Fast Calculating Feature Point's Main Orientation in SURF Algorithm

Bing Han
Xi'an Research Inst. Of Hi-Tech
Hongqing, Xi'an, P.R.China, 710025
icytg@126.com

Yongming Wang，Xiaozhi Jia
Beijing Research Inst. Of Hi-Tech
Qinghe, Beijing, P.R.China, 100085
dayechia@sina.com

***Abstract*—in SURF algorithm, original process for feature point's main orientation assignment is not optimized. It uses a sliding window covering an angle with 60 degrees shift around a circle region, and calculates the sums of all Haar responses to yield a vector in it. Along with the window rotating, many overlap regions generated. Therefore, a lot of responses summations need to be repeatedly taken. So the speed is slow. An algorithm for fast calculating feature point's main orientation in SURF was proposed. It wiped off all repeated responses summations in the overlap regions, and all the Haar responses were just needed to be computed once, which not only decreased the complexity, but also increased the speed. Verified by the experimental results, the consuming time for our fast main orientation calculating algorithm is just 60% of the original one.**

Keywords-SURF algorithm; feature point; main orientation; fast calculating

I. INTRODUCTION

SURF, proposed by Herbert Bay in 2006[1, 2, 3], was an efficient algorithm for fast calculating and describing local invariant features. It is very fit for recognizing[4] and tracking[5] objects in many real-time applications. SURF's high efficiency is mainly owed to the using of integral image[6] and box filters, and their effects present much distinct in the steps of extracting and describing features. Although SURF is faster than other algorithms, such as SIFT[7], Harris-Laplace[8] and so on, there are still some sub-steps need to be optimized. Calculating feature point's main orientation is the typical.

Original method for calculating feature point's main orientation in SURF is using a sliding window covering an angle with 60 degrees shift around a circle region, and then calculating the sum of all Haar responses in it. When the sliding window shifting, there are many big overlap regions generated. Therefore, a lot of responses summations need to be repeatedly taken. So algorithm's speed is influenced by these and tends to be slow. To solve this problem, an algorithm for fast calculating feature point's main orientation in SURF was proposed. In part II of this paper, the original algorithm for calculating main orientation in SURF was introduced, and pointed out how the overlap regions and repeated summations generated. In part III, we detailedly analyzed the processes of our fast orientation calculating algorithm. In it all the Haar responses just needed to be summed once. So the complexity was decreased. In part IV, an experiment was taken to test our algorithm's efficiency. Firstly, in order to verify our fast algorithm is correct, we compared the results between the original and our fast algorithms. Because the results achieved by these two algorithms were quite similar, the fast algorithm's correctness was verified. Secondly, we compared the speed between these two algorithms. The results presented that our fast algorithm's consuming time was just 60% of the original one's.

II. ORIGINAL ALGORITHM FOR CALCULATING MAIN ORIENTATION IN SURF

In original orientation assignment algorithm[1,2,3], it first calculates the Haar wavelet responses in horizontal and vertical direction within a circular neighborhood of radius 6*s* around the interest point, with *s* the scale at which the interest point was detected. The sampling step is scale dependent and chosen to be *s*. And the size of the wavelets is scale dependent and set to a side length of 4*s*. Therefore, integral image can be used here for fast filtering. See figure 1 and 5.

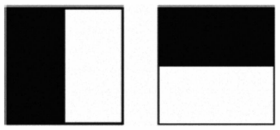


Figure 1. Haar wavelet filters to compute the responses in the horizontal (left) and vertical direction (right). The dark parts have the weight -1 and the light parts +1.

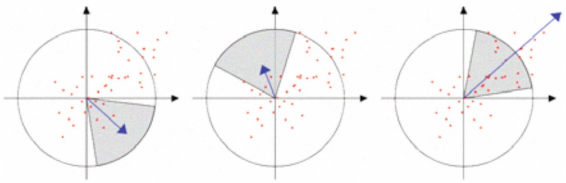


Figure 2. Orientation Calculation: As the 60 degrees window slides around the origin the components of the responses are summed to yield the vectors shown here in blue. The largest such vector determines the main orientation.

Once the wavelet responses are calculated and weighted with a Gaussian (σ=2s) centered at the interest point, the responses are represented as points in a space with the horizontal response strength along the abscissa and the vertical response strength along the ordinate. The main orientation is estimated by calculating the sum of all responses within a sliding orientation window of size 60

978-1-4244-7956-6/10/$26.00 ©2010 IEEE

degrees. The horizontal and vertical responses within the window are summed. The two summed responses then yield a local orientation vector. The longest such vector over all windows defines the orientation of the interest point. See Figure 2.

In literature [9], the shifting step of sliding window was chosen 5 degrees. So the local orientation vector could be initially calculated in the 0~60 degrees region, then in the 5~65 degrees region, and analogized like this. Among these two regions, 5~60 is an overlap region, when calculating the local orientation vector, the horizontal and vertical responses within the 5~60 degrees overlap region are needed to be summed twice. Along with the sliding window rotating, more overlap regions would be generated, responses in these regions need to be summed once and once. It made the algorithm process more complexity. So the original main orientation calculating method is not optimized. To solve this problem, an algorithm for fast calculating main orientation in SURF was proposed in part III.

III. FAST ALGORITHM FOR CALCULATING MAIN ORIENTATION IN SURF

In order to wipe off the repeatedly responses summations in the overlap regions, an optimized fast algorithm was proposed. Figure 3 shows a process of the sliding window shifting from 0~70 degrees. The initial location for the sliding window is in figure 3 (a), marked as a pink region. Figure 3 (b) shows this window contra rotated 5 degrees from (a)'s location, the real location was marked by the green region adding the pink region. Here the green region distinctly represents the overlap region between window in 0~60 and 5~65 degrees regions. Figure 3 (c) shows the sliding window contra rotated 5 degrees again from (b)'s location. Green region adding pink region also represents the window's real-time location in 10~70 degrees region. Green is the overlap region. So a rule could be found there. We don't need to repeatedly calculate the responses sum in every 60 degrees region any more. Summation results of the horizontal and vertical responses can be achieved just by using the sum in the front region add the next 5 degrees region's sum, and subtract the first 5 degrees region's sum in the front region, i.e. the summation results of responses in (b) could be achieved by the sum in (a)'s pink part subtracting the sum in (b)'s blue part and add the sum in (b)'s pink part. The process of our fast orientation calculating algorithm is designed as follows.

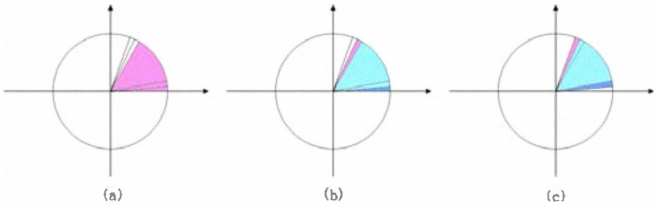


Figure 3. Process of the sliding window shifting from 0~70 degrees. For (a), the Haar responses sums in horizontal and vertical directions were marked as the pink region. For (b), the Haar responses sums in horizontal and vertical directions can be calculated by the frontal sum in (a) subtracting blue region's sum and adding pink region's sum in (b). For (c), the process is similar as (b).

Firstly, two arrays x and y with length 72 were defined to respectively store the Haar responses sum in horizontal and vertical directions in each 5 degrees region. That is like we divide a circle region into 72 parts and respectively compute the responses sum in them. Both values in these two arrays were initialized into 0. See figure 4.

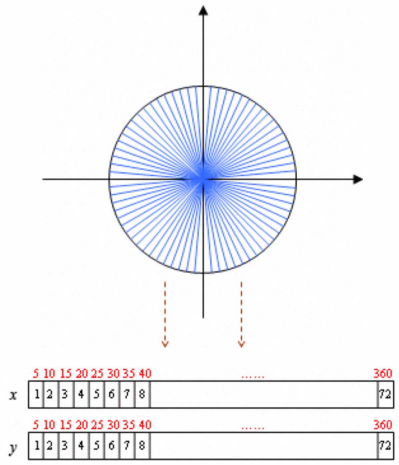


Figure 4. Two arrays with length 72 were defined to respectively store the Haar responses in horizontal and vertical directions in each 5 degrees region.

Secondly, we calculated the responses sum in horizontal and vertical directions in each 5 degrees region by using Haar wavelets and integral image. See figure 5. Suppose the responses in horizontal and vertical directions were defined as dx and dy, so the *angle* could be calculated using equation (1). After that, we using the *angle* divide 5. No matter the decimal part of the result whether was bigger than 0.5 or not, we just kept its integral number. So these responses dx and dy could be respectively added into arrays x and y based on equations (2) and (3). In these two equations index add one means the beginning indexes for x and y arrays were defined as 1.

$$angle = \arctan(dy / dx) \quad (1)$$

$$x[floor(angle/5)+1] = x[floor(angle/5)+1] + dx \quad (2)$$

$$y[floor(angle/5)+1] = y[floor(angle/5)+1] + dy \quad (3)$$

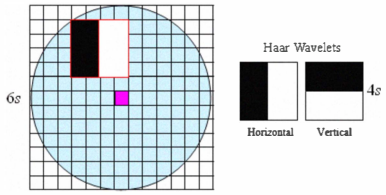


Figure 5. Calculating responses in horizontal and vertical directions by using Haar waveltes and integral image. Green region represents the locations which is need to calculate Haar responses.

Thirdly, local orientation vectors were calculated. After all responses sums were achieved in the blue region in figure 5, we summated each 12 values in x and y arrays for each 60 degrees region. The moving step was 1. Here we took a

circulation from index 1. If the index was bigger than 0 and smaller than 13, we directly added the sum values in x and y arrays one by one. Equations (4) and (5) were taken to do like this in the first 60 degrees region. If the index was bigger than 12 and smaller than 62, as discussed in the beginning parts, we used the frontal summation result add the sum in index n+11 and subtract the sum in index n-1. As shown in equations (6) and (7). Figure 6 introduced the whole process for calculating the responses sums, and it yielded an echo with figure 3. The local orientation vector could be calculated as equation (8).

$$sumx = sumx + x[n], (0 < n < 13) \quad (4)$$

$$sumy = sumy + y[n], (0 < n < 13) \quad (5)$$

$$sumx = sumx + x[n+11] - x[n-1], (12 < n < 62) \quad (6)$$

$$sumy = sumy + y[n+11] - y[n-1], (12 < n < 62) \quad (7)$$

$$vector = sumx \cdot sumx + sumy \cdot sumy \quad (8)$$

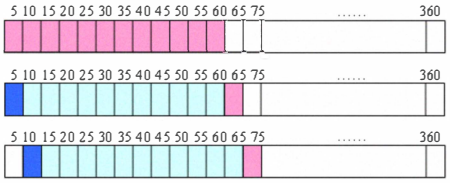


Figure 6. Calculting the reponses sums in each 60 degrees region.

At the end we choose the longest local orientation vector over all windows as the main orientation of the interest point.

Using this algorithm to calculate the orientation, the repeated summations were wiped off. All the summed responses were just called once. Comparing with the original algorithm, it decreased the complexity.

IV. Experiment Evaluation

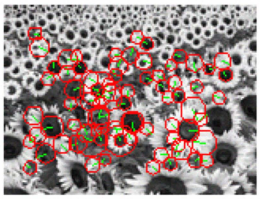
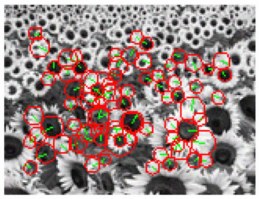
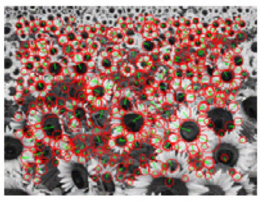
In order to verify our fast algorithm's effectiveness, an experiment was taken. All the results were obtained on a netbook with CPU Intel atom N270, running at 1.6GHz. The programming environment is MatlabR2009b. Test images is using a scaled sunflower field sequence, which size is from 160 × 120~ 800 × 600. The increasing step is 160 × 120. Figure 7 shows one picture in the sequence. Here we take 2 sub-experiments，one is to test our algorithm's correctness, by comparing the orientation calculating results with the original algorithm. The other one is to test our algorithm's speed whether faster or not than the original algorithm.

Figure 7. Test image sunflower field，size 320×240.

A. Result Evaluation

Firstly, we used the fast Hessian detector to extract the blob feature points in the sequence of test images. The number of the extracted feature points for these pictures is 78 453 1018 1843 and 2937 respectively corresponding to the picture sizes 160 × 120~800 × 600. Then we used the original and our fast algorithm both calculate these feature points' main orientation, and compared the results. These results were listed in Table I.

TABLE I. Orientation Results of Two Algorithms for Different Test Images

Images (Points)	Orientation Results of Two Algorithms	
	Original Algorithm	*Our Fast Algorithm*
160×120 (78)		
320×240 (453)		
480×360 (1018)		
640×480 (1843)		
800×600 (2937)		

Up to down, Table I listed the orientation calculation results for picture sunflower field in size 160×120~800×600. From these results we can see that the orientations calculated by these two algorithms are quite similar. So our fast algorithm is correct.

B. *Speed Evaluation*

After verified our fast algorithm's correctness for calculating orientation, we tested the speed based on image sequences using above. Firstly, we also used the fast Hessian detector to extract the blob feature points in each image. Numbers for these feature points respectively were still 78, 453, 1018, 1843 and 2937. Then based on these points, a comparison for the consuming time of these two algorithms was taken. Results were listed in Table II and figure 7.

TABLE II. COMPARING THE CONSUMING TIME OF TWO ALGORITHMS

Test Images and Feature Points Number		**Consuming Time for Two Algorithms**	
		Original Algorithm	*Our Fast Algorithm*
1	160×120 (78 Points)	0.226016	0.146972
2	320×240 (453 Points)	1.292421	0.806554
3	480×360 (1018 Points)	2.918384	1.792741
4	640×480 (1843 Points)	5.260386	3.234674
5	800×600 (2937 Points)	8.382818	5.173446

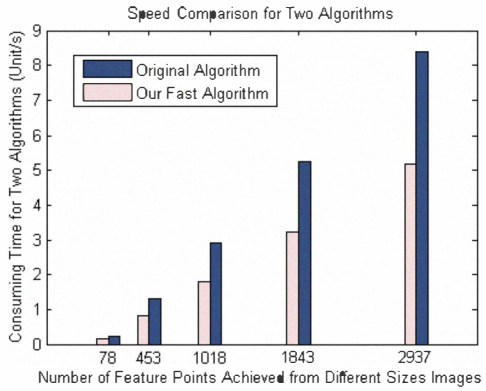


Figure 8. Bar Graphic for Comparing the Speed of Original Algorithm with Our Fast Algorithm

Table II shows the consuming time of these two algorithms. In order to introduce the comparison results more clearly, figure 8 was drawn based on the data in Table II. From these results it could be concluded that both the fast and original algorithm's speed were decreased along with the number of feature points increased. But the speed of our fast algorithm is always much faster than the original one, which is approximately just 61.7% of that for the original one.

V. CONCLUSION

An algorithm for fast calculating the feature point's main orientation in SURF was proposed. In original algorithm, when the sliding window rotated along a circle, some big overlap regions were generated. Therefore, a lot of repeated responses summation calculations contained in the original algorithm process, so its speed is slow. Our fast algorithm solved this problem. It not only keeps the correct results, but also decreased the algorithm's complexity, so it is much faster than the original algorithm.

REFERENCES

[1] H. Bay, T. Tuytelaars and L. Van Gool, "Surf: Speeded up robust features," in ECCV, 2006, pp. 404-417.

[2] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-Up Robust Features (SURF)," Computer Vision and Image Understanding, vol. 110, 2007, 346–359.

[3] H. Bay. "Wide-baseline point and line correspondences to 3D", Ph.D. Thesis, ETH Zurich, 2006.

[4] H. Bay, B. Fasel and L. Van Gool, "Interactive museum guide: Fast and robust recognition of museum objects," in Proceedings of the FirstInternational Workshop on Mobile Vision, 2006.

[5] A. C. Murillo, J. J. Guerrero and C. Sagues. "SURF features for efficient robot localization with omnidirectional images", 2007 IEEE International Conference on Robotics and Automation, Roma, Italy, April 2007, pp. 3901-3907.

[6] P. Viola and M. Jones. "Rapid object detection using a boosted cascade of simple features". IEEE Conference on Computer Vision and Pattern Recognition, 2001, pp. 511–518.

[7] D. G. Lowe. "Distinctive image features from scale-invariant keypoints". International Journal of Computer Vision, 2004, 60(2), pp. 91–110.

[8] K. Mikolajczyk, C. Schmid. "Scale and affine invariant interest point detectors", International Journal of Computer Vision, 2004, 60 (1), pp. 63−86.

[9] Christopher Evans. "Notes on the OpenSURF library", Online, Internet, available: http://www. chrisevansdev.com/. MSC Computer Science, University of Bristol, 2009.