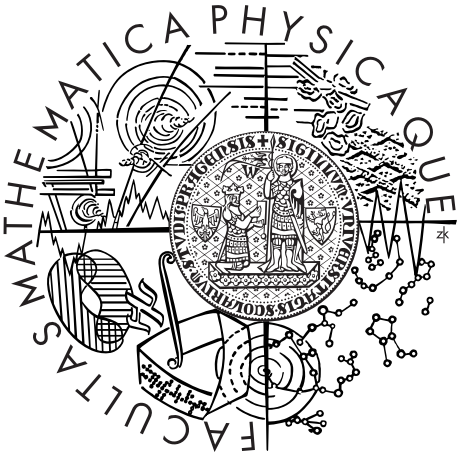
Charles University in Prague Faculty of Mathematics and Physics

**MASTER THESIS**




Matúš Dekánek

# Recognition algorithms for image viewers

Kabinet software a výuky informatiky

Supervisor of the master thesis: RNDr. Michal Šorel Ph.D.

Study programme: Informatics

Specialization: Software systems

Prague 2012

I would like to thank my supervisor RNDr. Michal Šorel, Ph.D. for all the valuable advice he gave me and his optimism. I also thank Matej Vitásek for his concern and help. And last but not least I must thank my parents for all the support during my studies.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague 12.04.2012

Název práce: Rozpoznávací algoritmy pro prohlížeče obrázků
Autor: Matúš Dekánek
Katedra / Ústav: Kabinet software a výuky informatiky
Vedoucí diplomové práce: RNDr. Michal Šorel, Ph.D.
Abstrakt: Nedávné období přineslo rozšíření digitální fotografie mezi jak amatéry tak profesionály. To s sebou nese také vznik velkých kolekcí fotografií. Tagování, neboli digitální označování, obrázků je jeden ze způsobů jak udržovat tyto kolekce uspořádané, je však časově náročné a tak mnoho uživatelů svoje fotografie neoznačuje. Cílem této práce je navrhnout algoritmus pro automatické označování obrázků za použití vizuálních znaků a modelu bag-of-words. Vyzkoušeny byly algoritmy pro získavání jak bodových (SURF) tak plošných (MSER) znaků využívajíc taky informaci o barvě. Ty se pak za pomocí zhlukovacího algoritmu CLARA převedou na "vizuální slova". Navrhnutý algoritmus běží v reálnem čase a v obrázku může detekovat několik objektů. Pro praktické užití by bylo nutné ho vyladit a ještě zvýšit úspěšnost, výsledky však naznačují možný potenciál.
Klíčová slova: označování obrázků, bag-of-words model, SURF, rozpoznávání obrazu

Title: Recognition algorithms for image viewers
Author: Matúš Dekánek
Department / Institute:
Supervisor of the master thesis: RNDr. Michal Šorel, Ph.D.
Abstract: Last decade brought wide spread of the digital photography among both amateurs and professionals, what resulted in huge photo collections. Image tagging is one way to bring an order to the photos, however doing this by hand is time consuming and most users simply do not tag most of their photos. This work aims on automated image tagging using the bag-of-words model. Attempts were made both with point-based (SURF) and area-based (MSER) visual features including also the colour information. CLARA clustering algorithm is then used to create a discrete mapping of the features to 'visual words'. Proposed object recognition algorithm can run in real time and is designed to recognize several objects on an image. The algorithm is currently not fit for practical use, as it would need higher success rate, but the results are promising.
Keywords: image tagging, bag-of-words, SURF, object recognition

# Contents

# 1 Introduction

## 1.1 Motivation

Invention of the digital photography together with the high capacities of hard disks brought the possibility to have photos in amounts not possible before. People have galleries of thousands of photographs, there are specialized programs just to manage picture galleries, thousands of photos are shared on the web image storages such as Picassa. One of the features of the image viewing software and web galleries is image tagging. Tag is additional text information that describes what is on the picture, be it objects, people, scenery, almost anything worth the photographer's attention.

If the user has most of his photographs tagged, he can really put the tags in use, he might for instance want to view only the images with a particular object. However tagging photographs by hand is time consuming. With several thousands of photos it is hard to have them all tagged and the user then of course looses the motivation to tag his images. This is where the automatic image tagging comes to use.

## 1.2 Goals

The goal of this work is to become familiar with image description methods and try to design an algorithm that would simplify the tagging of the images by detecting the objects. For this purpose both spatial information in the image and the file timestamp information will be used. We will not use methods based on spatial geometric information.

It should be noted in advance that this work is not focused on the face recognition. This field is already extensively studied and a serious work on this would require several years just to understand current development in the

field[1]. Nor is it focused on image categorization - this is again a different problem and used methods are not suitable for that.

### Typical use case

Typically a user has a few hundreds or thousands of photos of various content. We may suppose that a non-trivial part of them is tagged. Then the user adds to his collection several new photos, some of them can be photos of already known objects. Then the user wants to have new photos tagged. He could let the program guess the tags of the new images from the information it already has. Or he may tag a few of the new images first, mainly those with not yet known objects and then let the program guess the tags on the rest of the images in the collection.

## 1.3 Analysis and requirements

The program will have to quickly guess tags for a new image using the stored information from the already known images.

It is obvious that the program will have to hold a database of the already known images and tags associated with them. In addition to these data, the program should also store data extracted from the images and cached learning data. The algorithm will use the spatial information calculated from the images. Obtaining these information is difficult and time-consuming, it is therefore clear that it is better to store such information instead of calculating it each time it is needed.

### Problem formulation and decomposition

The algorithm will be given n images $I_1..I_n$. On each image are objects that represent the tags the photographer has assigned to the picture. The image $I_1$ contains objects $O_{1,1}..O_{1,m_1}$, the image $I_2$ objects $O_{2,1}..O_{2,m_2}$ etc. The same object can occur in more images. The algorithm is then given a new image $I$ and it should decide which objects from $O_1..O_n$ the image contains. The algorithm may use the information about objects detected in previous image(s).

The problem can be decomposed to several sub-problems:

[1] see http://www.face-rec.org/

- image and visual objects information representation and
- probabilistic model that would describe the problem based on the extracted visual information.

Design of the algorithm should also take into account that all these tasks need to be executed in a reasonable time. The size of the database with description of known images should not exceed the size of the images.

# 2 Approach

## 2.1 Related work

The field of digital image processing is immensely large and automated image tagging or annotation is just a small part of it. However, a lot of work has been done on this field in last decades.

Studied field is closely related to the field of the image retrieval. In the surveys [6] and [7] the authors mention lot of works and trends in this area. Both surveys put strong emphasis on image feature retrieval, in [6] highly focused on area and texture description. Both surveys mention advanced probabilistic models and algorithms such as multi-resolution hidden Markov model [20]and machine learning systems such as support vector machines (SVM) [4].

We will mention just a few works closely related to ours. Algorithm proposed in [13] extracts colour and texture features from the image. The texture information is retrieved using Daubechies 4 wavelet transform [8]. The feature descriptors distributions for each category are clustered using k-means clustering [14]and from these are created the concept models used to identify particular objects.

Wang et al. in [19] again proposed features extracted by a wavelet transform. For statistical modeling it proposes 2-dimensional hidden Markov model.

This work is also inspired by [16] and [3]. These works are more related to the scenery recognition problem however they contain several useful ideas.

Níster and Stevenius in [16] proposed iteratively built vocabulary tree, the algorithm proposed by [3] uses histogram based maximum likehood method based on the bag-of-words (also explained by Tirilly et al. [18]) probabilistic model of visual features extracted from an image.

## 2.2 Image retrieval and object recognition problem

Our design similarly to many other related works is based on typical work flow of first extracting visual features from the image (or in other words describing the image for the computer) and then creating a statistical model for the visual features.

The extracted features and their representation should be repeatable, quickly extracted and invariant to most the common image transformations such as scale change, rotation or affine transformation. This is further discussed in Section 3.1. We tried to work with both keypoint and area descriptors.

Paper [3] proposes for feature processing the bag-of-words model. This model understands the image as a set of extracted features, not considering their mutual position. We did not decide for a machine learning system such as the SVM or a neural network because that would require a lot of training input which is unlikely to be supplied by a typical user[1].

To apply this model we must be able to compare descriptors. Most of related works do this by assigning descriptors to the equivalence classes, many of them using some clustering algorithm. This is further discussed in Section 3.2.

The statistics used on the images is based on descriptors relation to a particular object. The guessing algorithm is trying to find subset of descriptors fitting to a known object as further explained in Section 3.3.

## 2.3 Using non-spatial file information

Besides the spatial information from the image this work also aims to to use non-spacial information from image file context; if an image contains a certain object, the next one taken a few seconds after is likely to contain it as well. This can be combined with image retrieval algorithms to further increase efficiency of the algorithm.

Most people do not adjust most of the photos they have taken and therefore the timestamp is unchanged, what makes it a good information source for probabilistic analysis of a photo collection. Many images also contain the EXIF tag that might contain the same information, and typically is not changed even when editing the image. It is simple to design variety of quite straight-forward probabilistic models that would put this information to use, some of them being discussed in Section 4.

[1] see Section 1.2

# 3 Object recognition problem

This chapter discusses the solutions for the object recognition problem in the images. It is partly influenced by papers [3] and [16] but our problem is more complicated, because mentioned works recognize the scenery while our algorithm is expected to recognize the presence of an object or objects. Also on the input the algorithm may get more objects per an image. That causes further problems, because there is no general way to tell which part of the image relates to which object.

The problem can be decomposed into several parts:

- image description and representation by feature descriptors,
- comparing and grouping the descriptors and
- probabilistic model, based on the bag-of-words concept.

These are to be used as described on Figure 3.1.

As was mentioned earlier, we did not use machine learning algorithms. This is due to the limited user input that would be used for learning. Typical self-learning algorithm would then have problem with correct generalization. Also the fact that each image can have multiple tags would lead to even higher complexity of these methods.

## 3.1 Image and object description

Objects that are to be tagged are hard to define: it can be a building, a tree or almost anything, it only has to be significant to the user. It is even harder to define them in the image - the same object can be larger or smaller on to different photos. It is nearly impossible to decide what is an important object on the image and what not, even though there were attempts to do so such as method proposed in [9]. And it still does not solve the problem of the object

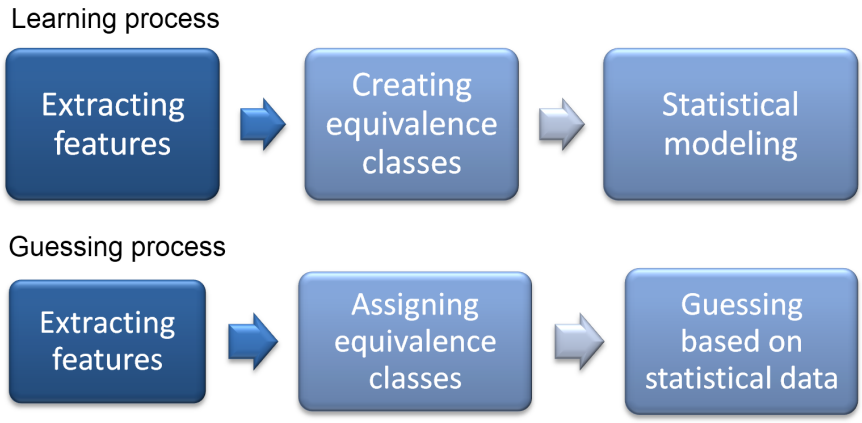



Figure 3.1: Learning and guessing algorithm design

representation as such. To describe contents of an image we need to extract the visual features and represent them, if possible in a way that would be invariant to most common image transformation. And of course these features are needed to be extracted quickly, typically in a grade of seconds.

Visual features can be in general divided to points of interest and areas. Interest point-based features are extracted and described by algorithms such as SIFT (see [17] chapter 4.1.2 (page 223) or Section 3.1) or SURF([10] or Section 3.1). There are also algorithms to extract areas such as MSER([17] page 220 or Section 3.1), which are used in image registration [2].

### Common techniques

These algorithms use several techniques that should be explained or mentioned before describing the algorithms that use them. It is expected that the reader is already familiar with the term convolution. This is widely used in image processing area. Especially the convolution with Gaussian kernel will be often mentioned and used.

Laplacian of the Gaussian kernel, usually addressed as Laplacian of Gaussian (LoG) is curl of the 2nd differential of Gaussian. It is used to find peaks of differential image which are usually on corners and edges. These are later used as points of interest or keypoints. Approximation of the LoG is Difference of Gaussian (DoG) which is a kernel representing difference of two Gaussian kernels with different sigma parameter.

To achieve scale invariance the keypoints are being found in several scales. The keypoints found in the scale space are then containing information about both the position in the image and the scale. To avoid creating the image in different scales, different sizes of the same convolution kernel is used instead, for example the Gaussian kernel with different sigma parameter.

The important property of image feature descriptors is repeatability. Keypoints and descriptors extracted from two images of the same object photographed from only slightly different position should be very similar. To achieve this the keypoint positions are usually calculated with sub-pixel accuracy using numeric techniques based on Taylor expansion.

### Descriptors of keypoints

Both SIFT and SURF are algorithms to extract and describe points of interest from a picture.

The SIFT algorithm first creates scale space of images blurred by convolution with different Gaussian kernels. By subtracting the values of the neighbour scale images the Difference of Gaussian(DoG) images are created. Keypoints are then identified as local maxima/minima in scale space of DoG images. This way the scale-invariance is achieved. The keypoints are then localized with sub-pixel accuracy using quadratic Taylor expansion of the DoG image. If the offset of the interpolated position from the original position of the keypoint is larger than 0.5 pixel, the keypoint is rejected as unstable. Also the low contrast keypoints and the points on the edges are discarded.

Each keypoint is then assigned orientation from the surrounding points. Gradient directions of the surrounding points are added to a histogram with 36 bins, taking into consideration also the magnitude of the gradient. Peaks in this histogram are said to be the dominant orientations. For each dominant orientation is created a keypoint with the same position. The keypoints are then described. Surrounding of each keypoint is divided into 4x4 16 pixel squares, for each this square is created 8 bins orientation histogram with gradient directions of points in each 16 pixel square creating a vector with 128 elements. These values are then normalized to reduce effect of illumination changes. Resultant descriptors are scale and rotation invariant and are robust even to affine transformation while being one of the most distinctive known descriptors.

SURF descriptors are an alternative to SIFT descriptors. The main differ-

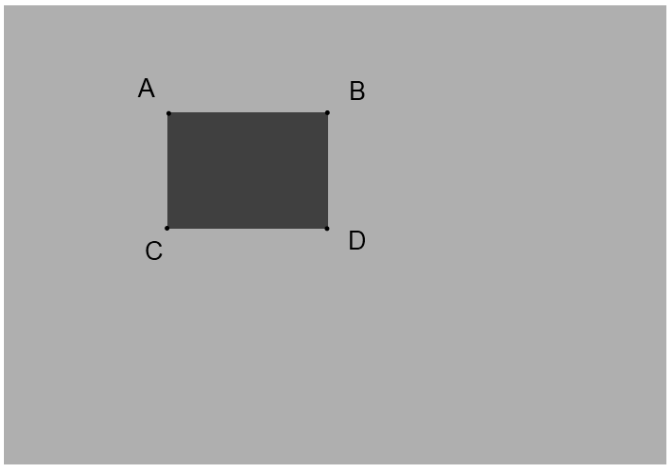



Figure 3.2: The sum of values in the rectangle area is A + D -(C + B)

ence from SIFT algorithm is that the SURF descriptor extraction algorithm uses the integral image [5] created from the original one and is based on the 2D Haar wavelet responses while the SIFT extraction algorithm uses the original spatial information and responses of DoG in scale space.

Integral image is used to rapidly calculate of upright rectangular area. Given the original image $I$ value of any point in the integral image $I_{\sum}$ is the sum of the values between the point and the origin. Calculating the sum of pixels in an upright rectangular area bounded by vertices A,B,C,D as in Fig 3.2 can be in the integral image performed in constant time by:

$$\sum = A + D - (C + B) \tag{3.1}$$

where each of $A,B,C,D$ represents value of the points in the integral image $I_{\Sigma}$.

Haar wavelets are here used as simple convolution filters that can be used to calculate gradient in particular direction. Their advantage for this algorithm is, that they fully utilize the capabilities of the integral image because they can be calculated as difference of summed pixel values in two upright rectangular areas.

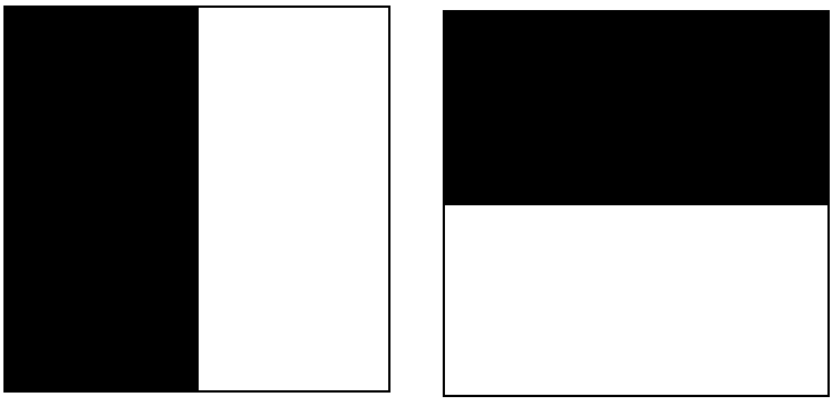


Figure 3.3: Haar wavelets calculating gradient in x and y direction respectively. Weights are 1 for the white and -1 for the black colour.

The algorithm first creates the integral image from the original one. The keypoints are then detected by a fast-hessian detector. This detector algorithm is based on the determinant of the Hessian matrix containing second order partial derivations of values in the image. Value of the determinant of the Hessian matrix, which is also the product of eigenvalues, is used to classify points - if the value is positive, the point is a local extreme; otherwise it is clearly not, because the eigenvalues have different sign. The values of the Hessian matrix would normally be calculated using Laplacian of Gaussian(LoG) or its approximation by Difference of Gaussians(DoG). SURF algorithm, however, calculates them by approximated convolution kernel consisting of a few rectangles, thus effectively using the properties of the integral image. Keypoint positions are then interpolated with sub-pixel accuracy using the quadratic Taylor expansion of the Hessian determinant. If the offset of the interpolated position is larger than 0.5 pixel the keypoint position is adjusted and the position interpolation procedure is repeated, until the offset is less than 0.5.

For each keypoint orientation is calculated. For points in the surroundings of the keypoints the Haar wavelet responses in x and y directions are calculated. These are then used to obtain the dominant orientation[1]. A square window in size corresponding to the scale at which it was detected oriented in the dominant orientation is created and divided into 4x4 squares

[1] for details see [10]


Figure 3.4: Description windows with size 20 times of the detected scale. The dominant direction is shown by a green line.

each containing 25 points. From each of these these are extracted four values $\sum dx, \sum dy, \sum |dx|, \sum |dy|$,thus creating a vector with (4x4x4)=64 values.

SURF algorithm is claimed to be significantly faster than SIFT while being more robust to scale change and rotation. It is not however designed to be invariant to affine transformation. Both these feature description algorithms are implemented in the Open-CV library[2]. There is also another open-source SURF implementation[3].

### Area-based features

The MSER features are areas that are supposed to be repeatably detected regardless of scale change and rotation. It stands for Maximally Stable Extremal Regions.

[2]http://opencv.willowgarage.com/wiki/
[3]http://www.chrisevansdev.com/computer-vision-opensurf.html

Figure 3.5: detected MSER in an image

A maximum intensity extremal region is a region in the image where for any point $p$ in the area and any point $q$ in the boundary of the area the intensity of $p$ is higher than the intensity of $q$. Similar definition is for the minimum intensity extremal region. Maximally stable extremal region can be then described as the one in the sequence of nested extremal regions with smallest relative boundary. Extraction of these features is considerably fast[4]. These areas can be described similarly as interest point surroundings in SIFT or SURF algorithms as mentioned in [11].

For the purpose of this work SURF descriptors were chosen, mainly because they can be calculated quickly, what is the key advantage for this work. This algorithm was enhanced to include colour information into the descriptor and the hue histogram of significant point surroundings (in scale-space) is taken. According to [3] this should significantly increase the distinguishing power of descriptors.

MSER algorithm was used too and an algorithm for their effective descrip-

[4] [15]

tion invariant to the most common continuous transformations was proposed[5]. However the results were better with the SURF descriptors.

## 3.2 Descriptor equivalence classes

A method for dividing a set of descriptors into equivalence classes is needed together with a method for assigning a descriptor to the correct equivalence class. That is because any probabilistic method based on the descriptors will need to match the known descriptors and the descriptors in an unknown image.

The method to create the equivalence classes should be fast and reflect the distribution of the descriptors in the space. It should allow to discriminate between objects in the images by assigning different equivalence classes to descriptors from different objects. The generalization ability[6] should be considered as well.

### Single descriptors

One way to solve this problem is to take each descriptor from known images as the only single representative of its equivalence class. The descriptors would not be matched at all in the learning phase. A descriptor from a new image will then be matched to the nearest known descriptor.

Using this approach we get excellent discriminability, however the generalization ability of this method is weak. Two almost equal descriptors on two different images will not be considered the same and if we then match a new descriptor to one of these, we will loose information about the second one. This is likely to negatively affect the results especially if in these two images were two different objects.

Another problem is related to the performance. The number of descriptors extracted from the images can easily exceed a million. That would mean more than a million equivalence classes. Each new descriptor in a new image would have to be assigned to the nearest known descriptor. Searching in all those equivalence classes for each extracted descriptor can last non-trivial time even if we consider a fast structure for searching in the equivalence classes. Not to mention the time required to create such a structure.

[5]There are many possibilities to do that, let's name just the possibility to copy the intensities of the points in the area after it's normalization. Used implementation is further explained in Section 5.3

[6]ability to recognize and group similar descriptors

### Clusters

Authors of both [16] and [3] proposed matching visual "words" by dividing the set of descriptors into several clusters, in other words by clustering. Many other works mention clustering as well. Known words can be split into clusters that would reflect their distribution in the descriptor space. This problem is well-known and there are suitable solutions. Both mentioned works used clustering into predefined count of clusters. Their results show, that about 10,000 clusters have good distinguishing power while still having reasonable generalization ability for teir purposes.

Clustering seems to solve most problems that would be encountered while trying to implement direct single-descriptor matching approach. Most known algorithms more or less reflect the distribution of descriptors in the descriptor space. Even the most basic algorithms give reasonably good results.

One of the simplest clustering algorithms is Loyd's algorithm known also as k-means clustering. This algorithm is iterative and partitions the clustered space into Voronoi cells[7]. It starts with fixed count of clusters. In each iteration it calculates center of each cluster an then assigns the nearest cluster to each point in the clustered set. If the assigned clusters did not change, the algorithm ends; otherwise the loop continues. It's main disadvantage is that it returns only constant number of clusters. This count can be estimated and it can still give reasonable results for some applications, but this could be problem in our problem where the image gallery could contain 100 images as well as 20,000. We would need algorithm that scales with the number of known descriptors.

Another problem is that partitioning something like a 1,000,000 items into several thousands clusters is incredibly computationally complex. In this most basic form this took (with 64-dimensional descriptors) several hours.

### Hierarchically organized clusters

In [16] the authors proposed clusters hierarchically organized into a tree-like structure. On each level the clusters are divided into smaller and smaller groups. This vastly increases the speed of the process. The paper[3] mentions further evolution of this algorithm, the Clustering LARge Applications (CLARA)[8] which takes only a small subset of the clustered points in the upper levels to avoid working with huge descriptor sets.

[7] see [1]
[8] described by [12]

**Algorithm 3.1** K-means clustering

```
input: Vectors set of vectors, K number of clusters
output: set of cluster means
Result <-- create K random vectors
ChangeOccured <-- true
while ChangeOccured do
    ChangeOccured <-- false
    foreach Vector in Vectors
        assign Vector to the nearest vector in Result
        if the assigned vector index is different then
            ChangeOccured <-- true
    foreach ClusterMean in Result
        recalculate position of ClusterMean from all
           vectors assigned to ClusterMean
endwhile
return Result
```

Our first implementation of the hierarchical clustering had constant number of levels and therefore created constant number of clusters.

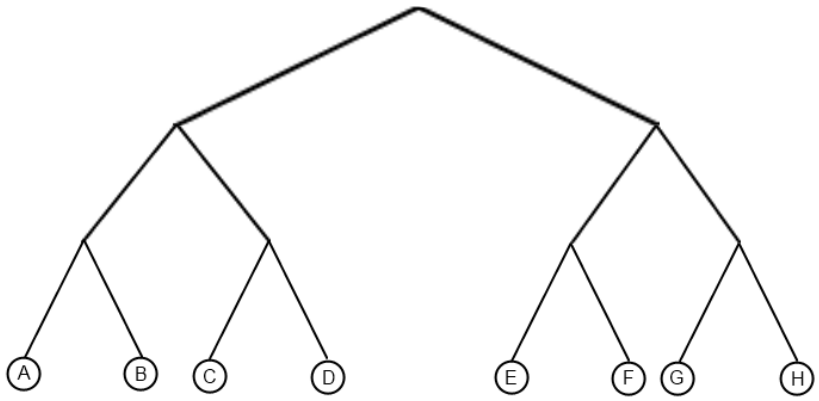



Figure 3.6: Hierarchy structure of clusters

**Algorithm 3.2** Hierarchical clustering

```
input: Vectors list of vectors, K number of divisions on
    each level, D depth of resultant cluster hierarchy
output: hierarchicaly organized clusters
if D = 0
    return mean of vectors
endif
VectorSubsets <-- array of K vector sets
Means <-- K-means clustering( Vectors, K )
foreach Vector in Vectors
    MeanIndex <-- find nearest vector in Means to Vector
    VectorSubsets[MeanIndex].add(Vector)
endforeach
Result <-- cluster structure
Result.Mean <-- mean of Vectors
for SubclusterIndex <-- 0 to K
    Subcluster <-- Hierarchical clustering(VectorSubsets
        [SubclusterIndex])
    Result.Subclusters.add(Subcluster)
endfor
return Result
```

### Distance from cluster mean

Once the algorithm created the equivalence classes of the known descriptors it will eventually be given a new image. This image will be described by a set of descriptors, some of them are representing already known features, some not. If the algorithm is given a new descriptor it will assign it to some cluster even if the descriptor is quite distant from the assigned cluster. Such descriptor would be considered to be as relevant as another descriptor that is fitting to its cluster much better. A method was needed to counter this effect.

This method should take into consideration the distance from cluster center and the distribution of the known descriptor within the cluster. One solution for this would be weighting the descriptor in the image by value between 0 and 1 given by a weighting function. The value of this function would rise with average distance of the descriptors in the cluster from the center and decrease with distance from descriptors distance from the center. A simple example of such a function might be weight multiplier equal to

$$\frac{1}{1 + dist//\sqrt{Var(dst)}} \tag{3.2}$$

This measure was implemented. In the end each descriptor assigned to a cluster has its weight and this is used in all calculations.

### Clusters with specified cluster size

When experimenting with various types of descriptors it was concluded that it would be much better if it was possible to set the maximum size of each equivalence class. When using this approach, a node in the cluster hierarchy has sub-nodes if and only if it contains a point(s) with distance from the mean greater than a specified threshold. This way the cluster hierarchy tree has no longer constant height and reflects the distribution of the clusters in the space much better.

The cluster structure also assigns a cluster to a descriptor only if the descriptor is near enough to the nearest cluster's center; otherwise no cluster is assigned to the descriptor at all. This approach eliminates outliers that would otherwise affect the outcome.

This solution scales much better with the number of clustered descriptors as shown in the Section 6.1. In our solution we use simple euclidean distance.

### Merging clusters

Practical tests have shown that the position of the starting clusters means may have significant effect on the results. It became clear that further analysis on the effect of clustering on resultant success rate of the algorithm should be done.

That has led to the idea that for each tag the descriptors should be clustered separately and then merged. This way the descriptors related to one object are kept together and more importantly it is probable, that they would remain separated from descriptors related to other objects thus increasing discriminability power of the equivalence classes.

When merging two cluster structures, all leaf clusters from one structure are iteratively merged into the other cluster structure. This is described by Alg. 3.3. For each cluster to be merged into a set of clusters a nearest leaf cluster is found. If these are too far from each other, a new node is created

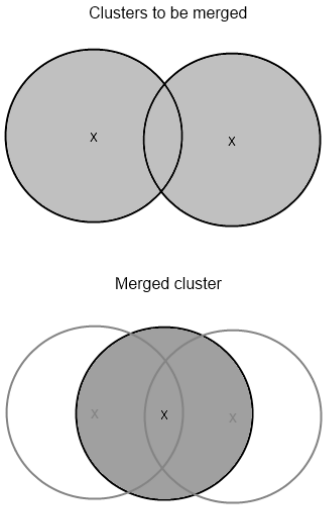



Figure 3.7: Merging will result in a smaller cluster effectively excluding previously assigned descriptors from the cluster.

in the tree cluster structure that will replace the nearest cluster. The merged cluster and the nearest cluster will then become it's sub-nodes. If the nearest cluster and the merged cluster are near enough, a new cluster is created to replace the nearest cluster.

However this may cause that resultant merged cluster set will not cover all the descriptors that were covered by the original sets of clusters, as shown in Fig. 3.7. To compensate this, after the merging of all cluster sets created for each known object also each descriptor has to be merged alone as a stand-alone cluster to make sure that each descriptor is covered by a clusters in the cluster set.

This clustering method performed well on smaller data sets, but the process of merging took too much time on larger image sets.

**Algorithm 3.3** Cluster merging

```
input: This cluster set, Other cluster set to be merged
output: there is no output, clusters are merged into
    This structure
foreach Cluster in Other begin
        MyCluster <-- This.nearestLeafCluster(Cluster)
        if MyCluster.Mean.distance(Cluster.Mean)>This.
            ClusterSize begin
                NewCluster <-- copy of MyCluster
                MyCluster.Mean <-- mean of NewCluster.
                    Mean and Cluster.Mean
                MyCluster.addChild(NewCluster)
                MyCluster.addChild(Cluster)
        end else begin
                MyCluster.Mean <-- mean of MyCluster.
                    Mean and Cluster.Mean
        end if
end foreach
```

## 3.3 Probabilistic model

A probabilistic model was needed to assign a tag to a new image represented again by a set of visual features. It was decided to use the bag-of-words model. This model is used in the text analysis to match documents and topics. It considers a document as an unsorted set of words which may be assigned a certain topic(s). Once there is an efficient way to extract visual features from the images it can be aplied to the image-tag matching problem as well, because then we can consider each image a document and each visual feature a word.

There are however several drawbacks to this model. The bag-of-words model does not take into account the position of the words. On the other hand this can be also an advantage - working with position in the image is not as simple as in a sequence of text. There are some algorithms and methods to work with the position but in general it can be assumed that they are not simple. Also these algorithm will become even more complicated if the aim is to detect more objects in each image. We therefore decided not to include in this work algorithms using the geometric information of the image features.

Using bag-of-words model clearly leads to use of the Bayesian statistics.

Let's have a set of images $I = \{i_1..i_n\}$. Each image $i_j$ is said to contain $m$ objects from the set of known objects $O = \{o_1..o_k\}$ (related to the tags) and

a set of descriptors $D_j = \{d_{j1}..d_{jm_j}\}$ has been detected on it. Generally it is not possible to decide which descriptors from $D_j$ are related to one particular object contained in the image, especially if the image is known to contain more objects. The best assumption that can be done is, that each descriptor from $D_j$ is with probability $p = 1/m$ related to a particular object object $o$ contained in the image.

### Count-based solution

Let $M$ be a matrix of weighted counts of occurrence of descriptors $d_1..d_N$ for objects $o_1..o_k$. These counts will take into account the probability that the descriptor is related to the object meaning that if there are $k$ objects in an image $I$ then each descriptor found on this image is in the matrix $M$ counted as $1/k$ for each of the objects. Each column vector in this matrix representing a certain descriptors relevance distribution is normalized so that the sum of this vector's values is 1. This way each such normalized vector represents estimated probability distribution of the relevance between the descriptor and the objects.

This way we can obtain estimation of the probability distribution of the relation of each descriptor to the known objects. Using this information the algorithm should decide whether an image represented by a set of descriptors $D$ contains particular object $o$. The number of objects in the image is unknown, the number of descriptors in an image is variable. A scoring system independent of count of descriptors or objects is needed. This scoring system must also reflect the probabilities that each descriptor is related to an object with a certain probability.

The simplest way might be to calculate likehood score for each object and pick one or a few most probable objects or pick those whose likehood is more than a specified threshold, which would depend on the maximum likehood score. This method is lacking some elegance, but it is invariant to the count of descriptors and objects. The main drawback is the threshold and the fact, that it actually does not give the answer to the question "is this object there?". It more likely answers the question "what objects are the ones most likely to be in the image?", what implies that there surely is at least one known object in the image. That does not involve possibility that there are no known objects. It could be possible to solve each of these problems, but in the end we would have an algorithm that has probably nothing common with the original idea of maximum likehood as such and definitely would be overly complicated.

**Algorithm 3.4** calculateRelevances

```
input: M relevance matrix, Indices list of descriptor
    indices with their weights, tagCount supplied by M
output: vector of relevances to the tags
    Result <-- vector with tagCount values set to 0
    foreach WeightedIdx in Indices
        DescriptorRelevances <-- M.getColumn(WeightedIdx
            .index)
        DescriptorRelevances.MultipleBy(WeightedIdx.
            weight)
        Result.Add(DescriptorRelevances)
```

Another approach was proposed instead. The algorithm should not directly say which objects are the most probable ones to be on the image but confirm or deny object's occurrence in the image. In other words it should give answer to the question "is this tag in the image?".

**Algorithm 3.5** guessTag

```
input: Indices list of descriptor indices with their
    weights, M relevance matrix, tagCount supplied by M,
    TresholdRatio
output: guessed tag index or NULL
    Relevances <-- calculateRelevances(Indices, M,
        tagCount)
    MostRelevant <-- index of the highest value in
        Relevances
    MaxRelevance <-- Relevances[MostRelevant]
    SecondMostRelevant <-- index of the 2nd highest
        value in Relevances
    SecondMaxRelevance <-- Relevances[SecondMostRelevant
        ]
    if MaxRelevance >= TresholdRatio*SecondMaxRelevance
        return MostRelevant
    else
        return NULL
    end if
```

In each of the images there are descriptors that are related to an object that is in it and then there are descriptors related to its surroundings or to other objects on the image. We presume that similar objects will contain similar

descriptors that can be distinguished from other descriptors. We may then assume that if an image contains a certain object then the set of descriptors that represents this image will contain a subset, that is related to this object far more than to any other known object. In other words, the proposed solution is based on finding the subset of descriptors "most likely related" to a certain object.

Unfortunately the size of such subset cannot be estimated exactly, even relative to size of descriptor set for one image. The object can be bigger or smaller and also its complexity may vary. Both of these properties contribute to the resultant count of descriptors describing it.

Fortunately this is not as significant problem as it looks like. Photographers have their habits of taking photos and therefore the objects size will not vary too much. More importantly, they do not usually take photos of more than two significant objects at once. And of course, not all of the descriptors describing an object are needed, a fraction of them with non-trivial size would suffice as well. Therefore a good estimate of the most-relevant subset size can be done for most cases.

The algorithm first sorts the descriptors from the image according to their relevance to a certain object. Then it selects the first $m = n/k$ descriptors where $n$ is the number of the descriptors in the image and $k$ is a parameter for the algorithm. Then the likehoods for all known objects are calculated. If the likehood of the examined object is far higher than any other then the object is accepted. Otherwise the most related not yet inserted descriptor is inserted into the subset and the procedure of calculating likehood is repeated again. This is iterated until the object is accepted or all descriptors have been inserted into the subset[9].

There are two parameters that have to be set - the threshold difference between likehoods to accept an object and the size of the starting subset. It is not as much a problem even if it makes the concept unclean because of the dependence on a certain parameter. But this is acceptable and the parameters can be experimentally set to fit the real world data.

This method can be prone to have many false positive hits, because it just allows it. The simplest maximum likehood method can have only limited count of false positive hits, because it guesses only one object. On the other hand its

[9]it was expected that the iterations would solve special situations where two different objects would share several descriptors. However it appears that such cases are very rare. If the object was not initially accepted then it was rejected in most cases

**Algorithm 3.6** isTagThere

```
input: Idx tag index, Indices weighted descriptor
    indices, M relevance matrix, R treshold ratio, K set
    division parameter
output: boolean
    sort Indices by their relevance to tag Idx using M
    N <-- Indices.size/K
    while N < Indices.size
        Subset <-- first N indices from Indices
        Relevances <-- calculateRelevances(Indices, M,
            tagCount)
        TagRelevance <-- Relevances[Idx]
            SecondMostRelevant <-- index of the 2nd
            highest value in Relevances
        SecondMaxRelevance <-- Relevances[
            SecondMostRelevant]
        if TagRelevance > R*SecondMaxRelevance
            return true
    end while
return false
```

higher acceptance allows it to find also the correct objects that the maximum likehood method would not find.

Another important matter to discuss is, whether such solution is acceptable in the means of performance. The computational time of the process of guessing the tags on an image should not be in linear or worse dependency with the number of descriptors in the database.

Considering that user has about 1,000 tags in the database and a few thousands descriptor equivalence classes, his descriptor-tag relevance matrix will have dimension of about 1,000x10,000. New image has a bit less then 1,000 descriptors. Using the right structures each descriptor can be assigned to a known descriptor equivalence class in logarithmic time (to the count of the descriptor equivalence classes), thus this is not an issue.

Building the most relevant descriptor subset of the descriptors extracted from an image for each possible tag is even with used heuristic unacceptably slow. If we define the maximum size of the most relevant descriptor subset, we can significantly increase the performance. The first guess might be, that the maximum size of the subset is 100. This will have only negligible effect on the accuracy of the algorithm – if the subset of maximum size is not “relevant”

enough, it is unlikely that a bigger subset will.

For a simple guess on one image only the "most related subset" method was used. However for solution that included also file timestamp was used the maximum likehod method as well as explained in section 4.

# 4 Using timestamp information

The algorithm already described is capable to guess tags in an unknown image using only the spatial information. Or, more precisely, it is designed to answer the question "is this object in this image?". However, user might want to tag a whole set of photos. These photos may or may not have a relation with previous ones. More importantly, they are taken in a time sequence and usually there is a relation between an image and an image taken shortly after it. This information can be used to achieve higher success rate of the algorithm while keeping the false positives rate on a reasonable level.

Using this information is meaningful only when guessing tags for a whole new image collection. Using it for single image guessing might cause that the new image will be put into a completely unrelated image sequence, what would lead to wrong results.

It should be noted that this approach will surely put objects that are not photographed in a sequence in disadvantage. The algorithm should be self-adjusting for the user that uses him - some users may take 20 or more photos of an object in a row and some may take at most three photos of the same object.

## 4.1 Usage of already designed algorithms

So far the algorithm design was highly modular - it is possible to choose from a variety of feature extraction algorithms, the descriptor equivalence problem is solved as an independent problem as well as the the descriptor-object relation mapping. The solutions to these sub-problems were partially chosen with respect to each other, but as such they do not depend on each other.

To sustain this modularity it was decided to incorporate the timestamp information processing independently of the spatial data. Current design allows

to use several variants of the image recognition algorithm and to combine them with each other and with the timestamp-based processing methods.

## 4.2 Approach for the timestamp processing

The experiments with proposed algorithm have shown that the spatial information based algorithm is not perfect and it either has low recall or low precision depending on its configuration. If the algorithm is set to refuse most tags in an image, then it is likely to guess the tags correctly, but it will refuse most tags leaving the new collection correctly but only sparsely tagged.

On the other hand if the algorithm is set to accept tags it will guess the correct tags on most of the images, but it will also have many false positives. We needed a way to decide in advance which tag should even be considered on an image before running this algorithm. That would significantly lower the false positives rate.

To do this, the timestamp information is used. The algorithm will be given a sequence of images and their tags and the next image in the sequence and it will decide whether possibility that an object is there should be considered.

The algorithm will first use the guessing algorithm with low acceptance setting on all images[1]. This will result in a few newly tagged images with relatively few mistakes. Then the algorithm will go trough all the new images sorted by the timestamp and for each decide whether to try the guessing algorithm with high acceptance rate or not.

## 4.3 Assumptions

We assumed that there are no timestamp collisions between photo collections of the user. That means users collections can be, after being sorted by timestamp, distinguished purely by specifying the right timestamp interval. This is important, because to study and process the photo and tag sequences it is needed not to have sequences mixed together.

This assumption is not always met in the real world - user may for example collect photos from several people that were on the same trip or occasion. However this is more problem of final application design than a research problem.

[1] This algorithm is also combined with the maximum likehood algorithm to further increase the precision.

The workaround that would take this into consideration is not complicated[2], but it would unnecessarily increase the complexity of the design process.

Another assumption is that timestamps of the photos are really related to the time when they were taken. This can be expected to be met in real world. Timestamps usually represent the last file modification and in most cases it is the time of creating the photo. If the user has changed the photo, most probably by adjusting it in a graphic editor, he will most probably take the effort to tag the photo by hand.

## 4.4 The decision algorithm

Using the information about the previous images in the sequence and objects in them the algorithm should decide whether the next image in the sequence is likely to contain a certain object or not. For each object the algorithm sees the sequence of the images as a sequence of boolean values representing occurrence of the object in an image.

The relationship between occurrences of different objects are not considered. It is unclear whether such approach would be really effective. Moreover, a new image collection usually contains new images with not yet known objects and thus the algorithm does not have enough information about their occurrence relations.

There are many models and many statistics that can be used to solve this problem, some more sophisticated, some very simple. Because the problem is based on a sequence of boolean values, it leads to Markov chain model with two states isomorphic to the boolean states of occurrence of an object in the sequence of the images. Or it may use the statistics for time difference between subsequent images containing the object and between an image that contains the object and a subsequent image that does not. Or statistics of the length of an uninterrupted sequence of images that contain the object.

Problem with discrete statistics is that they would consider the gap between two neighbouring images in the image sequence to be the same regardless of the timestamp difference which could be one minute or one day. Thus any used model should be either directly built on the timestamp difference or altered to take them into consideration. However such a change in originally discrete

[2]Instead of analyzing sequence of all known photos, the algorithm may analyze only sequences representing the collections.

model would make it much more complex while the real positive effect to the performance cannot be estimated.

It was therefore decided to consider the timestamp differences between images. A statistics can be created for the differences between files that contain the object and for the differences between the files that contain the object and subsequent images that do not. This statistic is very simple and still effective.

From these statistics a threshold that will be used to decide between the answers will be calculated. First the time differences that will be used in the statistics are collected. There are two types of the differences - between subsequent files when both files contain the object and between image that contains the object and subsequent one that does not. They are then sorted and the time difference interval that best distinguishes between the mentioned difference types is chosen. The threshold will be the middle of this interval. Then when assigning tags to a new image, if the last image contains the tag and its time difference from the new image is less than the threshold the algorithm will give the answer *yes*.

This timestamp difference can be taken for each tag separately. If there is not enough data for such a statistics for a particular object, average values are used instead.

# 5 Solution implementation

The algorithm has been implemented in a form of a dynamic library, so that it can be used by various programs. The interface had to be simple and straightforward, and the internal structure should allow to modify one part of the algorithm independently of the others.

Also a technology demonstrator in a form of a simple image viewer was created to demonstrate the use of the library. This is described in the appendix Section B.

## 5.1 Platform and libraries used

We decided to implement the algorithm in C++ language. Programs written in this language are fast and the language provides object oriented programming structures. It is compiled by the open-source GCC compiler.

The library and the technology demonstrator are implemented for Windows platform. It uses Qt toolkit[1] and open-source image processing library OpenCV[2], thus porting to other platforms should be simple. However, that was not tried yet.

Qt is a multiplatform toolkit intended originally for GUI programming but currently implementing a variety of functions that are otherwise different on each system. It was decided to use this library to load image files even in the algorithm implementation library. This makes sure that any image that will be loaded by a Qt application[3] is also loaded by the algorithm. It is also better to use only one library for a certain functionality in entire project.

OpenCV is an open-source library aimed at image-processing. It implements many various image processing algorithms from Fourier transform to

[1] http://qt.nokia.com/products/qt-sdk
[2] http://opencv.willowgarage.com/wiki/
[3] such as the technology demonstrator

feature extraction. Our implementation is however trying not to depend on this library in the future because it forces developers to download and build yet another extensive library while using only a small part of it.

## 5.2 Library modules

The implementation covers all the mentioned algorithms parts into a library that offers a simple and user friendly interface and handle tasks such as storing the image and descriptor information.

The sources of the library are divided into several namespaces each representing different functionality.

## 5.3 Descriptor extraction implementation

Several ways of image feature extraction based on SURF and MSER extraction were implemented or used. Because it was not clear which descriptor implementation would be the most suitable one, several variants of each were implemented.

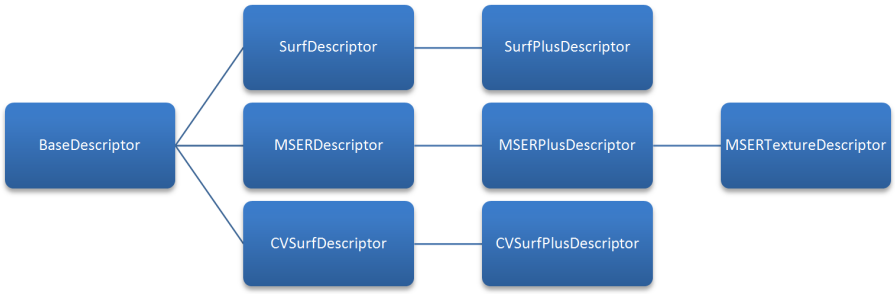



Figure 5.1: Class diagram of implemented descriptor algorithms

### BaseDescriptor

This is just a simple abstract class that specifies the interface for other descriptor classes. Each descriptor class has a static method to extract descriptors from an image and also methods accessing the descriptor values and

### SURF implementations and its variants

The *SurfDescriptor* is the basic SURF descriptor extractor. It is based on the OpenSURF implementation, but it was completely rewritten to make its interface friendlier and more object-oriented. Also it does much less rely on the OpenCV library and uses Qt library to load the image. Original implementation uses threshold to filter low-response interest points. We added a tweak to filter from these also the points whose response is less than the average response. This lowers the count of extracted descriptors by 2/3 and slightly improves the performance.

Original SURF descriptor does not take into consideration the colour information. According to [3] the colour information can be used to significantly increase the discriminative power of interest point surroundings descriptor, while its computation cost can be almost neglected compared to the calculation of SURF descriptor. The *SurfPlusDescriptor* implementing this enhancement creates 9 values hue histogram of the 10x10 pixel surrounding of the interest point.

The *CVSurfDescriptor* is a wrapper class over the OpenCV SURF implementation which was used to compare the implementations and their results. The *CVSurfPlusDescriptor* is evolution of this class adding the hue histogram of keypoint's surroundings. These OpenCV wrapper classes seem to be faster (see Section 6.1) than our implementation. Originally they were only created to compare them to OpenSURF based classes, but currently we use them in the final solution.

### MSER implementation

The MSER algorithm is an algorithm to extract an area in the image, but as such it does not describe the area. This had to be developed, either from scratch or by using already known algorithms as suggested by [11]. It was decided to try to propose and implement a new description algorithm that would be straightforward and simple yet with useful properties such as invariance to rotation and scale. If that would give reasonably good results hopefully outperforming the SURF descriptors then implementation of a more elaborate algorithm using techniques from SIFT or SURF description algorithms would have sense.

It was decided to describe the area without its surroundings. This is supposed to be one of the advantages - it describes a part of an object and not its

surroundings, so that the object can be recognized even with different background. The point-based descriptors such as SURF and SIFT are describing the surrounding of the interest point so they may often depend on the background.

First of all the area has to be normalized (see fig. 5.2). The same area may be of different sizes and photographed under different angles, but it should be repeatably detected and described. For this we first calculate the position of the centroid of the area $C$. From this we find the most distant point and the direction from the centroid to this point will be used as the area's orientation. The distance between these points is $d_1$.

Then the most perpendicularly distant point from the line passing trough the centroid and the most distant point is found. The perpendicular distance between this point and the line is $d_2$. After that a rectangle centered on the area's centroid with dimensions $(2d_1) \times (2d_2)$ oriented from the centroid to the most distant point is taken from the image[4] using a simple bilinear resampling for the rectangle rotation. This ensures the invariance to rotation.

After this an integral image from the extracted rectangle is created. This simplifies achieving scale invariance because no rescaling of the rectangle is needed. The contents of the extracted rectangle is then described.

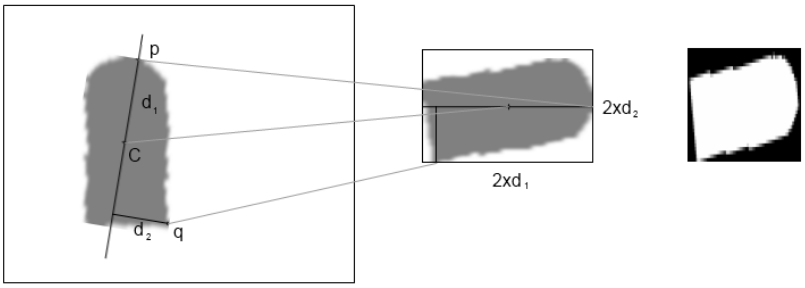



Figure 5.2: Area normalization

First the centroid of the area C is found, then the most distant point p and point q most distant from the line passing trough C and p. These are used as boundaries for the extracted rectangle. The rightmost image is the area normalized to a 100x100 image.

The simplest method is to divide it into NxN[5] squares and use the summed

[4]or the binary image containing only the area, as is explained later
[5]N being some reasonable constant chosen so that the descriptor is not too large

intensities in them as values of the descriptor. The values are of course normalized, because they would be otherwise affected by the size of the area. Another method is to calculate the Haar wavelet responses in the middle of each of the NxN squares. Both these variants were implemented together with some attempts to combine them.

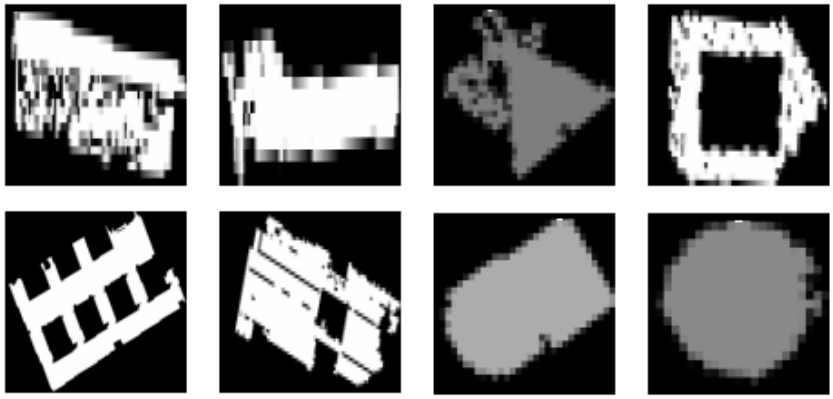


Figure 5.3: Extracted and normalized MSER areas

The first implemented variant was the MSER shape descriptor (class *MSERDescriptor*) which was taken from a binary image containing only the extracted MSER area. Similarly to the SURF-based classes it was decided to extract the colour information. From the pixels in the original image belonging to the area is extracted colour hue histogram. This histogram is then added to the original MSER descriptor (class *MSERPlusDescriptor*). The last variant describes not the MSER area shape, but the (greyscaled) pixels from the original image and adds also the colour histogram information (class *MSERTextureDescriptor*).

It was expected that these descriptor will preform similarly to the SURF descriptors with potential to even outperform them. However, the experiments have shown that these descriptors have been outperformed by the SURF descriptors (see Section A.1) and therefore they were not used. However their implementation has been left in the library for a case a new, much more efficient description method is proposed.

## 5.4 GenDesc module

This module contains the algorithms for clustering and statistical processing of the descriptors. It also contains a standalone descriptor structure independent of the descriptor structures in the image description module. For most structures it also implements the methods for serialization, so that they can be stored into a file.

### Clustering

All clustering algorithms use the k-means clustering algorithm. This algorithm does not require any additional classes, it only returns a list of cluster means. It works in several threads to take the advantage of multi-core processors. Even with this tweak the algorithm was too slow for the hundreds of thousands of descriptors extracted from the images in a small gallery.

Base class for clustering algorithm classes is the *AbstractClustering* class, which is an abstract class specifying the interface for other clustering algorithm implementing classes. This includes mainly the methods to assign a cluster to a descriptor and to code the clusters into a buffer or decode them from a buffer.

*HierarchicalClustering* is a class implementing basic hierarchical clustering algorithm as described in the Section 3.2. On each level it uses the basic k-means algorithm. It also can assign a weight to a descriptor, so that the descriptor weights mentioned in the Section 3.2 can be used.

The *HierarchicalEquivalenceClustering* contains an implementation of the clustering algorithm with specified maximum cluster size (explained in the Section 3.2). The *MergeableHEClustering* contains the same data as the *HierarchicalEquivalenceClustering* class, but contains methods to merge two clusters together and to merge a cluster with a single descriptor. Another evolution of this structure is *MergeableHECLARA* which builds the cluster hierarchy using the *CLARA*[12] algorithm increasing the speed of the process.

### Count and relevance matrix

A special class to hold matrix data and process them was implemented. For various reasons it is named *IncidenceMatrix*, but the user should use it as any other matrix. Except for the fact that it has several methods used to transform it into transposed matrix normalized either by columns or rows. This is used to transform the count matrix into the maximum likehood ma-

trix (*createMaximumLikehoodMatrix()* method) or into the relevance matrix (*createOpositeDirectionMatrix()* method).

The semantics of this class is that the count matrix is an $n \times m$ (columns and rows respectively) matrix where $n$ is number of descriptor equivalence classes and $m$ is the number of the known objects. The relevance and maximum likehood matrix is then obviously an $m \times n$ matrix.

The class also implements methods to calculate likehood scores for a histogram of descriptor equivalence classes (*getLikehoodProbabilities*) or weighted row set sum (*sumOfWeightedRowSubset*) used in the bag-of-words algorithms.

### Bag of words algorithms

There are several important functions named *isTagThere*. These are implementation of the algorithm based on constructing the most related descriptor set (explained in Section 3.3). It contains also the same algorithm using weighted descriptor relevances and also another algorithms explained later in Section A.2 and A.3 that are not currently used at all.

### Probability module

The probability module contains the algorithm to calculate the threshold timestamp distance mentioned in Section 4.4. It is not a part of the *genDesc* namespace. Originally it was planned to implement several algorithms and therefore a separate module was needed, but finally only one algorithm was implemented.

## 5.5 Image database library

This is the module that uses all the previously mentioned modules and puts them to use. It is responsible for storing and loading the information about known images, gathering descriptors from images or guessing the image tags using the other modules.

The *ImageInfo* class contained in this module holds all needed information about an image such as file name, timestamp, assigned tags and most importantly the descriptors.

The most important is the *Gallery* class. It is responsible for loading and storing the image database information. Besides the image information stored in the ImageInfo objects, it contains the clustering (currently the *MergeableHECLARA*), the descriptor-tag count matrix, relevance matrix and maximum

likehood matrix and some more data used by tag-guessing algorithms. And of course, all the functionality to insert new images, tag them, guess tags on them and to learn from gathered information.

# 6 Results

## 6.1 Descriptor extraction and clustering

The extraction of the descriptors from an image and their clustering are the two most time consuming tasks in the solution. All the other tasks are executed fairly quickly in comparison with these.

### Descriptor extraction

We measured the time needed to extract descriptor from images and also the number of extracted descriptors. If different descriptors had similar performance for tag guessing, this would be the next important criterion to consider when choosing the right implementation.

To reduce measurement errors the files were first loaded into the memory and resized[1]. Only the subsequent descriptor extraction process time was measured. We measured the time to extract descriptors from image sets with of 1 to 25 images.

It is important to mention that the time needed to extract descriptors from a real image would include also the time needed to read and parse the image file. That however depends on the size of the image and image parser implementation (in our case the *qImage* class in Qt framework).

The measurements were taken for SURF[2] and MSER descriptors.

The time to extract one MSER descriptor is on average about one-third longer than the time to extract one SURF descriptor (Figure 6.1). On the

[1] 480,000 pixels or less (800x600 pixels)
[2] SURF, SURFPlus and OpenCV SURF implementation

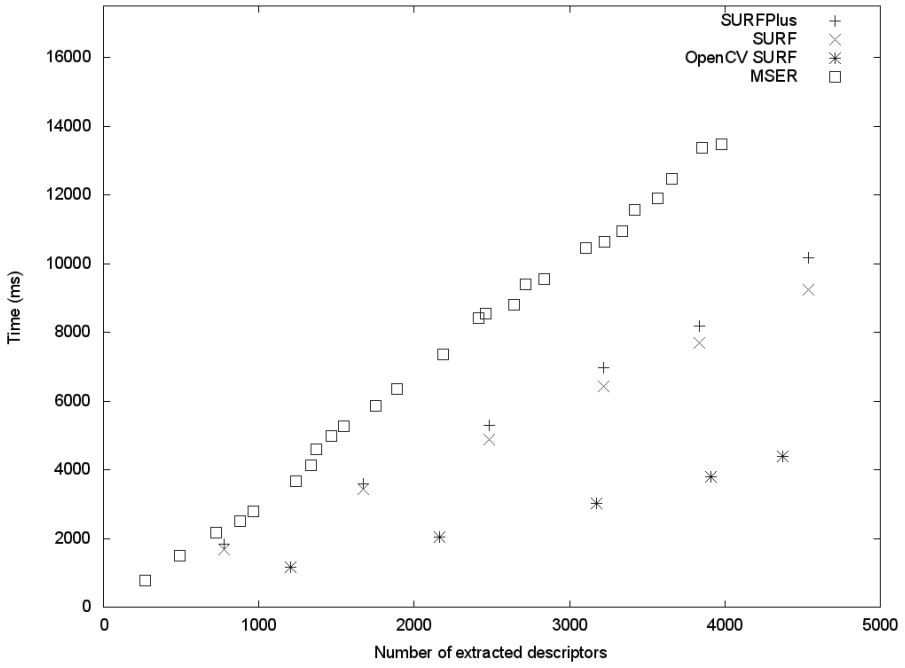



Figure 6.1: Time needed to extract descriptors

other hand, the number of extracted MSER descriptors is much lower and thus the overall time for describing one image is shorter.

We can see in Figure 6.2 that describing an image using our SURF implementation takes a bit less than two seconds.

### Clustering

The time for clustering was measured as well. We wanted to measure it on real data, so the clustering was run on descriptors[3] extracted from real images. We measured the time for all clustering algorithms used.

The duration of the clustering with specified cluster size and clustering with specified depth is almost the same. The CLARA algorithm clearly outperforms

[3]SURFPlus descriptors

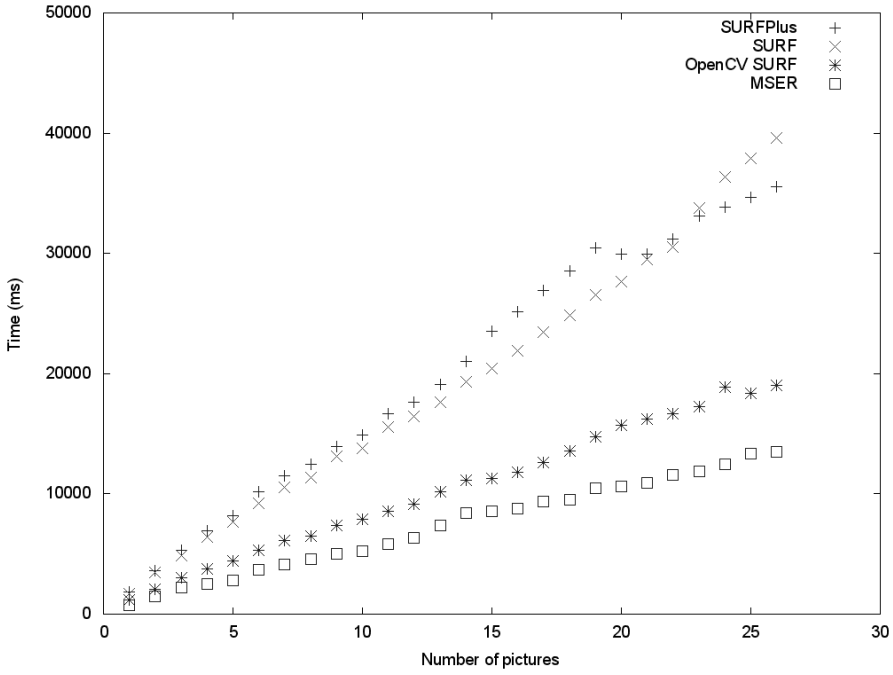



Figure 6.2: Time for descriptor extraction from the images

both of these. Not surprisingly, the clustering with specified cluster size scales much better with respect to the number of clusters.

## 6.2 Success rate

We performed measurements of the success rate on two different image sets. One was the benchmark image set used in [16] (later referred as *NS set*[4]). This set contains of 10,000 images, however while tuning the algorithm we used a subset of only 504 of these - tuning the algorithm using the full-scale test would simply take too long. This image set was however not suited for guessing with the timestamp information. To test the improvement gained by the timestamp information we needed to create our own testing set, one that would be based on some real photo collection. This set was used to test the algorithm both with and without the timestamp information to make the results comparable.

The *NS set* was used to test both the algorithm for guessing one (later

[4]freely available at http://www.vis.uky.edu/~stewe/ukbench/

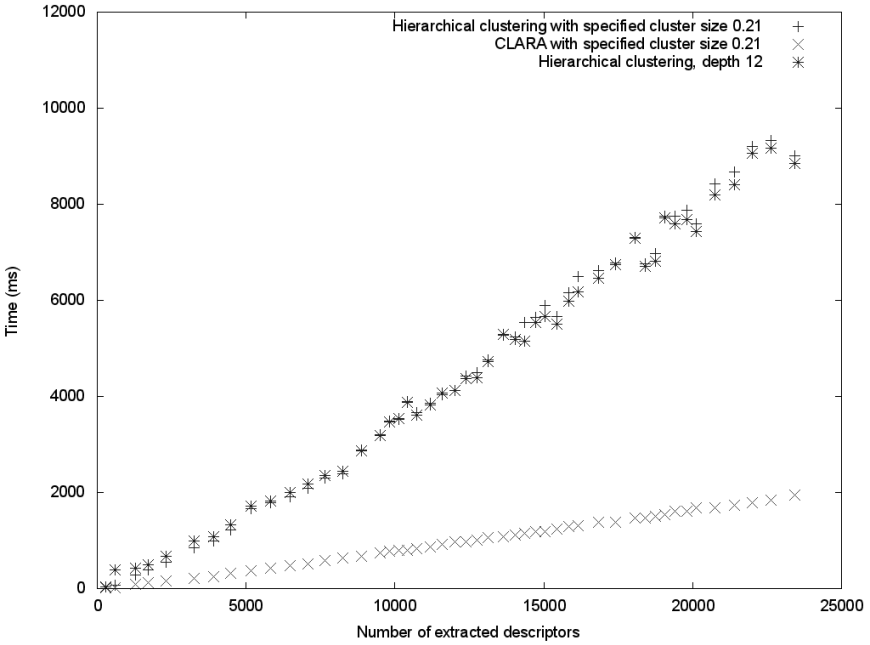



Figure 6.3: comparison of clustering with specified depth and specified cluster size

referred as *single-guess-algorithm*) and multiple tags(later referred as *multiple-guess-algorithm*) per image. Images in the set are numbered and grouped by objects they represent in quadruples. One image of each group was used as a member of the training set. The testing set consisted of all the images thus the algorithm will have to recognize the images from the training set as well (this was the methodics used in [16]).

The guessing performance was measured with different descriptors and several cluster sizes. We measured the precision[5] and recall[6] of the algorithms.

We can see in Fig 6.4 that the *single-guess-algorithm* algorithm has very high precision - almost each guess it takes is correct, however it accepts so few guesses, that the recall is at best only about 50% (training set being part of the testing set).

The *multiple-guess-algorithm* algorithm it is quite the opposite - it guesses about 90% of the tags (training set included), however the precision is poor. In both tests the OpenSURF-based descriptor implementation slightly out-

[5]ratio of correctly guessed tags and number of guesses
[6]ratio of correctly guessed tags and total number of tags to be guessed

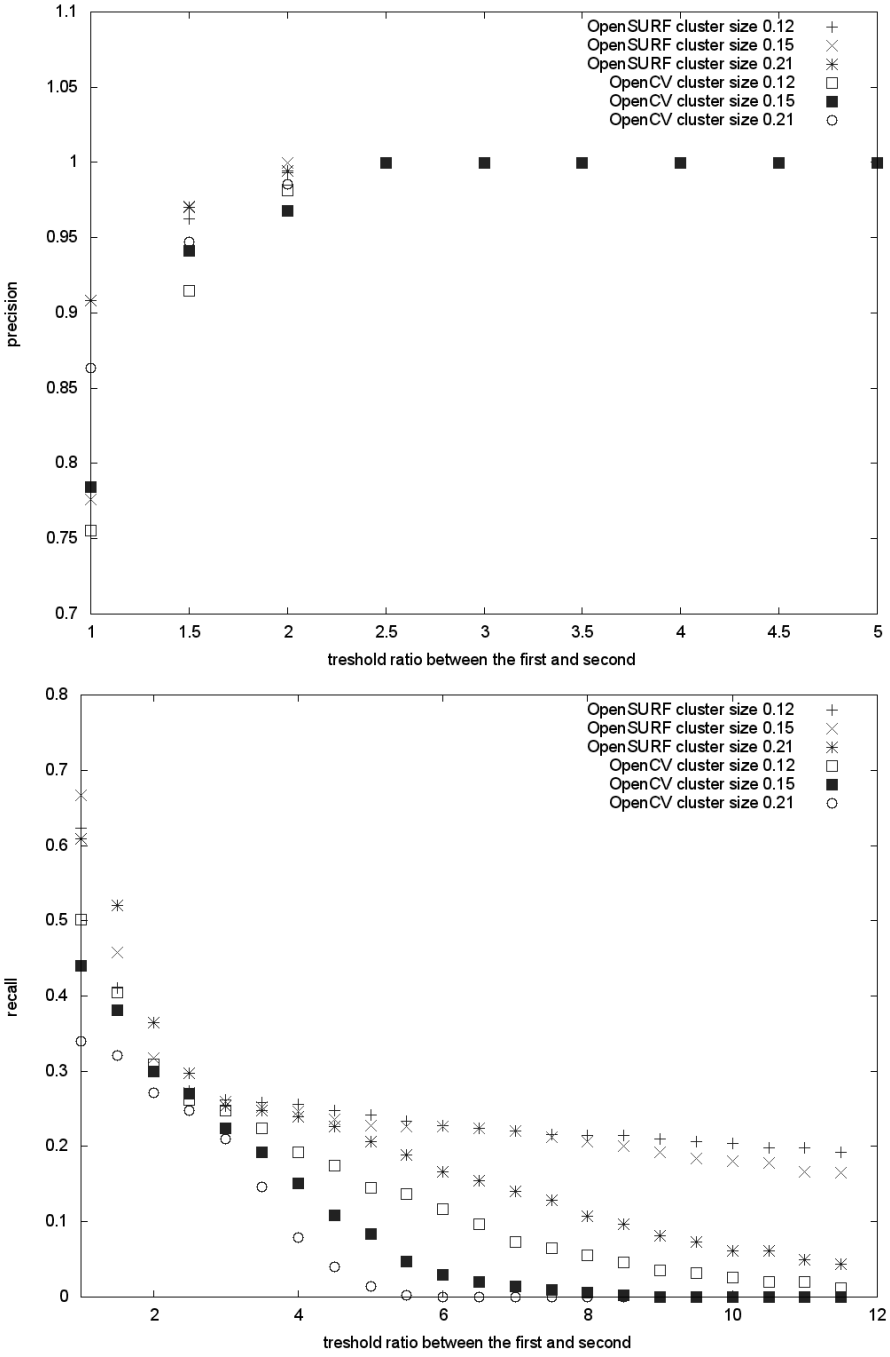



Figure 6.4: Precision and recall of the restrictive algorithm

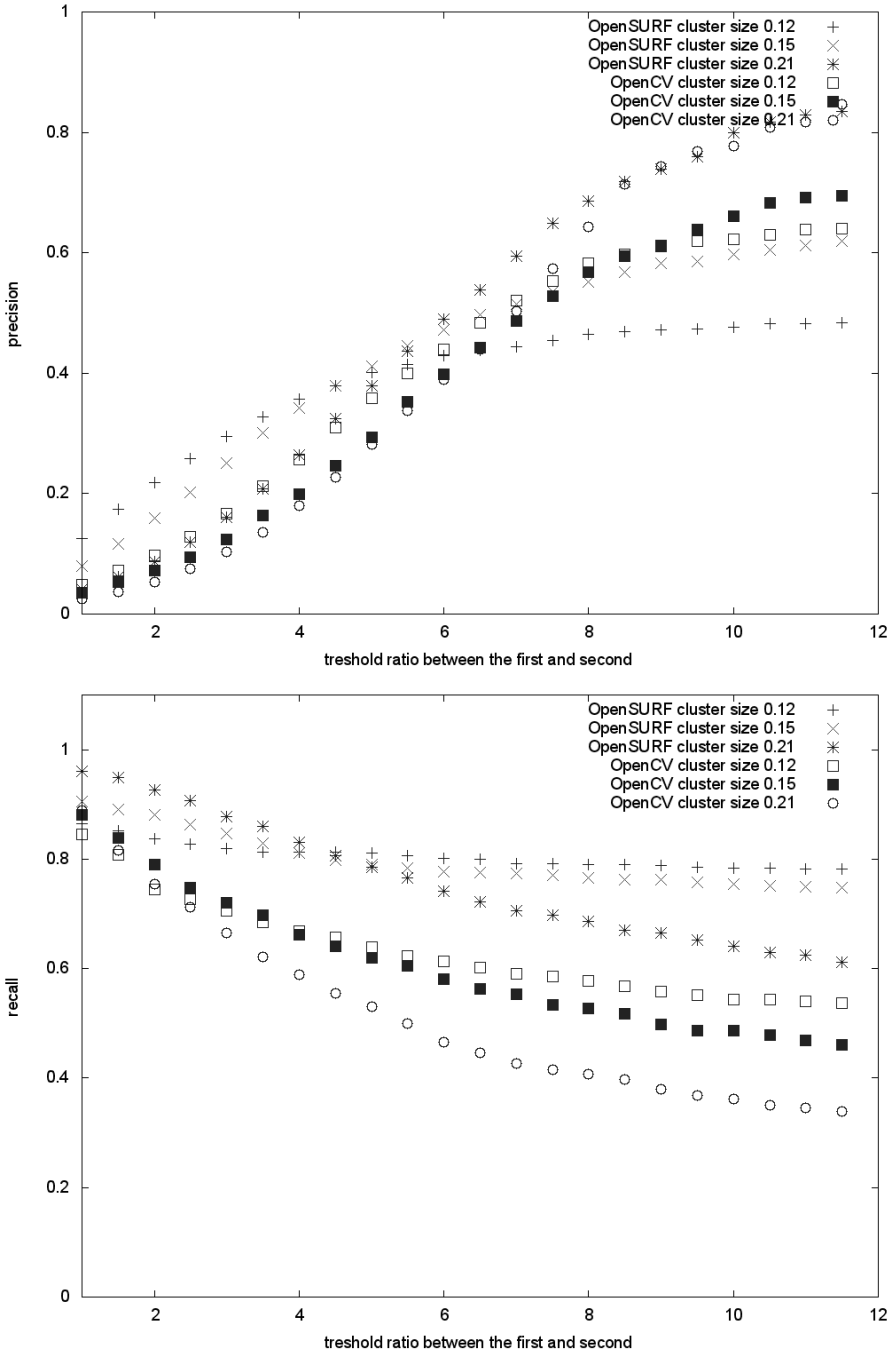



Figure 6.5: Precision and recall of the algorithm for multiple tags per image

performed the OpenCV implementation. Also the cluster size has significant impact on the tag acceptance rate. After tuning the cluster size[7], we ran the same experiments on a larger subset of the NS set consisting of 1252 images, results are in the Tab. 6.1 and Tab. 6.2.

The second image set (later referred as *timestamped* set) is based on a real photo collection. This was needed for the measurements of the algorithm using the timestamp information, because the timestamp information would be hard to simulate. It was divided into training and test set.

The training set contains 43 tagged images with 8 tags that did not occur in the testing set[8], 5 not tagged images and 30 images with tags that will occur in the testing set. The testing set then contains 116 images, 22 of which do not contain any object that should be recognized by the algorithm. It contains 26 different tags.

On the *timestamped* set we first ran test of the algorithm without use of the timestamp.We ran only the *multiple-guess-algorithm* test with wider range of threshold ratio to accept a tag. We used cluster size of 0.15. In Fig 6.6 can be seen that the precision rises almost linearly with acceptance ratio and the implementation with OpenSURF based descriptors performs slightly better that the one using OpenCV SURF descriptors. Similarly to the tests with the *NS set*, the algorithm has reasonable recall but weaker precision.

After that we ran the algorithm using the timestamp.

We can see in the Fig. 6.7 that applying the timestamp information in the probably most basic form significantly improved the precision.

The results for real data are promising, but despite the improvement achieved by the use of the timestamp, the results are not fully satisfactory. The algorithm has reasonable precision, but the recall is at best only about 50%. It should be noted, that this heavily depends on the image set used. The gallery we used consists of many pictures of historical buildings, which are often very similar.

[7] used cluster size was 0.15

[8] used to create timestamp statistics used by the algorithm

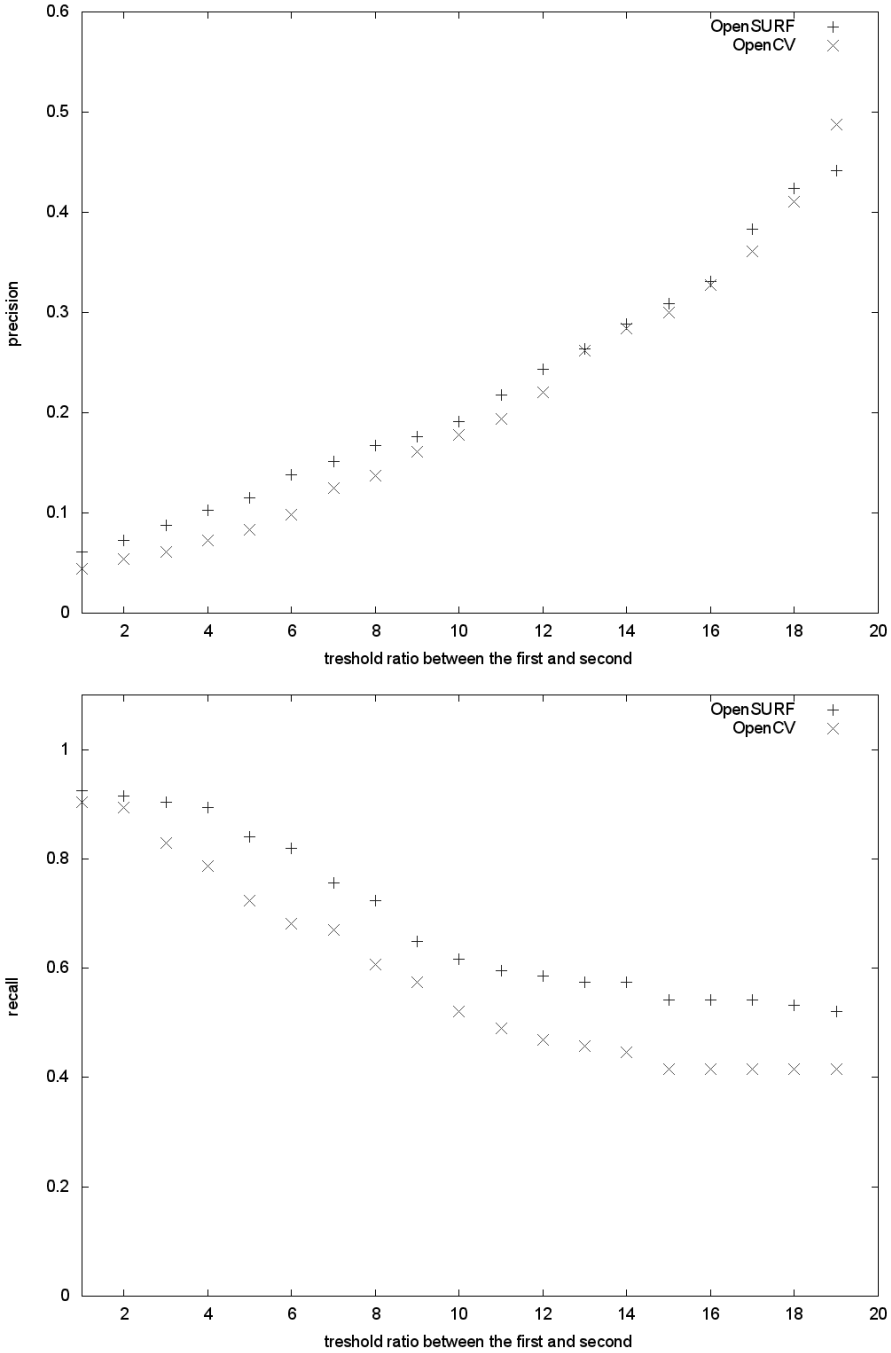



Figure 6.6: Precision and recall of the multiple tag guessing algorithm running on second test set

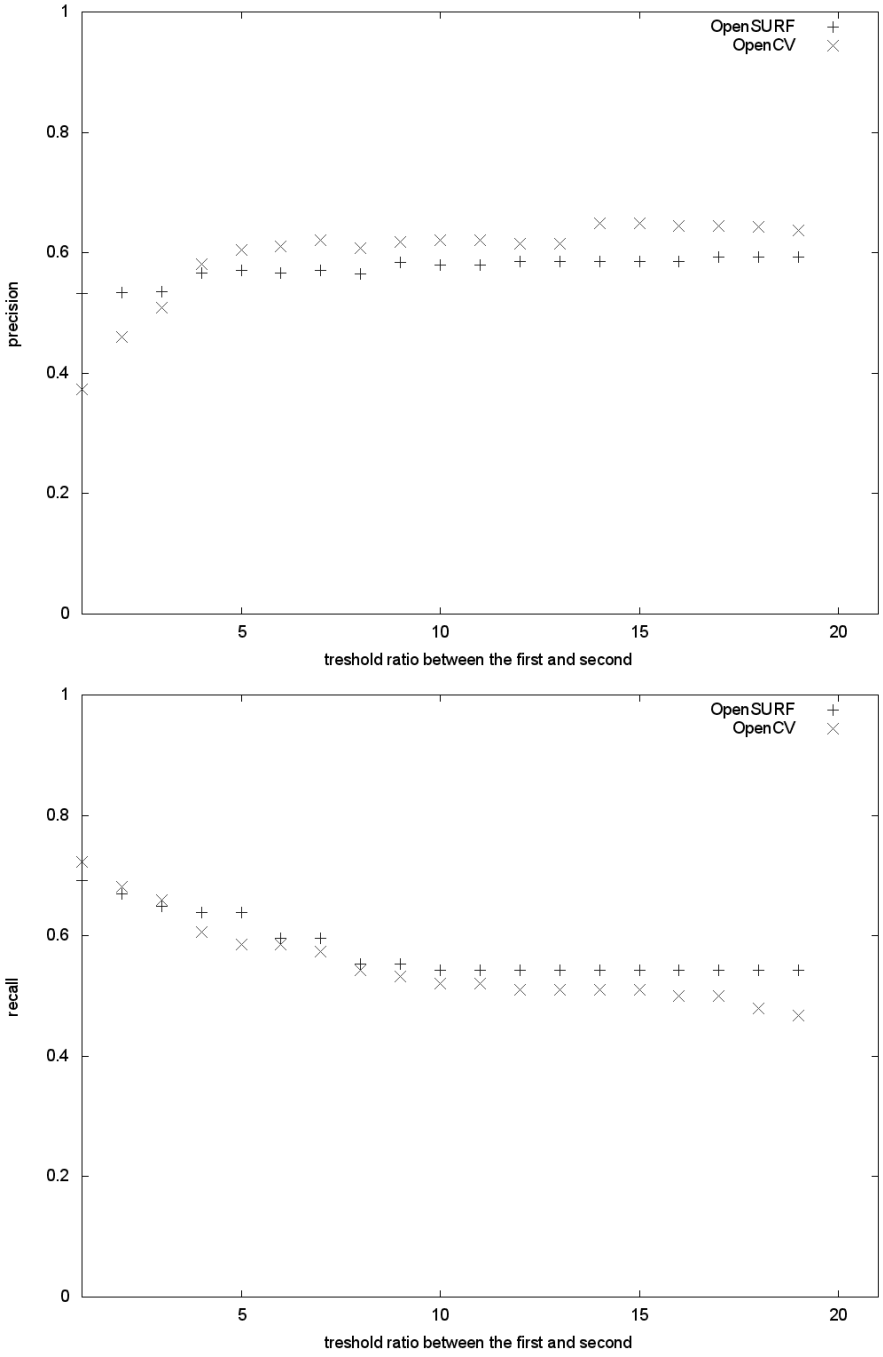



Figure 6.7: Precision and recall with use of the timestamp information

| Threshold ratio | Precision | Recall |
|---|---|---|
| 1.0 | 90% | 36% |
| 1.5 | 95% | 34% |
| 2.0 | 97% | 31% |
| 2.5 | 99% | 28% |
| 3.0 | 100% | 26% |

Table 6.1: Results for 1252 images from *NS set,* using *single-guess-algrithm*, cluster size 0.15

| Threshold ratio | Precision | Recall |
|---|---|---|
| 1.0 | 3% | 92% |
| 2.0 | 7% | 82% |
| 3.0 | 16% | 74% |
| 4.0 | 29% | 66% |

Table 6.2: Results for 1252 images from *NS set*, using *multiple-guess-algorithm*, cluster size 0.15

# 7 Future development and improvements

Paper [7] mentions use of segmentation and texture description. Our attempts to describe areas extracted by MSER algorithm did not bring much success, but with more sophisticated approach the result could be much better. Also we did not try to describe the texture of these areas. As was mentioned earlier, the area-based features have the advantage of ignoring their surroundings which can be different on each image.

Many extracted descriptors are actually not related to the tagged object but to other objects in the image or to its background. That might be a problem, because they will still affect the results. If we could eliminate such descriptors by filtering out those that are not important we could significantly decrease false positive rate. However it is difficult to define descriptor importance. There was attempt to do so by using only the descriptors occurring more than once in images with a certain tags but the results were not promising and the development in this area did not continue.

Also introducing some geometric constraints for descriptor matching might improve the matching precision. However, such methods are rather complex and in our case even more because of the fact that the learning algorithm does not know the correspondence between the features and the tags.

A bigger dataset with timestamps of the images would be needed for further development. Dataset used in this work is relatively small and it is hard to tell how successful the algorithm would be when applied to image sets with different type of objects.

Proposed use of timestamp is very simple but even this significantly improved the results. However more complex and robust system would be useful,

for example information from more recent images could be considered instead of just one.

If we discuss the use of additional information about the image files, we should also mention the geolocation information. Many digital cameras are able to obtain geolocation coordinates using a GPS module and to include them into the digital photography. However this information is only useful for some kind of objects, mainly buildings.

# 8 Conclusion

Aim of this work was to propose an algorithm for automatic image tagging based on the bag-of-words model and known spatial feature descriptors. We partially adapted the approach used in [3] original designed for recognition of whole image or scenery to fit our needs. This approach was designed to work in real time and to be fast. We used the SURF descriptors with additional colour information, CLARA clustering algorithm and the bag-of-words modeling.

To fit our needs a probabilistic model was designed to gather information from images with multiple tags and to allow to tag several objects in an image. Besides spatial information we successfully used the timestamp information to further enhance the algorithm. We also developed several algorithms that in the end were not very successful but brought several useful ideas which were later used in the proposed solution.

After tuning several parameters the algorithm is giving promising results, although it is still not fit for practical use. The results with synthetic benchmark image set are even more optimistic. Experiments have shown that using additional information about the image file such as the timestamp can significantly increase success rate of the algorithm even when used in the most basic models. Using more complex and robust models could even more enhance the automatic tagging system.

# Bibliography


[1] Aurenhammer, F. (1991). Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23.

[2] Barbara Zitová, J. F., J. F. (2003). Image registration methods: a survey. *Image Vision Comput. 21(11)*, 21:977–1000.

[3] Botterill, T.; Mills, S. G. R. (2008). Speeded-up bag-of-words algorithm for robot localisation through scene recognition. In *Image and Vision Computing New Zealand, 2008. IVCNZ 2008. 23rd International Conference*.

[4] Cortes, C. and Vapnik, V. (1995). Support-vector networks. In *Machine Learning*, pages 273–297.

[5] Crow, F. C. (1984). Summed-area tables for texture mapping. *SIGGRAPH Comput. Graph.*, 18:207–212.

[6] Datta, R., Joshi, D., Li, J., and Wang, J. Z. (2008). Image retrieval: Ideas, influences, and trends of the new age. *ACM Comput. Surv.*, 40:5:1–5:60.

[7] Datta, R., Li, J., and Wang, J. Z. (2005). Content-based image retrieval: approaches and trends of the new age. In *Proceedings of the 7th ACM SIGMM international workshop on Multimedia information retrieval*, MIR '05, pages 253–262, New York, NY, USA. ACM.

[8] Daubechies, I. (1992). *Ten lectures on wavelets*. Society for Industrial and Applied Mathematics, 1 edition.

[9] Dirk Walthera, Ueli Rutishausera, C. K. P. P. (2004). Selective visual attention enables learning and recognition of multiple objects in cluttered scenes. *Comput. and Neural Syst. Prog*, 139-74.

[10] Evans, C. (2009). Notes on the opensurf library. Technical Report CSTR-09-001, University of Bristol.

[11] Forssen, P.-E. and Lowe, D. (2007). Shape descriptors for maximally stable extremal regions. In *IEEE International Conference on Computer Vision*, volume CFP07198-CDR, Rio de Janeiro, Brazil. IEEE Computer Society.

[12] Kaufman, L. and Rousseeuw, P. (1990). *Finding Groups in Data An Introduction to Cluster Analysis*. Wiley Interscience, New York.

[13] Li, J. and Wang, J. Z. (2008). Real-time computerized annotation of pictures. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30:985–1002.

[14] Lloyd, S. (1982). Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28 issue: 2:129 – 137.

[15] Nister, D. and Stewenius, H. (2008). Linear time maximally stable extremal regions.

[16] Nister, D.; Stewenius, H. (2006). Scalable recognition with a vocabulary tree. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*.

[17] Szeliski, R. (2010). *Computer Vision, Algorithms and Applications*. Springer-Verlag New York, LLC.

[18] Tirilly, P., Claveau, V., and Gros, P. (2008). Language modeling for bag-of-visual words image categorization. In *Proceedings of the 2008 international conference on Content-based image and video retrieval*, CIVR '08, pages 249–258, New York, NY, USA. ACM.

[19] Wang, J. Z. and Li, J. (2002). Learning-based linguistic indexing of pictures with 2–d mhmms. In *Proceedings of the tenth ACM international conference on Multimedia*, MULTIMEDIA '02, pages 436–445, New York, NY, USA. ACM.

[20] Willsky, A. S. (2002). Multiresolution markov models for signal and image processing. In *Proceedings of the IEEE*, pages 1396–1458.

# List of Figures

# List of Tables

# A Unsuccessful attempts

## A.1 Image description algorithms

We developed and implemented MSER descriptors as was mentioned earlier in 5.3 but the results with the used standard approach were not not very promising (as can be seen in fig. A.1).

## A.2 Probabilistic models

After implementing the MSER descriptors, they were tested and tried for their ability to create matches on different images of the same object or scenery. The results were not very optimistic, either there were only a few matches or there were too many matches in images of different objects thus making the descriptors unusable. It turned out that the probabilistic algorithm used for SURF descriptors cannot be used here and another one had to be developed. Also the clustering (originally only the basic hierarchically organized was used) had to be changed - only the descriptors near enough to each other should be matched and thus considered to be in the same equivalence classes.

The need for a new clustering has led to a clustering with specified cluster size. This was later considered to be useful property of the clustering algorithm and was used in the final solution.

The probabilistic algorithm is based on the assumption that between similar images should be more matches than between the different ones. These counts are expected to be in grade of ones, independently of the count of detected MSER areas.

In each image's extracted descriptors are created clusters, so for two image it can be quickly said how many descriptors do they have in common. It was considered to be more suitable to have separate clusters for each image instead of having global descriptor equivalence classes.

Then there are two important statistics calculated for each of the object. One is the average count of common (or matched) descriptors between two images containing the object. The second is the average number of matched descriptors between an image with the object in it and one without. The midpoint of the interval between these boundaries is used as a threshold for the decision algorithm. It is expected that in a new image with the object the average count of descriptors matched with other images containing the object is higher than the threshold.

Experiments have shown that these numbers are on average different and this algorithm could have good results. However the variance of the data was too high, therefore the threshold based on the average counts could not be really precise measure to accept or deny a tag and the results were in the end not really promising.

Another drawback was that the learning was extremely slow - this algorithm compares each image to all the other images.. Global descriptor equivalence classes could be used to counter this.

## A.3 Use of refined descriptors

There was an attempt to filter less important descriptors and not to use them at all. Several methods to do so were proposed.

One was to use only descriptors whichs equivalence classes are represented by more than one descriptor. Another used only the descriptors that occur in more than one picture. Other were based on using only descriptors that are significantly more often related to one particular object. Also their combination was tried. However none of these attempts was really successful and therefore no descriptor filtering method was used in the end.

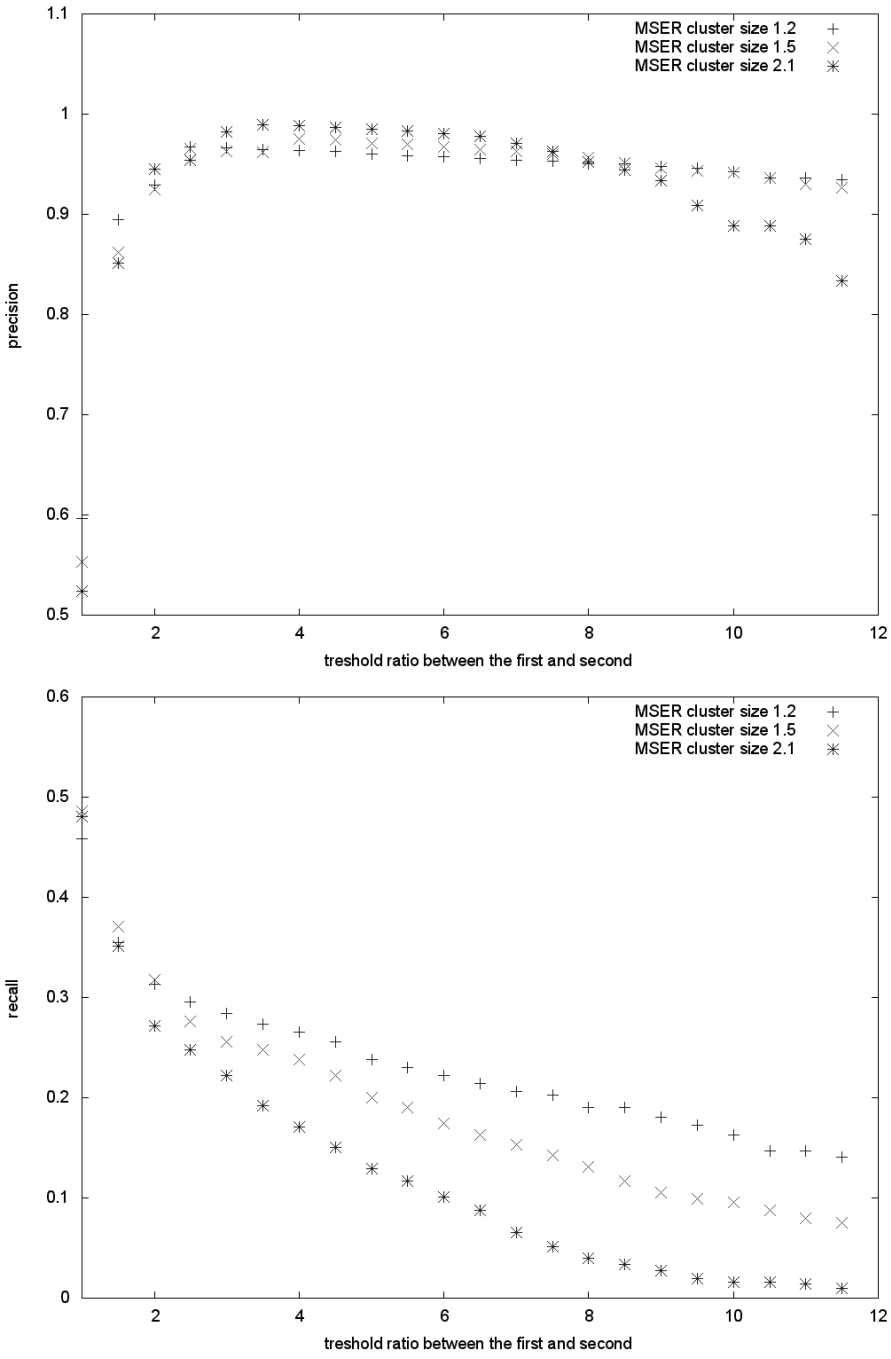



Figure A.1: Tags guessed using MSER descriptors (single-tag guessing)

# B Technology demonstrator

To demonstrate use of designed algorithm a technology demonstrator in the form of a simple image viewer was created. It has the most simple user interface, but it allows to view images, tag them and to guess tags on them or on a whole directory.

## B.1 User interface

User interface is as simple as possible. The user may either browse a directory or view an image.

When browsing a directory, the user sees the image thumbnails in it. If a file in directory is not an image file, it is represented by a blank file icon. Directories are represented by a directory icon. By clicking on a directory the user changes current directory, by clicking on an image he switches to image view.

In the image view the user views only one image. He can return to the browsing view by pressing the appropriate button or by pressing the "Escape" key. In the lower bar he can see the text area with tags assigned to the image. These are delimited by a semicolon. User may freely enter new or delete them and confirm changes by pressing the "Tag" button.

After pressing the "Guess tags" button the program will run the tag guessing algorithm (without timestamp information) and add the tags to the text area. The user may then accept the changes.

In both views the user may invoke learning of the algorithm from new image database information and also let the program guess tags on images in the current directory.

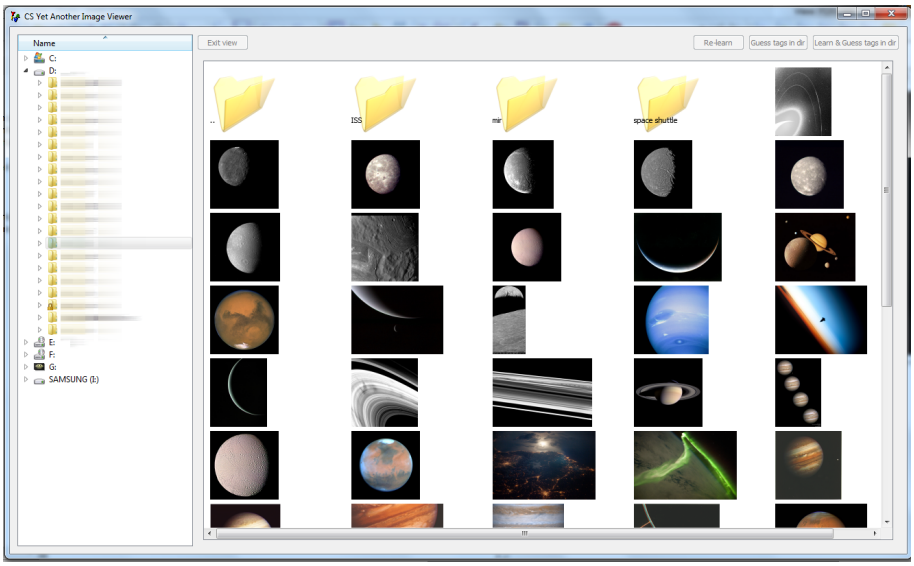



Figure B.1: Application screenshot, on left is the directory tree, on the right the view of selected directory, together with sub-directories

## B.2 Program design

The program uses the *smartGalleryLib* library for all image processing algorithms. Besides that its code is divided into *gui* and *logic* modules. The *gui* module contains classes related user interface and classes for threads that are used to load images. The *logic* module contains class used for communicating with *smartGalleryLib* classes and classes representing threads that are responsible for running algorithms in the *smartGalleryLib*.

All the time-consuming operations such as gallery learning and image loading are executed in separate threads. After the threads finish they emit corresponding event which is then propagated to the user interface thread. This is reason why access to the gallery is wrapped by the class *SafeGalleryWrapper* which handles locking of this class to avoid concurrent accesses of the gallery.