Fast Reliable Level-Lines Segments Extraction

N. Suvonvorn, S. Bouchafa, L. Lacassagne
Institut d'Electronique Fondamentale, Université Paris-Sud
91405 Orsay FRANCE
nikom.suvonvorn@ief.u-psud.fr, samia.bouchafa@ief.u-psud.fr, lionel.lacassagne@ief.u-psud.fr

Abstract

We propose a fast level-line extraction algorithm that could be used as a basic and generic feature extraction algorithm for several applications like motion analysis, robot navigation, image registration or stereovision. We give details of the implementation, the complexity and the execution time.

1. Introduction

In many applications (stereovision, motion, analysis or registration), we need to match features extracted from images. The nature of the features will play a significant role for the choice of the matching strategy. Less reliable features lead to very complex matching processes that have to compensate for their sensitivity toward several perturbations like contrast changes. Indeed, images to match are seldom shot in similar lighting conditions. However, commonly used primitives remain sensitive to contrast changes. Either they depend on gray-level values (points, regions), or they suffer a lack of real contrast-independent extraction methods (edges, contours). We choose *Level lines* as a basic feature for our motion analysis [1] and registration algorithms [2]. Their invariance property toward contrast changes has been proved by several authors [3][4][5] and verified experimentally using outdoor complex sequences [6][7]. We propose here a very fast level-line extraction process that could be used as a basic and generic feature extraction algorithm. Our paper is organized as follow: the first part concern to the reliability of line-segments by introducing the notion of *level-lines flow*, the second part, we propose a fast algorithm for level lines extraction, the third part, we discuss about the precision of approximated line-segments, and the last part, we show resulting speed-up and discussion.

2. Reliable line-segments

Let $I(\mathbf{p})$ be the image intensity at pixel location $\mathrm{p}(x,y)$. The *level set* of I is the set of pixels $\mathbf{p}$ with intensity equal or greater than l.

$$N_l^I = \{\mathbf{p}/I(\mathbf{p}) \geq l\} \tag{1}$$

The boundary of a level set N_l is called *level line* L_l. One could extract level lines by using a series of thresholds (such as λ={0,1,2...255} for an 8-bit image). We propose a method that exploits an elementary level-line property: level sets are included one in another. Thus, level lines could be locally juxtaposed but they never could cross. Therefore, we propose a simple recursive extraction process that tracks groups of level lines (the superimposed ones) until they separate. A group is called *level-lines flow [LLF]*.

Definition 1 *Let* $S_{\mathrm{p}(i,j),\mathrm{p}(x,y)}$ *be all the level sets between any two neighbors pixels* $\mathrm{p}(i,j)$ *and* $\mathrm{p}(x,y)$ *where* $|i-x|+|j-y|=1$. *The boundaries associated with all these level sets are called level-lines flow* $F_{\mathrm{p}(i,j),\mathrm{p}(x,y)}$

$$S_{\mathrm{p}(i,j),\mathrm{p}(x,y)} = \{N_l^I \,/\, l \in [l_l, l_u]\} \tag{2}$$
$$F_{\mathrm{p}(i,j),\mathrm{p}(x,y)} = \{L_l^I \,/\, l \in [l_l, l_u]\} \tag{3}$$

Where $l_l = \min(I_{\mathrm{p}(i,j)}, I_{\mathrm{p}(x,y)}) + 1$
And $l_u = \max(I_{\mathrm{p}(i,j)}, I_{\mathrm{p}(x,y)})$

Definition 2 *Let* $Card(F_{\mathrm{p}(i,j),\mathrm{p}(x,y)})$ *be the number of level lines in level-lines flow between pixels* $\mathrm{p}(i,j)$ *and* $\mathrm{p}(x,y)$, *called level-lines flow value.*

The *flow* allows treating level-lines by groups. Therefore, the extraction process is less time-consuming.

Definition 3 *A Flow Junction is located in every point where two different level-lines flows meet.*

3. Fast Level-Line Flow Segments Tracking

In this section, we show how level-lines flows could be tracked through any junction in the image and approximated by line segments. The tracking process is performed with 5 steps:

Step 1 consists of computing level-lines flows value on every pixel in the image and storing it in a memory M_f.

Step 2 consists of scanning each pixel in order to track its associated flow. The process begins on each point, calculates the current flow F_c and then tracks this flow in four different directions (4-connexity), determines which of its four neighbors is the successor. This tracking process is repeated until the following conditions are no more verified.

Condition 1 *The successor flow value in the memory* M_f *is greater than zero*.

Condition 2 *The successor flow* F_s *includes the current flow* F_c *(the current flow is a subset of the successor flow).*

$$F_c \subseteq F_s \tag{4}$$

The figure 1 (a) shows an example of flow junction where $F_1=\{L_\lambda / \lambda \in [11, 20]\}$, $F_2=\{L_\lambda / \lambda \in [6, 10]\}$ and $F_3=\{L_\lambda / \lambda \in [6, 20]\}$. We found that $F_1 \subset F_3$ and $F_1 \not\subset F_2$. Thus, if the current flow is F_1, the successor flow is F_3, not F_2.

In order to obtain all level-lines flows which pass through the current pixel. The process has to start in two different directions: left turning tracking (λ_u is on the left side of flow) and right turning tracking (λ_u is on the right side of the flow). The current pixel is then at the middle of level-lines flow.

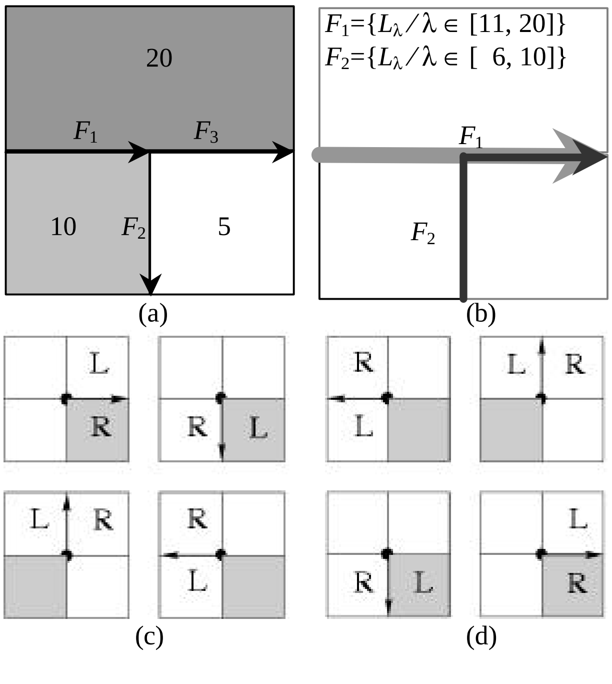

20
F_1
F_3
10
F_2
5
(a)
$F_1=\{L_\lambda / \lambda \in [11, 20]\}$
$F_2=\{L_\lambda / \lambda \in [6, 10]\}$
F_1
F_2
(b)
L
R
(c)
(d)

Figure 1. (a) Flow junction (b) Left turning tracking direction $I_L<I_R$ (c) Right turning tracking direction $I_L>I_R$.

Step 3 When the tracking process stops, in order to avoid to track a flow once again for another point, level-lines flow value in M_f on each analyzed pixel is updated by subtracting the current flow value.

Step 4 At this point, we get a list of pixels that corresponds to flow path. In this step, we will approximate this list by a line-segment. At the beginning of approximation process, we assume that this list of pixels could be approximated by only one line and then the distance from each pixel to this line will be calculated by using equation 5. If the maximum distance of these distances is greater than a given threshold, the line segment will be cut into two lines at the pixel that have the maximum distance. This process will repeat until the maximum distance is less than this threshold.

$$D = \frac{|Ax+By+C|}{\sqrt{A^2+B^2}} \tag{5}$$

Note that the result of this method depends on a threshold. We will explain in the next section how to choose automatically this value.

Step 5 consists of calculating the reliability R_l of level-lines segments.

$$R_l = L \times Card(F_c) \tag{7}$$

Where L is the length of level-lines flow.

4. Line-segment precision

In the previous section, we show that line segments could be approximated from a list of pixels by considering that the distances between each pixel in the list and line segment are less than a given threshold. In this section, we propose an automatic method to find the optimum threshold D_{opt}. In order to determine this threshold, we define the following terms:

Definition 4 *Let* $P(m)_{p(x_0,y_0)}$ *be set of pixels located at the distance m, far from pixel* $p(x_0,y_0)$ *, where* $p(x_0,y_0)$ *is a pixel in image space.*

$$P(m)_{p(x_0,y_0)} = \{p(x_m,y_m) / |x_m| + |y_m| = m\} \tag{8}$$

Definition 5 *Let* $G(m)_{p(x_0,y_0)}$ *be set of pixels associated to* $P(m)_{p(x_0,y_0)}$.

$$G(m)_{p(x_0,y_0)} = \{p(x_s,y_s) / x_s = nx_m, y_s = ny_m\} \tag{9}$$

Where $G(m)_{p(x_0,y_0)} \subset P(m)_{p(x_0,y_0)}$

And $n = 1 \ldots \infty$

Definition 6 *Let* $L(m)_{p(x_0,y_0)}$ *be set of lines in different degrees around* $p(x_0,y_0)$*, which is represented by* $G(m)_{p(x_0,y_0)}$.

$$L(m)_{p(x_0,y_0)} = \{x_m y - y_m x = 0\}$$
(10)

Definition 7 *Let* $q(m)_{p(x_0,y_0)}$ *be set of possible angles between any lines in* $L(m)_{p(x_0,y_0)}$.

$$q(m)_{p(x_0,y_0)} = \{|\tan^{-1}(\frac{j}{i}) - \tan^{-1}(\frac{j-n}{i-n})|\} \qquad (11)$$

Where $i = (m-1)n, (m-2)m, ..., n$

And $j = n, 2n, ..., (m-1)n$

The figure 2 (a) shows $G(m)$ and $L(m)$ in image at pixel $p(x_0, y_0)$, where m=3.

There are two properties according to these definitions:

Property 1 *For each value* m*, an image could be discretized with a finite number of angles.*

Property 2 *The more the value of* m *is large; the more the value of* $q(m)$ *is small.*

From these definitions, we found that a pixel $p(i, j)$ in $G(m)$ could be a part of three possible lines. The first is the line that passes though itself, the second is the nearest line on the left and the last is the nearest line on the right.

Definition 8 *If* $p(i, j)$ *is a pixel in* $G(m)$, $p(i, j)$ *will be a part of a line that the distance between them is minimum.*

However, we will not consider the first line because the minimum distance between them is always zero. Therefore, $p(i, j)$ could be a part of only the line located on its left side or on its right side. For $p(i, j)$ in every direction, we search the maximum distance. Thus, we can define the following relations.

$$L_{p(i,j)} = \{l_1, l_2\}$$
(12)

Where $l_1 \mapsto (i+n)y - (j-n)x = 0$

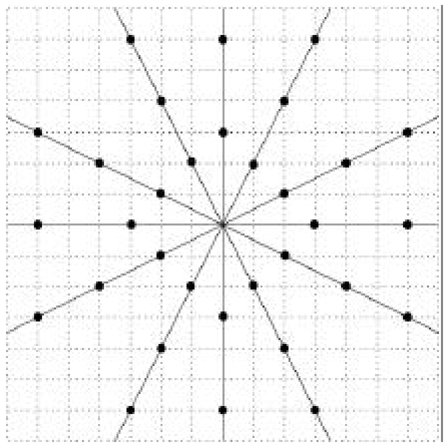
And $l_2 \mapsto (i-n)y - (i+n)x = 0$

$$D_{p(i,j)} = \max_{i,j}\{\min_{D}\{D_1, D_2\}\} \qquad (13)$$

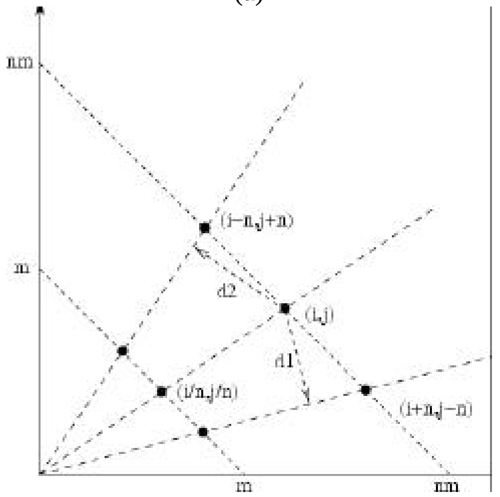
Where $D_1 = \dfrac{|n(i+j)|}{\sqrt{(i+n)^2 + (j-n)^2}}$

And $D_2 = \dfrac{|n(i+j)|}{\sqrt{(i-n)^2 + (j+n)^2}}$

With equation 13, we found that $D_{p(i,j)}$ is maximum at the pixel $p(mn/2, mn/2)$ (if m is even number) and at the pixel $p((m-1)n/2, (m+1)n/2)$) (if m is odd number). If we assume that we can represent lines at pixel $p(x_0, y_0)$ in very directions, it means that q(m) is near to zero (q(m)→0). Therefore, m is near to the infinity (m→∞).



(a)



(b)

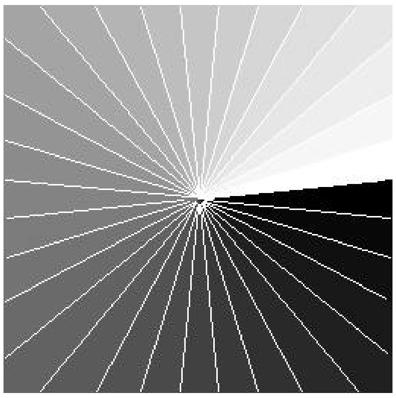
Figure 2. (a) shows $G(m)$ and $L(m)$ in image where $m = 3$ (b) the minimum distance between pixel $p(i, j)$ and two beside lines.

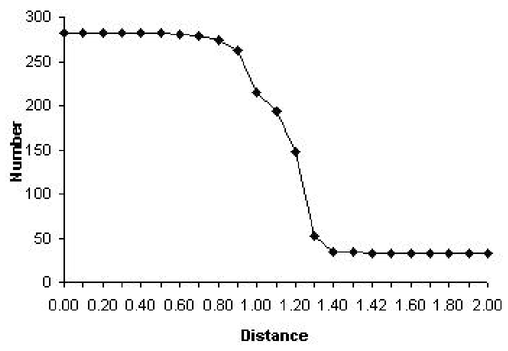
$$\lim_{m\to\infty}\left\{D\left(p(\frac{m}{2}n, \frac{m}{2}n)\right)\right\} = \sqrt{2}$$
(14)

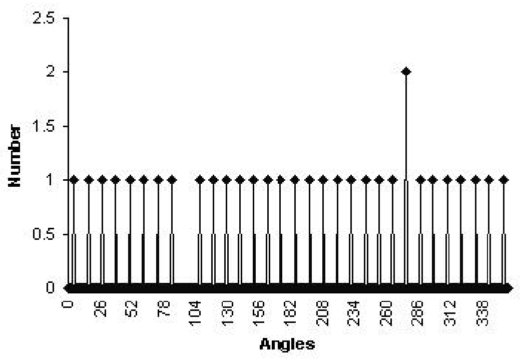
$$\lim_{m\to\infty}\left\{D\left(p(\frac{m-1}{2}n, \frac{m+1}{2}n)\right)\right\} = \sqrt{2}$$
(15)

From equations 14 and 15, we found that pixel $p(i, j)$ could be a part of line that the maximum distance between them is $\sqrt{2}$, which could be considered as the optimum threshold D_{opt}.

The figure 3 (a) is an image (with tracking result) that represents 32 lines in different directions around a point. The angle between two lines is 11.25 degrees. The figure 3 (b) shows histogram of lines according to the thresholds between zero and two. It show that if the threshold value is greater than 1.42, we can track 32 lines, which is the optimum number. The figure 3 (c) show histogram of lines according to the angles between 0 and 360 degree at threshold value 1.42, we found that each line can represent very well its direction. Like we can see that at the same degree 276, we have two lines. The figure 3 (d) show histogram of lines at threshold value 1.62, we found that there is only one line at 276 degree and the other line at 275 degree. It means that we have error of lines angle at this threshold. According to a series of experimental, we found that the optimum threshold is 1.42.







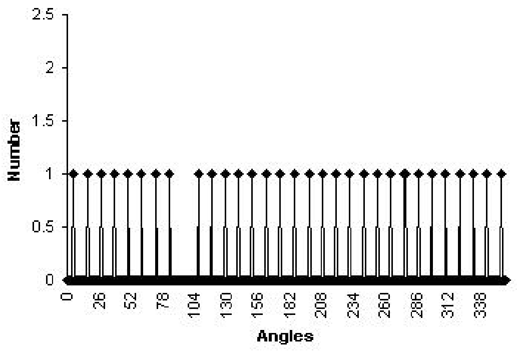


Figure 3. (a) Result of line extraction (b) Total number of extracted lines according to threshold distance value (c) histogram of angles using a distance threshold equal to 1.42 (d) histogram of angles using a distance threshold equal to 1.6

5. Multi-Resolution Line-Segments

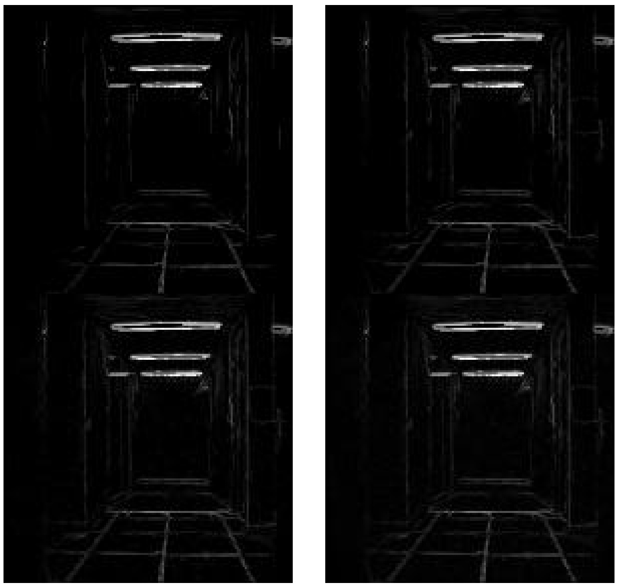


Figure 5. Shows

6. Benchmark

The theoretical complexity of such king of algorithms, very image-dependent, can’t be expressed easily. So, we decided to perform a benchmark, in order to estimate the impact, the speed-up, of the Flow optimization. We have tested different kind of images: outdoor natural image (parking), indoor classical images, (corridor & desk), synthesis image and high-textured image (leaves).

- Outdoor natural image (parking), for future application,
- Indoor classical images, (corridor & desk) for autonomous robot navigation,
- Synthesis image, for execution time low bound estimation,
- High-textured image (leaves) for high bound estimation.

The evaluation has been done on a Pentium 4, 2.4 Ghz, we used Intel 8.0 compiler, 75 % faster than Microsoft compiler. We prevent polluted measures from OS by running the benchmark in a loop and removing polluted results. We provide the execution time in *ms* and also the Cycle Per Points figure (*cpp* which is a frequency and image size independent figure) for both classical algorithm and the new algorithm

Name	Size	Org(ms)	New(ms)	Speed -up
Parking	640x480	1812	210	8.6
Corridor	256x256	953	62	15.4
Desk	256x256	796	60	13.3
Rays	256x256	3735	9	415
Leaves	256x256	5969	190	31.2

Table 1. Execution time (ms)

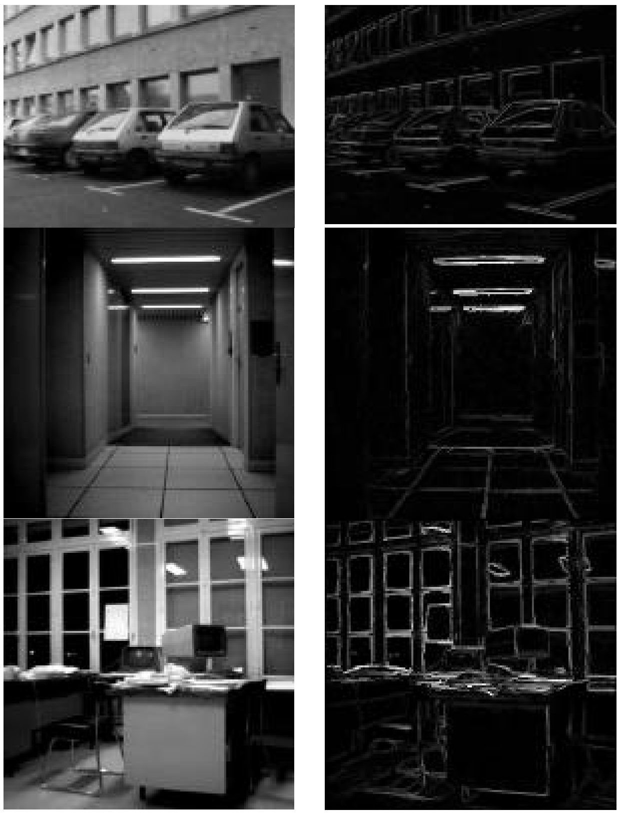
Name	Size	Org (cpp)	New (cpp)
Parking	640x480	14156	1641
Corridor	256x256	34900	2271
Desk	256x256	29150	2197
Rays	256x256	136780	330
Leaves	256x256	218591	6958

Table 2. Execution time (cpp)

We can notice, that the speed-up is very important: from 8 up to 15 for natural images, and 30 for high textured image. Such kind of results proves the LLF algorithm modification.

From a qualitative point of view, the *cpp* figures are very similar for natural image. This is also an important result for any embedded algorithm: the capability to predict, or to get a reasonable average execution time. More tests should be done to get a fine prediction, but this first result is interesting.

From a quantitative point of view, the gap to real-time execution is not so far. We expect to get a real-time implementation with a State of the Art or, at least, the next generation of RISC processor. High-end embedded processors like Texas Instrument DSP C64 should also provide enough CPU power.



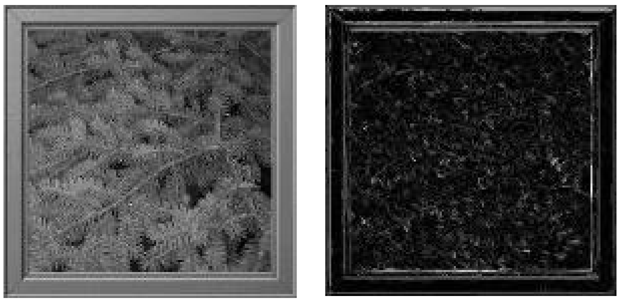


Figure 4. Shows the results of algorithm.

7. Conclusion

In this paper, we have introduced new algorithm for level lines extraction, based on flow computation called *level-lines flow*, which robust to contrast changes. We propose a recursive process for tracking the level-lines flow through flow junctions in image. The extracted level-lines flows are precisely approximated by sets of line-segments that could be used as a reliable basic feature for many applications such as motion analysis or stereovision.

The benchmark using representative images for usual applications show a speed-up of 8 up to 15. Such kind of results represents an advance for a real-time implementation.

8. References

[1] S. Bouchafa, *Motion detection invariant to contrast changes. Application to detecting abnormal motion in subway corridors*, PhD thesis., University Paris 6, 1998.

[2] S. Bouchafa and B. Zavidovique, “Cumulative level-line matching for image registration,” *ICIAP*, p. 176, September 2003.

[3] V. Caselles, B. Coll, and J. Morel, “Topographic maps and local contrast changes in natural images,” *International Journal of Computer Vision*, vol. 33, no. 1, pp. 5–27, September 1999.

[4] P. Monasse and F. Guichard, “Fast computation of a contrast-invariant image representation,” *IEEE Trans. on Image Proc.*, vol. 9, no. 5, pp. 860–872, 1998.

[5] C. Ballester, E. Castan, M. Gonzalez, and J. Morel, “Contrast invariant image intersection,” *C.M.L.A*, no. 9817, 1998.

[6] J.L.Lisani, P. Monasse, and L.I. Rudin, “Fast shape extraction and applications,” *C.M.L.A,*, no. 2001-16, 2001.

[7] J. Froment, *Image compression through level lines and wavelet packets*, Wolters Kluwer Acad. Pub., 2001.