

# A New Hierarchical Approach in Robust Real-Time Image Feature Detection and Matching

M. Langer and K.-D. Kuhnert

*Institute of Real-Time Learning Systems, University Siegen, Germany*  
{*langer,kuhnert*}@fb12.uni-siegen.de

## Abstract

*Object recognition forms a ubiquitous problem in digital image processing. The detection of robust image features of high distinctiveness is one important key in this regard. We present a new hierarchical approach in object recognition targeting at high robustness, yet trying to fulfill hard real-time constraints. The former will be achieved using SIFT and SURF operators, while the latter is done by employing a fast pre-processing step exploiting decision-trees.*

## 1. Introduction

Among various methods in pattern or object recognition, one popular method is to detect certain interest points at distinctive locations in the image, such as corners, blobs, or T-junctions. The interest point detector is supposed to be repeatable under varying viewing conditions (e.g. different lighting or different viewing angles). Having found distinctive interest points in an image, the interest points are examined more closely regarding their neighborhood. Taking the neighboring pixel into account forms an image feature or image descriptor, which has to fulfill certain criteria regarding distinctiveness, robustness against noise, detection errors, and image deformations. These image features can then be matched against features computed from other images. Matching features indicate a high likeliness of correspondence between the two images.

Classic interest point detectors used simple attributes like edges or corners. Among those, the probably most widely used, is the Harris corner detector introduced in 1988. However, this detector lacks scale-invariance. Lindeberg [2] tried to extend the Harris corner detector experimenting with the determinant of the Hessian matrix as well as the Laplacian to detect blob-like shapes. Mikolajczyk and Schmid [4] refined this method exploiting a combination of the Hessian matrix to deter-

mine the location, and the Laplace matrix to determine the scale of an image feature. This yielded the desired scale-invariance. Lowe [3] mainly focused on speeding up this computational costly processing by replacing the Laplacian of Gaussian by a Difference of Gaussian filter. In 2003 he presented his method called SIFT (scale invariant feature transform) to the public. SIFT features are highly distinctive, scale invariant, rotation invariant, and highly tolerant against illumination effects. In 2006, Bay, Tuytelaars and van Gool [1] showed that SIFT could be speeded up further, almost without losing any of its matching quality.

We will present a novel *hierarchical approach* in object recognition, which employs decision trees as a pre-stage analysis part in combination with SIFT and SURF operators as the main-stage object recognition part. This adds even better runtime performance to the very robust and distinctive image features.

## 2. Related Work

**SIFT.** One of the image feature operators used in this work is the *scale invariant feature transformation* (SIFT) as proposed by Lowe [3]. The features are highly distinctive and invariant to image scaling and rotation. There is also a partial invariance to illumination and occlusion. To exploit SIFT for object recognition, one has to build up a database of image features from a training set of images. After that, the features of a test set of images can be matched with this database. The algorithm for computing SIFT image descriptors is shown in Alg. 1. The yielded descriptors are stored in the database. These are the image features, which the objects that are to be analyzed are matched with. One such image feature vector has 128 components. It is also important to note (for the later comparison to the SURF operator), that Lowe uses a *Difference of Gaussian* (DoG) filter for scale space extrema detection.

To recognize an object, the image of the object itself has to undergo the previously described steps. The

---

**Algorithm 1:** SIFT

---

**Input:** Image  $I$   
**Output:** SIFT descriptor set  $D$   
**begin**  
   $S \leftarrow \emptyset$   
   $E \leftarrow \text{ScaleSpaceExtrema}(I)$   
   $K \leftarrow \text{LocalizeKeypoints}(E)$   
  **forall**  $k \in K$  **do**  
     $f \leftarrow \text{GenerateFeature}(k)$   
     $\text{AssignOrientation}(f)$   
     $D \cup \text{ComputeKeyPoint}(f)$   
  **return**  $D$   
**end**

---

yielded image descriptors are then compared to those in the database. To achieve reliable results, clusters of at least three features are identified, that agree on the object.

The SIFT transformation can easily yield thousands of image descriptors. Under hard real-time constraints, SIFT cannot be used without further optimizations.

**SURF.** The *Speeded Up Robust Feature* (SURF) operator developed by Bay, Tuytelaars and van Gool [1] is an extension and modification to the previously discussed SIFT operator. SURF aims at better run-time performance of the image feature detector, yet preserving the high reliability and accuracy of SIFT. This is basically achieved by two major modification to the SIFT operator, while the basic algorithm for computing the image feature vectors stays the same as in Alg. 1.

First, the DoG filter is approximated by a *Difference of Means* (DoM) filter. This approximation is admissible, because of the cropping of the continuous, infinite Gaussian kernel<sup>1</sup> in the SIFT scale space extrema computation. The use of the DoM filter has constant run-time complexity  $\mathcal{O}(1)$  due to the possibility of exploiting *integral images* for a DoM implementation. This technique also simplifies the SURF feature orientation assignment step (see the corresponding step in Alg. 1), which employs *haar wavelet* filter responses. It is also interesting to note, that during the scale space computation it is actually not the image, which is scaled, but the DoM filter (converse to the DoG filter).

The second important modification is the reduction of the image feature vector length to half the size of the SIFT feature vector length (from 128 components down to 64). This ensures higher matching speed between different feature vectors. The descriptor components are calculated over a rectangular region, which is subdivided into smaller  $4 \times 4$  sub-regions.

---

<sup>1</sup>which is actually an approximation of the real Gaussian kernel itself

Over each such region a descriptor vector  $v$  is computed representing the underlying intensity structure of the sub-region<sup>2</sup>. This vector has four components  $v = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|)$  with  $d_x$  denoting the haar wavelet response in x-direction and  $d_y$  in y-direction respectively. Hence, we have for each  $4 \times 4$  sub-region a 4 component descriptor resulting in a 64 dimensional image feature vector.

These modifications result in a computation speedup factor of 3 compared to the SIFT operator. In the next section we present a preprocessing step to further decrease computation time. Instead of trying to extend or modify the SIFT or SURF operators directly, we try to reduce the amount of data that needs to be processed *before* the costly matching operations are applied to the images employing binary decision trees.

**Decision Trees.** *Decision trees* are a classic tool rooting in artificial intelligence and are mainly used for data mining purposes. Though they were not invented to suit object recognition purposes, decision trees used in this context can mitigate the high runtime-complexities especially regarding image feature based approaches like SIFT and SURF.

In [5] decision trees are exploited to speed up the matching process of the introduced SIFT operator. The method is called *LAF-tree* (local affine frames tree). Instead of computing the image descriptor on a fixed sized image patch yielding fixed sized descriptor vectors, variable size patches are examined, which is beneficial for certain fine scaled structures like wire-frames. From these patches a decision tree is constructed that aids in the SIFT matching process. Patches of the query image need not be compared to patches in a previously constructed database linearly; the decision tree compares the query-patch with his nodes and descends until a leaf is reached representing the best match for the query. The tree-structure provides sub-linear run-time behaviour reducing  $\mathcal{O}(N)$  to  $\mathcal{O}(\log(N))$  with  $N$  denoting the object numbers. Employing this method, the time consumed for the matching stage is reduced to a few milliseconds. See [5] for further information on how the tree is trained, the conducted tests, and results.

**Bayesian Framework.** Another approach in optimizing the costly image feature matching process is presented in [6]. Instead of using hierarchical data structures like trees or hash-tables, the authors introduce a non-hierarchical data model coined *ferns*, which classify image patches in conjunction with *Bayesian classifiers*. Each fern is a set of binary tests that returns the likelihood of the patch belonging to any of the learned classes from the system's training phase. The responses are processed by the Bayesian framework introduced by

---

<sup>2</sup>for details on this computation see section 4.2 in [1]

the authors. It is shown that the presented system tolerates significant error rates, is easy to implement, scales well for a large number of classes, and allows fast and incremental training.

### 3. Preprocessing Employing Decision Trees

We use a fast classification method in this preprocessing phase exploiting binary decision trees. Objects, that have been correctly classified, do not need to undergo the costly SIFT and SURF operations anymore. The idea here is to recursively divide the feature space into sets of similar sizes and feed these sets to nodes into the decision tree. To classify an object, the object features are matched with the class feature sets each node holds. Each node decides whether an object belongs to one (or none) of its two class–feature sets. The decision made by the node determines the next subtree, which has to deal with the classification. This process is repeated until a leaf–node is reached. The feature represented by this leaf node yields the class the object belongs to. Because the tree needs not necessarily be built up completely, whole class or feature sets can be represented by leaf nodes.

We pursue a simple color cluster driven approach to train the decision tree based classifiers. Although yielding high performance, the training of the decision trees proved to be difficult regarding classification quality, which is at 66% of correctly classified objects on average. This is mainly due to the simple color based features we use, but poses no real disadvantage as we focus on speed only in this stage. To construct the tree, we apply Alg. 2. Each node divides the center of gravity set (and so the associated class set) into half. This way we achieve a runtime complexity of  $\mathcal{O}(M \log(N))$  with  $N$  denoting the number of classes and  $M$  the number of objects to classify.

The problem of low classification quality can be mitigated by moving to *decision forests* (see [8]). Decision forests integrate more than one decision tree into their decision process. These trees are trained with varying datasets which may lead to different classification results. Among all the trees in the forest a quorum is made (usually a simple majority decision) and the decision taken which should be correct most likely. Applying this method rises the correctly classified objects rate to 75%, but also increases the runtime complexity to  $\mathcal{O}(KM \log(N))$  with  $K$  denoting the number of trees in the forest.

The rate of correctly classified objects in this stage may seem to be quite low compared to other more sophisticated classifiers, but it is important to note that we employ the preprocessing stage only to speed up the

---

#### Algorithm 2: Generating a decision tree

---

**Data:** training sets  $S$  of images  
**Result:** generated decision tree  $d$

```

begin
   $C \leftarrow \emptyset$ 
  forall  $I \in S$  (with  $I \subset S$ ) do
     $G \leftarrow \emptyset$ 
    repeat
       $A \leftarrow P \leftarrow \emptyset$ 
      forall  $i \in I$  do
         $P \cup \text{GaussPyr}(i)$ 
         $\text{avg}_{RGB} \leftarrow P(i)_{\text{max. pyr. level}}$ 
         $A \cup \text{avg}_{RGB}$ 
       $G \cup \text{CalcCOGs}(A)$ 
    until  $|S| = 2$ 
     $C \cup G$  ( $C$  holds all CoG subsets)
   $d \leftarrow \text{BuildDecisionTree}(C)$ 
return  $d$ 
end

```

---

later object recognition process using SIFT and SURF. Every correctly classified object at the end of this phase means, that no costly search through an image feature database needs to be performed anymore.

### 4. Results

We use the Coil–100 image database<sup>3</sup> as a testbed for our algorithms. This library consists of 100 different, small, everyday use objects, which have about the same size. Each object is rotated around the Y–axis 72 times at a  $5^\circ$  angle each step, thus covering the full  $360^\circ$ , which yields 7,200 object view images in total. Each image has a size of  $128 \times 128$  pixel.

The two–stage image feature matching approach yields a very efficient object recognition method providing highly accurate results. The preprocessing of the algorithm drastically reduces the search–space at minimal computational costs of a few milliseconds.

A typical feature match of a Coil–DB image with the SIFT feature database performs at 75ms. Applying the preprocessing, which yields 75% of correctly classified objects<sup>4</sup>, the processing time reduces to 21ms. This time reduction is correlated linearly to the rate of correctly classified objects of the preprocessing phase. Generally the overall computation time can be computed by  $t_o = t_p + (1 - r)t_s + t_d$  with  $t_o$  denoting the overall time,  $t_p$  the time for precomputation,  $t_s$  the

<sup>3</sup><http://www1.cs.columbia.edu/CAVE/software/softlib/coil-100.php>

<sup>4</sup>which is the average rate for well trained decision forests in our implementation

**Table 1. Results of the object recognition process employing SIFT features.**

feature distance	correct positive	false positive
40,000	49.30%	0.00%
60,000	65.38%	0.09%
80,000	85.66%	1.74%
100,000	95.10%	8.85%
120,000	97.55%	31.78%

time for the SIFT matching process, and  $t_d$  the dead time (consumed by memory management and other administrative tasks).  $r$  is the ratio of correctly classified objects from the earlier preprocessing-stage. Considering the fact that  $t_p \ll t_s$ , applying the precomputation yields better overall timing.

Tab. 1 shows some results of the classification process applying SIFT. It clearly shows the high accuracy and reliability employing SIFT features. The used images are randomly taken from the Coil-DB. The feature distance denotes the threshold that is taken for feature comparison. Features are compared to each other using the squared euclidean distance over the 128 dimensional feature vectors. Only feature distances that are below this threshold are considered to be equal enough to yield a match. We averaged our results over several objects that were taken out of the Coil-100 database randomly. The angle of the object views were all  $0^\circ$ . The images we used were to be found in a subset of the database that consisted of all objects and object views from an angle of  $\pm 60^\circ$ . Angles between  $\pm 60^\circ$  and  $\pm 90^\circ$  cannot be reliably matched with the SIFT operator anymore. Values beyond  $\pm 90^\circ$  are impossible to use with SIFT, because the object views now show their backside of the object, whereas the reference object view shows its frontside. For SIFT this is equivalent to a total occlusion of the object. The optimal feature distance for a concrete image set is determined experimentally and changes according to different environments.

Exchanging the SIFT with the SURF operator the values from Tab. 1 stay almost the same (with only slight deviations), except for the even better run-time performance of the system. We could verify that the SURF operator is indeed almost 3 times faster than SIFT. The overall computation time of 21ms reduces to 10ms, if SIFT is replaced by SURF in the main-processing stage.

We also took our own image sets with standard CMOS cameras at standard sizes of  $320 \times 240$  pixel.

The object recognition system is capable of processing at a rate of 11fps (SIFT) and 16fps (SURF). This gets us close to the desired real-time behavior of 20fps, yet exploiting all the benefits of the SIFT and SURF operators.

## 5. Conclusion and Future Work

In this paper we showed an extension to the very reliable and accurate object recognition methods SIFT and SURF. We focused on speeding up these two operators even further targeting at real-time behavior (at least 20fps for average sized images). We used a preprocessing step exploiting decision trees to sort out as much data as possible in an early stage, trying to employ the costly SIFT and SURF matching process only at falsely recognized objects. We showed that this method is capable of further reducing overall computation time by the formulas given in section 4.

We intend to use this hierarchical approach for use in an *analysis by synthesis* system (see [7]). This system is supposed to reliably detect real-world objects from synthetic three dimensional scene models. Because this is an iterative computational process approximating, the best synthesis parameters for a given real-world object, one faces hard real-time constraints. We are confident, that our approach presented in this paper embodies the potency to fulfill those constraints.

## References

- [1] H. Bay, T. Tuytelaars, and L. V. Gool. Surf: Speeded up robust features. In *9th European Conference on Computer Vision*, May 2006.
- [2] T. Lindeberg. Feature detection with automatic scale selection. In *Int. Journal of Computer Vision*, volume 30, pages 77–116, 1998.
- [3] D. G. Lowe. Distinctive image features from scale-invariant keypoints. In *Int. Journal of Computer Vision*, volume 20, pages 91–110, 2003.
- [4] K. Mikolajczyk and C. Schmid. Indexing based on scale invariant interest points. In *ICCV*, volume 1, pages 525–531, 2001.
- [5] S. Obdrzalek and J. Matas. Sub-linear indexing for large scale object recognition. In *BMVC*, 2005.
- [6] M. Ozuysal, P. Fua, and V. Lepetit. Fast keypoint recognition in ten lines of code. In *Conference on Computer Vision and Pattern Recognition*, June 2007.
- [7] S. Todt, M. Langer, C. Rezk-Salama, A. Kolb, and K.-D. Kuhnert. Spherical lightfield rendering in application for analysis by synthesis. In *Dynamic 3D Imaging Workshop in Conjunction with DAGM 2007*, September 11, 2007.
- [8] H. Zhao and A. Sinha. An efficient algorithm for generating generalized decision forests. In *IEEE Transactions on Systems, Man and Cybernetics, Part A*, volume 35, pages 754–762, September 2005.