EFFICIENTLY SELECTING SPATIALLY DISTRIBUTED KEYPOINTS FOR VISUAL TRACKING

Steffen Gauglitz *Luca Foschini* *Matthew Turk* *Tobias Höllerer*

Department of Computer Science, University of California, Santa Barbara
{sgauglitz, foschini, mturk, holl}@cs.ucsb.edu

ABSTRACT

We describe an algorithm dubbed Suppression via Disk Covering (SDC) to efficiently select a set of strong, spatially distributed keypoints, and we show that selecting keypoint in this way significantly improves visual tracking. We also describe two efficient implementation schemes for the popular Adaptive Non-Maximal Suppression algorithm, and show empirically that SDC is significantly faster while providing the same improvements with respect to tracking robustness. In our particular application, using SDC to filter the output of an inexpensive (but, by itself, less reliable) keypoint detector (FAST) results in higher tracking robustness at significantly lower total cost than using a computationally more expensive detector.

1. INTRODUCTION

Keypoint-based solutions have successfully been applied to a variety of computer vision problems, including image stitching (e.g., [1]), six degree-of-freedom tracking of known targets (e.g., [2]) and in unknown environments (e.g., [3]). Several algorithms have been proposed for different steps in the processing chain, such as keypoint detection, description and matching. However, relatively little attention has been devoted to *keypoint selection* (from a set of detected keypoints) and *spatial distribution* of keypoints. In this paper, we show that spatially distributing keypoints can significantly improve visual tracking and we propose an algorithm that computes such a distribution much more efficiently than existing algorithms.

Although we demonstrate the usefulness of our algorithm for visual tracking, it is not specific to this application and can be employed in any process in which well-distributed features are needed.

To accurately and robustly track an object, it is desirable to rely on as many features as possible, as more outliers can be tolerated and the pose estimation can be based on more independent measurements. However, real-time constraints pose strict limits on the number of features that can be processed in each frame (especially for feature matching or filtering, where the complexity is superlinear). Thus, it is desirable to have as well-chosen features and as fine-grained control over their quantity as possible. This motivates the following work of designing a keypoint selection filter to efficiently select a subset of k well-distributed, strong features.

2. RELATED WORK

Compared to the vast bodies of work on keypoint detection, description, matching strategies and applications, relatively few papers explicitly address the problem of keypoint selection and spatial distribution, although there is evidence that spatial feature distribution is important (cf. [4] and [1, 5, 6]). The most notable exception is Brown et al. [1], who proposed the "Adaptive Non-Maximal Suppression" filter (ANMS). This algorithm has been adopted for various keypoint-based applications (cf. papers citing [1]). Since our algorithm is inspired by ANMS, we will discuss ANMS in detail in Section 3.

Behrens and Röllinger [5] compared filtering by strength, ANMS, and a k-d tree-based method by Cheng et al. [6], for image stitching and found that both k-d tree and ANMS significantly reduce the registration error. While ANMS produces a more uniform spatial distribution and a higher number of inliers, they recommend using the k-d tree approach for real-time applications due to the complexity of ANMS — this is precisely the problem for which our algorithm provides a remedy.

The keypoint selection algorithm by Nguyen and Andersen [7] aims at clustering keypoints in presumably salient regions instead of spatially distributing them and hence addresses other applications.

3. REVIEW OF ANMS

Given a set of keypoints P, Brown et al. [1] define the (minimum) suppression radius r_i for a keypoint $p_i \in P$ as the distance to the closest significantly stronger keypoint:

$$r_i = \min_j ||p_i - p_j|| \text{ where } f(p_i) < c_{\text{robust}} \cdot f(p_j),\ p_j \in P \quad (1)$$

where the factor $c_{\text{robust}} \leq 1$ is introduced to make the algorithm more robust if the same set of features is sought to be re-detected despite image noise. ANMS then retains the k keypoints with the largest suppression radii.

This algorithm can easily be implemented by processing keypoints one by one in decreasing order of strength; setting r_i to the minimum distance to all previously processed points (for the first point, $r_1 = \infty$); and finally returning the top-k keypoints by r_i.

ANMS is elegant, very easy to implement and (as we will show below) can significantly increase the robustness of tracking. Unfortunately, naïve implementations require $\Theta(n^2)$ running time, where $n = |P|$ is the number of input keypoints. Due to the simplicity of the operations involved, an algorithm with quadratic runtime is not impractical in general and it may be well suited for applications such as image stitching (cf. [1]). However, for many detectors the number of extracted keypoints n varies greatly for different images, and adjusting the respective thresholds to make the number manageable in one condition can lead to loss of crucial keypoints in other conditions. Hence an $\Theta(n^2)$ algorithm is not a good fit for real-time applications, where the processing time per frame needs to be almost constant.

4. SUBQUADRATIC ALGORITHMS FOR ANMS

In this section, we introduce two complementary schemes that implement ANMS in subquadratic runtime.

Scheme A: iteratively add points. The keypoints in P are inserted into a nearest neighbor data structure NN one by one, in decreasing order of strength. Upon insertion of p_i, all and only the keypoints stronger than p_i are present in NN. Therefore, r_i is the distance to the nearest neighbor of p_i in NN (returned by NN.query(p_i)). At the end, the k points with largest radius are returned as the output keypoint set. An outline of this scheme is given in Alg. 1.[1] Its runtime is n times the time needed to query and update NN.

Algorithm 1: Scheme A – iteratively adding points

```
sort P by strength, let p_1, p_2, ... denote the points in that order
NN.insert(p_1)
result = [(∞, p_1)]        //(radius,keypoint)
foreach p_i ∈ P do
    r_i = NN.query(p_i)
    result.pushback((r_i, p_i))
    NN.insert(p_i)
sort result by radius, return first k entries in result
```

The naïve implementation of ANMS (see Section 3) falls into this category, where NN is simply the list of previous keypoints, and the nearest neighbor is obtained by a linear scan of NN, resulting in a runtime of $O(n^2)$. This complexity can be significantly reduced by using more sophisticated data structures for NN. In particular, we implemented this strategy using Chan [8]'s Approximate Nearest Neighbor algorithm, with (as suggested by Chan) Randomized Search Trees [9] as the underlying data structure to allow queries and inserts in $O(\log n)$.[2] We report on this implementation dubbed F(ast)-ANMS in Section 6: While it scales better and is significantly faster than the naïve implementation for very large n, it is slower for many realistic values of n due to the overhead of managing the more complex NN data structure. Also, although Chan [8]'s NN algorithm is arguably the easiest-to-implement among those with $O(\log n)$ query and update time, the code complexity is considerably larger than for the naïve implementation.

Scheme B: guess radius and remove points. Scheme B stems from the observation that if the k-th largest suppression radius r_k for P was known beforehand, to find the k keypoints with the largest suppression radii one could simply filter out all keypoints within r_k of a stronger keypoint.

Since r_k is not known a priori, one can perform such filtering with several *guesses* r for r_k, until an $\hat{r}_k$ is found for which exactly k keypoints are retained.[3] As the number of surviving keypoints is monotonically decreasing with r, binary search can be used to find such r. As r cannot exceed the image width w, at most $\log w$ iterations are needed.

Alg. 2 illustrates the scheme B. For each guess r, all p_i's $\in P$ are processed in decreasing order of strength. For each p_i all points P' within distance r from p_i and not yet suppressed are retrieved querying a data structure NN_r, initially built on P. All points from P' are then removed from NN_r before the next iteration. If p_i is itself present in the returned list P', meaning that no stronger points suppressed it at radius r, then p_i is added to the output set.

Algorithm 2: Scheme B: guess radius and remove points

```
sort P by strength, let p_1, p_2, ... denote the points in that order
while binary search for suppression radius r do
    build NN_r on P
    result = ∅
    foreach p_i ∈ P do
        P' = NN_r.query(p_i)
        if p_i ∈ P' then result = result ∪ p_i
        NN_r.remove(P')
    if |result| = k then return result
```

The fact that a point is returned by NN_r only once per radius guess combined with a $O(\log n)$-query/delete data structure similar to [8] establishes the global asymptotic runtime of $O(n \log n)$ per radius guess. However, since $\log w$ guesses might be needed in the worst case, scheme A proves superior to scheme B with respect to runtime if an *exact* solution of Eq. (1) is sought. On the other hand, scheme B allows several degrees of flexibility that make fast approximate solutions possible, paving the way to our proposed algorithm.

5. SUPPRESSION VIA DISK COVERING

Our proposed algorithm, dubbed Suppression via Disk Covering (SDC), bears similarities with scheme B, although it departs from it in two main aspects.

Firstly, we simulate an *approximate* r-Near Neighbor query such that all points with distance less than $(1-\varepsilon_r)\cdot r$ from q are retrieved and no point with distance more than $(1+\varepsilon_r)\cdot r$ is retrieved (points in between may or may not be retrieved). We implement this by superimposing a regular grid G_r onto the keypoints and approximating the Euclidean distance $||p_i - p_j||$ between points p_i and p_j by the distance between the centers of the cells into which they fall. Given ε_r, the width of each cell in the grid is therefore fixed to be $c = \varepsilon_r \cdot r/\sqrt{2}$. Secondly, we eliminate candidates in such a way that at any given time, a keypoint that has not been included in the output cannot cause the suppression of any other keypoint of lesser strength in the future.

The outline of our algorithm is presented in Alg. 3, the main steps are illustrated in Fig. 1. The main idea is that the "covering" of a cell implies the suppression of all points contained in it, so that for each keypoint, we simply have to check whether or not its cell is already covered, and either remove it or cover the surrounding cells up to a radius r. This yields a very easy-to-implement $O(n)$ algorithm per radius guess. Note that the rather expensive cell-cover operation has to be performed at most k times, as the iteration can be aborted if the output set grows larger than k.

Algorithm 3: Suppression via Disk Covering (SDC)

```
sort P by strength, let p_1, p_2, ... denote the points in that order
while binary search for suppression radius r do
    set resolution of grid G_r to c = ε_r · r/√2
    uncover all cells of G_r
    result=∅
    foreach p_i ∈ P do
        if cell of p_i is not covered then
            result = result ∪ p_i
            cover all cells h of G_r with ||h − cell of p_i|| < r
    if |result| = k then return result
```

[1] Although omitted in this outline, Alg. 1 is easily adapted to deal with $c_{\text{robust}} < 1$: It is sufficient to decouple queries and updates to NN, such that when querying for p_i, NN contains all and only the keypoints with strength greater than $f(p_i)/c_{\text{robust}}$.

[2] Note that we violate two assumptions in Chan's theoretical analysis: we cannot claim that our points are statistically independent, and we use $\varepsilon = 0$, while the analysis is valid only for $\varepsilon > 0$. However, tests with random points did not reveal differences in runtime, and varying ε from 0 to 1 decreased query time by less than 10% (10^6 queries on a database with 10^6 points).

[3] $\hat{r}_k$ need not equal r_k, but the existence of r_k guarantees that such an $\hat{r}_k$ exists.

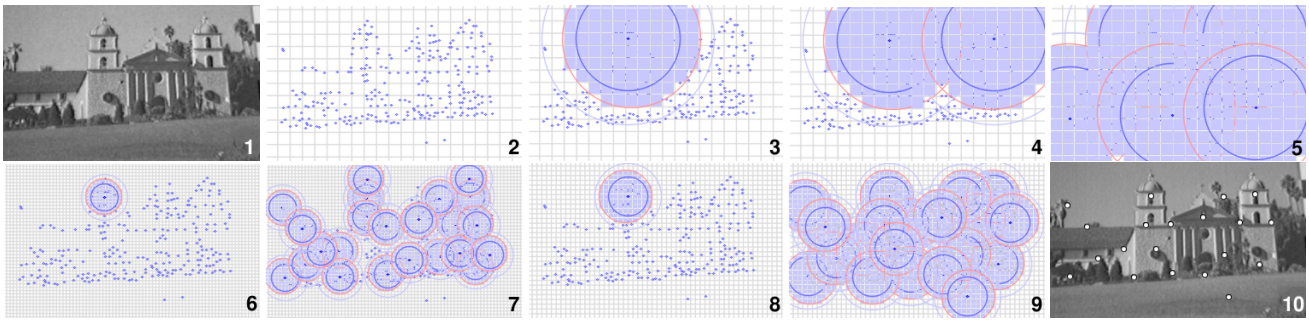

Fig. 1. Step-by-step illustration of our algorithm. **1.** Input image for which a set of $k = 20$ strong, well-distributed keypoints is sought. **2.** Keypoints found by detector (here: Shi-Tomasi). The first guess for r results in the indicated grid resolution. **3.** The first (strongest) keypoint is selected and all cells within the approximated radius r are covered. The red circle indicates the suppression radius around the cell center, the inner and outer blue circles denote the $r \cdot (1 \pm \varepsilon_r)$ rings around the *actual* keypoint location: no point within the inner circle may remain, no point outside the outer circle may be removed. The executed cover easily fulfills this. **4.** Strongest uncovered point is selected, surrounding cells covered. **5.** Finally, with this radius, five keypoints have been selected. This is below the desired k, so a new iteration (**6.**) is started with a smaller r. **7.** More than k points are selected and still uncovered points left: r is too small, the iteration can be aborted and the next iteration (**8.**) started. **9.** Finally, with this r, exactly k keypoints have been selected and are returned as result (overlaid onto the input image in **10**). In reality, five iterations were needed for this particular input set, of which we illustrate iterations 2, 4 and 5.

The runtime of Alg. 3 depends on *the number of cells per radius* $r/c = \sqrt{2}/\varepsilon_r$, as $(r/c)^2 = O(1/\varepsilon_r^2)$ cells have to be covered for each surviving keypoint. The total complexity therefore becomes:

$$O(n \log n) + \log w \cdot O(n + k/\varepsilon_r^2) \tag{2}$$

We found SDC to be robust to variations of ε_r, and chose $\varepsilon_r = 0.25$, i.e., a circle is approximated by 11x11 squares, for our experiments.

A main caveat for SDC is that due to the approximated near neighbor and the greedy approach used to eliminate candidates, the number of keypoints in the output set k is not guaranteed to be monotonically decreasing with the radius r. This means that a binary search is not guaranteed to find a radius $\hat{r}_k$ for which exactly k keypoints are retained. However, for all practical cases analyzed in Section 6, the function $k(r)$ turns out to be "almost" monotonic, so that a binary search strategy can still effectively guide the choice of a radius. Additionally, we observe that finding an $\hat{r}_k$ exactly is not required: any r *close* to r_k provides a well distributed set. Thus, we allow the binary search strategy to terminate as soon as the number of output keypoints is within $[k; (1+\varepsilon_k) \cdot k]$ and then return the top-k by strength among the surviving points. By increasing ε_k, one may thus balance the importance of distribution vs. keypoint strength. We found that with $\varepsilon_k = 0.1$, the number of radius guesses rarely exceeds 10, with no significant effect on performance.

We emphasize that with ε_r,$\varepsilon_k > 0$ and our elimination strategy, SDC does not implement Eq. (1) exactly. However, as we will demonstrate in Section 6, this does not hamper the quality of the distribution, while departing from Eq. (1) allows for a more efficient implementation[4].

The relatively expensive cell cover operation can be substantially sped up by using a single bit to store the state of each cell of G_r. This enables using bitwise OR operations to "cover" contiguous patches at once with precomputed bitmasks that implement the cover to be applied at a given bit-offset position[5].

[4]In practice, the results of ANMS and SDC turn out to be quite similar: For example, for the image shown in Fig. 1, the keypoints selected on three different octaves by ANMS and SDC have 74% overlap.

[5]Note that the bitmasks only depend on the the number of cells per radius $r/c = \sqrt{2}/\varepsilon_r$, *not* the current radius r. That means that they do not change between iterations (or even different images) and can be precomputed.

6. EVALUATION

To show that spatial distribution of keypoints improves robustness of tracking, we embedded the discussed algorithms into a tracking solution and measured the runtime of the selection algorithm as well as the fraction of correctly tracked frames.

Evaluation setup. We used a dataset of videos with ground truth described in detail in [10], featuring six different planar textures in 16 different motion paths each. Here, we used the motion paths "unconstrained," which contain unconstrained camera motion around the object including rotations, zoom, and fast motions (3000 frames total), and, for Fig. 3 bottom, the "motion blur" sequences, which contain camera panning with nine accurately controlled speeds for each of the six textures. Since we do not provide a means for tracking recovery, we split the "unconstrained" videos into sequences of 20 frames each for a total of 150 sequences. We define a frame to be successfully tracked if the average distance between the estimated positions of the target's four corners and their respective ground truth positions is less than 5 pixels.

Tracking algorithm. As a tracking algorithm, we use a multi-level, active search patch tracker with NCC-based matching: From the first frame (the template), a three-octave image pyramid is created and $k = 20$ keypoints on each octave are selected by applying a keypoint detector and then the respective keypoint selection algorithm. Each incoming frame gets warped according to the previous frame's estimated homography and NCC-based matching of an 8x8 template is used to establish each keypoint's new position. This is done for the top-most pyramid level first, then the homography is re-estimated using RANSAC and used to project the features into the next level. This ensures robustness to relatively large intra-frame movements despite a small template matching area. For ANMS, we use $c_{\text{robust}} = 1$, since we do not seek to re-detect the same set of points in the next frame.

This algorithm is similar to the patch tracking solutions described by Klein and Murray [3] and Wagner et al. [2] and was found to perform very favorably in terms of speed/robustness trade-off compared to several image alignment- and other keypoint-based algorithms in our own studies. However, the tracker is exactly the

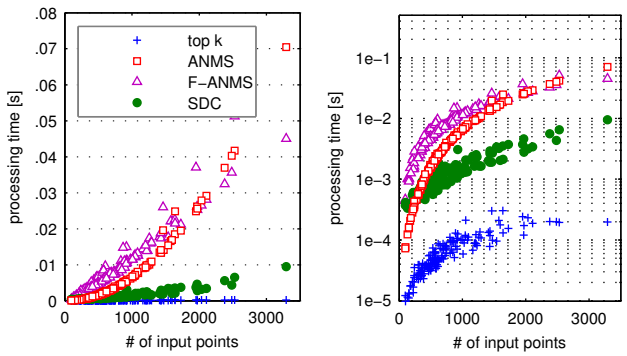

Fig. 2. Processing time vs. number of input points n for different keypoint selection filters. Data in both plots is the same, the only difference is the scale of the y axis (linear vs. logarithmic).

same in all conditions — the only difference is the algorithm to select the input keypoints — and therefore, the trends shown in the results should not be influenced by the inner workings of the tracker.

Processing time. As Fig. 2 shows, our filter is significantly faster than ANMS for all but very low n.

Evidence that spatial keypoint distribution is important. Fig. 3 shows that spatially distributing the keypoints improves tracking robustness: While the quantitative improvement varies with the condition and the detector used, the three spatial distribution filters perform very similarly and significantly better than selecting the strongest k keypoints. The improvement is much larger for FAST (Fig. 3 left column), where the strength computed by the detector is rather unreliable, but tracking with Shi-Tomasi is also improved especially for input images in which the "natural" features are not well distributed (texture **su**nset in Fig. 3 top), and for motion blur (Fig. 3 bottom right). Augmented with SDC, the tracking solution using FAST outperforms the one using Shi-Tomasi for most conditions.

7. CONCLUSIONS

We presented SDC, an algorithm that efficiently selects a subset of well-distributed, strong points among all keypoints detected in an image. We demonstrated that our algorithm is significantly faster even than efficient implementations of than ANMS. Compared to selecting the k strongest keypoints, SDC offers improvements in tracking robustness comparable to ANMS, for a variety of conditions.

In our application, using SDC combined with the inexpensive (but, by itself, less reliable) FAST detector leads to better average tracking robustness than using the more expensive Shi-Tomasi detector, while the time for the additional selection step is insignificant compared to the savings from using the cheaper detector.

8. REFERENCES

[1] M. Brown, R. Szeliski, and S. Winder, "Multi-image matching using multi-scale oriented patches," *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2005, vol. 1, pp. 510–517.

[2] D. Wagner, D. Schmalstieg, and H. Bischof, "Multiple target detection and tracking with guaranteed framerates on mobile phones," *Proc. 8th IEEE Intl. Symposium on Mixed and Augmented Reality (ISMAR'09)*, Oct. 2009, pp. 57–64.

[3] G. Klein and D. Murray, "Parallel tracking and mapping on a camera phone," *Proc. 8th IEEE Intl. Symposium on Mixed and Augmented Reality (ISMAR'09)*, Oct. 2009, pp. 83–86.

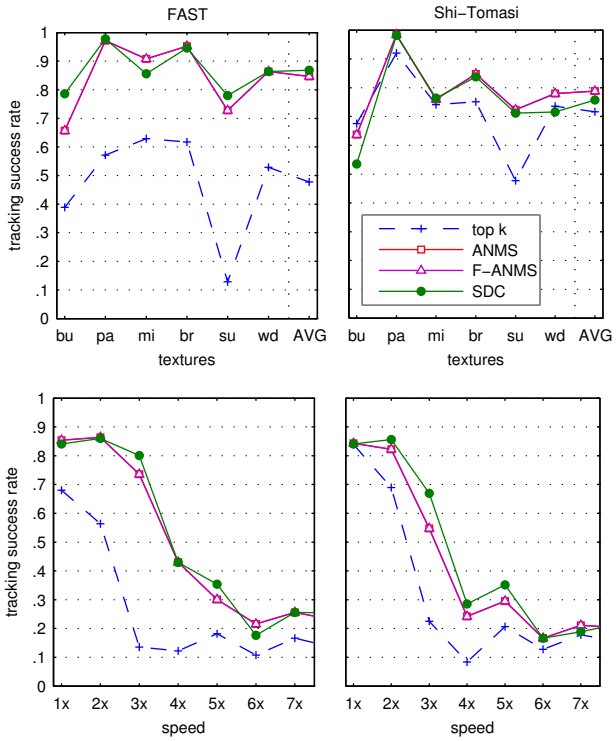

Fig. 3. Tracking success rate using a multi-level patch tracker limited to 20 keypoints per level selected by the respective filter. Top row: random camera movement for different textures (AVG indicates average of all frames), bottom row: different levels of motion blur (camera panning speed); left column: using FAST [11] as keypoint detector, right column: using Shi-Tomasi [12] as detector. As to be expected, the results for ANMS and F-ANMS are identical.

[4] L. Gruber, S. Zollmann, D. Wagner, D. Schmalstieg, and T. Höllerer, "Optimization of target objects for natural feature tracking," *Proc. 20th Intl. Conference on Pattern Recognition (ICPR)*, Istanbul, August 2010, pp. 3607–3610.

[5] A. Behrens and H. Röllinger, "Analysis of feature point distributions for fast image mosaicking algorithms," *Acta Polytechnica J of Advanced Engineering*, vol. 50, no. 4, pp. 12–18, 2010.

[6] Z. Cheng, D. Devarajan, and R. J. Radke, "Determining vision graphs for distributed camera networks using feature digests," *EURASIP J on Advances in Signal Processing*, January 2007.

[7] G. Nguyen and H. Andersen, "Context-based adaptive filtering of interest points in image retrieval," *Proc. 9th Intl. Conf. on Intelligent Systems Design and Applications*, 2009, pp. 529–534.

[8] T. M. Chan, "A minimalist's implementation of an approximate nearest neighbor algorithm in fixed dimensions," Tech. Rep., School of Computer Science, Univ. of Waterloo, May 2006.

[9] R. Seidel and C. R. Aragon, "Randomized search trees," *Algorithmica*, vol. 16, no. 4/5, pp. 464–497, 1996.

[10] S. Gauglitz, T. Höllerer, and M. Turk, "Evaluation of interest point detectors and feature descriptors for visual tracking," *Intl. Journal of Computer Vision*, 2011. doi:10.1007/s11263-011-0431-5

[11] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," *Proc. IEEE European Conference on Computer Vision*, May 2006, vol. 1, pp. 430–443.

[12] J. Shi and C. Tomasi, "Good features to track," *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 1994, pp. 593–600.