AGH University of Science and Technology

Faculty of Electrical Engineering, Automatics, Computer Science and Electronics

Master of Science Thesis

Bartłomiej Dudek

Development of software anti-shake filter for video stream

Supervisor:
Jarosław Bułat Ph.D

Kraków 2011

OŚWIADCZENIE AUTORA PRACY

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie i że nie korzystałem ze źródeł innych niż wymienione w pracy.

.....

TABLE OF CONTENTS

INTRODUCTION

Visual inspection has an irreplaceable meaning in the quality control and the data analysis processes. An eyesight is probably the most important of human senses, so the video documentation is so meaningful. Images sequence encapsulates huge amount of information but the distortions may make them useless. Nowadays, the video analysis, which is a branch of signal processing, is very advanced. There are two main variants of its operation: real-time and off-line analysis. Both of them have to achieve different requirements. The off-line video processing must provide very good efficiency but it can be achieved by taking some time. The real-time video processing at first must be fast and as far as the results are transient, the accuracy is not the most important thing.

Cameras' evolution and miniaturization made from them a very powerful inspection tool. The camera can be mounted on a moving or flying robot and used for research or operation in hazardous environment. Micro cameras have also very important role in endoscopy and visual inspection in hard to reach areas. In each of these applications, good eye-hand coordination is critical. Image shakes are a serious problem in case of micro cameras and can make teleoperation very hard and slow. The hardware stabilization can not be applied in each instance and it has also limited capability. The solution seems to be the real-time software stabilization usage.

The goal of this work was a development of such an anti-shake filter. As an application's environment the FFmpeg libraries set was used. The stabilization had taken place in a post processing what made it independent of processed video's format. The resultant application should have handle the real-time operation on an average PC machine. The stabilization was assumed as shakes suppression with preservation of the intentional camera move. Any time delay was not acceptable to not disturb the eye-hand feedback.

This paper contains three main parts: the theoretical introduction which covers all knowledge needed for the filter's development, a report from its proceedings and the evaluation of working application's performance. The development itself was

accomplished in two steps: a preliminary implementation of each part separately and the final implementation of the whole system. To achieve the real-time performance, some accuracy has to be dropped. However, the application was designed in such a way, that it still can use more accurate but slower algorithms which were implemented as optional ones.

CHAPTER 1

CAMERA VISION AND IMAGE TRANSFORMING

This chapter includes some basic knowledge about a projection of 3D world on the 2D plane, which takes place in a camera. It names basic kinds of camera moves and describe sources of camera shakes. There were also presented some informations about the image transformation process, main types of transformations and methods of estimation the transformation matrix.

1.1 Projective geometry and 2D transformations

In [1] the projective geometry was defined as the study of the projective plane $\mathbb{P}^2$. Each point of $\mathbb{P}^2$ is represented by a ray in Euclidean space $\mathbb{R}^3$, also each line in $\mathbb{P}^2$ is represented by a plane in $\mathbb{R}^3$ (figure 1.1). All of them intersect in the origin of $\mathbb{R}^3$.

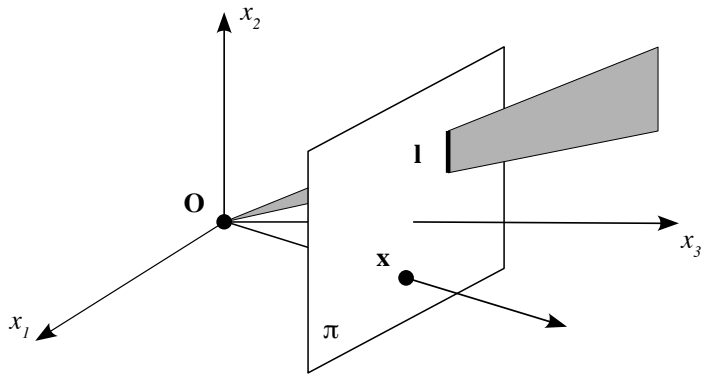


Figure 1.1. A model of projective plane – points from $\mathbb{P}^2$ are represented by rays in $\mathbb{R}^3$ (as for example the point $\mathbf{x}$ is), lines are represented by planes (the line $\mathbf{l}$ is an example). Their real shapes can be obtained by intersecting the set of rays and planes by the plane $x_3 = 1$ in the picture denoted as π.

Each point $\mathbf{x}$ of $\mathbb{P}^2$ is represented by the vector $k(x_1,\ x_2,\ x_3)^T$ in $\mathbb{R}^3$, where k varies along the ray.

1.1.1 Homogenous coordinates

Every single point of Euclidean plane is represented by a pair of real numbers $(x,\ y)$. In consequence of addition an extra coordinate, a triple $(x,\ y,\ 1)$ is obtained. That triple is called the homogenous coordinate of the point [1]. The rules for converting point coordinates between plane and homogenous coordinates are following [3]:

- To convert a point $(x,\ y)$ to homogenous coordinates, it is enought to add a third component equal to 1.

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

- To convert a homogenous coordinates triplet $(a,\ b,\ c)$ to the pair of 2D plane coordinates, first two components have to be divided by the third one.

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} \rightarrow \begin{pmatrix} \frac{a}{c} \\ \frac{b}{c} \end{pmatrix}$$

An individual point of plane, defined by a pair $(x,\ y)$, has an infinite number of representations in homogenous coordinates. In fact, points are represented by equivalence classes of homogenous coordinate triplets, where two triplets define the same point if they differ by a common multiple [1].

$$\begin{pmatrix} kx \\ ky \\ k \end{pmatrix} \equiv \begin{pmatrix} lx \\ ly \\ l \end{pmatrix}$$

1.1.2 Camera projection

To describe camera's operation, a pinhole camera model was used (figure 1.2). That model describes a real image acquisition system very well. It takes into consideration the central projection of points in a space on the plane called focal plane or image plane [2].

The camera centre (called also the optical centre) $\mathbf{C}$ is the origin of Euclidean coordinate system. The image plane is placed parallel to XY plane, in $Z = f$. Each

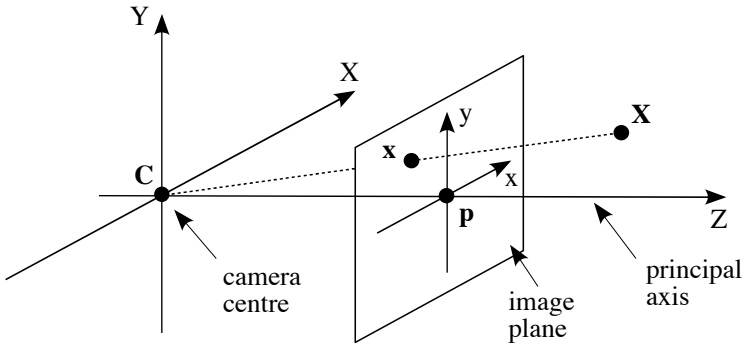


Figure 1.2. Pinhole camera geometry – **C** is the camera centre, **p** is the principal point.

point **X** from $\mathbb{R}^3$ space is mapped to the point **x** onto the image plane like in (1). This relation was graphically illustrated in the figure 1.3.

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \rightarrow \begin{pmatrix} f\frac{X}{Z} \\ f\frac{Y}{Z} \\ f \end{pmatrix} \tag{1}$$

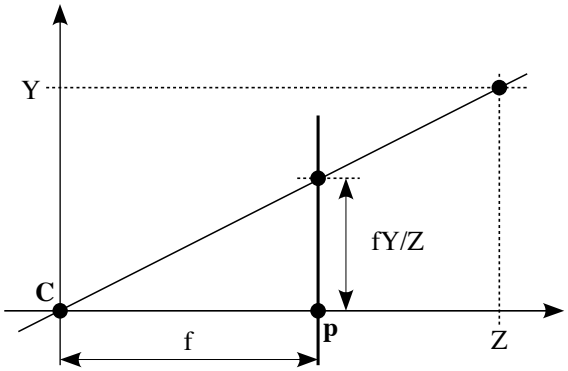


Figure 1.3. Pinhole camera geometry – the projection on YZ plane.

In homogenous coordinates, (1) can be rewritten as (2) or as the matrix equation (3), where **P** is called the camera projection matrix [1].

$$\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{pmatrix} \frac{fX}{Z} \\ \frac{fY}{Z} \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{2}$$

$$\mathbf{x} = \mathbf{PX} \tag{3}$$

In practice, it may happen that the origin of coordinates is not placed in the principal point. In that case, the camera projection with an offset correction can be expressed as

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}. \tag{4}$$

Introducing the camera calibration matrix $\mathbf{K}$ that

$$\mathbf{K} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{5}$$

the camera projection matrix can be expressed as

$$\mathbf{P} = \mathbf{K}\left[\mathbf{I} \mid \mathbf{0}\right]. \tag{6}$$

Until now, points were placed in the camera coordinate frame in which the origin of coordinate system is the camera centre. In general, it is often necessary to use the world coordinate frame (e.g. to distinguish between the point and the camera move). That two coordinate systems are related by the rotation matrix $\mathbf{R}_{3\times3}$ and the translation vector, which is equal to the camera centre placement in the world coordinates $\widetilde{\mathbf{C}}$. Therefore, the formula that converts the camera coordinates into the world ones has a form

$$\mathbf{X}_{CAM} = \begin{bmatrix} \mathbf{R} & -\mathbf{R}\widetilde{\mathbf{C}} \\ 0 & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{bmatrix} \mathbf{R} & -\mathbf{R}\widetilde{\mathbf{C}} \\ 0 & 1 \end{bmatrix} \mathbf{X}_{WORLD}. \tag{7}$$

From (7), the final form of the pinhole camera projection matrix can be expressed as [1]

$$\mathbf{P} = \mathbf{K}\mathbf{R}\left[\mathbf{I} \mid -\widetilde{\mathbf{C}}\right]. \tag{8}$$

That matrix, and as result the general pinhole camera itself, has nine degrees of freedom:

- 3 for elements of matrix $\mathbf{K}$ – f, p_x and p_y,
- 3 for rotation matrix $\mathbf{R}$,
- 3 for camera center position (translation) $\widetilde{\mathbf{C}}$.

1.1.3 A hierarchy of 2D transformations

In [3] a geometric transformation was defined as:

> "A function that is both onto and one-to-one, and whose range and domain are points."

Onto (suriective) is a function whose every point in range has a corresponding point in the domain. One-to-one (injective) function must fulfill the condition $x \neq y \implies f(x) \neq f(y)$.

Basic 2D transformation of point $\mathbf{x}$ into the point $\mathbf{x}'$ can be writen in the matrix form as

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \mathbf{T}_{2\times 2} \begin{pmatrix} x \\ y \end{pmatrix}. \tag{9}$$

A 2×2 transformation matrix allows only for "reflect", "flip", "zoom", "rotate" or "shear" operations. To perform translation or more complex projective transformation, the point coordinates have to be transformed into the homogenous ones and the 3×3 transformation matrix have to be used. In that case (9) becomes (10).

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \mathbf{T}_{3\times 3} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{10}$$

In [1] transformations are grouped in four classes of a hierarchy of transformations.

Class I: Isometries

Isometries are transformations that preserve Euclidean distance. That class of transformations is described by an equation

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \tag{11}$$

That transformations have three degrees of freedom: one for the rotation angle θ and two for the translation $(t_x,\ t_y)$. It can be computed from two corresponding points.

Class II: Similarity transformations

That transformations are combined from an isometry and scaling and can be described by an equation

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} s\cos\theta & -s\sin\theta & t_x \\ s\sin\theta & s\cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \tag{12}$$

That transformations have four degrees of freedom: one for the rotation angle θ, two for the translation (t_x, t_y) and one for the scaling factor s. It can be computed from two corresponding points.

Class III: Affine transformations

Affinities are linear transformations described by an equation

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \tag{13}$$

That transformations have six degrees of freedom – one for each element in first two rows of the transformation matrix. It can be computed from three corresponding points.

Class IV: Projective transformations

Projective transformation is a general non-singular linear transformation of homogenous coordinates

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \tag{14}$$

That transformation matrix has nine elements, but only their ratio matters. That implies that the transformation is defined by eight parameters and has eight degrees of freedom. It can be computed from four corresponding points with no collienarity of three of them on either plane.

A short description of all of transformation classes was contained in the table 1.1, basing on [1]. The figure 1.4 shows examples of transformation matrices and results of transformations accomplished by them.

1.2 Camera moves

As was mentioned in the subsection 1.1.2, a general pinhole camera model has nine degrees of freedom. In most cases, only seven of them can be directly controlled by the operator: camera moves along three axes, camera turns around three axes and the focal length changes. Combination of these simple moves allows to create more complex effects.

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(a) Original image.

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(b) Vertically reflected image.

$$T = \begin{bmatrix} 0.7 & 0 & 0 \\ 0 & 0.7 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(c) Image scaled by factor of 0.7.

$$T = \begin{bmatrix} 1 & 0 & 150 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(d) Horizontally translated image.

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 150 \\ 0 & 0 & 1 \end{bmatrix}$$

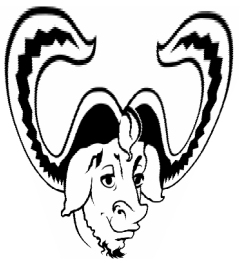
(e) Vertically translated image.

$$T = \begin{bmatrix} 1 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(f) Sheared image.

$$T = \begin{bmatrix} 0.87 & 0.5 & 0 \\ -0.5 & 0.87 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(g) Rotated image.



$$T = \begin{bmatrix} 2 & 2 & 0 \\ 0 & 6 & 0 \\ 0 & 0.01 & 1 \end{bmatrix}$$

(h) Example of projective transformation.

Figure 1.4. Examples of basic 2D image transformations and used transformation matrices.*

*Original image source: http://www.gnu.org/graphics/heckert_gnu.html

Table 1.1. A hierarchy of 2D transformations – $(t_x,\ t_y)$ is a 2D translation vector, $\mathbf{R} = [r_{ij}]$ is a 2D rotation matrix and $\mathbf{A} = [a_{ij}]$ is an invertible 2×2 matrix.

Group	DOF	Transformation matrix	Invariant properties
Euclidean	3	$\begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$	length, area
Similarity	4	$\begin{bmatrix} sr_{11} & sr_{12} & t_x \\ sr_{21} & sr_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$	ratio of lengths, angle
Affine	6	$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$	parallelism, ratio of areas, ratio of length on collinear or parallel lines, linear combinations of vectors
Projective	8	$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$	concurrency, collinearity, order of contact (intersection, tangency, inflections), cross ratio (ratio of ratio of lengths)

1.2.1 Camera movement techniques

In [4] and [5] following camera moves were named:

Pan – a horizontal pivot of the camera while it stays otherwise stationary. This technique can be used to look across wide landscapes, to follow characters or vehicles or to introduce a new element into the scene;

Tilt – a vertical pivot of the camera. This technique can be used to look over tall objects or to follow falling or raising objects;

Tilted horizon – a move in which the camera is tipped slightly to the side. Used to attract the viewer's attention or to increase the tension in a scene;

Pedestal – changes of the camera altitude without any pivot;

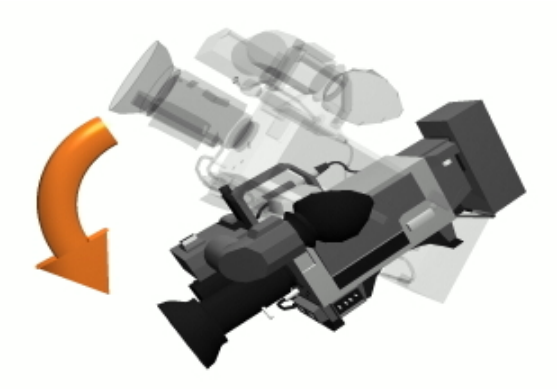
Track – a sideways camera move, often with keeping a certain distance from the tracked object;

Dolly – a physical camera move toward or away from an object without changing the focal length;

Zoom – changes of the camera focal length. Zooming allows to transition from wide scene to the close-up without any changes of the camera position.

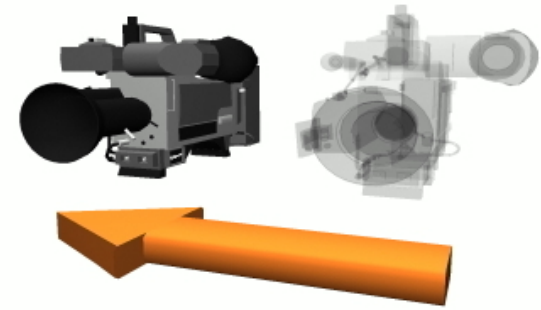
Examples of them were shown in the figure 1.5.

(a) Pan.

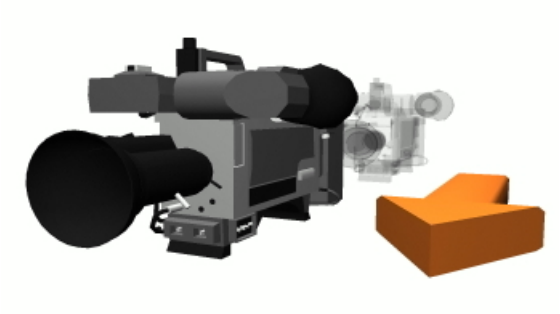


(b) Tilt.

(c) Pedestal.



(d) Truck.



(e) Dolly.

Figure 1.5. Simple camera moves illustration.*

*Source: http://www.tv-handbook.com/Composition and Camera Movement.html

1.2.2 Camera holding techniques

There are several ways of supporting the camera [6]:

- holding it in hand,
- resting it on a shoulder,
- using a body support,
- using a mounting.

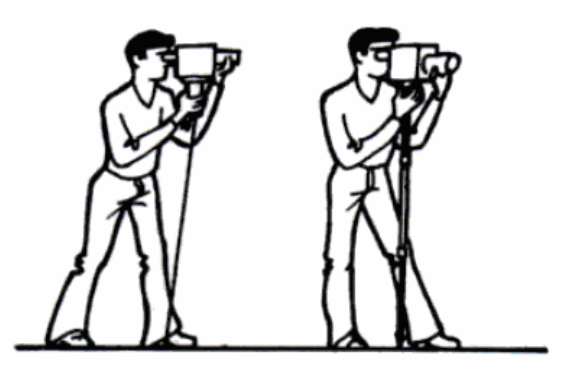
Some examples of body supporting camera were shown in the figure 1.6. To stabilize the camera shoot some equipment can be also used. The more stable solutions are a tripod or a dolly, but they limit the flexibility of camera movement. Typical ways of stabilize a hand-held camera were shown in the figure 1.7. The revolution in steadying the hand-held camera take was an invention of Steadicam made by Garrett Brown in 1970s. Steadicam is an elaborate harmess which separates camera and operator moves. A lightweight version of Steadicam (the Steadicam Junior) was developed in 1980s. The Steadicam is still in use, especially in professional applications.

Figure 1.6. Exemplary methods of taking steady hand-held camera shoots – with stable body positions and with nearby supports usage.*

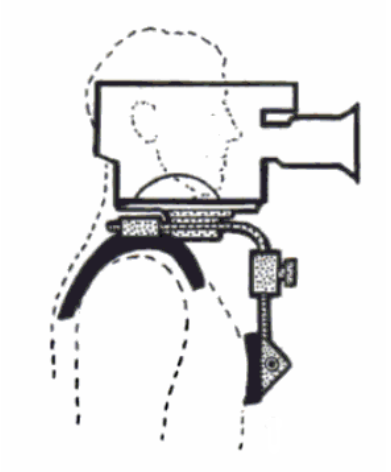
*Source: G. Millerson. *Video camera techniques.*

1.2.3 Undirected random shakes

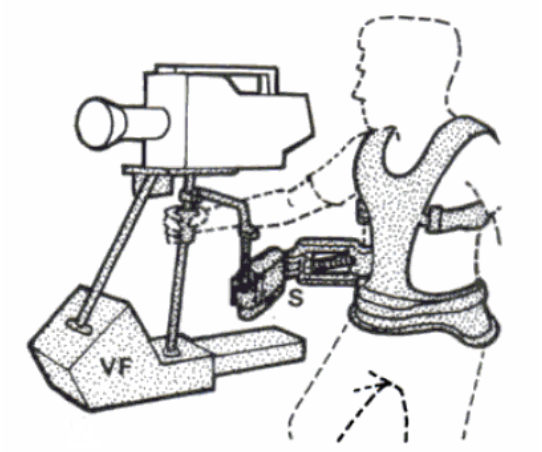
Besides intended camera moves, when a camera is hand-helded there are always some shakes (caused by steps, breathing or muscle contractions). Sometimes that



(a) Simple camera stabilization with a string, chain or monopod support.



(b) Schoulder support.



(c) Steadicam – an elaborate harmess with the spring suspension.

(d) Steadicam Jr. – a compact ballanced support, the hand-held version of a Steadicam.

Figure 1.7. Exemplary devices for camera stabilization.*

*Source: G. Millerson. *Video camera techniques.*

phenomenon is used to make the take more dynamic, but in most cases shakes are not welcome. The best way to diminish undirected camera moves is to use a steady camera support but it is not always possible (e.g. inspection cameras or endoscope). A good solution of the camera image stabilization issue is usage of an electronic image stabilization. Examples of that approach may be systems developed by SONY or Canon. The principle of theirs operation is to move the lens, the image sensor or both of them (hybrid stabilization) in way to compensate the camera shakes and keep the image still – figures 1.9 and 1.8. The main drawback of these solutions are big dimensions and complexity of whole system.

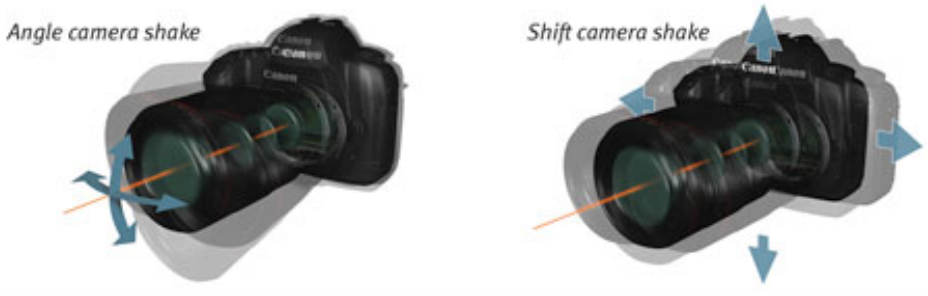


Figure 1.8. Types of camera shakes.*

*Source: http://www.usa.canon.com/cusa/consumer/standard_display/Lens_Advantage_IS

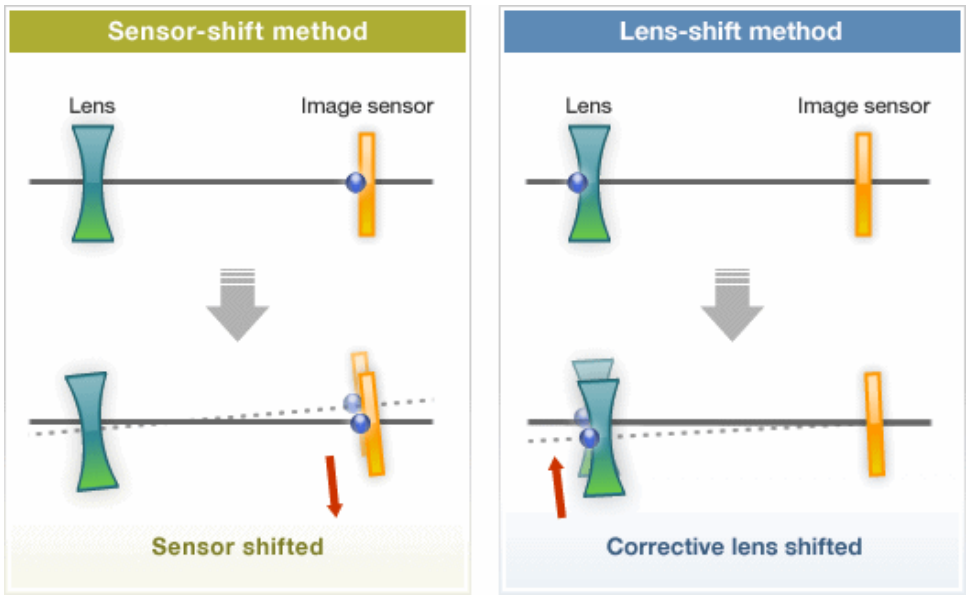


Figure 1.9. Operation modes of the electronic image stabilization.*

*Source: http://www.sony.net/SonyInfo/technology/technology/theme/alpha_01.html

1.3 Transformation matrix estimation

With a set of points $\mathbf{X}$ and the corresponding one $\mathbf{X}'$, it is possible to recover the transformation which maps each $\mathbf{x}_i$ to $\mathbf{x}'_i$. This section focuses only on the 2D transformations matrix estimation case.

1.3.1 Linear algorithms

The Direct Linear Transformation is an example of linear algorithm which allows to evaluate a transformation (homogeneity) matrix $\mathbf{H}$ of transformation described by the equation

$$\mathbf{x'}_i = \mathbf{H}\mathbf{x}_i. \tag{15}$$

According to [1], four corresponding points are necessary to compute that transformation matrix coefficients. Sought transformation (15) can be expressed in terms of the vector cross product as

$$\mathbf{x'}_i \times \mathbf{H}\mathbf{x}_i = \mathbf{0}. \tag{16}$$

Denoting the j^{th} row of the matrix $\mathbf{H}$ as a vector $\mathbf{h}^{j\mathrm{T}}$, the right-hand side of (15) may be written as

$$\mathbf{H}\mathbf{x'}_i = \begin{pmatrix} \mathbf{h}^{1\mathrm{T}}\mathbf{x}_i \\ \mathbf{h}^{2\mathrm{T}}\mathbf{x}_i \\ \mathbf{h}^{3\mathrm{T}}\mathbf{x}_i \end{pmatrix}. \tag{17}$$

In homogenous coordinates $\mathbf{x'}_i = (x_i,\ y_i,\ 1)^{\mathrm{T}}$, so (16) becomes

$$\mathbf{x'}_i \times \mathbf{H}\mathbf{x}_i = \begin{pmatrix} y'_i\mathbf{h}^{3\mathrm{T}}\mathbf{x}_i & - & \mathbf{h}^{2\mathrm{T}}\mathbf{x}_i \\ \mathbf{h}^{1\mathrm{T}}\mathbf{x}_i & - & x'_i\mathbf{h}^{3\mathrm{T}}\mathbf{x}_i \\ x'_i\mathbf{h}^{2\mathrm{T}}\mathbf{x}_i & - & y'_i\mathbf{h}^{1\mathrm{T}}\mathbf{x}_i \end{pmatrix} = \mathbf{0}. \tag{18}$$

Since $\mathbf{h}^{j\mathrm{T}}\mathbf{x}_i = \mathbf{x}_i^{\mathrm{T}}\mathbf{h}^j$, (18) can be expressed as

$$\begin{bmatrix} \mathbf{0}^{\mathrm{T}} & -\mathbf{x}_i^{\mathrm{T}} & y'^{\mathrm{T}}\mathbf{x}_i^{\mathrm{T}} \\ \mathbf{x}_i^{\mathrm{T}} & \mathbf{0}^{\mathrm{T}} & -x'^{\mathrm{T}}\mathbf{x}_i^{\mathrm{T}} \\ -y'_i\mathbf{x}_i^{\mathrm{T}} & x'_i\mathbf{x}_i^{\mathrm{T}} & \mathbf{0}^{\mathrm{T}} \end{bmatrix} \begin{pmatrix} \mathbf{h}^1 \\ \mathbf{h}^2 \\ \mathbf{h}^3 \end{pmatrix} = \mathbf{0}. \tag{19}$$

The equation set (19) can be represented in a matrix form as $\mathbf{A}_i\mathbf{h} = \mathbf{0}$, where $\mathbf{A}_i$ is a 3×9 matrix. One of its rows can be omitted, because only two of equations in (19) are linearly independent. Each pair of corresponding points introduces two independent equations in the entries of transformation matrix $\mathbf{H}$. When there are

four known correspondences, the set of equations

$$\mathbf{A}\mathbf{h} = \mathbf{0} \tag{20}$$

is obtained, where $\mathbf{A}$ is the matrix of equation coefficient build from rows $\mathbf{A}_i$ contributed from each correspondence and $\mathbf{h}$ is the vector made from the entries of estimated matrix $\mathbf{H}$.

$\mathbf{A}$ has rank 8, so it is enough to use only two equations from (19) for each points' correspondence. Due to its rank, $\mathbf{A}$ has 1-dimmensional null-space (a set of many solutions). Because $\mathbf{H}$ is only determined up to a scale, the solution $\mathbf{h}$ is sufficient. This scale may be arbitrarily chosen by a requirement on its norm such as $\|\mathbf{h}\| = 1$ [1].

1.3.2 Over-determined case and different cost functions

If more than four corresponding points pairs $\mathbf{x}_i \leftrightarrow \mathbf{x'}_i$ are given, then the set of equations (20) is over-determined and the matrix $\mathbf{A}$ has more linearly independent rows than 8. If the position of all points is exact, then the rank of the matrix $\mathbf{A}$ is still 8 and there is a single exact solution for $\mathbf{h}$. If points position is uncertain (e.g. due to a noise), there is no exact solution for $\mathbf{h}$. Instead of that, an approximate solution which minimizes a suitable cost function can be found. An additional constrain has to be formulated to avoid the solution $\mathbf{h} = \mathbf{0}$. Generally, the condition $\|\mathbf{h}\| = 1$ is used, but it is also possible to minimize the norm $\|\mathbf{A}\mathbf{h}\|$ instead. As was shown in [1], the solution is the eigenvector of $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ with the least eigenvalue. In numerical computations, the singular value decomposition (SVD) can be applied. In that case

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathrm{T}} \tag{21}$$

where the last column of the matrix $\mathbf{V}$ is the solution of (20).

1.3.3 Iterative minimization methods

Iterative methods minimize various geometric cost functions to obtain the estimate of transformation. The most popular of them are the Newton's method and the Lovenberg-Marquardt algorithm. According to [1], iterative approach have certain disadvantages with regard to linear algorithms:

- they are slower,
- they need an initial approximate estimation as starting conditions,
- they may not converge or converge to a local instead of the global minimum,
- termination criteria are not always easy to define.

In consequence of that, iterative techniques must be implemented very carefully. The iterative minimization process, in general case, consist of five steps:

1. Choice of the cost function – there are many possibilities, some of them was described in details in [1].

2. Parametrization – a choice of estimated entries in the matrix **H**.

3. Function specification – expression of the cost in term of the set of sought parameters.

4. Initialization – an initial estimate of starting point, usually using some kind of linear algorithm.

5. Iteration – an iterative improvement of the estimation parameters with the goal of minimizing the cost function.

1.3.4 Robust estimation – RANSAC

Up to this point, it had beed assumed that only uncertainty of corresponding points position is the measurement noise, which follows a Gaussian distributions. In practice, this assumption is almost never applicable, because of some amount of mismatched points. That mismatched points are outliers to the Gaussian error distribution. These outliers have a significant influence on the estimation process and can obviously disturb its result – figure 1.10(a). The goal of the robust estimation is to distinguish inliers from outliers and base the estimation only on inliers.

Random Sample Consensus (RANSAC) algorithm was developed by Fischler and Bolles in 1981 [7]. It can handle even a large proportion of outliers. The steps of RANSAC estimation are following:

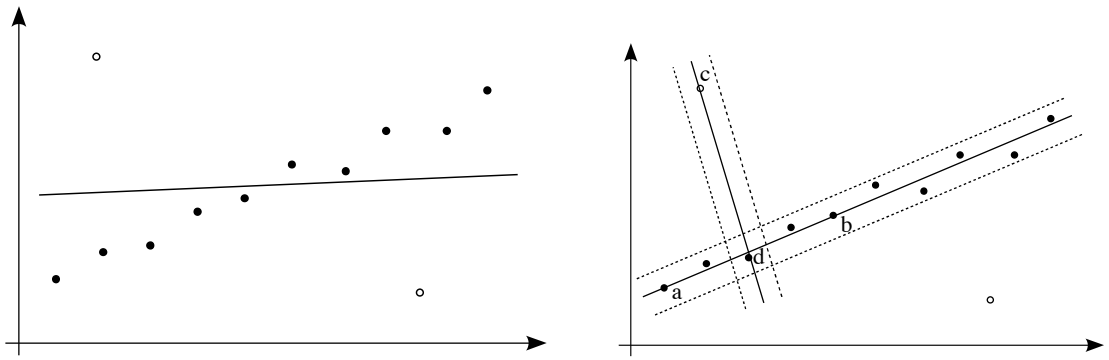
1. A random selection of a subset of s from S samples. The number of selected samples is a minimal number of points necessary to determine the model.

2. The pick of the support S_i of the i^{th} model as a colection of points within a distance threshold t from it – inliners (figure 1.10(b)).

3. If the size of S_i is greater than some threshold T, the output model is re-estimated basing on points from S_i and the algorithm terminates.

4. If the size of S_i is less than T, the new subset s is selected and points 2–4 are repeated.

5. After N iterations, the largest subset S_i is selected, the model is re-estimated basing on points within it and the algorithm terminates.



(a) A Maximum Likehood estimation (line) of the set of points affected by two outliers.

(b) Two lines defined by randomly selected points: the line $c - d$ with a support of 2, and the line $a - b$ with a support of 10 points.

Figure 1.10. Robust line estimation.

Table 1.2. Threshold values t for classifying a point as an inlier with the probability $\alpha = 0.95$ [1].

Codimension m	Model	t^2
1	line, fundamental matrix	3.84 σ^2
2	homography, camera matrix	5.99 σ^2
3	trifocal tensor	3.84 σ^2

The distance t is choose in a such way that the point is an inliner with a probability α. Usually, that value is picked empirically, but if the measurement of points' position error is Gaussian with zero mean value and standard deviation σ, t can be analytically computed. Values of threshold t for $\alpha = 0.95$ were shown in the table 1.2.

It is often unnecessary to try every possible subset of samples. There can be choose a number of samples N efficiently high to ensure with a probability p that at least one random subset s is free from outliers. Usually $p = 0.99$. Assuming a probability of selection an outlier ϵ, equal to the proportion of outliers, at least N selections of s points are required. Therefore

$$N = \frac{\log(1-p)}{\log(1-(1-\epsilon)^s)}. \tag{22}$$

The table 1.3 shows exemplary numbers of selections under different conditions.

One of RANSAC algorithm termination conditions is a size T of the model support, which is sufficient to find a solution good enough to stop searching. A general rule to pick that value is [1]

$$T = (i - \epsilon)n \tag{23}$$

where n is number of all samples.

Table 1.3. The number of N selections required to ensure at least one subsample of s samples free from outliers with a probability $p = 0.99$ [1].

Subset size s	Proportion of outliers ϵ						
s	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177

Determining the number of samples adaptively

In case, when a proportion of outliers ϵ in the data set is unknown, the idea of adaptively determination of maximal number of trials N can be applied. This approach works well in practice and solves a problem of setting a number of required samples and condition of terminating the RANSAC estimation at once. An algorithm of adaptive computation of N is [1]:

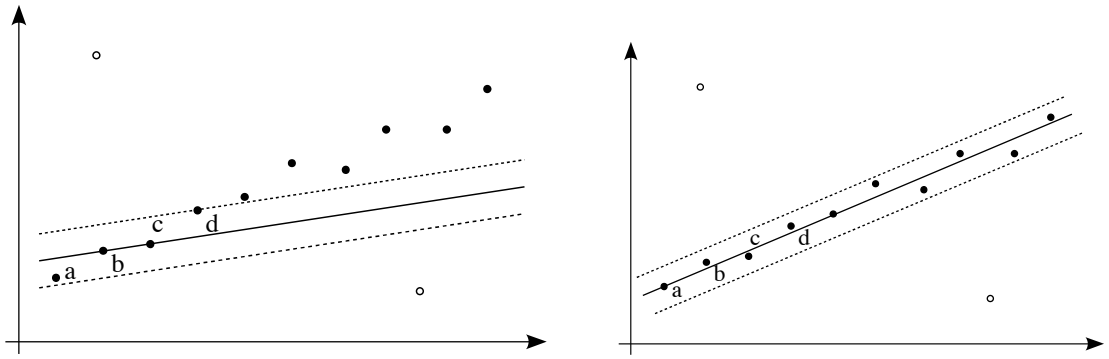
1. Assume $N = \infty$ and sample count $i = 0$.
2. While $N > i$, repeat:
   - choose a subset of s samples and count the support of the model,
   - set a new proportion of outliers $\epsilon = 1 - \frac{\text{number of inliners}}{\text{total number of samples}}$,
   - set N from ϵ and (22),
   - increment i by 1.
3. Terminate.

The block diagram of

A good way to improve the RANSAC estimation efficiency is to perform a two stage estimation:

1. Preliminary choice of a model with the greatest support.
2. Precise re-estimation using support of RANSAC model as an input and minimization of some kind of cost function.

An example of that operation was shown in the figure 1.11.



(a) A Line defined by **b** and **c** with the support of four $\{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}\}$ found by the RANSAC algorithm.

(b) The ML line fit to points $\{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}\}$ – much improved fit with the support of 10 points.

Figure 1.11. Two stages of the robust Maximum Likelihood estimation.

CHAPTER 2
MOTION VECTORS IN VIDEO COMPRESSION

A raw video stream is a huge amount of data (even above 1 Gbps). The video compression allows to reduce storage space needed to store it or required bitrate of the video transmission stream. There are two main types of compression: the lossless and the lossy one. The lossless compression algorithms reach at most compression ratio 6:1. Because of the video features (short display time of a particular frame, continuous tones) and human's eyesight limitations, even the quality loss in the lossy video compression can be imperceptible. Moreover, even if the change is visible, it often does not reduce the image utility, if there are no additional distortions. A lossy video compression can achieve even 50:1 compression ratio, still holding a very good image quality.

In general, the video compression reduces the amount of data which are stored or sent. To achieve this, three operations are mainstream:

- single frame compression,
- color planes subsampling,
- motion compensation.

2.1 Still image compression – the M-JPEG encoder

The simplest approach to the video compression is to compress each frame separately. This idea is a background of Motion JPEG (M-JPEG) video coding standard based on the JPEG compression. In this technique frames are non-related, so random access, fast forward, fast rewind and other operations can be performed very fast. Therefore, the M-JPEG is a good choice for video editing. It is also used because of its low complexity in terms of space and time and simplicity of implementation in both hardware and software. The main drawback of M-JPEG compression is low compression ratio, the same as for still images coded by JPEG – from 10:1 to 15:1, without significant distortions.

2.1.1 Discrete Cosine Transform (DCT) based coding

DCT-based coding is a basis of JPEG and MPEG encoders. As it was described in [8], the DCT decomposes every macroblock of a frame into a series of waveforms, each with a particular spatial frequency. A macroblock's size 8×8 pixels is the balance between good image quality and acceptable computational and memory requirements. The DCT of 8×8 macroblock is expressed as

$$y(k,\ l) = \frac{c(k)c(l)}{4} \sum_{i=0}^{7} \sum_{j=0}^{7} x_{ij} \cos\left(\frac{(2i+1)k\pi}{16}\right) \cos\left(\frac{(2j+1)l\pi}{16}\right), \tag{24}$$

where $k,\ l = 0, 1, \ldots, 7$ and

$$c(x) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } \ x = 0 \\ 1 & \text{otherwise} \end{cases} . \tag{25}$$

The Inverse Discrete Cosine Transform in case of 8×8 pixel macroblock has a form

$$x(i,\ j) = \sum_{K=0}^{7} \sum_{L=0}^{7} y_{kl} \frac{c(k)c(l)}{4} \cos\left(\frac{(2i+1)k\pi}{16}\right) \cos\left(\frac{(2j+1)l\pi}{16}\right). \tag{26}$$

The graphical representation of 64 basis function of 8×8 block decomposition was shown in the figure 2.1. Every image macroblock is described as a sum of these functions, multiplied by a 8×8 table of coefficients. Higher order coefficients in most cases can be neglected, so there is a possibility to reduce amount of data in the frequency domain, without corrupting data in the image domain.

2.2 Color planes subsampling

The most natural manner of processing colors is the RGB plane usage, in which each pixel is described by three parameters – one for red, green and blue component. However, in the video compression luminance-chrominance color planes are mostly used. There are many proposed transformations between color planes. One of them, the conversion from RGB to YUV color coordinate systems has a form [3]

$$\begin{aligned} Y &= 0.299(R - G) + G + 0.114(B - G), \\ U &= 0.493(B - Y), \\ V &= 0.877(R - Y). \end{aligned} \tag{27}$$

Human eyesight is much more sensitive to changes of luminance than for changes of colour. This property can be used to increase a compression ratio by reducing the resolution in chroma planes. In a very commonly used 4:2:0 format for each four pixels of luminance plane come only two pixels of chrominance planes. Such

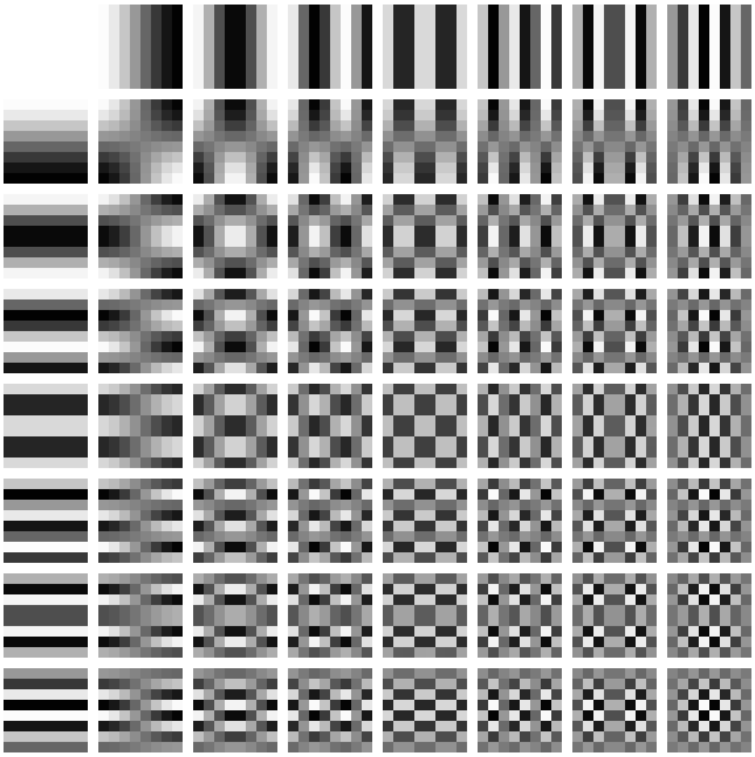


Figure 2.1. 64 basis function of 8×8 macroblock DCT.

approach allows to reduce the data volume required for store or send a single frame by a half.

2.3 Motion compensation

For achieve the better video compression ratio than in case of a still image compression, the motion compensation is used. Usually, consecutive frames are close to each other, so there is a possibility to use an existing frame's pieces to build a new one. A block diagram of such compression system was shown in the figure 2.2. The idea of motion compensation is following [8]:

1. Choice of a macroblock M in the reference frame on position $(x,\ y)$.
2. Calculation of a motion vector $(u,\ v)$ between frames.
3. Calculation of a prediction error e between frames.
4. Transmission of the motion vector's coordinates and the prediction error.

The prediction error can be compressed much more than the regular image, with an assumption that it is very small. For reconstruction of macroblock M in the current frame, the reference frame must be stored in the memory. There are three main types of frames (images) [8]:

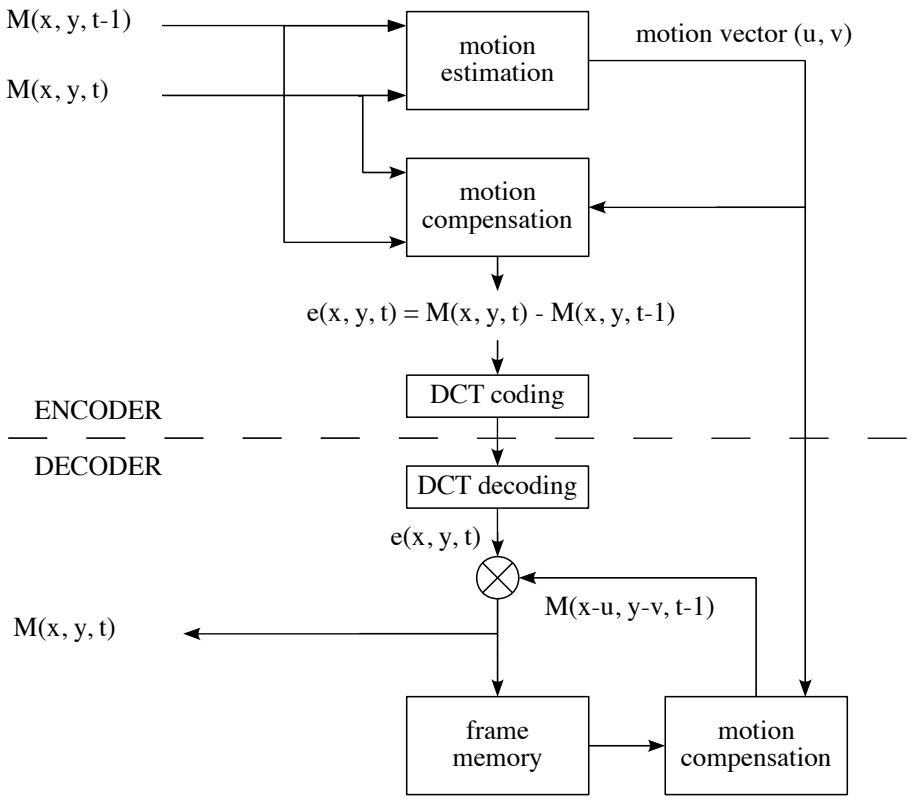


Figure 2.2. A generic video motion compensation system.

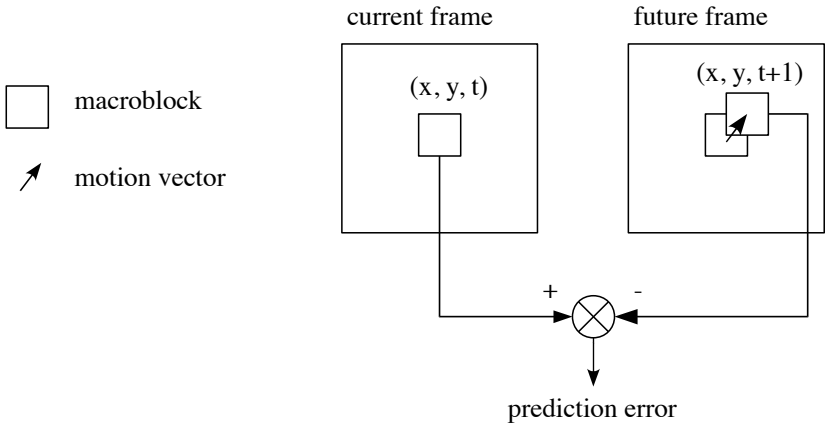
I-pictures – intra-pictures which does not refer to any other ones and must be sent or store as a still image.

P-pictures – predicted pictures which are coded using motion-compensated prediction from past I-type or P-type picture. Intermediate compression level and possibility to be used as a reference in the future.

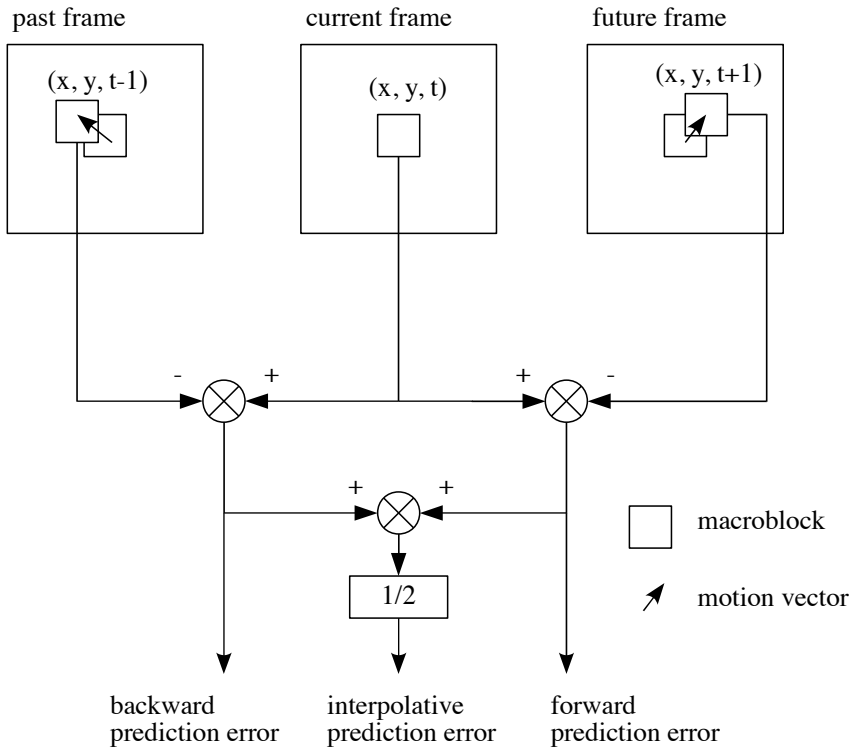
B-pictures – bidirectional predicted pictures coded using motion-compensated prediction from past and future I or P-type pictures. The highest degree of compression.

Techniques of uni- and bidirectional motion compensation and a prediction error calculation were illustrated in the figure 2.3. Example of inter-dependencies between I, P and B-pictures was shown in the figure 2.4.

The motion compensation is a basis of many video encoders like MPEG-1, MPEG-2, MPEG-4, H.261, H.263 and H.264 [9]. The motion vectors must be calculated fast (especially in case of live streaming the video) and precisely (to decrease the prediction error and avoid image distortions).



(a) The forward motion compensation.



(b) The bidirectional motion compensation.

FIGURE 2.3. Uni- and bidirectional motion compensation techniques.

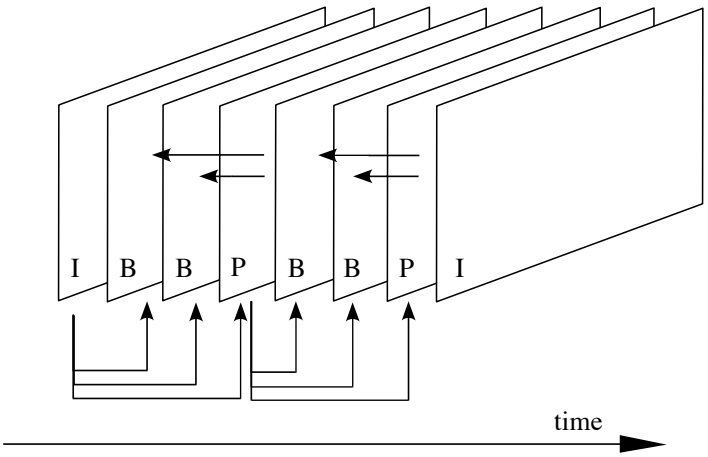


Figure 2.4. A generic video motion compensation system. Arrows represent dependencies between frames. Only I-pictures can be accessed independently.

2.4 Motion vectors estimation

There are four main groups of motion estimation techniques [10]. All of them rely on a hypothesis that the image intensity I on the position $\vec{r}$ is constant in the time period between two frames Δt and changes only due to displacement $\vec{d}$. Therefore

$$I(\vec{r},\ t) = I(\vec{r} - \vec{d},\ t - \Delta t). \tag{28}$$

The displacement frame difference (DFD) is defined as

$$\mathrm{DFD}(\vec{r},\ t,\ \vec{d}) = I(\vec{r},\ t) - I(\vec{r} - \vec{d},\ t - \Delta t). \tag{29}$$

2.4.1 Gradient techniques

Gradiet techniques solve the optical flow constrain equation [10]

$$\vec{v} \cdot \vec{\nabla} I(\vec{r},\ t) + \frac{\partial I(\vec{r},\ t)}{\partial t} = 0 \tag{30}$$

for finding a motion vector $\vec{v}$ on the position $\vec{r}$. For solving that, some additional constrains must be introduced. For example, a Horn-Schunck method minimizes the square of the optical flow gradient magnitude

$$\left(\frac{\partial v_x}{\partial x}\right)^2 + \left(\frac{\partial v_x}{\partial y}\right)^2 \text{ and } \left(\frac{\partial v_y}{\partial x}\right)^2 + \left(\frac{\partial v_y}{\partial y}\right)^2. \tag{31}$$

The drawback of all methods from this group are a big prediction error on moving objects boundaries caused by smoothness constrains and good accuracy only in case of dense motion field.

2.4.2 Pixel recursive techniques

This a group of recursively gradient techniques of prediction error minimization as DFD^2 (29). An example of pixel recursive method is the Netravali-Robbins method [10], which iteratively updates the displacement vector according to the formula

$$\vec{d}^{(k+1)} = \vec{d}^{(k)} - \epsilon \mathrm{DFD}(\vec{r},\ t,\ \vec{d}^{(k)}) \cdot \nabla_{\vec{r}} I(\vec{r} - \vec{d},\ t - \Delta t) \tag{32}$$

with a constant gain $\epsilon > 0$. The main drawbacks of this group of motion estimation techniques are a propensity to converge to the local instead of the global minimum and incapability to handle large displacements and motion field's discontinuities.

2.4.3 Block-matching method

Block-matching technique (described in details in the section 2.5) minimizes a disparity between macroblocks in two frames

$$\vec{d} = \arg\min_{\vec{d} \in S} \sum_{\vec{r} \in S} ||I(\vec{r},\ t) - I(\vec{r} - \vec{d},\ t - \Delta t)||. \tag{33}$$

It uses different cost functions $||x||$ and search algorithms, which was describe further. Motion estimation in way of block-matching is easy to implement and gives relatively good results, so it is widely used in a video coding.

2.4.4 Frequency domain techniques

That group of motion estimation techniques relies on the relationship between transformed to the frequency domain images (e.g. Fourier or Gabor transform) and measures a correlation factor with different phase shift (which corresponds to a translation in the image domain). That methods are quite complex and they are rather used in a signal analysis (e.g. in the synthetic aperture sonar) than in the image processing.

2.5 Block-matching motion estimation algorithms

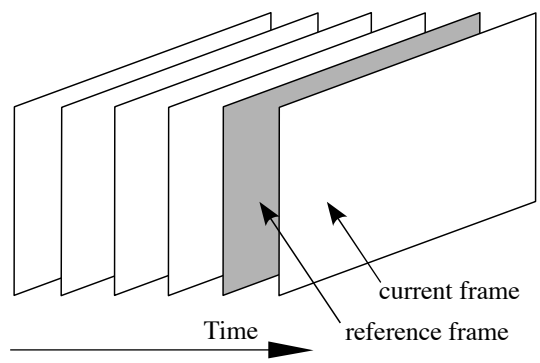
In [8] and [11] a several motion search methods based on block-matching were presented. Five of them were described in this section. The general idea of theirs operation was shown in the figure 2.5. That motion estimation process has following steps:

1. Choice of a motion vector search position $(x,\ y)$.

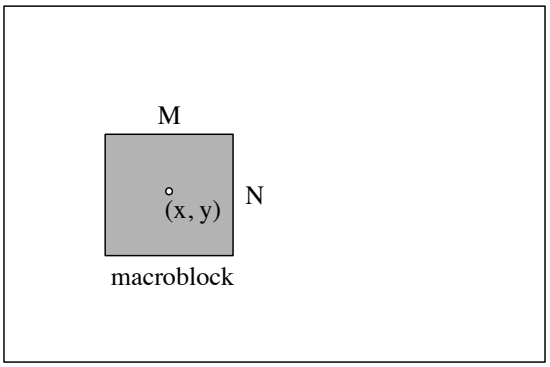
2. Selection of a macroblock $M \times N$ pixels.

3. Choice of a search range p.

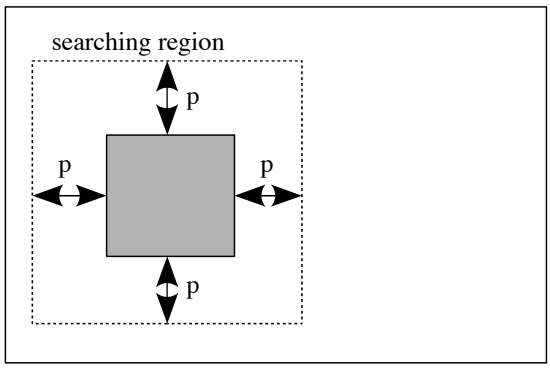
4. Search an offset $(i,\ j)$ with the best matching between the macroblock of a current frame and the corresponding macroblock of the reference frame.



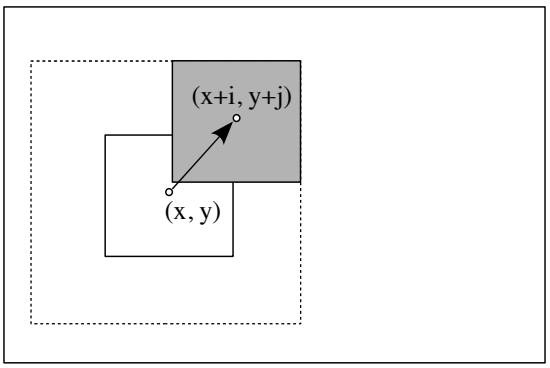
(a) Frames sequence.



(b) A macroblock selected from the current frame.



(c) A search region in the reference frame.



(d) A motion vector $(i,\ j)$ found in the best matching position.

Figure 2.5. Block-matching based motion estimation.

Macroblock's size and search range must be chose very carefully to keep a balance between the calculation complexity and the estimation accuracy. Small macroblocks (4–8 pixels) are prefered because of the smoothness constrains and the small amount of needed calculations, but the price is a reduction of the reliability of motion estimation. Large macroblock's size increases efficiency of fast motion estimation algorithms, but also the calculation intensity. Moreover, often it is impossible to match well large macroblocks in two frames. In a typical case, for video coding $M = N = 16$. Search range in video coding is usually set at $p = 6$, but in case of sport events (large-scale motion), it can be necessary to enlarge p even up to 64.

2.5.1 The Full-Search Method (FSM)

Full-search is the simplest to implementation, the most accurate, the most robust but also the most time-consuming way of finding the cost function's minimum. This

is a simple two-dimensional search done in a signle step. The figure 2.6 shows a graphical interpretation of example of such operation.

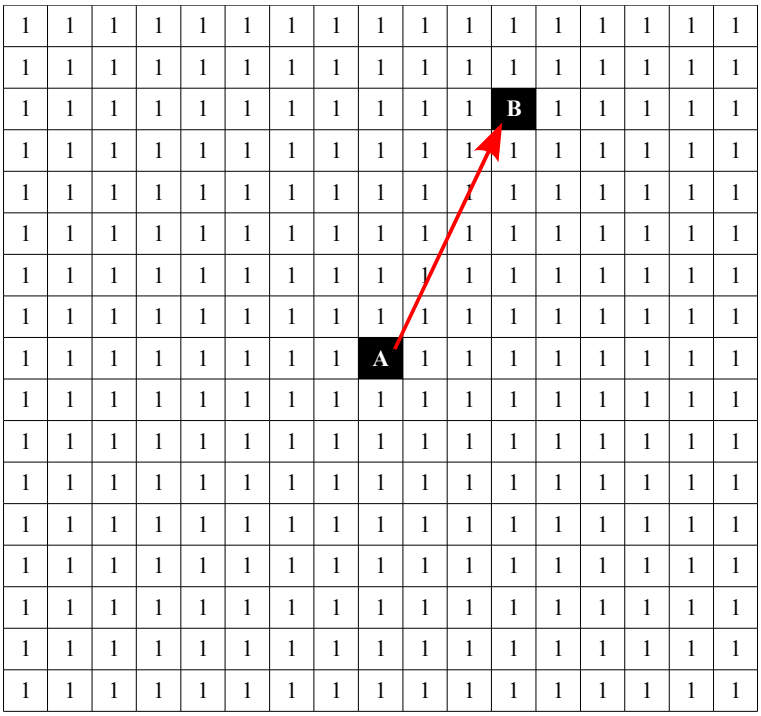


Figure 2.6. Full-Search Method – an exhausive search of the least cost function value. Only one step needed but colossal complexity. Numbers represent points processed in each step.

2.5.2 The Logarithmic Search Algorithm (LSA)

This is the first from fast search methods. The speedup is achieved by reduce a number of considered points. A basis of all logarithmic algorithms is to start in the centre of search region with a step size of $2^{\lfloor log_2 p \rfloor}$ pixels and move along the cost function's gradient with variable step. An example of two-dimensional logarithmic search was presented in the figure 2.7. It contains following stages:

Step 1 – Set a step size as $s = 2^{\lfloor log_2 p \rfloor}$ and calculate the cost function in the center of search region and four neighbor points: on the top, bottom, left and right.

Step 2 – Pick a point with the least cost function value and, if it is not a center, calculate the cost functions in two corresponding diagonal points.

Step 3 – Move (or stay) to the point with the least cost function's value from steps 1 and 2, divide the step size by 2 and calculate the cost function in four selected points neighbors, like previously.

Step 4 – Repeat step 2, but in the new starting position.

Step 5 and further – Continue division of step size by 2 and proceed until the step size is minimal or the point with cost below a threshold value is found.

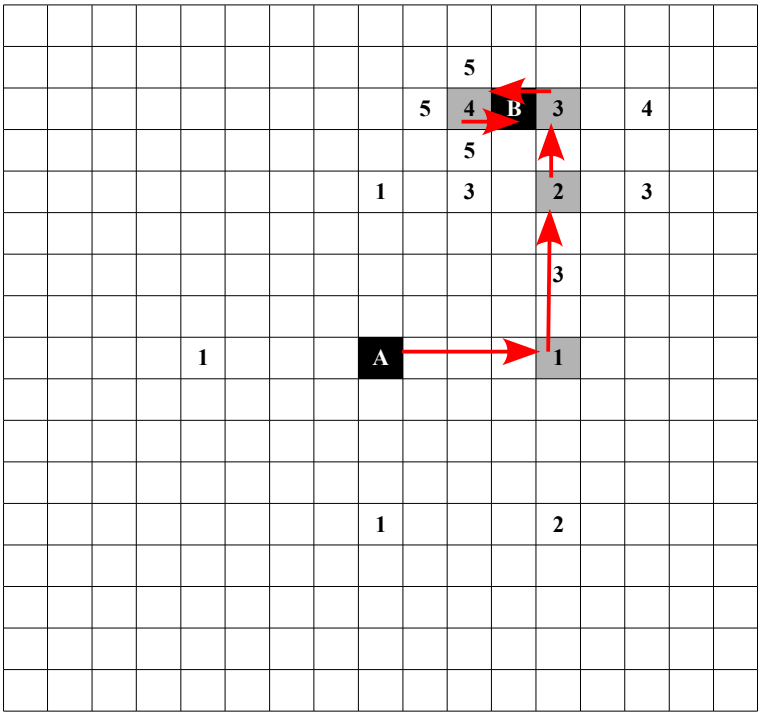


Figure 2.7. The Two-Dimensional Logarithmic Search – start in the center of search region with a step size of $2^{\lfloor log_2 p \rfloor}$ pixels and divide the step size by 2 after each turn. Numbers represent points processed in each step.

A special kind of logarithmic search is the 3-step search presented in the figure 2.8. A range of this algorithm is limited to ± 7 pixels, but in video coding is sufficient for good performance. The main difference between its operation and the classical logarithmic method is calculation of cost function's values in all eight neighbor points in one step.

2.5.3 The Parallel Hierarchical One-Dimensional Search (PHODS)

That algorithm reduces a number of search location to what is probably the absolute minimum. It bases on a logarithmic method, but the search is done independently along the two directions as follows:

Step 1 – Set a step size as $s = 2^{\lfloor log_2 p \rfloor}$, move to the center of searching region and denote the position as $(i_0,\ j_0)$.

Step 2 – In parallel, compute:

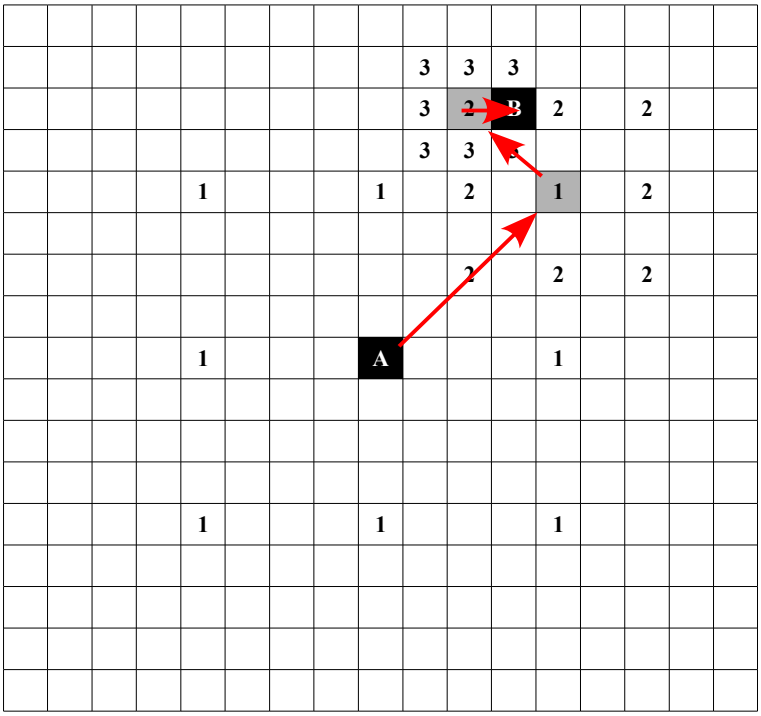


Figure 2.8. The Three-Step Logarithmic Search – start in the center of search region with a step size of 4 pixels and divide the step size by 2 after each turn. Numbers represent points processed in each step.

- i-axis local minimum – pick the best location (the least cost function's value) from $(i_0,\ j_0)$, $(i_0 - s,\ j_0)$ and $(i_0 + s,\ j_0)$, then move i_0 to this coordinate,
- j-axis local minimum – pick the best location (the least cost function's value) from $(i_0,\ j_0)$, $(i_0,\ j_0 - s)$ and $(i_0,\ j_0 + s)$, then move j_0 to this coordinate.

Step 3 – Divide s by 2 and repeat step 2 until the step size is minimal. The final $(i_0,\ j_0)$ is the motion vector with the best matching between two frames.

The figure 2.9 ilustrates the idea of such operation. In fact, the cost function value of found motion vector is not checked, so PHODS provide very poor efficiency when the cost function is not a perfect "bowl".

2.5.4 The Hierarchical Motion Estimation (HME)

This method, besides a reduction of number of searching locations, allows also reduce a number of compared pixels. There are many variation of the hierarchical search, but the generic idea was presenter in the figure 2.10. It has following stages:

Step 1 – Prepare several of subsampled copies of the current and the reference pictures.

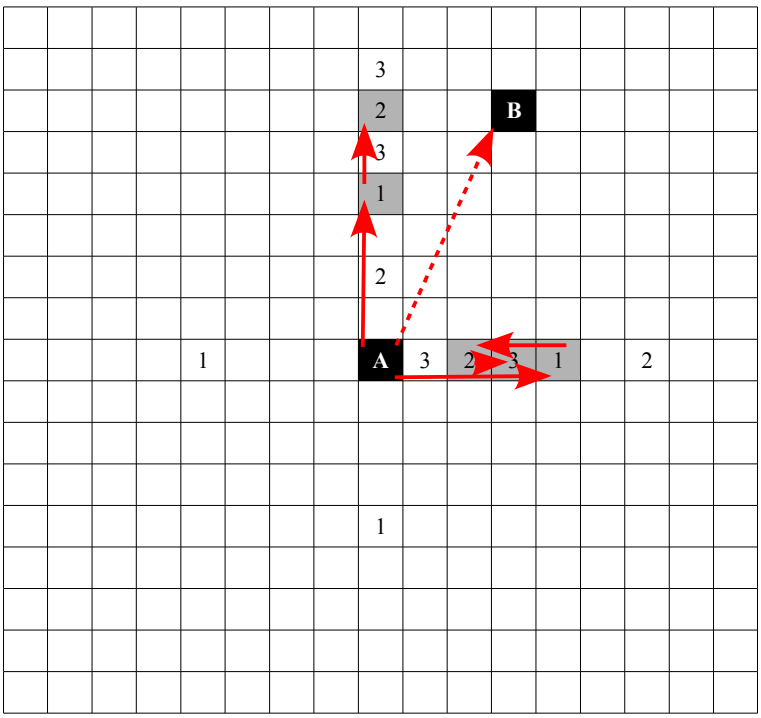


Figure 2.9. The Parallel Hierarchical One-Dimensional Search – start in the center of search region with a step size of $2^{\lfloor log_2 p \rfloor}$ pixels and proceed in two dimensions independently. Numbers represent points processed in each step.

Step 2 – Begin search at the lowest level with some fast method or even the full-search. Point $(i,\ j)$ is the best match position.

Step 3 – Continue search at next level, with the searching region ± 1 pixel, starting from the point $(2i,\ 2j)$. Denote the new best match position as $(i,\ j)$.

Step 4 and further – Repeat step 3 until the top level is reached. Final $(i,\ j)$ is the found motion vector.

2.6 Matching criteria

Each search algorithm requires a cost function which allows to rate the macroblocks matching level. In [8] and [11] several examples of cost function was described. In this section, the most important and the most widely used of them were presented. There had been assumed that R was the reference frame, C was the current frame, $(x,\ y)$ were coordinates of search position, M and N were macroblock's dimensions and $(i,\ j)$ were coordinates of the evaluated offset between frames. The best matching took place in the cost function's extreme.

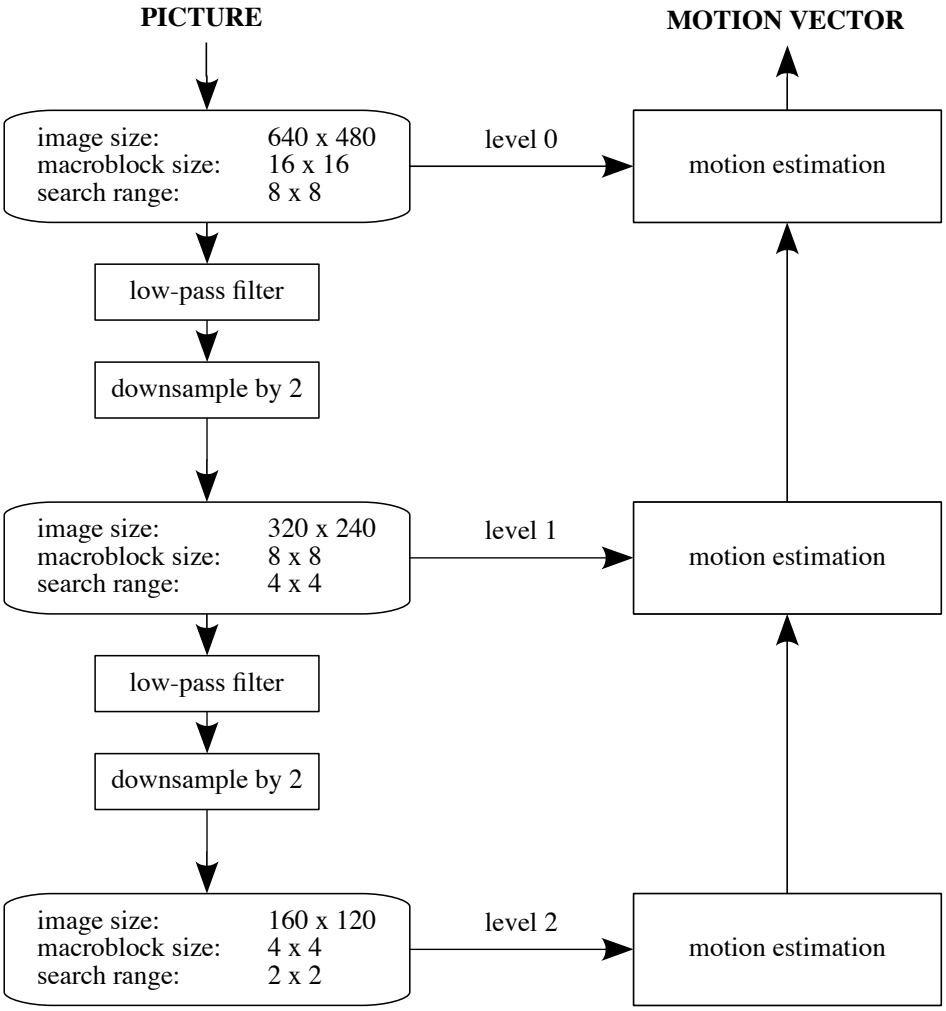


FIGURE 2.10. An example of hierarchical search – it starts with a rough motion estimation, but the accuracy increases with each step.

The Mean-Squared Difference (MSD)

This function produces a superior results. However, the main drawback of it is its high complexity. The result of that cost function may be interpreted as the Euclidean distance between macroblocks. The MSD function is defined as

$$MSD(i,j) = \frac{1}{MN} \sum_{k=-\frac{M}{2}}^{\frac{M}{2}} \sum_{l=-\frac{N}{2}}^{\frac{N}{2}} \Big[C(x+k,\ y+l) - R(x+k+i,\ y+l+j) \Big]^2. \quad (34)$$

The Mean Absolute Difference (MAD)

MAD cost function is a very popular one because of its simplicity and easy implementation. The main disadvantage is less accuracy and tendency to overemphasize small differences, which may slow convergence of fast search methods or even produce a wrong result. The formula describing this cost function is

$$MAE(i,j) = \frac{1}{MN} \sum_{k=-\frac{M}{2}}^{\frac{M}{2}} \sum_{l=-\frac{N}{2}}^{\frac{N}{2}} \Big| C(x+k,\ y+l) - R(x+k+i,\ y+l+j) \Big|. \quad (35)$$

The Cross-Correlation Function (CCF)

The cross-correllation cost function uses a statistical instruments. With assumption that macroblocks in both frames are sets of random variables, the dependency between macroblocks in two frames is equal to

$$CCF(i,j) = \frac{Cov(C_{x,y}, R_{x+i,y+j})}{Var(C_{x,y}) Var(R_{x+i,y+j})}. \quad (36)$$

The Pixel Difference Classification (PDC)

To reduce the computational complexity of cost's calculation, one plain block matching criterion was proposed by Gharavi and Mills in 1990. The Pixel Difference Classification simple counts matching pixel between two macroblocks. The function is defined as

$$PDC(i,j) = \sum_{k=-\frac{M}{2}}^{\frac{M}{2}} \sum_{l=-\frac{N}{2}}^{\frac{N}{2}} T_{i,j}(k,\ l) \quad (37)$$

where

$$T_{i,j}(k\ ,l) = \begin{cases} 1 & \text{if } \Big| C(x+k,\ y+l) - R(x+k+i,\ y+l+j) \Big| \leq t \\ 0 & \text{otherwise} \end{cases}. \quad (38)$$

The Binary Level Matching Criterion (BPROP)

The difference between two pixels can be expressed not only by an aritmetical defference, but also by exclusive disjunction of corresponding bits. Such approach is a background of BPROP cost function. This matching criterion is defined as

$$BPROP(i,j) = \sum_{k=-\frac{M}{2}}^{\frac{M}{2}} \sum_{l=-\frac{N}{2}}^{\frac{N}{2}} \mathbf{xor}\Big[\, C(x+k,\ y+l),\ R(x+k+i,\ y+l+j)\,\Big]. \quad (39)$$

This function can be modified by preliminary tresholding compared values. In that way, only the most significant bits are compared and small differences between macroblocks are neglected. Experimental results showed that the threshold of 16 gives the lowest error in motion search [8]. It can be easily realized by shifting compared values right by 4 bits.

2.7 Sub-pixel accurate motion estimation

A block matching based methods of motion search allows only for integer-pixel accuracy. To obtain sub-pixel resolution, one of compared images could be translated by less that one pixel and interpolated. After that, each of classical block matching methods can be applied and the result is a sum of the integer accuracy detected motion and the sub-pixel translation. That approach is very computation and time consuming. For a fast sub-pixel motion estimation, without the interpolation, one of dedicated algorithms can be applied. An example was presented in [8]:

1. At first, any regular method must be applied to find the best matching point with integer-pixel accuracy.

2. Then the MAD function in the found points neighborhood must be calculated. The figure 2.11 shows points with MAD values denoted as m_0 to m_4.

3. The MAD function in the found points neighborhood is approximated by a function of the form $p(i) = a|i - b| + c$ in both directions. Coefficients a, b and c can be calculated using three points: $p(x) = m_0$, $p(x-1) = m_3$ and $p(x+1) = m_4$ in the horizontal axis and $p(y) = m_0$, $p(y-1) = m_1$ and $p(y+1) = m_2$ in the vertical one.

4. To yield the smallest MAD value on the half-pixel grid in the i direction, one of following conditions must be fulfilled:

   - if $2(p(i-1)-p(i)) < (p(i+1)-p(i))$, the i coordinate should be decreased by 0.5,

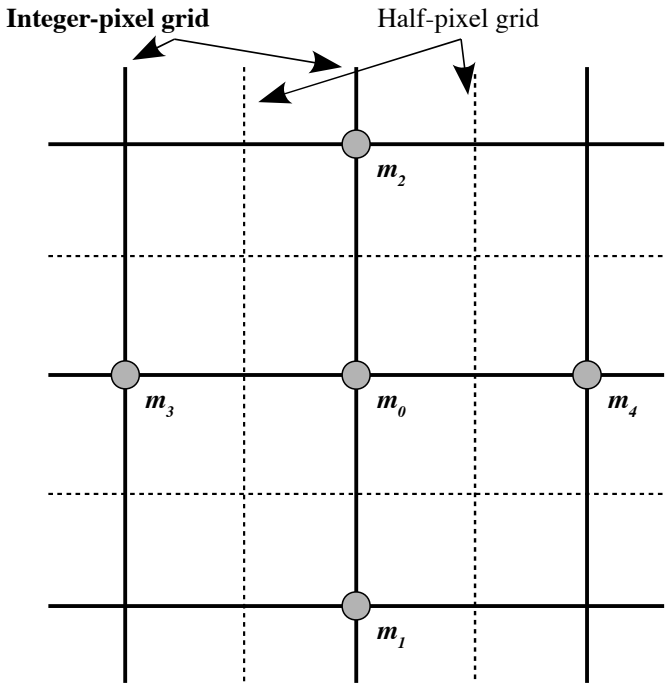


Figure 2.11. Fast method for half-pixel accurate motion vector estimation – MAD function values m_0 to m_4 in integer-grid points allows to find the best matching vectors position on integer and half-pixel grid, resulting the half-pixel accuracy.

- if $(p(i-1)-p(i)) > 2(p(i+1)-p(i))$, the i coordinate should be increased by 0.5.

This method requires some extra calculations, but is still much faster than ones based on the image interpolation. The main drawback is limited accuracy, but in many cases the half-pixel precision is sufficient.

CHAPTER 3
STABILIZATION FILTER DEVELOPMENT

The goal of a video stabilization filter's operation was to make image changes smooth, without any shakes or jerks. To simplify the stabilization algorithm, some assumptions was made:

1. The difference between two consecutive frames can be expressed as the 2D transformation.

2. The transformation between two frames can be expressed as an isometry (only rotation and translation coefficients).

$$T = \begin{bmatrix} \cos\alpha & -\sin\alpha & tx \\ \sin\alpha & \cos\alpha & ty \\ 0 & 0 & 1 \end{bmatrix} \tag{40}$$

3. Shakes are transient, much faster than directed camera moves.

A preliminary implementation of required functions took place in MATLAB because of simplicity of debugging and a large set of in-bulid signal processing functions, that can be used as a reference. A final implementation was made as a video post-processing filter in FFmpeg, because of fast operation and easy access to decoded video frames as pointers to the image buffers. The FFmpeg library was chosen also because of its popularity (e.g. MPlayer and VLC video players incorporate the FFmpeg) and multi-platform implementation. Therefore, the developed anti-shake filter had very wide usage area and could be easily used as a part of more complex projects.

3.1 Block diagram of anti-shake chain

The video stabilization chain was shown in the figure 3.1. At first, the motion field had to be calculated. The next step was a transformation matrix estimation. That

matrix describes the displacement between two consecutive frames. In way of low-pass transformation filtering, the directed camera move was separated from random shakes. Undirected moves had been removed and the current frame was transformed for the best fitting to a previous one, preserving the smooth camera move.

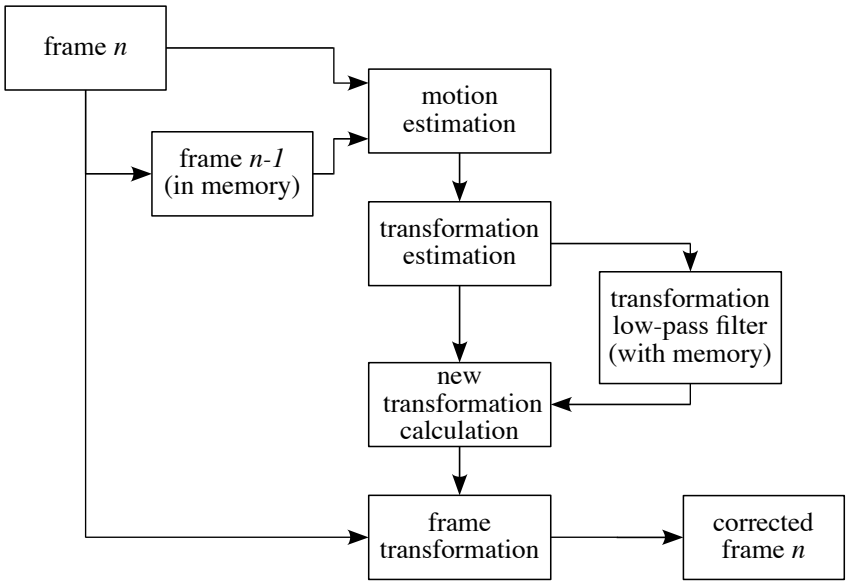


Figure 3.1. A block diagram of the stabilization chain.

3.2 Real camera shakes measurement

The first step in the stabilization filter development process was an investigation of real shakes properties. Two video sequences was taken into a consideration:

- a take of three-points constellation,
- a video from the sledge ride.

Both videos were made by Sony WX-1 camera. Measurement results were illustrated in figures 3.2 and 3.3. In the first case, measurements were done by points' position detection in the image coordinates domain. That approach provides very accurate results, but is not always applicable. The observation time was equal to 30 seconds (900 frames). The image's center position and the camera's horizon angle were measured. In the second video case, the measurement was done by estimate the global transformation between consecutive frames. Only 90 frames was taken into a consideration due to very time consuming data processing in the MATLAB environment, where this calculations took place. In this case coefficients of matrix (40) were estimated. The sine and the cosine of the rotation angle had been affected by the scaling factor, what made the angle calculation impossible. Due to this, the sine and the cosine values were processed independently.

Table 3.1. Real shakes measurement results for the three-points constellation – only muscle and breathing shakes.

SteadyShot enable	
Maximal change of the rotation angle Maximal change of the horizontal displacement Maximal change of the vertical displacement	0.17° 3.82 px 1.03 px
SteadyShot disable	
Maximal change of the rotation angle Maximal change of the horizontal displacement Maximal change of the vertical displacement	0.44° 5.82 px 3.11 px

Table 3.2. Real shakes measurement results for the sledge ride sequence.

Maximal change of the horizontal placement Maximal change of the vertical placement Maximal change of a sine of the rotation angle Maximal change of a cosine of the rotation angle	23.16 px 16.06 px 0.0253 0.0035

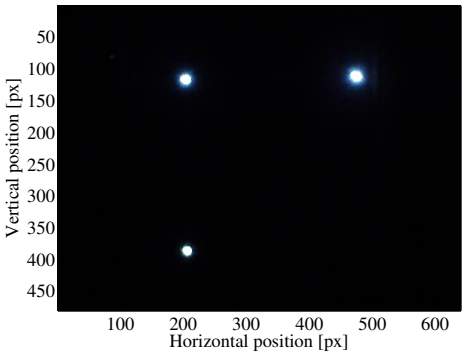
The only source of shakes in the first sequence were muscles contraction and the breathing. Measurement results were collected in the table 3.1. The shakes' amplitude was not big (up to 6 pixels). It was easy to notice that the optical stabilization SteadyShot avalible in the camera damped very fast shakes but it could not handle the slower ones. The camera move was still not smooth.

Shakes in the second video were much larger – the shakes amplitude reached even 23 pixels (results in the table 3.2). To achieve a good performance, video stabilization filter had to deal with such distortions.

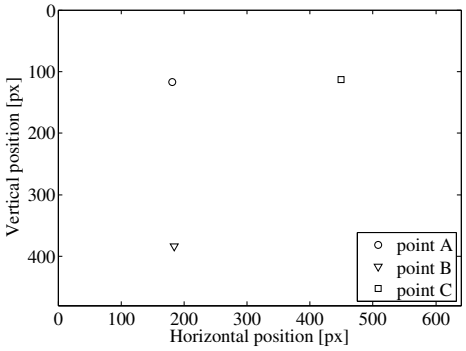
3.3 Rolling shutter distortions

Large part of cameras is equipped with CMOS instead of CCD sensors to reduce cost. Most CMOS sensors use a rolling shutter. Lines are scan sequentially in the top-to-bottom or bottom-to-top order. Due to this, when an object or the camera itself are moving, some distortions can appear. Ones caused by a the camera move along horizontal and vertical axes can be easily identified and described as a affine transformation (examples are shown in the figure 3.4). The distortion caused by an object motion or the camera tipping to the side can be very complicated [12].

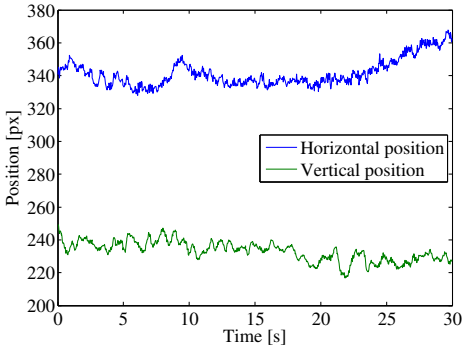
In this project the rolling shutter distortions were not considered, which may have introduce some wobbles in the output picture. However, even small inaccuracy



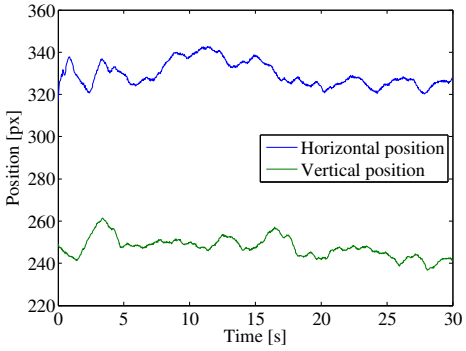
(a) Real frame of the movie.



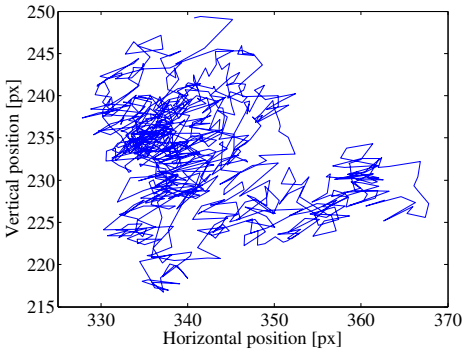
(b) Detected points layout.



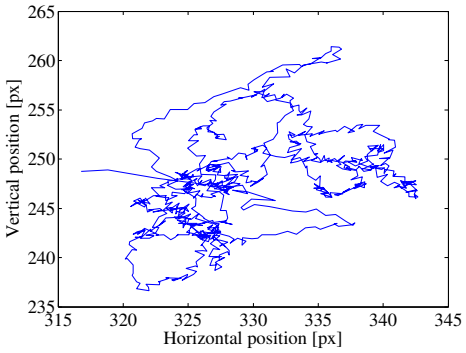
(c) Horizontal and vertical position of an image center in function of time – SteadyShot disable.



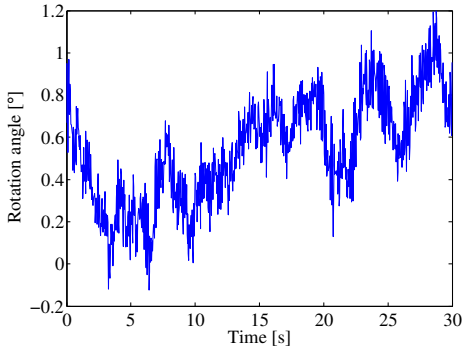
(d) Horizontal and vertical position of an image center in function of time – SteadyShot enable.



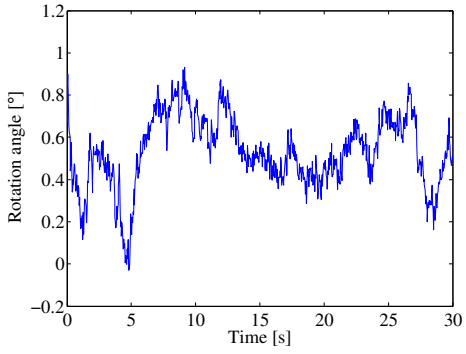
(e) Image center's move on a XY plane – SteadyShot disable.



(f) Image center's move on a XY plane – SteadyShot enable.

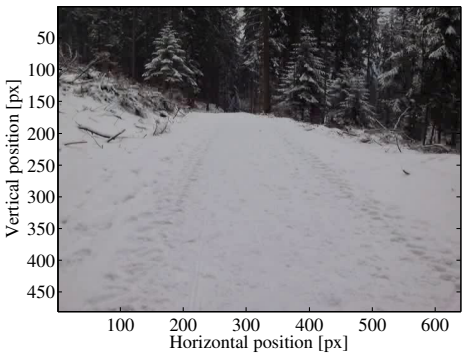


(g) Constellation rotation in function of time – SteadyShot disable.

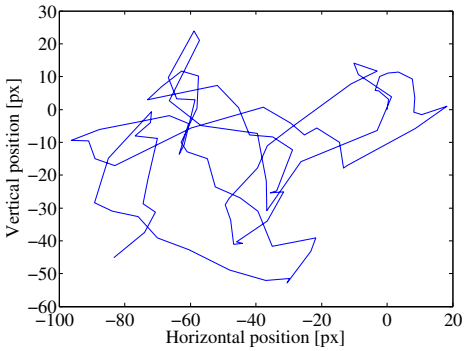


(h) Constellation rotation in function of time – SteadyShot enable.

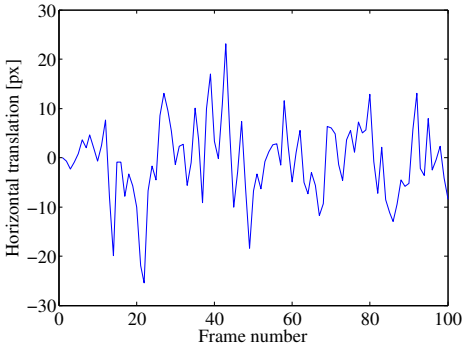
Figure 3.2. Real hand-held camera shakes measurement – three-points constellation.



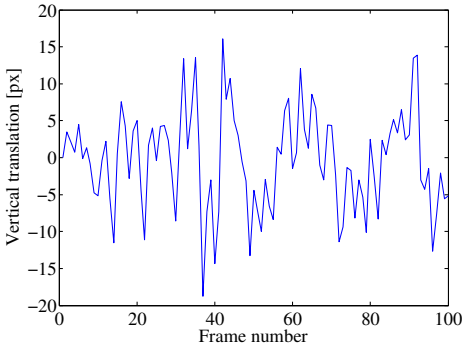
(a) Real frame of the movie.



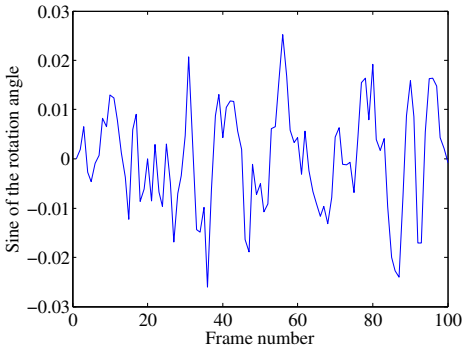
(b) Frame's center move on a XY plane.



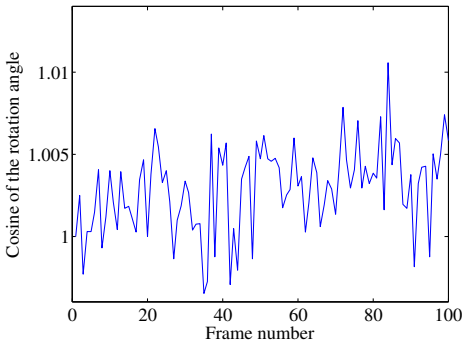
(c) Horizontal translation in function of time.



(d) Vertical translation in function of time.

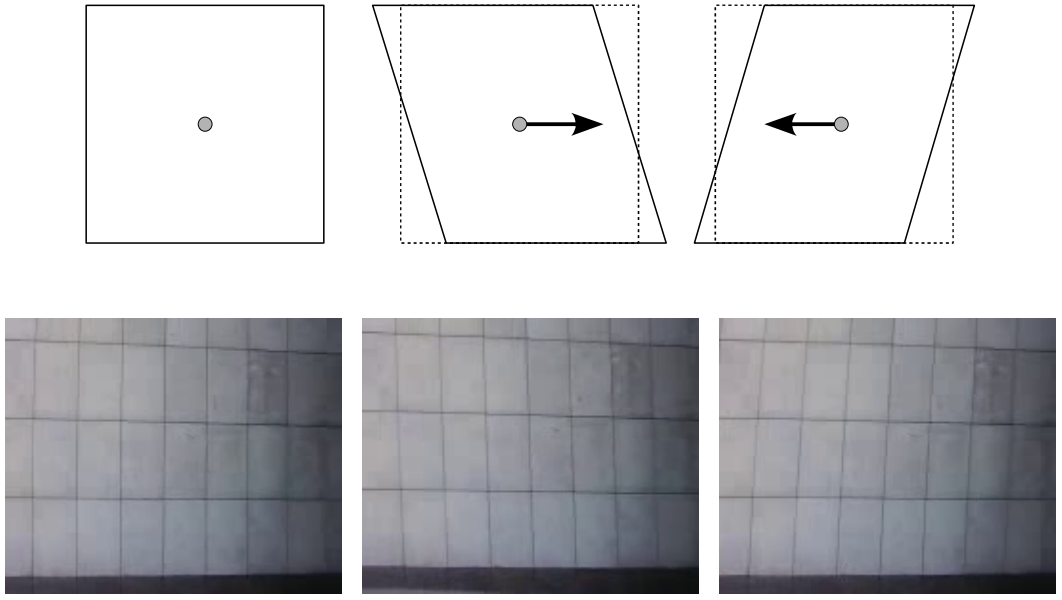


(e) Rotation angle sine value in function of time.

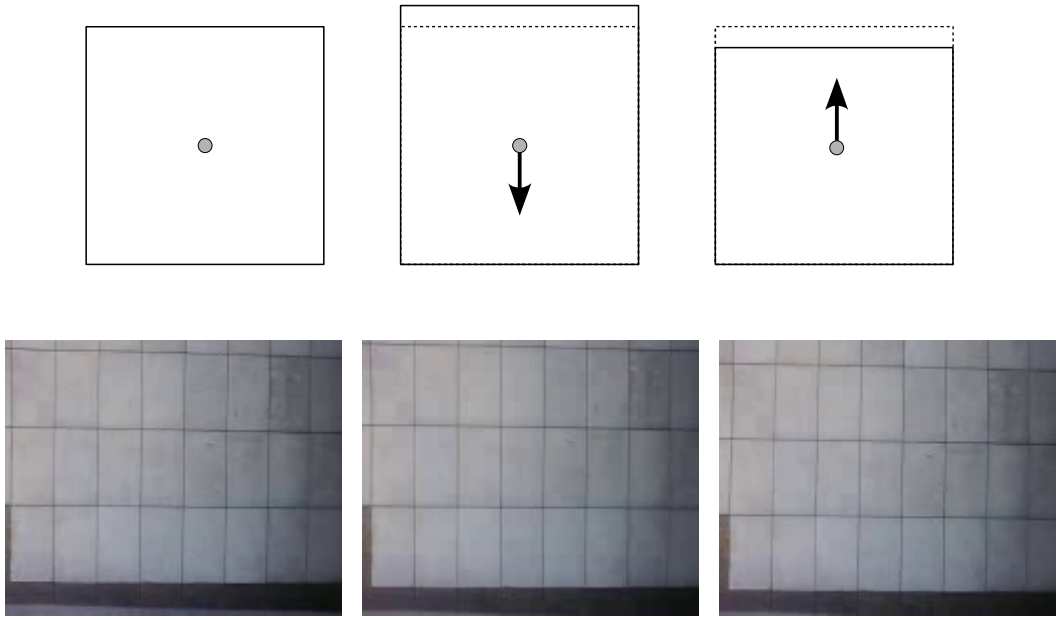


(f) Rotation angle cosine value in function of time.

Figure 3.3. Real hand-held camera shakes measurement – "sledge" test video sequence.



(a) Skew caused by fast camera pan or truck moves.



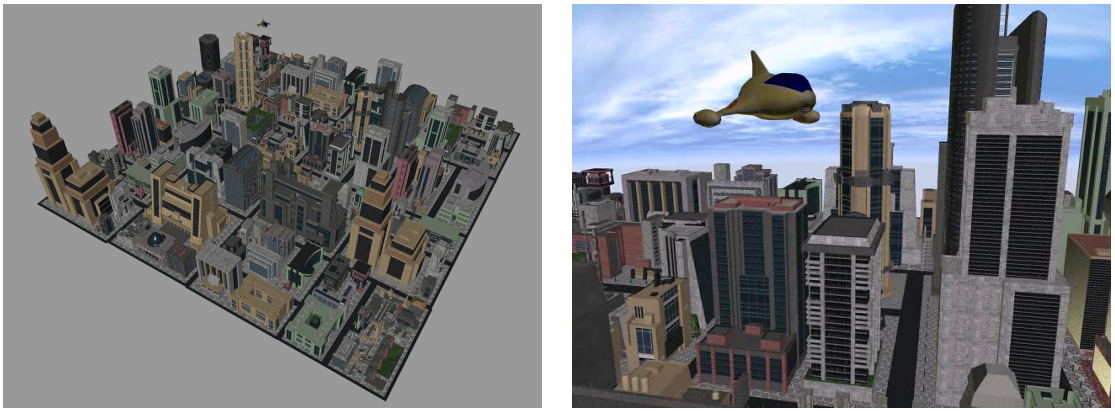
(b) Variable vertical scaling factor caused by camera tilt or pedestal moves.

Figure 3.4. Rolling shutter distortions caused by camera moves.

in affine transformation's coefficients estimation may have cause large image deformation, so a simpler similarity transformation was more robust and, all in all, gave better results.

3.4 Artificial 3D environment created in Anim8tor

The best way to achieve fully controlled camera moves without any sophisticated equipment was to create an artificial 3D environment. One had been created basing on free city buildings models from `http://www.daz3d.com/`, with usage of a free program for 3D models edition, animation and rendering – Anim8tor. The created environment was quite complex in order to emulate the real world as good as it was possible. An overview of it was shown in the figure 3.5. Rendered scenes ware included in the attached CD-ROM.



(a) Isometric projection. (b) First person look.

Figure 3.5. The 3D environment for fully controlled camera takes.

3.5 Preliminary MATLAB implementation

The preliminary implementation of chosen algorithms was made in MATLAB because of easy debugging and data access. The performance of developed functions was far form real-time operation, but they were a good background for the final implementation in the C language.

3.5.1 Existing motion vectors utilization

In some videos there are included motion vectors, but it is not a general rule – for example, in the M-JPEG video encoded there are no any. Moreover, even in case of motion compensation based encoders, some frames (I-pictures) does not have any information about motion vectors or refers either to the next, not only the previous

frame (B-pictures). So the motion vectors should have been estimated anew. In this project the block-matching method was used because its of implementation simplicity and accurate results for large displacements.

3.5.2 Comparison of different matching criterion

The cost functions are macroblocks matching criterion. Four of them were implemented:

- The Mean-Squared Difference (MSD),
- The Mean Absolute Difference (MAD),
- The Modified Pixel Difference Classification (PDC),
- The Binary Level Matching Criterion (BPROP).

The Pixel Difference Classification in a classical form has its maximum when the matching is the best. For the cost functions' unification (a minimum search), following modification was made

$$PDC(i,j) = \sum_{k=-\frac{M}{2}}^{\frac{M}{2}} \sum_{l=-\frac{N}{2}}^{\frac{N}{2}} T_{i,j}(k,\ l) \tag{41}$$

where

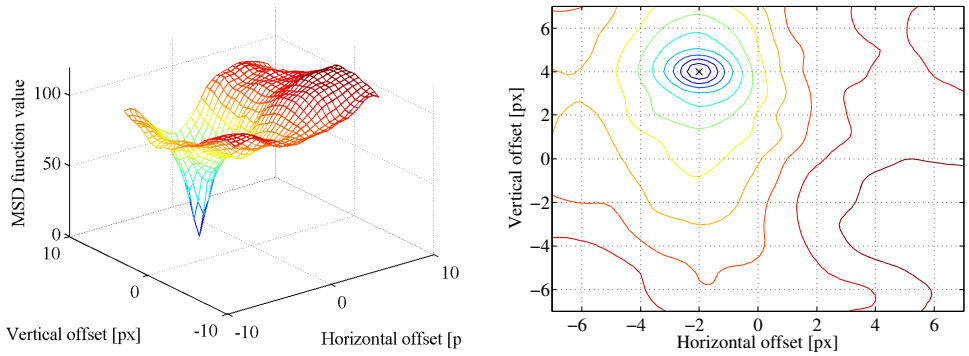
$$T_{i,j}(k\ ,l) = \begin{cases} 0 & \text{if } \left|C(x+k,\ y+l) - R(x+k+i,\ y+l+j)\right| \leq t \\ 1 & \text{otherwise} \end{cases} . \tag{42}$$

Different cost functions' shapes for the real video were shown in the figure 3.6. The MSD function had the best marked minimum, but its calculation was also the most exhaustive. The fastest was the BPROP function and it also had provided a sufficient accuracy, so that one was used for motion vectors estimation in a final stabilization filter implementation.

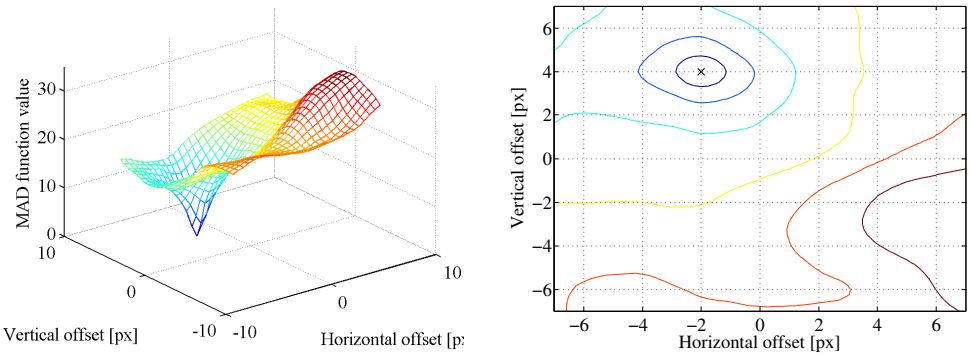
3.5.3 Comparison of different motion searching algorithms

Four of the motion searching algorithms were implemented:

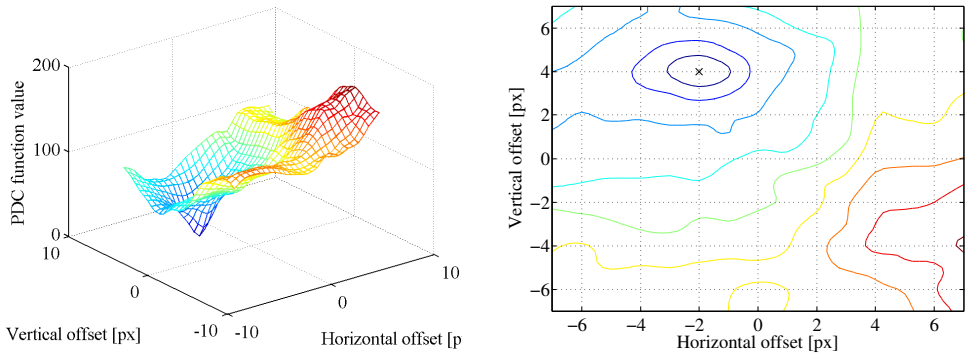
- The Full-Search Method (FSM),
- The Logarithmic Search Algorithm (LSA),
- The Parallel Hierarchical One-Dimensional Search (PHODS),
- The Hierarchical Motion Estimation (HME).



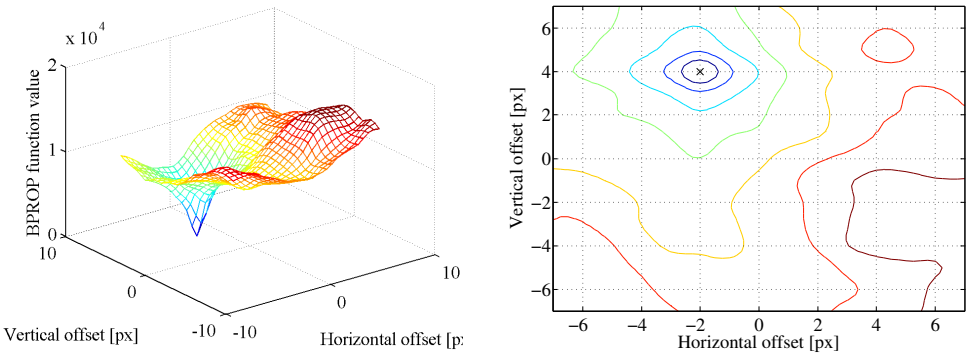
(a) The Mean-Squared Difference (MSD).



(b) The Mean Absolute Difference (MAD).



(c) The Modified Pixel Difference Classification (PDC).



(d) The Binary Level Matching Criterion (BPROP).

Figure 3.6. Exemplary shapes of different cost functions – search point $(170, 120)$ in first two frames of "sledge" video sequence ($\times$ is the known best matching position).

The comparison of estimation results for a real image was shown in figures 3.7 and 3.8. The image was transformed with known transformation parameters for the noise and distortions elimination.

The best results were provided by the full-search but it were also the most time consuming. The PHODS method was very inaccurate in case of large displacments. LSA and HME had provided very good results and had operated fast, so this two were taken into a consideration in a real-time implementation.

3.5.4 Edges detection

For search points' preselection an edge detection filter (43) described in [13] was used. Points with contrast lower than 20% were omitted, because it had beed very probable that the result in this locations may have been inaccurate. Also motion vectors on the edge of search region were not taken into a consideration. Examples of image's contrast thresholding were shown in the figure 3.9.

$$\begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & 24 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix} \tag{43}$$

3.5.5 Transformation estimation

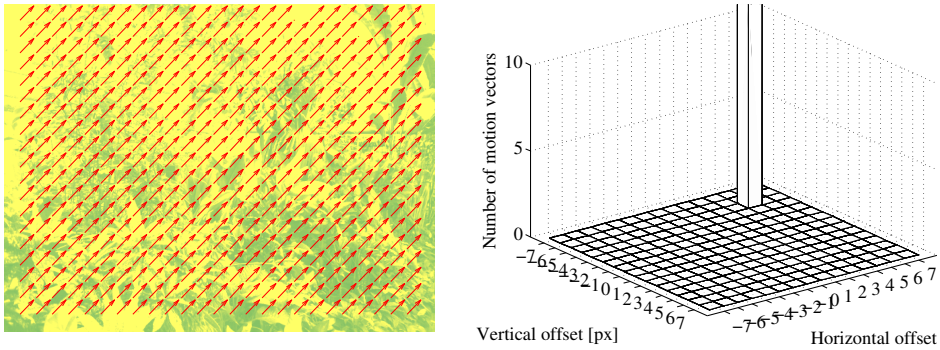
The first step of a robust transformation estimation was the RANSAC algorithm. The transformation (40) was expressed as a set of linear equations

$$\begin{aligned} x' &= x\cos\alpha - y\sin\alpha + t_x \\ y' &= x\sin\alpha + y\cos\alpha + t_y \end{aligned} \tag{44}$$

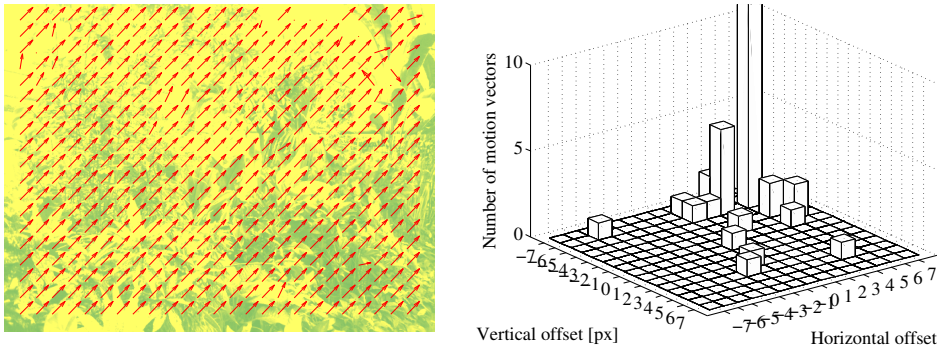
so the transformation's parameters could be evaluated from two motion vectors $(x_1,\ y_1,\ x_1',\ y_1')$ and $(x_2,\ y_2,\ x_2',\ y_2')$ with the formula

$$\begin{aligned} \sin\alpha &= \frac{(x_1'-x_2')(y_1-y_2)-(x_1-x_2)(y_1'-y_2')}{(y_2-y_1)^2-(x_1-x_2)^2} \\ \cos\alpha &= \frac{(y_1'-y_2')-(x_1-x_2)\sin\alpha}{(y_1-y_2)} \\ t_x &= \frac{x_1'+x_2'-\cos\alpha(x_1+x_2)+\sin\alpha(y_1+y_2)}{2} \\ t_y &= \frac{y_1'+y_2'-\sin\alpha(x_1"x_2)-\cos\alpha(y_1+y_2}{2} \end{aligned} \tag{45}$$

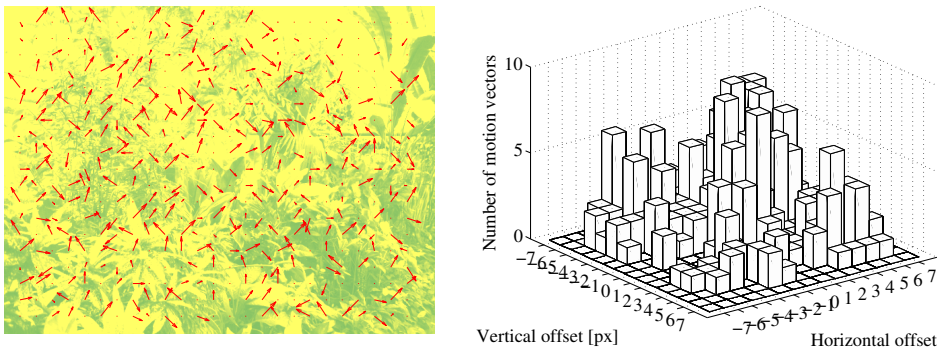
For re-estimation the transformation, based only on inliners, the Newton's method [1] was used. For the transformation matrix (40) and the inliners set



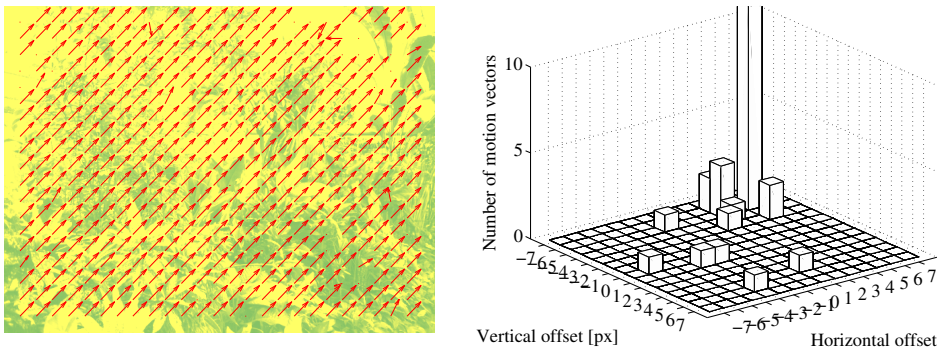
(a) The Full-Search Method (FSM).



(b) The Logarithmic Search Algorithm (LSA).

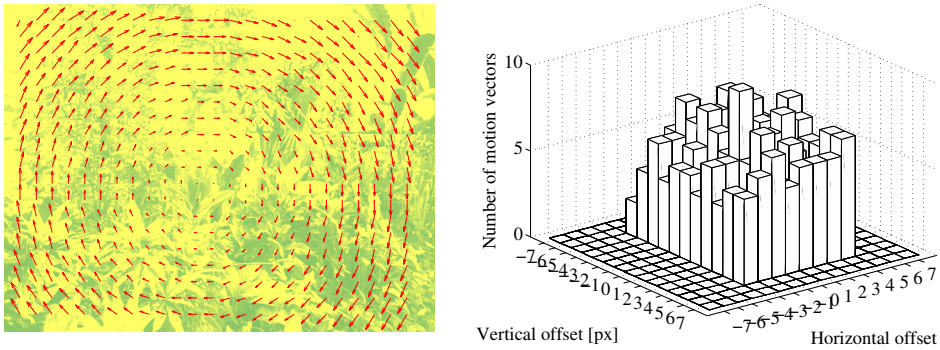


(c) The Parallel Hierarchical One-Dimensional Search (PHODS).

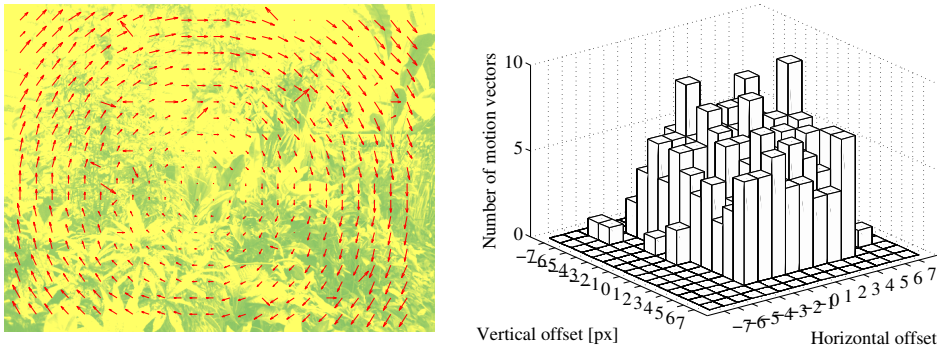


(d) The Hierarchical Motion Estimation (HME).

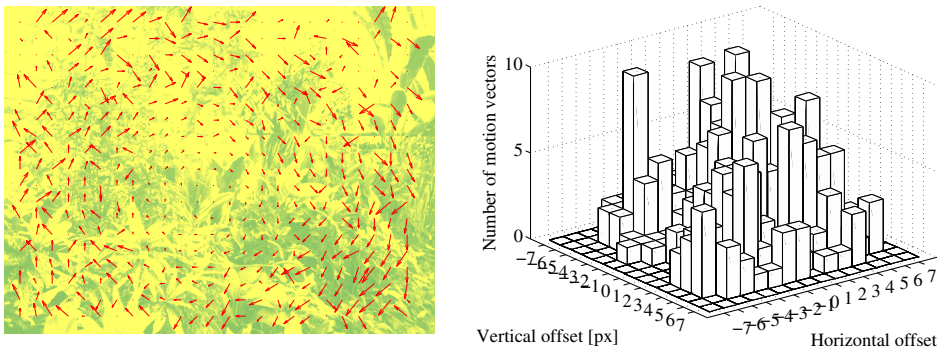
Figure 3.7. Motion vectors field and vectors' historgams for the real image's translation $tx = 5$, $ty = -5$.



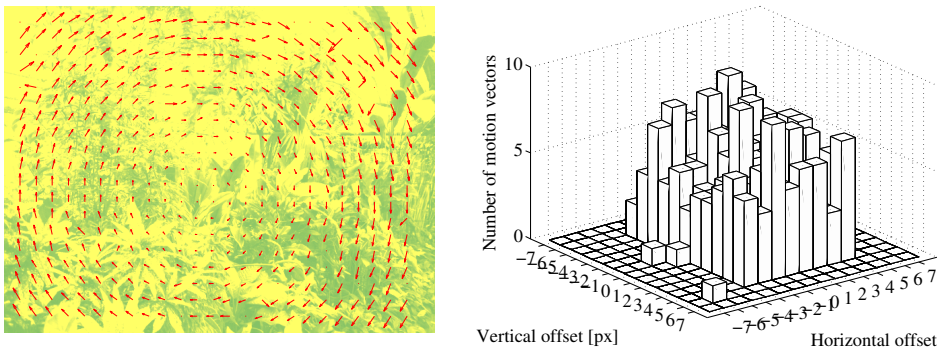
(a) The Full-Search Method (FSM).



(b) The Logarithmic Search Algorithm (LSA).



(c) The Parallel Hierarchical One-Dimensional Search (PHODS).



(d) The Hierarchical Motion Estimation (HME).

Figure 3.8. Motion vectors field and vectors' historgams for the real image's rotation by an angle $\alpha = 1°$.

(a) Original frames.

(b) Edge detection with the treshold value 20%.

(c) Edge detection with the treshold value 50%.

(d) Edge detection with the treshold value 80%.

Figure 3.9. Edges detection results for real pictures.

$(x_i,\ y_i,\ x'_i,\ y'_i)$, the total error could been expressed as

$$Q = \frac{1}{2}\sum_i \left[(x'_i - x_i\cos\alpha + y_i\sin\alpha - t_x)^2 + (y'_i - x_i\sin\alpha - y_i\cos\alpha - t_y)^2\right] \quad (46)$$

The total error was minimal, when all derivatives $\frac{\partial Q}{\partial x_i}$ were equal to zero. Therefore, function

$$\begin{bmatrix} \frac{\partial Q}{\partial \sin\alpha} \\ \frac{\partial Q}{\partial \cos\alpha} \\ \frac{\partial Q}{\partial t_x} \\ \frac{\partial Q}{\partial t_y} \end{bmatrix} = f\begin{bmatrix} \sin\alpha \\ \cos\alpha \\ t_x \\ t_y \end{bmatrix} = f(\mathbf{x}) \quad (47)$$

could be iteratively minimized. The Jacobian matrix had a form

$$\mathbf{J} = \begin{bmatrix} \frac{\partial^2 Q}{\partial \sin\alpha^2} & \frac{\partial^2 Q}{\partial \sin\alpha \partial \cos\alpha} & \frac{\partial^2 Q}{\partial \sin\alpha \partial t_x} & \frac{\partial^2 Q}{\partial \sin\alpha \partial t_y} \\ \frac{\partial^2 Q}{\partial \sin\alpha \partial \cos\alpha} & \frac{\partial^2 Q}{\partial \cos\alpha^2} & \frac{\partial^2 Q}{\partial \cos\alpha \partial t_x} & \frac{\partial^2 Q}{\partial \cos\alpha \partial t_y} \\ \frac{\partial^2 Q}{\partial \sin\alpha \partial t_x} & \frac{\partial^2 Q}{\partial \cos\alpha \partial t_x} & \frac{\partial^2 Q}{\partial t_x^2} & \frac{\partial^2 Q}{\partial t_x \partial t_y} \\ \frac{\partial^2 Q}{\partial \sin\alpha \partial t_y} & \frac{\partial^2 Q}{\partial \cos\alpha \partial t_y} & \frac{\partial^2 Q}{\partial t_x \partial t_y} & \frac{\partial^2 Q}{\partial t_y^2} \end{bmatrix}. \quad (48)$$

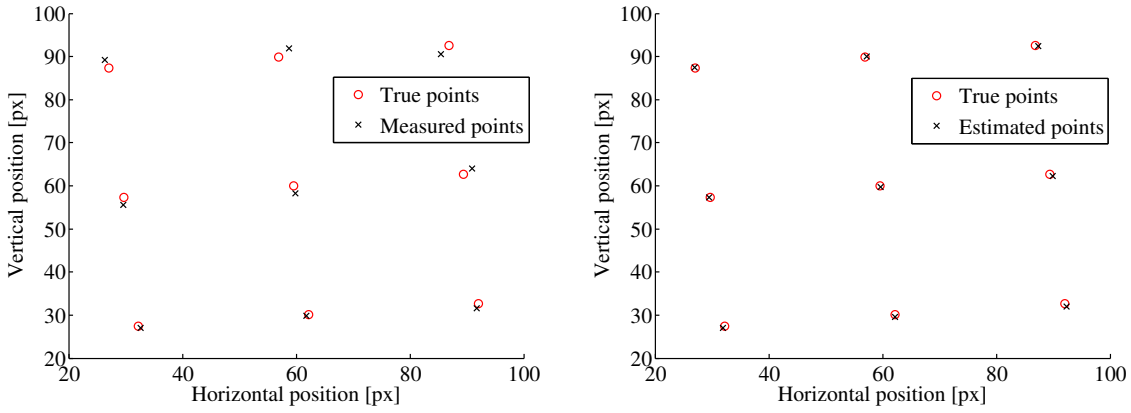
The parameters' correction in the i^{th} iteration was equal to

$$\Delta_i = -\mathbf{J}^{-1} f(\mathbf{x}_i) \quad (49)$$

and the parameters vector $\mathbf{x}$ in the i^{th} iteration was expressed as

$$\mathbf{x}_i = \mathbf{x}_{i-1} + \Delta_i. \quad (50)$$

An illustration of Newton's estimation results, in presence of a random noise, was shown on the figure 3.10.



(a) Measurement results. (b) Newton's estimation.

FIGURE 3.10. Results of Newton's transformation estimation – total error reduction from 29.97 to 2.13 (in case of 9 points).

3.5.6 Moving average motion filtering

The moving average (rectangular) filter is an optimal filter for reducing the random noise with preserving sharp step response. It has the best properties in the time domain, but the worst in the frequency domain [14]. It is also very easy to implement, so it seems to be a good choice for camera motion smoothing. Several of moving average filters was implemented. All of them was based on one-side filtering, because in the real-time operation samples from the future are not available. Some phase shift in the camera motion during stabilization was acceptable, in opposite to the time delay. As an input data the horizontal displacement of frame's center in the "sledge" video sequence was used. The filter had a length of 8 frames.

The figure 3.11 shows filtering results. A filter with decreasing samples' weights (figure 3.11(b)) is not a classical moving average filter, but it was shown as a reference. Filtration had provided better results for two-pass filtering, so that technique was implemented in the final stabilization filter. The motion estimation gave informations only about the displacement between frames and the subject of stabilization is the absolute camera position. Experiments had shown that differences filtration gave almost the same results as the possition filtration (figure 3.11(f)), so that approach could be successfully used.

3.5.7 Image transformation

Image transformation was implemented in two ways:

1. Based on the equation (10) – the source image had been scanned and each pixel $(x,\ y)$ was moved on its new position $(x',\ y')$. That approach caused artifacts in the shape of empty "holes" due to rounding errors. That effect was shown in the figure 3.12(b).

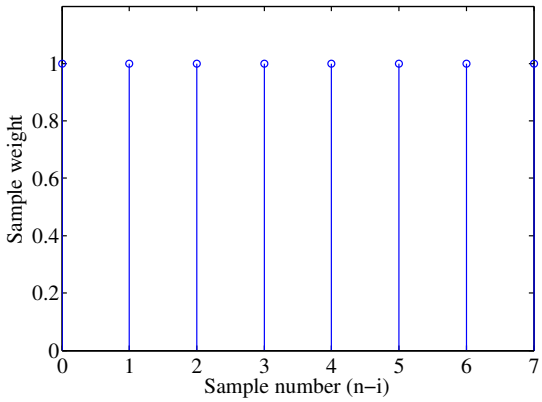
2. Based on the inverted transformation matrix (51) – the outcome image had been scanned and each pixel $(x',\ y')$ was filled with the one from the position $(x,\ y)$ in the source image. Then, there was no possibility to miss any point.

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \mathbf{T}_{3\times 3}^{-1} \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}. \tag{51}$$

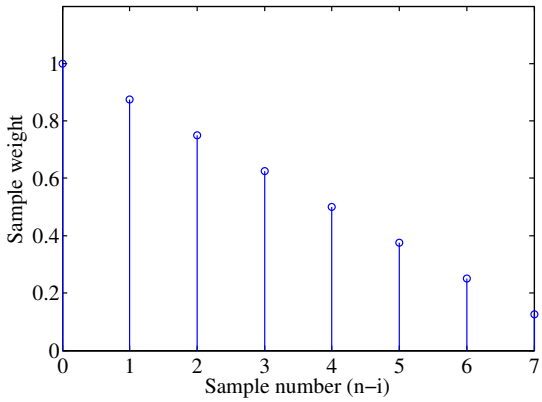
To deal with points with noninteger coordinates, some kind of interpolation technique had to be applied. Two of them were implemented:

nearest – rounding points' coordinates and get a value of the nearest neighbor,

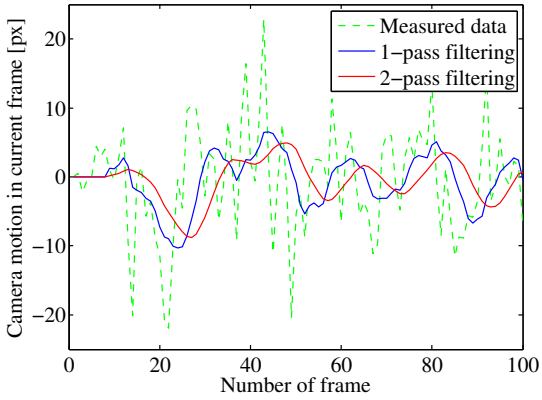
bilinear – computing the point's value using all neighbors' values and the interpolation function.



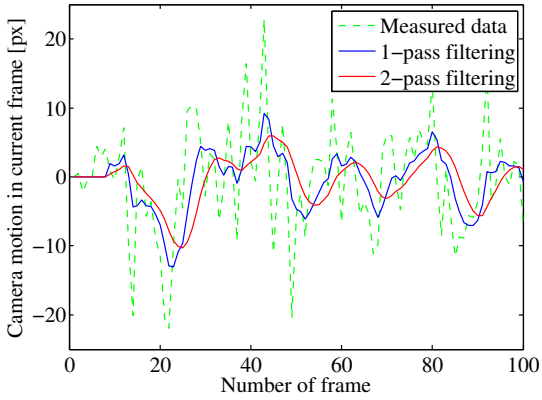
(a) Uniform samples' weights.



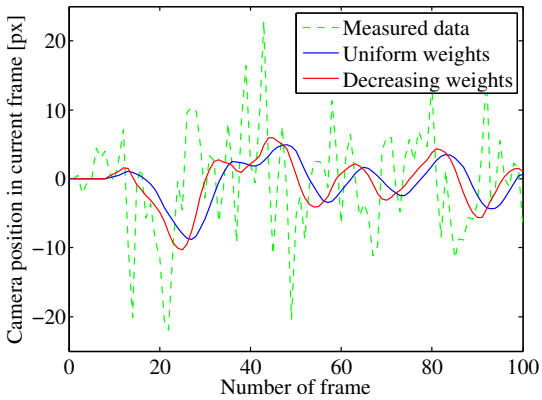
(b) Decreasing samples' weights.



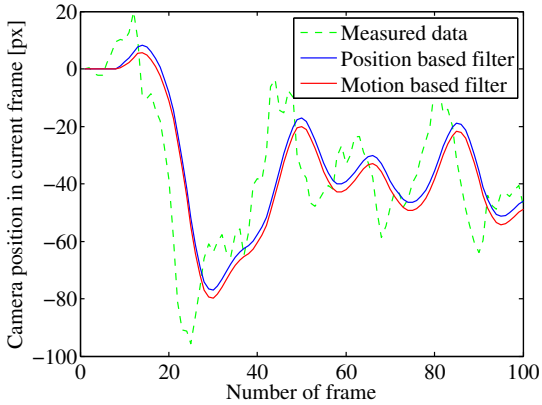
(c) Filtering with uniform weights.



(d) Filtering with decreasing weights.



(e) Comparison of results for different sample's weights.



(f) Comparison of results for differences and an absolute position filtering.

Figure 3.11. Moving average low-pass filtering results.

The bilinear interpolation was described in some details in the section 3.5.8.

3.5.8 Bilinear interpolation

The bilinear interpolation is one of ways to compute the discrete function's value in the point between known values [3]. It was used because of its simplicity in implementation and results sufficient for video processing – the frames' changing speed is large, so minor imperfections were imperceptible. In case of a two-dimmensional deiscete function, each point had four neighbors (figure 3.13). The assumption was that the function changed its values linearly for small argument's changes. Therefore

$$\begin{aligned} P(m,\ 0) &= P_{00} + (P_{10} - P_{00})m \\ P(m,\ 1) &= P_{01} + (P_{11} - P_{01})m \\ P(m,\ n) &= P(m,\ 0) + [P(m,\ 1) - P(m,\ 0)]\, n. \end{aligned} \tag{52}$$

Then, the final form of the bilinear interpolation function in that case was

$$P(m,\ n) = P_{00}(1-m)(1-n) + P_{01}(1-m)n + P_{10}(1-n)m + P_{11}mn. \tag{53}$$

Effects of a bilinear interpolation usage in the image transformation were shown in the figure 3.12.

3.6 FFmpeg implementation

FFmpeg is a complete, cross-platform set of tools for recording, playing, converting, processing and streaming audio and video. It includes a large set of useful libraries. The FFmpeg provides four command line interfaces:

ffmpeg – fast and flexible audio and video converter and grabber,

ffplay – simple and portable video player based on the SDL library,

ffprobe – tool for extraction informations about multimedia streams and files,

ffserver – server for streaming both audio and video.

Each of them use a common set of libraries, including libavfilter module [15].

3.6.1 Data structures

The main filter's structure `StabContext` included information about the filters settings, but also played the role of a filter's memory. There were stored pointers to the current and the reference frame buffers, motion vectors and inliners tables and transformation filters' histories. One primary data structure was necessary, because

(a) Original image.

(b) Illustration of ”holes” caused by rounding the pixels position. Black pixels were missed and they have the background color.

(c) Transformation with the nearest interpolation.

(d) Transformation with the bilinear interpolation.

Figure 3.12. An exemplary image transformation and its results for different transformation techniques – full image view and enlarged detail.

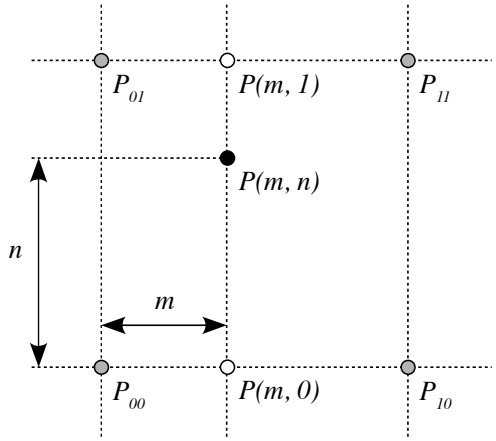


Figure 3.13. Illustration of a two-dimmensional discrete function's values grid and the point with noninteger coodrinates.

of filter's operation manner. The filter consisted of set of functions (entry points) which were called by the player or the video converter. Filter's context data structure allowed to share some region of memory between them. To made the filter's code more legible, two additional data structures were introduced: `Point` and `MotVector`. They encapsulated coordinates $(x,\ y)$ for point type objects and $(x,\ y,\ x',\ y')$ for motion vector type objects.

3.6.2 Motion estimation

As could be seen in (45), processed points could not have the same y coordinate. To avoid such a situation, not every points pair could be considered, and as result some points had to be omitted. Author had proposed a 10×8 motion search points grid, which was not a rectangular one, but slightly sheared in both directions due to ensure an unique x and y coordinate of each point. The goal of that approach was to avoid a division by the zero and to eliminate points' lost as well. Some margins had been also added to ensure the best matching point occurrence, even when the displacement between two frames were large. The example of such grid shape was shown in the figure 3.14.

For the motion estimation two techniques were implemented:

full search – search in each point in a search region with integer-pixel accuracy and the final estimation with a half-pixel accuracy,

fast search – a two-step hierarchical search – search in whole search region with a step of 3 pixels, then integer-pixel search in the best matching location in a region 3×3 pixels (figure 3.15) and the final estimation with a half-pixel accuracy.

The search region was set as ± 24 pixels from the starting location. That large range of possible motion vector placement was required to handle shakes with a large

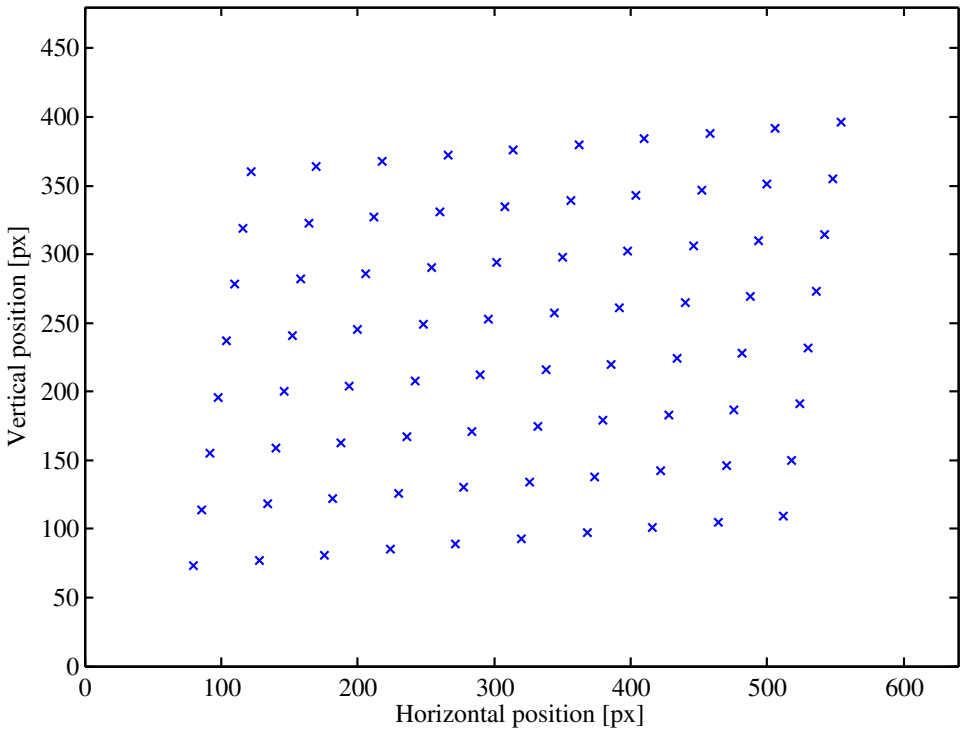


Figure 3.14. Motion search points grid in a VGA frame (640 × 480 pixels).

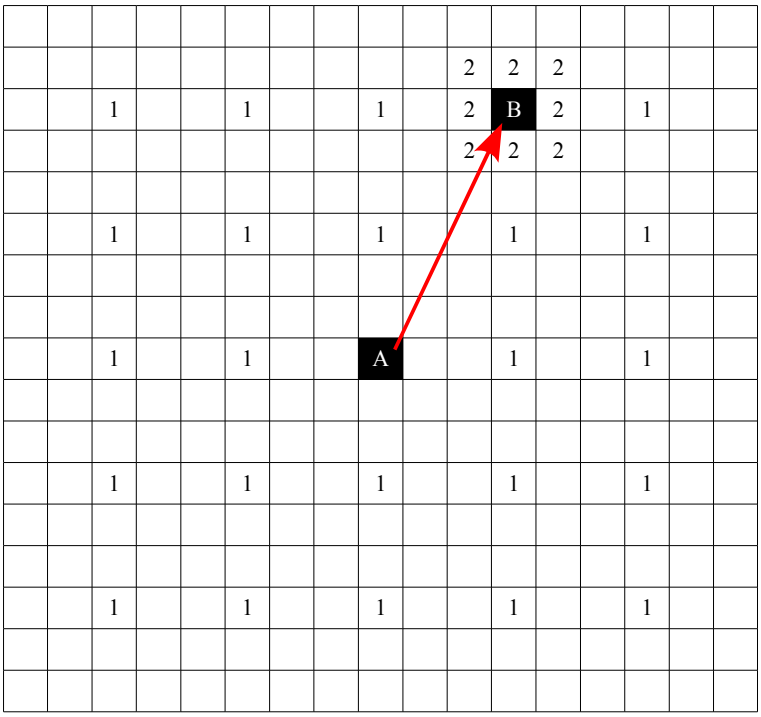


Figure 3.15. Motion search technique used in the stabilization filter. Numbers represent points processed in each step.

amplitude. Compared macroblocks had a dimmension of 16×16 pixels, but for a computational complexity reduction, they were subsampled by factor of two. As a result of that, only 64 points of each frame were used in the cost function's value calculation. As macroblocks' matching measure, the Binary Level Matching Criterion was used.

3.6.3 Transformation estimation and correction

The transformation between a current and the reference frame estimation, based on found motion vectors, was performed in two steps:

1. preliminary estimation with the RANSAC algorithm,
2. re-estimation with the Newton's method.

In the first step, from many motion vectors only inliners were selected. The second step was a minimization of total inliners displacement error for finding the best match.

In general, there is only one transformation which describes the difference between two frames. But in case of video stabilization, much more complex model had to be used. All of relations between two consecutive frames before and after transformation were illustrated in the figure 3.16. Only two of them were known:

- the reference frame's transformation T_{REF} (stored in the memory),
- the backward transformation between a current and the reference frame before its transformation T_{BACKW}.

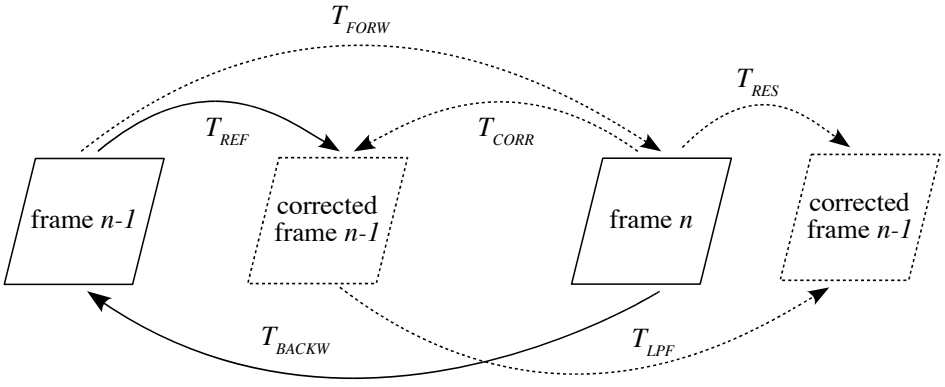


Figure 3.16. Relations between two frames transformations.

The reference frame had been also transformed for fit to a previous one, so to fit the current frame to the reference one, the correction transformation T_{CORR} had to be calculated

$$T_{CORR} = T_{BACKW} \cdot T_{REF}. \tag{54}$$

To preserve the directed camera move T_{LPF}, the forward transformation T_{FORW} between the current and the reference frame was filtered by a low-pass moving average filter

$$\begin{aligned} T_{FORW} &= {T_{BACKW}}^{-1} \\ T_{LPF} &= \mathrm{LPF}(T_{FORW}). \end{aligned} \tag{55}$$

Therefore, the resultant transformation for stabilize the current frame in respect to the previous one T_{RES} could be expressed as

$$T_{RES} = T_{BACKW} \cdot T_{REF} \cdot T_{LPF}. \tag{56}$$

3.6.4 Display styles

After an image transformation there were always some blank spaces. The image frame on the filter's output had been cropped to made a space for frame's transformation. The transformation coefficients were limited to avoid memory leaks. That approach had required the least computing effort which allowed to yield a real-time performance. There had been implemented three display styles:

full frame – despite of an original image, the stabilized one was displayed in the screen,

split – on the left half of the screen an original image was displayed and on the right half the stabilized one,

motion vectors – similar to the full frame display but with additional motion vectors plot.

The default one was the full frame stabilization. A different display style could be chosen by calling the filter with an appropriate parameter.

CHAPTER 4
EVALUATION OF STABILIZATION QUALITY

Results' analysis was the last step of video stabilization filter development. It had been hard to specify strict criteria, so the evaluation was based rather on subjective author's observations and feelings.

Main tests were run under MAC OS X 10.5.8 on the MacBook with Core 2 Duo 2.4 GHz CPU and 2 GB of RAM memory. All videos were played fluently with the CPU usage up to about 60%, so the real-time operation was fully achieved, even for HD ready videos.

Application was also run on the virtual machine with a single core CPU, RAM memory limited to 512 MB and Ubuntu 10.4 LTS operating system. In this case, the real-time performance was achieved only for VGA and SVGA videos.

4.1 Test video sequences stabilization

Filter's performance was evaluated in case of several different video sequences, all of them are included in the attached CD.

"earth" sequence:

- artificially rendered video,
- no shakes, but large moving objects in the frame,
- still camera,
- SVGA frame (800×600) pixels.

The original video was perfectly still. Filter's operation introduced small dithering (sub-pixel image shakes) which may have been annoying. The output video had had worse quality after than before filtering, but it was still acceptable.

"flight" sequence:

- artificially rendered video,
- large, short shakes but without any horizon tilt,
- complex camera move combined mostly from dolly and truck,
- SVGA frame (800×600) pixels.

In this case the improvement of video's quality was clear. There were still some tiny shakes, but overall result was very good. The shakes were significantly suppress and the camera move was preserved.

"sledge" sequence:

- real video,
- large but short shakes,
- advantage of dolly camera move,
- VGA frame (640×480) pixels.

That sequence was very hard to stabilize because of large area with low contrast, which decreased the number of usefull motion vectors. In this video the shakes were huge and some of them caused image strokes. The filter could not handle such large distortions. In general, the image quality was improved, but the result could have been better.

"funicular" sequence:

- real video,
- large but short shakes,
- still camera,
- HD ready frame (1280×720) pixels.

This was a good example of filters capability limitations. Shakes up to a certain amplitude were suppressed perfectly, but ones biggest caused image strokes.

"walk1" sequence:

- real video,
- large and slow shakes caused by walking,
- advantage of dolly camera move,
- HD ready frame (1280×720) pixels.

In this case the image stabilization worked well. Camera swings were removed with preservation of the directed camera dolly move. Some blur was present in the output video but it had not been introduced by filter's operation.

"walk2" sequence:

- real video,
- large and slow shakes caused by walking,
- advantage of truck camera move,
- HD ready frame (1280×720) pixels.

Stabilization results for this video were not perfect, however the output image had much better quality than the input. Camera pan move had been preserved and swings had been removed but there were still some dithers.

"walk3" sequence:

- real video,
- large and slow shakes caused by walking,
- advantage of truck camera move,
- HD ready frame (1280×720) pixels.

Developed filter barely handled this video. There were few motion vectors and the transformation estimation tended to be inaccurate. This resulted in many image strokes and swings. In some parts the image was steady, but the overall performance was rather poor.

4.2 Review of other existing solutions

When this work had been written, there were a few existing video stabilization applications or plugins. Some of them were free of charge like:

- deshaker plugin for VirtualDub,
- plugin for translate developed by Georg Martius,
- video stabilization in YouTube Video Editor,
- vReveal (with output limited to 480 pixel width),
- Video Stabilizer.

There were also some commercial ones:

- Digital Video Stabilizer plugin for Adobe PremiereThis ($49.99),
- piStabilize plugin for iMovie ($39),
- full version of vReveal ($39).

All of them were video converters without a real-time operation option, and some of them require even two-pass processing. The image stabilization provided by YouTube was tested with the "sledge" sequence. The output video (included in the attached CD) was almost perfectly steady, but the camera moves were suppose to be piecewise linear what looked slightly abnormal.

The only example of existing real-time video stabilization was `vf_deshake` filter for FFmpeg developed by Georg Martius and Daniel G. Taylor. Unfortunately, it had been developed within some elder version of FFmpeg library which was not compatible with the current one and the author was unable to compile and run that application.

CONCLUSIONS

The overall stabilization filter's performance was good. In most cases, shakes had been diminished what improved the image quality. An application ran in the real-time even for videos in HD ready resolution. Some improvement may have been implemented, but the goal was achieved.

Real-time operation requirement caused the computational complexity limitation. The most exhaustive step in shakes elimination was the motion estimation. There was a necessary to keep the balance between required accuracy and available execution time. The number of motion vectors, the search range and the compared macroblocks size was experimentally selected.

The largest impact on the stabilization performance had the motion estimation. It was worth to put more attention to accurate motion estimation that to dealing with outliers. In case of large displacements, fast motion estimation algorithms tended to converge to the local instead of the global cost function's minimum. a large number of outliers caused transformation estimation very inaccurate and time consuming and that abolished the profit from the fast motion vectors' search.

The two-step transformation estimation had seemed to work perfectly in a preliminary implementation, but due to inaccurate motion vectors' values and their small number in the final application, transformation estimation results was also imprecise. The developed filter handled better with the translation than with the rotation estimation. That was caused by much greater impact of rotation's inaccuracy on the output image. Due to this, it was impossible to stabilize the scaling factor and it had to been equalized to one. As far as the distortions made by imprecise rotation's compensation had been acceptable, the inaccuracy in the scaling factor made the output image unusable.

Stabilization filter's capability was limited. The output frame was cropped to keep transformed image inside it without any blank areas. That bounded the transformation range. In case of very strong shakes, the transformation had been to extensive and had to be clipped. That caused a mismatch between subsequent frames and visible image stroke.

Is some cases the output image quality was worst that the input one. Due to limited to half-pixel motion estimation accuracy, some dithering may have be introduced. When video shakes was small or none, this distortions could have override the stabilization profit.

When the shakes were slow, there distinction between directed and undirected camera moves was difficult. It may have been solved by implementation one more advanced motion filtering algorithm. Usage of motion's prediction may have allowed to use a two-side despite of the one-side filtering and decrease the introduced camera's move phase shift.

Stabilization result strongly depended on processed video sequence. Areas with low contrast were useless in motion estimation. Small number of motion vectors caused inaccurate transformation estimation and bad frames fitting as a result. In general, about 25% of motion vectors were discarded in the preliminary stage and similar part of them was classified as outliers and discounted. When the portion of lost because of too low contrast motion vectors had been more than 50%, the stabilization's result worsened.

For further work, one might attempt to use the higher class of estimated transformation. However, it can be impossible with current motion estimation accuracy. The motion vectors estimation accuracy might be increased up to quarter-pixel or even more, but it requires some additional computational effort. The other remedy is an increment of motion vectors' number, but it also will be very exhaustive. Another possible improvement field is the video display style. Image cropping is the simplest method, but it causes lost of field of view. There exist better solutions, but all of them are quite exhaustive, so they was not took into a consideration in this work. Development of GPU usage may solves the problem of complexity limitation and bring superior performance.

REFERENCES

[1] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision.* Cambridge University Press, 2003.

[2] M. I. Sezan and R. L. Lagendijk, editors. *Motion analysis and image sequence procesing.* Kluwert Academy Publishers, 1993.

[3] D. Salomon. *Computer graphics and geometry modeling.* Springer-Verlang New York Inc., 1999.

[4] J. Vineyard and J. Cruz. *Setting up your shots: great camera moves every filmmaker should know.* Michael Wiese Productions, 2008.

[5] http://www.indie-film-making.com/types-of-camera-movement, May 2011.

[6] G. Millerson. *Video camera techniques.* Focal Press, 1994.

[7] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with application to image analysis and automated cartography. *Communications of the ACM,* 24(6):381–395, 1981.

[8] V. Bhaskaran and K. Konstantinides. *Image and video compression standards: algorithms and architectures.* Kluwert Academy Publishers, 1997.

[9] T. Wiegand, G. J. Sullivan, G. Bjontegaard, A. Luthra. Overview of the H.264/AVC video coding standard. *IEEE transactions on circuits and systems for video technology,* 13(7):560–576, 2003.

[10] F. Dufaux, F. Moscheni. Motion estimation techniques for digital TV: a review and a new contribution. *Proceedings of the IEEE,* 83(6):858–876, 1995.

[11] B. Furht, J. Greenberg and R. Westwater. *Motion estimation algorithms for video compression.* Kluwert Academy Publishers, 1997.

[12] W. Hong, D. Wei, A. U. Batur. Video stabilization and rolling shutter distortion reduction. *17th IEEE International Conference on Image Processing,* pages 3501–3504, 2010.

[13] T. P. Zieliński. *Cyfrowe przetwarzanie sygnałów : od teorii do zastosowań.* Wydawnictwa Komunikacji Łączności, 2005.

[14] S. W. Smith *The scientist and engineer's guide to digital signal processing.* California Technical Publications, 1997.

[15] FFmpeg documentation, http://www.ffmpeg.org/documentation.html, May 2011.

Appendix A – `vf_stabilize.c` file reference

#include <math.h>
#include "avfilter.h"
#include "libavutil/lfg.h"
#include "libavutil/pixdesc.h"
#include "libavutil/timer.h"

Data Structures

struct	**MotVector** Motion vector parameters structure.
struct	**Point** Point coordinates structure.
struct	**StabContext** Filter structure.

Enumerations

enum	**InterpMethod** { NEAREST, BILINEAR_LUMA, BILINEAR } Interpolation method.
enum	**SearchMethod** { FAST, FULL } Search algorithm.
enum	**DisplayStyle** { FULL_FRAME, SPLIT, MOTION_VECTORS } Display style.

Functions

static int	**edges** (AVFilterBufferRef *img, int x, int y) Edges detection.
static int	**bprop** (AVFilterBufferRef *ref, AVFilterBufferRef *curr, MotVector mv, int dx, int dy, int m) BPROP cost function.
static int	**mad** (AVFilterBufferRef *ref, AVFilterBufferRef *curr, MotVector mv, int dx, int dy, int m) MAD cost function.
static void	**full_search** (AVFilterBufferRef *ref, AVFilterBufferRef *curr, StabContext *stab) Full-searching of motion vectors.
static void	**fast_search** (AVFilterBufferRef *ref, AVFilterBufferRef *curr, StabContext *stab) Fast-searching of motion vectors.
static void	**tranest** (MotVector mv1, MotVector mv2, StabContext *stab) Transformation estimation.

static int	**nsamp** (float eps) RANSAC number of tries calculation.
static void	**ransac** (StabContext *stab) RANSAC transformation estimation.
static void	**newton** (StabContext *stab) Newton's transformation re-estimation.
static void	**tranlpf** (StabContext *stab) Low-pass motion filtering.
static void	**tranmul** (StabContext *stab) Transformation multiplication.
static void	**traninv** (StabContext *stab) Transformation inversion.
static void	**transform** (AVFilterBufferRef *in, AVFilterBufferRef *out, StabContext *stab) Frame transformation.
static void	**split** (AVFilterBufferRef *in, AVFilterBufferRef *out, StabContext *stab) Split the screen between stabilized and original images.
static void	**putpixel** (AVFilterBufferRef *out, int x, int y, StabContext *stab) Draw a pixel on the screen.
static void	**drawmv** (AVFilterBufferRef *out, StabContext *stab) Draw motion vectors.
static av_cold int	**init** (AVFilterContext *ctx, const char *args, void *opaque) Initialization entry.
static av_cold void	**uninit** (AVFilterContext *ctx) Termination entry.
static int	**query_formats** (AVFilterContext *ctx) Set a list of supported formats.
static int	**config_props_input** (AVFilterLink *link) Configure filter parameters.
static int	**config_props_output** (AVFilterLink *link) Pass parameters to the next filter.
static void	**end_frame** (AVFilterLink *link) Process the frame - called when input data are sent.

Variables

AVFilter	**avfilter_vf_stabilize** Filter structure.

Appendix B – vf_stabilize filter users manual

I - DEPENDENCIES

FFmpeg requires some additional libraries to be compiled and to operate.
In Ubuntu Linux all of them can be installed with a command:

```
$ sudo apt-get install yasm libsdl1.2-dev libsdl1.2debian x264
libx264-dev
```

II - INSTALLATION

There are two ways to install FFmpeg with vf_stabilize filter.
The first option is:

1) Download the newest version of FFmpeg sources.

2) Add the file vf_stabilize.c into the libaafilter directory.

3) Add a line:
   `REGISTER_FILTER (STABILIZE, stabilize, vf);`
   to the file libavfilter/allfilters.c

4) Add a line:
   `OBJS-$(CONFIG_STABILIZE_FILTER) += vf_stabilize.o`
   to the file libavfilter/Makefile

5) Compile the application in a standard manner.

The second option is to use binaries from an attached CD:

1) Unpack an archive with:

   ```
   $ tar -pxzf ffmpeg.tar.gz
   ```

2) Enter the ffmpeg/build directory.

3) Run build.sh script which will configure, make and install
   the appliction.

III - USAGE

Vf_stabilize filter can be used with ffplay video player or with ffmpeg
video converter. The syntax is similar in both cases:

```
    $ ffplay <input_file> -vf <filter>
    $ ffmpeg -i <input_file> -vf <filter> -b <bitrate> <output_file>

The section <filter> can be just:

    stabilize

for a default settings. There are also some configuration possibilities:

    "stabilize=A:B:C:D:E"

where:

    A - motion vectors searching method (default 0):
        0 - fast search,
        1 - full search.
    B - interpolation method (default 0):
        0 - nearest (no interpolation),
        1 - bilinear in the luma plane, nearest in the chroma plane,
        2 - bilinear in both planes.
    C - display style (default 0):
        0 - full frame stabilization,
        1 - split the screen and stabilize only a half of it,
        2 - stabilize whole frame and show motion vectors.
    D - filter length in frames, limited in range 1-30 (default 12).
    E - crop margin in percent, limited in range 0-30  (default 5).

Only first n parameters must be passed (n varies from 0 to 5). For example:

    $ fflpay <input_file> -vf stabilize
    $ fflpay <input_file> -vf "stabilize=0"
    $ fflpay <input_file> -vf "stabilize=0:0"
    $ fflpay <input_file> -vf "stabilize=0:0:0"
    $ fflpay <input_file> -vf "stabilize=0:0:0:12"
    $ fflpay <input_file> -vf "stabilize=0:0:0:12:5"

Default usage with ffplay are equal to:

    $ ffplay <input_file> -vf stabilize

and:

    $ ffplay <input_file> -vf "stabilize=0:0:0:12:5"

For the conversion, it is recommended to use more accurate motion search,
an interpolation and bitrate equat to 1.5 Mbps or above. For example:

    $ ffmpeg -i <input_file> -vf "stabilize=1:2" -b 2M <output_file>
```

Appendix C – CD-ROM

CD-ROM content:

/**3d** –3D scenes rendered in Anim8tor,

/**doxygene** –filter's code documentation,

/**ffmpeg** – FFmpeg binaries with working stabilization filter included,

/**matlab** – m-files used in this work,

/**video** – test video sequences,

paper.pdf – electronic version of this paper.