

**POLITECNICO DI MILANO**  
Degree in Master of Science of Computer System Engineering  
Dipartimento di Elettronica e Informazione



## **Online Video Stabilization for UAV**

**Motion Estimation and Compensation for Unmanned Aerial  
Vehicles**

**AI & R Lab  
Laboratorio di Intelligenza Artificiale  
e Robotica del Politecnico di Milano**

**Relatore: Matteo Matteucci PhD**

**Master Degree Thesis of:  
Rodrigo José Ortiz Cayón  
Matricola 764561**

**Academic Year 2012-2013**



*All professors I have had, all my relatives and friends (who are my family too). They have been fundamental part of my life, however I rather prefer to mention one single person name who is not even here anymore. She left long time ago, I was still a child nevertheless she taught me sensitivity, humility and solidarity with the others.*

*Thanks for all your sacrifices and your constant affection given to me. I keep on my mind your bravery, your desire to live and your words: "I must take care of my little boy".*

*to my mom*

***Carmen María Cayón Sánchez***



# Abstract

Unnamed Aerial Vehicles (UAV) are known for their application in surveillance and tracking with on-board cameras. Videos from UAV usually suffer from jitter and high frequency unintended movements which makes necessary stabilize the footage. This is a Computer Vision problem, particularly difficult due to level of noise in acquisition and different types of scenarios which make hard to implement online stabilizer (on the fly).

The present thesis explains in depth algorithms, methods and implementations of the OpenCV module *videostab*<sup>1</sup>. We presented a version in C++ of *videostab* (*videostab\_polimi*<sup>2</sup>) in context of video from UAV.

Two main blocks constitute the current implementation: motion estimation and motion stabilization. In the former, the system robustly estimates global displacement in between every consecutive pair of frames. The latter can be used in two possible ways: Assuming that estimated motion contains intended UAV motion and high frequency vibrations or assuming that all estimated motion is undesirable. The first case is solved with a Gaussian filter to smooth motions and, in the second case, a filter called Zero Motion was implemented to give sensation of stillness.

Results show that the implementation is robust and can work in different scenarios of UAV. Moreover, it is possible to run it in a general purpose computer with high speed performance. Online performance can be achieved using some function with graphic processing unit (GPU).

---

<sup>1</sup><http://docs.opencv.org/trunk/modules/videostab/doc/videostab.html>

<sup>2</sup>[https://github.com/rodrygojose/opencv/tree/videostab\\_polimi/modules/videostab](https://github.com/rodrygojose/opencv/tree/videostab_polimi/modules/videostab)



# Acknowledgments

I am very grateful with Professor **Matteo Matteucci** for propose me the topic, make suggestions and reviewing the manuscript in spite of his very busy schedule.

Also, I would like to thanks **Ana Stella Martínez**, **Marta Korzec** and **Anne-Sophie Petit** for encourage me keep working on this written document.





# Contents

<b>Abstract</b>	<b>I</b>
<b>Acknowledgments</b>	<b>III</b>
<b>1 Context</b>	<b>7</b>
1.1 Overview . . . . .	7
1.2 Video Stabilization Methods . . . . .	7
1.3 Thesis of the Scope . . . . .	10
1.4 Structure of the Thesis . . . . .	10
1.5 Problem Description . . . . .	11
1.6 Objectives . . . . .	12
1.6.1 Main Objective . . . . .	12
1.6.2 Specific Objectives . . . . .	12
1.7 Challenges . . . . .	12
<b>2 State of Art</b>	<b>13</b>
2.1 Digital Video Stabilization . . . . .	13
2.2 Camera Model . . . . .	13
2.3 Related Works in Motion Estimation . . . . .	14
2.3.1 Motion Models . . . . .	14
2.3.2 Motion Estimation Methods . . . . .	15
2.3.3 Error Metrics and Outliers in Estimation . . . . .	17
2.4 OpenCV and GPU . . . . .	17

## CONTENTS

---

<b>3</b>	<b>Proposed Solution</b>	<b>19</b>
3.1	Implementation Overview . . . . .	19
3.2	Motion Estimation . . . . .	20
3.2.1	Feature Tracking . . . . .	22
3.2.1.1	Lucas-Kanade Optical Flow . . . . .	23
3.2.1.2	Inverse Compositional for Feature Tracking . . . . .	27
3.2.1.3	Pyramidal Lucas-Kanade Feature Tracker . . . . .	30
3.2.1.4	Sub-pixel Accuracy . . . . .	32
3.2.2	Feature Selection . . . . .	32
3.2.3	Global Motion Estimation with Affine Model . . . . .	35
3.2.4	Global Motion Estimation with Homography . . . . .	39
3.3	Motion Stabilizing . . . . .	42
3.3.1	Gaussian Motion Filter . . . . .	43
3.3.2	Zero Motion Filter . . . . .	46
3.3.3	Image Alignment . . . . .	48
3.4	Diagrams . . . . .	49
<b>4</b>	<b>Experiments and Discussion</b>	<b>51</b>
4.1	Performance Analysis and Discussion . . . . .	51
4.1.1	Feature Selection Quality: Texturedness . . . . .	51
4.1.2	Feature Tracking Accuracy: Dissimilarity . . . . .	53
4.1.3	Profiling . . . . .	56
4.1.4	Outlier Rejection in Global Motion Estimation . . . . .	58
4.1.5	Motion Filtering . . . . .	59
<b>5</b>	<b>Conclusion and Further Works</b>	<b>63</b>
<b>A</b>	<b>Logic Project Documentation</b>	<b>65</b>
A.1	Class Diagram . . . . .	65
A.2	Activity Diagrams . . . . .	65
<b>B</b>	<b>User Manual</b>	<b>75</b>

<b>C Result Examples</b>	<b>77</b>
C.1 Affine vs Homography . . . . .	77
C.2 Amount of Correction . . . . .	77
<b>Bibliography</b>	<b>79</b>

## CONTENTS

---

# List of Algorithms

3.1	Pyramidal LK Tracking Algorithm. . . . .	31
3.2	Shi-Tomasi Feature Selection. . . . .	35
3.3	Isotropic Point Normalization. . . . .	38
3.4	RANSAC . . . . .	39
3.5	LMedS . . . . .	42

## LIST OF ALGORITHMS

---

# List of Figures

1.1	Frequency and Amplitude range for different vibrations sources.[19]	8
1.2	Optical Image Stabilization.	8
1.3	Mechanical stabilizer.	9
1.4	Stabilization methods on Frequency-Amplitude coverage.	10
1.5	Video Surveillance Scenarios.	11
2.1	Projective Geometry.	14
3.1	General steps of video stabilization algorithm.	20
3.2	Global Motion Estimation Process.	20
3.3	Frames and Motions.	21
3.4	Pixel values around features in consecutive frames.	23
3.5	Optical Flow for images of Figure 3.4.	24
3.6	Schematic diagram for Inverse Compositional Optical Flow Tracker.	29
3.7	Pyramid Representation of the Image.	30
3.8	Computation of image levels.	31
3.9	Possible Problems in Feature Selection.[17]	32
3.10	Shi Tomasi regions for eigenvalues.	33
3.11	Impulse Response of Gaussian filter with $r = 20$ .	45
3.12	Magnitude Response of Gaussian filter.	45
3.13	Phase Response of filter.	46
3.14	Schema for apply the filter to the video stream.	47
3.15	Aliment of two consecutive images.	49

## LIST OF FIGURES

---

4.1	Test video frames. . . . .	51
4.2	Feature quality for frames in Figure 4.1. . . . .	52
4.3	Iteration Vs Error in tracking over four different combinations of windows and pyramid level. . . . .	53
4.4	Iteration Vs Error for 2000 consecutive frames. . . . .	55
4.5	Motion Vectors. . . . .	55
4.6	Profile Pie Chart. . . . .	56
4.7	$\epsilon$ Vs Number of iterations. . . . .	57
4.8	Order of computational complexity for motion estimation. . .	57
4.9	Outlier Rejection. . . . .	58
4.10	Motions before and after filtering. . . . .	60
4.11	Translation before and after filtering. . . . .	61
A.1	Class Diagram . . . . .	66
A.2	Video Stabilizing. . . . .	67
A.3	Prepare Motion Estimation Builders. . . . .	68
A.4	Building Stabilizers. . . . .	69
A.5	Build Motion Estimators. . . . .	70
A.6	Running the system in the highest level of abstraction . . . .	71
A.7	Stabilize Next Frame. . . . .	72
A.8	Motion Estimation. . . . .	73
A.9	Motion Stabilizing. . . . .	73



# List of Tables

2.1	Motion Models. . . . .	15
2.2	Parametric Motion Models. . . . .	16
4.1	Information about videos for testing. . . . .	52
4.2	Size of frames at maximum pyramid level. . . . .	54

## LIST OF TABLES

---

# Chapter 1

## Context

### 1.1 Overview

Unmanned Aerial Vehicles, or UAVs are flying machines usually provided with inexpensive on-board cameras used in situations where are required small air-crafts without an on board pilot. The range of applications comes from surveillance, sensing, search and rescue, exploration, among the others. Depending on environmental conditions and specific characteristics of the vehicle, different types of vibrations, at different levels, affect UAVs (see Figure 1.1). Being UAVs unsteady platforms, the video obtained by the on board camera is highly disturbed and should be stabilized to provide good quality video. Furthermore, in applications as tracking a stabilization block is required.

Video Stabilization is a time expensive process. The possibility of doing online stabilization for UAV's conditions is a interesting topic of research. Additionally, this opens the door for using the stabilization process in other applications such tuning the control parameters of a physical gimbal used for active motion stabilization on an aerial vehicle.

### 1.2 Video Stabilization Methods

Depending on the approach to stabilize images from video, three types of stabilizers are found: Optical Image Stabilization (OIS), Mechanical Image Stabilization (MIS) and Digital Image Stabilization (DIS). Next, a brief explanation of them.

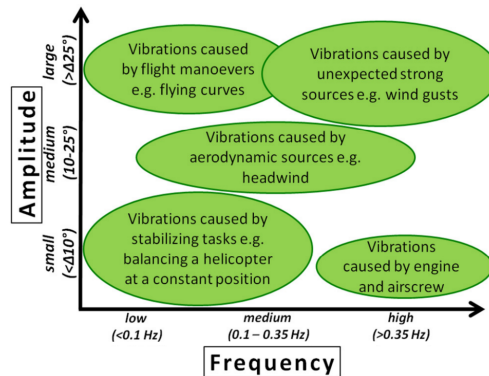


Figure 1.1: Frequency and Amplitude range for different vibrations sources.[19]

### Optical Image Stabilization

Optical Image Stabilization systems, manipulates the image before it gets to the camera sensor. When the lens moves, the light rays from the subject are bent relative to the optical axis, resulting in an unsteady image because the light rays are deflected. By shifting image stabilization lens group on a plane perpendicular to the optical axis to counter the degree of image vibration, the light rays reaching the image plane can be steadied. Two vibration-detecting sensors for yaw and pitch are used to detect the angle and speed of movement because vibrations might occur in both horizontal and vertical directions. An actuator moves the lens group horizontally and vertically thus counteracting the vibration and maintaining the stable picture (Figure 1.2<sup>1</sup>)[2].

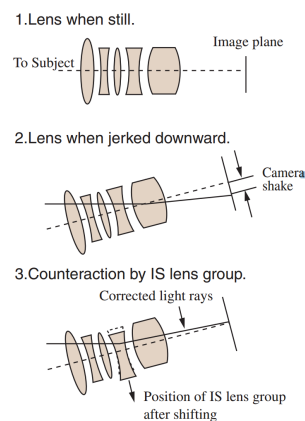


Figure 1.2: Optical Image Stabilization.

<sup>1</sup>Canon Digisuper 100xs, Product Manual: <http://www.canon.com/bctv/products/pdf/100xs.pdf>

### Mechanical Image Stabilization

Mechanical image stabilization involves stabilizing the entire camera. This type of stabilization can use a motion sensor as a gyroscope or mechanical devices such as shock absorbers for passively damp any kind of vibrations. Mechanical stabilizer has the advantage of instantly smooth high frequency vibrations. Commercial products of this type are available as Steadicam<sup>2</sup> (see Figure 1.3).

### Digital Image Stabilization

Digital Image Stabilization (DIS) systems, use electronic processing to control image stability. Unlike OIS, the image is manipulated after reaching the sensor. DIS systems detects what they think is camera vibration, it slightly moves the image so that it remains in the same place on the sensor. To correct the camera vibration, the camera electronics detect the direction of the movement and shifts the active field so that it meshes with the memorized field. A major risk of this system is the level of noise for some scenes; for instance, a large object moving in the frame may be interpreted as camera vibration and the camera will attempt to stabilize the subject causing a blurring of the image and reduction in picture resolution. An important issue becomes the robustness of DIS systems.

In some DIS, the camera can also use motion sensors to detect vibrations. Since this method senses movements in the camera and not the image, movement of a object in image cannot fool it. Unfortunately, not all cameras are equipped with motion sensors then it becomes necessary to use a Computer Vision approach.

---

<sup>2</sup><http://www.steadicam.com/>



Figure 1.3: Mechanical stabilizer.

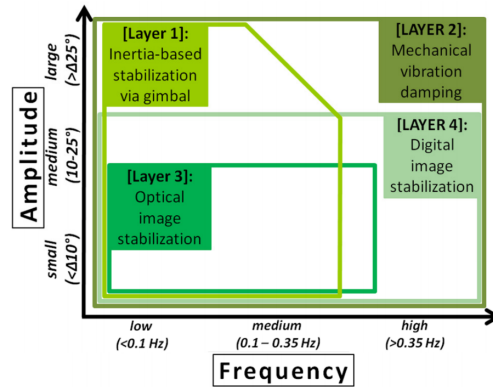


Figure 1.4: Stabilization methods on Frequency-Amplitude coverage.

In Figure 1.4 an image for different types of stabilization is reported with the range of Frequency-Amplitude they cover [19].

### 1.3 Thesis of the Scope

Motion in video images is caused by either object motion or camera movement. Digital image stabilization system endeavors to produce a compensated video sequence so that image motion due to camera's undesirable vibrations. The goal of this research is to explore and implement the state of the art to stabilize sequences in context of UAV. The developed algorithm should provide fast and robust stabilization system, and attempt on line performance.

### 1.4 Structure of the Thesis

This thesis is organized into six chapters and three appendices. The first chapter has presented the introduction, problem statement, mentions three types of image stabilization methods and the goal of the research. Chapter two describes in deep the research problem. Chapter three presents basic concepts in the field of image processing which are necessary to understand the method used to solve the problem being studied. Chapter four explains the methods and the techniques used to implement the various algorithms. Chapter five documents data resulted from the algorithms test. Chapter six summarizes the research, including limitations and areas of future work.

## 1.5 Problem Description

One of the main applications of Unmanned Air Vehicles is to provide video surveillance and target tracking. Video Surveillance systems use sequences of images from a camera to monitor scene activities. Regard to camera movement, situation of figure 1.5a or the one in figure 1.5b may raise.

The first situation shows a stationary camera, hence any differences between two consecutive frames is due movement of objects in scene, noise in acquisition or unexpected vibration in the camera. Examples of first situation are the surveillance in a train station or bridge of highways having the cameras on the bridge's structure.

On the other hand, in Figure 1.5b the scene involves an Unmanned Air Vehicle. This is a more complex situation because the recording device is over an unsteady platforms then we have all possible motions from first situation, but the intended motion of the camera, called *Ego-motion*, with more intense unexpected vibrations. Ego-motion is the major influence of wide disparities between consecutive frames and in tracking system with this kind of video, and sometimes stabilization procedure must be performed.

OIS and MIS systems are not suitable for UAV. With the former, it is necessary to use cameras with optical stabilizer. On the other hand, MIS has difficulties of size, weight and usually consume power, which makes mechanical video stabilization suitable for small UAVs at the current time.

Not generic online solutions exist for DIS. Some commercial ad hoc products as Acadia ILS-6000 [8] are expensive platform-based solutions for most applications, avoiding the widespread use in computer vision applications.

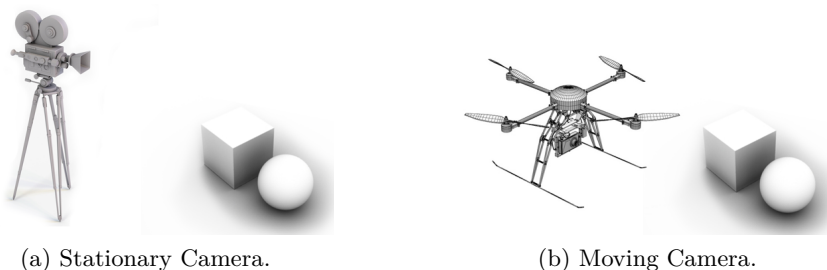


Figure 1.5: Video Surveillance Scenarios.

## 1.6 Objectives

### 1.6.1 Main Objective

- Video stabilization in context of tracking with UAV.

### 1.6.2 Specific Objectives

- Estimate camera's motions and measuring the amount of correction needed to stabilize frames.
- Implement a way to smooth vibrations.
- Implement a way to cancel movements as is UAV is still.
- Attempt to do all this online.

## 1.7 Challenges

UAV is always moving itself even during station-keeping it is never perfectly stable, making necessary to compensate images for applications as motion detection, tracking and others. The challenge is to determine the vehicle's undesirable motions from the image sequence to compensate the motion under UAV's conditions: unexpected and quick movement, high frequency vibration due the engine, with high noise rates.

Another challenge in to keep computation online to respect the constraint of number of frame per second and also to keep latency as minimum as possible for tracking applications.



## Chapter 2

# State of Art

### 2.1 Digital Video Stabilization

Video stabilization is the process of removing unwanted movement from a video stream. Mechanical stabilization physically dampens out vibration or unintended movement with gyroscopes or dampers. Optical stabilization is when sensors detect mechanical movement and shift the lens slightly. Both of these methods stabilize the image before digital conversion[5]. Digital video stabilization modifies each input frame to maintain a steady image after it is converted to digital. To do so, a transformation matrix is calculated via different motion estimation methods and is used to move the image. In this Chapter we mention common transformation matrix models and methods for motion estimation.

### 2.2 Camera Model

An image is a 2D pattern resulting from the projection of a 3D scene onto a 2D plane. In Figure 2.1 the basic pinhole camera model is used to conveniently simplify a linear mapping from  $\mathbb{P}^3$  to  $\mathbb{P}^2$ . Correspondences under perspective projection between 3D camera coordinate points  $\mathbf{X} = \begin{pmatrix} X & Y & Z \end{pmatrix}^T$  and points  $\mathbf{x} = \begin{pmatrix} x & y \end{pmatrix}^T$  are formed by intersection of the 2D image plane with a ray linking the camera lens and  $R$  [11].

If furthermore the scales of image and world coordinate systems match and the focal length is  $f = 1$ , the projection of a world point onto image plane is simplified in equation 2.1:

$$\tilde{\mathbf{x}} = \begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{Z} \begin{pmatrix} X \\ Y \end{pmatrix}. \quad (2.1)$$

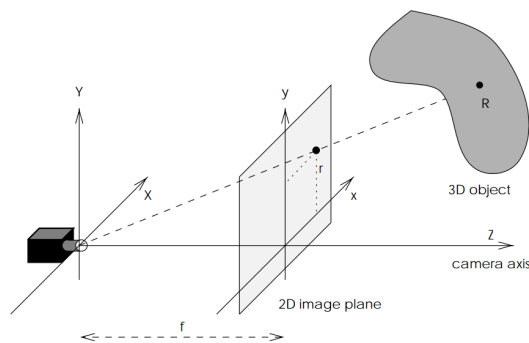


Figure 2.1: Projective Geometry.

The presence of motion manifests itself on the image plane by changes of pixel intensity values. These changes are used to recover motion of objects in scene. Thereafter, the 3D motion information of the objects is also projected onto the image plane. The fact that a 3D scene is projected onto a 2D image explains that a difference exists between the apparent and the real motion.

## 2.3 Related Works in Motion Estimation

In general, in motion estimation, firstly, we select a parametric model. This represents our a priori knowledge about the camera movement. Then we use data to fit the model. Data are usually the motion vectors produced by pixel or feature-based approaches. We also need a criteria, to measure how well the model represents the experimental data. The problem might be stated as a optimization problem where the best model is the one which fits better the data in the sense defined by a given criteria. An outlier method is also needed to filter noisy samples.

### 2.3.1 Motion Models

Motion models establish the mathematical relationships for mapping pixel coordinates from one image to another. A variety of such parametric motion models are possible that basically reflects camera motions such as: dolly (moving the camera forward or backward), track (moving the camera left or right), boom (moving the camera up or down), pan (rotating the camera around its  $Y$  axis), tilt (rotating the camera around its  $X$  axis), and roll (rotating the camera around the view axis).

In Table 2.1, Models are organized hierarchically by the number of degrees of freedom, showing transformation effects on a chessboard. Pixel loca-






Transform	D.O.F.	Preserves	Icon
Translation	2	Orientation	
Rigid	3	Lengths	
Similarity	4	Angles	
Affine	6	Parallelism	
Homography	8	Straight lines	

Table 2.1: Motion Models.

tion  $\mathbf{x} = (x, y)$  can be mapped to  $\mathbf{x}'$  by  $\mathbf{x}' = M\tilde{\mathbf{x}}$ , where  $\tilde{\mathbf{x}}$  is the homogeneous coordinates version of  $\mathbf{x}$  and  $M$  is the matrix transformation given in Table 2.2. Same table gives motion Jacobians for 2D planar transformations.

### 2.3.2 Motion Estimation Methods

#### Pixel-based Methods

Pixel-based Methods use all pixel-to-pixel matching images. An example is Fourier-based alignment, relying on the fact that Fourier transform of a shifted signal has the same magnitude as the original signal but linearly varying phase. In [15], and “Improved Fourier Mellin Invariant Method” was implemented, with the hypothesis information for alignment is necessary the whole image and not only sparse set of features. Fourier alignment works well in featureless scenarios. Another example of pixel based is the computational expensive Dense Lucas Kanade.

#### Feature-based Methods

The second class of algorithms works by extracting a sparse set of features and then matching or tracking them frame to frame. Examples of these algorithms are as Scale-invariant feature transform (SIFT), Speed up robust features (SURF), sparse Lucas Kanade and Geometric registration. Feature-based approaches are widely used for their advantages. Feature-based approaches have the advantage of being more robust against scene movement and are potentially faster.

Transform	Matrix	Parameters	Jacobian $J_x'$
Translation	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$	$(t_x, t_y)$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
Rigid	$\begin{bmatrix} c_\theta & -s_\theta & t_x \\ s_\theta & c_\theta & t_y \end{bmatrix}$	$(t_x, t_y, \theta)$	$\begin{bmatrix} 1 & 0 & -s_\theta x - c_\theta y \\ 0 & 1 & c_\theta x - s_\theta y \end{bmatrix}$
Similarity	$\begin{bmatrix} 1+a & -b & t_x \\ b & 1+a & t_y \end{bmatrix}$	$(t_x, t_y, a, b)$	$\begin{bmatrix} 1 & 0 & x & -y \\ 0 & 1 & y & x \end{bmatrix}$
Affine	$\begin{bmatrix} 1+a_{00} & a_{01} & t_x \\ a_{10} & 1+a_{11} & t_y \end{bmatrix}$	$(t_x, t_y, a_{00}, a_{01}, a_{10}, a_{11})$	$\begin{bmatrix} 1 & 0 & x & y & 0 & 0 \\ 0 & 1 & 0 & 0 & x & y \end{bmatrix}$
Homography	$\begin{bmatrix} 1+h_{00} & h_{01} & h_{02} \\ h_{10} & 1+h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix}$	$(h_{00}, h_{01}, h_{02}, h_{10}, h_{11}, h_{12}, h_{20}, h_{21})$	$\frac{1}{D} \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x'x & -x'y \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y \end{bmatrix}$ $D = h_{20}x + h_{21}y + 1$

Table 2.2: Parametric Motion Models.

### 2.3.3 Error Metrics and Outliers in Estimation

Once a suitable motion model is chosen to compare and describe the alignment between a pair of images, a method to estimate its parameters is needed. In Computer Vision, the mathematical norm is a total size or length of all vectors in a vector space or matrices is widely used. Error metrics for functions  $F$  and  $G$  (it may be images) respect to displacement  $\mathbf{h}$ .

- First order norm:  $L_1(\mathbf{h}) = \sum |F(\mathbf{x} + \mathbf{h}) - G(\mathbf{x})|$
- Second order norm:  $L_2(\mathbf{h}) = (\sum |F(\mathbf{x} + \mathbf{h}) - G(\mathbf{x})|^2)^{1/2}$

First order norm is called Sum of Absolute Difference (SAD) and second Sum of Squared Difference (SSD). We can make the above error metric more robust to outliers by using a function that grows less quickly than the quadratic penalty associated with least squares. SAD grows less quickly, however, since this function is not differentiable at the origin, it is not well suited to gradient-descent approaches.

Norms combined with outlier detection methods need to be implemented to give robustness. A datum is considered to be an outlier if it will not fit the true model instantiated by the true set of parameters within some error threshold that defines the maximum deviation attributable to effects of noise. In Computer Vision, RANSAC and LMedS methods are common implementations for outlier filter out.

## 2.4 OpenCV and GPU

OpenCV (Open Source Computer Vision Library: <http://opencv.org>) is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms. OpenCV 2.x API is essentially a C++ API, as opposite to the C-based OpenCV 1.x API. OpenCV has a modular structure, which means that the package includes several shared or static libraries [1]. Focus on real-time applications and with Intel Integrated Performing Primitives (IPP) install it makes programmed code running faster. The library comes with a GPU module (Graphic Processing Unit) implements the compute unified device architecture (CUDA), Developed by NVIDIA, to accelerate some time consuming functions. GPU module is specialized for data parallel computing and uses more transistors to data processing than flow control or data storage.



## Chapter 3

# Proposed Solution

In this chapter, we present a solution using the state of the art in video stabilization to implement an Online version with extra functionality suitable for UAV. During developing of this thesis, OpenCV began the releases of *videostab* module for C++ (see OpenCV's change log<sup>1</sup> at April 2012). The module has been “under active development”.

“Under active development” means that *videostab* module is being continuously improved. This brings us to use the “*trunk*” version of OpenCV which is the cutting-edge since repository is been updated frequently.

We follow next methodology in this chapter:

- Section Section 3.1 gives a global view of current solution.
- In Section 3.2, we talk about the specific motion estimation algorithms implemented in *videostab* module.
- In Section 3.3, we talk about stabilizing methods in *videostab* module.
- Then, in Section 3.4 we show the specific diagrams for the implementation of *videostab* to clarify how the Motion Estimation part and the Stabilizing part are linked together.

### 3.1 Implementation Overview

Generic global steps of a digital video stabilization system are shown in Figure 3.1. These include two blocks: *Motion Estimation* and *Motion Stabilizing*. As an input we have a *Video Stream* object to be stabilized.

---

<sup>1</sup><http://code.opencv.org/projects/opencv/wiki/ChangeLog>

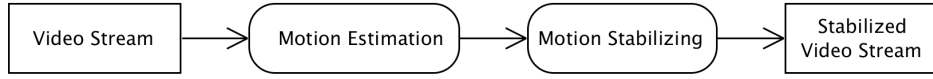


Figure 3.1: General steps of video stabilization algorithm.

Final quality of stabilization for video (*Stabilized Video Stream*) and the number of frames that system processes in time, depend mostly on *Motion Estimation* process performed on *Video Stream* (see Figure 3.1). *Motion Stabilizing* block smooths vibrations assuming that intended motions are in low frequencies or simply there was no intention of motion at all.

Apart from the quality, another important fact is the number of frames processed per minute or Throughput. Enforcing real time Throughput challenges such kind of systems. Throughput is mainly driven by the estimation each pair of frames because this activity is an iterative time consuming part. Exist a natural trade-off between computation speed and estimation accuracy, which limits stabilization system throughput .

Next, we describe in detail content of these blocks and how they are implemented in Section 3.2 and Section 3.3. We emphasize in detailed explanation of *Motion Estimation* since, as we mentioned, this part has a significant computational cost.

### 3.2 Motion Estimation

For Motion Estimation, we must find a mathematical relationships that map pixel coordinates from one image to the other. Inside the *Motion Estimation* block, we can differentiate between *Frame-to-frame Motion* and *Cumulative Motion* (an Activity diagram is shown in Figure 3.2). In the

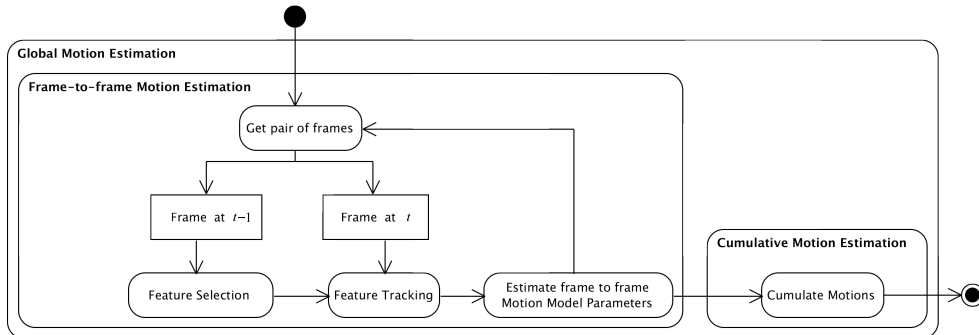


Figure 3.2: Global Motion Estimation Process.



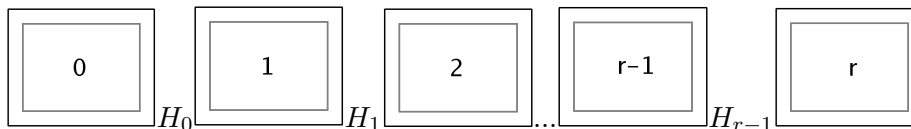


Figure 3.3: Frames and Motions.

former, mapping is done between consecutive frames and its relation with the latter is defined in equation 3.1.

Figure 3.3 shows a graphical representation of frames and motions  $H_i$  between every pair from frame 0 to frame  $r$ . Some commonly used parametric models were reviewed in Section 2.3.1 so  $H_i$  can take the form of any of them. To smooth motions in every frame, we need to consider estimation for a number of forward frames and the same number of backward frames. This number is the stabilization radius ( $r$ ) and waiting for those  $2r$  estimations ( $r$  forward and  $r$  backward frames) introduces latency. More details about this will be discussed later in Section 3.3.

Let  $H_i$  be the estimated motion model between frame  $i$  and  $i+1$  in equation 3.1. Estimates  $H_i$  is called Forward Motion Estimation (or Backward Motion Compensation).  $H_{C_{i,j}}$  represents *Cumulative Motion* from frame  $i$  to  $j$  and it is computed as composition of all frame to frame estimations between  $i$  and  $j$ .

$$H_{C_{i,j}} = H_i \circ H_{i+1} \circ \dots \circ H_{j-1}. \quad (3.1)$$

For image alignment, Szeliski[17] said that Feature-based methods offer faster computation over pixel-based methods. Feature-based methods select salient points in an image, then find the correspondent points in another image. With point to point correspondences between consecutive frames, we can compute feature motion vectors and use them as input data to feed an algorithm for computing their parameters of a motion model. Motion vectors represent displacements from each detected feature in the previous frame (at  $t-1$ ) to current frame (at  $t$ ). Section 3.2.2 and Section 3.2.1 explain how to select feature points and then how match them in consecutive frames.

Although the logical order says first to find features, afterward track them; we present the steps in invert order for a reason: feature selection process was adapted to a specific tracking method called Lucas-Kanade Optical Flow. Then we need to understand firstly Lucas-Kanade to understand the feature selection method. Lucas-Kanade Optical Flow in its original version was developed during 1981 in “*An Iterative Image Registration Technique*

*with an Application to Stereo Vision*”[10]. Later, a feature selection method was presented in 1994 by Shi and Tomasi in a paper called “*Good Features to Track*”[16]. Shi and Tomasi developed their ideas around Optical Flow framework and they used some mathematical procedure that will be explained in Section 3.2.1.

After computing accurate feature motion vectors, we use them to estimate the model parameters handling outliers. Under condition of UAV, some motion models have no sense. For instance, in [9] the author showed that their translational model can not accurately classify motion that occur in UAV and outlier rejection techniques become unreliable. This is because aerial vehicles are continuously subjected to scaling and rotation movements that Translational model can not deal with.

Affine Model offers better results; Section 3.2.3 explains in detail how the *videostab* module estimates the parameters for the Affine matrix. Additionally, same explanation is given for more complex motion model, Homography in Section 3.2.4.

In next sub-sections, nomenclature and definitions are a consensus of the most important papers of current *videostab* OpenCV implementations. To summarize, the order of ideas is:

- Feature Tracking (in current frame).
- Feature Selection (using previous frame).
- Motion Estimation with Affine (between consecutive frames).
- Motion Estimation with Homography.

### 3.2.1 Feature Tracking

Let’s assume that we have already properly selected key-points in a video frame. Now we need to retrieve a new frame and match on it the key-points from the previous frame. These matches represent movement vectors that are used in the upcoming step for frame to frame motion estimation. A sample situation is in Figure 3.4<sup>2</sup>. An example of selected feature in frame  $t - 1$  (upper part of Figure 3.4) was located at coordinates (121, 84) with pixel value 0.9. A tracker must find the feature in frame  $t$  at position (122, 102) taking advantage of small inter-frame displacements in video. The feature moved 19 pixels (1 in  $x$  and 18 in  $y$ ). Notice that pixel values around the feature are not equal in both images, however they keep similar values.

---

<sup>2</sup>Test car video is available in Matlab Computer Vision System Toolbox.



Figure 3.4: Pixel values around features in consecutive frames.

Tracking features we estimate motion that occur at pixel of interest (Local Motions). Current solution uses Optical Flow to track salient points. Method for calculate Optical Flow is Lucas-Kanade's method (LK). Following section gives a detailed explanation of operation principle .

### 3.2.1.1 Lucas-Kanade Optical Flow

In 1981 Lucas and Kanade presented an iterative algorithm[10] for image registration using spatial intensity gradient to align a pair of images. The algorithm took advantage of proximity between a pair of images for some applications as in case of consecutive frames from video record. Algorithm could be used to estimate Optical Flow. An example of LK Optical Flow result is presented in Figure 3.5 (yellow lines are motion vectors).

We represent input image  $I(\mathbf{x})$  (current frame at time  $t$ ) where  $\mathbf{x} = (x, y)^T$  are pixel coordinates. For some operation with matrices, pixel coordinate must be in homogenous form as  $\mathbf{x} = (x, y, 1)^T$ , even though we keep same notation  $\mathbf{x}$ . Template image  $T(\mathbf{x})$  represents a region of interest extracted

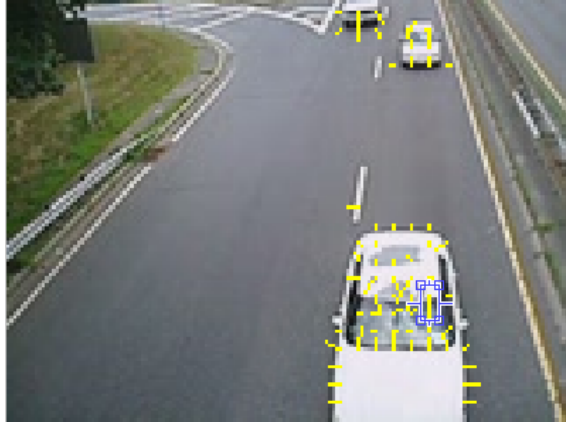


Figure 3.5: Optical Flow for images of Figure 3.4.

around a single key-point located at  $\mathbf{c} = (c_x, c_y)$  in the previous frame (at  $t - 1$ ). A region of interest is a square windows of size  $2w + 1$  centered at coordinates  $\mathbf{c}$  of a detected feature. In example of Figure 3.4  $w = 2$  so the overall window size is 5.

Transformations of pixel positions from one image to another are given by a parametrized function  $W$  with parameters  $\mathbf{p} = (p_1, p_2, \dots, p_n)^T$ .  $W(\mathbf{x}, \mathbf{p})$  represents the new (sub-pixel) location in the coordinates of  $I$  obtained by warping the position of a pixel at  $\mathbf{x}$  in coordinate frame of template  $T$ . An important issue is how to interpolate sub-pixel values of an image  $I$  at  $W(x; p)$  to compute  $I(W(x; p))$ . we will discuss about in Section 3.2.1.4, but for now assume that new coordinates  $W(\mathbf{x}, \mathbf{p})$  of  $\mathbf{x}$  are always integer values. An example of the form function  $W$  can take is the Affine warping  $W(\mathbf{x}, \mathbf{p}) = H_{affine}\mathbf{x}$ . The vector of parameters  $\mathbf{p} = (p_1, p_2, p_3, p_4, p_5, p_6)^T$  fills the Affine matrix as it is shown in equation 3.2.

$$H_{affine} = \begin{bmatrix} 1 + p_1 & p_3 & p_5 \\ p_2 & 1 + p_4 & p_6 \end{bmatrix}. \quad (3.2)$$

Sum of squared differences is the error measure used for tracking in  $I$  a features found at  $\mathbf{c} = (c_x, c_y)^T$  in a previous frame to  $I$ . The goal is to minimize the error L2 norm between template and warped image. For 3.3 and for subsequent equations, symbol  $\sum_{\mathbf{x}}$  is a simplified representation of:

$$\sum_{x=c_x-w}^{c_x+w} \sum_{y=c_y-w}^{c_y+w} \equiv \sum_{\mathbf{x}}.$$

Minimization of 3.3 with respect to  $\mathbf{p}$  is a non linear optimization problem. In fact, no relationship exists between warped pixel coordinate  $W(\mathbf{x}, \mathbf{p})$  and pixel value. Newton-Raphson method offers an iterative approach to compute displacement. Optimization process assumes to know a current estimation of  $\mathbf{p}$  (guess). Thus iteratively solves  $\Delta\mathbf{p}$  until  $\Delta\mathbf{p} \leq \epsilon$  or a maximum number of iterations. Parameter updating rule is given by 3.4.

$$\sum_{\mathbf{x}} [I(W(\mathbf{x}, \mathbf{p})) - T(\mathbf{x})]^2. \quad (3.3)$$

$$\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}. \quad (3.4)$$

Reasons to follow this scheme will be clarified as we move forward in explaining, but here are some valid assumptions:

1. LK makes assuming that  $\Delta\mathbf{p} \sim 0$ . It makes sense in consecutive sequences from video with small patch changes (*Temporal Persistence*).
2. For most of videos recorded by UAVs, valid assumptions are *Brightness Constancy* and *Spatial Coherence*. The first one assumes that a pixel keep approximately its appearance frame to frame and the second one says that points belonging to the same surface in the real world have similar projected motion in image plane.
3. guess or initial guess can be initialized in 0. This has sense in pyramidal implementation of LK (Section 3.2.1.3) and it is quiet useful for passing the results between levels as will be discussed in Section 3.2.1.3.

Equation 3.3 with guess presented to be minimized with respect to  $\Delta\mathbf{p}$  becomes equation 3.5.

$$\arg \min_{\Delta\mathbf{p}} \varepsilon(\Delta\mathbf{p}) \quad \varepsilon(\Delta\mathbf{p}) = \sum_{\mathbf{x}} [I(W(\mathbf{x}, \mathbf{p} + \Delta\mathbf{p})) - T(\mathbf{x})]^2. \quad (3.5)$$

Equation 3.5 can be linearized with first order of Taylor expansion like this.

$$I(W(\mathbf{x}; \mathbf{p} + \Delta\mathbf{p})) \approx I(W(\mathbf{x}; \mathbf{p})) + \mathbf{J}_{\mathbf{p}}(I(W(\mathbf{x}; \mathbf{p})))\Delta\mathbf{p}. \quad (3.6)$$

where  $\mathbf{J}_{\mathbf{p}}$  is the function that computes all first order partial derivatives with respect to  $\Delta\mathbf{p}$ . This is called Jacobian.

$$\mathbf{J}_{\mathbf{p}}(I(W(\mathbf{x}; \mathbf{p}))) = \frac{\partial[I(W(\mathbf{x}; \mathbf{p}))]}{\partial\mathbf{p}} = \nabla I \mathbf{J}_W, \quad (3.7)$$

$$\nabla I = \left[ \frac{\partial I(W(\mathbf{x}; \mathbf{p}))}{\partial x}, \frac{\partial I(W(\mathbf{x}; \mathbf{p}))}{\partial y} \right],$$

$$\mathbf{J}_W = \frac{\partial W(\mathbf{x}; \mathbf{p})}{\partial \mathbf{p}}.$$

Where gradient of image  $\nabla I$  is evaluated at warp  $W(\mathbf{x}; \mathbf{p})$  and  $\mathbf{J}_W$  is the Jacobian matrix of the transformation (Jacobians for 2D planar transformation were shown in Table 2.2).

Replacing 3.7 in 3.6 and 3.6 in 3.3 we obtain:

$$\varepsilon(\Delta \mathbf{p}) = \sum_{\mathbf{x}} [I(W(\mathbf{x}; \mathbf{p})) + \nabla I \mathbf{J}_W \Delta \mathbf{p} - T(\mathbf{x})]^2.$$

Applying partial derivation with respect to  $\Delta \mathbf{p}$  we have:

$$\frac{\partial \varepsilon(\Delta \mathbf{p})}{\partial \Delta \mathbf{p}} = 2 \sum_{\mathbf{x}} [I(W(\mathbf{x}; \mathbf{p})) + \nabla I \mathbf{J}_W \Delta \mathbf{p} - T(\mathbf{x})] [\nabla I \mathbf{J}_W]^T,$$

$$\frac{1}{2} \frac{\partial \varepsilon(\Delta \mathbf{p})}{\partial \Delta \mathbf{p}} = \sum_{\mathbf{x}} \nabla I \mathbf{J}_W \Delta \mathbf{p} [\nabla I \mathbf{J}_W]^T + \sum_{\mathbf{x}} [I(W(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})] [\nabla I \mathbf{J}_W]^T,$$

$$\frac{1}{2} \frac{\partial \varepsilon(\Delta \mathbf{p})}{\partial \Delta \mathbf{p}} = \sum_{\mathbf{x}} [\nabla I \mathbf{J}_W]^T \nabla I \mathbf{J}_W \Delta \mathbf{p} + \sum_{\mathbf{x}} [I(W(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})] [\nabla I \mathbf{J}_W]^T. \quad (3.8)$$

In 3.8 the term  $\nabla I \mathbf{J}_W$  is called steepest decent image.  $\Delta \mathbf{p}$  does not depend in summation with respect to  $\mathbf{x}$ , so we can represent the matrix of second order partial derivatives  $\sum_{\mathbf{x}} [\nabla I \mathbf{J}_W]^T \nabla I \mathbf{J}_W$  with  $H(W(\mathbf{x}; \mathbf{p}))$ . This is called Hessian matrix. Difference  $I(W(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})$  represents an error image  $\delta T(\mathbf{x})$ . Using new representations in 3.8 we have a shorter equation 3.9. Solution for  $\Delta \mathbf{p}$  is equation 3.10.

$$\frac{1}{2} \frac{\partial \varepsilon(\Delta \mathbf{p})}{\partial \Delta \mathbf{p}} = H(W(\mathbf{x}; \mathbf{p})) \Delta \mathbf{p} + \sum_{\mathbf{x}} \delta T(\mathbf{x}) [\nabla I \mathbf{J}_W]^T. \quad (3.9)$$

$$\Delta \mathbf{p} = -H(W(\mathbf{x}; \mathbf{p}))^{-1} \sum_{\mathbf{x}} \delta T(\mathbf{x}) [\nabla I \mathbf{J}_W]^T. \quad (3.10)$$

An inconvenient for finding final Optical Flow is the computational time. Remember that  $\mathbf{p}$  is computed iteratively and so for every new warp  $W(\mathbf{x}; \mathbf{p})$  we need to recompute  $H(W(\mathbf{x}; \mathbf{p}))$  and  $\delta T(\mathbf{x})$ . Particularly it is an expensive process to calculate the Hessian matrix evaluated in new warping and then its inverse. Fortunately, the Inverse Compositional Method for LK Optical Flow in Section 3.2.1.2 makes  $H$  independent of  $W(\mathbf{x}; \mathbf{p})$  therefore  $H^{-1}$  can be precomputed for being reused in each iteration.

In original Lucas-Kanade algorithm used for Image Registration [10] the authors used an Affine warp function to handle rotation, scaling and shearing for template alignment. In our case, we need it for tracking independently patches of key-points. It is enough so, to use Translational warps for local motions because in further steps we can use a higher level motion model to estimate global motion. Moreover, experiments in [16] show that the best choice for tracking is a pure translational model because of its higher reliability and accuracy over the small inter-frame motion of camera. Parameter vector has just two unknown  $\mathbf{p} = (p_1, p_2)^T$  and warping is  $W(\mathbf{x}, \mathbf{p}) = H_{trans}\mathbf{x}$ , where  $H_{trans}$  and Jacobian of the warp  $\mathbf{J}_W$  are significantly simpler:

$$H_{trans} = \begin{bmatrix} 1 & 0 & p_1 \\ 0 & 1 & p_2 \end{bmatrix} \quad \mathbf{J}_W = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

### 3.2.1.2 Inverse Compositional for Feature Tracking

Inverse Compositional Lucas-Kanade inverts the role of  $T$  and  $I$ : we want align a region of interest of an image  $I$  at time  $t$  to a template image from time  $t - 1$ .

$$\varepsilon(\Delta\mathbf{p}) = \sum_{\mathbf{x}} [T(W(\mathbf{x}; \Delta\mathbf{p})) - I(W(\mathbf{x}; \mathbf{p}))]^2. \quad (3.11)$$

Update rule becomes  $W(\mathbf{x}; \mathbf{p}) \leftarrow W(\mathbf{x}; \mathbf{p}) \circ W(\mathbf{x}; \Delta\mathbf{p})^{-1}$ . When  $W$  is the translation matrix the rule is simplified as the one used in equation 3.4 and additionally,  $I(W(\mathbf{x}; \mathbf{p}))$  is simplified to  $I(\mathbf{x} + \mathbf{p})$ .

Linearization of 3.11 with Taylor looks like this:

$$T(W(\mathbf{x}; \Delta\mathbf{p})) \approx T(W(\mathbf{x}; \mathbf{0})) + \mathbf{J}_P(T(W(\mathbf{x}; \mathbf{0})))\Delta\mathbf{p}. \quad (3.12)$$

For warping  $W(\mathbf{x}; \mathbf{0}) = \mathbf{x}$  because  $p_1$  and  $p_2$  are both zero. Rewriting 3.12:

$$T(W(\mathbf{x}; \Delta\mathbf{p})) \approx T(\mathbf{x}) + \mathbf{J}_P(T(\mathbf{x}))\Delta\mathbf{p}. \quad (3.13)$$

$$\mathbf{J}_{\mathbf{p}}(T(\mathbf{x})) = \nabla T \mathbf{J}_W = \nabla T. \quad (3.14)$$

Where  $\nabla T$  is the template image gradient evaluated  $\mathbf{x}$ . Pay attention that it does not depend on  $W$  so it does not change on every iteration.  $\nabla T$  is computed in OpenCV with the *Scharr* operator with next kernel for  $T_x$  and  $T_y$  respectively:

$$\begin{bmatrix} +3 & +10 & +3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix} \quad \begin{bmatrix} +3 & 0 & -3 \\ +10 & 0 & -10 \\ +3 & 0 & -3 \end{bmatrix}.$$

Replacing 3.14 in 3.13 and 3.13 in 3.11 we obtain:

$$\begin{aligned} \varepsilon(\Delta \mathbf{p}) &= \sum_{\mathbf{x}} [T(\mathbf{x}) + \nabla T \Delta \mathbf{p} - I(\mathbf{x} + \mathbf{p})]^2, \\ \frac{\partial \varepsilon(\Delta \mathbf{p})}{\partial \Delta \mathbf{p}} &= 2 \sum_{\mathbf{x}} [T(\mathbf{x}) + \nabla T \Delta \mathbf{p} - I(\mathbf{x} + \mathbf{p})] \nabla T^T, \\ \frac{1}{2} \frac{\partial \varepsilon(\Delta \mathbf{p})}{\partial \Delta \mathbf{p}} &= \sum_{\mathbf{x}} [\nabla T \Delta \mathbf{p} \nabla T^T] - \sum_{\mathbf{x}} [I(\mathbf{x} + \mathbf{p}) - T(\mathbf{x})] \nabla T^T, \\ \frac{1}{2} \frac{\partial \varepsilon(\Delta \mathbf{p})}{\partial \Delta \mathbf{p}} &= \sum_{\mathbf{x}} \nabla T^T \nabla T \Delta \mathbf{p} - \sum_{\mathbf{x}} [I(\mathbf{x} + \mathbf{p}) - T(\mathbf{x})] \nabla T^T, \\ \sum_{\mathbf{x}} \nabla T^T \nabla T &= H. \end{aligned} \quad (3.15)$$

$$I(\mathbf{x} + \mathbf{p}) - T(\mathbf{x}) = \delta T(\mathbf{x}). \quad (3.16)$$

Hessian becomes a constant spatial gradient matrix for every computation in a windows  $2w + 1$  centered at coordinates  $\mathbf{c}$ , also called co-variation matrix of derivatives.

$$\frac{1}{2} \frac{\partial \varepsilon(\Delta \mathbf{p})}{\partial \Delta \mathbf{p}} = H \Delta \mathbf{p} - \sum_{\mathbf{x}} \delta T(\mathbf{x}) \nabla T^T. \quad (3.17)$$

$$\sum_{\mathbf{x}} \delta T(\mathbf{x}) \nabla T^T = \mathbf{b}_k. \quad (3.18)$$

Where  $\mathbf{b}_k$  is a gradient-weighted residual vector usually called mismatch vector. Replacing 3.18 in 3.17 we finally have all those fancy equations



elegantly represented in system 3.19. Making  $\frac{\partial \varepsilon(\Delta \mathbf{p})}{\partial \Delta \mathbf{p}} = \mathbf{0}$ , and solving for  $\Delta \mathbf{p}$  we obtain equation 3.20.

$$H \Delta \mathbf{p} = \mathbf{b}_k, \quad (3.19)$$

$$\Delta \mathbf{p} = H^{-1} \mathbf{b}_k. \quad (3.20)$$

Where  $\mathbf{b}_k$  is called image mismatch vector. Sub-index  $k$  is placed because it changes on each iteration. Inspired by [3], we present a schematic diagram of Inverse Compositional Algorithm for Optical Flow Computation in Figure 3.6.

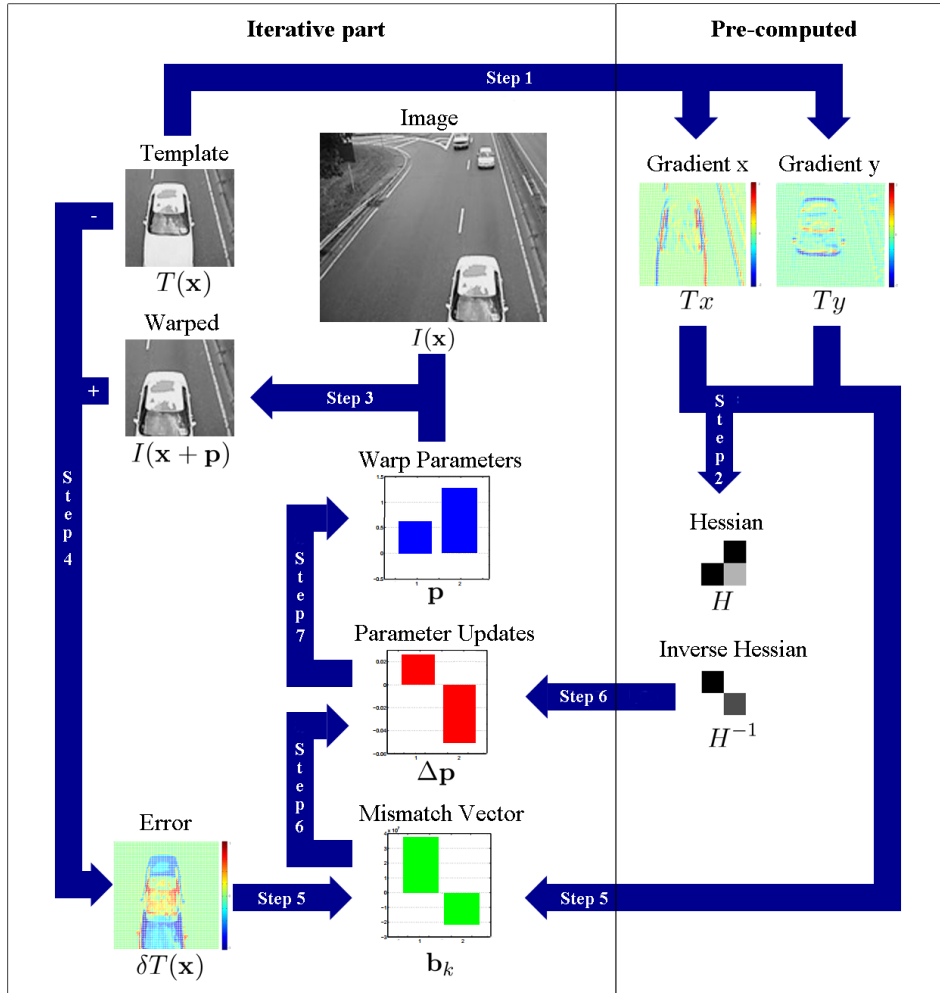


Figure 3.6: Schematic diagram for Inverse Compositional Optical Flow Tracker.

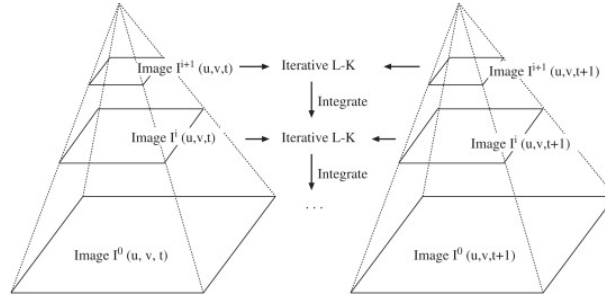


Figure 3.7: Pyramid Representation of the Image.

### 3.2.1.3 Pyramidal Lucas-Kanade Feature Tracker

An issue of using conventional Lucas-Kanade algorithm is to accomplish assumptions of *Brightness Constancy* and *Temporal Persistence* to track robustly. A way to do that is enforcing the use of high resolution cameras with high rates of frame-rate sequences but this kind of constrains the range of applications. Also we can accomplish large movements with a large integration window so more points fall within the local window. In fact, displacement in Figure 3.4 must be actually found with a minimum windows of size 37 ( $w = 18$ ) because  $y$  coordinate moved 18 pixels from one image to another (see 3.21). A bigger windows allows bigger frame to frame displacements.

$$\mathbf{p} = (p_1, p_2) \leq (w, w)^T. \quad (3.21)$$

Nevertheless, enlarging the window around a key-point makes computation more expensive and also affects the precision (for instance at occluding boundaries). Since videos recorded by UAV have large motions, *robustness* is a necessary conditions while keeping *accuracy*. To deal with this tradeoff between local *accuracy* and *robustness* OpenCV implemented a pyramidal version of LK. At first, the algorithm builds recursively a pyramidal representation with  $L_m + 1$  image levels, being  $L_m$  height of the pyramid (see Figure 3.7<sup>3</sup>).

The highest resolution image (initial image  $I^0$ ) is the base of pyramid and the top image is the lowest resolution built version ( $I^{L_m}$ ). Equation 3.22 describes the maximum feature displacement in for pyramidal version. For the example of Figure 3.4, if we want to keep  $w = 2$ , we need to build a pyramid with  $L_m = 4$  which would allow displacements of 32 pixels (max. displacement  $2^4 \times 2$ ):

$$\mathbf{p} = (p_1, p_2) \leq 2^{L_m} \times (w, w)^T. \quad (3.22)$$

<sup>3</sup>Image taken from <http://www.sciencedirect.com/science/article/pii/S0925231208000908>

$$\begin{aligned}
 I^L(x, y) &= \frac{1}{4}I^{L-1}(2x, 2y) + \\
 &\quad \frac{1}{8}(I^{L-1}(2x-1, 2y) + I^{L-1}(2x+1, 2y) + I^{L-1}(2x, 2y-1) + I^{L-1}(2x, 2y+1)) + \\
 &\quad \frac{1}{16}(I^{L-1}(2x-1, 2y-1) + I^{L-1}(2x+1, 2y+1) + I^{L-1}(2x-1, 2y+1) + I^{L-1}(2x+1, 2y-1))
 \end{aligned}$$

Figure 3.8: Computation of image levels.

---

**Algorithm 3.1** Pyramidal LK Tracking Algorithm.

---

1. Build pyramid representation of images  $T$  and  $I$ .
  2. Initialize pyramid guess:  $\mathbf{g}^{L_m} = (0, 0)^T$
  3. For  $L = L_m$  to 0 for each feature located at  $\mathbf{c}$ :
    - (a) Locate  $\mathbf{c}$  in  $T^L$ :  $\mathbf{c}^L = \mathbf{c}/2^L$
    - (b) Compute for current level in a window  $2w + 1$  around  $\mathbf{c}^L$ :  $\nabla T^L$  as in 3.14 and  $H^{-1}$  inverse of 3.15
    - (c) Initialize iterative IC feature tacking algorithm:  $\mathbf{p}_k = (0, 0)^T$
    - (d) Do from  $k = 1$  with step of 1 while  $k \leq 30$  and  $\|\Delta \mathbf{p}_k\| < \epsilon = 0.01$ 
      - i. Compute  $k$ -th image difference at level  $L$ :  $\delta T_k(\mathbf{x})$  as in 3.16
      - ii. Use  $\nabla T^L$  and  $\delta T_k(\mathbf{x})$  to compute image mismatch vector  $\mathbf{b}_k$  as in 3.18
      - iii. Compute  $\Delta \mathbf{p}_k$  with 3.20.
      - iv. Update parameters:  $\mathbf{p}_k \leftarrow \mathbf{p}_k + \Delta \mathbf{p}_k$
    - (e) Set guess for next level:  $\mathbf{g}^{L-1} = 2(\mathbf{g}^L + \mathbf{p}_k)$
  4. Final optical from is  $\mathbf{g}^0 + \mathbf{p}_k$  and new location of feature is  $\mathbf{c} + \mathbf{g}^0 + \mathbf{p}_k$
- 

In Figure 3.8 is displayed an image from the equation presented by Bouguet[4] for building each level  $L$  of the pyramid representation. OpenCV does this with the C++ function called **buildOpticalFlowPyramid**. In his paper, Bouguet has a detailed explanation about how to build pyramid in including what to do with boundaries and for cases whether image size is pair or uneven.

The tracking process starts from lowest detailed version of image (top of pyramid) applying iterative inverse compositional optical flow (Section 3.2.1.2) and then going down to lower levels with finer details passing as a guess the scaled estimation of found optical flow. Algorithm implemented in C++ by OpenCV Video Analysis module **calcOpticalFlowPyrLK** function is presented in Algorithm 3.1.

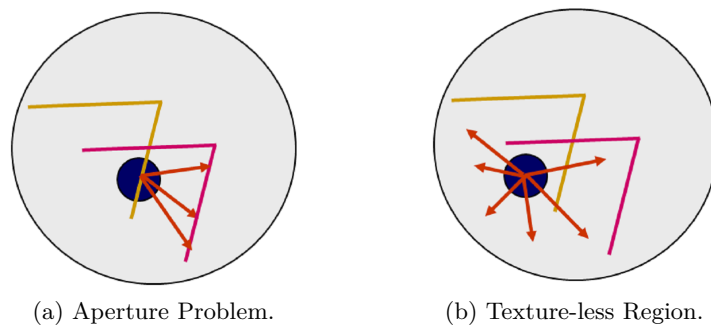


Figure 3.9: Possible Problems in Feature Selection.[17]

### 3.2.1.4 Sub-pixel Accuracy

In general, after transforming the coordinate  $\mathbf{x}$  to a new location  $W(\mathbf{x}; \mathbf{p})$  we usually obtain non integer values. So, which pixel value should be assign to  $I(W(\mathbf{x}; \mathbf{p}))$  This is a crucial issue to keep precision in tracking. Problem is addressed by pixel interpolation where the new image values are refined by bi-linear function helping to calculate intensity of a transformed pixel with better accuracy. Procedure well described in [4].

## 3.2.2 Feature Selection

Module *videostab* estimates frame motion by tracking features. Selecting key-points that can be accurately tracked frame to frame is essential for Optical Flow algorithm to work (described in Section 3.2.1.3). Figure 3.9 shows examples of problems that tracker can face if the selected points are not adequate.

Which features should be selected and how to measure Feature quality? The answer is given by [16]: “A good feature is one that can be tracked well, so that the selection criterion is optimal by construction”. It means if equation 3.19 can be solved reliably. The problem is addressed making features identifiable and unique (concept of **Texturedness** in Section 3.2.2). Another fact is to enforce that a key point exists from frame to frame (**Dissimilarity** Section 3.2.2). Besides those problems, it is important that each key-point provides new spatial information (Section 3.2.2).

### Texturedness

The idea of Textureness is to provide a rating of texture to make features within a windows identifiable and unique. For instance, lines are not good features since are not unique (see Figure 3.9a).

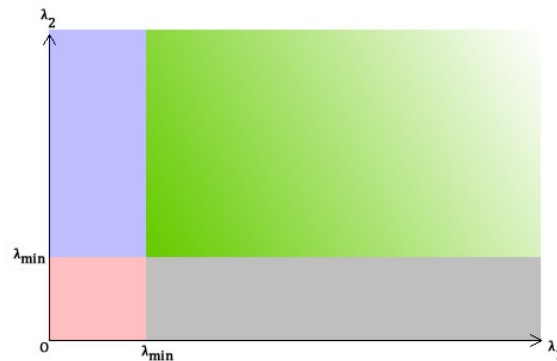


Figure 3.10: Shi Tomasi regions for eigenvalues.

To solve equation 3.19 it must be possible to invert the Hessian matrix. In practice next conditions must be satisfied:

1. Equation 3.19 must be well-conditioned: its eigenvalues cannot differ by several orders of magnitude.
2. Eigenvalues of Hessian overcome image noise levels  $\lambda_{noise}$ : implies that both eigenvalues of  $H$  must be large.

For the first condition we know that the greatest eigenvalue cannot be arbitrarily large because intensity variations in a window are bounded by the maximum allowable pixel value.

Regarding to second condition, being  $\lambda_1$  and  $\lambda_2$  two eigenvalues of  $H$ , following situations may rise (See Figure 3.10<sup>4</sup>):

- Two small eigenvalues  $\lambda_1$  and  $\lambda_2$ : means a roughly constant intensity profile within a window (Pink region).
- A large and a small eigenvalue: means unidirectional texture pattern (Violet or gray region).
- $\lambda_1$  and  $\lambda_2$  are both large: can represent a corner, salt and pepper textures or any other pattern that can be tracked reliably (Green region).

In practice, when the smaller eigenvalue of  $H$  is sufficiently large to meet a threshold (noise criterion), equation 3.19 is usually also well conditioned. OpenCV offers C++ function `cornerMinEigenVal` to calculate the minimal eigenvalue of  $H$ .

<sup>4</sup>Image taken from <http://www.aishack.in/>.

In conclusion, quality  $\Lambda$  of a feature is defined as:

$$\Lambda = \min(\lambda_1, \lambda_2). \quad (3.23)$$

Thus, we accept all windows satisfying:

$$\Lambda > \lambda_{noise}. \quad (3.24)$$

In practice, to make fast computation of this process, the Hessian used is a simplified version (see equation 3.25<sup>5</sup>) of the actual Hessian in equation 3.19. “Simplified” means to use a smaller windows to speed up computation at selection time. The windows usually has  $w = 1$  being a square block neighborhood of  $3 \times 3$  represented by  $S(p)$ .

Afterward, during tracking, the windows might be increased to enable capturing larger displacements.

$$H_S = \begin{bmatrix} \sum_{S(p)} \left(\frac{dI}{dx}\right)^2 & \sum_{S(p)} \left(\frac{dI}{dx} \frac{dI}{dy}\right)^2 \\ \sum_{S(p)} \left(\frac{dI}{dx} \frac{dI}{dy}\right)^2 & \sum_{S(p)} \left(\frac{dI}{dy}\right)^2 \end{bmatrix}. \quad (3.25)$$

### Dissimilarity

Texturedness does not guarantee that high rated features exist in consecutive frames. For instance, a feature may be the result of an intersection that does not exist in the real world as in cases of two edges at different depths. Occlusion is another issue to deal with. Also, temporary noise (say salt and pepper) may be consider a “good” feature to track. Mentioned cases are quiet common in UAV context.

Those situations are not easy to identify during the selection process, but They are possibly identifiable during tracking. Dissimilarity detects when those situations appear with the error measure defined by equation 3.11. Case when a feature is declared lost is explained by [4].

### Non-Maximum Suppressor

Provide new information is another desirable characteristic for features; for instance, tracking adjacent features does not give useful information to estimate displacement. Distribute even features over an image is not a good

---

<sup>5</sup>Derivatives are computed using *Sobel()* operator

**Algorithm 3.2** Shi-Tomasi Feature Selection.

---

1. Function calculates corner quality measure at every pixel  $p$  in source image:
    - (a) Calculates co-variation matrix of derivatives (Hessian) as in 3.25.
    - (b) Compute eigenvalues  $\lambda_1, \lambda_2$  of  $H_S$ .
    - (c) Calculate  $\Lambda$  as in 3.23.
  2. Apply non maximal suppression in  $S(p)$  neighborhood.
  3. Set threshold  $\lambda_{noise}$  in 3.24 as the 10% of the maximum quality measured in image.
  4. Points with the minimal eigenvalue less than  $\lambda_{noise}$  are rejected (application of equation 3.24).
  5. Sort remaining features by quality measure in descending order.
  6. Non maximal suppression is applied in feature qualities with Euclidean distance less than  $minDistance$ .
  7. Return the maximum number of desired features.
- 

idea because we do not know where movement will appear. A solution is to select points that are within a specified minimum separation distance applying non-maximum suppression over feature qualities. It means, to select feature with local maximum rating within a minimum distance.

In summary, in Feature Selection the most prominent points in the image are found as described in Algorithm 3.2 which is the current implementation of function `goodFeaturesToTrack` in OpenCV.

### 3.2.3 Global Motion Estimation with Affine Model

The first model used to estimate motion between frames is the Affine Model. This model was review in section Section 2.3.1. Because the model is linear and two dimensional, it can not handle depth, nevertheless estimation is computationally faster. Johansen demonstrated in [9] that motion of an UAV can be approximated by the Affine model when objects are far enough from the camera.

As stated before, to estimate motion, we need a set of correspondent points in both frames, a model (Affine is this case) and an algorithm to compute the parameters with error measure. In captured video by an UAV a large quan-

tivity of points that do not fit Affine model are expected. Due to those noisy points, a robust method should be used. Used method is Random Sample Consensus (RANSAC [7]) and it will be discussed in detail in Section 3.2.3. The used error measure is Least Squares with L2-norm as error metric.

### Least Squares to Calculate Affine Parameters

Estimate parameters of Affine model with a set of  $n$  features motions (at least three motions) can be done in Least Square sense. We have for each motion vector, represented by a pair of points  $\mathbf{x}_i$  (in current frame) and correspondent  $\mathbf{x}'_i$  (in next frame). Homogeneous coordinate expresses these points as  $\mathbf{x}_i = (x_i, y_i, 1)^T$  and  $\mathbf{x}'_i = (x'_i, y'_i, 1)^T$  respectively.

For an Affine form matrix  $H_{Affine}$ , the relationship between a pair of points is given by equation 3.26

$$\begin{bmatrix} x'_i \\ y'_i \end{bmatrix} = H_{Affine} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}, \quad (3.26)$$

where  $H_{Affine}$  is presented in 3.27 with parameters we want to find

$$H_{Affine} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}. \quad (3.27)$$

Thus, computing the right side of 3.26 using 3.27 we have 3.28

$$\begin{bmatrix} x'_i \\ y'_i \end{bmatrix} = \begin{bmatrix} ax_i + by_i + c \\ dx_i + ey_i + f \end{bmatrix}. \quad (3.28)$$

For  $n$  motion vectors we can represent 3.28 as a least square problem in 3.29

$$b = Ah. \quad (3.29)$$

Where

$$b = \begin{bmatrix} x'_1 & y'_1 & x'_2 & y'_2 & \dots & x'_n & y'_n \end{bmatrix}^T,$$

$$A = \begin{bmatrix} a0_1 & a1_1 & a0_2 & a1_2 & \dots & a0_n & a1_n \end{bmatrix}^T,$$



$$a0_i = \begin{pmatrix} x_i & y_i & 1 & 0 & 0 & 0 \end{pmatrix},$$

$$a1_i = \begin{pmatrix} 0 & 0 & 0 & x_i & y_i & 1 \end{pmatrix},$$

$$h = \begin{bmatrix} a & b & c & d & e & f \end{bmatrix}^T,$$

Affine matrix is found solving for  $h$  which gives the values in the right side of equation 3.27.

$$\arg \min_{h_{sol}}(RMSE),$$

$$RMSE = \frac{1}{n} \|b - Ah_{sol}\| = \sqrt{\frac{\sum (b - Ah_{sol})^2}{n}}. \quad (3.30)$$

Solution  $h_{sol}$  for 3.29 is found with Gaussian elimination to minimize the Root Mean Squared Error (see equation 3.30). In *videostab*, this was implemented with the function **estimateGlobalMotionLeastSquares**.

### Isotropic Point Normalization

Solution for equation 3.29 depends on the origin and the scale of the coordinate system in the image. Thus, it is convenient to normalize input points the coordinates in order to achieve better numerical stability.

Normalization consists in centering the coordinate and scale points to have average distance from their origin equal to  $\sqrt{2}$ . Steps to compute the Affine matrix with the isotropic point normalization are shown in Algorithm 3.3.

Implementation is found in *videostab* in **global\_motion.cpp** with the name **normalizePoints**.

### Robust Estimation with RANSAC

The Affine model can adequately estimate motion between frames under certain conditions in the scene. One condition is so the objects in scene are far enough from camera that scene can be considered as a flat scene (without depth). This is because Affine transformation lacks the ability to represent a true three dimensional motion which occurs in a video captured from an UAV. The approximation brings error in the estimation and it produces outliers in the data set. Another outlier sources are objects moving in scene with significant velocity with respect to the video frame rate. Other errors

**Algorithm 3.3** Isotropic Point Normalization.

---

1. Compute similarity transformation  $T_0$  that takes points  $\mathbf{x}_i$  to a new set of points  $\tilde{\mathbf{x}}_i$  and transform  $T_1$  that takes  $\mathbf{x}'_i$  to  $\tilde{\mathbf{x}}'_i$ , such that new centroid is the coordinate origin and their average distance from origin is  $\sqrt{2}$ .
2. Solve 3.29 with  $\tilde{\mathbf{x}}_i$  and  $\tilde{\mathbf{x}}'_i$  instead of  $\mathbf{x}_i$  and  $\mathbf{x}'_i$  so that instead of obtaining  $H_{affine}$  in 3.27 we will obtain  $\tilde{H}_{affine}$ .
3. Actual Affine matrix is given by equation 3.31. Notice that to compute the product in 3.31 matrices must be presented in homogeneous coordinates adding at the bottom of each matrix the row  $\begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$ .

$$H_{affine} = T_1^{-1} \tilde{H}_{affine} T_0. \quad (3.31)$$

---

occurs because condition of the scene, like in case of occlusion. All error sources in UAV video oblige to implement a robust method for estimation.

Random Sample Consensus (RANSAC) is an iterative and re-sampling technique which generates candidate solutions by using the minimum number of observations required to estimate the parameters of a predefined model. It was proposed by Fischler [7] in 1981 and since then it has been widely adopted by Computer Vision researchers to filter out outliers during estimation.

RANSAC has the ability to deal with non zero mean noise; it assumes that noise is uncorrelated and a minimum subset of data is correlated with respect to the model. In our case all movements not described within a threshold by the Affine model, will be interpreted as noise.

The RANSAC algorithm is presented in Algorithm 3.4. We start by computing the number of iterations  $N$  with equation 3.32. In that equation,  $m$  is the size of the minimum set of correspondences required to estimate model parameters,  $\epsilon$  is the maximum outliers ratio presented in the initial data set and  $p$  is the probability of success (usually set in 99%)

$$N = \frac{\log(1 - p)}{\log(1 - (1 - \epsilon)^m)}. \quad (3.32)$$

The RANSAC algorithm attempts to find the best hypothesis: minimum subset of motions that contains a maximum number of inliers within a threshold  $\tau$ . In our case this threshold is measured in pixels. Therefore, the projected points that fit the estimated Affine transformation in every

---

**Algorithm 3.4** RANSAC

---

1. Calculate the number  $N$  of iterations to perform.
  2. Randomly select three feature points of current frame and their correspondences in the next frame (current hypothesis).
  3. Estimate  $H_{affine-iter_k}$  by means of Least Squares.
  4. Determine number of inliers with respect to obtained model and a predefined threshold  $\tau$ .
  5. If the current inliers set contains the maximum number of found inliers so far, save model as the best model (best hypothesis).
  6. Reaped  $N$  times from step 2 to step 5.
  7. Return the best found model.
- 

iteration ( $H_{affine-iter_k}$ ) with an error less than  $\tau$  are considered as inliers (see equation 3.33, assuming homogeneous coordinates)

$$\|\mathbf{x}'_i - H_{affine-iter_k} \mathbf{x}_i\| < \tau. \quad (3.33)$$

Each motion vector offers two equations ( $x$  and  $y$  for equation 3.28), then we need at least three pairs of motions to estimate the six variables of equation 3.27 in an Affine model ( $m = 3$ ). It means three non collinear points in current frame and corresponding points in the next frame to obtain a  $3 \times 3$  linearly independent system. Therefore, in each iteration we randomly select three motion vectors and we find the model parameters as described in Section 3.2.3. Next, determine how many points out of the whole set are classified as inliers with equation 3.33. This process is repeated  $N$  iterations and, at the end, the set with the major number of inliers is used to calculate the final model parameters and the final root mean squared error. Implementation in OpenCV has the name `estimateGlobalMotionRansac` in `videostab` module.

### 3.2.4 Global Motion Estimation with Homography

Homography model (also known as Projective or Perspective transformation) is the second model for motion estimation. It operates on homogeneous coordinates. Direct Linear Transformation is introduced in Section 3.2.4 as procedure to calculate its parameters. To deal with outliers, a robust method

for Homography estimation is presented in Section 3.2.4 (Least Median of Square Regression).

### Direct Linear Transformation

Direct Linear Transform (DLT) is an algorithm that uses point correspondences between source and destination planes to compute the homography matrix  $H$ . In homogeneous coordinates, relationship between two correspondent points  $\mathbf{x}_i$  and  $\mathbf{x}'_i$  is given by equation 3.34 where  $s_i$  is a non-zero constant,  $\mathbf{x}_i = \begin{pmatrix} x_i & y_i & 1 \end{pmatrix}^T$ ,  $\mathbf{x}'_i = \begin{pmatrix} x'_i & y'_i & 1 \end{pmatrix}^T$  and form of  $H$  is in equation 3.35

$$s_i \begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} \sim H \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}, \quad (3.34)$$

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}. \quad (3.35)$$

From 3.35 and 3.34 we obtain 3.36, and then dividing first row by the third row and the second row by the third row, we obtain equations 3.37 and 3.38

$$s_i \begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11}x_i & h_{12}y_i & h_{13} \\ h_{21}x_i & h_{22}y_i & h_{23} \\ h_{31}x_i & h_{32}y_i & h_{33} \end{bmatrix}, \quad (3.36)$$

$$-h_{11}x_i - h_{12}y_i - h_{13} + (h_{31}x_i + h_{32}y_i + h_{33})x'_i = 0, \quad (3.37)$$

$$-h_{21}x_i - h_{22}y_i - h_{23} + (h_{31}x_i + h_{32}y_i + h_{33})y'_i = 0. \quad (3.38)$$

Equation 3.39 is the matrix representation of 3.37 and 3.38 for a number  $n$  of point correspondences

$$0 = Ah. \quad (3.39)$$

Where

$$A = \begin{bmatrix} a_{01} & a_{11} & a_{02} & a_{12} & \dots & a_{0n} & a_{1n} \end{bmatrix}^T,$$

$$a0_i = \begin{pmatrix} -x_i & -y_i & -1 & 0 & 0 & 0 & x_i x'_i & y_i x'_i & x'_i \end{pmatrix},$$

$$a1_i = \begin{pmatrix} 0 & 0 & 0 & -x_i & -y_i & -1 & x_i y'_i & y_i y'_i & y'_i \end{pmatrix}.$$

Each point correspondence provides two equations, so four motion vectors are sufficient to solve  $h$  in 3.39 with restriction of no three collinear points. To find the Homography we must find  $h_{sol}$  that gives the values equation 3.35 and minimizes the Sum of Squared Difference (SSD) in 3.40.

$$\arg \min_{h_{sol}}(SSD),$$

$$SSD = \|Ah_{sol}\|^2 = \sum (Ah_{sol})^2. \quad (3.40)$$

As stated for Affine, also for Homography estimation, to obtain a solution independent from the coordinate system in the image, Isotropic Point Normalization (as it was described in Algorithm 3.3) must be done in data input.

### Robust Estimation with Least Median of Squares

To achieve robustness in homography estimation with respect to outliers, method Least Median of Squares (LMedS[14]) is used. LMedS tries to find the parameters that produce the smallest value of median of squared residuals in data set. This is done by replacing the Sum of Squared Difference of equation 3.40 with Median of Squared Residuals (equation 3.41) and solving the nonlinear minimization problem

$$\arg \min_{h_{sol}}(\text{med}_i\{r_i^2(h_{sol})\}),$$

$$r^2(h_{sol}) = (Ah_{sol})^2. \quad (3.41)$$

To solved equation 3.41, a search must be performed in the space of possible estimated models. To avoid a full space search, a randomly subset of data is chosen to be analyzed. Similarly as described for Ransac method, for LMedS the size of subset is the minimum number of parameter needed to estimate the model, which in case of Homography, four motions (with no three collinear points). Also equation 3.32 is used to determine the number of iterations. Algorithm for LMedS is outlined in Algorithm 3.5[6]. LMedS

has the advantage over RANSAC that it does not require a prior knowledge: no need to set a threshold ( $\tau$ ) neither how much error to expect ( $\epsilon$ ). However, disadvantage is that LMedS does not work properly when more than the half of data are outliers.

---

**Algorithm 3.5** LMedS

---

1. Calculate the number  $N$  of iterations to perform.
  2. A Monte Carlo type technique is used to randomly select 4 motions.
  3. With four selected motions, estimate  $h_{sol-iter_k}$  with Direct Linear Transformation.
  4. Use  $h_{sol-iter_k}$  to determine median of the squared residuals, denoted in equation 3.41, with respect to the whole set of motion.
  5. Retain  $h_{sol-iter_k}$  if median of the squared residuals is minimal.
  6. Repeated  $N$  times from step 2 to step 5.
  7. Return found  $h_{sol-iter_k}$  in the form of 3.35 for which the median of the squared residuals is minimal.
- 

### 3.3 Motion Stabilizing

The problem statement in Section 1.5 gave an overview about motion sources captured in video. Those sources are: intended motion, moving objects in scene and unwanted jitter. A question is how to separate those motion sources.

We consider motion of small objects in foreground as outlier in Global Motion Estimation process so we assume that the estimated transformation matrix does not include motion of minor parts in first plane (for instance, cars moving). Reason to say that is because those motions are small moving areas which do not fit all together parametrized motion model (there is no correlation between ego-motion and moving objects in scene). RANSAC and LMedS method were in charge of filter out motion outliers. This is desirable because those small moving areas might be useful information for tracking applications.

The problem is now simplified to separate two motion sources: intended motion and unwanted vibrations. In UAV, intended motions may be seen as a low frequency component of the movement. Vibrations may be seen as high frequency random deviation from intended motion, so we can use a filter approach.

In [9], Johansen proposed a method named Parabolic Fit Camera to filter out online vibrations. Author assumes that intended motion components follow a parabolic behavior. Thus, adding a delay in a number of frames, Johansen adjusts to a parabola a set of a motion components (treating independently each motion component of an Affine model as scale, rotation and translations). The result of adjustment is the estimated intended motion, and removing from Affine transformation at each frame to frame to obtain a transformation matrix corresponding only to unwanted jitters. This method limits the usage of more complex motion models as Homography and assumption of independence among motion components may be not valid for aerial vehicles.

OpenCV *videostab* module implements a Gaussian filter approach to smooth the whole motion model matrix. This enables to filter out vibrations also for Homography transformations. Explication of how it works is given in Section 3.3.1. Additionally, to simulate the situation of still camera, in Section 3.3.2 we explained an implementation to filter all motion (including ego-motion).

### 3.3.1 Gaussian Motion Filter

Assuming that intended motion is correlated within  $n$  consecutive frames and unwanted displacements may be correlated within up to  $m$  consecutive frames where  $n \gg m$ . We can smooth motions by weighting with coefficients of a Normal distribution the cumulative movement of  $2r$  frames around the target frame (with  $m < 2r + 1 < n$ ). Radius  $r$  represents the number of forward frames (and the same number of backward frames) that Motion Stabilizing process uses to filter out vibrations hence we need a buffer of size  $2r + 1$ .

Retake idea of Figure 3.3,  $H_i$  is the transformation matrix from frame  $i$  to  $i + 1$ . Redefining equation 3.1 to include cumulative motion computations of backward frames, we have 3.42:

$$H_{C_{i,j}} = \begin{cases} \prod_{k=i}^{j-1} H_k, & i < j \\ \left( \prod_{k=j}^{i-1} H_k \right)^{-1}, & j < i \\ H_{C_{i,i}} & i = j \end{cases} \quad (3.42)$$

Where  $H_{C_{i,i}}$  is the  $3 \times 3$  identity matrix. Discrete Gaussian filter is an approximation by sampling and truncating the continuous Gaussian of equa-

tion 3.43. Parameter  $a$  is related to  $3dB$  bandwidth-symbol time  $BT_s$  given by 3.44, where  $T_s$  is the sampling period[12][13]

$$h(t) = \frac{\sqrt{\pi}}{a} e^{-\frac{\pi^2 t^2}{a^2}}, \quad (3.43)$$

$$BT_s = \frac{1}{a} \sqrt{\frac{\log 2}{2}}. \quad (3.44)$$

In *videostab*, a discrete Gaussian impulse response is sampled in a finite number of frames from  $-r$  to  $r$  is in equation 3.45, where  $G = \sum_{i=-r}^r e^{-\left(\frac{i}{\sigma}\right)^2}$  and  $k = [-r, r]$ . From equation 3.43 and 3.45 we have  $\sigma = a/\pi$ . Then,  $3dB$  bandwidth for 3.45 is in equation 3.46.

$$g[k] = \frac{1}{G} e^{-\left(\frac{k}{\sigma}\right)^2} \Big|_{k=-r}^r, \quad (3.45)$$

$$f_{3dB} = \frac{1}{\sigma\pi} \sqrt{\frac{\log 2}{2}}. \quad (3.46)$$

Figure 3.11 plots equation 3.45 and its respective magnitude response in Figure 3.12 with  $r = 20$  and  $\sigma = 4$ . Parameter  $r$  also represents sampling error or aliasing due to the fact that a Gaussian frequency response is not really band-limited in a strict sense. Choosing  $r$  is a compromise with computational effort and latency which appears because we need to fill the buffer with  $r$  forward frames before computing corrections. Latency depends on video stream rate ( $fps$ ) as following:  $latency = \frac{r}{fps}$ .

Meaning of  $f_{3dB}$  depends on video rate  $fps$ . In Figure 3.12 we can see  $f_{3dB} \approx 0.0468$  hence the value of  $fps \times f_{3dB} \approx 1.4054$  which means that the energy of motions changing at 1.4054 frames per second, will be decreased at the half of initial motion energy. For faster motions, attenuation would be bigger than the half.

Another fact is the filter Phase Response. A linear phase is important for avoid introducing distortions in filtering process. For a finite impulse response filter, having symmetry in the equation 3.45 is enough condition to get a linear phase (in the attenuation region) as is shown in Figure 3.13.



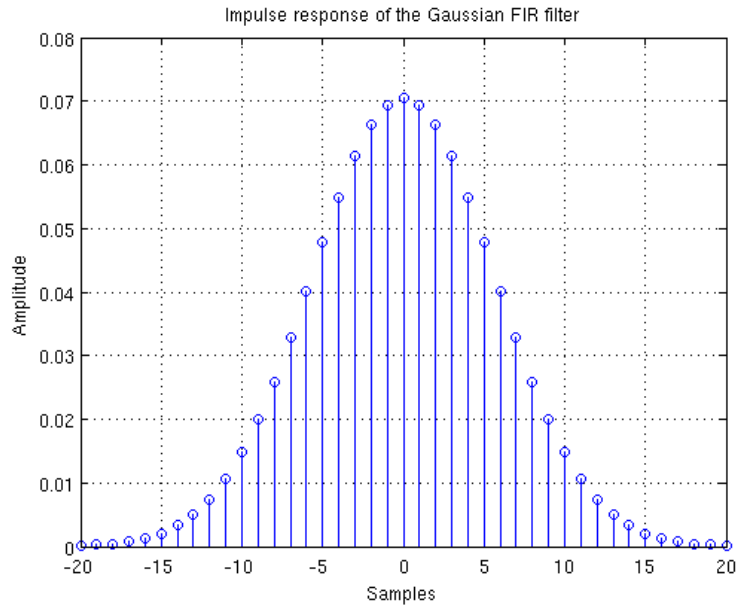


Figure 3.11: Impulse Response of Gaussian filter with  $r = 20$ .

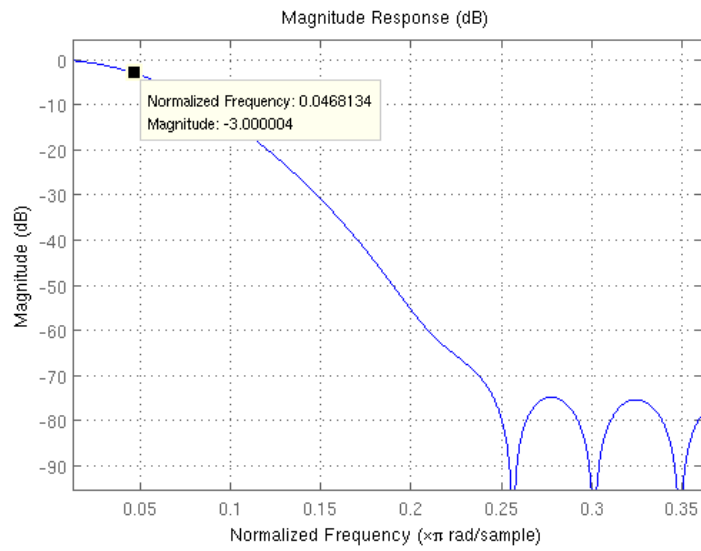


Figure 3.12: Magnitude Response of Gaussian filter.

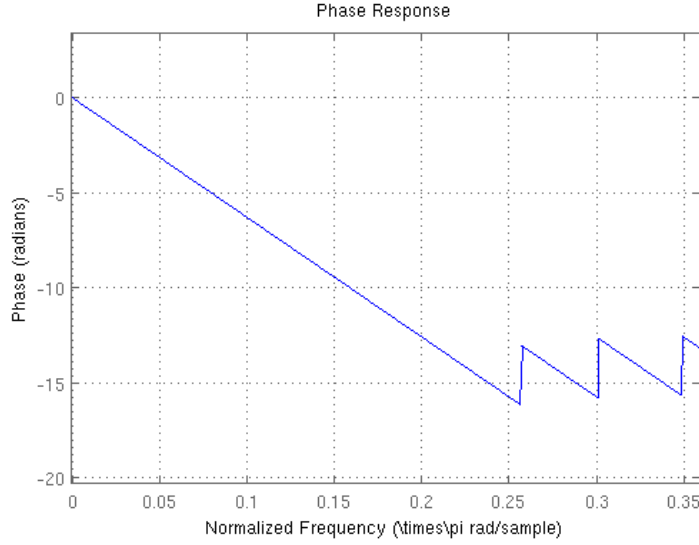


Figure 3.13: Phase Response of filter.

Mathematical description to obtain transformation matrix  $H_{stab}$  to stabilize the current frame can be expressed as in equation 3.47. Transformation matrix,  $H_{stab}$  stabilizes frame at position  $n$ . In Figure 3.14 there is a graphical representation of how to obtain  $H_{stab}$  for filtering the video stream.

$$H_{stab}[n] = \sum_{k=-r}^r H_{C_{n,n+k}} g[n]. \quad (3.47)$$

In Figure 3.14, having the buffer  $b_{H_C}$  of cumulative motions around  $n$ ,  $H_{stab}$  can be expressed in equation 3.48 as a convolution with the discrete filter  $g$ .

$$H_{stab}[n] = (b_{H_C} \star g)[n]. \quad (3.48)$$

Particular cases rise for initial and final video frames when there is less than  $r$  backwards frames or forwards respectively to fill the buffer of frame to frame motions. In those cases,  $3 \times 3$  identity matrix can be used while buffer is filled. Gaussian motion filter is found in *videostab* module as a class named **GaussianMotionFilter** inheriting attributes from abstract class **MotionFilterBase**.

### 3.3.2 Zero Motion Filter

Zero Motion filter is an implementation assuming zero intended motion from a specific frame at position  $z$ . All following frames must be transformed with

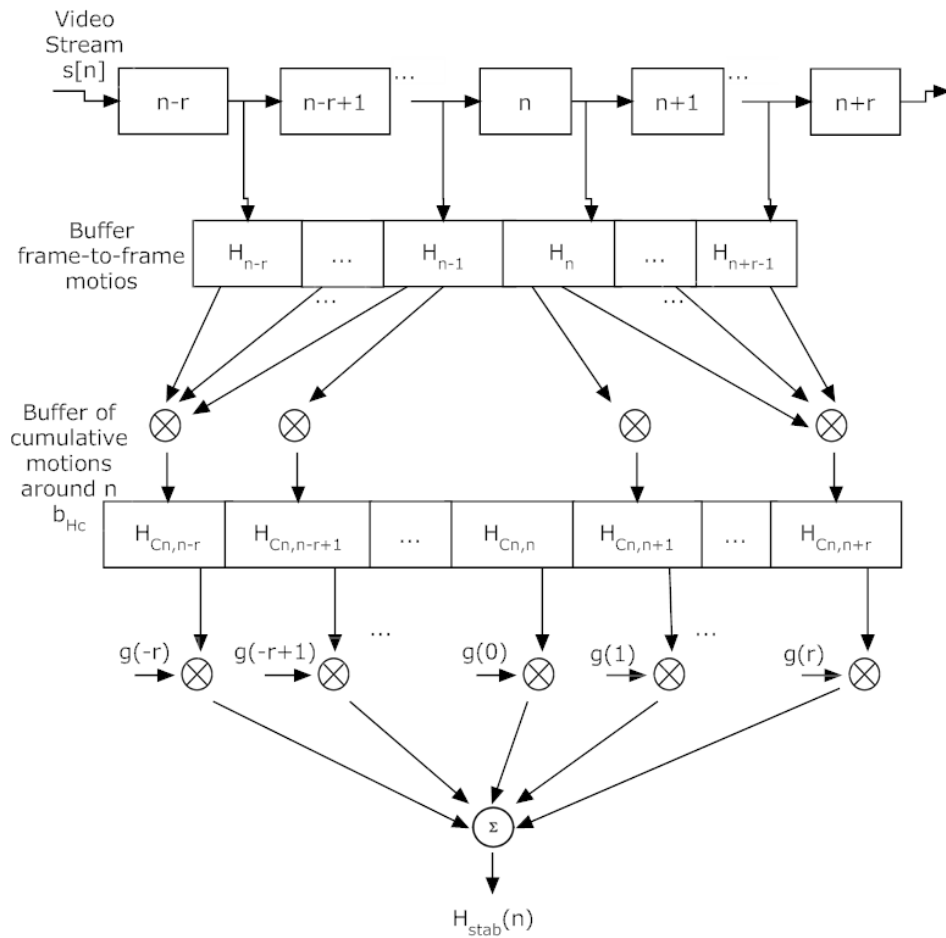


Figure 3.14: Schema for apply the filter to the video stream.

respect to frame at  $z$  with  $H_{stab}$  equals to the product of all consecutive cumulative motions. There are two problems with this implementation. First, we accumulate error on every estimation. Another problem is when assumption of zero ego-motion is far to be accomplished. We have the risk of not being able to transform the current frame into the reference frame at  $z$  because the motion are out of the image range. A dummy solution for both problems is to drop the reference frame, updating it every  $r$  frames to initialize cumulative error. Implementation was named **ZeroMotionFilter** inheriting attributes from abstract class **MotionFilterBase**.

### 3.3.3 Image Alignment

After filtering frame  $I_n$  we obtain transformation matrix  $H_{stab}[n]$  to remove the current jitter creating an aligned frame  $I_{n-stab}$  by transforming  $\mathbf{x} = (x, y)$  to  $\mathbf{x}' = (x', y')$  as in 3.49. Alignment is the last step of the process, done by OpenCV with functions **warpAffine** and **warpPerspective** for Affine model and for Homography model respectively. In general, transformation domain values are not integer and for that reason we should chose an interpolation methods for warping functions. Nearest-neighbor interpolation was set for UAV since it is the fastest computationally

$$I_{n-stab}(x, y) = I_n(x', y'), \quad (3.49)$$

where, for Affine we have:

$$\begin{aligned} x' &= H_{stab}(1, 1)x + H_{stab}(1, 2)y + H_{stab}(1, 3), \\ y' &= H_{stab}(2, 1)x + H_{stab}(2, 2)y + H_{stab}(2, 3); \end{aligned}$$

and for Homography:

$$\begin{aligned} x' &= \frac{H_{stab}(1, 1)x + H_{stab}(1, 2)y + H_{stab}(1, 3)}{H_{stab}(3, 1)x + H_{stab}(3, 2)y + H_{stab}(3, 3)}, \\ y' &= \frac{H_{stab}(2, 1)x + H_{stab}(2, 2)y + H_{stab}(2, 3)}{H_{stab}(3, 1)x + H_{stab}(3, 2)y + H_{stab}(3, 3)}. \end{aligned}$$

An example of overlapping two consecutive images from UAV after alignment is in Figure 3.15. Regions in red are pixel values in which previous images has bigger values than current image (vice versa, cyan regions). To decrease the effect of annoying undefined pixels at borders, we can trim frames borders to have smaller output video size. Furthermore, *videostab* implements an Inpainting technique called Fast Marching[18] to make quick correction in remaining undefined areas of video.

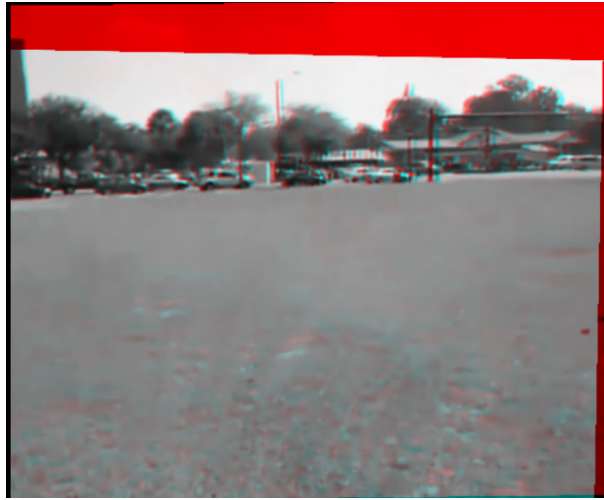


Figure 3.15: Alimnet of two consecutive images.

### 3.4 Diagrams

A class diagram (in Section A.1 ) and Activity Diagrams (in Section A.2) of implementations are presented for *videostab* module In Appendix A with a high level of abstraction. The intention is to give a global view of modules that make up the system and the relevant dynamic aspects, but it does not exactly match the code.



## Chapter 4

# Experiments and Discussion

In this chapter we show experiments for some blocks and for the entire system. We formulate some questions which are solved after analyzing graphical results. Links to other experiments can be found in Appendix C.

### 4.1 Performance Analysis and Discussion

We analyze accuracy and computational time for different system blocks. Different video scenes from real UAV were used for experimentation. Table 4.1 contains information about used videos for testing.

#### 4.1.1 Feature Selection Quality: Texturedness

Section 3.2.2 describes the process of selecting points to track, but not how many of them we should use. The question is important due to trading off between accuracy and computational speed. To know this, we plot feature quality values (from equation 3.23) for different images.

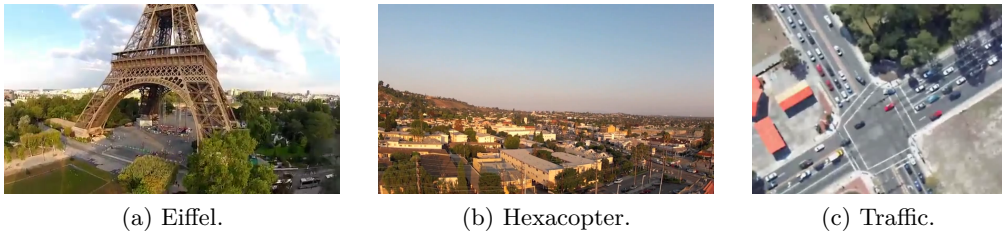


Figure 4.1: Test video frames.

Name	Frame Size	fps	Length
<i>Eiffel.mp4</i>	360 × 640	29	331
<i>Hexacopter.mp4</i>	720 × 1280	29	481
<i>Traffic.avi</i>	23 × 40	25	22

Table 4.1: Information about videos for testing.

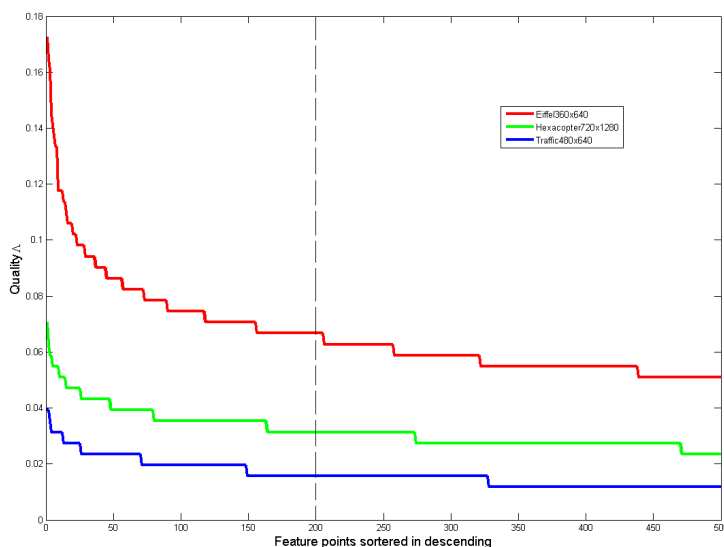


Figure 4.2: Feature quality for frames in Figure 4.1.

Footage in Figure 4.1 were used for testing. Images were converted to gray scale. Size of windows in equation 3.25 and minimum distance among features were 3 and 1 pixels respectively. Figure 4.2 shows the result for the first 500 key points in descend order by quality. Quality values were normalized with respect to maximum pixel value. For example, if image was coded with 8 bit per pixel, then normalization value is 255.

One conclusion is that order quality values does not depend of image size but image scene. We see, for instance, values for picture 4.1a with Eiffel tower in foreground overcomes other pictures, however, it was not the biggest sized. As we expected, frame 4.1c presents lowest quality values since that frame was slightly blurry.

A negative power decay was observed for all graphics. Same shape was observed with other sample images. About the first 200 key points exhibit remarkable value. It was more evident with an histogram where 10% of the lowest rank values, consisted of about 300 feature points. In conclusion we suggest to work with 200.



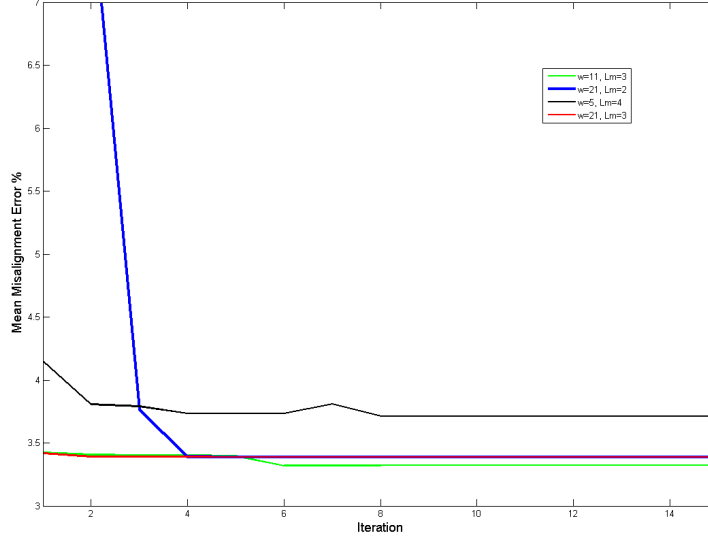


Figure 4.3: Iteration Vs Error in tracking over four different combinations of windows and pyramid level.

#### 4.1.2 Feature Tracking Accuracy: Dissimilarity

Texturedness tries to alleviate some possible problem in tracking, as aperture problem. On the other hand, dissimilarity shows how effective was tracking by measuring aligning error around each tracked point. Three parameters rule runtime tracking performance: pyramid maximum level  $L_m$ , window size  $2w + 1$  and maximum number of iteration  $maxIter$  for displacement computation at each pyramid level.

To know which parameter combination we should use, we propose plot the tracking error with respect to number of iterations for a fixed pair  $(L_m, w)$ . Error is the average for all features ( $n = 200$ ) of the  $L_1$  distance between patches around the original and a point moved  $\Delta \mathbf{x}$ , divided by number of pixels in a window (see equation 4.1)

$$\frac{1}{n} \sum_{i=1}^n \left[ \frac{1}{(2w + 1)^2} L_1(\Delta \mathbf{x}_i) \right]. \quad (4.1)$$

$L_m$  defines the figure size at top of pyramid dividing the original size by  $2^{L_m}$ . Table 4.2 shows values of size at maximum level of the pyramid. For  $L_m = 5$  we found frame sizes too small if we take into consideration that windows sizes values are around 20 and 60 pixels.

$L_m$	Eiffel	Hexacopter	Traffic
2	$90 \times 160$	$180 \times 320$	$120 \times 160$
3	$45 \times 80$	$90 \times 160$	$60 \times 80$
4	$23 \times 40$	$45 \times 80$	$30 \times 40$
5	$11 \times 20$	$23 \times 40$	$15 \times 20$

Table 4.2: Size of frames at maximum pyramid level.

Window sizes  $(2w + 1)$  is another trade off among alignment precision and robustness against local minimum. Additionally, combined with  $L_m$  gives the maximum inter-frame pixel displacement  $2^{L_m} \times w$ .

For testing we use two consecutive frames from video 4.1a. In Figure 4.3 we measure the error defined in equation 4.1 on each iteration for four combination  $w$  and  $L_m$  using 200 key points. Error values were normalized to a percentage of maximum pixel value.

In Figure 4.3, for black line, we could see that it did not converge with other lines. The reason was to use a large  $L_m = 4$ . What probably happened is that before reaching the bottom of pyramid ( $L = 0$ ), the algorithm deviated to a local minimum. For  $L_m = 2$  (blue line), iterations at  $L = 0$  start with a higher error value compare with the others. The reason is that there were not enough pyramid levels for reaching to level zero with an accurate guessing.

Then we found, the optimal maximum pyramid level should be equal to 3 and more specifically to use a windows with  $w = 11$  (green line) which offer slightly lower error, keeping a pertinent range of maximum allowable frame to frame pixel displacement of 88 pixels ( $2^3 \times 11$ ). For this test, on average, the final misalignment error was 3.4% by feature with 15 iteration on each pyramid level. It means on average the precision for tracking at every tracked point was 96.6%. Drift error may occur for many factors, including noise in acquisition, illumination variation, among others.

To figure out if we can use a lower number of iterations we tested tracking using 200 features points, a window of size 23 ( $2 \times 11 + 1$ ) and  $L_m = 3$ . We Showed in Figure 4.4 iteration versus error (not normalized) in a sequence of 2000 frames for video Eiffel.

We observed a nearly constant tendency after 5 iterations. This is because when we arrive to the lowest level of pyramid, initial guess contributes to start with a low error from the first iteration. It means that we can set the algorithm with a maximum number of iterations around 7 saving computational time and running it in constant time.

For more graphical results in Figure 4.5 we show tracking between 2 consecutive frames. Parameters were set as following: 200 feature,  $w = 11$ ,

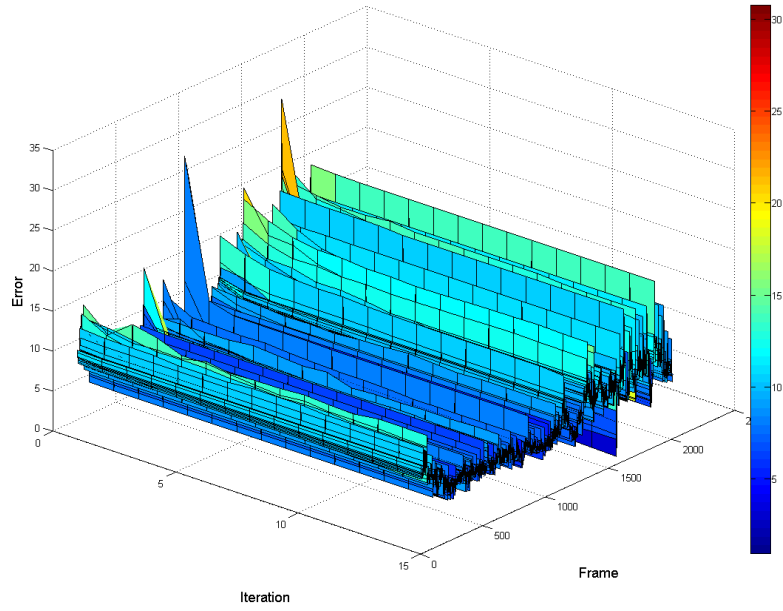


Figure 4.4: Iteration Vs Error for 2000 consecutive frames.

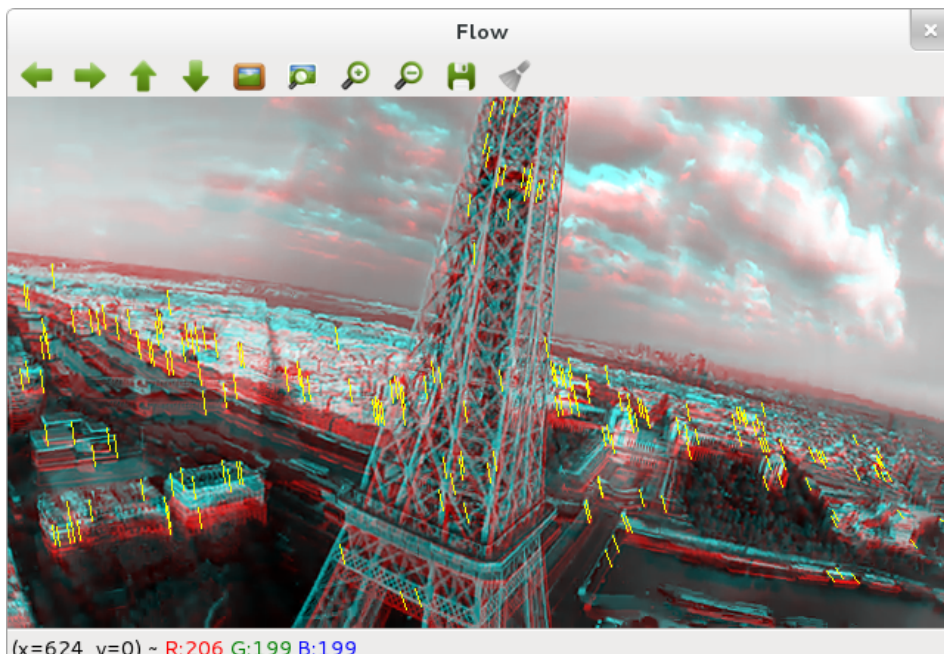


Figure 4.5: Motion Vectors.

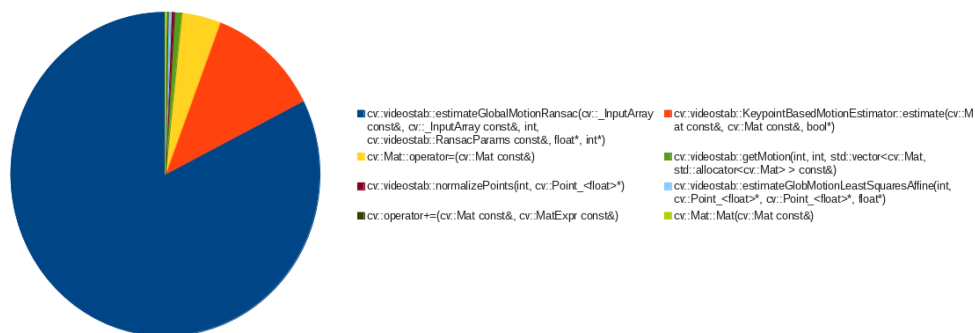


Figure 4.6: Profile Pie Chart.

$L_m = 3$  and 5 iterations. Two consecutive frames are overlapped. Regions in red are pixel values in which previous images has bigger values than pixels in current image (vice versa for cyan regions). Motion vectors are plotted in yellow.

### 4.1.3 Profiling

Profiling is an important aspect of software programming. Through profiling one can determine the parts in program code that are time consuming and need to be optimized. To attempt make the program faster, we use gprof which is a GNU profiler tool<sup>1</sup>. We obtained a profile table with all functions in *videostab*.

Sorting the mentioned table by time consumed and calls by function, we selected the 8 more expensive function and graph them in a pie char (see Figure 4.6). Function **estimateGlobalMotionRansac** was clearly the most time expensive with about 80% of total execution time.

We saw in Section 3.2.3, that, to ensure robustness **estimateGlobalMotionRansac** executes  $N$  iterations to guarantee (with probability  $p$ ) outlier rejection (for threshold  $\tau$ ) from a set with maximum outliers ratio  $\epsilon$ . In Figure 4.7 we plot the number of iterations for RANSAC with 0.99% of success probability and the subset  $m = 3$  for the Affine. The domain represents the maximum outliers ratio  $\epsilon$ .

Accordingly, transformation matrix for  $n$  points is computed  $N$  times. The computational complexity or computational order gives an idea of the number of instructions executed for an algorithm. The order to compute transformation matrix is  $O(2 \times n)$ . Hence, the total order  $O_T(\bullet)$  for robust motion estimation is given by equation 4.2.

$$O_T = O(N) \times 2O(n) = N \times n. \quad (4.2)$$

<sup>1</sup><http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html>

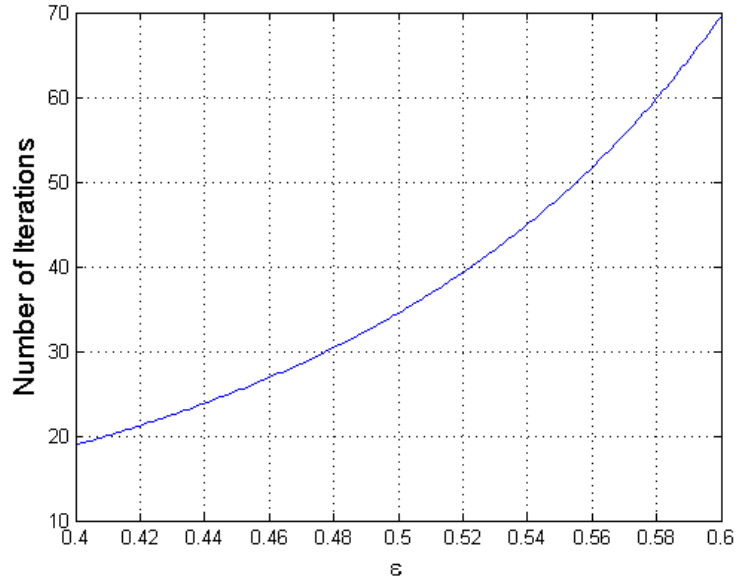


Figure 4.7:  $\epsilon$  Vs Number of iterations.

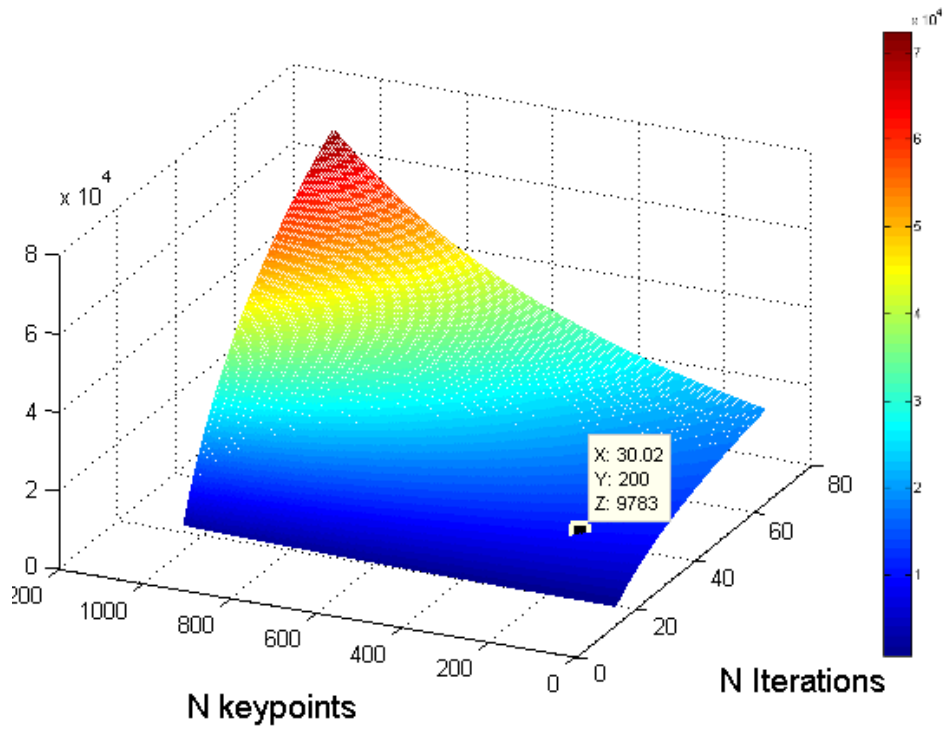


Figure 4.8: Order of computational complexity for motion estimation.

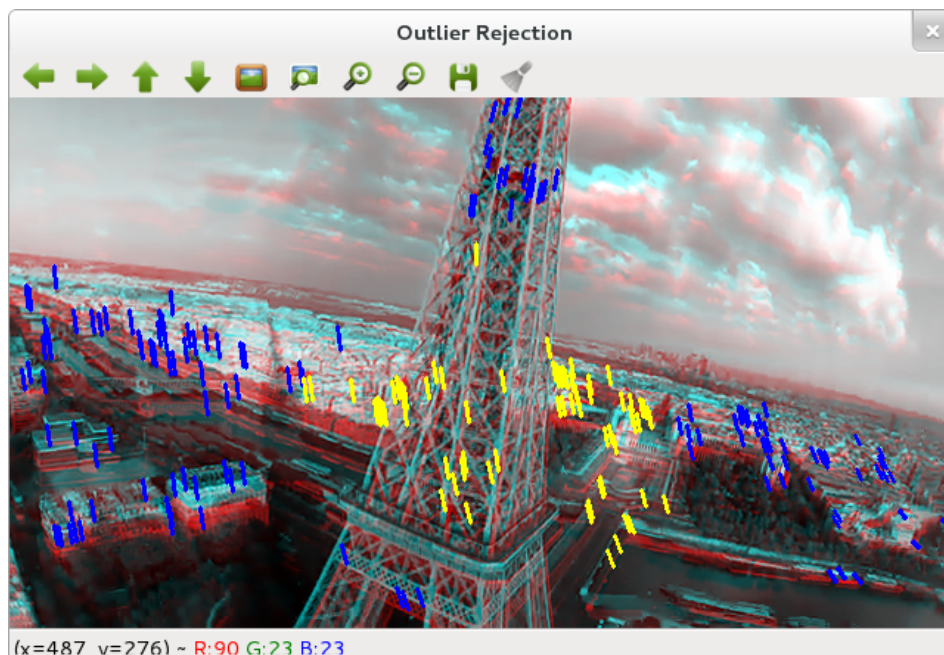


Figure 4.9: Outlier Rejection.

In Figure 4.8 we graph the number of computations. Iterations do not vary linearly because they depend on  $\epsilon$  as in Figure 4.7. For  $N = 30$  ( $\epsilon = 0.48$ ) and  $n = 200$  we have 9783 computations. With these values we keep in dark blue part of the graphic, with computation of the order of  $10^3$  instead of  $10^4$ .

#### 4.1.4 Outlier Rejection in Global Motion Estimation

An example of outlier rejection result with parameter set as in previous subsection ( $\epsilon = 0.48$  and  $n = 200$ ) and threshold  $\tau = 0.5$  is shown in Figure 4.9. Inliers are drawn in yellow (in total 66 inliers around Eiffel tower).

Some motion vectors in blue are evidently outliers due to their orientation (see for instance, the ones in tower's top and the ones at right side). More than 48% of data were considered outliers (134 motion vectors). The conclusion is that we can not accomplish the assumption with 99% as we assumed with  $\epsilon = 0.48$ . The reason is the strict threshold of 0.5 which means that model must fit sub-pixel precision of 0.5 pixels. We should use a less constrained  $\tau = 0.75$ .

### 4.1.5 Motion Filtering

We can not give a precise numeric value of motion filtering quality because the final effectiveness depends on many factors changing frame by frame as: motion estimation, characteristics of the scene (type of vibration, motion of the objects in scene, relative speeds, texture of the images, so on). However, we can obtain the motion from estimated Affine model and show visual results after smoothing.

Affine has 6 degree of freedom. However, we use 4 to show movements in 2D and scaling as for similarity motion model (see equation 4.3). This model allows for rotation ( $\theta$ ), uniform  $x$  and  $y$  scaling ( $s$ ), and  $x$  and  $y$  translation( $T_x, T_y$ ).

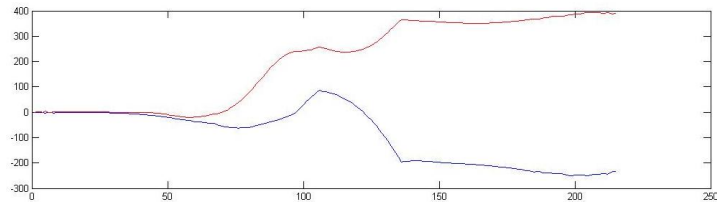
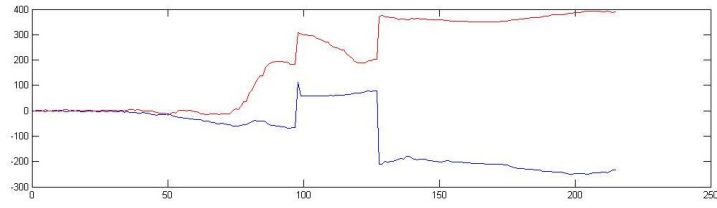
$$H_{Affine} = \begin{bmatrix} s \cos(\theta) & -s \sin(\theta) & T_x \\ s \sin(\theta) & s \cos(\theta) & T_y \\ 0 & 0 & 1 \end{bmatrix}. \quad (4.3)$$

From 4.3, we obtain an approximation to displacements, rotations and scales as:

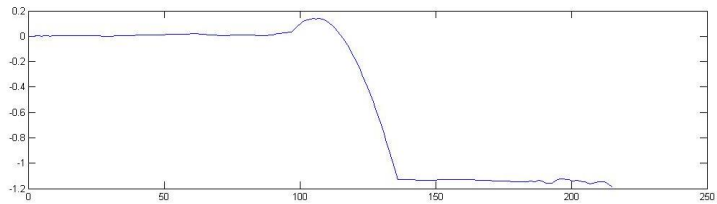
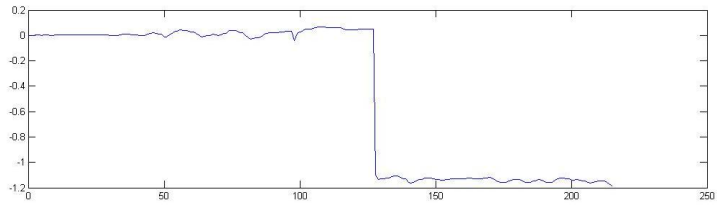
$$\begin{aligned} Displacement &= \begin{bmatrix} T_x & T_y \end{bmatrix}^T \approx \begin{bmatrix} H_{Affine}(1, 3) & H_{Affine}(2, 3) \end{bmatrix}^T, \\ Rotation = \theta &\approx \left( \arctan\left(\frac{-H_{Affine}(1, 2)}{H_{Affine}(1, 1)}\right) + \arctan\left(\frac{H_{Affine}(2, 1)}{H_{Affine}(2, 2)}\right) \right) \times 2^{-1} = \theta_{Approx.}, \\ Scaling = s &\approx \left( \frac{H_{Affine}(1, 1)}{\cos(\theta_{Approx.})} + \frac{H_{Affine}(2, 2)}{\cos(\theta_{Approx.})} \right) \times 2^{-1}. \end{aligned}$$

For testing this, we use a video of size  $480 \times 640$  called *taking\_off.avi*. The scene is an highly shaky footage with an UAV taking off from the ground. Parameters for estimation were set as we discussed in this section. Stabilization radius  $r$  was set to 20.

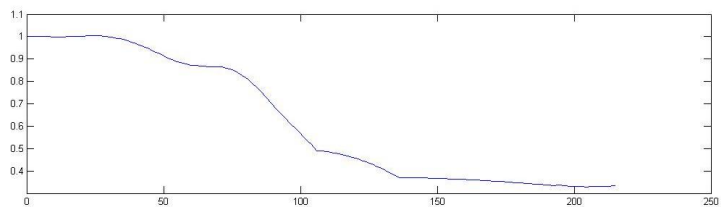
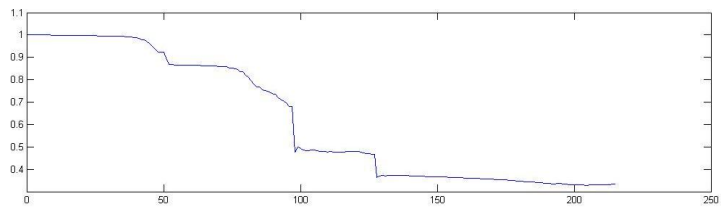
In Figure 4.10 we show result of cumulative motions before (upper part of each graph) and after filtering. We could see the effect of previous and consecutive frames. With some intuition we can see that the UAV rises up since there are positive translations in  $y$ . At the very beginning of each graph there was no much motion, which corresponds to the real video when the UAV starts turning on helix. For rotation we saw a phase change at the middle of footage. For scaling we observed that cumulative depth of background is less than 1 which means we moved away (at the beginning, ground was at foreground).



(a) Translation estimation: blue  $x$ , red  $y$



(b) Estimated Rotation.



(c) Scaling Estimation.

Figure 4.10: Motions before and after filtering.



Another example in Figure 4.11 shows  $x$  and  $y$  axis together. We can observe that even with large  $(x, y)$  displacement (dashed line without markers at the right side of figure), the filtering process smoothly interpolate positions. In conclusion, it is possible, to measure the amount of correction needed by a visual motion compensation algorithm to stabilize the images by combining estimated displacements, rotations and scaling at running time and make correction of UAV's engine.

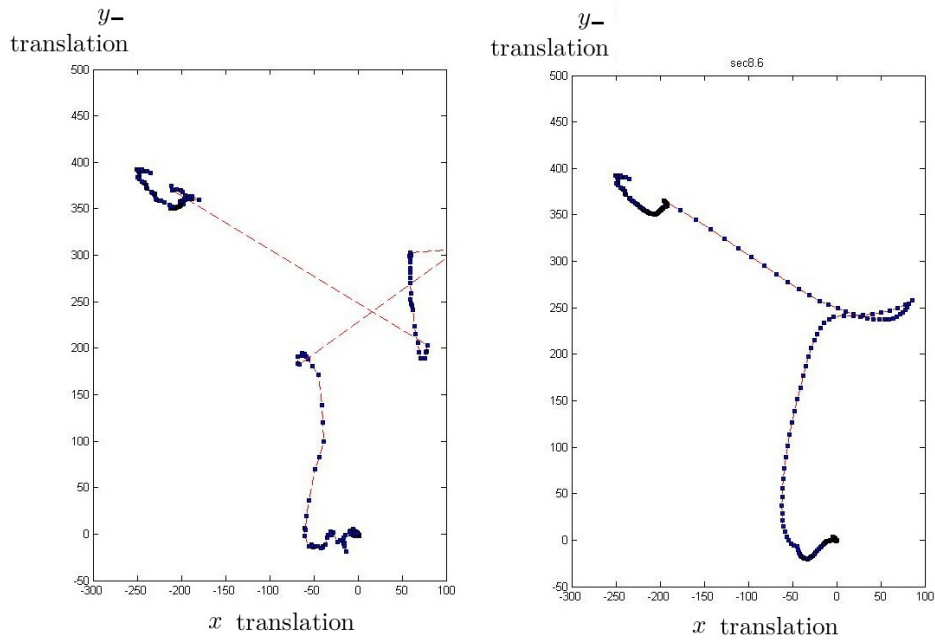


Figure 4.11: Translation before and after filtering.



## Chapter 5

# Conclusion and Further Works

Video stabilization for UAVs highly consumes computational time mainly because we want to ensure robustness during motion estimation, nevertheless, UAV's video provides large set of data inherently corrupted by high noise levels. Feature-based method, with Tomasi-Shi and Lucas-Kanade algorithms, help to quickly select input data to estimate inter-frame motion, however not all samples adequately fit motion models. Not only because changes in assumptions of external situations in feature selection and tracking with mentioned algorithms (brightness constancy, spatial coherence, temporal persistence, flat world) but also because of internal conditions: noise in acquisition system, image distortion due to camera parameters and more complexity of real world motions than parametrized motion models.

An evidence of the latter asseveration can be found testing **ZeroMotion-Filter** using Affine model versus Homography model (In Appendix C, see example for Section C.1).

A solution to increase quality in alignment is to include camera parameters in computations by measure them before UAV starts recording. Knowing the distortion that the camera introduces in footage allows to correct each frame and to have better alignment. Furthermore, after correction, probably many motion vector will not longer be consider as outliers, allowing us to decrease outlier ratio, hence decrease the iterations in motion estimation, making faster computations.

To increase fidelity of motion estimation and even do it faster, another idea is dynamically change the motion model (Rigid, Affine, Homography). To do so, the system should be able to identify when it is necessary a more complex motion model than Affine, then jump to projective (full homography). The other way around is when at some moment, it is enough to use

less computational complex motion model as similarity, rigid or translation.

Talking about smoothing movements, with Gaussian FIR filter we can successfully achieve online stabilization paying an unavoidable price of latency due the radius of stabilization. As a further work, another stabilizing subsystem might be implemented and compared with current version. It might be either filter-based (as Kalman filter, particle filter, among others), linear programming-based or any other stabilizing method. Class diagram in Figure A.1 provides a structure to easily integrate new stabilizer solutions in **motion\_stabilizing** package with the class **MotionFilterBase** implementing the interface **IMotionStabilizer**.

Finally, we saw in Section 4.1.5 that measured movements corresponds to real motion of UAV. Displacement, rotation and scaling from Affine might be used as index of the amount of correction needed to stabilize the aerial vehicle. This opens the door for using the stabilization process in tuning the control parameters from UAVs. To see a graphical test showing a global index of measured vibration while UAV is on the flight, refer to Section C.2 in Appendix C.

## Appendix A

# Logic Project Documentation

A.1 Class Diagram

A.2 Activity Diagrams



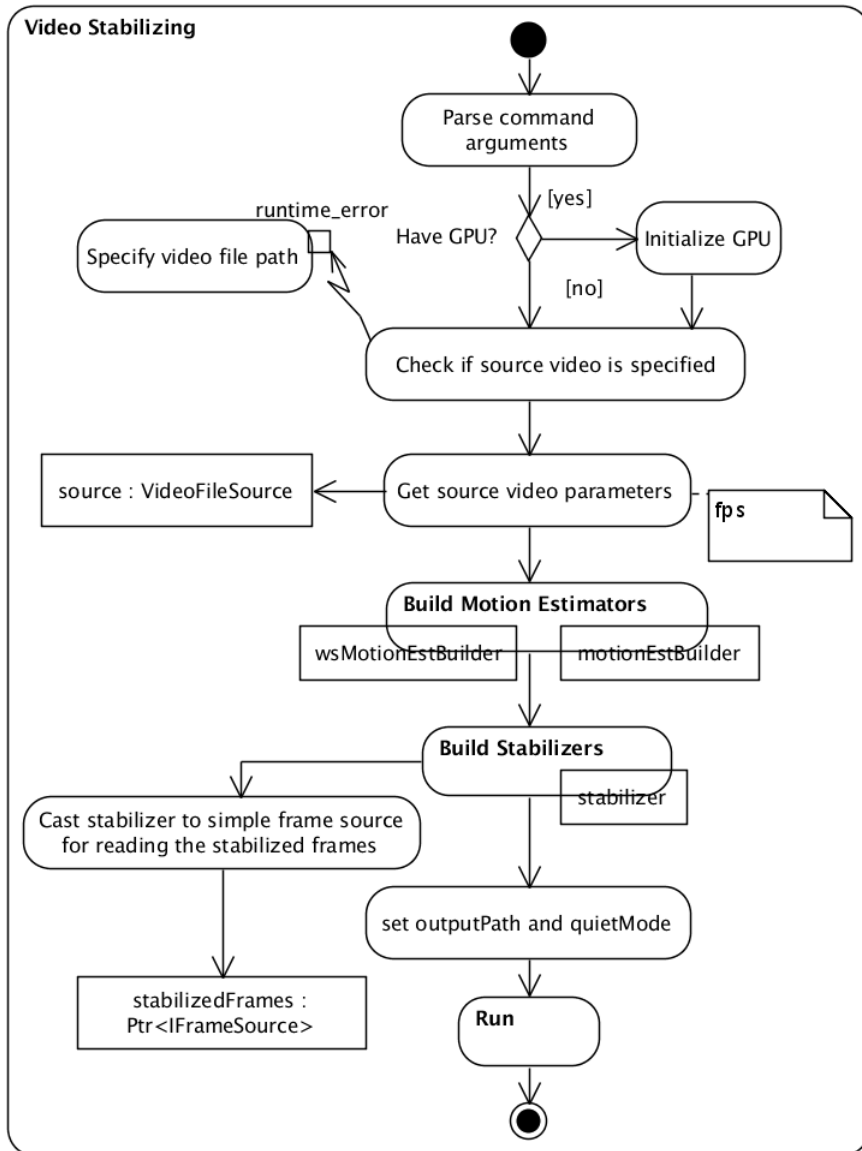


Figure A.2: Video Stabilizing.

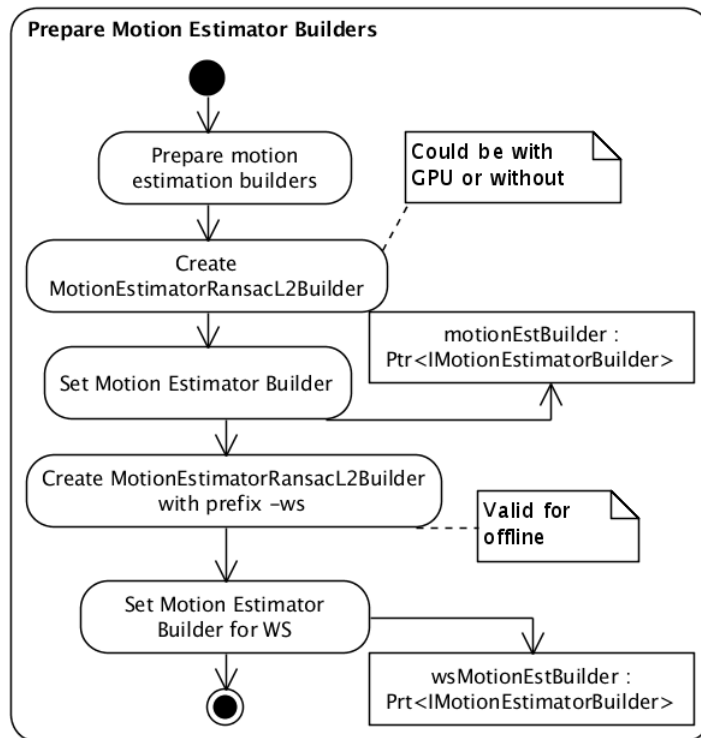


Figure A.3: Prepare Motion Estimation Builders.



APPENDIX A. LOGIC PROJECT DOCUMENTATION

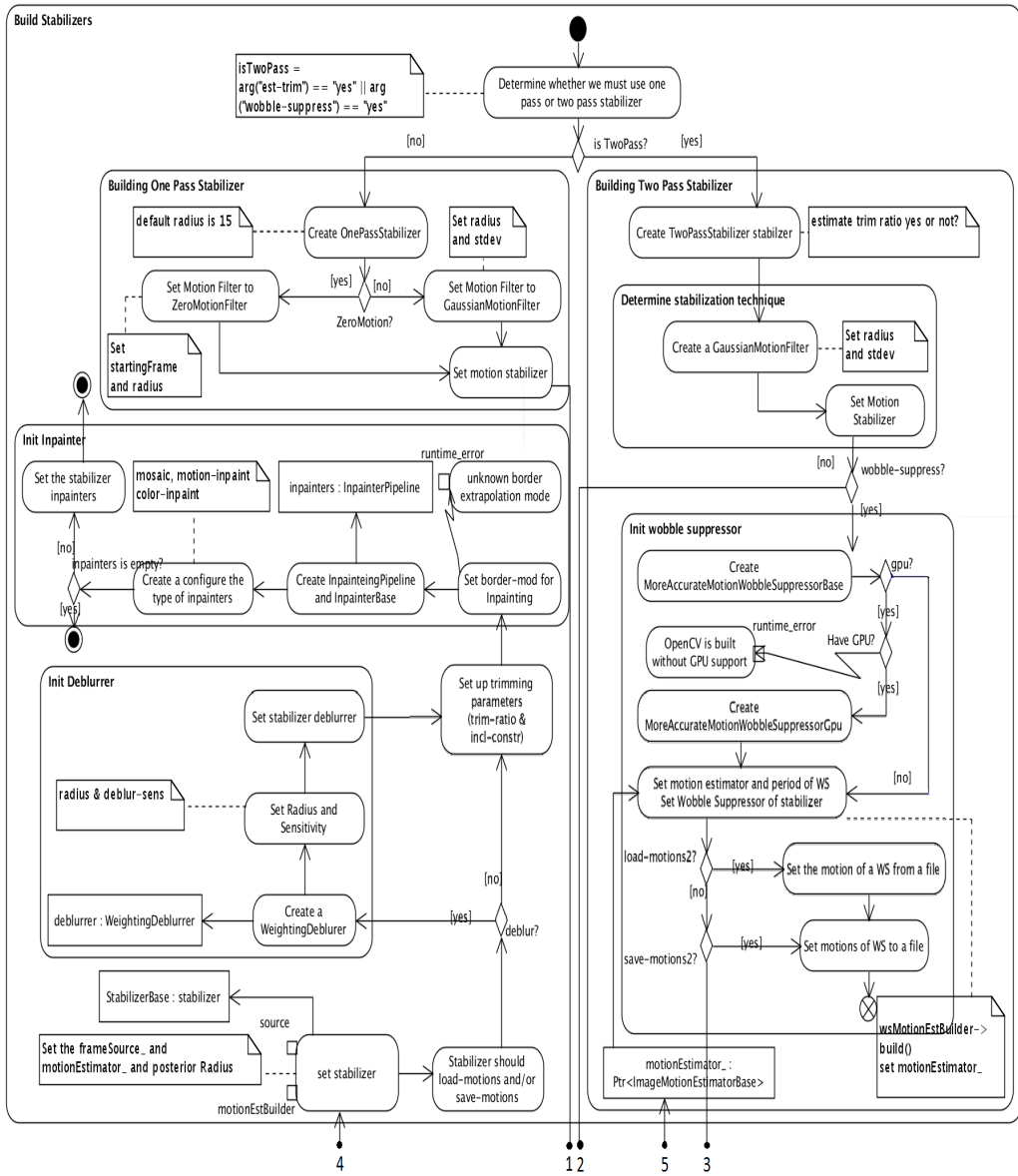


Figure A.4: Building Stabilizers.

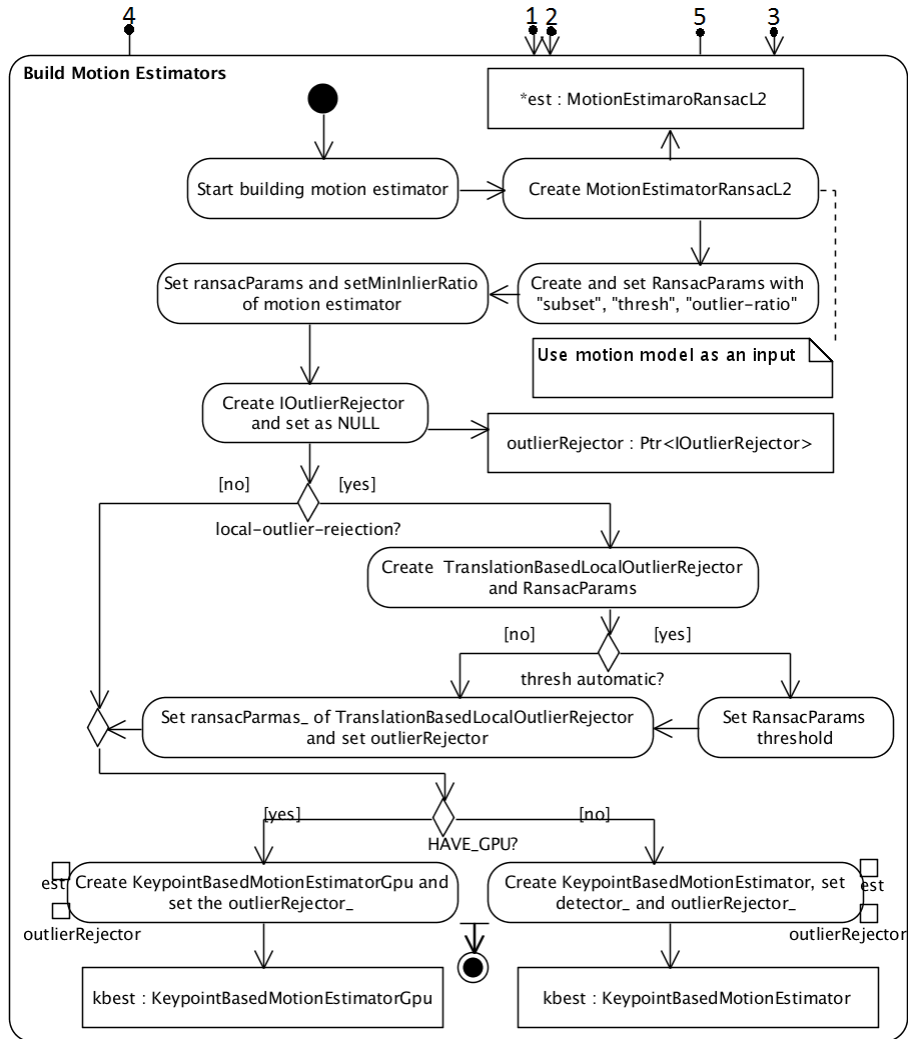


Figure A.5: Build Motion Estimators.

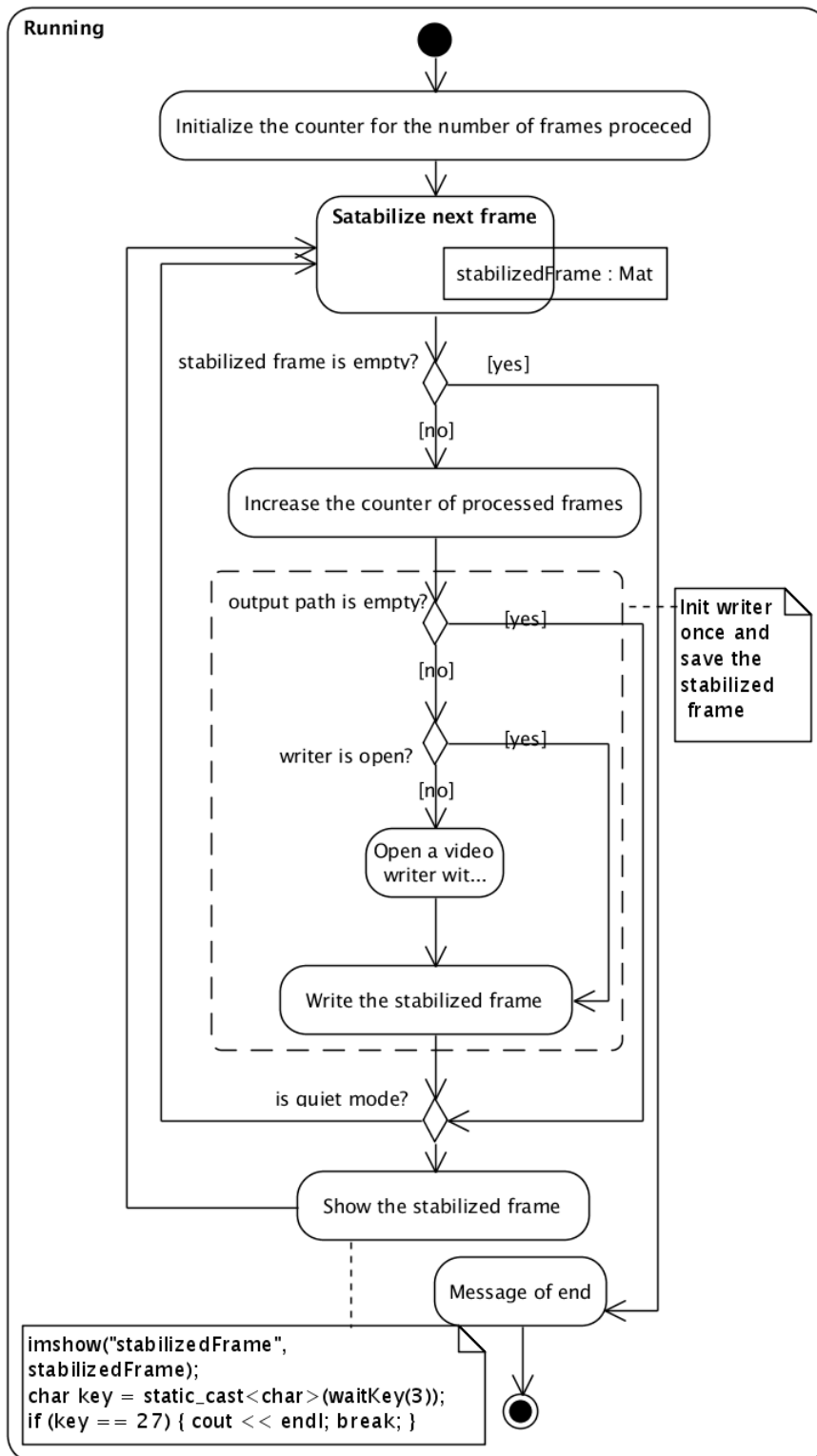


Figure A.6: Running the system in the highest level of abstraction

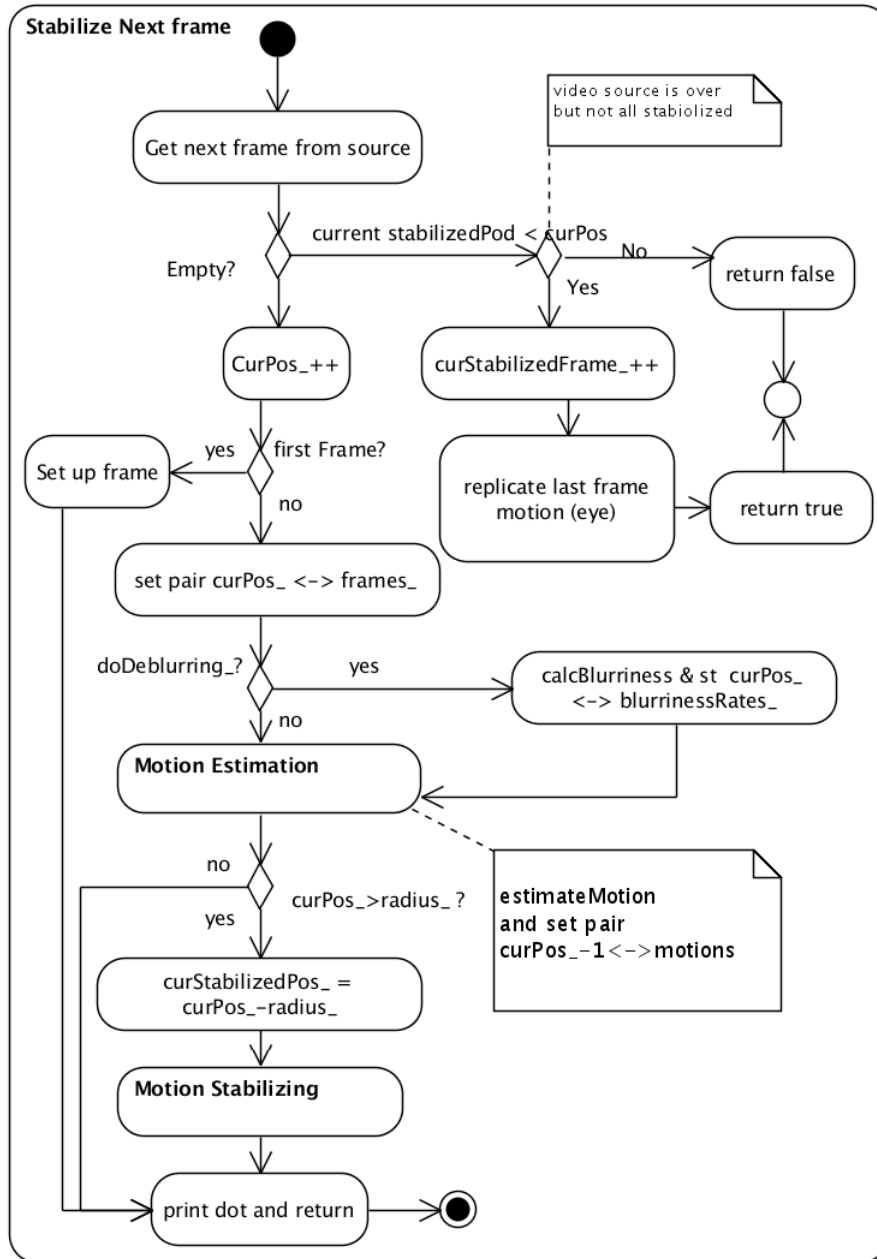


Figure A.7: Stabilize Next Frame.

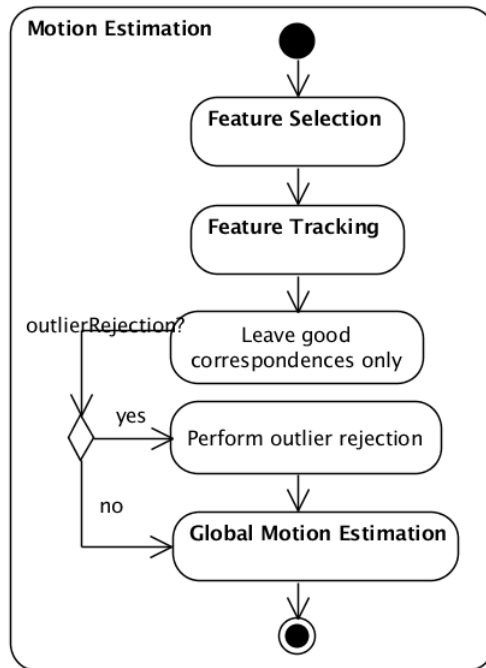


Figure A.8: Motion Estimation.

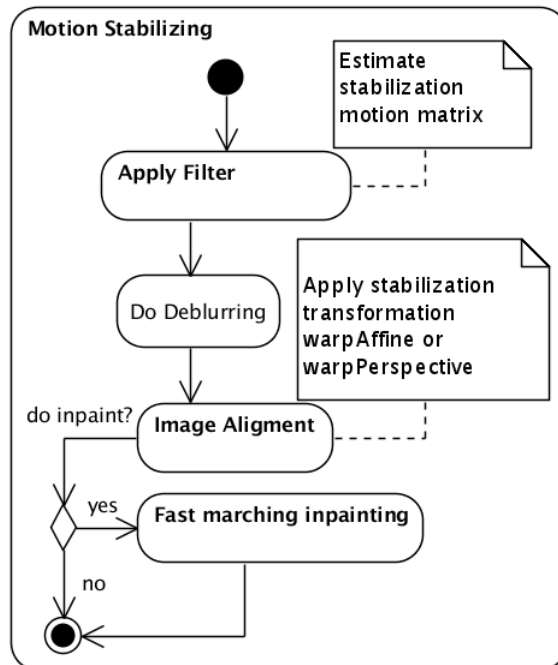


Figure A.9: Motion Stabilizing.



## Appendix B

# User Manual

To use the code clone or fork from repository in GitHub: [https://github.com/rodrygojose/opencv/tree/videostab\\_polimi/modules/videostab\\_polimi](https://github.com/rodrygojose/opencv/tree/videostab_polimi/modules/videostab_polimi)

Folder named *include* contains header files. Folder named *src* contains .cpp files. Inside src, *videostab.cpp* contains main function and interface for input parameters. To the parameter use *-help*.

To compile the code, the minimum OpenCV version is 2.4.0.





## Appendix C

# Result Examples

### C.1 Affine vs Homography

<http://www.youtube.com/watch?v=SzHAAutvjaE>

### C.2 Amount of Correction

<http://www.youtube.com/watch?v=ZyMAllty7DA>



# Bibliography

- [1] *The OpenCV Reference Manual - Release 2.4.5.0*. 2.4
- [2] Mohammed A. Alharbi. Fast video stabilization algorithms. Master's thesis, Department of Electrical and Computer Engineering, Air Force Institute of Technology, 2006. 1.2
- [3] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *Int. J. Comput. Vision*, 56(3):221–255, February 2004. 3.2.1.2
- [4] Jean-Yves Bouguet. Pyramidal Implementation of the Lucas Kanade Feature Tracker: Description of the Algorithm. Technical report, Intel Corporation Microprocessor Research Labs, 2000. 3.2.1.3, 3.2.1.4, 3.2.2
- [5] Nicholas Stewart Cross. Onboard video stabilization for unmanned air vehicles, 2011. 2.1
- [6] Elan Dubrofsky. Homography estimation, 2009. 3.2.4
- [7] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981. 3.2.3, 3.2.3
- [8] SRI International. Acadia, real-time video processors. 1.5
- [9] David L. Johansen. Video stabilization and target localization using feature tracking with small uav video. Master's thesis, Brigham Young University, December 2006. 3.2, 3.2.3, 3.3
- [10] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. pages 674–679, 1981. 3.2, 3.2.1.1, 3.2.1.1
- [11] Xavier Marichal. *Motion Estimation and Compensation for Very Low Bitrate Video Coding*. PhD thesis, Universite catholique de Louvain, 1998. 2.2

## BIBLIOGRAPHY

---

- [12] Signal Processing Toolbox MathWorks. Fir gaussian pulse-shaping filter design. 3.3.1
- [13] Y. Matsushita, E. Ofek, Weina Ge, Xiaou Tang, and Heung-Yeung Shum. Full-frame video stabilization with motion inpainting. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(7):1150–1163, 2006. 3.3.1
- [14] PJ Rousseeuw. Least median of squares regression. *Journal of the American statistical association*, pages 79 (388), 871–880, 1984. 3.2.4
- [15] S. Schwertfeger, A. Birk, and H. Bulow. Using ifmi spectral registration for video stabilization and motion detection by an unmanned aerial vehicle (uav). In *Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on*, pages 61–67, 2011. 2.3.2
- [16] Jianbo Shi and C. Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, pages 593 –600, jun 1994. 3.2, 3.2.1.1, 3.2.2
- [17] Richard Szeliski. Image alignment and stitching: a tutorial. *Found. Trends. Comput. Graph. Vis.*, 2(1):1–104, January 2006. (document), 3.2, 3.9
- [18] A. Telea. An image inpainting technique based on the fast marching method. *Journal of Graphics Tools*, 9(1):23–34, 2004. 3.3.3
- [19] J. Windau and L. Itti. Multilayer real-time video image stabilization. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 2397–2402, 2011. (document), 1.1, 1.2