

Real-Time MMX-Accelerated Image Stabilization System

Shaokang Chen and Brian Lovell

Intelligent Real-Time Imaging and Sensing (IRIS) Group
The School of Computer Science and Electrical Engineering
The University of Queensland, Australia QLD 4072
{shaokang,lovell}@csee.uq.edu.au

July 25, 2001

Abstract

Recent advances in computer hardware and software have led to the possibility of implementing low-cost real-time computer vision systems on conventional PC hardware based upon the Intel processor supporting the Multimedia Extended instruction set (MMX). Here we describe an demonstration image stabilization system for real-time removal or minimization of unwanted scene motion due to rapid camera motion. Typical applications would be the removal of handshaking in hand held cameras or the removal of vibration effects from a camera mounted in a moving car or boat. Unlike many commercial stabilization systems, this system suppresses the effects of both severe translation and rotation of the camera.

1 Introduction

A serious impediment to developing real-time computer vision systems has been the computational power and level of skill required to process video streams in real-time. This has meant that many researchers have either analysed video streams off-line or used expensive dedicated hardware acceleration techniques. While valuable, off-line demonstrations tend to be unconvincing

as they are typically performed on just a few sequences. Hardware accelerated real-time systems suffer from high cost, lack of portability, and high maintenance due to the enormously rapid development in image processing hardware platforms. Recent software and hardware developments have greatly eased the development burden of real-time image analysis systems leading to the development of portable systems using cheap PC hardware and software exploiting the Multimedia Extension (MMX) instruction set of the Intel Pentium chip.

This paper describes the implementation of an Image Stabilization System (ISS) system using efficient C++ coding and MMX-acceleration to achieve real-time performance on low cost hardware. The ISS is composed of three stages: the edge detection stage, motion detection stage, and the motion compensation stage. It can process almost any kind of video data stream including live video from a camera, mpeg, avi, and network streamed formats such as ASF. It processes video at the rate of 15 fps on a 566Mhz Pentium computer.

2 Development Environment

2.1 Why use C++ for computer vision programming?

Many languages and development environments have been suggested for the programming of computer vision systems. Our group met much success in programming a hand gesture recognition system [?] and a face recognition systems [?] in MATLAB using the Vision for MATLAB ?? toolbox. This environment is excellent for prototyping systems and trying out ideas, but it is just not fast enough for producing real-time demonstration systems. Another problem is that Vision for MATLAB only supports video cameras, so this environment is unsuitable for processing prerecorded video from DVDs and archived television footage. One solution used very successfully in the MIME hand gesture system [?] was to completely recode the MATLAB prototype system into the C language to take advantage of the Microsoft Video for Windows API and the hardware acceleration techniques available in certain graphics cards. A major drawback of this approach was that the system became very dependent on a particular brand of graphics hardware and would not work with later versions without major recoding. Unlike processors where the hardware doubles in speed every 18 months, graphics is doubling every 6

months — so hardware specific implementations have a very short lifespan.

The current development environment for real-time demonstration systems uses the Microsoft Direct-X SDK to provide an transparent API for access to all video sources supported by the Windows 2000 operating system including cameras, MPEG-2, AVI, MOV, and network streamed formats such as MPEG-4 and ASF [7]. Rather than using custom graphics card hardware to provide image processing acceleration, we use the Intel Image Processing Library which accesses the very fast MMX instruction set of the standard Pentium (II and above) chip to provide video acceleration. The Intel Open Computer Vision Library (OpenCV) uses the same development environment to produce their example applications.

Since the Windows 2000 operating system and all of these libraries are coded in C++, it is natural to adopt this language as our development language. Fortunately, it is a good choice of language for other reasons as well. Experience shows that C++ programs are naturally small and fast, often better in this regard than C. The C++ language allows interaction with hardware and manual control of memory allocation which are also very suited to real-time implementations. Finally, C++ was designed from the outset to encourage code reuse and reduce coding errors through strong object typing — essential features of a language used to build large systems such as almost all popular modern operating systems. Unfortunately, many programmers code C++ like C and avoid these most elegant aspects of the language leading to bug-ridden unmaintainable code.

2.2 Software System Structure

The software system consists of three components: Microsoft Visual C++ code, Microsoft DirectX 8 and Intel IPL C++ libraries, see figure 1. Microsoft DirectX 8 offers support for two-dimensional (2-D) and three-dimensional (3-D) graphics, audio and other multimedia devices, and support for networked applications. It is very versatile since it can support many multimedia hardware devices such as video cards and cameras and many of multimedia formats such as AVI, MPEG, and Quicktime formats. The major advantage is that we can use the environment to develop applications that transparently support many different devices and multimedia formats without changing our code [7].

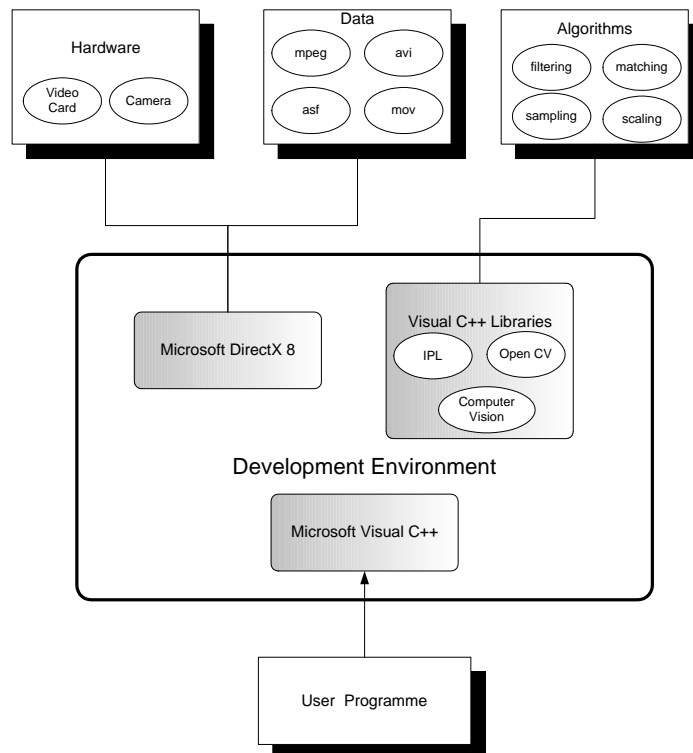


Figure 1: Software System Structure

3 The Image Stabilization System

This system is used to remove unwanted shaking and movement of images due to camera motion. It runs fast enough for real-time processing with a maximum of 16 frames per second on an Intel Celeron 566Mhz processor.

3.1 The Algorithm for Image Stabilization

1. *Model*

Suppose the image is captured by a camera which is carried by a moving carrier such as a vehicle or a walking human. The camera is likely to move as well as shake due to the vibration and hand trembling. To stabilize the image stream, we need to remove this unwanted shaking of the video stream. Therefore, we match successive frames in order to find out the movement of the camera from frame to frame. This movement is a mixture of vibration and the gross movement of the camera. It is helpful to separate the vibration from gross camera motion. Fortunately, the vibration usually has higher frequency spectrum than the overall motion of the camera. By low pass filtering the motion, we can remove this high frequency noise [1].

After removal of vibration noise, camera motion is considered to be a mixture of translation(parallel to the camera) and rotation, see figure 2. Thus every motion is represented by both a translation vector Υ and an rotation angle θ . If the rotation center is at x_0, y_0 , then the whole movement Ω

$$\Omega = \Upsilon + \nu + (y_0 \sin \theta - x_0 \cos \theta, -x_0 \sin \theta - y_0 \cos \theta)$$

where $\nu = (x \cos \theta - y \sin \theta, y \cos \theta + x \sin \theta)$, and (x,y) is the original coordinate of any point in the image.

Therefore, a rotation with rotation center (x_0, y_0) can be regarded as rotation at origin with the same angle plus a translation vector $\delta = (y_0 \sin \theta - x_0 \cos \theta, -x_0 \sin \theta - y_0 \cos \theta)$. In order to estimate the motion Υ and rotation angle θ of the image, we need to obtain sufficient motion information from different points in the two successive frames. Each point p_i has a motion vector $\omega_i = v_i + \nu_i + \delta$. If these points are

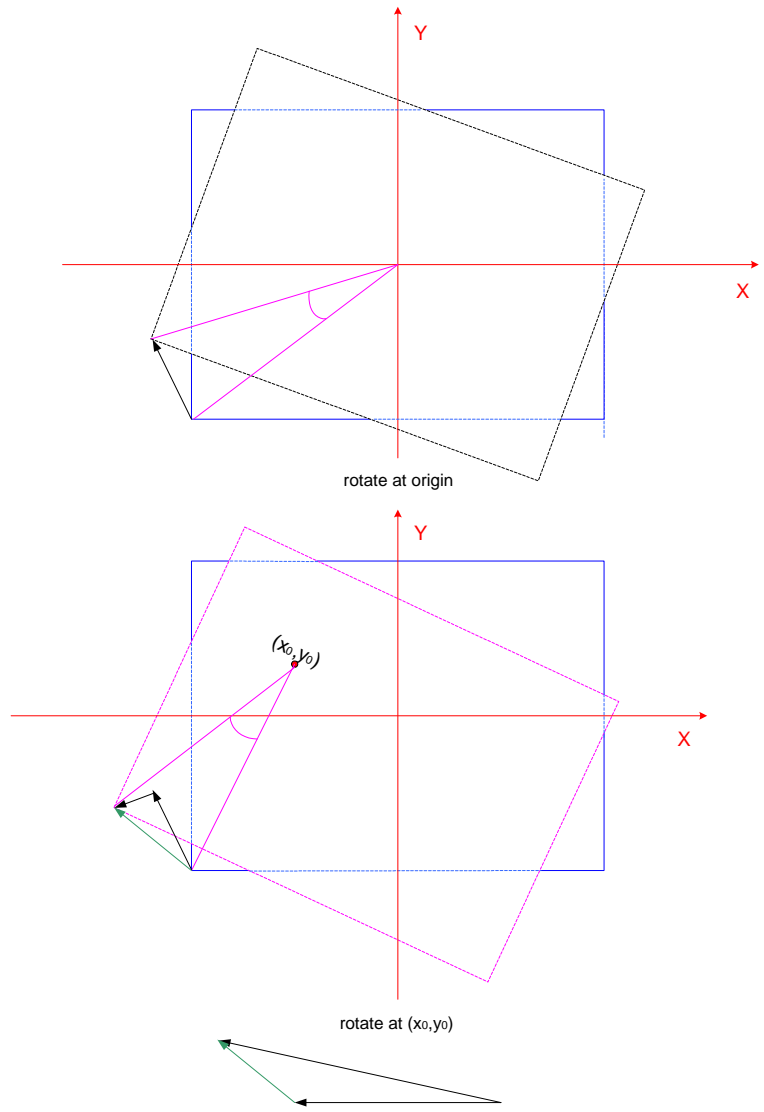


Figure 2: Rotation between adjacent two frames

symmetric, then

$$\begin{aligned} \sum_{i=1}^N \omega_i &= \sum_{i=1}^N v_i + \sum_{i=1}^N \nu_i + N * \delta \\ &= \sum_{i=1}^N v_i + 0 + N * \delta. \end{aligned} \tag{1}$$

So the average motion vector $\bar{\omega}$ is

$$\begin{aligned} \bar{\omega} &= \frac{1}{N} \left(\sum_{i=1}^N \omega_i \right) \\ &= \frac{1}{N} \sum_{i=1}^N v_i + \delta. \end{aligned} \tag{2}$$

From the above, we get the rotation angle θ from every vector ω_i .

2. *Implementation Issues*

The most popular method used to determine the motion between two frames is block matching. But, if we match all blocks in the image as in [1], it takes a very long time to process one frame and is not fast enough for real-time stabilisation. In order to make it fast enough for real-time processing, we use an edge detection stage, which is used to abstract features from the images to make the match more accurate in cases where simple block matching fails (regions of low contrast) [3] [4].

For the matching method, we found it was better to use minimum differential matching, because this matching is quite fast and is accurate enough, especially for regions containing edges. A most important feature is that this method is suitable for determining the rotation vector of blocks, whereas sign change matching is not as effective. Cross-correlation matching was found to be inaccurate translation detection — it sometimes caused image shaking for static images especially when illumination changed. Moreover cross-correlation was computationally expensive.

A final concern was to decide on the block size. The block size should be as small as possible to ensure efficient block matching, but if the block size is too small, it may not contain sufficient features and this

may cause the matching to be inaccurate. This effect is even more serious when the image is rotating — larger block can match the correct features between adjacent frames while small blocks can lead to mismatches. Finally, we settled on using minimum differential matching in the feature regions with a block size larger than 25×25 .

3.2 Image Stabilization System Structure

The whole system consist of three parts: the edge detection stage, the motion detection stage and the motion compensation stage, see Figure 3.

1. *Edge Detection Stage*

In this stage, the first step is to convert the colour image into grayscale. Then horizontal and vertical Sobel filters are used to detect object edges. The image is divided into 16 regions so that we can determine blocks with the most features (edges) for detecting motion. Thus we set a threshold such that if no block contain enough edges in a certain area, we just ignore that region. To optimize the matching for the next step, we try to find blocks that are symmetric about the origin. If there are no such blocks, then select those that contain more features.

2. *Motion Detection Unit*

The Motion detection stage is the key part of the system. We use block matching to detect the motion vectors of all valid blocks around reference points and store them in an array. We then remove those block vectors that are obviously massively different to others. We also set a threshold to remove motion and rotation estimates that are too large to be genuine. If the rotation is larger than, say, 5° between successive frames, then the matching may not be accurate enough to detect the rotation vectors.

3. *Motion Compensation Unit*

After extracting all the motion vectors, we need to filter out unwanted vibration as well as rotation from these vectors. We average the block motion vectors p_i to get the whole frame motion vector \bar{p} . This vector is the combination of the actual carrier motion and the shaking. Then we use a temporal low pass filter on \bar{p} , such as triangle filter or more

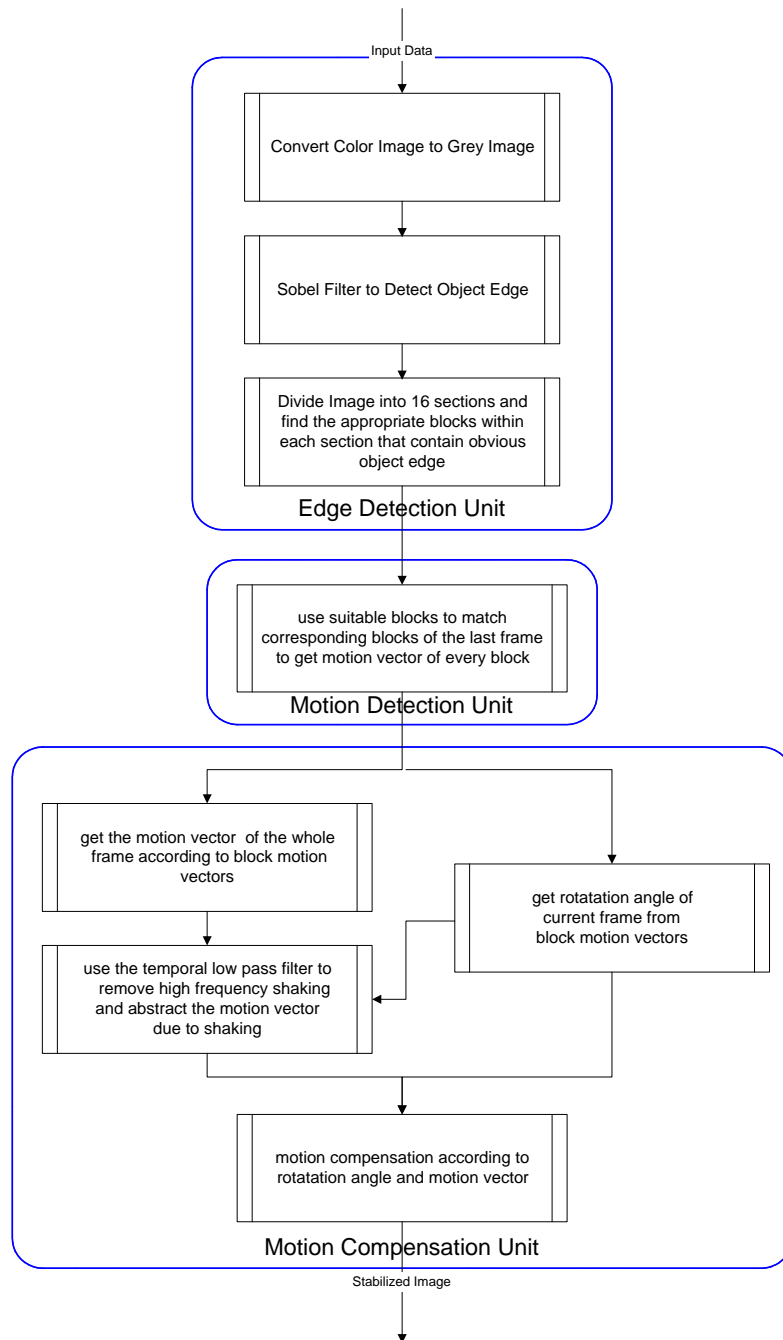


Figure 3: Process for Image Stabilization

sophisticated low pass filters as in [1], to remove high frequency noise to get the estimated motion c .

Then the vibration vector $s = \bar{p} - c$. If prediction is selected, the system will estimate the position of the next frame and set reference points for matching according to this prediction to speed up processing. For rotation angle, we obtain the rotation vector $r_i = p_i - \bar{p}$. The rotation angle will obey the following formula: $\sin \theta_i = \frac{r_i + \delta}{L_i}$, where L_i is the distance between the center of i^{th} block to the origin. To obtain δ , we just use several pairs of vectors from symmetric blocks,

$$\begin{aligned} \frac{r_m + \delta}{L_m} &= \frac{r_n + \delta}{L_n} \\ \delta &= \frac{r_m L_n - r_n L_m}{L_m - L_n} \end{aligned} \quad (3)$$

The average of angles θ is $\bar{\theta} = \frac{1}{N} \sum_{i=1}^N \theta_i$. We then integrate the vector s and rotation angle $\bar{\theta}$ for the compensation of the frames.

4 Experimental Results and Conclusion

We have implemented the system on Pentium celeron 566 computer with S3 Trio 3D/2X video card. Table I shows the results for 320×240 , 24bit video stream. The stabilized image streams are available at www.csee.uq.edu.au/~iris.

block size	search range	filter length	synchronization	pyramid	predict	speed	effect
$\frac{1}{8} \times \frac{1}{8}$ image (40×30)	7	7	Y	N	N	14.93	Very Good
	4	7	Y	Y	Y	15.84	Excellent
	7	7	N	N	N	16.29	Excellent
$\frac{1}{16} \times \frac{1}{16}$ image (20×15)	7	7	Y	N	N/Y	15.77	Bad
	4	7	Y	Y	Y	16.22	Good
	7	7	N	N	N	16.82	Good
$\frac{1}{32} \times \frac{1}{32}$ image (10×7)	7	7	Y	N	N	16.22	Bad
	7	7	N	N	N/Y	17.44	Poor
32×32	7	7	Y	N	N	14.93	Good
	7	7	N	N	N	15.78	Very Good
25×25	7	7	Y	N	N	15.39	Average
	7	7	N	N	N	16.05	Good
	4	7	Y	Y	Y	16.07	Very Good
	4	7	N	Y	N	17.07	Very Good
16×16	7	7	Y	N	N	16.17	Poor
	4	7	Y	Y	Y	16.77	Good
	7	7	N	N	N	16.93	Good

From the above table, we can see that:

- Changing of block size does not affect the processing speed greatly. From 10×7 to 40×30, the block size become 16 times larger, but the speed is only 1 fps slower. The major influence of changing block size is on the quality of the processing effect. The larger the block size is, the better the effect tends to be.
- The size of the search range has some influence on the processing speed. By change it from 7 to 4, the speed improves up to 1 fps. However, by change the search range, the quality of the compensation effect may be affected. For efficiency it is helpful necessary to use image pyramids, which we use a zoomed image for detecting motion. Our results show that by using pyramid, the quality of the compensation is not obviously affected, but reduce the search range by 50%. (what does this mean???)
- It is possible to process the image without clock synchronization. This means that no time clock is used to synchronise the output frames with the original recording rate. In this case our system will achieve better compensation results and higher frame rates. That is because decoding occupies 40% of the CPU time and the decoder runs faster

than the stabilization system resulting in occasional skipped frames to maintain time synchronisation. If there is no synchronisation, no frames are skipped so compensation is more accurate. However, if we use image pyramids and active prediction to improve the speed, we can obtain results equivalent to using no synchronization clock.

5 Conclusions

We have implemented a real-time image stabilization system using C++ coding and MMX-acceleration. By detecting motion in high contrast areas of the image stream, the system is capable to remove unwanted effects of camera vibration, rotation, and translation between successive frames at a real-time processing speed of 16fps.

References

- [1] Jesse S. Jin, Zhigang Zhu and Guangyou Xu *A Stable Vision System for Moving Vehicles*. IEEE Transactions on Intelligent Transportation Systems, Vol.1, NO.1, pp. 32-38, March 2000.
- [2] A.Engelsberg and G.Schmidt *A Comparative review of Digital Image Stabilising Algorithms for Mobile Video Communications*. IEEE Transactions on Consumer Electronics, Vol.45, NO.3, pp. 591-597, August 1999.
- [3] Joon Ki Paik, Yong Chul Park and Sung Wook Park *An Edge Detection approach to Digital Image Stabilization based on Tri-state adaptive linear neurons*. IEEE Transactions on Consumer Electronics, Vol.37, NO.3, pp. 521-524, August 1991.
- [4] Joon Ki Paik, Yong Chul Park and Dong Wook Kim *An Adaptive Motion Decision System for Digital Image Stabilizer Based on Edge Pattern Matching*. IEEE Transactions on Consumer Electronics, Vol.38, NO.3, pp. 607-615, August 1992.
- [5] *Image Processing Labrary Reference Manual*. developer.intel.com/software/products/perflib/ipl/index.htm
- [6] *Intel Open Source Computer Vision Library Reference Manual*. www.intel.com/research/mrl/research/opencv/
- [7] *Microsoft DirectX 8.0 Reference*. www.msdn.microsoft.com/library