15th Triennial World Congress, Barcelona, Spain

A FAST SEARCH STRATEGY FOR TEMPLATE MATCHING

Escudero-Rodrigo D. [(1)] **and Sánchez-Salmerón A.J.** [(2)]

Departamento de Ingeniería de Sistemas y Automática.
Universidad Politécnica de Valencia.
Camino de Vera. Apdo. 22012, E-46071, Valencia, Spain.
[(1)] *e-mail: dieesrod@inf.upv.es*
[(2)] *e-mail: asanchez@isa.upv.es*

Abstract: An active vision system with two cameras has to be calibrated on-line to perform stereo vision applications. A calibration of a binocular system is possible using a small number of homologous features in both images. The correspondence problem can be resolved by using traditional template matching algorithms but it has a high computational cost. In this paper, some dynamic programming aspects are introduced to deal with template matching in order to reduce the execution time. Some experimental results demonstrate their viability and suggest other lines to improve its performance. *Copyright © 2002 IFAC*

Keywords: Search methods, image matching, dynamic programming, calibration, stereo vision.

1. INTRODUCTION

An active vision system with two cameras (Fig. 1) has to be calibrated on-line to perform stereo vision applications (Trucco and Verri, 1998). Camera calibration in stereo vision is needed to extract metric information from 2D images. Calibration has three different phases: feature selection, correspondence problem and parameter estimation of fundamental matrix (Zhang, 1996; Sanchez and Calatayud, 2001).

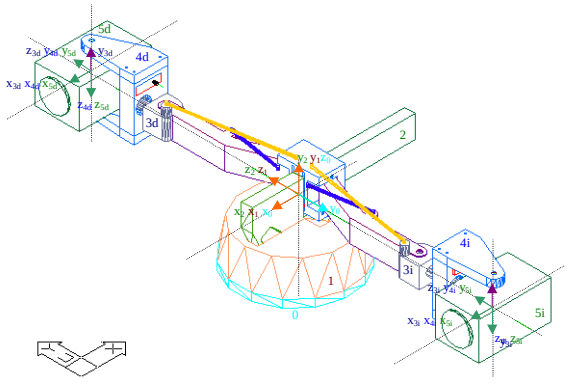

Fig. 1. Active vision binocular system called SiviS.

The main problem is to extract a set of *n* points from each image and to match each point in the first image to its corresponding point in the second one.

Feature selection involves extracting a valid set of features that appear in both images. The selected features for each image are windows (templates) around corners detected by SUSAN algorithm (Smith and Brady, 1997).

The second phase, correspondence problem, is most costly because it has to match each feature in one image with all features in the other. To overcome the correspondence problem, several schemes have been developed, which can be grouped into two categories: area-based (Kanade and Okutomi, 1994) and feature-based. In this work, the feature-based scheme is used.

The template matching algorithm has very high computational cost, the time spent in this phase is nearly ¾ of the calibration process. In this paper, the idea is to apply dynamic programming aspects to the template matching algorithm in order to improve its computational cost. If this cost is improved then the time in correspondence problem decreases. Therefore, the calibration process (Fig. 2) will be on-line.

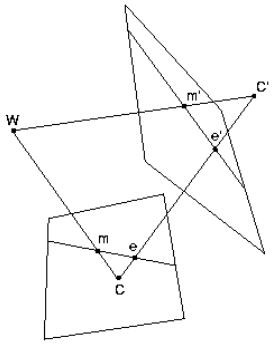

Fig. 2. Fundamental matrix: $m^T \cdot F \cdot m' = 0$.

2. TEMPLATE MATCHING ALGORITHMS

To locate an image template *T*, with pxq dimensions, in an mxn image *I*, it is necessary to find the minimum distance *D*(*p*) among those of all possible locations *p* that exist in an image. The possible locations are each window of the image with pxq dimensions that can be represented in it. The solution *s* is the location *p* that has the minimum distance of them all (1).

$$d = \min_{p=1}^{mxn} D(p) \quad (1)$$

The traditional algorithm (Fig. 3) searches for the best location in a unique scan of the image testing the template and calculating the distance for each pixel. Most template matching applications commonly use the correlation or the sum-of-squared-difference (SSD) as a distance measure to determine the best match. Unfortunately, these measures are sensitive to outliers and they are not robust to variations in the template. New applications are based on 'Hausdorff distance' measure (Huttenlocher et al., 1993) which provides robustness to partial variations of the template.

-Initialise first solution: d=D(1).
For each location p in the image...
  For each pixel t in the template...
    -Update accumulator D(p) with d_t.
  -Update solution: d=min (*d, D(p)*).

Fig. 3. Traditional algorithm.

It is assumed that the cost of calculating the distance *t* in the algorithms is worthless. Then, the computational cost of the traditional algorithm is C_t (2).

$$C_t = mxn \cdot pxq \cdot t \quad (2)$$

But it is not real because the search does not scan the whole image. There is part of image where the template does not fit. The real computational cost is (m-p+1)x(n-q+1) x pxq. That means that the influence of the template and the image is the same, and when the size of the template or image increases, the cost grows quickly.

Another important feature of this algorithm is that the position of the template in the image does not have an influence on its cost. The algorithm will scan all the locations in the image even if it has already found a local solution before completing the image.

In order to reduce the computational cost of the traditional algorithm a 'multi-resolution search' technique is often used (Rosenfeld, 1984). Recently new strategies, like 'eigenspace approximation' (Huttenlocher et al., 1999), have been introduced to improve this cost.

3. PROPOSED SEARCH STRATEGY

This search strategy can be used in any multi-resolution level and for any distance measure. The strategy has the next key features:

1. Array of possible locations.
2. Pruning.
3. Ordering by 'Quicksort'.

The proposed algorithm has been developed using some dynamic programming aspects like the pruning. So, it is necessary to change the way in which the image and the template are processed respect to that of the traditional one.

To calculate the distance from the possible locations in the image to the template it is necessary to create an array of possible locations (Fig. 4). This array must contain information of all possible locations in the image because the algorithm works with all the possible locations at the same time. Each component of the array contains the global position $p(x_0,y_0)$ in the image and a distance accumulator to calculate *D(p)*. This array must be dynamic because its size changes; when a possible location cannot be the solution it has to be pruned. The algorithm starts with an array size of (m-p+1)x(n-q+1) elements, these are the number of possible locations that can be in the image.

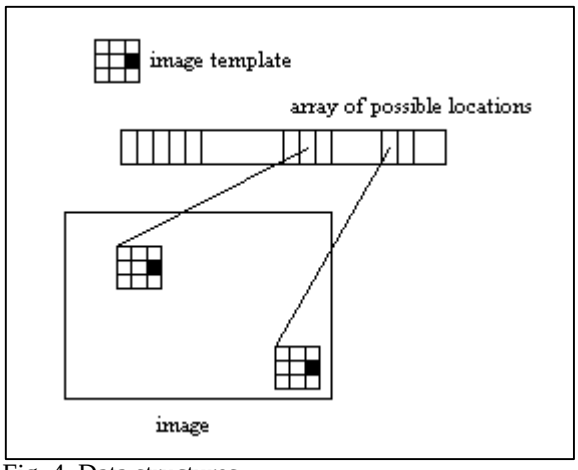

Fig. 4. Data structures.

-Initialise the array of possible location with d_1.
-Initialise d.
For each pixel t in the template...
  For each element p of the array...
    -Update accumulator D(p) with d_t.
  -Apply 'Quicksort' to the array.
  -Select most promising location s to calculate D(s).
  -Update solution: d=min (*d, D(s)*).
  -Prune the rest of elements if the accumulator is greater than or equal to the actual distance d.

Fig. 5. Scheme of the algorithm.

The proposed algorithm (Fig. 5) is not as easy to be understood as the traditional one. The process consists in two stages, width-accumulation and depth-search. First it carries out a width-accumulation in the array of possible locations and then a depth-search only for the most promising location *s* to update *d* which is used to prune.

The width-accumulation means that for each processed pixel of the template, all the accumulators of the array of possible locations are updated with pixel distance d_t. The distance accumulator is updated by adding the distance d_t from template pixel *t* to the corresponding image pixel relative to location *p*. To calculate the distance d_t it is necessary to know the pixel situation of the possible location in the image; it is known by using its global position, which is stored in the array, plus the relative position, which is that of pixel of the template being processed (Fig. 4).

After width-accumulation the array has to be ordered by distances using 'Quicksort'.

The depth-search consists in selecting the most promising location, which is the first element of the array because it has previously been arranged. Then a possible solution is obtained: it is the most promising location after having calculated the distance for all the template of the image. This possible solution is used to prune. Pruning is applied to all possible locations whose stored distance is greater than or equal to the distance of the current solution. Because they cannot be better than the current calculated solution.

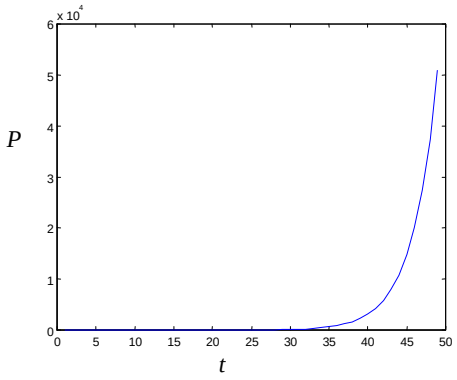

Fig. 6. Pruning function. This function is an estimation of the average amount of pruning locations for all pixels in the template.

The pruning function *P* is an important factor of the proposed search strategy because it is used to eliminate all the possible locations that cannot be improve the current calculated solution. This pruning function is not regular. In the first processed pixels of the template the number of possible pruned locations is small, as the algorithm advances in the process of the template the pruned elements grow exponentially (Fig. 6). The more pixels of the template have been processed the more probabilities for the current solution to be the final one, due to the fact that the number of possible pruned locations increases too.
Another important factor in the algorithm is 'Quicksort' because it is used to order the array of possible locations. After ordering the array it is easy to select the most promising location and to prune.

In the proposed algorithm, the same as in the traditional one, the position of the solution in the image does not influence on cost because the array of possible locations is ordered.

3.1 Computational Cost

The idea is that of the cost of the proposed algorithm is better than that of the traditional one because in the worst case, when the algorithm has not pruned the array, the cost of scanning all the pixels of the template, and for any of them all the pixels of the image, is the same as that of the traditional algorithm. And by pruning only one possible location the computational cost of the proposed algorithm is being reduced. As it is not necessary to scan all the possible locations of the image then the cost is reduced in (p×q)-(*k*×*l*) for each possible location pruned in the iteration *kl*. The cost of the algorithm when it prunes P_{kl} possible locations is (p×q)×(m-p+1)×(n-q+1) – (p×q-*k*×*l*)×P_{kl}. The cost interval of the algorithm can be determined analysing the extreme cases. In the best case: all possible locations can be pruned in the first iteration *k*×*l* = 1 then P_{kl} = (m-p+1)×(n-q+1)-1 and the cost is p×q + (*k*×*l*)×[(m-p+1)×(n-q+1)-1]. In the opposite case, the worst: no possible location is pruned P_{kl} = 0 in all iterations and the cost is equal to that of the traditional algorithm. This theory seems great but it is not completely true.

The computational cost of the proposed algorithm is C_d (3).

$$P = P + P_{kl}$$

$$C_d = \sum_{k=1}^{p}\sum_{l=1}^{q}\left(\sum_{k'=k}^{p}\sum_{l'=l}^{q} t + \sum_{u=1}^{v=[(m-p+1)(n-q+1)]-P} t + v\cdot\log v\cdot t'\right) \quad (3)$$

where:

P is the pruning function;

P_{kl} is the amount of possible locations pruned in the iteration *kl*;

v is the size of the array of possible locations;

$\sum_{k=1}^{p}\sum_{l=1}^{q}$ is the size of the template;

$\sum_{k'=k}^{p}\sum_{l'=l}^{q} t$ is the cost of depth-search;

$\sum_{u=1}^{v=[(m-p+1)(n-q+1)]-P} t$ is the cost of width-accumulation.

Although everything explained before is true there is one factor of the algorithm that has been ignored: to sort the array of possible locations by 'Quicksort'. It makes the algorithm more efficient, because the most promising location is always processed and the probability of pruning is increased.

The only problem is the cost of 'Quicksort'. Although it has the minimum medium cost of all sort-algorithms, its computational cost is vx$\log(v)$, when the array has v elements. That would not be a problem if they had to be ordered only once: when the first pixel of the template is processed. But they have to be ordered for all the pixels of the template, so the cost is pxq x vx$\log(v)$. This is only in theory, in practice the cost is lower. To order the array costs vx$\log(v)$ only the first time, the next ones the cost is less because the array has already been ordered and now it has fewer elements to be ordered. Also the number of elements of the array decreases when the pruning is applied.

With the ideas exposed above, it can be said that the relation between the size of the array and the number of the pruned elements is what defines the efficiency of the proposed algorithm. Because of this the proposed algorithm is not better than the traditional one in all the cases; there are a few defined cases where the traditional algorithm is more efficient (Fig. 7).

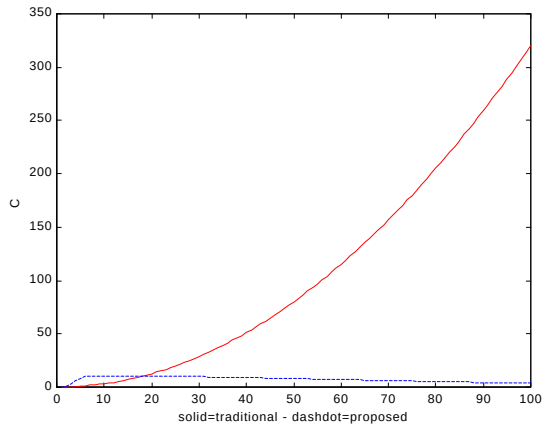

Fig. 7. It represents the cost functions of the traditional and the proposed algorithm. The functions are calculated for templates whose size is from one pixel to one hundred pixels, with an image with forty thousand pixels; for example an image of 200x200.

4. EXPERIMENTAL RESULTS

This section shows some of the results obtained using the described method. Experiments are performed using C in 550 MHz Pentium III PC running 'Red Hat Linux 6.2'. A set of real images has been tested using 'NetPBM' image library.

Table 1. Results of experimental tests. The third column is the amount of experiments performed to calculate the average costs $\overline{C_t}$ and $\overline{C_d}$.

Image	Template	Experiments	$\overline{C_t}$ sec.	$\overline{C_d}$ sec.
50x50	10x10	1626	0.22	0.13
50x50	25x25	663	0.57	0.006
100x100	10x10	9311	0.93	0.10
100x100	45x45	3872	7.43	0.034
200x200	5x5	33787	1.15	1.43
200x200	50x50	22165	67.53	0.68
200x200	95x95	10721	121	0.23
300x300	5x5	79678	2.72	8.34
300x300	10x10	14768	10.23	8.72

Table 1 gives some running times of the algorithms applied to different images and templates (Fig. 8). In this case, the sum-of-squared-difference (SSD) is used as a distance measure to determine the best match.

Fig. 8. A processed image and three searched templates.

5. CONCLUSIONS

Experimental results confirm that the proposed algorithm is more efficient than the traditional one but not in all the cases, as it has been stated before. The cases where the traditional one is better are those when the searched template is much smaller than the image. An example of it is figure 7, where the proposed algorithm has less cost than the traditional one if it uses a template with more than twenty pixels. Thus to use this algorithm is a good decision in cases when it is necessary to match all templates in an image except the small ones mentioned. Also with experimental tests it is discovered that the best results are obtained when the image size is twice the template. The obtained benefit is enormous respect to the traditional algorithm. It is because, if the difference between the template size and the image size is small, there are few possible locations for the image and then applying 'Quicksort' to the array is not as expensive as in other cases. This and the pruning function reduce the algorithm global cost.

At the moment the investigation continues. A new version of the algorithm can be implemented by using these ideas to improve the cost exposed in this paper. The image is divided to a number of sub-images, called windows, whose dimensions are twice

the dimensions of the template. Once that is done, the solution is only in one of the windows, because of this the pruning is slower in the other windows. Efficiency depends on which window the solution is, because the sooner the algorithm get the window where the solution is, the faster the pruning in the other windows. So the cost depends on the solution situation in the image. That is why it is necessary to make the algorithm independent of the solution situation in the image.

This problem would be solved if there was a slight idea about in which window the searched solution is. A 'multi-resolution search' could be a good technique to solve it.

The proposed search strategy permits to resolve efficiently the correspondence problem when the template size is bigger enough. In this case the cost of the correspondence problem is reduced and the calibration process will be on-line.

REFERENCES

Huttenlocher D.P., Klanderman G.A. and Rucklidge W.J. (1993). Comparing images using the Hausdorff distance. *In IEEE Trans. Pattern Analysis and Machine Intelligence*, **15(9)**, 850-863.

Huttenlocher D.P., Lilien R.H. and Olson C.F. (1999). View-based recognition using eigenspace approximation to the Hausdorff measure. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **21(9)**, 951-955.

Kanade T. and Okutomi M. (1994). A stereo matching algorithm with an adaptive window: theory and experiment. *IEEE Transactions of Pattern Analysis Machine Intelligence*, **16(9)**, 920-932.

Rosenfeld, A. (1984). *Multiresolution Image Processing and Analysis*. Springer-Verlag, New York.

Sanchez A.J. and Calatayud R. (2001). Auto-calibración de un sistema binocular de visión activa. *In Jornadas de Automática 2001*.

Smith S. M. and Brady J.M. (1997). SUSAN A new approach to low level image processing. *International Journal Computer Vision*, **23(1)**, 45-78.

Trucco E. and Verri A. (1998). *Introductory Techniques for 3-D Computer Vision*, Prentice-Hall, Englewood Cliffs, NJ.

Zhang Z. (1996). Determining the epipolar geometry and its uncertainty: A review. *Research Report*, No.2927, INRIA Sophia-Antipolis.