2016 23rd International Conference on Pattern Recognition (ICPR)
Cancún Center, Cancún, México, December 4-8, 2016

Fast Template Matching Using Brick Partitioning and Initial Threshold

Fubito Toyama, Hiroshi Mori and Kenji Shoji
Graduate School of Engineering, Utsunomiya University
7-1-2, Yoto, Utsunomiya-shi, 321–8585 JAPAN
fubito@is.utsunomiya-u.ac.jp

***Abstract*—Template matching is a technique for finding a part of reference image which matches a template image. This paper presents a new fast template matching algorithm which can detect the most similar position. In the proposed method, first, an effective initial threshold is calculated using Winner Update Algorithm. Next, very fast template matching is achieved by using this initial threshold in Multilevel Successive Elimination Algorithm. Furthermore, Brick Partitioning which is a new partitioning method is used to reduce the computational cost of comparing a template with each position within reference image. Experimental results indicate that the proposed method can search faster than previous methods.**

I. Introduction

Template matching is a technique that compares the template with each position within reference image. The most similar position to template is found by template matching. A part of reference image which is compared with template image is called "window". Template matching is a basic algorithm which is applied to many applications, including image search, object tracking and machine vision. Many fast algorithms for template matching have been proposed in the past [1]–[7].

The Sequential Similarity Detection Algorithm (SSDA) [1] is one of traditional methods. SSDA reduces the cost of similarity calculation between a template and windows. Active Search [2] is a fast template matching method using a color histogram. By skipping neighborhood windows using upper bound of similarity, Active Search reduces the number of comparisons between a template and windows. However, it is difficult to detect object position precisely because the position data of color pixels is lost. To solve this problem, Kawanishi et al. proposed Adaptive Window Skipping method (AWS) [3]. In this method, Sum of Absolute Difference (SAD) is used as the evaluation of similarity (dissimilarity). The lower bound of distance between a template and a window is calculated using a subtemplate and a subwindow. AWS can find the most similar position faster than SSDA.

There are many methods [4]–[7] based on the lower (or upper) bound of similarity (dissimilarity). For examples, the calculation of SAD is skipped by the lower bound of dissimilarity in Successive Elimination Algorithm (SEA) [4]. The lower bound is calculated by the absolute difference between sum of pixels in a template and sum of pixels in a window. Gao et al. extended the SEA to multilevel successive elimination strategy (MSEA). In MSEA, higher (tighter) lower bounds are obtained by dividing a template and windows into multiple blocks. Therefore, many windows are skipped without calculation of SAD. On the other hand, although Winner Update Algorithm (WUA) [6] also uses the same lower bound as MESA, the search order of windows is optimized. By repeating updating the lower bound of the window which has the lowest lower bound, the number of calculations of SAD is minimized. However, when the number of windows is increased, much time is consumed to sort windows in ascending order of lower bound. The calculation of lower bounds is accelerated by an Integral Image [9]. Fast WUA using an Integral Image is proposed by Jung et al. [7].

Korman et al. proposed Fast-Match [10] which is a fast template matching algorithm under 2D affine transformations. However, the search space of 2D affine transformations is huge compared to that of translations. Therefore, we focus on the matching algorithm under all possible translations.

In this paper, we propose a new fast template matching algorithm which can detect the most similar position. The computational cost of comparing a template with all windows is reduced using Winner Update Algorithm (WUA) and Multilevel Successive Elimination Algorithm (MSEA). In the proposed method, first, an effective-initial threshold is calculated from a small number of windows using WUA. Next, very fast template matching is achieved by using this initial threshold in MSEA. Furthermore, Brick Partitioning which is a new partitioning method is used to reduce the computational cost of comparing a template with all windows. Brick Partitioning is applied to the template image. Experimental results indicate that the proposed method reduces search time substantially. Furthermore, the proposed method can search faster than fast WUA with an Integral Image.

II. Related works

A. Full Search (FS)

Let T be a template image of $M \times N$ pixels, where M and N are the horizontal and vertical size of the template. I represents the image under examination. Let the size of the image I be $X \times Y$. The Full Search (FS) algorithm calculates and compares the similarity (dissimilarity) for all the search positions (windows). In this paper, SAD is used as

978-1-5090-4846-5/16/$31.00 ©2016 IEEE

the evaluation of similarity (dissimilarity). The dissimilarity $d(x, y)$ between a template and a window is defined as

$$SAD : d(x, y) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} |T(i, j) - I(x + i, y + j)| \quad (1)$$

where (x, y) represents the position of windows in the image I, and $T(i, j)$ and $I(i, j)$ are the pixel values at position (i, j) of the template and window images. The number of windows in the image I is $(X - M + 1) \times (Y - N + 1)$. In FS, the dissimilarity $d(x, y)$ is calculated for all the windows. Then the position which has the lowest dissimilarity value is detected.

B. MSEA

MSEA reduces the computation cost by skipping the calculation of SAD. The lower bound of dissimilarity is used to decide whether the calculation of SAD can be skipped. When the lower bound between a template and a window is over a threshold θ, the window can be skipped. θ is the lowest dissimilarity value at the current search point. When the lower bound is higher, the probability of skipping a window is higher. In MSEA, higher lower bounds are obtained by dividing a template and windows into multiple blocks. Let l be the number of partitioning blocks (partition level). In the first partitioning ($l = 1$), a template and a window are partitioned into four subblocks with size $M/2 \times N/2$. Then, each subblock is partitioned into four subblocks with size $M/4 \times N/4$. This process is repeated until the size of the subblocks becomes 1×1 (see Fig. 1). In Fig. 1, UB_l represents the lower bound at partition level l. When the partition level is increased, the lower bound is also increased. The range of partition level is from 0 to $L-1$, where $L = log_2 N (N < M)$. The number of subblocks is $K_l = 4^l$. When the calculated UB_l is over the threshold θ, the window can be skipped without the calculation of SAD. The partition level starts from 0. If the $UB_L - 1$ (final level) $\leq \theta$, SAD is calculated. This process is performed for all windows. Then the window which has the smallest SAD is found.

1) Calculation of lower bound: Let $T_{sub(k)}$ and $W_{sub(k)}$ be k-th subblock at the partition level l, where the number of subblocks K_l is 4^l. The absolute difference between the subblock summations of the template and window is defined as

$$UBsub_{l,k}(x, y) = |\sum_{p=0}^{P-1} T^p_{sub(k)} - \sum_{p=0}^{P-1} W^p_{sub(k)}| \quad (2)$$

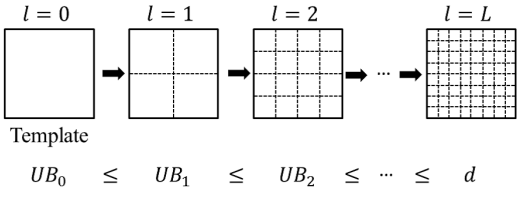


Fig. 1. Multilevel partitioning process.

where $T^p_{sub(k)}$ and $W^p_{sub(k)}$ are the p-th pixel values of subblocks $T_{sub(k)}$ and $W_{sub(k)}$, P is the number of pixels in the subblock. Therefore, the lower bound $UB_l(x, y)$ is defined as

$$UB_l(x, y) = \sum_{k=0}^{K_l-1} UBsub_{l,k}(x, y). \quad (3)$$

2) Relationship between lower bound and SAD: In MSEA, Minkowski's inequality is used for decision of skipping windows. By using the following inequality

$$|a + b| \leq |a| + |b| \quad (a, b \ni \Re) \quad (4)$$

the following relation is obtained

$$|\sum_{p=0}^{P-1} T^p_{sub(k)} - \sum_{p=0}^{P-1} W^p_{sub(k)}| \leq \sum_{p=0}^{P-1} |T^p_{sub(k)} - W^p_{sub(k)}|$$

$$\therefore \sum_{k=0}^{K_l-1} |\sum_{p=0}^{P-1} T^p_{sub(k)} - \sum_{p=0}^{P-1} W^p_{sub(k)}| \leq \sum_{k=0}^{K_l-1} \sum_{p=0}^{P-1} |T^p_{sub(k)} - W^p_{sub(k)}|$$

$$\therefore UB_l(x, y) \leq d(x, y) \quad (5)$$

where the left side of (5) represents the lower bound, and the right side of (5) indicates SAD. Similarly, the relationships between the lower bounds at each level are derived:

$$UB_0(x, y) \leq ... \leq UB_l(x.y) \leq UB_{l+1}(x, y) \leq ... \leq d(x, y). \quad (6)$$

(6) indicates that when the partition level l is increased, the lower bound UB_l also is increased. In MSEA, the lower bound UB_l is calculated in the order of $0, 1, \ldots, L-1$. If $UB_l > \theta$, the calculations of the remaining UB_l and $d(x, y)$ are skipped. The sums of blocks and subblocks are calculated very fast using an integral image. MSEA reduces the computational cost by window skipping process using the lower bounds.

C. WUA

WUA also uses the lower bounds obtained by partitioning a template and windows. However, the biggest difference is that WUA changes the order of window selection (search) based on the lower bounds. In MSEA, windows are selected and evaluated in the order of raster scan. On the other hand, in WUA the window that has the lowest lower bound is selected and partitioned into subblocks (partition level is increased by 1). Then the new lower bound of the partitioned window is recalculated. This selection and partitioning process is repeated until the window which has the smallest SAD value in lower bounds or SAD values of all windows is found. Therefore, the number of calculations of SAD is minimized. However, when the number of windows is increased, much time is consumed to sort windows in ascending order of the lower bound. The process of WUA is as follows.

1) Sums of pixels in each subblock of a template are calculated for each partition level.
2) The lower bounds of dissimilarity between a template and all windows are calculated at partition level 0 ($l = 0$)
3) The window which has the smallest lower bound $UB_l(x, y)$ is selected.
4) Each block of the selected window is partitioned into four subblock. Then $l = l + 1$. If $l = L$ then go to 5), otherwise go to 6).
5) SAD $d(x, y)$ is calculated. If the $d(x, y)$ is the smallest in all windows, then the position (x, y) is returned as the matching result and the process is finished, otherwise go to 3)
6) The new lower bound $UB_l(x, y)$ of the partitioned window is calculated. Then go to 3).

III. Template Matching Using Brick Partitioning and Initial Threshold

In this paper, we propose a fast template matching using an Effective Initial Threshold and Brick Partitioning (EIT+BP). Brick Partitioning is a new segmentation method for template matching. The initial threshold is obtained by WUA. First, a small number of windows are selected at equal intervals. WUA is applied to these windows. The time for sorting windows is very short because the number of windows is small. Therefore, an effective (small) initial threshold can be obtained very fast. Next, MSEA is applied to the rest of windows. A large number of calculations of SAD can be skipped by the effective initial threshold in MSEA. Furthermore, we propose Brick Partitioning which is a new partitioning method. The higher lower bounds can be obtained by Brick Partitioning. Therefore, many windows can be skipped without calculations of SAD. The proposed method achieves very fast template matching by Brick Partitioning and the effective initial threshold. Our algorithm strictly guarantees the same accuracy as FS.

A. Effective initial threshold estimation using WUA

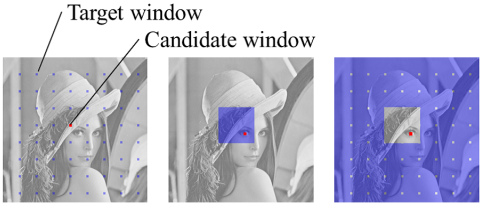
In the proposed method, the search process is divided into three stages. A coarse search based on WUA is conducted in order to obtain an effective initial threshold at stage 1 and stage 2. Next, MSEA is applied to the rest of windows at stage 3. In the first stage, windows are selected at equal intervals. Let $w(x, y)$ be the window of position (x, y) in a reference image. The set of all windows U is defined as

$$\mathbf{U} = \{w(x, y)|0 \leq x \leq X - M, 0 \leq y \leq Y - N\} \quad (7)$$

In the set U, the search windows at first stage are as follows

$$\mathbf{V_1} = \{w(x, y)|x = Sn, y = Sn, n = 1, 2, ...\} \quad (8)$$

where S is an interval parameter. Fig. 2(a) shows an example of target windows at stage 1. WUA is applied to the target windows. The window position with the smallest SAD value is obtained from the target windows. The red rectangle of Fig. 2(a) shows an example of the output window at stage 1. In stage 2, the neighborhood of the window obtained at stage 1 is searched (Fig. 2(b)). The window position which has smaller



(a) Stage1 (b) Stage2 (c) Stage3

Fig. 2. Target window at each stage.

SAD value is obtained from a small number of windows. Let the window position obtained at stage 1 be $w(x^1_{min}, y^1_{min})$. The search windows at stage 2 are defined as

$$\mathbf{V_2} = \{w(x, y)|x^1_{min} - S \leq x \leq x^1_{min} + S, \\ y^1_{min} - S \leq y \leq y^1_{min} + S\} \quad (9)$$

where S represents the size of the neighborhood. WUA is applied to these windows as well as stage 1. Then the window position $w(x^2_{min}, y^2_{min})$ with the smallest SAD value is obtained. $w(x^2_{min}, y^2_{min})$ has the smallest SAD value in V_1 and V_2. In stage 3, the remaining windows defined by equation (10) are searched.

$$\mathbf{V_3} = \{w(x, y)|w(x, y) \notin V_1, w(x, y) \notin V_2\}. \quad (10)$$

Fig. 2(c) shows an example of target windows at stage 3. MESA is applied to the remaining windows. The SAD value of $w(x^2_{min}, y^2_{min})$ is used as the initial threshold θ in MESA. In general MESA, it is difficult to perform effective window skipping while the threshold θ is high. On the other hand, in the proposed method a large number of windows can be skipped from the beginning because the small threshold θ is obtained through stage 1 and stage 2. In general WUA, when the number of search windows is large, much time is consumed to sort windows. On the other hand, in the proposed method, the time for sorting windows is very short because the number of search windows is small at stage 1 and stage 2. The proposed method achieves very fast template matching by incorporating the advantage of WUA and MESA. The final matching result (x_{min}, y_{min}) is obtained through stage 3.

B. Brick Partitioning

In general MESA and WUA, the size of subblocks at each level is the same. But the lower bounds obtained from the same size partitioning are not effective for skipping windows. We propose an adaptive partitioning method in which effective lower bounds can be obtained. If the distribution of pixel values in each subblock is highly nonuniform, the probability of obtaining a small lower bound is high. Because there is a possibility that the absolute difference between the sums of the two subblocks is smaller even though the dissimilarity between two subblocks is very high. On the other hand, if each subblock image is smooth, the probability of obtaining a large lower bound is high. Therefore, if a template can be partitioned into subblocks such that the dirtribution of pixel values in each

subblock is uniform, the higher (tighter) lower bounds can be obtained. We propose a new partitioning method, Brick Partitioning, in which effective lower bounds can be obtained. In Brick Partitioning, a template is partitioned based on image complexity. Gradient Magnitude is used to measure the image complexity. Gradient Magnitude is defined as

$$\begin{aligned} \| G[T(i,j)] \| &= \| \nabla T(i,j) \| \\ &= \sqrt{G_x(i,j)^2 + G_y(i,j)^2} \\ &\approx |G_x(i,j)| + |G_y(i,j)| \end{aligned} \tag{11}$$

where $G[T(i,j)]$ represents Gradient of the template image $T(i,j)$. $G_x(i,j)$ and $G_y(i,j)$ denote Gradient of x-axis and y-axis directions, respectively. $G_x(i,j)$ and $G_y(i,j)$ are defined as

$$G_x(i,j) = T(i+1,j) - T(i,j) \tag{12}$$

$$G_y(i,j) = T(i,j+1) - T(i,j) \tag{13}$$

First, partitioning positions along the x axis are calculated. The sums of Gradient Magnitude in the x direction are calculated by

$$H_y(j) = \sum_{i=0}^{M-1} \| G[T(i,j)] \| . \tag{14}$$

The sum of $H_y(j)$ is calculated by equation (15).

$$SH = \sum_{j=0}^{N-1} H_y(j). \tag{15}$$

u-th partitioning position $sp_y^l(u)$ at partition level l is obtained from equation (16).

$$\sum_{j=sp_y^l(u-1)}^{sp_y^l(u)-1} H_y(j) = \frac{SH}{2^l} \tag{16}$$

where $sp_y^l(0) = 0$, $sp_y^l(2^l) = N$ and $0 \leq u \leq 2^l$.

Next, partitioning positions along the y axis are calculated. The partitioning positions are calculated for each partial region from $sp_y^l(u-1)$ to $sp_y^l(u)$. The sums of Gradient Magnitude in the y direction are calculated by

$$H_x^u(i) = \sum_{j=sp_y^l(u-1)}^{sp_y^l(u)-1} \| G[T(i,j)] \| . \tag{17}$$

The sum of $H_x^u(i)$ is calculated by equation (18).

$$SH^u = \sum_{i=0}^{M-1} H_x^u(i). \tag{18}$$

In the partial region from $sp_y^l(u-1)$ to $sp_y^l(u)$, u-th partitioning position $sp_x^l(u,v)$ at partition level l is obtained from equation (19).

$$\sum_{i=sp_x^l(u,v-1)}^{sp_x^l(u,v)-1} H_x^u(i) = \frac{SH^u}{2^l} \tag{19}$$

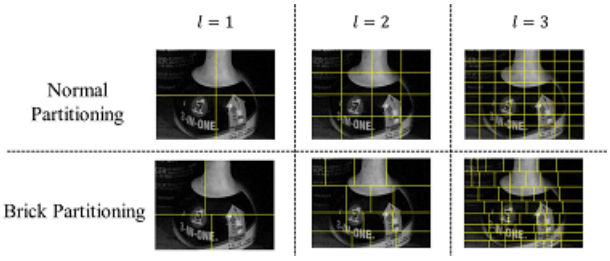


Fig. 3. Example of template partitioning results by each method.

TABLE I
DESCRIPTION OF EXPERIMENTAL DATA.

Template	Reference size	Template size	Correct point (x,y)
Lenna1	512×512	128×128	(231, 244)
Lenna2	512×512	64×64	(96, 128)
Object1	800×600	160×120	(240, 60)
Object2	800×600	200×150	(440, 405)
Bread1	800×600	100×75	(100, 150)
Bread2	800×600	160×120	(320, 360)
Face1	896×592	64×64	(118, 305)
Face2	896×592	400×300	(150, 515)
Cabriolet1	896×592	112×74	(303, 296)
Cabriolet2	896×592	224×148	(672, 148)
SUV1	896×592	112×74	(516, 385)
SUV2	896×592	224×148	(448, 148)
Road1	1760×1168	224×148	(775, 980)
Road2	1760×1168	400×300	(280, 575)
Yard1	1760×1168	200×150	(1065, 900)
Yard2	1760×1168	400×300	(150, 515)

where $sp_x^l(u,0) = 0$, $sp_x^l(u,2^l) = M$ and $0 \leq v \leq 2^l$. The sums of Gradient magunitude in each subblock partitioned by $sp_y^l(u)$ and $sp_x^l(u,v)$ is almost the same. Therefore, obtaining higher lower bounds is expected. Fig. 3 shows an example of template partitioning results by normal partitioning and Brick Partitioning. The subblock size of smooth region is large in Brick Partitioning.

IV. EXPERIMENTAL RESULTS

In our experiments, we compared the proposed method with the previous methods. FS and WUA [7] with Integral Image were selected as the previous methods. Furthermore, in WUA and the proposed method, normal (equal) partitioning (WUA, EIT) and Brick Partitioning (WUA+BP, EIT+BP) are applied and compared. Integral Image [9] was used in the calculation of the lower bound. The parameter S in the proposed method was set to 8. All of the algorithms in the experiments were run on the same machine (CPU: Intel(R) Xeon(R) X5650 2.67GHz). The programs for all methods were written in C++ and compiled in g++ with option -O3. "Lenna" image and images from Caltech dataset were used in the experiments. Each image was converted from RGB to gray-scale. Two template images were selected from an image. Gaussian noise was added to each template image. Therefore, the minimum value of SAD is over 0 ($d(x_{min}, y_{min}) > 0$). Dataset used for the experiments are shown in Table I. Fig. 4 shows examples of

TABLE II
COMPUTATION TIME [MS] AND RATIO OF COMPUTATION TIME(SET FS TO 100%).

Template	FS		WUA		WUA+BP		EIT		EIT+BP	
	Time	%	Time	%	Time	%	Time	%	Time	%
Lenna1	4551	100.00	93	2.04	87	1.91	34	0.74	**27**	**0.59**
Lenna2	1643	100.00	130	7.91	121	7.36	40	2.43	**34**	**2.06**
Object1	10441	100.00	182	1.74	172	1.64	62	0.59	**55**	**0.52**
Object2	14009	100.00	172	1.22	168	1.19	61	0.43	**50**	**0.35**
Bread1	5075	100.00	224	4.41	179	3.52	97	1.91	**88**	**1.73**
Bread2	10270	100.00	204	1.98	182	1.77	61	0.59	**53**	**0.51**
Face1	3044	100.00	147	4.82	148	4.86	50	1.64	**33**	**1.08**
Face2	15692	100.00	114	0.72	101	0.64	36	0.22	**33**	**0.20**
Cabriolet1	6236	100.00	251	4.02	243	3.89	**82**	**1.31**	84	1.34
Cabriolet2	17333	100.00	197	1.13	165	0.95	70	0.40	**53**	**0.30**
SUV1	6200	100.00	260	4.19	244	3.93	80	1.29	**62**	**1.00**
SUV2	17032	100.00	189	1.10	178	1.04	85	0.49	**57**	**0.33**
Road1	87265	100.00	527	0.60	522	0.59	221	0.25	**203**	**0.23**
Road2	242396	100.00	396	0.16	401	0.16	130	0.05	**121**	**0.04**
Yard1	80851	100.00	650	0.80	607	0.75	170	0.21	**168**	**0.20**
Yard2	240544	100.00	465	0.19	432	0.17	177	0.07	**166**	**0.06**
Average	47661	100.00	263	0.55	247	0.51	91	0.19	**81**	**0.16**

Reference : Object

Template : Object 1

Template : Object 2

Fig. 4. Reference image and corresponding templates.

reference and template images. The proposed method, FS and WUA were applied to the dataset. These methods guarantee the global optimality of the best match. Therefore, matching resluts of all experiments were the same in all algorithms (Correct points of Table I are obtained in all algorithms). Table II shows the computation time [ms] and the ratio of the computation time compared to that of FS algorithm (set FS to 100%). The best values are printed in boldface. The computation time of EIT and EIT+BP are reduced dramatically by the proposed initial threshold estimation. The proposed EIT+BP was the fastest method on average time. The average time of EIT+BP was only 0.16% of that of FS (about 600 times speedup). Furthermore, the computation time of EIT+BP was 0.04% and 0.06% of that of FS with respect to Road2 and Yard2 templates. The size of these template and reference images is larger than others. Thus the proposed method works effectively for large size images. In comparison of the partitioning methods, the average time of EIT+BP was 89% of that of EIT. Furthermore, the average time of WUA+BP was 93.9% of that of WUA. These results indicate that the proposed Brick Partitioning is an effective partitioning method for the lower bound based template matching.

V. CONCLUSION

In this paper, we have proposed a fast template matching algorithm using Brick Partitioning and an effective initial threshold. An effective-initial threshold is calculated from a small number of windows using WUA. Then, very fast template matching is achieved by using this initial threshold in MSEA. Furthermore, Brick Partitioning which is a new partitioning method is used to reduce the computational cost of comparing a template with all windows. Experimental results showed that the proposed method reduced the computational time dramatically.

REFERENCES

[1] D. I. Barnea and H. F. Silverman, "A class of algorithm for fast digital image registration", IEEE Trans.Comput., Vol.C-21, No.2, pp.179-186, 1972.

[2] V. V. Vinod, H. Murase, "Focussed color intersection with efficient searching for object extraction", Pattern Recognition, 30(10), pp.1787-1797, 1997.

[3] T. Kawanishi, T. Kurozumi, K. Kashino and S. Takagi, "A fast template matching algorithm with adaptive skipping using inner-subtemplates' distances", Proceedings of the 17th IAPR International Conference on Pattern Recognition (ICPR2004), Vol.3, pp.654-657, 2004.

[4] W. Li and E. Salari, "Successive elimination algorithm for motion estimation", IEEE Trans.ImageProcess., Vol.4, No.1, pp.105-107, 1995.

[5] X. Q. Gao, C. J. Duanmu, C. R. Zou, "A multilevel successive elimination algorithm based on the block sum pyramid",IEEE Trans.ImageProcess., Vol.9, No.3, pp501-504. 2000.

[6] Y. Chen, Y. Hung, and C. Fuh, "Fast block matching algorithm based on the winner-update strategy", IEEE Trans. Image Process., vol. 10, No. 8, pp.1212-1222, 2001.

[7] J. Jung, H. Lee, J. Lee and D. Park, "A Novel Template Matching Scheme for Fast Full-Search Boosted by an Integral Image", IEEE single processing letters, Vol.17, No.1, pp.107-110, 2010.

[8] http://www.vision.caltech.edu/archive.html

[9] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features", Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR2001), Vol.1, pp.511-518, 2001.

[10] S.Korman, D.Reichman, G.Tsur and S.Avidan, "FasT-Match: Fast Affine Template Matching", The IEEE Conference on Computer Vision and Pattern Recognition(CVPR), pp.2331-2338, 2013.