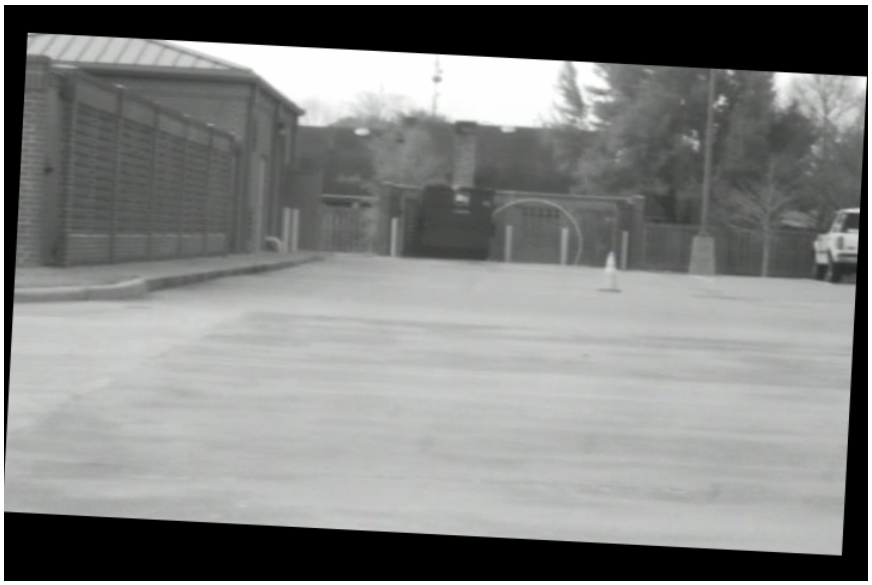


Figure 5-3: The constructed image by rotating the original image by 0.0524 radians

When rotation is applied, as shown in Figure 5-3, the rotation angle is accurately estimated but with not well estimated shifts. Table 5-2 shows the actual parameters along with the estimated ones.

Table 5-2: Estimated Motion Parameters for Pure Image Rotation

Parameter	Value	
	Actual	Estimated
$\hat{\sigma}_x$	1	0.95496
$\hat{\sigma}_y$	1	1.0236
$\hat{\theta}$	0.0524	0.0577
$\hat{\alpha}$	0	0.0126
$\Delta\hat{x}$	0	31.662
$\Delta\hat{y}$	0	-18.287

The errors in the shift estimation are due to the interpolation approximation made when rotating the image.

Finally, the original image in Figure 5-1 and the two estimated motion vectors in Table 5-1 and Table 5-2 are used as inputs to the reconstruction part of the algorithm. Figure 5-4 and Figure 5-5 show the reconstructed images from the estimated motion parameters in Table 5-1 and Table 5-2 respectively.

Figure 5-4: Reconstructed image from Table 5-1

Figure 5-5: Reconstructed image from Table 5-2

The precision of the image in Figure 5-4 was sufficient to have a good reconstruction. However, the image in Figure 5-5 is in acceptable precision except for the shifts.

5.1.2 Experiment

As described in the last section, the images used in the experimental testing are a real world images without any preprocessing. Two images taken at time=*t* and time=*t+1* were to be considered. These two images are shown in Figure 5-6 and Figure 5-7 respectively.

Figure 5-6: Image taken at time = *t*

Figure 5-7: Image taken at time = *t+1*

Because of the small amount of motion between the two consecutive images, the non-overlapping part between them is small. Figure 5-8 shows an inverted version of the difference between the two images.

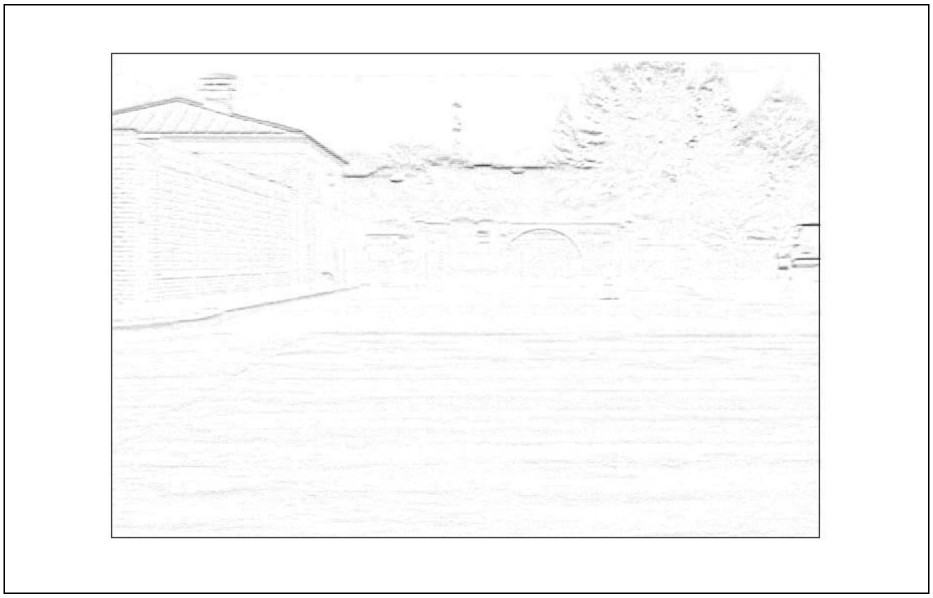


Figure 5-8: Difference between image at time= *t* and image at tim= *t+1*

Unlike the simulation section, we will have an inside and more deep look at the algorithm in this section. Firstly, we will start by plotting the histogram and the cumulative histogram of the image at time=*t* which is the start point of the segmentation process. The histogram and the cumulative histogram are shown in Figure 5-9 and Figure 5-10 respectively. Figure 5-10 also shows the gray levels of the six segmented subimages B_1 to B_6. Figure 5-11 shows these subimages.

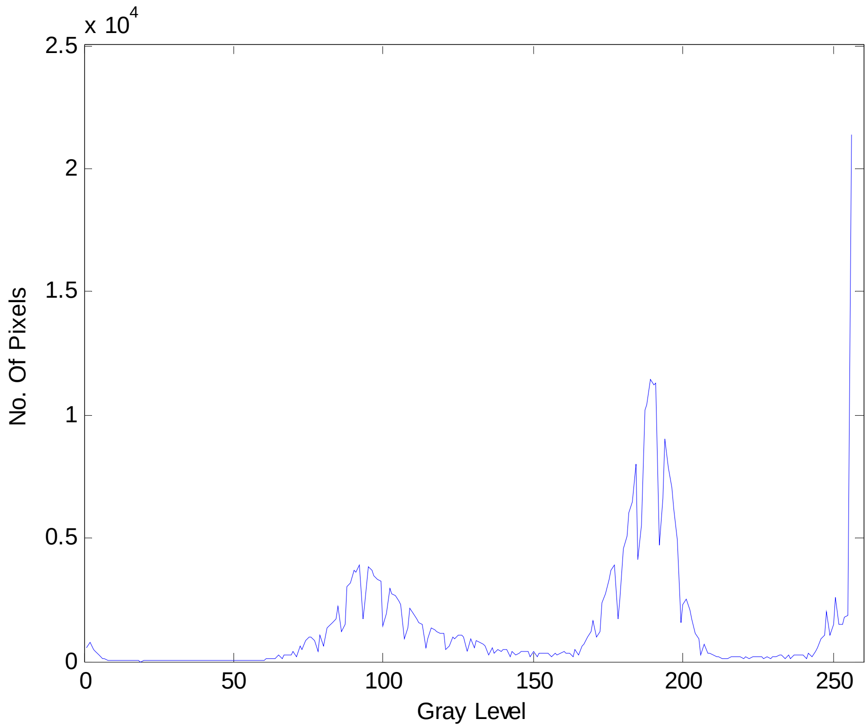


Figure 5-9: Image Histogram

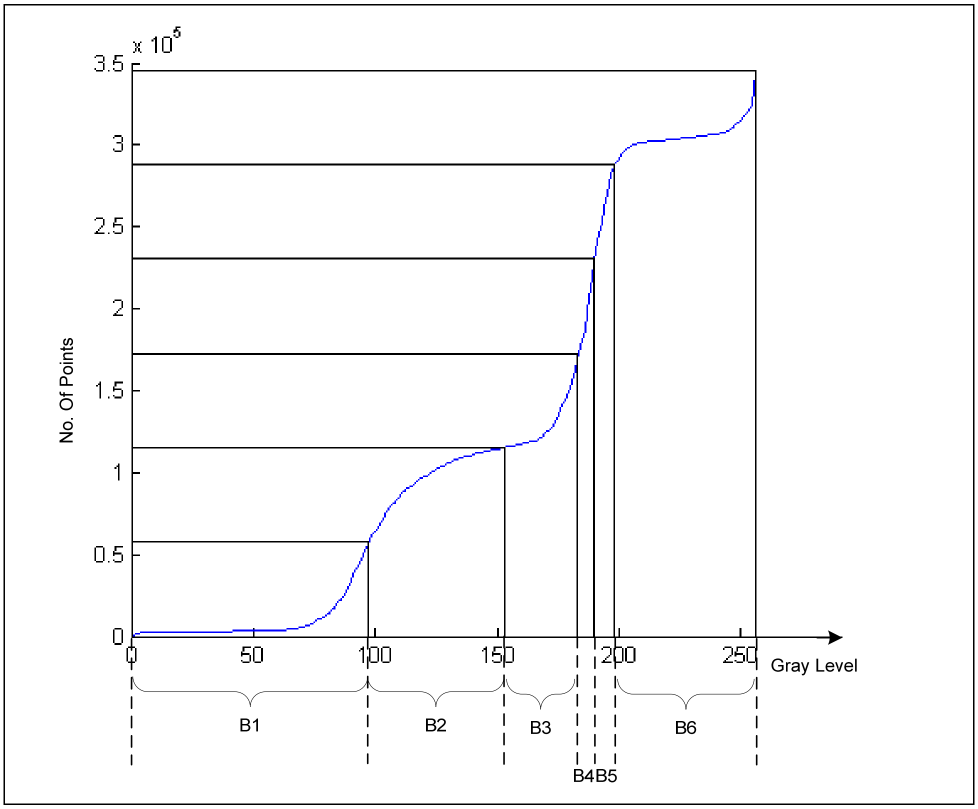
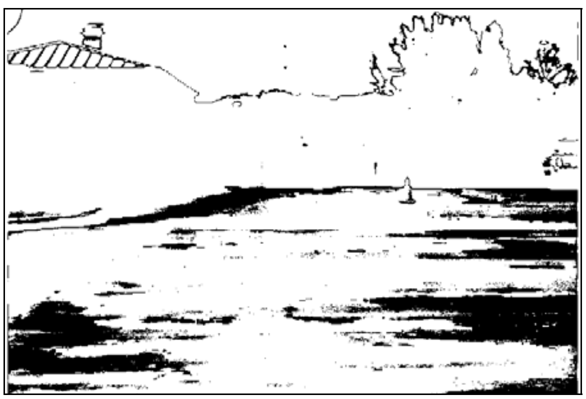


Figure 5-10: Cumulative Histogram

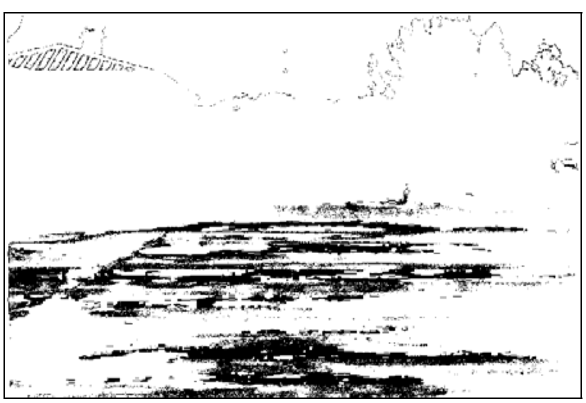
Subimage B_1

Subimage B_2



Subimage B_3

Subimage B_4



Subimage B_5

Subimage B_6

Figure 5-11: Binary subimages resulted from segmentation process

After extracting the binary subimages, the correlation of rotation angle θ and deformation angle α is plotted to determine the proper group of subimages that should be used to calculate the motion vectors. Figure 5-12 shows rotation-deformation angles plot.

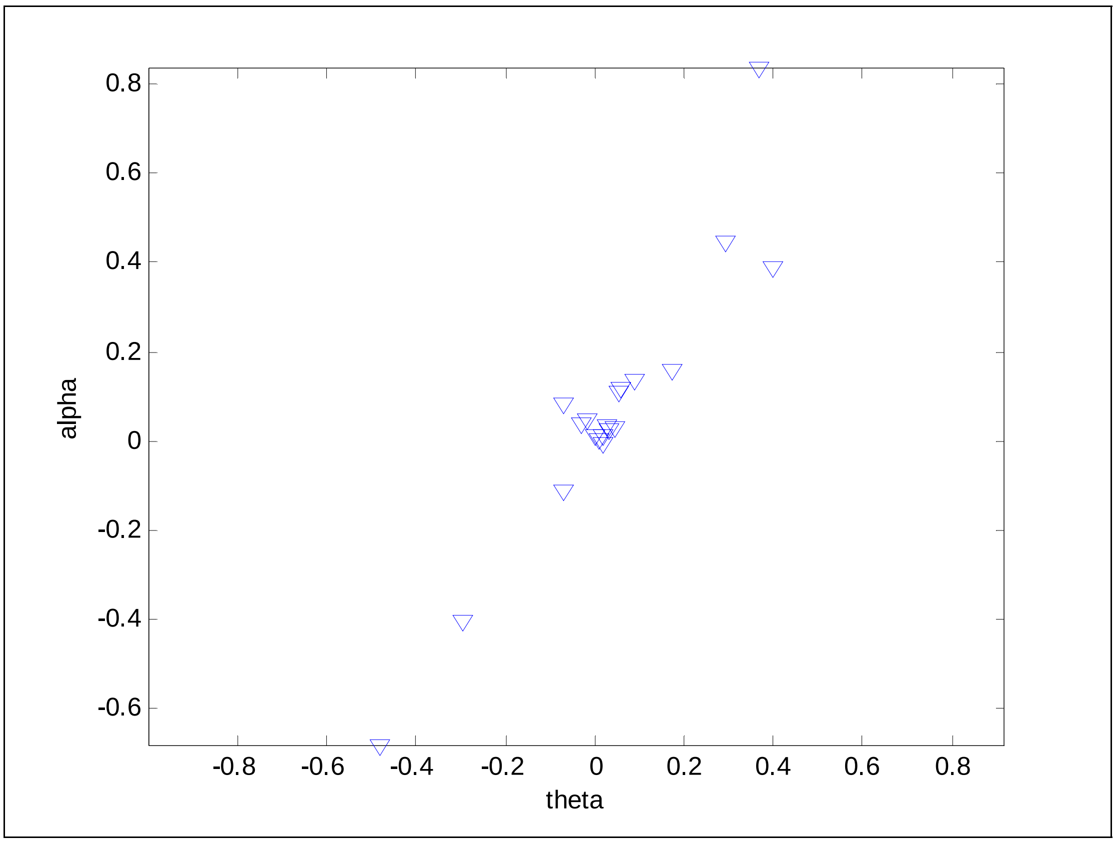


Figure 5-12: Rotation-Deformation angles

By using the approach described in section 3.1.6, we can find the best set of subimages which can be used to calculate the motion vectors. Table 5-3 shows the final estimated motion vectors.

Table 5-3: Estimated motion vectors

Parameter	Value
$\hat{\sigma}_x$	1.0009
$\hat{\sigma}_y$	1.0145
$\hat{\theta}$	0.014805
$\hat{\alpha}$	0.003251
$\Delta\hat{x}$	-1.7065
$\Delta\hat{y}$	-11.281

At this point, we can use the estimated motion vectors to reconstruct the image at time=*t+1*. Figure 5-13 show the reconstructed image.

Figure 5-13: Reconstructed image at time=t+1

The original image at time=t+1 in Figure 5-7 and the reconstructed image in Figure 5-13 shows some differences. Figure 5-14 shows these differences.

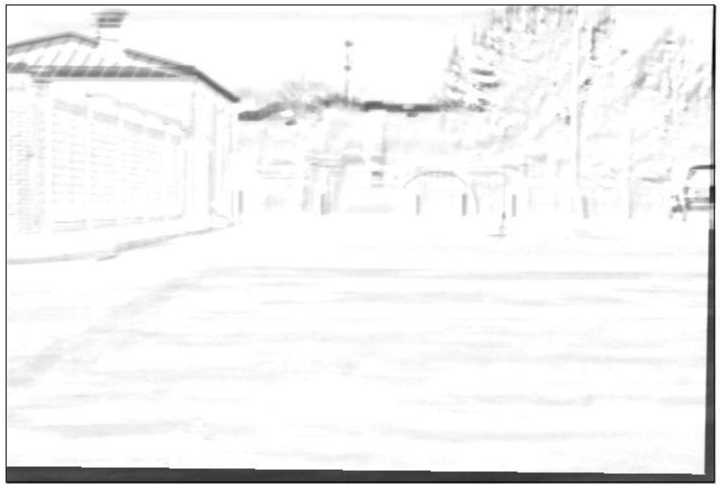


Figure 5-14: Difference between image at time=*t+1* and the constructed image

In order to get more precise motion vector estimates, the difference between the image at time=t+1 and the constructed image must be minimized.

5.2 Frequency Domain Approach to Estimate Image Translation

Following the same scheme used in the previous section to evaluate an algorithm, two types of test are conducted. We will have a simulation and an experiment for the same reason mentioned previously.

5.2.1 Simulation

For the part of simulation, we used the image shown in Figure 5-15 as an original image. This image is of size 620 x 400 pixels with a zero-padded background, making the total size of 720 x 480 pixels. A new image has been constructed by applying image translation effect to the original image. Figure 5-16 shows this constructed image.

Figure 5-15: Original image

Figure 5-16: Constructed image by shifting the original image

These two images are used as inputs to the FFT motion estimator to estimate the motion translation between the constructed image and the original image. The FFT estimator gave perfect results during the process of simulation. It could produce the exact values of translations. Table 5-4 shows this result.

Table 5-4: Estimated motion translation of the simulated images

Parameter	Value	
	Actual	Estimated
Δx	-15	-15
Δy	-40	-40

5.2.2 Experiment

As in the previous section, in this part of FFT estimator's evaluation, a more realistic sequence of images is used as inputs. Figure 5-17 and Figure 5-18 show two images that are taken from a video stream.

The translation between these two images is to be estimated using FFT estimator developed. It has been assumed that the rotation and scaling factors between these two images are very small and hence can be neglected. In this case, the image at time=$t+1$ can be totally reconstructed using the estimated translations.

Figure 5-17: Image at time=*t*

Figure 5-18: Image at time=*t+1*

We have seen that the shift parameter Δx and Δy can be computed as the slope of the phase difference between the two images. The first step here is to plot the phase difference of the two images. Figure 5-19 shows such plot.

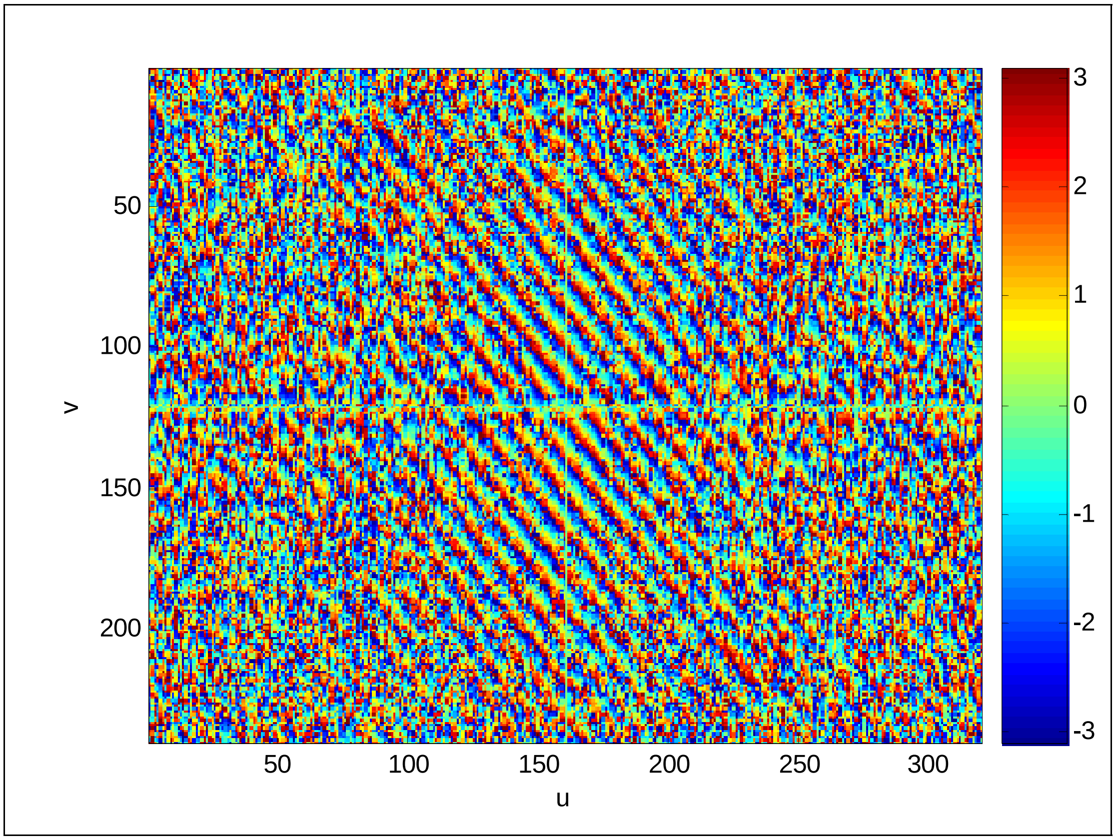


Figure 5-19: Phase difference of images at time=*t* and time=*t+1*

The parameter Δx is the shift in *u*-direction. It is computed by eliminating the effect of shift in *v*-direction. This is done by plotting the phase difference along *u*-axis and setting *v=0* as in Figure 5-20. Same method is used to get the parameter Δy.

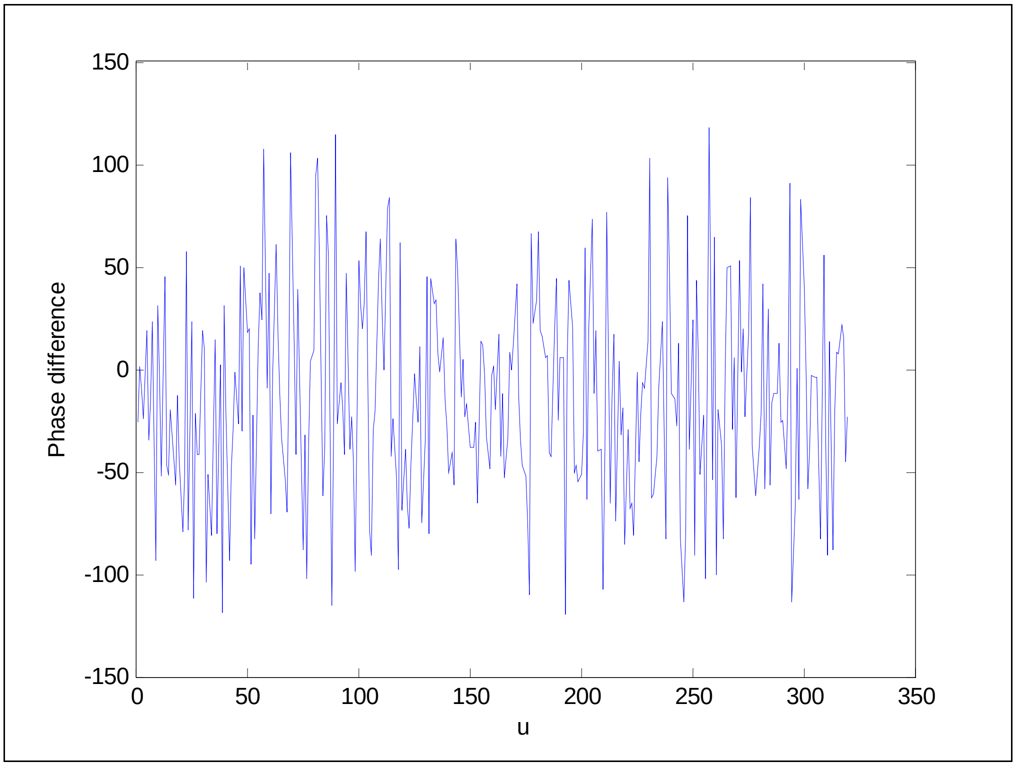


Figure 5-20: Phase difference in *u-axis*

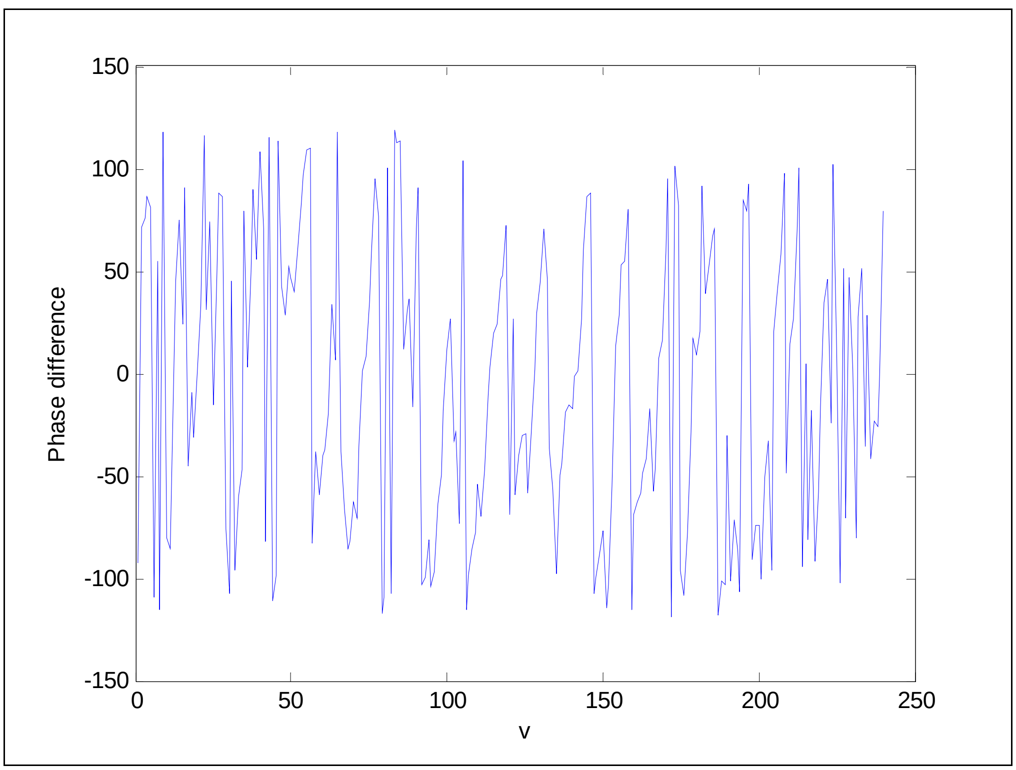


Figure 5-21: Phase difference in *v-axis*

As mentioned earlier, the slope of phase differences in each axis represent the estimated shift parameters. Table 5-5 shows the final results.

Table 5-5: FFT estimated motion translation of the simulated images

Parameter	Value
Δx	-27
Δy	13

In order to evaluate the results obtained, a new image can be constructed from image at time=*t* and the estimated motion translation then compared to the image at time=*t+1*. Figure 5-22 shows the constructed image.

Figure 5-22: Reconstructed image from translation estimated values

Figure 5-23 shows the error in difference between the constructed image and the image at time=t+1. This difference has been inverted for better viewing.

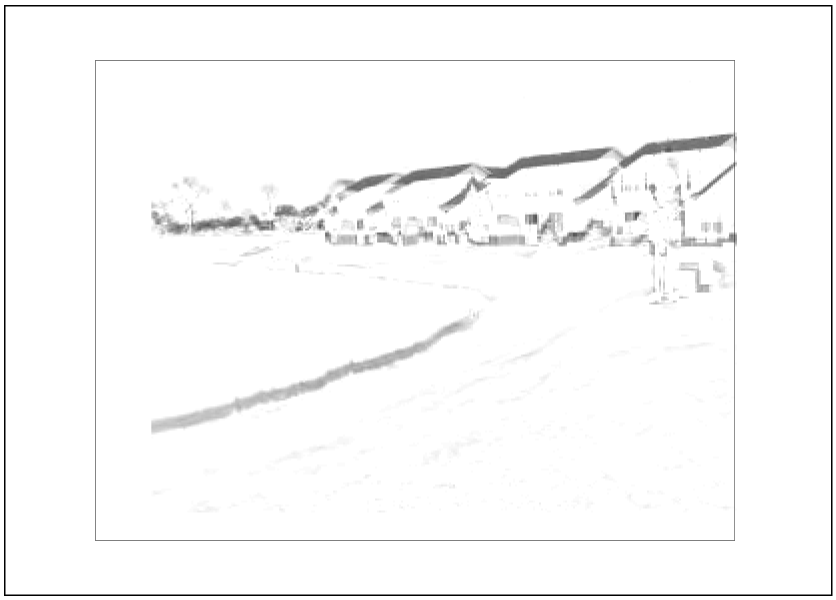


Figure 5-23: The difference between the constructed image and image at time = ***t+1***

We can notice that the estimated values are precise enough to produce a good constructed image.

Only translation was considered. The other motion vectors like rotation and scaling which are to be investigated in a future work using the Fourier Domain approach.

5.3 *Jitter Detection Algorithm*

This section presents experimental results obtained from a video sequence. The image frame sequence is of size 320 x 240 pixels and contains 336 frames in length. The frame rate is 28 frames per second.

The video is used as input to the developed jitter detection algorithm. The Motion parameter θ values of the sequence are depicted in Figure 5-24.

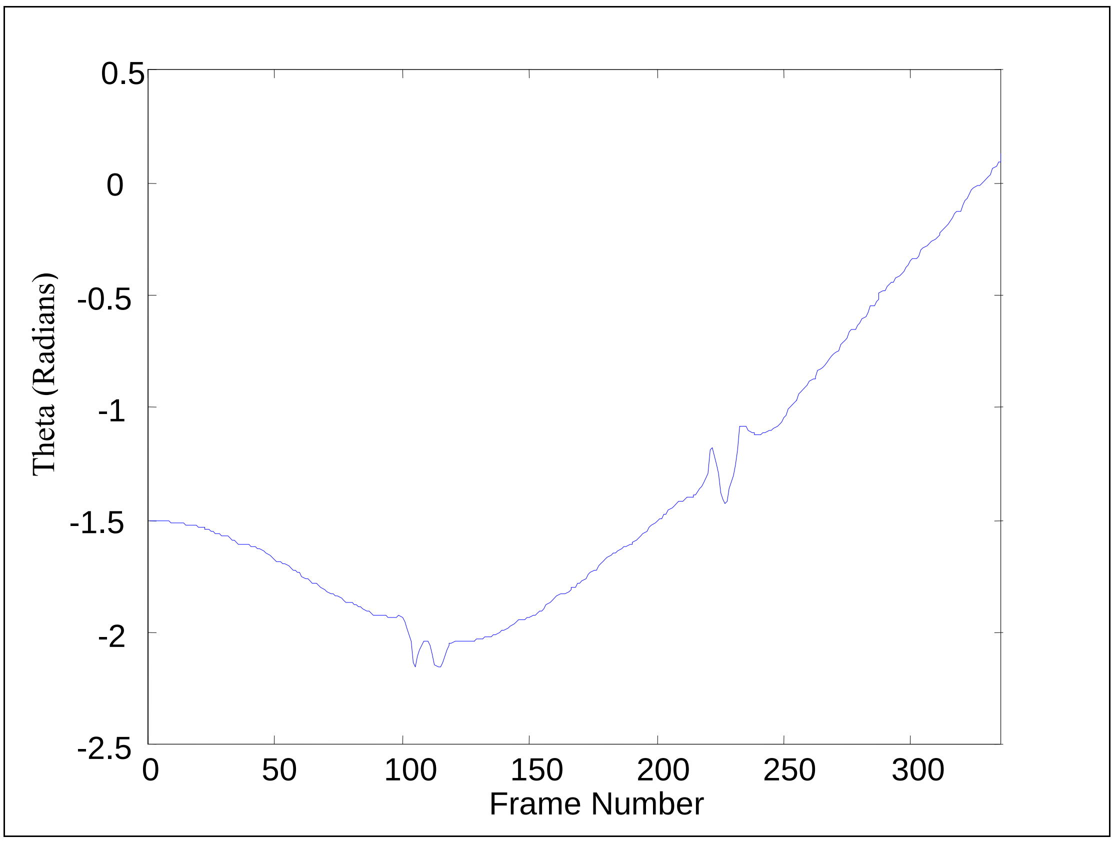


Figure 5-24: The motion parameter θ of the frames sequence

The output of convolving this signal with the multi-resolution LoG filters shown in Figure 5-25.

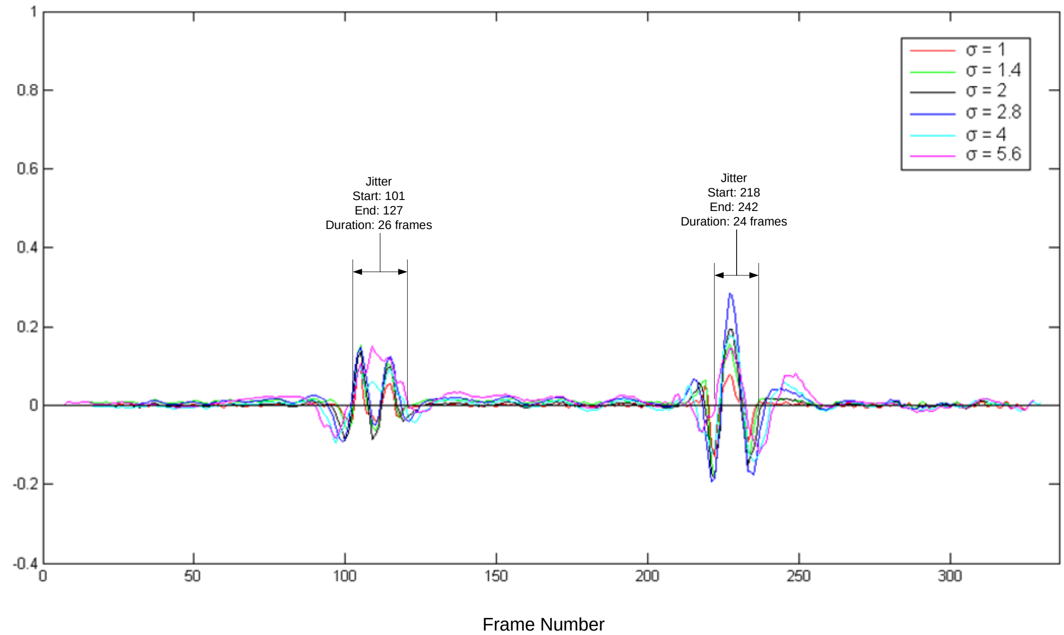


Figure 5-25: Convolution output *of θ* and

It shows the existence of two jitters. The first jitter starts at frame 101 and lasts for 26 frames. The second jitter starts at frame 218 and lasts for 24 frames. Unfortunately, still images are not the proper way to display dynamic process like video stabilization. But, the result can be shown as a sequence of still images. Figure 5-26 and Figure 5-27 show the frame images of these two video jitters.

Figure 5-26: The frames of the first jitter

Figure 5-27: The frames of the second jitter

Frame 101 must be registered with frame 126. Same thing should be done to frame 218 and frame 242. Either image registration algorithms developed in this thesis can be used to do this task. FFT approach has been selected to do such job. Table 5-6 shows the resulted estimated motion parameters.

Table 5-6: Estimated Motion Vectors for the two jitters

Parameter	Estimated Value
Δx	-38
Δy	17
Frame 101 to Frame 127	

Parameter	Estimated Value
Δx	-26
Δy	11
Frame 218 to Frame 242	

The next step now is to reconstruct images according to the estimated motion vectors. We can note that only the image translations were estimated, that is because the FFT motion estimator is selected to estimate the motion vectors. This is acceptable because of the nature of the dataset under testing. Dropped frames were substituted by a reconstructed version of the last frame in the jitter. To give impression of a smooth transition between frames, the amount of translation in these frames were gradually increased until last frame of the jitter reached. Figure 5-28 and Figure 5-29 show the final results.

Figure 5-28: The frames of the first jitter after stabilization

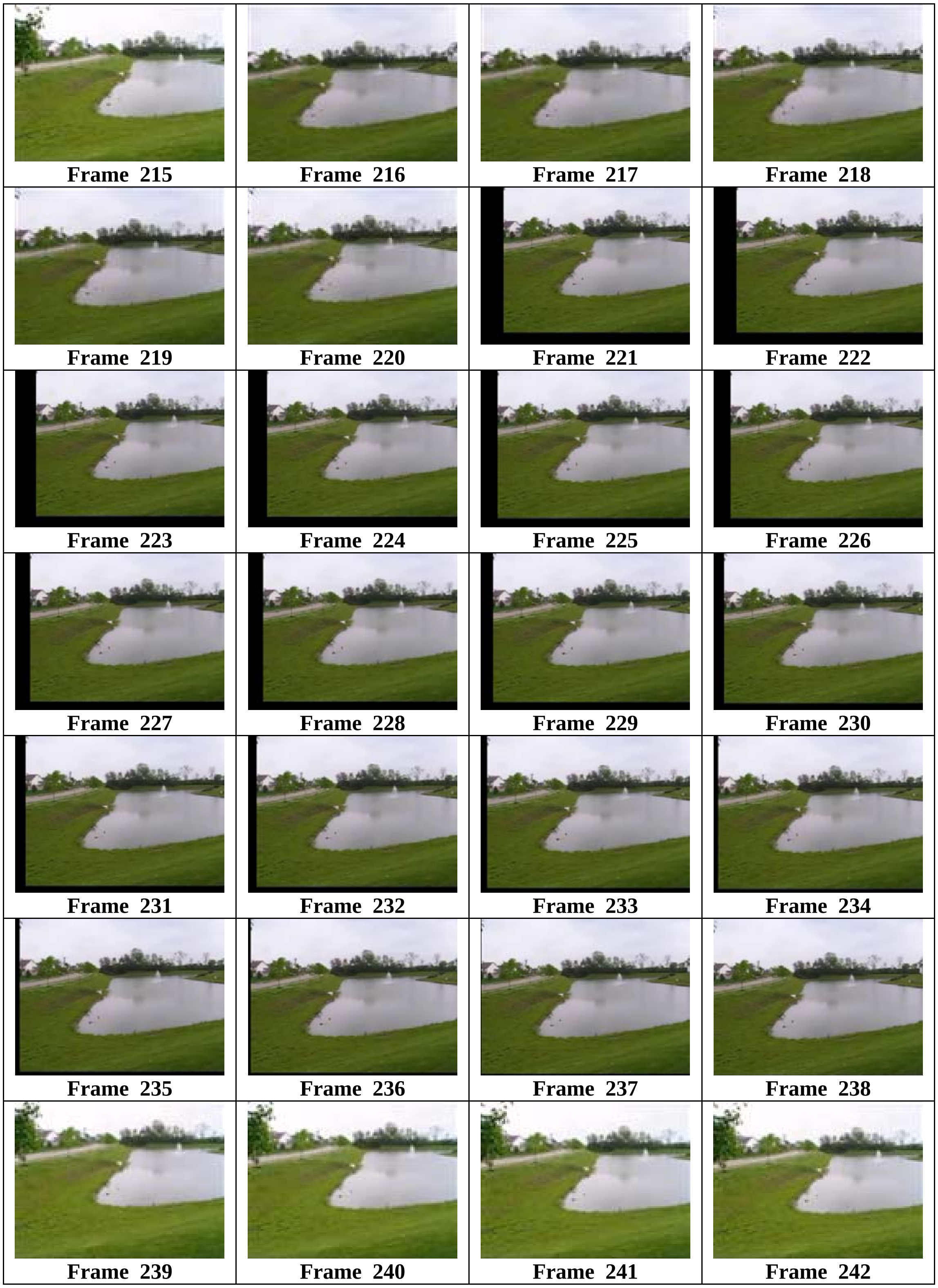


Figure 5-29: The frames of the second jitter after stabilization

As mentioned earlier in describing the jitter detection technique developed in this thesis, the missing part of the images resulted from images reconstruction, should be substituted from future frames. This should not be very difficult and it is recommended to be done as a future work for this thesis.

VI. Conclusion

6.1 Summary

In this thesis, we have presented a video stabilization technique. The underlying technology of motion-estimation, jitter detection, and image registration, have been described. We presented the formulation we used to implement real-world video stabilization algorithms and the results obtained with these algorithms. We also presented the required analysis to fully develop the approach. An Affine-based approach that tracks a small set of features was used to estimate the movement of the camera between consecutive frames. Fourier transform was also used to demonstrate translation estimation between images. The resulting translation estimate was robust and fast.

6.2 Limitations

The displayed frames are always delayed by several units of time. In general, up to five frames of delay is both adequate and acceptable. In a realistic video acquired for 30 frames/second, this delay amount to 1/6 of a second. Research studies on human perception of images indicate that this delay is not of adverse impact for routine tasks.

When the scene is comprised of ground moving targets, interpolation based on the first and the last frame as used here, will not be adequate. A more complex technique will have to be developed. It is left for future development of this effort.

6.3 Additional Remarks

We have applied an expanded version of the jitter detection and compensation on a real world UAV data. The dataset and the experimental findings are not included in this document for logistic reasons.

6.4 Future work

Current work from this thesis can be extended to improve the performance and reduce the constraints on camera motion. Possible improvements include:

1. Extending the FFT estimator to estimate rotation and other motion vectors besides translation.
2. Adding a process to the jitter detector to compensate the missing parts of images which occur due to image reconstruction.

Appendix A

This appendix lists the Matlab code developed in this research.

affinmethod.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author: Capt. Mohammed A. Alharbi
% Date  : 12-January-2006
% Description:
%   The function is the controller of the based affine
%   transformation method to register two
%   images. It takes two images and construct a third
%   image %based on the calculated motion vectors.
% Usage:
%   image3=affinmethod(image,image2)
% Input:
%   image1  - the first RGB image at time=t
%   image2  - the second RGB image at time=t+1
% Output:
%   image3  - the reconstructed image based on the
%             estimated motion vectors
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [image3]=affinemthod(image1,image2)
 clear all;
 clc;
 [theta,alpha,sigmax,sigmay,c1,c2,sizes1,sizes2]=…
 main(image1,image2);
 E=FullTree(theta,alpha);
 nMST=MinSpanTree(E);
 dMAX=max(E(:,3));
 MST=E(nMST,:);
 z=MST(:,3)<=dMAX*0.1;
 Clusters=MST(z,:);
 F=forests(Clusters(:,1:2));
 plotter(theta,alpha,Clusters);
 [properCluster,SubimagesNumbers]=findCluster2(Clusters,E);
 [alphabar,thetabar,sigmaxbar,sigmaybar,c1bar,c2bar]=…
 calcAverages(properCluster,sizes1,alpha,theta,…
 sigmax,sigmay,c1,c2)
end
```

main.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description:
%   This function takes two images as inputs and calculated
```

```
%   the motion vectors between them
% Usage:
%   [Q1,Q2,Q3,Q4,Q5,Q6,sizes1,sizes2] = main(image,image2)
% Input:
%   image1  - the first RGB image at time=t
%   image2  - the second RGB image at time=t+1
% Output:
%   Q1      - Rotation angle (theta)
%   Q2      - Deformation angle (alpha)
%   Q3      - Scaling factor in x-axis
%   Q4      - Scaling factor in y-axis
%   Q5      - Shift in x-axis
%   Q6      - Shift in y-axis
%   sizes1  - The size of binary images of the first image
%   sizes2  - The size of binary images of the second image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Q1,Q2,Q3,Q4,Q5,Q6,sizes1,sizes2] =
myfun(im1,im2)
     [M1,sizes1]=extract8features(im1,1);
     [M2,sizes2]=extract8features(im2,2);
     Qalpha=[];
     Qtheta=[];
     Qsigmax=[];
     Qsigmay=[];
     QC1=[];
     QC2=[];
     e=1;
     for i=1:6
       for j=i+1:6
         for k=j+1:6
                 X=[M1(2*i-1),M1(2*j-1),M1(2*k-…
                 1);M1(2*i),M1(2*j),M1(2*k)];
                 Xbar=[M2(2*i-1),M2(2*j-1),M2(2*k-
                 …1);M2(2*i),M2(2*j),M2(2*k)];
            Z=simequ2(X,Xbar);
 matrices
[Q,Qalpha,Qtheta,Qsigmax,Qsigmay,QC1,QC2]=trans(Z,Qalpha
,Qtheta,Qsigmax,Qsigmay,QC1,QC2);
        end;
       end;
     end;
    Q1 = Qtheta;
    Q2 = Qalpha;
    Q3 = Qsigmax;
    Q4 = Qsigmay;
    Q5 = QC1;
    Q6 = QC2;
end
```

Extract6features.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description:
%   This function extracts six binary subimages out of an
%   input image. Then, it outputs the sizes and the
%   centroid of these binary images.
```

```
%
% Usage:
%   Extract6features(image1)
% Input:
%   image1     - an RGB image.
% Output:
%   centroids  - The centroid of the six binary subimages.
%   Imsizes    - The sizes of the six binary images.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [centroids,imsizes] = features(imfile,imno)
 writeimages=1;
 showimages=0;
 close all;
 I2 = imread(imfile);
 if isrgb(I2)
  I2=rgb2gray(I2);
 end
 h = imhist(I2);
 H=cumsum(h);
 [M,N]=size(I2);
 k=M*N;
 k1=k/6;
 k2=2*k/6;
 k3=3*k/6;
 k4=4*k/6;
 k5=5*k/6;
 k6=6*k/6;
 [val,indx]=min(abs(H-k1));
 k1hat=indx;
 [val,indx]=min(abs(H-k2));
 k2hat=indx;
 [val,indx]=min(abs(H-k3));
 k3hat=indx;
 [val,indx]=min(abs(H-k4));
 k4hat=indx;
 [val,indx]=min(abs(H-k5));
 k5hat=indx;
 [val,indx]=min(abs(H-k6));
 k6hat=indx;
 % The first binary image
 B1=I2;
 B1=double(I2<k1hat);
 B1=logical(B1);
 B1=~B1;
 imsizes(1)=sum(sum(B1));
 % The second binary image
 B2=double(I2>k1hat&I2<k2hat);
 B2=logical(B2);
 B2=~B2;
 imsizes(2)=sum(sum(B2));
 % The third binary image
 B3=double(I2>k2hat&I2<k3hat);
 B3=logical(B3);
 B3=~B3;
 imsizes(3)=sum(sum(B3));
 % The fourth binary image
 B4=double(I2>k3hat&I2<k4hat);
```

```
 B4=logical(B4);
 B4=~B4;
 imsizes(4)=sum(sum(B4));
 % The fifth binary image
 B5=double(I2>k4hat&I2<k5hat);
 B5=logical(B5);
 B5=~B5;
 imsizes(5)=sum(sum(B5));
 % The sixth binary image
 B6=double(I2>k5hat&I2<k6hat);
 B6=logical(B6);
 B6=~B6;
 imsizes(6)=sum(sum(B6));
 if imno==1
  filename1='images\B11.bmp';
  filename2='images\B21.bmp';
  filename3='images\B31.bmp';
  filename4='images\B41.bmp';
  filename5='images\B51.bmp';
  filename6='images\B61.bmp';
 else
  filename1='images\B12.bmp';
  filename2='images\B22.bmp';
  filename3='images\B32.bmp';
  filename4='images\B42.bmp';
  filename5='images\B52.bmp';
  filename6='images\B62.bmp';
 end;
if writeimages
 imwrite(B1,filename1);
 imwrite(B2,filename2);
 imwrite(B3,filename3);
 imwrite(B4,filename4);
 imwrite(B5,filename5);
 imwrite(B6,filename6);
end;
a=centroid(filename1);
MM(1,1)=a(1);
MM(2,1)=a(2);
a=centroid(filename2);
MM(1,2)=a(1);
MM(2,2)=a(2);
a=centroid(filename3);
MM(1,3)=a(1);
MM(2,3)=a(2);
a=centroid(filename4);
MM(1,4)=a(1);
MM(2,4)=a(2);
a=centroid(filename5);
MM(1,5)=a(1);
MM(2,5)=a(2);
a=centroid(filename6);
MM(1,6)=a(1);
MM(2,6)=a(2);
if showimages
 figure;
 imshow(B1);figure;
 imshow(B2);figure;
```

```
 imshow(B3);figure;
 imshow(B4);figure;
 imshow(B5);figure;
 imshow(B6);
 end;
 centroids = MM;
end
```

centroid.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description:
%   This function calculated the centroid of a binary
%   image.
% Usage:
%   B_centroid = centroid(imfile)
% Input:
%   imfile     - The file name of the binary image.
% Output:
%   B_centroid – The calculated centroid.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function B_centroid = centro(imfile)
 im = imread(imfile);
 [rows,cols] = size(im);
 x = ones(rows,1)*[1:cols];
 y = [1:rows]'*ones(1,cols
 area = sum(sum(im));
 meanx = sum(sum(double(im).*x))/area;
 meany = sum(sum(double(im).*y))/area;
 B_centroid = [meanx,meany];
end
```

simeq2.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description:
%   This function solves three simultaneous equations.
% Usage:
%   result = simeq2(co1,co2)
% Input:
%   co1    -  Coefficients of the first set of equations
%   co2    -  Coefficients of the second set of equations
% Output:
%   result -  The result of the solved equations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function result=myfunc(co1,co2)
 x11=co1(1,1);
 x12=co1(1,2);
 x13=co1(1,3);
 y11=co1(2,1);
 y12=co1(2,2);
 y13=co1(2,3);
 x11bar=co2(1,1);
 x12bar=co2(1,2);
 x13bar=co2(1,3);
 y11bar=co2(2,1);
```

```
 y12bar=co2(2,2);
 y13bar=co2(2,3);
 XX =[x11 y11 0 0 1 0;
 0 0 x11 y11 0 1;
 x12 y12 0 0 1 0;
 0 0 x12 y12 0 1;
 x13 y13 0 0 1 0;
 0 0 x13 y13 0 1];
 b=[x11bar ;y11bar ;x12bar; y12bar ;x13bar ;y13bar ];
 result = XX\b;
end
```

trans.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description:
%   This function calculates the values of the scaling in
%   x-axis and y-axis. Also, it calculates the values of
%   the rotation angle and the deformation angle. Then, it
%   reorganizes shift values and put them in matrices. It
%   produces all vectors of motion as arrays.
% Usage:
%   [Qalphas,Qthetas,Qsigmaxs,Qsigmays,QC1s,QC2s] =
%    trans(M,Qalpha,Qtheta,Qsigmax,Qsigmay,QC1,QC2)
% Input:
%    Qalpha  - Deformation angle.
%    Qtheta  - Rotation angle.
%    Qsigmax – Scaling factor in x-axis
%    Qsigmay – Scaling factor in y-axis
%    QC1     - Shift in x-axis
%    QC2     - Shift in y-axis
% Output:
%    Qalphas  - Array of deformation angles.
%    Qthetas  - Array of rotation angles.
%    Qsigmaxs – Array of Scaling factors in x-axis
%    Qsigmays – Array of Scaling factors in y-axis
%    QC1s     - Array of Shifts in x-axis
%    QC2s     - Array of Shifts in y-axis
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Qalphas,Qthetas,Qsigmaxs,Qsigmays,QC1s,QC2s] =
 myfun(M,Qalpha,Qtheta,Qsigmax,Qsigmay,QC1,QC2)
 sigmax=sqrt(M(1)+M(2));
 sigmay=sqrt(M(3)+M(4));
 C1=M(5);
 C2=M(6);
 alphaminustheta=atan2(M(2),M(1));
 alphaplustheta=atan2(M(3),M(4));
 alpha=(alphaminustheta+alphaplustheta)/2;
 theta=alphaplustheta-alpha;
 Qalphas=[Qalphas ; alpha];
 Qthetas=[Qthetas ; theta];
 Qsigmaxs=[Qsigmaxs; sigmax];
 Qsigmays=[Qsigmays; sigmay];
 QC1s=[QC1s; C1];
 QC2s=[QC2s; C2];
end
```

Fulltree.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description:
%   This function creates a full tree graph of a given two
%   sets of points.
% Usage:
%   [Q]=Fulltree(X,Y)
% Input:
%   X  - The x coordinates of the points to be converted in
%        a fully tree graph.
%   Y  - The Y coordinates of the points.
% Output:
%   Q  - Edges of the created full tree graph.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Q] = myfun(X,Y)
z=1;
for i=1:19
  for j=i+1:20
    Q(z,:)=[i j sqrt((X(i)-X(j))^2 + (Y(i)-Y(j))^2)];
    z=z+1;
  end
end
```

MinSpanTree.m

```
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 % This function solves the minimal spanning tree problem
 % for a connected graph.
 % Input parameter:
 %   E(m,2) or (m,3) - the edges of graph and their weight;
 %     1st and 2nd elements of each row is numbers of
 %     vertexes;
 %     3rd elements of each row is weight of edge;
 %     m - number of edges.
 %     If we set the array E(m,2), then all weights is 1.
 % Output parameter:
 %   nMST - the list of the numbers of edges included
 %     in the minimal (weighted) spanning tree in the
 %including order.
 % Uses the greedy algorithm.
 % Author: Sergiy Iglin
 % e-mail: iglin@kpi.kharkov.ua
 % or: siglin@yandex.ru
 % personal page: http://iglin.exponenta.ru
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function nMST=MinSpanTree(E)
 [m,n,E] = Evalidation(E);
 En=[(1:m)',E];
 [Emin,nMST]=min(En(:,4)); nVer=[En(nMST,2:3)];
 En=En(setdiff([1:m],nMST),:);
 while length(nVer)<n,
  Encurr=[];
  for k=1:size(En,1),
   if sum(ismember(En(k,2:3),nVer))==1,
    Encurr=[Encurr;En(k,:)];
   end
  end
 if isempty(Encurr),
  error('The graph is not connected!')
```

```
 end
 [Emin,imin]=min(Encurr(:,4));
 nEdge=Encurr(imin,1);
 nMST=[nMST,nEdge];
 nVer=unique([nVer,Encurr(imin,2:3)]);
 En=En(setdiff([1:size(En,1)],find(En(:,1)==nEdge)),:);
 end
 return
end
```

forests.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description: This algorithm builds forests out of a group
% of nodes using Kruska’s algorithm.
%
% Usage:
%        [Q]=forests(A).
% Input:
%        A  - A group on nodes.
% Output:
%        Q  - A graph consist of forest.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Q]=myfun(A)
B=[];
k=max(A(:));
M = sparse(A(:,1),A(:,2),1,k,k);
M=M+M';
M1 = zeros(size(M));
flag = 1;
while flag
 M1=double((M+M*M)~=0);
   if isequal(M,M1)
     flag = 0;
   end
 M=M1;
end
M=unique(M,'rows');
M(all(~M,2),:)=[];
for (i=1:size(M,1))
 a=find(M(i,:));
  for (j=1:size(A,1))
   if (length(intersect(a,A(j,:))) > 0)
    B=[B; A(j,:)];
   end
  end
 B=[B; [0 0]];
end
Q=B;
```

Plotter.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description:
%   This function plots a set of nodes in a graph.
% Usage:
%   Plotter(X,Y,zo)
% Input:
%   X   -  The x-coordinates of the nodes
```

```
% Y - The y-coordinates of the nodes
% zo - Distances between the nodes
% Output:
% none
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function myplot(X,Y,zo)
 plot(X,Y,'*');
 axis equal
 ylim=[-2 2.5];
 ylabel('alpha');
 xlabel('theta');
 t=[1:20]';
 T=num2str(t);
 text(X,Y,T);
 for (i=1:size(zo,1))
  s=zo(i,1); e=zo(i,2);
  line([X(s); X(e)], [Y(s) ;Y(e)])
 end
```

findcluster2.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description:
%   The function finds the best group of nodes (binary
%   subimages) that can be used to determine the motion
%   vectors.
% Usage:
%   [m,subno] = findcluster2(n,E)
% Input:
%   n      -  The nodes (binary subimages) in the graph
%   E      -  The cluster contains these nodes
% Output:
%   m      -  The nodes (binary subimages) that should be
%             used to determine motion vectors.
%   subno  -  The index of these nodes.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%The following function takes clusters matrix
%It generates the data structure necssary to determine the
% best cluster to use
function [m,subno]=myfun(n,E)
 n=sortrows(n,3);
 maxdistance=0.3;
 m=[];
 subno=[];
 subnocell=cell(10,1);
 q=[];
 for i=1:6
  for j=i+1:6
   for k=j+1:6
     q=[q; i,j,k];
   end
  end
 end
 M=cell(10,1);
 Mi=1;
 current=n(1,:);
 M{1}=current;
```

```
  subnocell{1}=union(q(current(1),:),q(current(2),:));
  if size(subnocell{1},2)==5
   m=M{1};
   subno=subnocell{1};
   return
  end
  for i=2:size(n,1)
   notinserted=true;
   current=n(i,:);
   j=1;
   while j<=Mi & notinserted
    if  find(M{j}(:,1:2)==current(1)|…
      M{j}(:,1:2)==current(2))
      MM=[];
      MM=[M{j}; current];
      if clusthrecheck(E,MM)<=maxdistance
        M{j}=MM;
        subnocell{j}=union(subnocell{j},…
        union(q(current(1),:),q(curent(2),:)));
         if size(subnocell{j},2)==5
           m=M{j};
           subno=subnocell{j};
           return
         end
      end
     notinserted=false;
    end
   j=j+1;
   end
  if notinserted
  Mi=Mi+1;
  M{Mi}=current;
  subnocell{Mi}=union(q(current(1),:),q(current(2),:));
   if size(subnocell{Mi},2)==5
     m=M{Mi};
     subno=subnocell{Mi};
     return
   end
  end
end
```

calcaverages.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description:
%   This function contains the average motion vector values
%   for all selected binary subimages.
% Usage:
%   sigmaybar,c1bar,c2bar]=...
%   calcaverages(propCluster,sizes,alpha,…
%   theta,sigmax,sigmay,c1,c2)
% Input:
%   propCluster,
%   sizes   -   The total size of each image.
%   alpha   -   The deformation angle of each pair of
%               images.
%   theta   -   The rotation angle of each pair of images.
%   sigmax  -   The Scaling factor in the x-axis.
%   sigmay  -   The scaling factor in the y-axis.
```

```
%   c1       -   The shift in x-axis.
%   c2          - The shift in the y-axis
% Output:
%   propCluster – The indices of the nodes that contains
                  the best binary images that can be used
                  to calculate motion vectors.
%   sizes       - The total sizes of these binary images.
%   alpha       - The averaged deformation angles of these
%                 images.
%   theta       - The averaged rotation angles of these
%                 images.
%   sigmax      - The averaged scaling factors in the x-
%                 axes.
%   sigmay      - The averaged scaling factors in the y-
%                 axis
%   c1          - The averaged shift in the x-axis
%   c2          - The averaged shift in the y-axis
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [alphabar,thetabar,sigmaxbar,…
 sigmaybar,c1bar,c2bar]=...
 myfun(propCluster,sizes,alpha,theta,sigmax,sigmay,c1,c2)
 q=[];
 w=[];
 for i=1:6
  for j=i+1:6
   for k=j+1:6
    q=[q; i,j,k];
   end
  end
 end
 pc=unique(propCluster(:,1:2));
 si=q(pc,:);
 subsizes=sizes(si);
 for i=1:size(subsizes,1)
  w(i)=min(subsizes(i,:));
 end
 w=w';
 alphabar=sum(w.*alpha(pc))/sum(w);
 thetabar=sum(w.*theta(pc))/sum(w);
 sigmaxbar=sum(w.*sigmax(pc))/sum(w);
 sigmaybar=sum(w.*sigmay(pc))/sum(w);
 c1bar=sum(w.*c1(pc))/sum(w);
 c2bar=sum(w.*c2(pc))/sum(w);
end
```

Evalidation.m

```
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 % The validation of array E - auxiliary function for
 % GrTheory Toolbox.
 % Author: Sergiy Iglin
 % e-mail: iglin@kpi.kharkov.ua
 % or: siglin@yandex.ru
 % personal page: http://iglin.exponenta.ru
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 function [m,n,newE] = Evalidation(E)
 if ~isnumeric(E),
```

```
  error('The array E must be numeric!')
 end
 se=size(E);
 if length(se)~=2,
  error('The array E must be 2D!')
 end
 if (se(2)<2),
  error('The array E must have 2 or 3 columns!'),
 end
 if ~all(all(E(:,1:2)>0)),
  error('1st and 2nd columns of the array E must be
  positive!')
 end
 if ~all(all((E(:,1:2)==round(E(:,1:2))))),
  error('1st and 2nd columns of the array E must be…
  integer!')
 end
 m=se(1);
 if se(2)<3,
  E(:,3)=1;
 end
 newE=E(:,1:3);
 n=max(max(newE(:,1:2)));
 return
end
```

clusthrecheck.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description:
%   This function check the maximum threshold between all
%   nodes and it outputs the maximum value.
% Usage:
%   maxdis = clustercheck(E,a)
% Input:
%   E      -    The clusters to be checked.
%   a      -    The distances between the nodes within
%               clusters.
% Output:
%   maxdis -    The maximum distance between the nodes in a
%               cluster
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 %This function check the maximum distance (threshold)
 %between all nodes in the cluster a
 function maxdis=myfun(E,a)
 q=unique(a(:,1:2));
 w=[];
 for i=1:size(q,1)
  for j=i+1:size(q,1)
   w=[w; q(i), q(j)];
  end
 end
 [tf,loc] = ismember(w,E(:,1:2),'rows');
 maxdis=max(E(loc,3));
end
```

moveitem.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description:
%    This function removes a node from a cluster and puts
%    it to another cluster.
%
% Usage:
%    [src,dist]=moveitem(src,dist,vals)
% Input:
%    src  :  The source value.
%    dist :  The distination.
%    vals :  Values of all nodes in the clusters.
% Output:
%    src  :  The source value after removing.
%    dist :  The distination after removing.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [src,dist]=moveitem(src,dist,vals)
 [tf,loc] = ismember(vals,src(:,:),'rows') ;
 src=src(setdiff(1:size(src,1),loc),:);
 dist=[dist; vals];
end
```

removerelated.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description:
%   Remove loops in clusters.
% Usage:
%   [m,templist]=removerealted(m,templist,val)
% Input:
%   m        -  The nodes in the cluster.
%   templist –  A temporary list used for switching nodes
%   val      -  The values of these nodes.
% Output:
%   m        -  The nodes in the cluster after removing
%               loops
%   templist - A temporary list used for switching nodes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [m,templist]=removerealted(m,templist,val)
 [r,c]=find(m(:,1:2)==val(1) | m(:,1:2)==val(2));
 d=m(r,:);
 [m,templist]=moveitem(m,templist,d);
end
```

sortcluster.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description:
%   This function sort the clusters according to their
%   edges.
% Usage:
%   [c1]=sortcluster(c]
% Input:
%   c       -  Unsorted cluster.
% Output:
%   c1      -  Sorted cluster.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [c1]=ext(c)
 c1=[];
 m=[];
 h=1;
 i=1;
 while i<size(c,1)
   while c(i,1)~=0
    i=i+1;
   end
   temp=c(h:i-1,:);
   temp=sortmat(temp)
   m=[m;temp;0 0 0];
   i=i+1;
   h=i;
  end
  c1=m;
end
```

con4.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description:
%   Given a reference image and motion vectors, this
%   function constructs a new image based on the reference
%   image and motion vectors.
% Usage:
%   function [newImage]=con4(image1,theta,alpha,sx,…
%   sy,c1,c2)
% Input:
%   image1    - The reference image.
%   theta     - The rotation angle.
%   alpha     - The deformation angle.
%   sx        - The scaling factor in x-axis.
%   sy        - The scaling factor in y-axis.
%   c1        - The shift in x-axis.
%   c2        - The shift in y-axis.
% Output:
%   newImage  - The new constructed image base on the
%               reference image and the motion vectors.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [newImage]=myfun(image1,theta,alpha,sx,sy,c1,c2)
 close all;
 A=[sx*cos(alpha-theta) sx*sin(alpha-theta);
 sy*sin(alpha+theta) sy*cos(alpha+theta)];
 figure;
 imshow(image1);
 for x=1:size(image1,2)
  for y=1:size(image1,1)
    xpos=x*A(1)+y*A(2)+c1;
    ypos=x*A(3)+y*A(4)+c2;
    fx = floor(xpos);
    fy = floor(ypos);
    apha=xpos-floor(xpos);
    beta=ypos-floor(ypos);
    if ~(fx+1>size(f,2) | fy+1>size(f,1) | fx<1 | fy<1)
```

```
          newImage(y,x) = ...
          image(fy, fx) *  (1 - apha)*(1 - beta)+...
          image(fy, fx+1) * apha*(1-beta) +...
          image(fy+1, fx) * (1-apha)*beta +...
          image(fy+1, fx+1) *apha*beta ;
        elseif
         (fx>0 && fy>0) && (fx==size(image1,2) ||…
          fy==size(image1,1))
         && ~(fx>size(image1,2) || fy>size(image1,1))
          newImage(y,x) = image(fy, fx);
        else
          newImage(y,x) = 0;
        end
      end
    end
    figure;
    imshow(newImage,[])
  end
```

de5.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description:
%   This function takes two images and plot the phase
%   difference of FFT of them.
% Usage:
%   [m,n]=de5(im1,im2)
% Input:
%   im1    -  The first image.
%   im2    -  The second image.
% Output:
%   m      -  The u values of the phase difference.
%   n      -  The v values of the phase difference.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [m,n]=myfun(im1,im2);
 close all
 F1=fft2(im1);
 F2=fft2(im2);
 F1=fftshift(F1);
 F2=fftshift(F2);
 P=F1./F2;
 Pph=angle(P);
 figure;
 imshow(Pph);
 s=size(im1,1);;
 z=s/2;
 Psurf=Pph.*(s/(2*pi));
 PP=Psurf(z,:);
 PP=PP';
 figure; plot(PP);
 QQ=Psurf(:,z);
 figure; plot(QQ);
 jp=z;
 while PP(jp+1)-PP(jp)<max(PP)*0.3 && jp<s
  jp=jp+1;
 end
 ip=z;
```

```
  while PP(ip)-PP(ip-1)<max(PP)*0.3 && ip>2
   ip=ip-1;
  end
  jp=jp-1;
  P1=PP(ip:jp);
  m=PP;
  p=polyfit([ip:jp]',P1,1)
  f = polyval(p,[ip:jp]);
  figure; plot([ip:jp],P1,[ip:jp],f,'-r');
  jq=z;
  while QQ(jq+1)-QQ(jq)<max(QQ)*0.25 && jq<s
   jq=jq+1;
  end
  iq=z;
  while QQ(iq)-QQ(iq-1)<max(QQ)*0.25 && jq>2
   iq=iq-1;
  end
  jq=jq-1;
  Q1=QQ(iq:jq);
  n=QQ;
  p=polyfit([iq:jq]',Q1,1)
  f = polyval(p,[iq:jq]);
  figure; plot([iq:jq],Q1,[iq:jq],f,'-r');
 end
```

conv1d.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description:
%   This function takes two signals and plots the
%   convolution of them.
% Usage:
%   conv1d(mask,q)
% Input:
%   mask  - The first signal (which is the digital filter)
%   q     - The signal intended to be convoluted.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function myfun(mask,q)
 h1=mask(1,2:7);
 ds=[1 2 4];
 sum1=0;
 z1=[];z2=[];
 co=['r' ;'g'; 'k' ;'b' ;'c'; 'm'];
 cp=1;
 for (s=1:3)
  for (n=6*ds(s):(336-ds(s)*6))
   for (k=2:6)
    sum1=sum1+(q(n+(k-1)*ds(s))+q(n-(k-1)*ds(s)))*h1(k);
   end
  z1(n)=h1(1)*q(n)+sum1;
  sum1=0;
 end
 h2=mask(2,2:9);
 sum1=0;
 for (n=8*ds(s):(336-ds(s)*8))
  for (k=2:6)
   sum1=sum1+(q(n+(k-1)*ds(s))+q(n-(k-1)*ds(s)))*h2(k);
 end
```

```
 z2(n)=h2(1)*q(n)+sum1;
 sum1=0;
 end
 plot(z1,co(cp));
 hold on;
 cp=cp+1;
 plot(z2,co(cp));
 cp=cp+1;
 end
 hold on;
 plot([0 400],[0 0],'k')
end
```

fullrangephi.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description:
%   This function takes the values of rotation angles and
%   out them as full range.
% Usage:
%   [q]=fullrangephi(phi,FramesNumber)
% Input:
%   phi          - The rotation angel.
%   FramesNumber – The number of the frame which the phi
%                  belong to.
% Output:
%   q            - A full-ranged alpha.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [q]=myfun(phi,FramesNumber)
 q=[];
 q(1)=phi(1);
 cdif=0;
for (i=2:FramesNumber)
 dif=phi(i)-phi(i-1);
  if dif>2
    cdif=cdif-2*pi;
  end
  if dif<-2
   cdif=cdif+2*pi;
  end
 q(i)=phi(i)+cdif;
end;
```

jd.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description:
%  This function takes plots the zero-crossing values that
%  yield from convolution of the rotation angle and the
%  mask.
% Usage:
%  jd(moments,mask)
% Input:
%   moments – The moments of the processed frames.
    mask    - The digital filter used in the convolution
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function jd(moments,mask)
 phi=atan2(moments(:,2),moments(:,3));
```

```
 q=full_range_phi(phi,size(moments,1));
 plot(q);
 conv1d(mask,q)
end
```

get_moments.m

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description:
%   This function takes an image and calculates the moments
%   of this image.
% Usage:
%   [weight,S00,M00,M01,M10,M11,M20,M02]=get_moments(img)
% Input:
%   img    - The input image.
% Output:
%   weight – The weight used to calculate the moments.
%   M00    - The M00 moment.
%   M01    - The M01 moment.
%   M10    - The M10 moment.
%   M11    - The M11 moment.
%   M20    - The M20 moment.
%   M02    - The M02 moment.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [weight,S00,M00,M01,M10,M11,M20,M02]=myfun(img)
 ii=320;
 jj=240;
 w_sigma_len=105;
 for (i=1:ii)
  for (j=1:jj)
   u(j,i)=((i-ii/2)^2+(j-jj/2)^2)/(2*w_sigma_len^2);
  end;
 end;
 weight=255*exp(-1*u.^2);
 weight=u;
 im=double(im);
 S00=0;
 S10=0;
 S01=0;
 S11=0;
 S20=0;
 S02=0;
 for (x=1:ii)
  for(y=1:jj)
   cx=x-ii/2;
   cy=y-jj/2;
   S00=S00+im(y,x)*weight(y,x);
   S10=S10+cx * im(y,x) * weight(y,x);
   S01=S01+cy * im(y,x) * weight(y,x);
   S11=S11+cx*cy*im(y,x)*weight(y,x);
   S20=S20+cx*cx *im(y,x)*weight(y,x);
   S02=S02+cy*cy *im(y,x)*weight(y,x);
  end;
 end;
 M00=S00;
 M10=S10;
```

```
 M01=S01;
 M11=S11;
 M20=S20;
 M02=S02;
end
```

Createmovie.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description:
%   This function create a movie out of a set of images.
% Usage:
%   [avifilename]=Createmovie(A,frames_location)
% Input:
%      A               - Set of images.
%      frames_location - The location of the images.
% Output:
%      avifilename     - The file name of the movie.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [avifilename]=myfunc(A,frames_location)
 aviobj = avifile(avifilename,'fps',28);
 icon=[];
 num=size(A,2);
 for (j=1:num)
  i=A(j);
  if i<10
  image1=strcat(frames_location,…
  'frame00',int2str(i),'.bmp')
  elseif i<100
    image1=strcat(frames_location,'frame0',…
    int2str(i),'.bmp')
  else
   image1=strcat(frames_location,'frame',…
   int2str(i),'.bmp')
  end
 if ~ismember(i,[115:126,230:239])
  im=imread(image1);
  imshow(im);
  frame=getframe;
  aviobj = addframe(aviobj,frame);
 end
end
aviobj = close(aviobj);
end
```

get_angle_dif.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description:
%   This function calculated the rotation angles difference
    between two images.
% Usage:
%   [ang_dif]=get_angle_dif(im1,im2)
% Input:
%   im1     - The first image.
%   im2     - The second image.
% Output:
```

```
%   ang_dif – The difference in rotation angles between the
%   two images.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ang_dif]=myfunc(im1,im2)
 [M00_1,M01_1,M10_1,M11_1,M20_1,M02_1]=get_moments(im1);
 [M00_2,M01_2,M10_2,M11_2,M20_2,M02_2]=get_moments(im2);
 F1=[M20_1 M11_1; M11_1 M02_1];
 F2=[M20_2 M11_2; M11_2 M02_2];
 [V1,D1] = eig(F1);
 [V2,D2] = eig(F2);
 ang1=atan2(V1(2,1),V1(1,1));
 ang2=atan2(V2(2,1),V2(1,1));
 ang_dif=ang2-ang1;
end
```

vsplitter2.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description:
%   This function split a movie into images.
% Usage:
%   [im1]=vsplitter2(moviefilename,destinationfilename)
% Input:
%   moviefilename        - The movie’s file name.
%   destinationfilename  - The name of the extracted
%                          images.
% Output:
%   im1                  - The image file name.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [im1]=myfun(moviefilename,destinationfilename)
 mov=aviread(moviefilename);
 for i=1:size(mov,2)
  im1=mov(i).cdata;
  if i<10
   filename=strcat(destinationfilename,int2str(i),'.bmp')
   elseif i<100
    filename=strcat(destinationfilename,int2str(i),'.bmp')
    elseif i>=100
     filename=strcat(destinationfilename,int2str(i),'.bmp')
    end
    imwrite(im1,filename);
   end
   end
  end
end
```

Bibliography

[1] Q. Zheng and R. Chellappa. A computational vision approach to image registration. IEEE Trans. Image processing, 2:311-326, 1993

[2] C. Morimoto, R. Chellappa, ”Fast electronic digital image stabilization”, Proceedings of the 13th International Conference on Pattern Recognition, vol. 3,, pp.284-288, 25-29 August 1996

[3] JY. Chang, WF. Hu, MH Chang and BS Chang, “Digital Image Translational and Rotational Motion Stabilization using Optical Flow Technique”, IEEE Trans. on Consumer Electronics, Vol. 48, No. 1, pp. 108-115, 2002.

[4] C. Morimato and R. Chellappa, “Evaluation of Image Stabilization Algorithms”, Proc. of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, Vol. 5, pp. 2789-2792, 1998.

[5] A. Jain and R. Dubes , “Algorithms for clustering data”, Prentice Hall, 1988.

[6] A. Engelsberg and G. Schmidt, “A comparative review of digital image stabilizing algorithms for mobile video communications,” IEEE Trans. On Consumer Electronics, vol. 45, no. 3, pp. 591-597, Aug, 1999.

[7] M. Oshima; T. Hayashi; S. Fujioka; T. Inaji; H. Mitani; J. Kajino; K. Ikeda; and K. Komoda, “VHS camcorder with electronic image stabilizer”, IEEE Trans. on Consumer Electronics, Vol. 35, No. 4, pp. 749 – 758, Nov. 1989

[8] J.P. Derutin; F. Dias; L. Damez; and N. Allezard, “SIMD, SMP and MIMD-DM parallel approaches for real-time 2D image stabilization”, Computer Architecture for Machine Perception, 2005. CAMP 2005. Proceedings. Seventh International Workshop on 4-6 July 2005 Page(s):73 – 80.

[9] C. Morimoto, R. Chellappa, "Fast electronic digital image stabilization for off-road navigation**",** Proc. of IEEE on Real-Time Imaging**,** Vol.2 , No. 5,Pages: 285 - 296 ,1996.

[10] A. Smolic, T. Sikora, and J.-R. Ohm, "Direct estimation of long-term global motion parameters using affine and higher order polynomial models", in Proc. PCS'99, Picture Coding Symposium, Apr. 1999.

[11] P. Vandewalle; S. Süsstrunk; and M. Vetterli, "Double resolution from a set of aliased images", In: Proc. IS&T/SPIE Electronic Imaging 2004: Sensors and Camera Systems for Scientific, Industrial, and Digital Photography Applications V, vol. 5301, pp. 374-382,2004.

[12] R. Gonzalez, R. Woods, and S. Eddins," Digital Image Processing Using MATLAB", Prentice Hall, 2004.

[13] iMultimedia, "Use Image Stabilization – Gyroscopic Stabilizer", [online],URL http://www.websiteoptimization.com/speed/tweak/stabilizer. [Accessed 13-January-2006].

[14] R. Arther, "Fundamental of Electronic Image Processing", The SPIE/IEEE Series on imaging science & engineering, 1996.

[15] Canon Digisuper 100xs, Product Manual, [online] URL http://www.canon.com/bctv/products/pdf/100xs.pdf [Accesses 20-May-2006]

[16] E. Thomas , "Measuring the Effectiveness of Image/Video Processing for Stabilizing a Video Image Using a Commercial Media Processor", [Online], URL http://www.poly.asu.edu/ctas/dcst/Projects/pdf /WettThomas.pdf, [Access 13-Febrauary-2006].

[17] T. Vieville, E. Clergue, and P. Facao., "Computation of ego-motion and structure from visual and internal sensors using the vertical cue", In Proc. International conference on Computer Vision, Pages 591-598, Berlin, Germany, 1993.

[18] M. Irani, B. Rousso, and S. Peleg., 'Recovery of ego-motion using image stabilization", In Proc. Of IEEE conference on Computer Vision and Pattern Recognition, Pages 454-460, Seattle, WA, June 1994.

[19] O. Kwon, R. Chellappa, and C. Morimoto., ”Motion compensated subband coding of video acquired from a moving platform”, In Proc. Of IEEE International conference on Acoustics, Speech, and Signal Processing, pages 2185-2188, Detroit, MI, January 1995.

[20] P. Burt and P. Anandan., “Image Stabilization by registration to a reference mosaic”, In Proc. DARPA Image Understanding Workshop, pages 425-434, Monterey, CA, November 1994.

[21] C. Morimoto, D. DeMenthon, L. Davis, R. Chellappa, and R. Nelson, ”Detection of independently moving objects in passive video”, In Proc. Of Intelligent Vehicles Workshop, pages 270-275, Detroit, MI, September 1995.

[22] S. Balakirsky., “Comparison of electronic image stabilization systems”, Master’s thesis, Department of Electrical Engineering, University of Maryland, College Park, 1995.

[23] G. Robert., “ VideoMaker Magazine The End Of Shaky Camera”,[online], URL http://www.videomaker.com/scripts/article_print.cfm?id=9999,[Accessed 13-January-2006]

[24] L.S. Davis, R. Bajcsy, R. Nelson, and M. Herman. “RSTA on the move” In Proc.. DARPA Image Understanding Workshop, pages 435-456, Monterey, CA, November 1994.

[25] M. Hansen, P. Anandan, K. Dana, G. van der Wal, and P.J. Burt. “Real-time scene stabilization and mosaic construction”. In Proc. DARPA Image Understanding Workshop, pages 457-465, Monterey, CA, November 1994.

[26] H. Sawhney, S. Ayer, and M. Gorkani. “Model-based 2d and 3d dominant motion estimation for mosaicing and video representation” In Proc.. International Conference on Computer Vision, pages 583-590, Cambridge, MA, June 1995.

[27] Z. DuriC and A. Rosenfeld. “Stabilization of image sequences” Technical Report CAR-TR-778, Center for Automation Research, University of Maryland, College Park, 1995.

[28] C. Morimoto and R. Chellappa. ”Fast 3d stabilization and mosaicking”. In Proc. IEEE Conference on Computer Vision and Pattern Recognition, Puerto Rico,PR, June 1997.

[29] Y. Yao, P. Burlina, and R. Chellappa.,” Electronic image stabilization using multiple visual cues”. In Proc. International Conference on Image Processing, pages 191-1 94, Washington, D.C., October 1995.

[30] S. Kim and W. Su, “Subpixel accuracy image registration by spectrum cancellation” in Proc. IEEE International Conference on Acoustics, Speech and Signal Processing, Vol. 5, pp. 153, 1993.

[31] H. Stone, M. Orchard, E. Chang, and S. Martucci, “A fast direct Fourier-based algorithm for subpixel registration of images,” IEEE Transactions on Geoscience and Remote Sensing 39, pp. 2235{2243, October 2001.

[32] B. Marcel, M. Briot, and R. Murrieta, “Estimation of translation and rotation by Fourier transform,” Signal Vol. 14, pp. 135-149, 1997.

[33] K. Kimura, “Angular Velocity Measuring Instrument”, USP 2544646, 1985.

[34] A tutorial on Clustering Algorithms, [On Line] URL http://www.elet.polimi.it/upload/matteucc/Clustering/tutorial_html /index.html [Accesses 22-January-2006].

[35] R. Fisher, “The Ransac (Random Sample Consensus) Algorithm” , [online], URL http://homepages.inf.ed.ac.uk/ rbf/CVonline / LOCAL_COPIES / FISHER/ RANSAC/index.html, [Access 23-May-2006].

[37] P. Vandewalle, S. Susstrunk, and M. Vetterli, “A frequency domain approach to registeration of aliased images with application to super-resolution,” EURASIP Journal on Applied Signal Processing, 2006.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From – To)*
13-06-2006	**Master's Thesis**	Oct 2004 – May 2006

4. TITLE AND SUBTITLE

Fast Video Stabilization Algorithms

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Alharbi, Mohammed A., Captain, RSAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN), Bldg 640
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/06-02

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
A fast and robust electronic video stabilization algorithm is presented in this thesis. It is based on a two-dimensional feature-based motion estimation technique. The method tracks a small set of features and estimates the movement of the camera between consecutive frames. It is used to characterize the motions accurately including camera rotations between two imaging instants. An affine motion model is utilized to determine the parameters of translation and rotation between images. The determined affine transformation is then exploited to compensate for the abrupt temporal discontinuities of input image sequences. Also, a Frequency domain approach is developed to estimate translations between two consecutive frames in a video sequence. Finally, a jitter detection technique to isolate vibration affected subsequence of an image sequence is presented. The experimental results of using both simulated and real images have revealed the applicability of the proposed techniques. In particular, the emphasis has been to develop real time implementable algorithms, suitable for unmanned vehicles with severe payload constraints.

15. SUBJECT TERMS
Video Stabilization, Motion Analysis, Image Registration, Video Processing

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
					Guna S. Seetharaman, Ph.D., (ENG)
REPORT	**ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(Include area code)*
U	**U**	**U**	**UU**	126	(937) 255-3636, ext 4612; e-mail: guna.seetharaman@afit.edu

Standard Form 298 (Rev: 8-98)
Prescribed by ANSI Std. Z39-18