AnySURF: Flexible Local Features Computation

Eran Sadeh-Or and Gal A. Kaminka
Computer Science Department
Bar Ilan University, Israel

Abstract. Many vision-based tasks for autonomous robotics are based on feature matching algorithm, finding point correspondences between two images. Unfortunately, existing algorithms for such tasks require significant computational resources and are designed under the assumption that they will run to completion and only then return a complete result. Since partial results—a subset of all features in the image—are often sufficient, we propose in this paper a computationally-flexible algorithm, where results monotonically increase in quality, given additional computation time. The proposed algorithm, coined AnySURF (Anytime SURF), is based on the SURF scale- and rotation-invariant interest point detector and descriptor. We achieve flexibility by re-designing several major steps, mainly the feature search process, allowing results with increasing quality to be accumulated.
We contrast different design choices for AnySURF and evaluate the use of AnySURF in a series of experiments. Results are promising, and show the potential for dynamic anytime performance, robust to the available computation time.

1 Introduction

The use of computer vision in autonomous robotics has been studied for decades. Recently, applications such as autonomous vision-based vehicle navigation [1], 3-D localization and mapping [11, 4, 3] and object recognition [10] have gained popularity due to the combination of increased processing power, new algorithms with real-time performance and the advancements in high quality, low-cost digital cameras. These factors enable autonomous robots to perform complex, real-time, tasks using visual sensors.

Such applications are often based on a local feature matching algorithm, finding point correspondences between two images. There are many different algorithms for feature matching, however in recent years there is a growing research on algorithms that use local invariant features (for a survey see [16, 13]). These features are usually invariant to image scale and rotation and also robust to changes in illumination, noise and minor changes in viewpoint. In addition, these features are distinctive and easy to match against a large database of local features.

Unfortunately, existing algorithms for local feature matching [2, 11, 12] are designed under the assumption that they will run to completion and only then return a complete result. Many of these algorithms therefore require significant computational resources to run in real-time. As we show in the experiments, this prohibits some of the algorithms from being used in current robotic platforms (where computation is limited). For instance, a Nao[1] humanoid robot computing the full set of features in an image of size 640×480 requires 2.4 seconds using a state-of-the-art implementation of the SURF algorithm [2, 15].

Note, however, that for many robotics applications, even partial results—a subset of all features in the image—would have been sufficient (for example, to estimate the pose of the robot for obstacle detection). On the other hand, being able to invest computation time in getting higher-quality results is also important, e.g., in object recognition or in building accurate maps. Indeed, robots can benefit from computationally-flexible algorithms, where the computation time is traded for the accuracy requirements of the task. To do this, simply interrupting the algorithm when needed is not enough: We need to guarantee that the results of the algorithm would necessarily monotonically increase in quality, given additional computation time. This class of algorithms is called *Anytime* [17].

In this paper we present AnySURF, an anytime feature-matching algorithm, which can accumulate results iteratively, with monotonically increasing quality and minimal overhead. We achieve flexibility by re-designing several major steps in the SURF algorithm [2], mainly the feature search process and the order of interest point detection. We additionally discuss the design choices underlying AnySURF.

We evaluate the use of AnySURF in a series of experiments. We first demonstrate that non-anytime feature matching indeed suffers from significant computation time on limited platforms (including, in particular, the Nao humanoid robot). Then, we contrast different design choices for AnySURF, and analyze its performance profile under different image types. We also demonstrate the usability of AnySURF in computing approximate homography.

2 Related Work

Image matching using local features (or interest points) has been around for almost three decades – the term "interest point" was first introduced by Moravec in 1979 [14]. A decade ago Lowe [10] introduced Scale Invariant Feature Transform (SIFT), which had a significant impact on the popularity of local features. Since SIFT was published, several new algorithms inspired by SIFT have emerged, including PCA-SIFT [8], GLOH [12] and SURF [2].

SURF [2] is a state of the art algorithm for local invariant feature matching - a scale and rotation invariant interest point detector and descriptor. SURF is composed of three steps similar to SIFT, however it uses faster feature detection / extraction algorithms (approximation of the Hessian matrix and using the distribution of Haar-wavelet responses within the interest point neighborhood, relying on integral images to reduce computation time). SURF is faster to compute than SIFT, while allowing for comparable results.

[1] http://www.aldebaran-robotics.com

SIFT, SURF and other such algorithms are not anytime algorithms. Although several authors did accomplish complex real-time visual tasks such as Visual SLAM, using SIFT-like features [3] and correlation with reference templates [4], these implementations were tailored for a specific platform and are not computationally flexible. Therefore, they do not answer out research goals.

3 Methodology

The proposed AnySURF algorithm is based on SURF, which was selected over SIFT and SIFT-like algorithms since it is more suitable for an anytime implementation while also having an excellent quality/run-time ratio. It is more suitable for a flexible implementation because whereas SIFT begins with the computationally expensive operation of constructing several scale space representations (DoG, Difference of Gaussians approximation), SURF is based on a basic approximation of the Hessian matrix where an integral image is computed once for all scales and only the filter size changes when working on each scale. This means that in SURF there is very little overhead for working with specific scales or areas and this is very suitable for a flexible algorithm.

3.1 How does SURF works?

SURF [2] is composed of three steps: Detecting interest points, calculating descriptors and matching them (Alg. 1).

The first step, detection of interest points, starts with scale-space extrema detection: search over all scales and image locations is performed, using an approximation of the Hessian matrix to identify potential interest points that are invariant to scale and rotation. Calculation of the Hessian approximation relies on an integral image to reduce computation time. Interest points are first thresholded so that all values below a predetermined threshold are removed, then a non-maximal suppression is performed to find candidate points (each pixel is compared to its 26 neighbours, comprised of the 8 points in the native scale and 9 in each of the scales above and below) and finally they are localized in both scale and space by fitting a 3D quadratic.

The second step, calculation of the keypoint descriptors, is based on a distribution of Haar-wavelet responses within the interest point neighborhood, again relying on integral images for speed. The SURF descriptor describes how the pixel intensities are distributed within a scale dependent neighbourhood around each detected interest point. It is calculated by first assigning a repeatable orientation via Haar wavelet responses weighted with a Gaussian centered at the interest point, then a square oriented window is constructed around the interest point, divided into 4×4 regular sub-regions. For each sub-region 4 Haar wavelets responses are summed up (d_x, d_y, $|d_x|$, $|d_y|$), so a vector of length $4 \times 4 \times 4 = 64$ is produced.

The third step, matching different descriptors, is done via the Euclidean distance of their feature vectors. A fast nearest-neighbor algorithm is used that

Algorithm 1 Generic SURF
(Input: image; Output: list of matched descriptors)

```
0.  Construct integral image
1.1 Over all octaves (Fine-to-Coarse)
1.2   Pre-calculate discriminants
1.3   Over inner octave layers
1.4     Over all pixels
1.5       Find interest point
2.1 Over all interest points
2.2   Calculate descriptor
3.1 Over all descriptors
3.2   Match descriptor
3.3   Add matched descriptor to list
4.  Return list of matched descriptors
```

can perform this computation rapidly against large databases. SURF uses the sign of the Laplacian (the trace of the Hessian matrix) to distinguish bright features on dark background from the reverse situation. Since SURF's descriptor uses 64 dimensions, time for feature computation and matching is reduced.

3.2 Making SURF computationally flexible

In order to make SURF computationally flexible, several important design decisions had to be made. These are: accumulating results iteratively, using a suitable search strategy and calculating the Hessian in-place. An in-depth explanation of these design decision follows. The impact of these decisions is presented in Section 4.

Guaranteeing Monotonically-Improving Descriptor List The first step in making an anytime version of SURF is trivial: Accumulate results iteratively. SURF divides the work to several large consecutive steps (get all interest points from all scales, compute descriptors for all interest points, match all descriptors against database - Alg. 1, steps 1–3) and so if the final stage is not reached – there might be no useful results. Contrary to this batch approach, we propose an iterative approach, where results are accumulated during the execution of the algorithm and are returned when the algorithm is interrupted.

This can be achieved by computing a full result, including a descriptor, in each iteration. The new descriptor can immediately be used to match against a database. This change is trivial yet vital as it guarantees good anytime functionality: usable results are generated such that the number of results is monotonically increasing.

Generating Descriptors Faster Now that we can guarantee that the list of matched descriptors will be monotonically-increasing in length, we can explore design choices that can make sure quality descriptors are generated faster. Below we discuss two such design choices.

Search strategy Detection of interest points (Alg. 1, step 1) is done by scanning the entire image in multiple octaves. This search usually starts with the smallest kernel and continues applying kernels of increasing size to the image. Since our flexible algorithm accumulates results iteratively, we have an opportunity to select an ordering on the search of octaves for interest points (Alg. 1, step 1.1), thereby allowing detection of more promising features earlier. Note that this search strategy need not be hard-coded, but can be changed according to the image or task at hand.

We considered two types of general search strategies: Coarse-to-Fine and Fine-to-Coarse. Coarse-to-Fine means we start with the largest filter size and continue to use smaller filter sizes so that we find larger features first and smaller ones later, while Fine-to-Coarse means the exact opposite. Note that if the algorithm runs to completion the search order does not matter and exactly the same features are found. Additional search strategies are also possible: order of going over inner octave layers (Alg. 1, step 1.3), order of going over pixels (Alg. 1, step 1.4), however we did not consider them here.

Selecting an appropriate search strategy according to the image type (e.g., blurry image) can maximize the number of features detected during the early phase of the search. However, sometimes the number of features is not what we prefer to optimize. For example, some vision tasks work better when the features have a good spatial distribution over the image (e.g., homography calculation [7]) or when coarse features are first matched and only then fine features are searched for in a limited area to complete the match (e.g., object recognition [10]). In such cases it might be preferable to use Coarse-to-Fine search, even if the initial number of features is smaller when compared to the Fine-to-Coarse strategy. In other cases, such as when we have prior information that there are very few Coarse features or when we know we search for small (Fine) features, Fine-to-Coarse strategy can be used.

Calculating the Hessian discriminants in-place All SURF implementations we inspected (Pan-o-matic, OpenSURF[5], OpenCV) pre-calculate the determinant of Hessian (discriminant) for each octave, over the entire image (Alg. 1, step 1.2). This step has high initial computational cost, however once calculated, results are faster to compute so the total running time is lower. Since we assume the algorithm might not run till completion, it might be preferable to sacrifice some of the running-time in order to get initial results sooner.

Memory consumption by the pre-calculated arrays is another issue to consider. Pre-calculating the determinant of Hessian requires several 2D arrays to be kept in memory. The SURF implementations we inspected (see above) use arrays the size of a full image to simplify coding (smaller arrays can however be used). So we have number of layers$\times$image_width$\times$image_height, which means multiple arrays each one the size of a full image are saved in memory. For large images or platforms with little memory available, this can be quite problematic.

Algorithm 2 AnySURF
(Input: image; Output: list of matched descriptors)

```
0.   Construct integral image
1.   While not interrupted
2.1    Over all octaves (Coarse-to-Fine)
2.2      Over inner octave layers
2.3        Over all pixels
2.4          Find interest point
2.5          Calculate descriptor
2.6          Match descriptor
2.7          Add matched descriptor to list
3.   Return list of matched descriptors
```

Obviously, when pre-calculation is not used and the determinant of Hessian is calculated in-place, there is no need to save multiple arrays in memory.

3.3 AnySURF - Anytime SURF

The following algorithm (Alg. 2), coined AnySURF (Anytime SURF), is a computationally flexible SURF algorithm. Results are accumulated iteratively, with a descriptor computed in each iteration. Octaves are searched in Coarse-to-Fine order and the determinant of Hessian is calculated in-place. We believe these design choices are appropriate for a generic Anytime SURF algorithm and an analysis of the Anytime performance profile is performed in Section 4.

A possible variant of AnySURF is to use pre-calculation. Compared to a batch approach such as panosurf, this alternative is more suitable to anytime since results are produced earlier yet the total computation time is exactly the same. Compared to the AnySURF without pre-calculation, the total computation time of this variant is lower yet first results are generated much later since pre-calculation has a high initial computational cost (see Figure 1).

4 Results

A flexible algorithm is required only when the non-flexible algorithm is slow and when partial / low accuracy results are useful. In this section we will show that both criteria are met in SURF. In addition, we analyze the design decisions explained in Section 3.2 and present an example of using AnySURF to approximate homography between 2 images.

To demonstrate that SURF is not fast enough for real-time full image feature search on current robotic platforms which have limited computational power, Table 1 shows computation time on multiple platforms for the same image in different sizes (QVGA: 320×240, VGA: 640×480, 3MP: 2048×1536). Evaluation was done using Pan-o-matic open-source SURF implementation [15] with default parameters, which produces very similar results compared to the published SURF binary [6] to which source code is not available.

Table 1. SURF detector-descriptor computation time (ms) on different image sizes and platforms

Platform	QVGA	VGA	3MP
Desktop PC (Intel Q9400 2.66GHz)	27	103	1021
Mini-ITX (Intel T7200 2.0GHz)	74	249	1599
Nao Robot (x86 AMD GEODE 500MHz)	560	2425	26367
Nokia N900 (ARM Cortex-A8 600MHz)	938	3656	442512

From Table 1 it is clear that in order to run real-time full-image feature search with SURF we need to work on a small resolution image coupled with a powerful platform. In addition, in this test the CPU and memory were devoted entirely to the SURF process, while in robotic applications additional non-vision tasks might also require processing time and memory usage (localization, mapping, motion generation, behavior selection, etc.).

Table 2 shows where the processing time is spent across the different major steps. The Intel Q9400 platform was selected for this test, to eliminate as many bottlenecks as possible and allow the optimal behavior of the algorithm show.

Table 2. Analysis of SURF detector-descriptor computation time (ms), on Intel Q9400 2.66 GHz

Image size	integral image	keypoints	descriptors
QVGA	1 (3.7%)	19 (70.4%)	7 (25.9%)
VGA	3 (2.9%)	80 (77.7%)	20 (19.4%)
3MP	32 (3.1%)	910 (89.1%)	79 (7.8%)

As can be seen in Table 2, calculation of the integral image is minor (~3.2%) and detection of keypoints takes most of the time (~79%). In the context of a flexible algorithm, since detecting keypoints takes most of the time, it seems beneficial to calculate the descriptor immediately upon keypoint detection, thus significantly shortening the time till partial results are available. We now turn to analyzing the impact of the various design choices of AnySURF (presented in Section 3.2).

The image database used in all following figures is a standard evaluation set, provided by Mikolajczyk [12]. It contains 48 images across 8 different scenes. All images are of medium resolution (approximately 800×640 pixels) and are either of planar scenes or the camera position is fixed during acquisition, so that in all cases the images are related by homographies (plane projective transformations). The scenes contain different imaging conditions: viewpoint changes, scale changes, image blur, JPEG compression and illumination changes.

Figure 1 shows the averaged rate of acquiring descriptors (%) as a function of run-time (%). Three alternatives are considered: First, a SURF implementation called Pan-o-matic [15] (henceforth will be referred to as panosurf). This implementation first detects all keypoints at all scales and only then calculates descriptors (as in Alg. 1). In addition, the determinant of Hessian is pre-calculated. Next, we tested two variants of the AnySURF algorithm (Alg. 2), where descriptors are computed immediately upon keypoint detection. In the first variant pre-calculation of the determinant of Hessian is used and in the other it is calculated in-place.

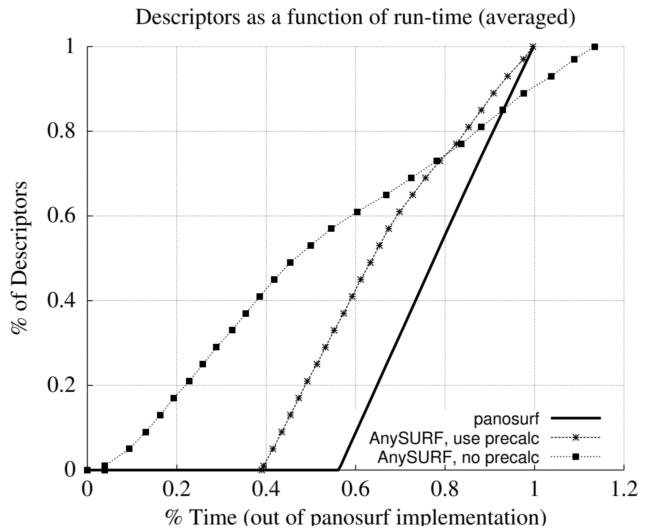

Fig. 1. The descriptors (%) vs. time (%) graph for different algorithms. Data is averaged across all (48) database images. Time (%) is compared to panosurf time (therefore can be > 1.0).

Figure 1 allows us to inspect the impact of calculating descriptors immediately upon keypoint detection and also the effect of pre-calculation. First, let us consider AnySURF with pre-calculation: calculating descriptors immediately is beneficial compared to the original panosurf approach since it does not adversely affect the total computation time while allowing results to start accumulating earlier (after 39% of time passed instead of 57% as in panosurf). Now, let us consider AnySURF without pre-calculation: although the algorithm does take longer to complete (13.5% more on average), we start getting results almost immediately (after 4% of time passed), with a near-linear acquire rate. Note that pre-calculation is only useful when all descriptors are needed or when it can be assumed that the algorithm will run to near completion (AnySURF with pre-calculation supersedes the no pre-calculation version after 80% of the time passed).

Next, let us inspect the impact of the search strategy. As explained in Section 3.2, since AnySURF accumulates results iteratively, we can select an ordering on the search for interest points. The following figure is of a specific images (1st image of "bricks" sequence), showing the number of descriptors as a function of run-time. All four combinations are shown (with/without pre-calculation, Coarse-to-Fine/Fine-to-Coarse search strategy), compared to the baseline panosurf.

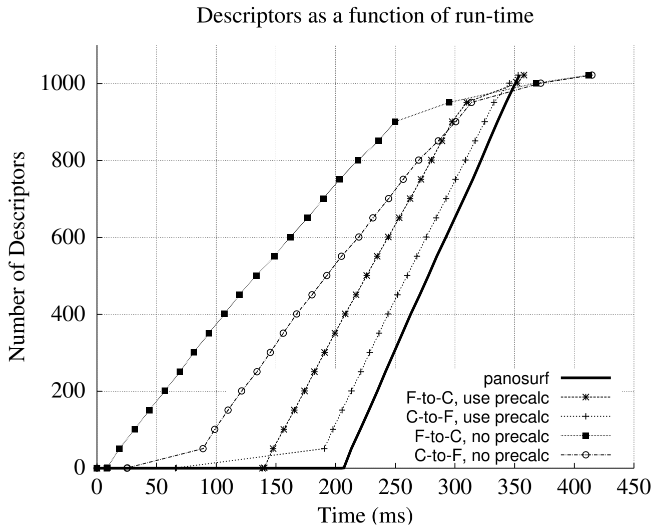

Fig. 2. 1st image of bricks sequence, panosurf displayed as a baseline, AnySURF with Coarse-to-Fine and Fine-to-Coarse search strategies displayed with and without pre-calculation

In Figure 2, the Fine-to-Coarse search strategy produces results much faster than Coarse-to-Fine (in other images the opposite might be true). Note that results start sooner, accumulate faster and the difference in number of descriptors is significant, up to an order of magnitude (e.g., after ~30ms). This means that appropriate selection of the search strategy is vital for an efficient Anytime performance.

As for the use of pre-calculation, it seems that our previous conclusion holds and for an anytime algorithm an in-place calculation is preferred. However, in other images (not shown here due to lack of space) the pre-calculated (Coarse-to-Fine) version supersedes the in-place (Fine-to-Coarse) version. This only stresses that selecting the appropriate search strategy is very important indeed: when selecting the correct search strategy, the no pre-calculation (Coarse-to-Fine) version triumphs again (at least until most of the descriptors are found, as explained earlier).

After witnessing a major difference in the above specific image according to the chosen search strategy, it is interesting how the Fine-to-Coarse vs. Coarse-to-Fine search strategies perform in our full image database, across the different scenes. To test this, Figure 3 shows the effect of search strategy in different scenes on the first 50ms of AnySURF (without pre-calculation).

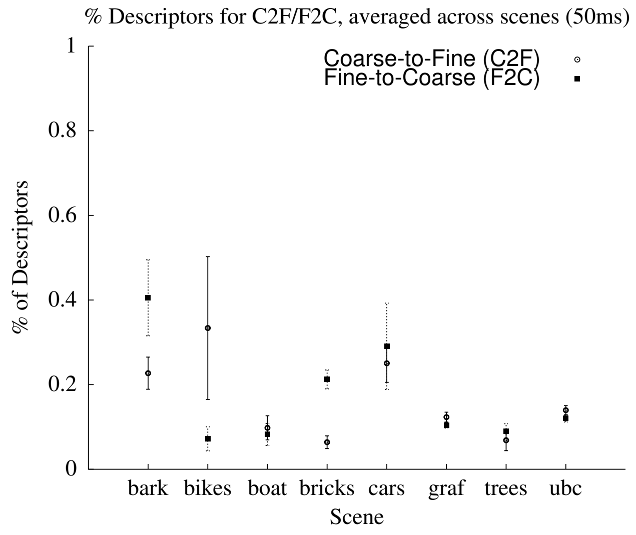

Fig. 3. The descriptors (%) for Coarse-to-Fine and Fine-to-Coarse tested on different scenes (first 50ms of AnySURF). The markers represent the mean for each scene, the error bars are two standard deviation units in length.

Figure 3 demonstrates that it is beneficial to select the appropriate searching strategy according to the type of image at hand if the number of descriptors is to be optimized. The image sequences "bark" and "bricks" are clearly more suited for Fine-to-Coarse search whereas "bikes" is more suited for Coarse-to-Fine search. The reason behind this is that "bikes" is a sequence of blurred images (so there are less fine features and processing time is wasted on searching for them) while "bark" and "bricks" contain images with many fine features and few coarse ones. It is also interesting to note that between 5% to 40% of all descriptors can be acquired within 50ms (however, higher percentages are usually for images with a lower total number of descriptors and a lower total computation time). The average number of descriptors acquired after 50ms on our image database is 119 for Coarse-to-Fine and 138 for Fine-to-Coarse.

A higher number of descriptors is not necessarily better. For example, coarse features are larger, fewer and usually more spread over the image so they might suit some tasks better than fine features. One such task is homography estimation [7]. Figure 4 shows the time passed till the homography between the 1st and 2nd images in each scene could be estimated (to within an order of magnitude from the optimal value). Homography was calculated via the RANSAC approach [7, 9].

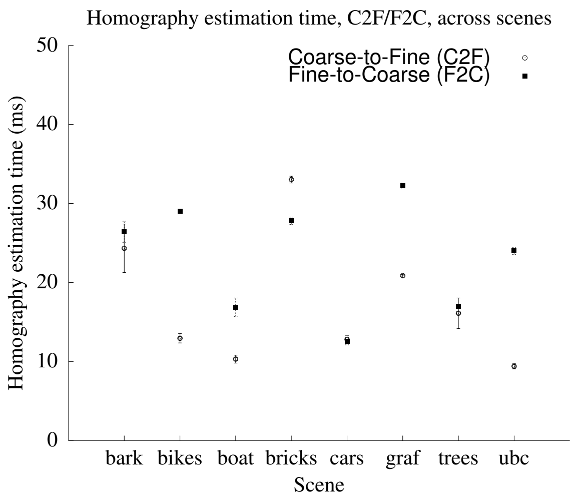

Fig. 4. Homography approximation time for Coarse-to-Fine and Fine-to-Coarse, tested across scenes. In each scene image 1 was processed fully. Image 2 was processed till the homography could be estimated (to within an order of magnitude from optimal value). The markers represent the mean for each sequence, the error bars are two standard deviation units in length.

Figure 4 demonstrates that coarse features enable a quicker homography estimation compared to fine features. The Coarse-to-Fine search strategy took equal or less time to estimate the homography in most scenes (7/8) and more time in just one scene (the bricks scene, which contains very few coarse features and so a lot of time is wasted searching for coarse features). The results for the bark, cars and trees scenes do not differ significantly, while results for others do (two-tailed t-test, p=0.01). Also note that the homography could be estimated within a very short time (~20 ms).

Finally, we go back to evaluating the use of flexible local feature matching on the Nao robot platform. Prior to AnySURF, estimating a homography between two images on this platform took on average 4 seconds (averaged across all images in database, similar to Figure 4). This processing time was spent not on estimating the homography itself, but on computing all descriptors in the image. However, for estimating a homography a subset of the results suffice, so AnySURF can be used. Using AnySURF, this task is completed within 0.33 seconds, faster by an order of magnitude. Note that this homography can actually assist in computing the remaining descriptors faster, since we can now estimate their location.

5 Conclusion

We presented and analyzed AnySURF, a SURF-based anytime local feature matching algorithm, which can trade quality of results for computation time:

It guarantees that the number of matched descriptors monotonically increases with computation. For robotics applications that can work with a subset of descriptors, this allows for much faster response times.

We discuss and carefully evaluate design choices in the feature search process, using several computational platforms. We demonstrate that changing the feature search order can significantly impact the rate at which descriptors are generated. Surprisingly, avoiding pre-calculation steps that are intended to optimize the search process, leads to generating results at a faster rate. Future work will focus on the problem of dynamically managing AnySURF within the context of a flexible real-time vision-based autonomous navigation system.

References

1. DARPA grand challenge. http://www.darpa.mil/grandchallenge/index.asp, 2007.
2. H. Bay, T. Tuytelaars, and L. V. Gool. SURF: speeded up robust features. In *Computer Vision – ECCV 2006*, pages 404–417. 2006.
3. D. Chekhlov, M. Pupilli, W. Mayol-cuevas, and A. Calway. Real-time and robust monocular SLAM using predictive multi-resolution descriptors. *In 2nd International Symposium on Visual Computing*, 2006.
4. A. Davison, I. Reid, N. Molton, and O. Stasse. MonoSLAM: Real-Time single camera SLAM. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052–1067, 2007.
5. C. Evans. Notes on the OpenSURF library. Technical report, University of Bristol, Jan. 2009.
6. D. Gossow, D. Paulus, and P. Decker. An evaluation of open source SURF implementations. In *RoboCup 2010: Robot Soccer World Cup XIV*. 2010.
7. R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2 edition, Apr. 2004.
8. Y. Ke and R. Sukthankar. PCA-SIFT: a more distinctive representation for local image descriptors. *null*, 2:506—513, 2004.
9. P. D. Kovesi. MATLAB and Octave functions for computer vision and image processing. http://www.csse.uwa.edu.au/~pk/Research/MatlabFns, 2000.
10. D. G. Lowe. Object recognition from local Scale-Invariant features. In *Proceedings of the International Conference on Computer Vision - Volume 2*, page 1150. IEEE Computer Society, 1999.
11. D. G. Lowe. Distinctive image features from Scale-Invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov. 2004.
12. K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(10):1615–1630, 2005.
13. K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. V. Gool. A comparison of affine region detectors. *Int. J. Comput. Vision*, 65(1-2):43–72, 2005.
14. H. P. Moravec. *Visual Mapping by a Robot Rover*. 1979.
15. A. Orlinski. Pan-o-matic - automatic control point creator for hugin.
16. T. Tuytelaars and K. Mikolajczyk. Local invariant feature detectors: a survey. *Found. Trends. Comput. Graph. Vis.*, 3(3):177–280, 2008.
17. S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.