Motion detection, labeling, data association and tracking, in real-time on RISC computer

Lionel Lacassagne\textsuperscript{1,2}, Maurice Milgram\textsuperscript{1}, Patrick Garda\textsuperscript{1}
\textsuperscript{1}Laboratoire des Instruments et Systèmes
Université Pierre et Marie Curie BC 252
4 place Jussieu, 75252 Paris - FRANCE
lionel | garda@lis.jussieu.fr maum@ccr.jussieu.fr
\textsuperscript{2}IMASYS
Le quadril 23 bis rue E. Nieuport
92150 Suresnes - FRANCE
lionel@imasys.fr

\textit{Abstract} - This article introduces some algorithms for motion detection and tracking. To be able to track moving objects, the analysis rate must be fast enough in order to keep the considered movements slow (and the features of the objects not changing too fast). We are presenting a Markov Random Field based motion detection algorithm running in real-time on Pentium II and DSP C6x. For the medium level, we have developed a fast labeling algorithm, a benchmark is provided. The high level is fast because the previous processing steps are running at a faster rate than the video rate.

I. MOTION DETECTION

The motion detection (MD) is the base of all systems for tracking objects and image compression. The MD algorithms are mainly belonging to two classes: optical flow and correlation. These methods have been widely explored, everyone of them has some advantages and drawbacks. The simplest method is an image difference, where a significant variation of the grey level is associated to a movement.

At the same time, Markov Random Field based algorithms have assert themselves in a lot of image processing areas for regularising ill posed problems (edge detection, region segmentation, restoration)\textsuperscript{[6]}. If their main advantage is the robustness, their main drawback is their CPU consuming due to the huge amount of calculations. This leads researchers to study a lot of solutions to speed their execution up, as Parallel Machine, or dedicated architecture \textsuperscript{[2],[1],[5]}. We describe the use of MRF for enhancing an image difference. After explaining the different kinds of optimizations, we show how to reach real time on RISC and DSP.

A. A MRF approach

The aim of the markovian process is to improve the quality of the image difference with a relaxation algorithm. Because of changes of illumination, this image is very noisy: a lot of pixels are labelled as in motion and the regions are not filled. The energy model has been introduced by the Lis-Grenoble lab \textsuperscript{[3],[4]}.

Let $I_t$, be the grey level image at the moment $t$, and $O_t$ the observation ($O_t = |I_t - I_{t-1}|$). Then thresholding $O_t$ initializes the estimated field of label $E_t$. The background/movement sites are labeled 0/1. The Iterated Conditional Modes (ICM fig1) is then applied to find the relaxed label field $E_{t-1}$ from $\bar{E}_t$, $\bar{E}_{t-1}$ and $E_{t-2}$. Relaxation being deterministic, the algorithm converges in few iterations (fixed to four), to the first local minimum, so the initialization is very important. The straightforward image difference may be enhanced by a maximum likelihood \textsuperscript{[7]}, with or without confidence (CPU expensive).
B. Energy function, clique.

The energy model \( u \) used is the sum of two energies: 

\( u \) model energy that is a regulation term which ensures a spatiotemporal homogeneity and which also expresses the relation with the neighboring sites, and \( u_o \) adequation energy which prevents the model to drift too far from the initialization.

\[
\begin{align*}
    u(o_s, e_s) &= u_m(e_s) + u_o(o_s, e_s) \\
    u_m(e_s) &= \sum_{c \in C} V_c(e_s, e_r) \\
    u_o(o_s, e_s) &= \frac{1}{2\sigma^2} [o_s - \Psi(e_s)]^2 \\
    \Psi(e_s) &= \begin{cases} 
        0 & \text{if background} \\
        \alpha > 0 & \text{if movement}
    \end{cases}
\end{align*}
\]

Clique associated to \( u_m \) energy is a spatiotemporal and second order clique (Fig 2). The spatial potential function \( V_s \) is:

\[
V_s(e_s, e_r) = V_c(e_s, e_r) + V_p(e_s^t, e_s^{t-1}) + V_f(e_s^t, e_s^{t+1})
\]

With \( V_p, V_f \) defined as:

\[
V_p(e_s^t, e_s^{t-1}) = \begin{cases} 
    -\beta_p & \text{if } e_s^t = e_s^{t-1} \\
    +\beta_p & \text{if } e_s^t \neq e_s^{t-1}
\end{cases}
\]

\[
V_f(e_s^t, e_s^{t+1}) = \begin{cases} 
    -\beta_f & \text{if } e_s^t = e_s^{t+1} \\
    +\beta_f & \text{if } e_s^t \neq e_s^{t+1}
\end{cases}
\]

With the Ising model (spin up/spin down : \( +1/-1 \)):

\[
u_m = -\beta_s \sum_{c \in C} x_s^t x_r^t - \beta_p x_s^t x_s^{t-1} - \beta_f x_s^t x_s^{t+1}
\]

with \( x_s^t \), the spin of the site \( s \), at \( t \).

In order to make the trace of \( I_{t-1} \) to disappear, and to fill the overlapping area due to the image difference, it is convenient to respect the inequality: \( \beta_p < \beta_s < \beta_f \).

Experimental tests demonstrate that the required parameters (\( \beta_p, \beta_s, \beta_f, \alpha \)) do not have to be tuned according to the image sequence. They are set to the values:

\[
\beta_p = 10, \beta_s = 20, \beta_f = 30, \alpha = 20
\]

C. Optimization

1. The optimizations techniques

   The most common used technique is the ‘loop-unrolling’ (LU). It consists in the duplication of the loop body, but also, when it is possible to make some computation in ‘parallel’. Even so, the C6x pipeline is not always full. The software pipelining (SP) can solve this problem by maximizing the use of the different units.

2. Algorithm characteristics

   Two consecutive masks overlap on two columns. So the computation of the spatial energy is then split-up in three columns, and only one new column is computed instead of the three previously required. The entire computation is performed by Look-Up
Tables (LUT) to avoid CPU intensive multiplications.

The background/movement sites are labeled 0/1 rather than -1/1. Instead of comparing the state of each site to the central site $e_s$, we just take into account sites which state are 1 ($p_{s_1}, s_1$ and $f_1$ for the past, present and future images). So if the central site state is 1 the spatial energy is:

$$u_{m1} = (8 - 2s_1)\beta_s + (1 - 2p_1)\beta_p + (1 - 2f_1)\beta_f, \text{else}$$

$$u_{m0} = -u_{m1}$$

If $u_{m1} + u_{a1} < u_{m0} + u_{a0}$, the state is set to 1 otherwise it is set to 0. The change is performed whatever the previous state is. The test can be simplified by factoring the model and adequation terms: $2u_{m1} < u_{a0} - u_{a1} = \frac{\beta_s}{2\beta} (2\beta_s - \alpha)$. This a priori minor modification can save 1 cycle on a total of 6 on DSP C6x, that is to say 17%!

3. Implemented optimizations

The algorithm has been implemented on a Pentium II 400 MHz, and has been estimated (from the number of cycles of the loop) for the TI-C6202 at 250 MHz. On the Pentium the number of available registers is insufficient to perform the Loop-Unrolling. So this is the 8-accesses-per-point version, but with a LUT, that has been implemented. On the C6x there are enough registers. A Loop-Unrolling of order 3 is performed. Because the model energy is computed faster with multiplications (4 cycles) than with LUT (5 cycles) we use multiplications for $u_m$ and LUT for $u_a$. The figure 5 gives the scheduling on a C6x (the indexes correspond to the delay with the current index $t$).

![Figure 5 – C6x scheduling](image)

Per point, and on a C6202, the estimated time, depending on the performed optimizations (none, Loop-Unrolling, or Software Pipelining) are:

<table>
<thead>
<tr>
<th>optimization techniques</th>
<th>none</th>
<th>LU</th>
<th>SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>instructions</td>
<td>22</td>
<td>66</td>
<td>66</td>
</tr>
<tr>
<td>cycles</td>
<td>15</td>
<td>25</td>
<td>15</td>
</tr>
<tr>
<td>ipc</td>
<td>1.47</td>
<td>2.64</td>
<td>4.40</td>
</tr>
<tr>
<td>% charge</td>
<td>15</td>
<td>26</td>
<td>44</td>
</tr>
<tr>
<td>MIPS</td>
<td>367</td>
<td>660</td>
<td>1100</td>
</tr>
</tbody>
</table>

A comparison of the different architectures implementing the ICM relaxation is given in the figure 6. Standard RISC processors, programmed in C, achieve real time execution for image size larger than $256^2$ (up to $320^2$), and fixed point DSP, in assembly language for image size larger than $512^2$ (up to $700^2$).

II. LIGHT SPEED LABELING: A FAST AND NEW ALGORITHM

This paragraph introduces a fast algorithm for labelling binary images. Its speed is almost independant from data. Moreover its structure makes the 8-connected version as fast as the 4-connected ver-

---

3
Figure 6 – different implementations

The proposed algorithm is optimum with the number of created labels, and it can be easily and efficiently parallelized.

A. Classic labeling, RLC-based labeling

![Diagram of Classic labeling, RLC-based labeling](image)

Figure 7 – 4 and 8-connected neighbors labeling

Standard algorithms use either the pixel approach (Fig 7: comparison of the current label, with the neighboring pixels) or the Run Length Coding (search for an adjacent segment in order to find out the connectivity). Both of them involve a great amount of tests, and, at the CPU level this leads to many pipeline stalls (waiste of time) because the test result is, of course, unpredictable. Lastly and it is not its least drawback, the pixel labeling produces many useless labels.

B. LSL algorithm

The main idea in LSL is to replace all the tests (label comparisons, fusion sort for segment intersection) by a set of simple instructions without (or with much less) tests. These instructions are fast because pipelined.

The speed of LSL is based on 3 key points:

- intensive use of RLC
- introduction of “relative” and “absolute” labels for a direct access (LUT) to the label of the intersected segments
- a cyclic implementation of equivalences instead of a “flat” implementation.

Here are the different steps of the algorithm. The explanations will be illustrated by the example of the figure 8.

$$\begin{align*}
|j & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
L_{st} & & & & & & \boxed{1} & & & & \\
L_{t} & & & & & & \boxed{1} & & & &
\end{align*}$$

Figure 8 – Lines $L_i, L_{i-1}$

1. Computation of $ER_i$ and $RLC_i$

$$\begin{align*}
|j & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
ER_{st} & & & & & & 1 & 1 & 1 & 2 & 3 \\
ER_{t} & & & & & & 0 & 0 & 1 & 1 & 1
\end{align*}$$

Figure 9 – table $ER$

The $ER_i$ table holds, for each line $L_i$, the associated relative label $e_r$ of each segment (object with an odd number) and each non-segment (background with an even number).

$$\begin{align*}
\text{RLC}_{st} & 0 & 2 & 4 & 5 & 8 & 8 \\
\text{RLC}_{t} & 2 & 9
\end{align*}$$

Figure 10 – table $RLC$

The $RLC_i$ table holds the “begin” and “end” indexes of each segment $[a, b]$. The $j$-th segment of relative label $e_r = 2j + 1$ has the boundaries $a = RLC_i[2j]$, $b = RLC_i[2j + 1]$. The current segment has an intersection with the segments numbered $e_{r0} = ER_{i-1}[a]$, $e_{r1} = ER_{i-1}[b]$. The labels $e_{r0} / e_{r1}$ are eventually adjusted to point to the next / previous segment and not to the background:

To perform a 8-connected labeling, which is equivalent to test the diagonal labels (NW, NE), it is enough to increment $a$ by 1 and to decrement $b$ by 1, just before looking up the $ER_{i-1}$ table. That makes
the 8-connected version as fast as the 4-connected version.

2. Computation of \( \text{ERA}_i \)

![Diagram of absolute and relative labels](image)

Figure 11 – absolute and relative labels

The \( \text{ERA}_i \) associates the relative and absolute label of a line. The table is calculated during the top-down labels propagation. The relative label is associated with the absolute label of the first intersected segment:

\[
\begin{align*}
  e_r &= 1 \\
  a &= \text{RLC}_i[0] = 2 \Rightarrow a = 1 \\
  b &= \text{RLC}_i[1] = 9 \Rightarrow b = 9 \\
  e_r &= 6 \Rightarrow e_r = 5 \\
  e_a &= \text{ERA}_{i-1}(e_r) = 1, \text{ERA}_i(e_r) = e_a
\end{align*}
\]

3. Computation of \( \text{EQ} \)

To solve equivalence problems (computing the transitive closing of a graph is a complex problem) like adding an item to a class, or more generally to fusion two classes, the equivalence classes have been implemented with cyclic graphs in the EQ table.

![Diagram of equivalence cycle](image)

Figure 12 – equivalence cycle

During the fusion, \( e_a \) must be associated with the other absolute labels \( e_{ak} \) of the intersected labels.

\[
\begin{align*}
  e_{rk} &\in [e_{r0} + 2..e_{r1}], e_{ak} = \text{ERA}_{i-1}[e_{rk}] \\
  \text{fusion}(e_a, e_{ak})
\end{align*}
\]

with the example of the previous section:

\[
\begin{align*}
  e_{rk} &\in [e_{r0} + 2..e_{r1}] \\
  e_{rk} &\in [3, 5], e_{ak} \in \{2, 3\}
\end{align*}
\]

for the absolute labels, we have the equivalences.

The equivalence of the 3 labels will be processed like in figure 12.

4. Computation of \( \text{EA}_i \)

The absolutes labels of \( \text{Li} \) are obtained by LUT:

\[
\text{EA}_i = \text{ERA}_i(\text{ER}_i)
\]

5. Packing labels and relabeling

At the end, labels are packed (to get continuous number) from \( \text{EQ} \) to \( A \). The whole image is then updated: \( \text{EA}_i = A(\text{EA}_i) \)

C. Benchmark of LSL

The algorithm has been compare to classic pixel based algorithm, eventually enhanced by cyclic equivalence. All versions use 16-bit labels. LSL is also implemented in 8-bit because sometime the number of labels is inferior to 256. We tested the algorihms on 4 cases of labeling : small/big regions few/many characters. We give the results in ms for 4-connected and 8-connected versions. The image size is 512 \( \times \) 512.

![Image with benchmarks](image)

<table>
<thead>
<tr>
<th></th>
<th>bolts</th>
<th>logo</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixel</td>
<td>32</td>
<td>8C</td>
</tr>
<tr>
<td>pixel+cycle</td>
<td>20</td>
<td>4C</td>
</tr>
<tr>
<td>LSL 16bit</td>
<td>21</td>
<td>26</td>
</tr>
<tr>
<td>LSL 8bit</td>
<td>12</td>
<td>20</td>
</tr>
<tr>
<td>LSL vs pixel gain</td>
<td>2.7</td>
<td>5.3</td>
</tr>
<tr>
<td>caps 72</td>
<td></td>
<td></td>
</tr>
<tr>
<td>caps 28</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>bolts</th>
<th>logo</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixel</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>pixel+cycle</td>
<td>24</td>
<td>20</td>
</tr>
<tr>
<td>LSL 16bit</td>
<td>20</td>
<td>23</td>
</tr>
<tr>
<td>LSL 8bit</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>LSL vs pixel gain</td>
<td>1.7</td>
<td>2.5</td>
</tr>
</tbody>
</table>

Our algorithm outperform by a factor from 2.4 to 5.3 pixel-labeling for the type of image used in real motion detection. Even for OCR, LSL is a bit faster.
III. Region matching

The matching system remains simple. It can not track neither two crossing regions, nor manage an object splitted into many regions. But it is fast and it will be the base to compare to clever systems using artificial intelligence [8].

For two consecutive images \((I_{t-1}, I_t)\), the intersection area is computed (in grey on the Fig 1). For each region these surfaces are sorted into descending numerical order. Associating labels is then equivalent to search the best matching all over the tree. In this tree a node corresponds to a specific matching between two regions of two consecutive images, and a branch corresponds to a set of matchings. The chosen sub-tree is this which maximizes the sum of the intersected surfaces. Thanks to the sort, the best sub-tree is located on the left of the tree, even it is the leftiest sub-tree of the tree.

The following images are extracted from the famous “taxi” sequence. Figures \(I_3\) and \(I_4\) are the original images, figures \(\hat{E}_3\), \(\hat{E}_4\) are the absolute difference \(O_t\) binarized with a threshold value of 5, and figures \(E_3\), \(E_4\) are the ICM relaxed images.

IV. Conclusion

In this article, we have shown how to get a real time and a robust motion detection based on MRF on a RISC and DSP. We have proposed an new labeling algorithm, faster than the classical ones. Then, because these algorithms are fast and because we get good results, it remains enough time to perform a region matching and a tracking. Because all of these algorithms have been optimized, the whole processing is running in real time, at the video rate for \(256 \times 256\) images.

REFERENCES


