TreeCANN - k-d tree Coherence Approximate Nearest Neighbor algorithm

Igor Olonetsky and Shai Avidan

Dept. of Electrical Engineering,
Tel Aviv University
igor.olonetsky@gmail.com,avidan@eng.tau.ac.il

Abstract. TreeCANN is a fast algorithm for approximately matching all patches between two images. It does so by following the established convention of finding an initial set of matching patch candidates between the two images and then propagating good matches to neighboring patches in the image plane. TreeCANN accelerates each of these components substantially leading to an algorithm that is $\times 3$ to $\times 5$ faster than existing methods. Seed matching is achieved using a properly tuned k-d tree on a sparse grid of patches. In particular, we show that a sequence of key design decisions can make k-d trees run as fast as recently proposed state-of-the-art methods, and because of image coherency it is enough to consider only a sparse grid of patches across the image plane. We then develop a novel propagation step that is based on the integral image, which drastically reduces the computational load that is dominated by the need to repeatedly measure similarity between pairs of patches. As a by-product we give an *optimal* algorithm for **exact** matching that is based on the integral image. The proposed exact algorithm is faster than previously reported results and depends only on the size of the images and not on the size of the patches. We report results on large and varied data sets and show that TreeCANN is orders of magnitude faster than exact NN search yet produces matches that are within 1% error, compared to the exact NN search.

Key words: Approximate nearest neighbor search, patch matching.

1 Introduction

Patch-based methods are at the heart of many applications such as texture synthesis [1], image de-noising [2] and image editing [3], to name a few. These methods can often be reduced to Nearest Neighbor Field (NNF) estimation, where the goal is to find, for each patch in one image, the most similar patch in the other image.

The number of patches in an image is roughly equal to the number of pixels and can be in the millions for high-resolution images. Therefore, NNF calculation is a time consuming task that is usually performed using approximation methods. Previous approximation approaches were mostly based on hierarchical-tree

structures, such as k-d trees [4], coupled with dimensionality reduction methods (e.g. PCA). This works quite well in practice but is too slow to be used in interactive editing tools, or so it was believed.

Recently, a novel method was introduced, termed PatchMatch [5], that follows a different strategy for estimating the NNF. It achieves a substantial speedup (compared to k-d trees) by exploiting the coherency of NNF. PatchMatch works in rounds where in each round patches are assigned a random match and good matches are propagated to their neighbors in the image plane. This achieves good results even after a small number of iterations. The downside is that PatchMatch is not accurate enough in its recommended configuration (5 iterations), compared to the ground truth error, which is measured as the result of an exact NN search. Moreover, when the coherency assumption does not apply, PatchMatch might fail and lead to many mismatches, which severely degrade the mapping quality. Therefore, applications that require accurate NNF might prefer k-d trees that are slower but more accurate. The random search of PatchMatch was replaced with Locality Sensitive Hashing (LSH) in [6] that showed this to improve both accuracy and speed.

We show that a sequence of design decisions lets us accelerate the use of k-d tree for seed initialization and a novel use of the integral image (II) lets us accelerate the propagation step.

For seed initialization we use an extremely aggressive dimensionality reduction coupled with relaxed k-d tree search. Relaxed search means that we only traverse the tree from root to leaf and do not perform boundary tests to determine if the closest point might in fact be in a nearby branch of the tree. The loss of accuracy is partially compensated by the k-nn retrieval, as we retrieve the k top neighbors from the tree and revaluate all of them. These design decisions accelerate k-d tree search by an order of magnitude. Further acceleration is achieved by working only on a sparse grid of patches.

In the propagation step we make novel use of the II. Specifically, consider a region, in the source image, consisting of 3×3 overlapping patches and suppose we wish to match it to the corresponding region in the target image, based on the current assignment of the central patch. Clearly, we can compute 8 patch similarities to determine if to propagate the patch assignment from the central patch to any of its 8 neighbors. But because the patches overlap we can save considerable amount of time by calculating the difference image between the two regions and constructing an II based on it. Computing patch similarity becomes constant in the size of the patch. And since each patch participates in 9 such region-to-region comparisons we obtain, in effect, a propagation step at a fraction of the computational cost.

This propagation step leads naturally to a novel algorithm for *exact* NNF estimation over the entire image. This is done by shifting the source image across all locations of the target image, taking the difference image and computing the II on it. Patch similarity can now be computed in constant time, regardless of patch size. The overall complexity of this algorithm depends only on the size of the images and is independent of the size of patches.

We have extensively tested the TreeCANN algorithm on the recently presented database [6]. Our experiments indicate that TreeCANN outperforms PatchMatch and CSH, sometimes by up to an order of magnitude speedup, for the same accuracy levels. In addition, TreeCANN can be tuned to reach nearly ground-truth accuracy levels (less than 1% error), presenting more than $\times 100$ speedup compared to exact NN search.

2 Related Work

Patch-based sampling methods have become a popular tool for a wide variety of computer vision and graphics applications.

In practice, most of these applications rely on the process of NNF calculation, which is defined as follows: given two images (or regions) S and T, for every patch in S find the NN (in terms of appearance) in T under a certain metric (usually L_2). When trying to deal with this task in a naive, brute-force way, the computational time complexity of the algorithm is $O(mM^2)$ (where m is the patch size and M is the number of patches in the image), denying it practical use. Over the years more sophisticated techniques have been developed for exact NN matching. For example, it was shown in [7], that the m factor can be eliminated from the time complexity by exploiting the sequential overlap between patches. This brings the overall cost to $O(M^2)$. Other exact methods are mostly based on various hierarchical-tree structures.

Since exact NN methods are not fast enough, another group, known as Approximate NN algorithms, has been developed. All the hierarchical-tree based techniques, such as TSVQ [8], FLANN [9] and the most commonly used k-d tree [4] (frequently coupled with PCA dimensionality reduction technique), have been successfully used.

In parallel with the development of the tree-based techniques, several algorithms employed a different strategy, based on the coherent structure of images. Ashikhmin [10] was the first to introduce an algorithm, which used a *local propagation* technique during the texture synthesis process. This was shortly after extended by Tong et al. [11], who presented the *k-coherence*.

The local propagation methods exploited the natural structure of images and reduced memory foot print, relative to the tree-based algorithms, but failed to define a general framework and have been implemented only for the specific task of texture synthesis. However, this situation changed with the introduction of PatchMatch [5], which is also the one that inspired our work. PatchMatch and its generalized version [12] outperformed previous tree-structured techniques (specifically ANN+PCA) by up to an order of magnitude, and provided interactive performance rates for a wide range of patch-based image editing applications. The PatchMatch algorithm starts with an *initialization stage*, that performs a random assignment of every patch in the source image S to a patch in the target image T. Then it proceeds with a propagation of the good matches to the neighboring patches in the image plane. This is followed by another random assignment step, which prevents the algorithm from being stuck in local min-

ima. The propagation and the random search stages is performed in an iterative manner, and the algorithm usually converges after a small number of iterations.

Recently, a new algorithm, called Coherency Sensitive Hashing (CSH) [6], was introduced. In CSH the *random search* stage of PatchMatch was replaced by a much more efficient process, based on the LSH [13] technique. As a result, CSH is more accurate, as well as 2-3 times faster than PatchMatch.

TreeCANN share the overall structure with CSH. They both use an established ANN method to seed the propagation step, but there are several important distinctions. First, we carefully choose the k-d tree parameters and show empirically that they can bring k-d tree to perform on par with PatchMatch and CSH. We then show that working on a sparse grid is enough to establish the initial matches quickly. Finally, we proposes a novel use of the II to speed up the propagation step. Aa a result, our propagation step requires just a single iteration. TreeCANN is faster, more accurate and with suitable parameter tuning approaches the accuracy of exact NN while being orders of magnitude faster.

3 The TreeCANN Algorithm

Given source image S and target image T we define the NNF problem as a function $f : \mathbb{Z}^2 \mapsto \mathbb{Z}^2$ of values, defined over all possible patch coordinates (the locations of patches' upper-left corners). We assume that both images are of equal size, denoted M. The size of a patch edge is denoted by r, and $m = cr^2$ denotes the total number of values of a patch where c is the number of channels in an image. We take the distance metric $dist(s, t)$, between patches s and t to be the L_2 distance, where s and t are the locations of these patches in images S and T, respectively.

Following the convention of [5, 6] our algorithm consists of two main phases. An initial guess (search) step that finds an initial mapping and a propagation step that propagates good matches to neighboring patches. Because our initial step is so effective we make do with a single propagation step.

The estimation of the TreeCANN algorithm's performance is mostly accomplished by observations of the error measure, which is defined as a ratio between the results' errors , obtained by our algorithm, and the ground truth error levels, calculated using an exact NN algorithm.

3.1 ANN Search

We make a number of design decisions to accelerate the performance of k-d trees, and test them extensively, to make the best choice possible.

Aggressive Dimensionality Reduction: We evaluated a large number of target dimensions and found that aggressive dimensionality reduction provides the best trade off between accuracy and speed. Specifically, we define the target dimension, $dim(r)$, of a patch of size r to be a simple linear equation: $dim(r) = 3 + r/2$. For example, an 8×8 RGB patch will be reduced from $192 = 8 * 8 * 3$ to only $7 = 3 + 8/2$ dimensions. This is the first design decision.

We achieve dimensionality reduction by means of PCA and use a very small set of patches to compute it. In all our experiments we use $L = 100$ random patches (randomly selected from both S and T) to compute leading principal components. This is the first design decision we make. We also evaluated the use of the Walsh-Hadamard kernels, as suggested in [6] and found that they accelerate the dimensionality reduction step but hurt the k-d tree retrieval and overall give comparable results. Therefore, we only focus of the use of PCA for dimensionality reduction.

Relaxed ANN Search: k-d tree is extremely fast when the database contains good matches to the query point. In this case k-d tree simply traverse the tree from root to leaf and returns the nearest point encountered along the way. This simple procedure is complicated because of boundary problems, where the search must visit nearby branches of the tree to make sure that they do not contain a closer point to the query. To address this, we relax k-d tree to retrieve points which are within a factor of $1+e$ of the true closest point, for a certain $e \geq 0$ [4]. This technique enables a substantial reduction of the number of leaf cells that are visited, results in at least 3 fold improvement of the overall running-time, while causing only a slight degradation of the accuracy levels. In our experiments we found that $e = 3$ constitutes a good compromise between the speed and the accuracy of our algorithm. This is the second design decision of our algorithm.

k-NN retrieval: An aggressive dimensionality reduction, combined with a relaxed ANN search hurts accuracy and to combat that we retrieve k nearest neighbors and then choose the nearest patch out of the k based on measuring distance between the retrieved patches and the query patch in the original image space. This is the third design decision we make.

We show the results of our design decisions in figure 1. The figure compares several combinations of dimensionality reduction and k values. We show only the case of $r = 8$ (i.e., RGB patches of size 8×8 pixels) on image of size $M = 0.4$ mega-pixels, and compare target dimensions of $5, 7$ and 9. Results are averaged over the data set of [6]. In all cases we use a relaxed k-d tree search. The graph shows retrieval speed compared to retrieval error, where error is measured as the ratio between the retrieved NN and the ground truth NN as computed by exact NN search. As expected, increasing the target dimension reduces error but increases retrieval time. We also include the PatchMatch curve for comparison. For $k = 1$ the error obtained by a k-d tree, is much higher than the error obtained by PatchMatch, and this is also to be expected. Nevertheless, as k grows, the error levels drop sharply (for instance, for $dim(8) = 5\, or\, 7$ the error drops by a factor of 3, while the run-time increases only by a factor of 1.5). We found that the value of $k = 4$ offers a good tradeoff between speed and accuracy and, consequently, use this value in all our experiments.

Somewhat surprisingly, k-d tree alone, through a sequence of judicious design decisions, outperforms PatchMatch in many points along the accuracy-speed curve. We hope that highlighting these design decisions can benefit other applications that rely on ANN.

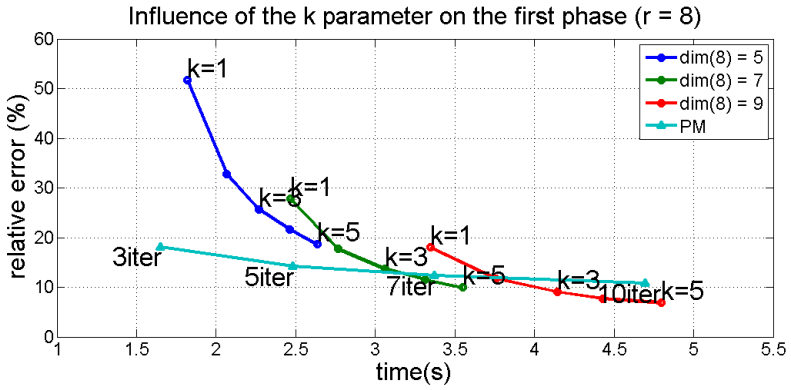

Fig. 1. The performance, obtained by our algorithm, when only its first phase is activated, for different k values. The results of the PatchMatch algorithm are added as a reference.

Working on a sparse grid: We further accelerate seed assignment, and reduce memory footprint, by working on a sparse grid of patches. Specifically, we define a sampling grid g_S, g_T on images S and T, respectively. For $g_T = 1$ we use all patches in the image T in the k-d tree search. When setting $g_T = 2$, then we use only a quarter of the patches, which leads to much faster ANN search.

Likewise, for $g_S = 1$ we use all patches in S to query the k-d tree, while for $g_S > 1$ we use less patches for query. The run-time of the TreeCANN algorithm is roughly inversely proportional to the g_S^2 parameter, as it directly influences the second phase and the k-d tree search stage.

When increasing g_S, we create *passive* (non-grid) patches, which passively obtain their final mapping from the active grid patches, without participating in the propagation process.

3.2 The Propagation Phase

Applying approximate NN search methods in conjunction with such an aggressive PCA reduction would inevitably degrade the accuracy of the results (in comparison to the earlier methods). Therefore, it is quite obvious that the results of the first phase of the algorithm are not enough and that additional processing is required in order to achieve the performance levels, which can compete with the earlier methods.

The key observation here is that evaluating patch similarity is the most time consuming part of both PatchMatch and CSH. Therefore, PatchMatch uses early termination to quickly discard bad patch matches and CSH relies on Walsh-Hadamard kernels as a fast approximation of the true Euclidean distance between patches. We, on the other hand, compute the *exact* distance between patches and use the II to speed up the process. This is a crucial ingredient of our algorithm.

Specifically, consider a region, in the source image, consisting of 3×3 overlapping patches. For example, for patches of size 8×8 pixels this will correspond to a region of size 10×10 pixels. Now let the central patch s of the region match some patch t in the target image and take a similar region around patch t.

In order to propagate good matches we wish to compute the similarity between each of the 9 patches in the source region to their corresponding patches in the target region. Naively doing so will require 9 patch similarity comparisons. But because the patches overlap we can reduce the computational cost considerably using the II.

To do so we take the difference between the source and target regions and compute its II. Now we can compute the patch similarity for every patch in that region in constant time, using the II.

This approach relies on the assumption that NNF is coherent so if patch s is mapped to patch t, then the neighbor of patch s will match, with high probability, to the corresponding neighbor of patch t.

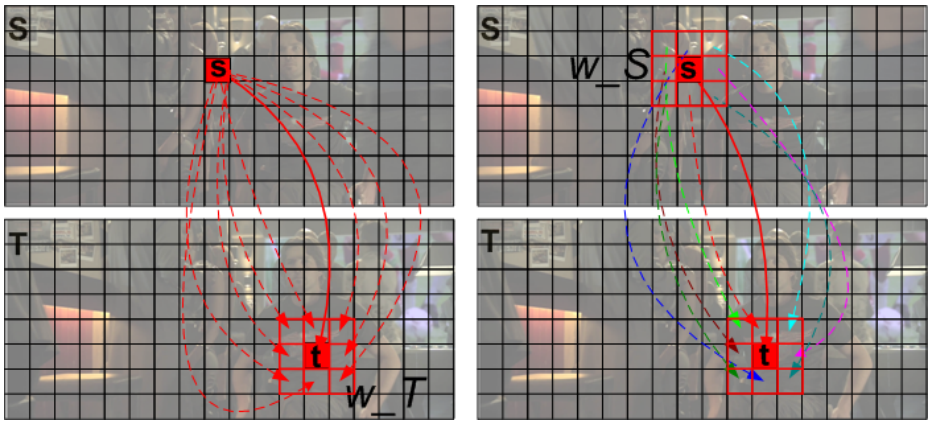

Fig. 2. *Left:* Exploiting the piece-wise constant property of the T image. All the red patches in the T image compose a window attributed to w_T parameter (in this case $w_T = 3$). *Right:* Exploiting the coherency of the S image. All the red patches in image S compose a window, attributed to the w_S parameter (in this case $w_S = 3$). *Both:* Squares on the image represent pixel (and correspond to the upper-left corner of patches). The full arrow represents an initial mapping, while the dotted ones represent all the additional distances calculation.

But there is another assumption that is often made and it is that images are piece-wise constant. This means that if patch $s \in S$ was matched to patch $t \in T$, then because image T is piece-wise constant, there is a high probability that s will also match one of the 8 neighbors of T (see Figure 2). In the experimental section we show empirically (see Table 1) that the first assignment stage, using k-d tree alone, brings about 50% of the patches in the source image to within a

distance of up to two pixels, in the image plane, from their optimal location in the target image.

This motivates us to perform the II based matching between a region centered around patch s and regions centered around each of the 8 neighbors of patch t, in addition to the matching between regions centered around t. This means that, in total, each patch in the source image is matched against $81 = 9 * 9$ patch locations.

As a concrete example, in case of patch size $r = 8$ the use of the II brings to more than $\times 5$ speed up of the propagation phase ($15mM$ instead of $81mM$ operations), and about a factor of 2 speed up for the overall algorithm.

4 An Exact NNF Algorithm

Ignoring the k-d tree initialization step and taking the II based propagation step to the extreme we derive a novel exact NNF algorithm. Specifically, we shift the source image over the target image, compute the integral difference image for each such shift and store the patch similarity score (if smaller than current minimum) of this shift for every patch in the source image. This leads to an algorithm with complexity of $O(M^2)$ instead of $O(mM^2)$. Kumar at al. [14] pointed out that finding the exact NN for all 21×21 patches between two images, that are about 800×600 pixels each, would take over 250 hours. Our exact NNF approach takes less than 20 minutes. We are also faster than the method of Xiao at al. [7] because the constants of our algorithm are smaller.

5 Experiments and Results analysis

We use the efficient ANN (Approximate-k-Nearest-Neighbors) package of Mount and Arya [15], coupled with a Matlab wrapper[1]. Our code is available online. We profiled our code and found that running time is dominated by propagation (about 40% of the time) and k-d tree search (about 30% of the time).

5.1 Choosing Database and Test-setup

We compare TreeCANN with PatchMatch and CSH on a number of data sets and report results in Figure 3. The first is the recently released database presented in [6], that contain pairs of non-consecutive video frames, taken from the same video scene (the distance between the images of one pair can vary from few to several dozen frames). The second dataset consists of the Caltech-256[2] object recognition data set, where we divide this experiment into two tests. One where both source and target images come from the same object class and another experiment where the source and target images come from different classes. Finally, we also evaluate our algorithm on the stereo[3] database.

[1] www.wisdom.weizmann.ac.il/ bagon/matlab.html
[2] http://www.vision.caltech.edu/Image_Datasets/Caltech256/
[3] http://vision.middlebury.edu/stereo/data/scenes2006/

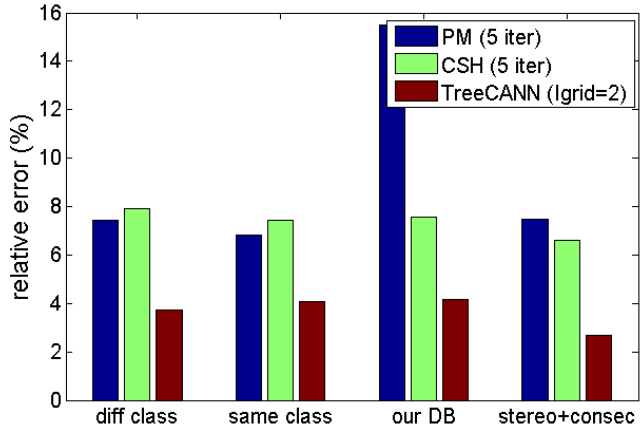

Fig. 3. The results, obtained by the PatchMatch, CSH and TreeCANN algorithms for four different types of image pairs ($r = 8, M = 1.6MB$) : *diff class* - random images from the caltech-256; *same class* - random images from the same classes in the caltech-256; *our DB* - images from database presented in [6]; *stereo+consec* - consists of consecutive frames and stereo image pairs.

There are a number of interesting observations to be made. First, we observe that PatchMatch achieved its higher error rates on the dataset of [6]. This can be explained by the fact that textured scenes are abundantly found in real-world images (such as movie frames), and often cover a large part of these images. These scenes usually contain similar repetitive patterns, which may cause the PatchMatch algorithm to be stuck in local minima for a large number of image regions, due to its mostly local nature. It is also worth noting that there is very little difference in the performance of within vs. between Caltech-256 evaluation. This suggests that variation within and between classes is quite similar.

We have performed a wide range of tests on the database of [6]. Our image samples range from 0.1MB to 1.6MB size (all the image samples were produced by the means of an under-sampling process of the same database), while the chosen patch sizes are 4, 8 and 16. For all the cases an exact NN computation was performed in order to obtain ground truth error levels. All the critical parts of our algorithm were implemented in C++, while Matlab provided the required code flow encapsulation. We use the PatchMatch and CSH code provided by the authors of the respective papers. All our experiments were executed on a single core configuration on a i5 750 (2.66 GHz) machine with 4GB of RAM memory.

5.2 Sparse Grid Acceleration

Our experiments show that we can significantly compensate for the performance degradation when using $g_S > 1$ with larger regions, denoted w_S, and set $w_S =$

$2g_S + 1$ in all our experiments. This ensures that w_S will be just the right size to cover all the eight neighboring grid-patches, relative to one particular grid-patch, but not more than that, in order to avoid unnecessary computations.

The grid approach also favorably affects the overall memory consumption of the algorithm, as it equals to $O(\frac{dim(r)}{g_I^2}M)$ (since equivalent values of g_S and g_T are used in all our experiments, we substitute them with the g_I parameter).

5.3 Performance comparison

The main objective of our experiments was to perform a reliable comparison between the PatchMatch, CSH and TreeCANN algorithms on various set ups. Unlike previous methods, which presented an absolute error (an averaged L_2 distance between the matching patches of a source and a target image), we produce our graphs with a relative (to the ground-truth calculation) error, which allows a true understanding of the algorithm's accuracy.

The results of our test runs on the dataset of [6] are shown in Figure 4. It shows, for example, that at the error level reached by PatchMatch after 5 iterations (5 is the number of iteration that was suggested in [5] as the most cost-efficient point in the average case), our algorithm is five times faster (on average) than PatchMatch and about two to three times faster than CSH. If we examine a more specific set-up, like $[r = 4, M = 1.6MB]$, the speed up is almost an order of magnitude. More importantly, it appears that the biggest improvement occurs in the most challenging (from the runtime perspective) case, i.e. in the large image sizes. In this scenario, PatchMatch and CSH could not provide reasonable run-times (and low error levels) for interactive applications.

In general, we can determine that while the minimal error level, which can be achieved by the PatchMatch and the CSH algorithms, degrades as the image size increases, our algorithm maintains almost identical accuracy results. Furthermore, when comparing the runtime performances for lower error levels (for example, $g_I = 3$), the gap between the algorithms increases dramatically. Finally, PatchMatch and CSH can not compete in the range of the lowest error rates ($g_I = 2\, or\, 1$), obtained by TreeCANN algorithm.

In addition, TreeCANN approaches the absolute ground-truth, which was previously accomplished only by exceedingly slow LSH and k-d tree algorithms. For small patch sizes we are only 3% less accurate than the ground-truth, and the accuracy improves for larger patch sizes. And, as already noted, these performance levels can be reached in a very reasonable time (less than 10 PatchMatch iterations). Moreover, if the error rates are all that matters, one can slightly tune several parameters of the algorithm, so that the distance to the ground-truth will be reduced even further. For instance, changing the dimensionality reduction function $dim(r)$, and lowering the e parameter to $e = 2$, will result in additional reduction of the already very low error rates, reaching accuracy levels lower than 1% for all the patch sizes as shown if Figure 5.

In table 1 we explore another characteristic of the TreeCANN algorithm and show the average mapping distance between the TreeCANN algorithm and the

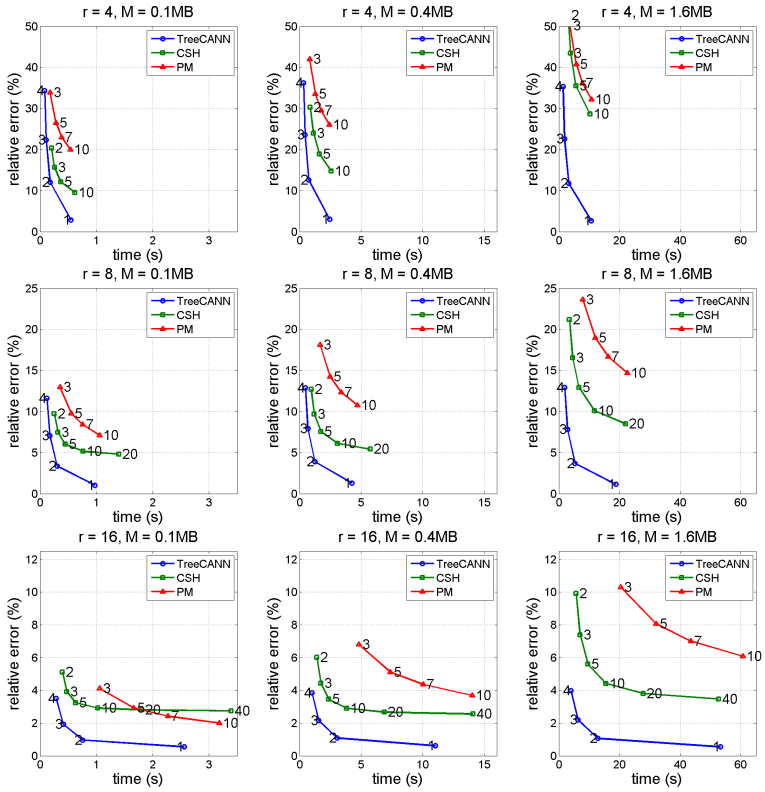

Fig. 4. The relative error performance comparison between the PatchMatch, CSH and the TreeCANN algorithms, versus the absolute run-time for different image sizes and patch sizes. The numbers on the TreeCANN line indicate the values of the g_i parameter (i.e., how sparse is the grid that TreeCANN operate on), while those on the PatchMatch and the CSH lines represent the number of iterations.

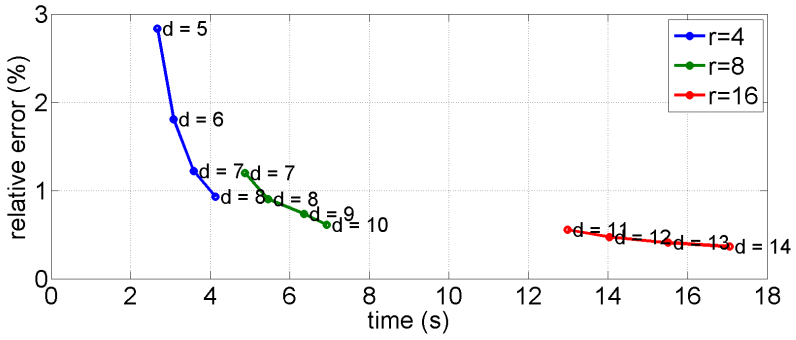

Fig. 5. Further reduction of the minimal error levels that can be obtained by TreeCANN algorithm ($M = 0.4MB$).

Table 1. The mapping distance error results for $r = 8$, $M = 0.4MB$ and $g_I = 1$. After the k-d tree search, roughly 50% of the patches are matched to patches that are at most two pixels away from the Ground Truth (GT) location. After the propagation step this number grows to almost 83%.

dist to GT →	0	1	2	>2
Only k-d tree	28.5%	17.7%	4.0%	49.8%
k-d tree+prop.	81.6%	0.9%	0.4%	17.1%

Ground Truth. That is, we measure the distance, in the image plane, between the mapping suggested by TreeCANN and the mapping found by an exact NN search. As can be seen, already after the k-d tree phase more than 50% of the patches obtain either their optimal matching or one in a very close proximity to the optimal location ($dist \leq 2$ pixels). Furthermore, after the Propagation phase the TreeCANN algorithm finds the optimal mapping for almost 82% of the patches (Figure 6 depicts these results visually for a specific pair of images). Additionally, for a particular image pair we show error images (in inverse colors) which represent the (scaled) difference between the original image S and the reconstructed images of the three algorithms. If examined closely it becomes evident that smaller mapping errors eventually translate to smaller reconstruction errors, as TreeCANN algorithm presents the best results.

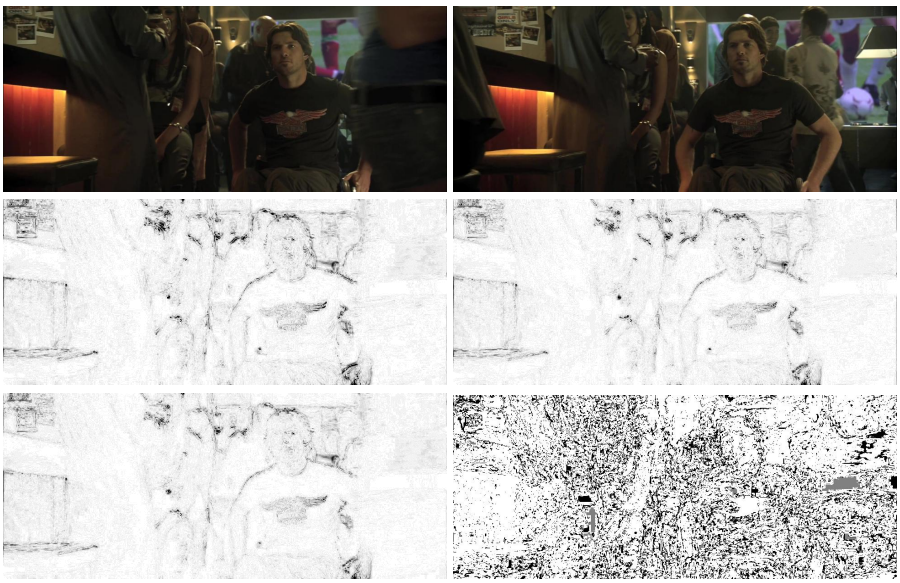

Fig. 6. From top-left to bottom-right: Image S, Image T, PM error, CSH error, TreeCANN error and accuracy map. Error images shown in inverse color (brighter color represents smaller error). The white pixels in the accuracy map indicate that the optimal mapping was found for that specific patch.

5.4 Exact NNF performance

We have tested the performance of our exact NNF against those of [7] who performed an extensive compassion between various exact NN methods. As can be seen in Figure 7, our algorithm, which is not software optimized or hardware accelerated, is more than four times faster compared to the *N column CPU* method of [7], and is on par with their *N column GPU* approach.

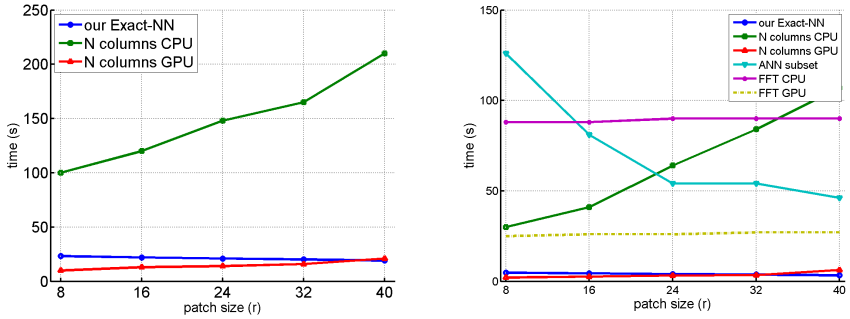

Fig. 7. Exact NNF Comparing our method to that of [7]. *Left:* The results of [7] (fig. 3a) with our results (denoted our Exact-NN). (the size of the S image is 256x256, and the size of the T image is 278x278). *Right:* The results of the various methods reported in [7] (fig. 9a), with our results overlaid for comparison (the size of the S image is 256x256, and the size of the T image is 128x128).

6 Discussions and Future Work

TreeCANN is the fastest algorithm for NNF estimation reported to date. It does so by properly combining existing techniques at their optimal cost-effective point. We show that k-d tree can perform as fast as other methods simply by properly tuning its parameters. And the novel use of the II makes it possible to match multiple patches at once, leading to large improvement in the speed of the propagation step. Taken to the extreme the integral image can be used in an optimal algorithm for *exact* NNF that is faster than previously reported results.

A wide group of applications, such as object detection, de-noising, and symmetry detection, require the NN patch matching algorithm, which finds several closest matches rather than a single match. Thus, a simple functionality extension of our algorithm would be a detection of k nearest neighbors. With respect to the TreeCANN's performance, one of the obvious and probably the most significant speedup improvements of our algorithm would be an implementation of its multi-threaded and GPU versions.

Acknowledgments. This work was supported in part by an Israel Science Foundation grant 1556/10.

References

1. Efros, A. A., Leung, T. K.: Texture synthesis by non-parametric sampling. In ICCV, vol. 2, pp. 1033-1038. (1999)
2. Buades, A., Coll, B., Morel ,J.: A non-local algorithm for image denoising. In CVPR, vol. 2, pp. 60-65. (2005)
3. Simakov, D., Caspi, Y., Shechtman, E., Iran, M.: Summarizing visual data using bidirectional similarity. In CVPR, pp 1-8. (2008)
4. Arya, S., Mount, D., Netanyahu, N., Silverman, R., Wu, A.: An optimal algorithm for approximate nearest neighbor searching. In ACM, vol. 45, pp. 891-923. (1998)
5. Barnes, C., Shechtman, E., Finkelstein, A., Goldman, D.: PatchMatch: a randomized correspondence algorithm for structural image editing. ACM Transactions on Graphics (Proc. SIGGRAPH), vol. 28 (2009)
6. Korman, S., Avidan , S.: Coherency sensitive hashing. In ICCV, pp. 1607-1614. (2011)
7. Xiao, C., Liu, M., Nie, Y., Dong, Z.: Fast exact nearest patch matching for patch-based image editing and processing. In IEEE Trans. Vis. Comput. Graph., vol. 17, pp. 1122-1134. (2011)
8. Wei, L.-Y., Levoy, M.: Fast texture synthesis using tree-structured vector quantization. In SIGGRAPH, pp. 479-488. (2000)
9. Muja, M., Lowe, D. G.: Fast approximate nearest neighbors with automatic algorithm configuration. In VISSAPP, pp. 331-340. (2009)
10. Ashikhmin, M.: Synthesizing natural textures. In Proc. symposium on Interactive 3D graphics, pp. 217-226. (2001)
11. Tong, X., Zhang, J., Liu, L., Wang, X., Guo, B., Shum. H.: Synthesis of bidirectional texture functions on arbitrary surfaces. In ACM Trans. on Graphics, vol. 21, pp. 665-672. (2002)
12. Barnes, C., Shechtman, E., Goldman, D. B., Finkelstein, A.: The generalized PatchMatch correspondence algorithm. In ECCV, vol. 6313, pp. 29-43. (2010)
13. Indyk , P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In Symposium on Theory of Computing, pp. 604-613. (1998)
14. Kumar, N., Zhang, L., Nayar, S.: What is a good nearest neighbors algorithm for finding similar patches in images? In ECCV, pp. 364-378. (2008)
15. Mount, D. M., Arya, S.: Ann: A library for approximate nearest neighbor searching. `http://www.cs.umd.edu/~mount/ANN/` (2006)