Video enhancement becomes more and more important with the increasing prevalence of digital visual media. Videos made with a handheld video camera suffer from unexpected image motion caused by any unintentional shake of a human hand. Nowadays one of the most essential enhancements is the process for generating a new compensated sequence of video frames where undesirable image motion is removed, which is called video stabilisation. In this article, I will describe a video stabilisation method based on OpenCV library.

Video stabilisation on iOS

At first, the need for additional software-based video stabilisation on the iOS might seem questionable, as Apple offers its own standard stabilisation. But the problem is that the standard video stabilisation developed by Apple is available for a limited number of Apple devices and only starting with the iOS 6.0 platforms. Another vital issue is that not all source formats and video resolutions are supported.
As my experience shows, the use of OpenCV allows handling those issues and provides high quality video stabilisation.

Why use OpenCV?

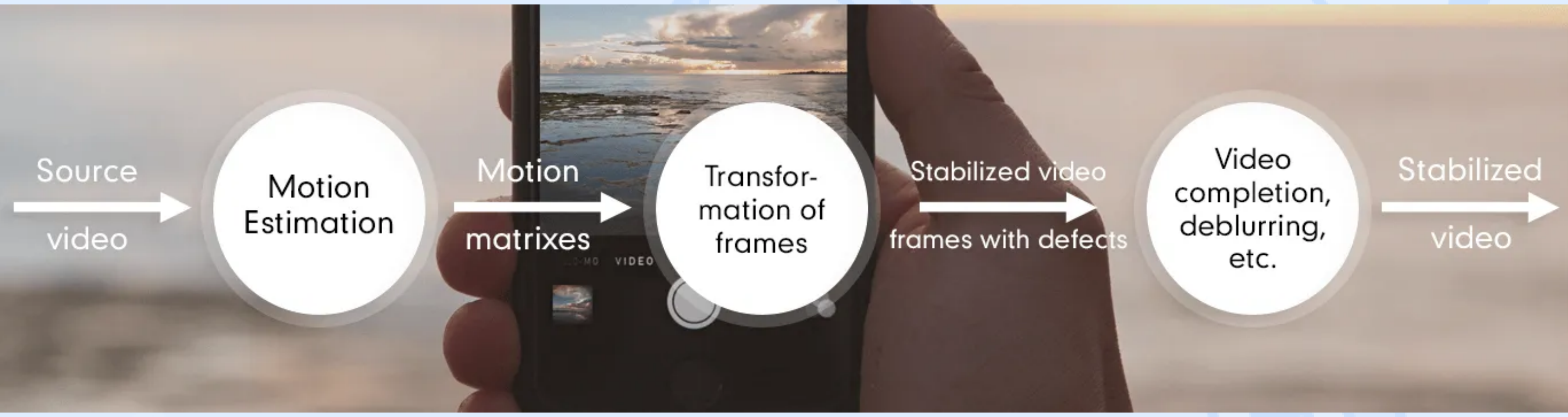
OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision, developed by Intel, and now supported by Willow Garage and Itseez. OpenCV is released under a BSD license and hence it's free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android.
OpenCV has a modular structure, which means that the package includes several shared or static libraries. The video stabilisation module contains a set of functions and classes for global motion estimation between point clouds or between images.

Algorithm

+44 20 7183 5820

info@magora.co.uk
sales@magora.co.uk

A video stabilisation algorithm usually includes the following steps:



Let’s describe the process in detail. Video stabilisation is achieved by first estimating the interframe motion of adjacent frames. At the end of this step we have the array of matrices 3×3, each of them describes the motion of the two adjacent frames. The accuracy of the global motion estimation is crucial as the first step of the stabilisation.
The next step is generating a new sequence of frames based on estimated motions, and doing additional processing (smoothing, deblurring, border extrapolation, etc.) to increase the quality of stabilisation.
After computing the global transform chain, the second step is removing the annoying irregular disruptions. There are approaches which assume a camera motion model, which works well when the assumed camera motion model is correct.
Filling in the missing image areas in a video is called video completion.
Motion blur, a fundamental image degradation process which is caused by moving scene points traverse several pixels during the exposure time, is another problem in video stabilisation. So, such methods provide a deblurring mechanism.

Here is the list of the basic parameters for OpenCV classes (motion estimators, stabilisers), which we need for video stabilisation:

1.
```
enum                                                                   MotionModel
{
                            TRANSLATION                                          =
```

2. Outlier and inlier ratio for RANSAC parameters in motion estimation (the default outlier ratio is 0.5, default inlier is 0.1)

3. Smoothing radius for stabiliser, which is used to reduce image noise (the default is 15)

4. Border extrapolation mode:

```
enum
{
BORDER_REPLICATE=IPL_BORDER_REPLICATE,  BORDER_CONSTANT=IPL_BORDER_CONSTANT,
```

5. Deblurring sensitivity from 0 to +inf (the default is 0.1)

Setting up

The OpenCV library comes as a so-called framework. First of all we should directly drag-and-drop the iOS version of this framework into our XCode project.

To add a camera controller to our view controller I used the following code:

```
using                                namespace                             cv;
@interface                            ViewController                       ()
{
```

When initialising, I used the parameters:

```
_videoCamera              =              [[CvVideoCamera              alloc]
initWithParentView:&amp;lt;imageView&amp;gt;];
_videoCamera.delegate                                =                  self;
```

In this case, I initialise the camera and provide the imageView as a target for rendering each frame. CvVideoCamera is basically a wrapper around AVFoundation, so I provide as properties some of the AVFoundation camera options. The property defaultFPS sets the FPS of the camera. If the processing is slower than the desired FPS, the frames are automatically dropped.

Also, I add framework dependencies of the OpenCV framework. Finally, you should have at least the following frameworks in your project:

- opencv2
- Accelerate
- AssetsLibrary
- AVFoundation
- CoreGraphics

- CoreImage
- CoreMedia
- CoreVideo
- QuartzCore
- UIKit
- Foundation

I initialised the stabiliser and the estimator this way:

```
StabilizerBase                                                    *stabilizer;

GaussianMotionFilter    *motionFilter    =    new    GaussianMotionFilter();
```


+44 20 7183 5820

info@magora.co.uk
sales@magora.co.uk

```
stabilizer->setRadius(15); // See point 3 from the previous paragraph
stabilizer->setBorderMode(BORDER_REPLICATE);
```

Processing frames

I follow the delegation pattern to provide access to each camera frame. View Controller has to implement the CvVideoCameraDelegate protocol and has to be set as delegate to the video camera. You should implement – (void)processImage:(Mat&)image delegate method and call:

```
[_videoCamera start];
```

To stabilise the sequence of frames we should do the following:

```
Mat stabilizedFrame;
while (!(stabilizedFrame = stabilizedFrames->nextFrame()).empty())
```

Don't forget to free up used memory.

Results

To evaluate the performance of the used methods, I have conducted a lot of experiments on short videos. The computational cost of our current implementation is about 2 frames per second on iPhone 4s for a video of 640×480 size, 30 FPS, using a two pass OpenCV stabiliser, AFFINE motion model, BORDER_REPLICATE mode and default values for other parameters (without GPU usage). These results are insufficient for real-time video stabilisation. The stabilisation speed decreases as the source video resolution increases.

The speed of video processing also depends on the type of scene (feature count, light). But the bottleneck of the process is motion estimation. For example, it is possible to increase the processing speed up to 40 FPS with the same settings using already estimated motions.

Therefore, it is important to speed up the local motion estimation part using GPU and OpenCL for real-time stabilisation.

Links:

http://opencv.org/

http://www.cc.gatech.edu/~irfan/p/2011-Grundmann-AVSWROCP.pdf

http://en.wikipedia.org/wiki/RANSAC

+44 20 7183 5820
info@magora.co.uk
sales@magora.co.uk

magora

magora

+44 20 7183 5820

info@magora.co.uk
sales@magora.co.uk