BIM: Ball Intersection Multi Template Matching

Bat El Shlomo
batelshlomo@mail.tau.ac.il
Electrical Engineering
Tel-Aviv University

Shai Avidan
avidan@eng.tau.ac.il
Electrical Engineering
Tel-Aviv University

Abstract

BIM is a multi-template matching algorithm. As opposed to traditional template matching algorithms that match a single template to a single image, BIM attempts to match multiple templates to a single image at once. A naive approach to multi-template matching would be to run a standard template matching algorithm sequentially with each of the templates and report the best result. Instead, each template processed by BIM further restricts the search space of the following templates, thus speeding up the overall process. In particular, we extend a recently introduced method for single template matching under 2D affine transformation to work with multiple templates at once. As a result, given a library of templates we can efficiently find the best 2D affine transformation for each of them in a target image. Experiments on real data sets reveal speedups of between 10 and 17.

1 Introduction

A variety of computer vision problems rely on template matching as a core component. This includes 3D reconstruction, object tracking and texture synthesis to name a few. Typically, template matching algorithms attempt to match one template to one image. Here we consider the case where we have multiple templates and one target image. Our goal now is to find a match for each template and do it *efficiently*.

In some cases, we are given multiple templates and wish to match each and every one to the query image. A naive approach is to run the base template matching algorithm with each template separately. We show that we can take advantage of information gathered after processing k templates to speed up the processing of template $k+1$.

Template matching algorithms usually work in a sliding window approach. This implies a 2D translation model. But templates occur in an image in a variety of scales, rotations and positions. Therefore, it is beneficial to consider a richer family of transformations, such as 2D affine transformations.

The space of 2D affine transformations is huge yet it was shown [8] that it is feasible to sample this space efficiently in reasonable time. The key observation there is that the sampling rate in transformation space depends on the smoothness of the image. A smooth image implies a lower number of transformations to be evaluated, and vice-versa. The algorithm, termed FAST-Match, is further accelerated with a branch-and-bound scheme. The advantage of this approach is that it is now possible to report a guaranteed approximation to the globally optimal transformation.

Our algorithm treats templates as points in some high dimensional space and consists of two steps. In a pre-processing step we compute all pair-wise distances between the templates. Then, at run time, we run FAST-Match a few times with different reference templates. The distance between a template and the query region serves as the radius of a ball in high dimensional space. The intersection of these balls quickly shrinks, leading to a reduction in the number of potential template matches.

We make two additional extensions to FAST-Match. First, we use each additional template to tighten the thresholds of the Branch-and-Bound scheme reducing even further the number of transformations that should be evaluated. Second, we extend FAST-Match to work with deep features, where each pixel is now represented with the response to layers of a deep convolutional network. The problem with this representation is that run time now increase linearly with the dimension of the deep features. We circumvent this added cost by using a fast and efficient quantization step.

We test our approach on a number of real world datesets and observe consistent speed up of between 10 and 17, compared to sequential application of the base template matching algorithm.

2 Background

Template Matching has been studied extensively and a full review is beyond the scope of this paper. See [16] for a recent survey of template matching algorithms under 2D translation model.

Traditionally, template matching methods match a single template to a single query image. However, in recent years a different approach emerged that matches many templates to a target image. The approach, termed Approximately Nearest Neighbor Fields (ANNF), tries to match each patch in the source image to a corresponding patch in the target image. The key insight of such methods is that information from one template (i.e., patch) is propagated to nearby patches. However, the propagation is based on proximity in the image plane and *not* on similarity in appearance space.

A seminal work on the topic of ANNF is that of PatchMatch [2] and its extension, that supports rotations and scales, Generalized PatchMatch [3]. These methods leverage the fact that multiple templates are matched at once to propagate information between templates. An alternative solution, based on KD-trees, was proposed by He and Sun [7] and later by Olonetsky and Avidan [15]. These methods store all patches of the target image in a KD-tree. Template matching is thus reduced to sequential querying of the KD-tree. Significant speedup is achieved by propagating information from nearby templates. Again, the propagation is based on proximity in the image plane and not on similarity in appearance space. A limitation of all these methods is that they do not generalize well when the transformation space increases, say to the space of 2D affine transformations.

An alternative approach to ANNF is key-point matching. These techniques reduce images and templates to a collection of key-points where each key-point is described by a descriptor. Matching keypoints boils down to matching their descriptors. Examples include SIFT [9] and its affine version ASIFT [13], Harris-Affine [12], and MSER [6]. To handle large number of descriptors, Muja and Lowe developed FLANN [14], a fast approximate nearest neighbor library. Other fast ANN methods include the work of Arya *et al.* [1] and

Silpa-Anan and Hartley [17] that both use kd-trees. These methods work very well in practice but suffer from two important limitations. First, they mainly support 2D translation models and it is not trivial to extend them to support rich geometric transformations such as 2D affine transformations. Second, they mainly focus on *single* template matching (i.e., a single query retrieval). That is, the result of one template match does not affect the performance of the next template match.

This paper focus on multi template matching without assuming that the templates are chosen to be image key-points, (i.e. we can consider templates from smooth areas of an image) nor does it assumes that the templates are located nearby in the image plane. On top of that, our approach can inherently deal with complex geometric transformations, such as 2D affine transformation.

BIM builds upon and extends FAST-Match (Korman *et al.* [8]). FAST-Match is a fast algorithm for approximate template matching under 2D affine transformations that minimizes the Sum-of-Absolute-Differences (SAD) error measure. It samples the space of 2D affine transformations with density that depends on the smoothness of the image. The algorithm is further accelerated in two ways: first, it approximates the SAD error by randomly examining only a small number of pixels. Second, FAST-Match uses a branch-and-bound scheme. This way the algorithm samples promising regions densely while pruning other regions of transformation space.

FAST-Match was compared to alternatives such as optic-flow and feature point methods and found to be superior. In particular, optic-flow methods, such as Lucas-Kanade [10] can handle 2D affine transformations but require a good initial guess. Interest point based methods such as SIFT [9], or its extension ASIFT [13] do not require an initial guess but require that the template has enough interest points that can be reliably described and matched.

Another method that inspired our work is that of RIANN, or Ring Intersection Approximate Nearest Neighbor Search [4]. RIANN is an algorithm for approximate nearest neighbor field estimation in video. It deals with multi-template matching and it uses ring intersection, which inspired our Ball Intersection scheme. However, RIANN relies on the temporal coherency of video for its high efficiency. We do not make this assumption. We, on the other hand, give an analytical solution to trim to the number of potential template matches. In addition, RIANN assumes that all query patches in the target image have a good match to the set of templates. We do not make this assumption.

3 Algorithm

The BIM algorithm consists of a pre-processing step in which it computes all pair-wise template distances. Then, BIM selects a small set of reference templates and given the query image, it runs standard FAST-Match on each of the templates separately. BIM relies on reference templates information to quickly reject the vast majority of transformations for the remaining templates, as we describe next. BIM assumes that all patches are points in some high dimensional appearance space and that the distance measure used supports the triangle inequality.

3.1 Formulation

Let $T = \{t_i : 1 \leq i \leq N\}$ be a set of templates. For simplicity, we assume all templates are of the same size. Let $d_i = d(t_i, p_{t_i})$ denote the L_1 distance, in appearance space, between

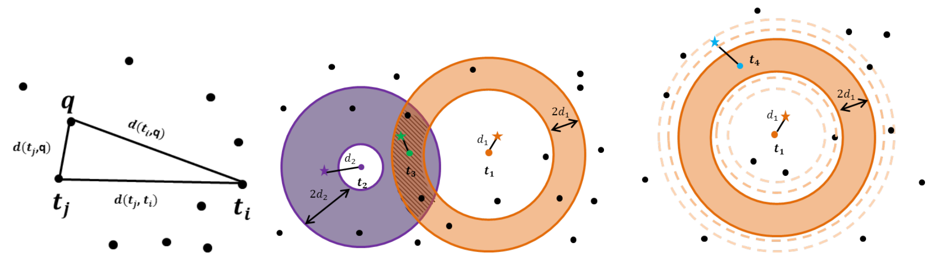

Figure 1: BIM search area illustration: Left image - triangle inequality including a non-reference template t_j, its match q and a reference template t_i. Middle image - example of area intersection by two templates. t_1, t_2, reference templates that creates rings areas for a non-reference template t_3. The transformations candidates are ones in the dashed area. Right image - example of expanding search area when it is empty. In this example, two area expansions are needed to reach candidates.

template t_i and its closest match in the image, denoted p_{t_i}. The task of finding best matched template can be written as follows:

$$t_{match} = \underset{t_i}{\operatorname{argmin}}(d_i),\ 1 \leq i \leq N \tag{1}$$

Observe we only look for a *single* template with a *single* transformation that minimize Eq 1. We will later show that the approach can be extended to find the best transformation for *each* template with very little additional computational cost.

By convention, let the first k (we use $k = 3$) templates denote the reference set. A non-reference template t_j $(j > k)$ will become t_{match} if its match, denoted as q, has a smaller error in terms of a defined distance error function. We efficiently perform the evaluation using the following observations. The first observation is that t_j is a better template than t_i only if a patch q exists such that

$$d_j = d(t_j, q) \leq d_i \tag{2}$$

Assume we have such a match q, can we say something about $d(t_i, q)$? The triangle inequality can be used to derive a connection between distances from the non-reference template t_j and the reference template t_i as follows

$$d(t_i,t_j) - d(t_j,q) < d(t_i,q) < d(t_i,t_j) + d(t_j,q) \tag{3}$$

Figure 1 shows an illustration of the triangle inequality. Merging Eq 2 into Eq 3 outputs the search area of q using distance from a *reference template*

$$d(t_i,q) \in [d(t_i,t_j) - d_i, d(t_i,t_j) + d_i] \tag{4}$$

This is the crux of the algorithm. Only transformations that lie within the range defined by Eq 4 from template t_i will be considered as candidates of template t_j. The ranges form a search area for each non-reference template. Figure 1 shows a 2D visualization using a Euclidean distance measure (in practice we use the L_1 distance). If $d_i < d(t_i,t_j)$ then the area of candidates defines an annulus with radius $d(t_i,t_j)$ around t_i and with width $2d_i$. Otherwise, it defines a circle with radius $d(t_i,t_j) + d_i$ around t_i.

In case of multiple reference templates, the search area becomes the intersection of multiple balls. BIM - ball intersection match, stands for the intersection area under L_p ball distance metric.

Extending the algorithm to find the best transformation for each template is simple. If a non-reference template has at least one candidate in the ball intersection area, then we are done. In case the intersection area is empty, we expand the ring width until at least one candidate enters it. Figure 1 illustrates template pruning using two reference templates and the extension needed for acquiring a match per template.

3.2 Implementation

There are a number of implementation issues that should be addressed.

First, how to choose the reference templates? One approach is to randomly select k templates. However this approach introduce randomness into the process and makes the algorithm less consistent in terms of speedup gains. Instead, we do the following. Given the set of templates, we compute the leading k principal components and find, for each component the closest template, in terms of dot-product. This generates a reference set of templates that spans the space of templates as best as possible.

Another subtle issue is that FAST-Match takes a branch-and-bound approach. As a result, the set of transformations evaluated per template differs. Therefore, we can not use the bound of Eq 4. To overcome this, we only use the first level of the branch-and-bound which is shared by all templates. Once we obtain a reduced set of transformations per template, using Eq 4 we proceed with the usual FAST-Match algorithm.

We can further prune candidate templates by taking ring width as the minimal match distance found by previous templates. The requirement of valid candidates then becomes:

$$d(t_i,q) \in [d(t_i,t_j) - d_{min}, d(t_i,t_j) + d_{min}], \quad d_{min} = min([d_1,..,d_{j-1}]) \tag{5}$$

In addition, we make use of previous templates in Fast-Match thresholds settings. Fast-Match algorithm defines a threshold for deciding which transformations are considered in next level of branch-and-bound scheme. This threshold makes sure the desired match will be reached. The suggested threshold in [8] for template t_j is $d_j^1 + o(\delta_1)$, where δ_1 is a user specified parameter that determines how tight the approximation should be. We replace it with the "shared threshold"

$$sharedThr_j = \min_{i=1..(j-1)} \left\{ d_i^1 \right\} + 2o(\delta_1), \tag{6}$$

where d_i^1 is best distance in level 1 in terms of SAD error of template t_i and δ_1 is Fast-Match accuracy parameter in first level. We call it shared threshold because it is shared by all templates. We prove that using the new threshold guarantees that the correct transformation will not be missed when shifting to the next level. See supplementary for proof. **Algorithm 1** specifies the final BIM algorithm steps.

4 Experiments

4.1 Speedup Evaluation

The first experiment compares BIM to sequential FAST-Match and a variant of FAST-Match that is combined with KD-trees. Sequential FAST-Match runs FAST-Match separately on

Algorithm 1 BIM Multi Template Matching

Input: Grayscale image I , Fast-Match accuracy parameter δ, set of templates T, number of total and reference templates N, N_{ref} respectively

Output: t_{match} best matching template, p_{match} matching patch in I, p - array of all templates matching patches in I

```
 1: $d(t_i,t_j)$ = SAD$(t_i,t_j)$, $\forall t_i,t_j \in T$
 2: $d_{min} = \infty$
 3: $A$ = Fast-Match transformation net, depends on $\delta$
 4: for i = 1:$N_{ref}$ do
 5:     $p_{t_i}, d_i = FastMatch(t_i, I, A)$
 6:     $d_{min}$ = MIN$(d_{min}, d_i)$
 7: for j = $N_{ref}+1$:$N$ do
 8:     $patches_{t_j} = \bigcap_{k=1}^{N_{ref}} [q \in A$ that fulfill (5) with $t_k]$
 9:     while $patches_{t_j}$ is empty do
10:         Extend search areas by $\varepsilon$
11:         Repeat line 8
12:     $p_{t_j}, d_j = FastMatch(t_j, I, A[patches_{t_j}])$
13:     $d_{min}$ = MIN$(d_{min}, d_j)$
14: $match$ = MIN INDEX$(d_{min})$
15: Return $t_{match}, p_{match}, p$
```

each template and reports the one with best score. The second variant stores the templates in a KD-Tree. At run time each query image patch is compared only to the k-nearest neighbors templates found by searching the tree. We use $k = 4$ as it performs well in our experiments. Since KD-Tree suffers from the curse of dimensionality, we perform dimensionality reduction to the templates and query patches using PCA.

We evaluate the speedup of BIM on 40 image pairs from the Zurich building dataset [6]. Each image pair consists of an image from which the templates are extracted and a query image in which we search for templates matches. We choose 100 templates from the templates image and seek the best matched template and its corresponding transformation to the query image.

Table 1 shows the results of the different approaches. For fair comparison, we ignore platform and implementation differences, and report the total number of affine transformations that was measured in total for 100 templates. We follow this measure in other experiments as well. We also measure the standard deviation of the number of transformations per template and find it to be low. This suggests that speed up is consistent across different templates. We present the speedup relative to sequential FAST-Match, which is the baseline approach. It is clear that BIM uses the smallest number of transformations out of the three approaches and is more stable than KD-Tree approach (i.e., it has a lower standard deviation).

4.2 Handling Deep Features

We now extend BIM to work with deep features. In particular, we represent our data using VGG [18] deep features. We take a concatenation of response to layers conv2_1 (64 features) and conv3_4 upsampled by 4 using bi-linear interpolation (256 features). The dimen-

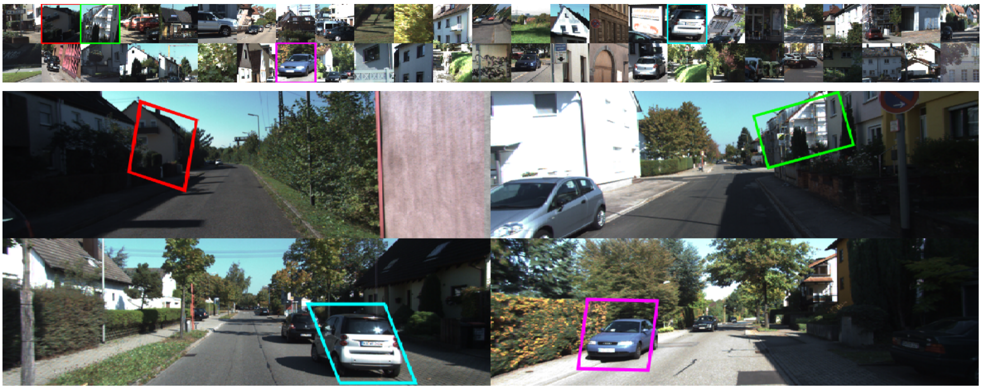
Figure 2: BIM example results on KITTI dataset: Upper image - templates cropped from different images in KITTI dataset (please zoom in to see templates). Four BIM results of affine matches are presented. BIM template and its match in query image are shown with same color. See section 4.3 for more details.

sion of features per pixel is 320 which makes it computationally expensive to work with. To solve this issue, we cluster deep features as follows. We take the features from a sparse grid of pixels (stride = 10), and compute and store the 8 leading principal components. We then project all pixel features onto this space giving us an $8D$ vector per pixel. We take the sign of each component, giving us a binary $8D$ vector per pixel. This $8D$ binary vector defines the cluster membership of every pixel. Each index cluster is set to be the average deep feature of all pixels with the same index. BIM will measure pixel distances using a pre computed 256×256 cluster-distance matrix. We show in figure 4 example of clustered deep features images using 3 and 8 principal components. 8 components image looks informative. We tested BIM with $3, 8, 10$ components and got an accuracy of $18\%, 88\%, 90\%$, respectively. Since going above 256 does not seem to contribute significantly to accuracy, we use 256 indexes in our experiments. In our experiments, clustering the features using standard k-means takes roughly 30 seconds. The proposed approach takes roughly 0.1 seconds.

4.3 Multi Affine Template Matching

KITTI dataset [11] contains real-world sequences taken from a driving platform. The dataset consists of scene image-pairs. Matching between images in each pair can be challenging due to different materials and lighting condition, non-lambertian surfaces and areas with large displacements. We use fifty scenes to test BIM as follows: We form the set of templates T

Method	Transformations No. (10^6)	STD (10^6)	Speedup
Naive FAST-Match	456.75	161.503	-
KD-Tree PCA	66.098	61.487	6.910
BIM	26.553	14.117	**17.201**

Table 1: BIM speedup experiment on Zurich bulidings datset. BIM is compared to FAST-Match and KD-Tree. BIM performs better is terms of speed up and is more stable than KD-Tree.

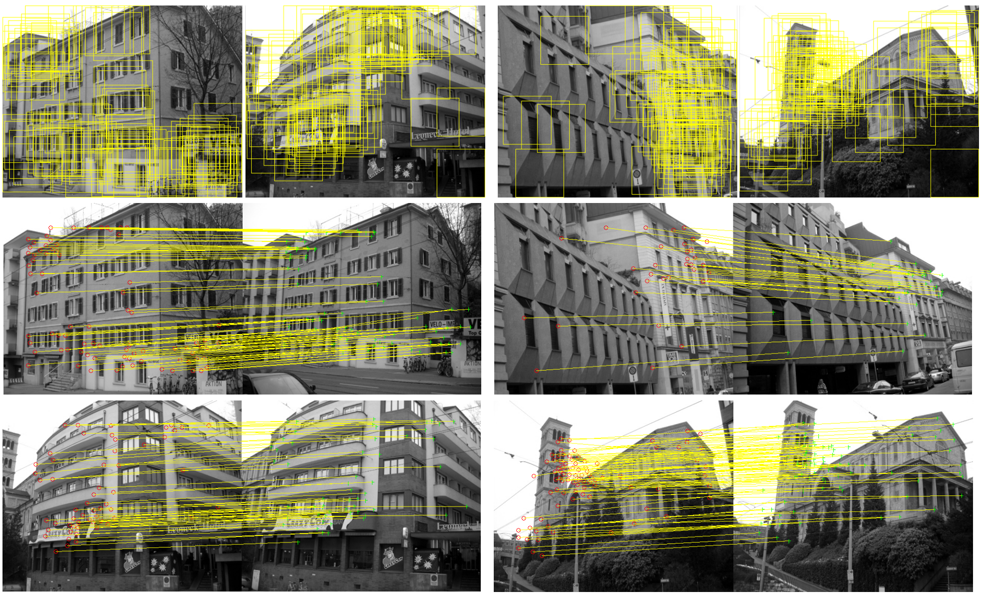

Figure 3: BIM Multi Template Matching on Zurich building dataset: Top row - source images with templates selected using the k-center algorithm (yellow rectangles). Middle and Bottom rows - images pairs with BIM matches (left - templates image, right - query image). We show only matches that pass left-right consistency check.

by choosing a single template from $I_1(k), k \in [1,50]$, where $I_1(k)$ is the first image in each pair and get $|T| = 50$. Each template is generated by choosing a parallelogram in $I_1(k)$ and warping it to a unit square.

The other image, $I_2(k)$, in each pair serves as the query image. We run $BIM(T, I_2(k))$ and get the best template and its location in the query, for all queries (50 runs). Figure 2 shows template set T and examples of query images with BIM result template matches. Results show that BIM copes well with affine transformations.

We work with clustered deep features as described in section 4.2. Unlike gray scale representation, clustering of deep features produces prototypes (i.e., cluster centers) that are far from each other, hence the distance to the nearest neighbor of each index is larger than one would like. We apply a stretching function of x^2 (where x is the original distance between prototypes) to increase the difference between closest and farthest cluster for each index and repeat the experiment. We evaluated different sizes of ring width and found that for deep features a ring width of $2 \times (0.2 \times d_i)$ produces the same accuracy as $2 \times d_i$ with better speedup so we use it with this representation. We also test BIM with gray scale

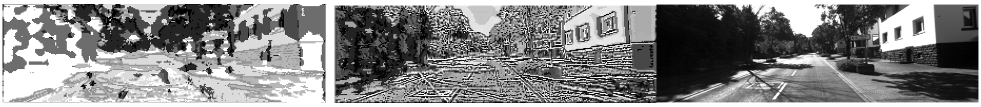

Figure 4: Deep features quantization with different clusters number: Left - 8 clusters, Middle - 256 clusters, Right - gray scale image. We use 256 clusters in BIM algorithm. See text for accuracy tests of different clusters number.

Method	Transformations No. (10^6)	STD (10^6)	Speedup	Accuracy(%)
Naive FAST-Match	75.992	1.2178	-	84
BIM Deep	18.745	4.222	4.054	**88**
BIM Deep Stretched	7.2003	1.012	**10.554**	84
BIM GL	12.889	1.263	5.8957	40 (top-5)

Table 2: BIM speedup experiment on KITTI dataset: BIM accelerates FAST-Match while keeping accuracy. BIM templates order affects area filtering and hence BIM std is larger than iterative FAST-Match (see text for details).

representation.

KITTI speedup results are reported in table 2. BIM best speedup for KITTI is achieved using stretched deep representation and reaches 10.5 with the same accuracy as the sequential (or Naive) FAST-Match. Accuracy is defined as the percentage of queries with correct template match as tested visually. Gray scale representation leads to poor results in accuracy. Deep features without stretching maintain the accuracy but do not have the same speedup as the stretched version. The algorithm was implemented in an unoptimized combination of Matlab and C code. Average time for the pre-processing step is 0.056 seconds. The average time for a reference template with 3.3 million transformations is roughly 16.5 seconds. The average time for a non-reference template is about 0.9 seconds.

4.4 Suggested Application: BIM Image to Image Matching

We suggest using BIM for image matching and show example results on Zurich buildings dataset. In each image scene pair we use one image I_1 to extract the templates and the other image I_2 as the query image. We choose templates from I_1 using the k-centers algorithm. We do not choose areas with low variance to avoid sky-like areas. We then use BIM multi matching to find matches to all templates in I_2. Since there are areas without a proper match due to the different view points between images and BIM returns a match for each template, we use left-right consistency check to eliminate bad matched templates i.e. we perform BIM from target match patches in I_2 and image I_1.If a match is found close to the original template, we declare it a good match, else we drop it. Figure 3 shows some examples. Notice that there are a few miss matches that still exist.

5 Conclusions

Template Matching usually occurs in the inner most loop of computer vision applications. Hence, accelerating it has clear and immediate impact on their performance. We proposed a method to accelerate template matching in the case of multi-template-matching.

Our algorithm treats templates as points in a high dimensional space and uses triangle inequality to quickly prune the number of templates that must be evaluated in each location. The proposed algorithm can deal with templates undergoing 2D affine transformation and works with different pixel representations including gray scale values, as well as deep features. Experiments on real data sets show a speedup of between 10 to 17, compared to a naive, sequential, algorithm of template matching.

Acknowledgment: Part of this research was supported by ISF grant 1917/2015.

References

[1] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, 1998.

[2] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 28(3), August 2009.

[3] Connelly Barnes, Eli Shechtman, Dan B Goldman, and Adam Finkelstein. The generalized PatchMatch correspondence algorithm. In *European Conference on Computer Vision*, September 2010.

[4] N. Ben-Zrihem and L. Zelnik-Manor. Riann: Approximate nearest neighbor fields in video. In *CVPR*, 2015.

[5] Michael Donoser and Horst Bischof. Efficient maximally stable extremal region (mser) tracking. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1*, CVPR '06, 2006.

[6] T. Svoboda H. Shao and L. Van Gool. Zubudzurich. buildings database for image based recognition. *Tech. Report*, 2003.

[7] Kaiming He and Jian Sun. Computing nearest-neighbor fields via propagation-assisted kd-trees. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pages 111–118, 2012.

[8] Simon Korman, Daniel Reichman, Gilad Tsur, and Shai Avidan. Fast-match: Fast affine template matching. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 1940–1947. IEEE, 2013.

[9] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2), November 2004.

[10] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence, IJCAI '81, Vancouver, BC, Canada, August 24-28, 1981*, pages 674–679, 1981.

[11] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[12] Krystian Mikolajczyk and Cordelia Schmid. Scale &amp; affine invariant interest point detectors. *Int. J. Comput. Vision*, 60(1), October 2004.

[13] Jean-Michel Morel and Guoshen Yu. Asift: A new framework for fully affine invariant image comparison. *SIAM J. Img. Sci.*, 2(2):438–469, April 2009. ISSN 1936-4954.

[14] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP 2009 - Proceedings of the Fourth International Conference on Computer Vision Theory and Applications, Lisboa, Portugal, February 5-8, 2009 - Volume 1*, pages 331–340, 2009.

[15] Igor Olonetsky and Shai Avidan. Treecann - k-d tree coherence approximate nearest neighbor algorithm. In *Computer Vision - ECCV 2012 - 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part IV*, pages 602–615, 2012.

[16] Wanli Ouyang, Federico Tombari, Stefano Mattoccia, Luigi di Stefano, and Wai-kuen Cham. Performance evaluation of full search equivalent pattern matching algorithms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(1):127–143, 2012.

[17] Chanop Silpa-Anan and Richard I. Hartley. Optimised kd-trees for fast image descriptor matching. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008), 24-26 June 2008, Anchorage, Alaska, USA*, 2008.

[18] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.