Proceedings of the 2003 IEEE
International Conference on Robotics & Automation
Taipei, Taiwan, September 14-19, 2003

Real-Time Object Tracking using Multi-Res. Critical Points Filters

Jérôme Durand and Seth Hutchinson *(jdurand@uiuc.edu, seth@uiuc.edu)*

University of Illinois at Urbana-Champaign

Abstract

*In this paper, we will propose a new method for object tracking, which is primarily based on the results from prof. Shinagawa's image matching. We will provide a method that tracks an object and follows it in real-time through a sequence of images which are given, for example, by a robotic camera. The main feature of the method is that it is not affected by the movements (within a certain reasonable range) of the camera or the object; such as, translation, rotation or scaling. The algorithm is also insensible to regular changes of the object's shape. For real-time applications, the algorithm allows the tracking of an object through a sequence of 64*64 images, at a rate of over 8 frames/second.*

1 Previous work and related studies

Object tracking has been one of the main fields of study in Computer Vision for the last 20 years. The applications for real-time object tracking are numerous (medicine, security, industry), as the robot is increasingly helping the human in his regular tasks.

1.1 segmentation-based algorithms

Some of the previous work in this field implies a segmentation-based method: all the basic 'blob' algorithms, based on intensity, color (best known [1]), motion, texture... The main limitations for these algorithms is that the target has to remains roughly constant in size, and cannot move out of the region of interest. Furthermore, these algorithms can be computationally time consuming.

1.2 template-based region tracking

The idea of this algorithm is to match the direct appearance from frame to frame [2]. However, this algorithm assumes that the appearance of the object remains roughly the same, and it cannot handle rapid changes in position, lightning etc.

1.3 Snake algorithm

This is the best known algorithm. The main idea is to minimize an energy, and the algorithm is usually based on splines [3]. The limitations of this algorithm towards what we are going to propose is that it assumes it is possible to get good local contour information so that least squares can work. Despite its relatively good accuracy, the algorithm remains slow.

1.4 Other algorithms

Some other algorithms are based on probabilistic methods[8]. The most recent methods [5] are based on linear dynamic models (using Kalman filter [9]) or non-linear dynamic models. Here, the method is much faster and does not require much information about the object, the scene or the movement to achieve the tracking.

2 The Multi-resolution Critical Points Filters

They where first introduced by Yoshihisa Shinagawa in 1998 [4][6]. The goal of this paper is, of course, not to describe them in detail, but we will present the main ideas. First of all, we have to consider 2 images, the source and the destination. The aim of the algorithm is to match points of the source image (in our case, the points of the contour of the object) to points of the destination image. Let's suppose the size of the image is $N = M = 2^n$. If the size of the image is not a power of 2, we have to shrink it to be so.

2.1 Computing the Multi-resolution hierarchy

We will compute a multi-resolution hierarchy of size $2^l * 2^l$ $(1 \leq l \leq n)$ images. At each level of the hierarchy, 4 images are calculated. Let's call $p_{(i,j)}^{(l,m)}$, the point (i,j) of the image number m, from the source image, at level number l. The images are recursively computed as follows:

$$p_{(i,j)}^{(l,0)} = min(min(p_{(2i,2j)}^{(l+1,0)}, p_{(2i,2j+1)}^{(l+1,0)}),$$
$$min(p_{(2i+1,2j)}^{(l+1,0)}, p_{(2i+1,2j+1)}^{(l+1,0)}))$$
$$p_{(i,j)}^{(l,1)} = max(min(p_{(2i,2j)}^{(l+1,1)}, p_{(2i,2j+1)}^{(l+1,1)}),$$
$$min(p_{(2i+1,2j)}^{(l+1,1)}, p_{(2i+1,2j+1)}^{(l+1,1)}))$$

$$p_{(i,j)}^{(l,2)} = min(max(p_{(2i,2j)}^{(l+1,2)}, p_{(2i,2j+1)}^{(l+1,2)}),$$
$$max(p_{(2i+1,2j)}^{(l+1,2)}, p_{(2i+1,2j+1)}^{(l+1,2)}))$$
$$p_{(i,j)}^{(l,3)} = max(max(p_{(2i,2j)}^{(l+1,3)}, p_{(2i,2j+1)}^{(l+1,3)}),$$
$$max(p_{(2i+1,2j)}^{(l+1,3)}, p_{(2i+1,2j+1)}^{(l+1,3)}))$$

where

$$p_{(i,j)}^{(n,0)} = p_{(i,j)}^{(n,1)} = p_{(i,j)}^{(n,2)} = p_{(i,j)}^{(n,3)} = p(i,j)$$

which is the point of the original image.
The reason why they are called critical point filters is because they 'extract' the minimum, maximum and saddle points at each level of the hierarchy. The computing of the multi-resolution hierarchy is achieved on both the source and destination images. We'll call $q_{(i,j)}^{(l,m)}$ the point (i,j), of the image number m, from the destination image, at level number l.You can see an example of the images computed, on figure 1.

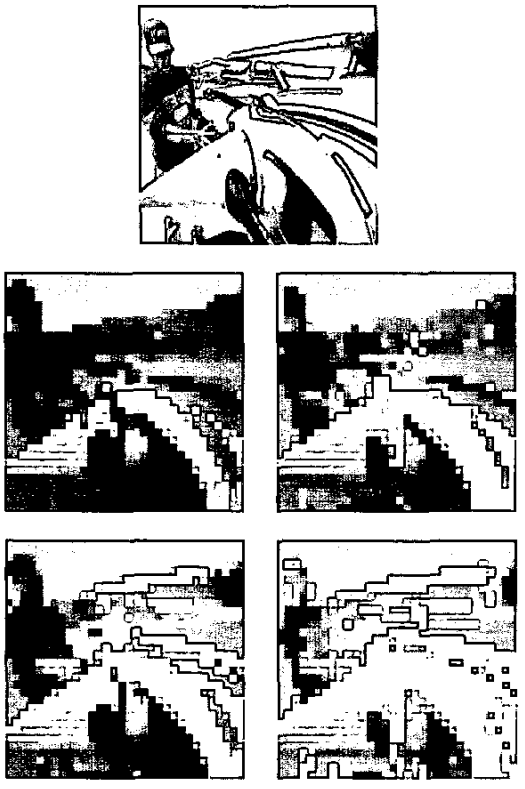

Figure 1: *The original image and the four images computed at level 5 of the hierarchy (original image is 256*256).*

2.2 Determining the mapping from one point to another

Once the hierarchy is created, we must determine the matching between the points: this is done recursively. Let's take an example (figure 2): suppose we want to map a point p (at level m), from the source image, to a point q, of the destination image. First of all, we determine the 4 nearest neighbors of point p, which we call a, b, c, d. In our algorithm, we take the four 'diagonal' nearest points, but we could have taken the four other neighbours without any significant change in the result. For each of them, we map the parents (A, B, C, D) recursively to A', B', C', D'. If one of them is located out of the image (this may be possible if point p belongs to the border), it is then mapped to the nearest border. For each of the parents, we take one child (if b is the top left child of B, then b' will be the top left child of B'). The four children define what we will call the *inherited quadrilateral*. The point p will be mapped to one of the points inside this quadrilateral. We will then choose q so that it minimizes a certain distance (between p and q), that we are going to describe below.

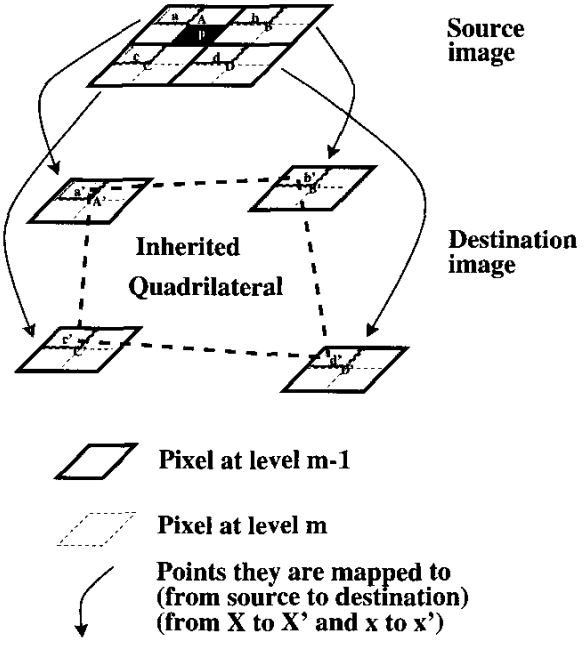

Figure 2: *The definition of the inherited quadrilateral.*

3 The predicted parameters

3.1 The center of the object

Using the previous positions of the center of the object (defined as the average of the position of the points), and a polynomial interpolation, we can predict the next position of the center of the object. By using a center-based definition of the object, we allow translation in the viewing plane and rotation around the viewing axis. This prediction can also count for the camera, as the movement of the camera and the relative movement of the object form only one single movement, from the viewer point of view.

3.2 The surface of the object

With a polynomial interpolation on the previous size of the object (typically the last 5), we can make an estimation of the value of the surface of the object. This value is used to normalize the value of the distance between the center and a given point, through different images. By doing so, we allow the object to move forward or backward on the viewing axis. We will also be able to track a deformable object, as long as the color doesn't change.

3.3 The different movements we can track

As we have seen, the algorithm might be able to track the object during translation, rotation and scaling of the object or of the camera (as long as the object remains in the scene!). However, if the object rotates on an other axis than the viewing axis, we may lose it because its shape (from the viewer point of view) will change, and may be the color will change too (think of a cube with a different color on every face).

4 The elements that compose the distance

The distance defined above takes several elements into account. Each of these elements is affected by a coefficient that will determine its relative weight towards the others. The coefficient is a function of the level of the hierarchy.

4.1 The intensity of the pixel

This is the most important part of the distance; it is based on the multi-resolution hierarchy described before. If we call $p_{(i,j)}$ the point to map and $t_{(i',j')}$ the point to test with inside the inherited quadrilateral, then the 'intensity' distance at level l of the hierarchy between the 2 points is

$$\sum_{0 \leq l \leq 3} |p_{(i,j)}^{(l,m)} - t_{(i,j)}^{(l,m)}|^2$$

where m is the number of the image in the multi-resolution hierarchy. As the level (in the multi-resolution hierarchy) of the pixel increases, the 'intensity coefficient' is decreasing, because as we 'get closer' to the original image (level n of hierarchy), intensity only counts for adjustments.

4.2 The position of the pixels

Introducing the center $c(c_x, c_y)$ of the source image ,$c'(c'_x, c'_y)$ the predicted position of the center in the destination image, S the surface of the object in the source image, S' the predicted surface of the object in the destination image, and with the same notations as before, the distance is

$$|\frac{(|i - c_x|^2 + |j - c_y|^2)}{S} - \frac{(|i' - c'_x|^2 + |j' - c'_y|^2)}{S'}|$$

Which is the difference between the normalized (by the surface of the object) square distance between a point and the center (or estimated center) of the object.

4.3 The value related to the edge

In order for the mapping to be more accurate, we use 2 edge detection filters (Sobel filters) for level n of the hierarchy: these two filters create horizontal and vertical edge detection images ($edgeh^{source}$ and $edgev^{source}$ for the source image, and $edgeh^{dest}$ and $edgev^{dest}$ for the destination image). You can see one example of this in figure 3. For the other levels of the hierarchy, we only average the value of the pixels from the filter (e.g. $edgeh_{(i,j)}^{(m,source)} = \frac{1}{4}(edgeh_{(2i,2j)}^{(m+1,source)} + edgeh_{(2i,2j+1)}^{(m+1,source)} + edgeh_{(2i+1,2j)}^{(m+1,source)} + edgeh_{(2i+1,2j+1)}^{(m+1,source)})$ with the same notations as before). Then, the 'edge' distance between the 2 points at level m of the hierarchy is:

$$|edgeh_{(i,j)}^{(m,source)} - edgeh_{(i',j')}^{(m,dest)}|^2 +$$

$$|edgev_{(i,j)}^{(m,source)} - edgev_{(i',j')}^{(m,dest)}|^2$$

The importance of this distance is independent of the level of the hierarchy.

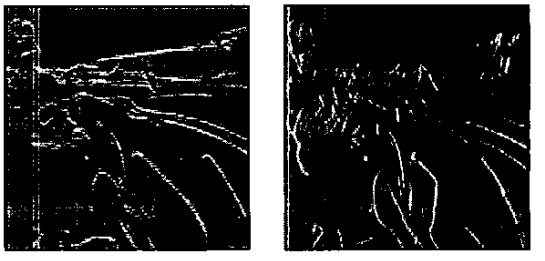

Figure 3: *the vertical and horizontal edge filters (same image as before).*

4.4 Summary

This section will summarize, into a table, the different components of the distance we want to minimize, in function of the hierarchy level. Table 1 gives some indication of the coefficients affected with each element of the distance. In our experiments, we used linear function of the level, with a fixed slope. The next step will be to automatically adapt these functions during the tracking, using information provided by the previous mappings.

Element of the distance	impact for low levels	impact for high levels
intensity	very high	medium
position	low	low
edge	medium	medium

Table 1: *The elements of the distance.*

5 How to define the object we track

5.1 The initial definition

The object is initially defined manually by the user, who draws the contours of the object to track. Then, the algorithm finds the nearest contour from the drawing; using a certain tolerance, this contour is based on the intensity of the pixels, the edge detection, and the distance from the original drawing. Figure 4 is an example of the drawing of the user, and what the algorithm infers.

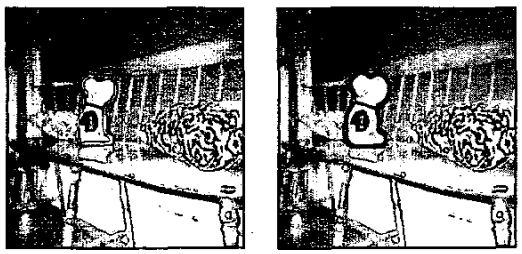

Figure 4: *original drawing and what is deduced.*

5.2 The points that are mapped

The surface that is deduced by the algorithm will define the points of the source image we will be tracking. As we don't want to map every point (this would take too much time), only several point are mapped. As we do not have any precise information about the structure of the object (it might be deformable or not for example), we will only map the points of the objects that are also at the intersection of the lines of a grid (the size of the grid is automatically adapted, depending on the surface of the object, and the number of points we want to map). Figure 5 is the result of such a mapping.

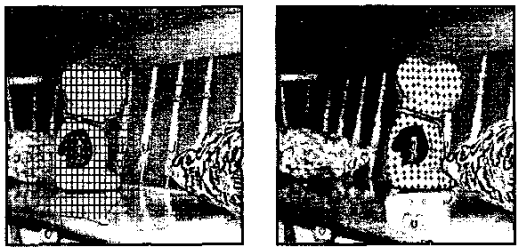

Figure 5: *Only the white points on the left image are mapped to the points on the right image*

5.3 How to recover the object

Once the mapping is done, we have to deduce the object from the points. This is done with a kind of painter algorithm starting with every mapped point, but with several restrictions :

- The points cannot spread further than n points from the original point. Typically, n is equal to the distance between two lines on the grid, normalized by the ratio of the predicted surface of the object on the destination image over the surface of the object on the source image.
- The points cannot spread over a edge, with a certain threshold (to be fixed by the user).
- The points can exceptionally spread over n, if an already spread point is nearer than k point from it (for example k equals 2 or 3), and there is no edge between them (over a certain threshold). This is done in order to fill in the gaps that may appear between the points.

Once this is done, we can deduce the new center of the object, the new surface, calculate the predicted parameters and run the algorithm on the new image. This allow real-time tracking of the object. Here is an example (figure 6) of pixels mapped on the destination image, and what the algorithm deduced from these pixels(for easy understanding, only the borders have been drawn and they are bigger than what the algorithm actually returns).

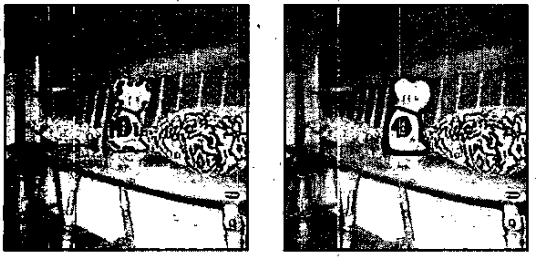

Figure 6: *mapped points and what is deduced.*

5.4 The case of multiple objects (or splitting object)

In the initial definition of the object, it is possible for the user, to define several objects at a time. As the algorithm maps points of the 'global' object, and then spreads them, the two objects will be mapped separately and will give two different objects on the destination image. The case where the object splits is equivalent, as the points are mapped independently. Then, the algorithm can easily tell how many objects there are on the screen on a given time.

6 Results

The results received from the algorithm (see the last page of the article) were obtained on a 1.8 GHz PC, using 256*256 images. The left image is the source image, and the right one is the destination. The borders are drawn thicker so that it is easier to see them.

6.1 Tracking with a fixed camera

Figures 7,8 and 9. During these tests, the camera was fixed, and only the object was moving. The results appear quite good, with only a minor error at the end of the legs. These good results are mainly based on the fact that the camera is not moving, so the algorithm 'matches' the background correctly.

6.2 Tracking with a mobile camera

Figures 10,11 and 12. During these tests, the object was not moving. The results are correct for translation, but for rotation and zooming, we still find some errors (especially for rotation). These problems will be partially solved when we will use the Kalman filter to predict the position of the camera (see the *Conclusion and future work* chapter)

6.3 Time of computation

This is the average time of computation and number of frames per second, we can handle, for the complete mapping of an object (of about 10% of the size of the image) with several resolutions: it appears that a 64*64 resolution is a good compromise between speed and quality.

Resolution	time/mapping	frames/second
*512*512*	3.33	0.3
*256*256*	1.25s	0.8
*128*128*	0.40s	2.5
*64*64*	0.12s	**8.2**
*32*32*	0.03s	30.7

Table 2: *Time of computation.*

7 Conclusion and future work

Here, we have shown a new method for object tracking, based on the results from Y.Shinagawa. The results are more than satisfactory, because the tracking almost never fails, even if there are small errors, despite the movement of the camera (in a certain range of amplitude) or the object. The fact that the algorithm can be used with a moving camera allows for use with a mobile robot. The tracking of 64*64 images at a rate of over 8 frames/second allows for real-time applications in a wide range of fields. We expect the algorithm to be optimized in the near future, by imaging the same number of frames/second, but for 128*128 images.
The most important focus, over the next few months, will be to automatically adjust the parameters for the different elements that compose the distance. By now, they are given manually, and remain the same during the tracking process. One other important improvement of the algorithm will be to track the object during a rotation that is not along the viewing axis. This could be done, for example, by using a database of what the object might look like, from several views. Finally, it could be interesting to take occlusions into account, by keeping track of the background.

Acknowledgments

I'd like to thank Yoshihisa Shinagawa for his permanent help with understanding his algorithm, and the way he actually implements it. Without this help, I would have certainly gone in the wrong direction on several points of his algorithm. I would also like to express my appreciation to Seth Hutchinson, who continually gave me excellent advice on what I needed to implement. His sense of what may work and what may not certainly saved me a lot of time. I also would like to thank everyone in the Robotics Lab at the University of Illinois, for their friendliness and the time they spent helping me with the computers and the robots.

References

[1] C. Rasmussen, K. Toyama, and G. Hager: "Tracking Objects By Color Alone",1996

[2] G. Hager and P. Belhumeur: "Efficient region tracking with parametric models of geometry and illumination",1998

[3] M. Kass, A. Witkin, and D. Terzopoulos: "Snakes: Active contour models",1988

[4] Y. Shinagawa and T.L. Kunii: "Unconstrained Automatic Image Matching Using Mutltiresolutional Critical-Point Filters",1998

[5] J. Ponce and D. Forsyth: "Computer Vision : a modern approach",2001

[6] K. Habuka, Y. Shinagawa and M. Hilaga: "Image interpolation using enhanced Mutltiresolution Critical-Point Filters: with applications to virtual pseudo-3D model generation and keyframe interpolation of videos",1999

[7] C. Tomasi and T. Kanade: "Shape and motion from image streams: a Factorization Method",1992

[8] E.P. Simoncelli, E.H. Adelson and D.J. Heeger: "Probability distributions of Optical Flow",1990

[9] G. Welsh and G. Bishop: "An introduction to the Kalman Filter",2001

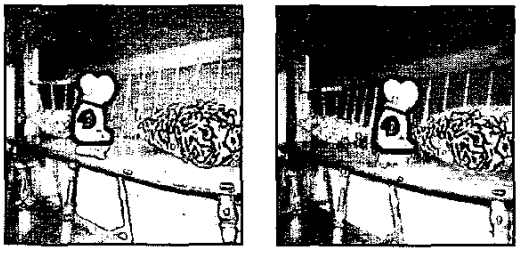

Figure 7: *tracking during object translation.*

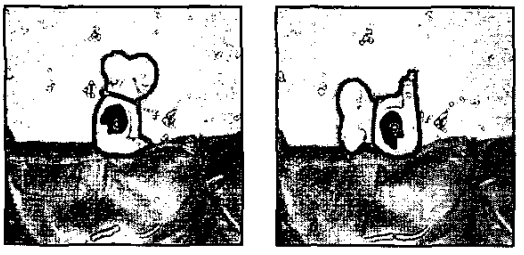

Figure 8: *tracking during object rotation.*

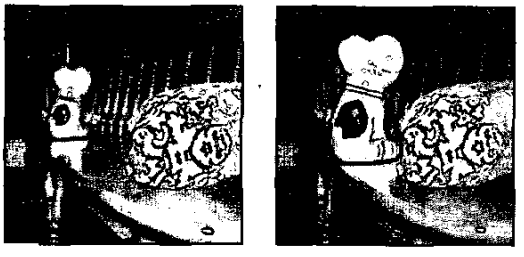

Figure 9: *tracking during object scaling.*

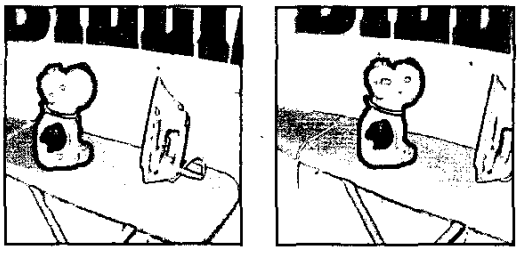

Figure 10: *tracking during camera translation.*

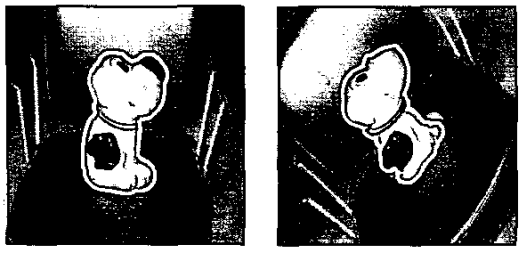

Figure 11: *tracking during camera rotation.*

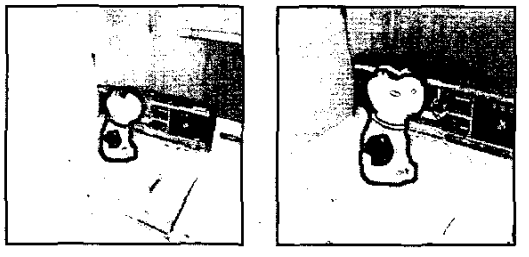

Figure 12: *tracking during camera zooming.*