MVA2005 IAPR Conference on Machine VIsion Applications, May 16-18, 2005 Tsukuba Science City, Japan

8-1

Real-time 2D Image Stabilization: Algorithmical Aspects and Parallel Implementation

Fábio DIAS, Jean Pierre DERUTIN, Lionel DAMEZ

LASMEA - LAboratoire des Sciences et Matériaux pour l'Electronique et d'Automatique
Université Blaise Pascal — Clermont-Ferrand — France
24 Avenue des Landais, AUBIERE Cedex – 63177
dias, derutin, damez @ univ-bpclermont.fr

Abstract

We present a real-time image stabilization method, based on a 2D motion model and different levels of parallel implementation. This stabilization method is decomposed into three main parts. First, the image matching is determined by a feature-based technique. Then the motion between consecutive frames is estimated and filtered to extract the unwanted motion component. This component is finally used to correct (warp) the images, resulting in a stable sequence. To validate our stabilization approach in a real-time on-board system context, the algorithm was implemented and tested over different hardware platforms, allowing a performance evaluation in function of the adopted architecture. In this paper, we present some of the results concerning the parallel implementation using the SIMD ALTIVEC® instructions set and a symmetric multi-processor architecture (SMP).

Keywords *– 2D image stabilization, parallel implementation, real-time application, SIMD instructions, SMP architecture.*

1 - Introduction

We are interested in the general study case of a camera rigidly mounted on a mobile system. This configuration is frequently found in tele-operation or aided-driving systems. The image sequence from this camera has informations about the movement of the vehicle in its environment. This movement may be divided in two components: one due to the driven motion of the vehicle, and a second component due to the parasite motion (unintended) suffered by the camera (bumpy ground, vibrations, etc.). Depending on its amplitude, this parasite component can strongly interfere in the visualization process and understanding of the image sequence, whether it is by a human observer/operator, or by an artificial vision system.

In these situations, stabilizing the image sequence consists in eliminating or smoothing of the unintended motion component, while leaving the driven motion component intact. This process is called "on-demand" or "selective" stabilization.

Although electronic image stabilization is widely explored, the architectural hardware approach allowing these systems to work in real-time, while respecting all the specificities of an on-board system, has received little attention. However, it's important to simulate the algorithmical approach in realistic experimental conditions, specially timing conditions. To achieve this task, this kind of software needs a special hardware architecture, with parallel processing abilities.

Our approach to deal with the "Algorithm-Architecture Adequation" is based on standard computational systems, also called "Commodity of the Shelf" (COTS). In the first stage of the work, we have verified the precision and robustness of our method in a sequential way. Then, in a second development stage, we did its real-time implementation using the parallel structures offered by the COTS systems.

In section 2, we describe briefly the different existing stabilization methods, and also the processing blocs which generally compose them. In the 3rd section, we present a more detailed description of our stabilization approach. Section 4 presents the various hardware structures we have adopted. Section 5 explains how the algorithm was programmed to be executed in a parallel way. Finally in section 6, we present some results related to the implementation on the different hardware platforms.

2- The Electronic Image Stabilization

In the last years, several electronic image stabilization methods were proposed. These methods may be classified into three main families, according to the adopted motion model: 2D or planar methods *[Mo97]*, 3D methods *[Dur03]* and 2,5D methods *[Zhu98]*.

In fact, stabilization algorithms are composed of a sequence of processing blocs, which have different levels of complexity. Generally, three main processing stages are completed: image matching, global motion processing using the adopted model, and filtering/compensation of the unwanted motion, getting as result a stable sequence.

2.1 - Image matching: We aim to calculate the movement in the 2D image plan of a real-world point or region. This movement is the 2D projection of the object's 3D motion in the observed scene. The most current ways to solve this problem are the optical flow extraction *[Dur03]* and feature-based approaches *[Mo97]*.

Even if the optical flow extraction method (explained in *[Horn81]* and *[Bar94]*) has already been employed for image stabilization purposes, it is constraining because of its mathematical complexity (that may be relatively high). The assumption that the optical flow fields are a 3D motion fields projection is another constraint *[Ve89]*.

In this study, to process image matching we use detection and tracking of visual features. This method consists in two steps: first, searching in the image *i* for regions with strong visual information (e.g. strong luminance contrast, corners, edges, etc…) called visual features, then identifying the same regions in the image *i+1*. Different tools for visual features detection are known, for instance, the "corner and edge detector" *[Har88]*, the Laplacian operator and Harr's wavelets (the latter is presented in the next section).

Once our features have been detected, we must be able to find them in another image. This task is done using a correlation method combined with a search strategy. Multi-resolution techniques allow a smaller processing time, through a "coarse-to-fine" approach. Several correlation methods may be employed, from the simplest ones being SSD & SAD *[Pou02]*, to light-changes robust methods like normalized cross correlation *[Tsai03]*.

2.2 - Global motion estimation: Once image matching has been achieved we can proceed to the second stage of the stabilization processing chain: the estimation of the motion parameters, which are determined by the adopted motion model.

2D models suppose a planar or almost planar scene. All points tracked in the preceding stage must lie in approximately the same distance from the camera. In this case, there are three parameters to estimate: two translations (horizontal and vertical) and one rotation around the camera optical axis. A fourth parameter may be included, to take into account scale changes caused by the camera forward/backward motion.

3D models suppose that only rotational parasites are relevant. So, we have to estimate and correct 3D rotations to stabilize our image sequence. Knowing that camera rotation effects in images are independent from scene depth, we are able to estimate camera rotation parameters, using quaternions for instance *[Mo97]*.

The 2.5 model presented in *[Zhu98]* presuppose the availability of preliminary information about camera motion leading us to estimate three global motion parameters, plus one independent parameter for each analysed point (tracked). This last one is a depth-related parameter. It allows us to work with structurally sophisticated scenes, without needing an advanced 3D model.

2.3 - Motion compensation: Finally, after estimating the global motion between images, we're going to compensate its unwanted or unintended component. This last processing stage is closely related to the application framework. The definition of "unwanted motion" depends entirely from the kind of "stability" required in each application. Several methods can achieve "full compensation" corresponding to static background scene, low-pass or inertial filtering *[Zhu98]* and low-order polynomial fitting for 3D rotations *[Dur03]*.

3 - Our Stabilization Method

General description: We have developed a stabilization method based on a 2D motion model, with visual features detection by Harr's wavelets, applied over a transformed image (integral image). The search of matching points is done using a multi-resolution pyramidal strategy, with three resolution levels. A SSD (Sum of Squared Differences) operator is applied to measure the similarity between the searched feature and its potential matching. Once we get matching points between two successive images, we can estimate the 2D motion model parameters (x, y and θ), using the Median Least Squares Method (MLSM). This technique is powerful as it limits the influence of incorrect matchings that could be found in the previous stage. Finally, the movement parameters are filtered and the obtained unwanted motion component is used to warp the respective image, stabilizing the video sequence (figure 2).

Detailed description: From each image acquired by the camera (coded in 256 grey levels, image size adjusted by the user) three intermediary images are produced: one integral image, that will be used for visual features detection, and two sub-sampled images (½ and ¼ pixels), used to construct the multi-resolution searching pyramid. The integral image has in its *(X, Y)* position the sum of all pixels inside the rectangle delimited by *i(0, 0)* and *i(X, Y)*, where *i(x, y)* is the original image. The calculation uses the formulae of recurrence given below, where *ii(x, y)* is the integral image and *s(x, y)* is an intermediate value (sum accumulated line by line on column *x*):

$$s(x, 0) = i(x, 0)$$
$$s(x, y) = s(x, y - 1) + i(x, y)$$
$$ii(0, y) = s(0, y)$$
$$ii(x, y) = ii(x - 1; y) + s(x, y)$$

Harr's wavelet processing consists in the convolution between an image region (pattern) and one wavelet mask (figure 1). The obtained value represents the luminance gradient in a given direction. Wavelet's processing is strongly accelerated when using an integral image. In this case, we can evaluate the sum of all pixels inside a rectangle of any size performing only 4 memory access and 3 sum operations *[Vio01]*. This property is also exploited for sub-sampled images creation. The mean value of pixels inside a square region (size 2*x*2 or 4*x*4) of the original image is obtained using the integral image.

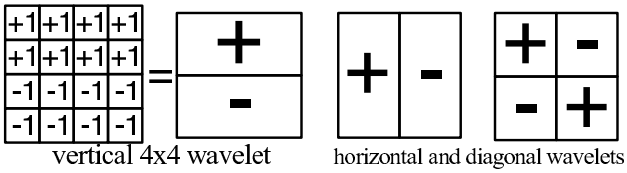

Figure 1. Three examples of wavelet masks.

Features are detected applying the wavelets over a pre-defined zone. We use the upper half of the image to search features present in the horizon line. Normally, these regions are far away from the camera, enough to respect the planarity constraint of the 2D motion model. The detection zone is divided in *n/3* vertical bands, *n* being the desired features number, set by the user. Three types of wavelets (vertical, horizontal and diagonal) are applied into each band, and the three regions presenting the

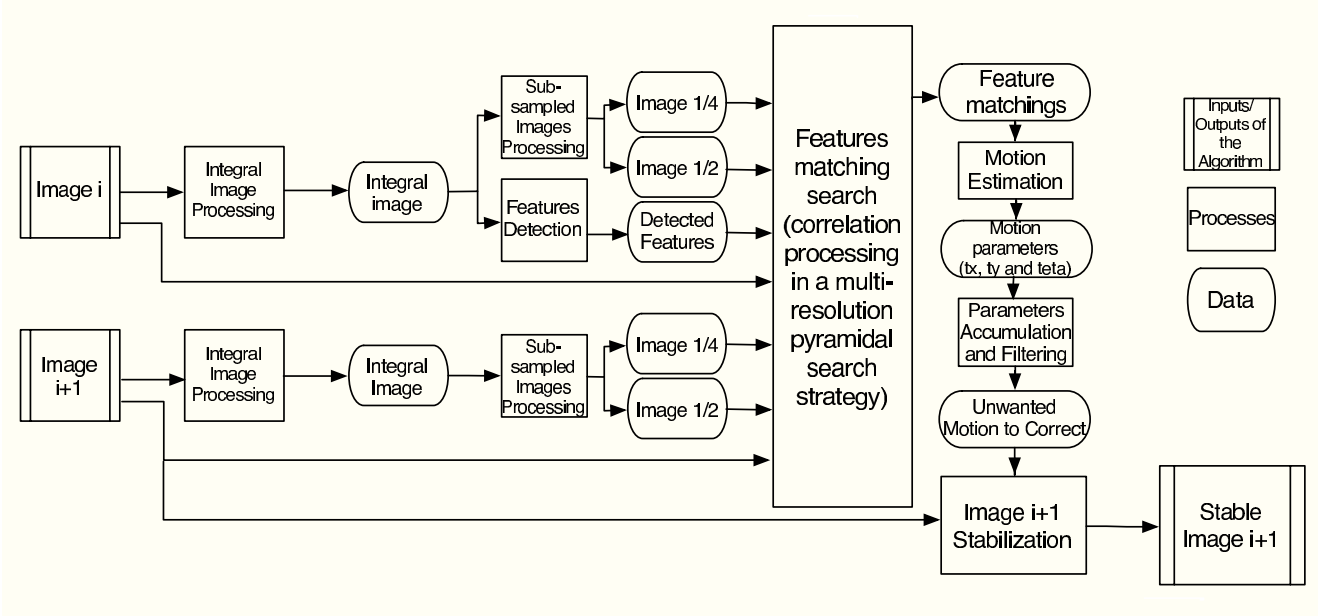

Figure 2. Synoptic scheme of the stabilization algorithm.

biggest values of vertical, horizontal and diagonal gradients respectively are selected as features (figure 3).

After the detection stage in image *i*, we seek the *n* corresponding features in image *i+1*. A "search window" with size *2Tx2T* is defined around the position where a feature was detected. The SSD is calculated between each region inside this window and the feature selected in image *i* (figure 3). The region in image *i+1* which minimizes the SSD is considered to be the match of the respective feature. This operation is repeated for each one of the *n* features detected in the preceding stage, giving us *n* matching points between two successive images.

T is the largest feature displacement, between two images, that can be measured by the system. This configurable parameter directly influences processing time, searching time being linked (non linearly) to the distance *T*. This obvious bond between the largest displacement and the processing time is extremely important. The goal is to maximize the multiplication of the largest displacement (in pixels) by the number of images processed per second. This indicates the greatest speed of an object (in pixels per second) that the system can deal with. This way, the parameter *T* must be carefully set, taking into account its influence on the processing time.

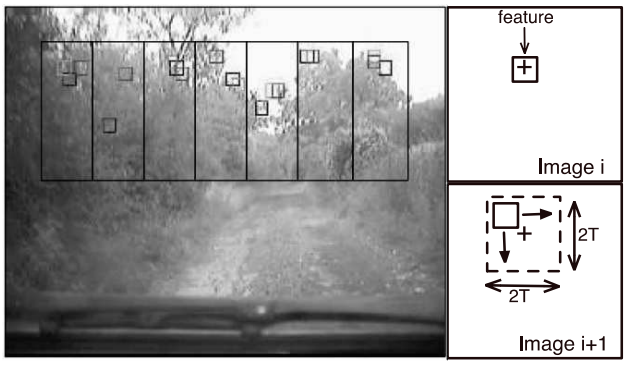

Figure 3. Features detection (left) and tracking (right).

To reduce the processing load, the search for matching features is executed in a multi-resolution approach. We start using a ¼ sub-sampled image, and looking for a SSD minimization inside a *T/2 x T/2* window. This provides us with a first estimation for the matching point position. Based on this estimation, a second search process begins, using a ½ sub-sampled image and a 3x3 search window, placed around the first estimated position. A second estimation is thus obtained, more accurate than the first one. Finally, the final search stage is executed, using the original image and a sub-pixel precision of 1/8 pixel. A 2x2 search window is analysed, and the value of regions lying between pixels is calculated through a bilinear interpolation of the adjacent pixels.

Having found the *n* points matching between images *i* and *i+1*, we can estimate the 2D motion model parameters describing the movement from one image to other. This movement can be modelled by a homogeneous transformation matrix, composed of a rotation around the optical axis, vertical and horizontal translations. Three matrix parameters, related to each of these movements, must be estimated. The *n* point matching result is applied to the model, and the error is minimized using the Median Least Squares Method.

The motion parameters obtained are added to those processed before, in order to find the total camera movement from the beginning of the video sequence. The found values are filtered by first-order linear filters. Each parameter has an independent filter, and the coefficients of all filters can be set by the user. This method allows us to have flexible stabilization intensity, adjustable to the application. We can also have different stabilization levels for translations and rotation.

The filtered values are used to get the inverse homogeneous transformation matrix that is applied to stabilize image *i+1*, bringing it back to a dynamic reference position (figure 4). This dynamic reference position tries to follow the commanded camera motion, respecting the passing band determined by the coefficients of the filter.

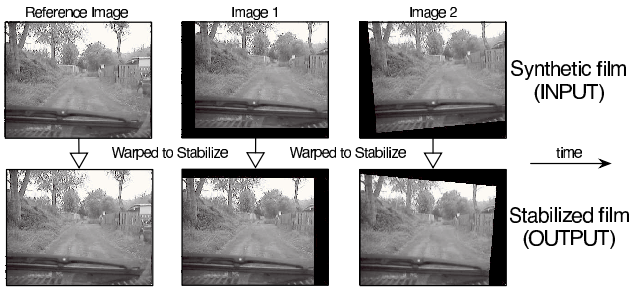

Figure 4. Stabilization of a synthetic sequence.

This stabilization method was tested with several real and synthetic incoming video sequences. The obtained precision for synthetic sequences (for which ground truth information is available) is very accurate. The motion parameters estimation between two successive images has mean error of 0.2 pixels for translations estimation, and 0.05 degrees for rotations.

The synthetic sequences were produced from a real road image (figure 3) and applying on it successive 2D rigid transformations in order to simulate the camera motion (figure 4). The camera motion model was extracted from a real sequence took in off-road conditions.

4 – Hardware Presentation

The processing chain descrided in section 3 was developped on a PC machine, featuring a AMD Athlon XP 1700 processor. After the evaluation and validation of algorithm's efficiency, and in order to enable a real-time processing, the application was transferred toward Apple PowerMac machines, with shared memory biprocessor symmetric architectures. These machines have two processors Motorola MPC7455 (PowerG4) or two IBM PowerPC 970 (PowerG5). The operation system is MacOS X. The selected systems have some features enabling parallel processing, in two different levels:

- Inside each processor, through super scalar processing devices, and with SIMD instructions set (Altivec).
- With two parallel processors working simultaneously, sharing the machine's memory (SMP architecture).

4.1 - SIMD instructions set: This type of extension is found in several microprocessor families: MMX, SSE and SSE2 for Intel processors, 3DNOW! for AMD, MDMX for MIPS and VIS for SPARC processors. All these extensions of the instructions set are based on two principles:

- first principle: it offers SIMD processing capacities, making possible to execute a logical or arithmetical operation on a multiple data set, with one instruction only.
- second principle: it gives a set of instructions strongly inspired by DSP systems: satured arithmetics, cabled and type conversion operateurs.

These instructions are applied on fixed size vectors (128 bits for Altivec), but the number of processed elements inside a vector can vary: four 32 bit, eight 16 bits or sixteen 8 bits elements.

The utilisation of SIMD instructions implies a fine grain parallelization, recommended for repetitive operations. In this case, if the "operations/memory access" ratio is high enough, we can have almost linear speed-up factors (4, 8 or 16) when processing integer type of data, or even over-linear speed-up factors for floating point type of data. The performances obtained with this type of parallelization are discussed in *[Fal04]*.

However, the SIMD parallelization is limited to the framework of repetitive regular operations. Another inconvenient is that it imposes a low abstraction level, and it's necessary to completely rewrite the software code for that tasks we hope to accelerate.

4.2 - SMP architecture: The second level of parallelization lies on the exploration of two processors communicating via a shared memory. The use of this type of architecture results in a big grain parallelization, sharing the tasks or the data between both processors.

The operation system (MacOS X) manages the executing tasks distribution in a "preemptive" way. It's also conceived to share the tasks between both processors, making the parallel processing transparent for the user. However, in order to execute a same task over two processors simultaneously, we must split this task in lighter processes called "threads". In our case, two threads are created and executed at the same time, enabling the system to distribute the work between the available processing resources.

The creation of these threads is possible using the standard functions library "pthread". This library defines some rules and tools for threads creation and fusion, including locking functions to manage tasks synchronization and mutual exclusion.

Otherwise parallelization through SIMD instructions, this type of parallelization allows a high abstraction level, enabling a fast implementation without entirely rewriting the software code.

5 – Parallel Implementation

To propose a parallelization scheme exploiting to the best the hardware features introduced in the previous section and presenting good timing performances, we adopted the following methodology: the different processing stages of the sequential version were carefully analysed, according to two different criteria: "operations/memory access" ratio and processing data volume.

Based on this analysis (table 1), we tried to concentrate the parallelization efforts on that stages where the speed gain may be potentially high. From this analysis, we can assume that *step_1*, *step_2* and *step_4* are the most processing consuming stages. The relative importance (in sequential processing time) of these stages in relation to the whole stabilization loop is shown in table 2.

It's noticeable that *step_4* is the most time consuming task. However, when image size is increased, intermediary images creation (*step_1* and *step_2*) becomes a time consuming task too. So, in order to obtain a noteworthy speed-up factor, it's interesting to execute these three tasks with a parallel approach.

Table 1. Decomposition and potential parallelism of each step of the algorithm.

Processing Stage	Intrinsic parallelism	Operations /memory access	Processing data volume
step_1: Integral Image Processing	data	low	high
step_2: Sub-sampled Images Processing	tasks and data	low	high
step_3: Features Detection	tasks and data	low	low
step_4: Features Matching	tasks and data	very high	very high
step_5: Motion Parameters Estimation	sequential	low	low
step_6: Motion Filtering and Correction	sequential	low	very low

Table 2. Relative importance of the *steps 1*, *2* and *4*.

Processing Stage	Image 320x240	Images 640x480	Images 1280x960
step_1 and step_2: Intermediary Images Processing	3%	12%	24%
step_4: Features Matching	93%	81%	65%

5.1 - Parallel implementation on a shared memory biprocessor symmetric architecture: As explained before, these machines have two potential types of parallelism: data parallelism through vectorial instructions (Altivec library, SIMD mode), and data or task parallelism through biprocessor architecture (SMP mode).

- The integral image creation (*step_1*) is processed in SMP mode. Image is divided in two equal horizontal bands, and the integral of each band is calculated by one processor. The image division causes a data dependence break. In consequence, an extra correction stage must be executed to achieve the complete integral image.

- Sub-sampled images are calculated from the integral image (*step_2*), and are completely independent one from the other. So, a SMP mode is employed, with each processor being responsible for one sub-sampled image creation.

- Features detection (*step_3*) is done in SMP mode, using the division in vertical bands, like explained in section 3. Each processor searchs the half of the desired features number (*n/2*), processing the half of the *n/3* vertical bands that were defined inside the detection zone.

- In the sequential analysis (tables 1 and 2) we noticed that the features matching stage (*step_4*) is the most time consuming task. So, it brings two parallelism levels in its parallel implementation. Each processor searches the matching for that features it own has previously detected. So, each processor is therefore responsible for tracking the half of the desired features number (*n/2*).

The second parallelism level is in the correlation computation. SSD results are obtained using Altivec SIMD instructions. We are able to process up to 16 pixels in only one operation and in this case, we have 16 times less operations to process.

The SIMD function for SSD computation exists in two different versions: a simpler one, working with integer type data, and a more complex second version, dealing with floating-point numbers and bilinear interpolations.

- After the tracking stage, matching points lists of both processors are merged, and the motion model parameters are estimated and then filtered (*step_5* and *step_6*). These two last stages, not presenting a relevant complexity, are processed in sequential mode.

Table 3. Characteristics of the used machines.

System	***[Arch_1]*** Athlon XP 1700+	***[Arch_2]*** PowerMac G4	***[Arch_3]*** PowerMac G5
OS	Windows XP	Mac OS 10.3	Mac OS 10.3
Compiler	gcc 3.2	gcc 3.3	gcc 3.3
µP number	1	2	2
µProcessor	Athlon XP	MPC7455	PowerPC970
Frequency	1,47 GHz	1 GHz	2 GHz
Memory size	512 Mo	512 Mo	1Go
Cache size L1 L2 L3	 128 Ko 256 Ko	 64 Ko 256 Ko 1 Mo	 64 Ko 512 Ko

6 – Results

The temporal performances of the stabilization algorithm were measured with 3 image sequences with sizes 320x240 (*bench_1*), 640x480 (*bench_2*) and 1280x960 (*bench_3*). The software was configured to search $n = 42$ visual primitives at a maximal distance T near to 5% of image size: $T = 15$ pixels for *bench_1*, 30 pixels for *bench_2* and 60 pixels for *bench_3*. Characteristics of the machines used for tests are shown in table 3. In tables 4 and 5 we present the execution time of the parallelized functions and of the whole stabilization loop (*step_1* to *step_6).*

Measures correspond to the delay between two system time function calls, averaged over 1000 iterations of the stabilization loop.

With 320x240 images (table 4), we obtain a very satisfying speed-up factor on *step_4* (around 12), due to the SIMD mode, making possible to stabilize images in less than 10ms, using only one processor.

SMP mode has a weak theoretical speed-up due to the number of processors (only 2). On *step_4*, speed-up is almost linear with approximately 1,9 for *Arch_2* and *Arch_3*. For *step_1* and *step_2*, it is higher on *Arch_3* (approximately 1,8) than on *Arch_2* where it varies between 1,4 and 1,7. The processing time of step_3 remains short in comparison with the whole stabilization loop.

Even if SMP mode does not compensate the increasing complexity of *step_1* and *step_2* with image size augmentation, the final speed-up is at least 3 for *Arch_2* and 5 for *Arch_3.*

Table 4. Temporal performances (in ms) for ***Arch_1***, ***Arch_2***, and ***Arch_3*** with *bench_1* parameters: image size 320x240, $n = 42$ primitives and $T = 15$ pixels.

Processing stage	*[Arch_1]* sequential	*[Arch_2]* sequential	*[Arch_2]* SIMD	*[Arch_2]* SIMD + SMP	*[Arch_3]* sequential	*[Arch_3]* SIMD	*[Arch_3]* SIMD + SMP
step_1 and step_2	1,4	2,1	2,1	1,5	0,9	0,9	0,5
step_3	0,3	0,3	0,3	0,2	0,2	0,2	0,1
step_4	33,2	56,7	4,7	2,5	31,1	2,6	1,4
Total (*step_1 to 6*)	37,1	61,7	9,7	7,9	33,3	4,9	3,8

Table 5. Temporal performances (in ms) for each implementation with ***Arch_2*** and ***Arch_3***. Left side shows results for *bench_2* parameters (image size 640x480, $n = 42$ primitives and $T = 30$ pixels) and right side shows results for *bench_3* parameters (image size 1280x960, $n = 42$ primitives and $T = 60$ pixels).

Processing stage	*[Arch_2]* sequential	*[Arch_2]* SIMD + SMP	*[Arch_3]* sequential	*[Arch_3]* SIMD + SMP	*[Arch_2]* sequential	*[Arch_2]* SIMD + SMP	*[Arch_3]* sequential	*[Arch_3]* SIMD + SMP
step_1 and step_2	9,7	5,8	3,8	2,1	40,3	25,1	15,8	8,8
step_3	0,9	0,7	0,4	0,3	1,8	1,0	0,7	0,5
step_4	66,8	4,0	37,8	2,0	108,2	8,0	83,2	2,8
Total (*step_1 to 6*)	82,7	17,1	44,0	6,9	166,4	51,6	105,2	20,4

7 – Conclusions and Future Work

This paper presents an efficient method for 2D image stabilization with a precision near the $1/5^{th}$ of pixel. Furthermore, the originality of features detection by Harr's wavelets using an integral image allows an important decreasing in the number of operations, which allied to the efforts of parallel processing gives very fast temporal performances.

Very high speed-up factors were obtained with parallel processing, particularly using the SIMD instructions set. Processing time was drastically reduced, making possible to use this stabilization method as initial (pre-processing) stage in a chain of artificial vision algorithms. We're able to deal with big format images (1280x960), in real-time, using only a commercial (COTS) system, instead of an expensive dedicated architecture.

The parallel implementation of this stabilization method was led as pre-study for algorithms dedicated to the old films restoration (working with very big size images). Further optimizations are possible. Processing regions of interest corresponding to detection bands in *step_1* and *step_2*, instead of processing whole images should reduce operations number and avoid the data dependency correction described in part 5.1.

The implementation of an optimized version of this same stabilization method on a MIMD-DM architecture has already been led, and the obtained results will be shown in our next publication [Der05]. We used a cluster of 14 biprocessor PowerG5 machines, interconnected thru a 1 Gbit Ethernet network, and communicating via message passing, using the MPI library.

This last implementation has been studied for a future implementation of this algorithm in an electronic chip, using a network of communicating homogeneous processors and SPMD approach.

8 – References

[Bar94] J. Barron, D. Fleet, S. Beauchemin and T. Burkitt. "Performance of optical flow techniques". *International Journal of Computer Vision*, 12(1) : 43-77, 1994.

[Der05] J. P. Derutin, F. Dias, L. Damez and N. Allezard. "SIMD, SMP and MIMD-DM parallel approaches for real-time 2D image stabilization". *International Workshop on Computer Architecture for Machine Perception CAMP'2005*, July 2005.

[Dur03] Z. Duric and A. Rosenfeld. "Shooting a smooth video with a shaky camera". *Machine Vision and Applications*, 13 : 303-313, 2003.

[Fal04] J. Falcou, J. Serot. "E.V.E., An object oriented SIMD Library". *International Conference on Computational Science ICCS'2004*, Part 3, pages 323-330, 2004.

[Har88] C. Harris and M. Stephens. "A combined corner and edge detector". *Proceeding of the 4th Alvey Vision Conference*, pages 147-151, 1988.

[Horn81] B. Horn and B. Schunck. "Determining optical flow". *Artificial Intelligence*, 17 : 185-204, 1981.

[Mo97] C. Morimoto. "Electronic Digital Stabilization: Design and Evaluation, with Applications". *PhD thesis*, 1997.

[Pou02] H. Pourreza, M. Rahmati and F. Behazin. "Weighted multiple bit-plane matching, a simple and efficient matching criterion for electronic digital image stabilizer application". *6th International Conference on Signal Processing*, 2 : 957-960, 2002.

[Tsai03] D. Tsai, C. Lin and J. Chen. "The evaluation of normalized cross correlations for defect detection". *Pattern Recognition Letters*, Vol. 24, pages 2525-2535, 2003.

[Ve89] A. Verri and T. Poggio. "Motion field and optical flow: Qualitative properties". *IEEE Trans. Pattern Analysis and Machine Intelligence"*, 11(5) : 490-498, 1989.

[Vio01] P. Viola and M. Jones. "Rapid object detection using a boosted cascade of simple features". *Proceedings IEEE Conference On Computer Vision and Pattern Recognitinion*, 2001.

[Zhu98] Z. Zhu, G. Xu, Y. Yang and J. Jin. "Camera stabilisation based on 2.5D motion estimation and inertial motion filtering". *International Conference on Intelligent Vehicles*, 1998.