DOAS 2006 Project Article: Mobile Landmark Recognition

Nikolaj Groeneweg   Bastiaan de Groot   Arvid Halma   Bernardo Quiroga
Maarten Tromp

{njgroene, brgoot, ahalma, bquiroga, mtromp}@science.uva.nl

February 2, 2006

Abstract

In this article we describe the development of an application for mobile landmark recognition called 'Poire' (Point Of Interest Recognizer). A user should be able to walk through an unknown environment, make a picture of a landmark/point of interest, after which our application returns on topic information. The approach taken was twofold. On the one hand it was attempted to find robust, simple and cheap features that might be used for an initial selection of promising candidates for further classification. Such features might also be fruitfully employed in a hierarchical learning scheme. Several methods that were initially considered interesting have been described, which include several color based approaches, several shape based techniques and two new algorithms which attempt to correct for similarity and affine transformations. A subset of these methods was selected for implementation on the mobile device. The performance of these techniques was evaluated within MATLAB on three different databases: the Zürich Building Database and two different flavors of a custom database containing images of buildings of the 'Roeterseiland complex' of the University of Amsterdam. The obtained performance measures indicate that the application we propose is indeed feasible although it will require some more work to obtain a commercially viable application.

1 Introduction

In todays society mobile technology plays an important role. With the widespread use of next generation mobile phones a large percentage of the population carries a potent processing unit, which nowadays is accompanied by a high resolution camera. This setting has given rise to a number of interesting applications, but so far AI techniques have played a limited role in this setting. In this article we outline an innovative AI application for mobile phones incorporating techniques from the field of computer vision and pattern recognition applied to embedded systems with limited processing capacity. We will suggest and describe the development of an application for landmark recognition on a mobile phone. The idea behind mobile landmark recognition is that it enables a user to take a picture of a landmark of interest and have it automatically recognized by the application, after which the user is shown some useful information on the landmark in question. Within the literature, the study of landmark recognition is focused on high-level implementations using full fledged computers, with little to no practical work being done on the implementation on low capacity embedded devices, as a recent dissertation from the university of Dublin illustrates [9].

Although many applications of mobile landmark recognition exist, we focus on recognizing buildings. As an illustration of the problems faced by our envisioned application, imagine someone (the user) walking through an unknown city. That user might take a picture, using the camera of his mobile phone, of a building in which he is interested during any time of the day, under different weather conditions, from any type of angle. Therefore the suggested application will require a learning algorithm that uses robust feature extraction methods that work under different angles and lighting conditions. If possible, the learning method should be invariant to similarity transformations and would ideally also be invariant under affine transformations. On the one hand we are faced with these high demands of the application, on the other we are confronted with practical constraints of the implementation platform. Developing a computer vision application that works in acceptable time on an embedded device such as a mobile phone requires a careful selection of techniques, as the mobile platform imposes constraints on both computational power and storage capacity. These constraints greatly influence the choice in algorithms and restrict us to a selection of relatively cheap techniques which require as little storage as possible. This implies that great care should be taken to ensure that the representation of buildings in the local database (on the mobile device) is compact. Ideally, one would train a learning algorithm offline and provide the mobile device with a compact representation of the original database which contains only those image features which are required to correctly classify the buildings in the database. The mobile device would subsequently perform features extraction/construction calculations on the captured image and feed these to a compactly represented decision rule. We note that the computational effort can be greatly reduced by the assumption that we have some information on the approximate location of the user of the application, which can be obtained through observing at what 'cell' within a city the roaming mobile device is currently registered. This assumption is quite reasonable (such information is available in practice when working on an actual network) and cuts down the number of required comparisons for each classification drastically (since only a small subset of the available database needs to be considered).

It seems clear that there exists a certain tradeoff between the computational complexity of our methods and the performance our application can hope to achieve. In the hope of finding a solution that meets both our performance criteria and our practical constraints, we have opted to explore two different approaches simultaneously. The first approach entailed an attempt to find cheap, robust features that themselves might not yield good classification performance, but might be combined into a good classifier using some sort of hierarchical decision making process or voting scheme. The second approach entailed looking for a single robust algorithm that could correctly classify buildings using more complicated operations over the input image while still operating within an acceptable time on the mobile device.

This article is structured in the following manner. First, we will describe some approaches and techniques that were initially considered to be of interest. The advantages and disadvantages of these techniques will be discussed, with an emphasis on the feasibility of implementing them in on a mobile device. Subsequently we will discuss which subset of these techniques were actually selected to be implemented on the mobile phone. The performance of the selected techniques evaluated within MATLAB on a standard industrial dataset (the Zurich Building Database (ZuBuD), see [14]), and two more difficult custom datasets - the results of which will be discussed in some detail. Subsequently the hard- and software used for the implementation of our techniques on the mobile phone will be discussed, as well as the time-performance of the implemented methods on the mobile phone. Finally we propose several possible improvements for our algorithms which might be explored in future research.

2 Methods

Within the computer vision literature some general approaches for the recognition of landmarks have been described. Most of these techniques rely on rather expensive computational techniques however, and are therefore not interesting for our current application, but others are less computationally expensive and might applied on an embedded system successfully. Here we give a brief overview of some approaches we initially felt to be promising, describing both techniques found within the literature and additions of our own. We will divide this discussion into two parts: one dealing with relatively cheap methods of obtaining robust features to be combined into a classifier and one dealing with finding a single robust algorithm.

2.1 Robust features for weak classifiers

When focusing on feature extraction on the input image, two basic approaches are widely employed: those based on color and/or intensity and those based on the shape of objects within the image. In both approaches, one is faced with the challenge of extracting only the information on the objects of interest. There are different ways in which once could attempt to solve this problem. One could attempt to only extract information from objects in which one is interested - i.e. apply some sort of preprocessing filter to the image. Alternatively, one could apply some sort of weighing operation (e.g. multivariate gaussian) on the image in order to assign less importance to regions which are likely to include noise. Some techniques are more suited for the first approach, some for the latter. We will discuss the manner in which we have dealt with this problem as we come across it.

In the category of color based techniques, we considered naive histogram based approaches using rgb, hue and the number of transitions between color-clusters in the HVC colorspace. In the category of shape based techniques, we investigated the possibility of using line clustering techniques to characterize buildings, finding similarity transformation invariant descriptors of the skyline of a building as a means of characterization and a corner detection approach using a likelihood based comparison method. We will treat these techniques in more detail below.

2.1.1 Color based approaches

The most straightforward approach of comparing two images is perhaps that of calculating color histograms of the images and determining the distance between these histograms using some sort of metric. We have considered two such approaches: one using histograms in the rgb colorspace and one using histograms using only the hue values from the HSV colorspace. Both are invariant to viewpoint, object shape and illumination intensity [2]. Hue has the added advantage of being invariant to highlights and requiring only a single dimension for its histogram representation, which might be a significant advantage when implementing the technique on the mobile device.

Having decided on a colorspace in which to calculate color histograms, one needs to pick a metric to compare them. Several options are available. One could for instance use the Euclidean distance between the two histograms, but this is a rather expensive measure. Or one could use measures which have proved useful in practice but are not strictly distance metrics, such as the Bhattacharyya distance[1] or a χ^2 metric [12]. The latter measure has the added benefit of providing in an easy way of determining an uncertainty measure [12]. Of course the χ^2 measure does assume a normal distribution of the histogram, an assumption which is unlikely to hold in practice. The measure might however still prove useful, assuming that it is robust enough to still allow meaningful comparisons between the histograms. There is some indication within the literature that this is indeed the case [12].

As was mentioned before, one should consider if one want to weigh all regions of an image uniformly when constructing a histogram. Generally this is not the case; the borders of an image are likely to contain a large percentage of noise caused by occlusion or undesirable camera effects, assuming the building is centered within the image. Therefore one might weigh the image with a Gaussian kernel with μ on the center of the image, thus assigning less importance to objects nearer to the image border. This will give the desired effect of assigning most weight to the building itself. Such a weighting scheme seems like a useful extension of any histogram based approach.

Histogram based approaches like the ones described above have the substantial disadvantages that they throw away all spatial information, even though such information might be of importance in characterizing a building. An example of a color based method which does not suffer from this disadvantage is one taking into account the color transitions within an image. If one were to perform some sort of color clustering on an image, thusly reducing the number of colors, such techniques become feasible in practice. One example of the use of color clustering can be found in the research of Li and Shapiro, who introduce a feature in their building recognition system which is based on clustering in the HVC color space [5]. The basic idea is to obtain a small number of clusters, in the order of 20, and to characterize each pixel by the cluster to which it belongs. This way one not only takes color information into account but also includes some spatial information about which color clusters occur adjacently. This information can then be used to characterize a building. The advantage of using the HVC color space is that it provides a straightforward distance metric between different colors. The HVC colorspace is a mathematical simplified version of the Munsell colorspace, in which the distance between two colors is linearly depended on the difference people experience between the two colors [8](for the interested reader, more details on the conversion of RGB to HVC can be found in appendix A). Because of this property the distance between two different colors obtains a clear meaning, which is required if one wants to perform a clusterings algorithm.

2.1.2 Shape based approaches

The additional use of shape based features to characterize a building has some advantages over using only color features. It it is not unreasonable to assume that the shape of a building characterizes a building in ways that color based features cannot. Unfortunately shape based features are generally more sensitive to similarity and affine transformations, which means special care has to be taken in selecting appropriate features.

One potentially interesting option for employing shape would be to cluster lines occuring in the image based on certain line features. Once could for example cluster lines that share a certain orientation (i.e.: horizontal, vertical, sharing some angle $\alpha - \epsilon$, or oriented towards one of the vanishing points) and use statistical measures over such clusters to compare buildings. A particular building might have a certain configuration of clusters which can help identify that building when combined with other features. When applying this feature extraction technique, one should consider how to extract only features from the object of interest. For lines of certain orientations this is not such a big issue, as they are likely to belong to buildings. Lines that are oriented towards one of the vanishing points for example, are very likely to belong to man made structures and as such will often belong to buildings[5]. For lines of other orientation (a specific angle for example), one might consider applying some sort of preprocessing before applying lineclustering techniques in order to extract the building of interest. One of the biggest problems with such a feature is that it is not invariant to rotation or affine transformation. Therefore, one might expect it to fail rather easily.

Another shape based feature that might prove interesting is one which attempts to give a transformation invariant description of the skyline. In many cases the skyline gives a good description of a building as it might include characteristic rooftops and other distinguishing features. One manner to go about finding such a description is to view the skyline as a function defined over the width of an image. This function could be represented by a fourier series, thusly describing the skyline in terms of a set of weighted sine and cosine functions. Such a descriptor would be scale invariant, in the sense that the a scaling of the image would merely result in a scaling of the function describing the skyline. It is also somewhat robust against affine transformations; these cause the resulting function to be somewhat squashed and oriented along a rotated x-axis. This means that in principle it should be possible to correct for such transformations, although it remains to be seen how this can be done in practice.

Finally, one might consider running a corner detector to for instance find corners of windows, which occur very frequently in buildings. Although this might seem like a very crude feature, one could easily refine it by dividing the image into different subimages and detecting corners in certain regions. Subsequently one could compare an image based on the number of corners within each region and find a nearest neighbor in the database based on this comparison. There are corner detection algorithms, such as that of Zheng & Wang[13], which are robust under affine transformations, can deal with reasonable levels of scaling and are also relatively fast compared to the often used Harris detector developed by Harris and Plessy [4]. Such corner detection algorithms are especially well suited for our application and seem like promising cancidates for extracting shape based features.

Above we described some features that we consider interesting for using in a learning algorithm that combines different features that themselves are not particulary strong classifiers. Some of the proposed features are more robust than others, but all might serve as a useful initial screening of potential matches when trying to classify landmarks. In the next section we describe single algorithms which are more robust and can hopefully classify a large percentage of the buildings correctly. These could be used in conjunction with the techniques described above, to classify those instances that an initial screening has shown to be most promising.

2.2 Robust algorithms

We propose two algorithms, based to some extend on ideas found within the literature, that we hope are robust enough to distinguish between buildings even under similarity and affine transformations. The first of these algorithms employs a Radon transform combined and attempt to correct for transformations by normalization within the transformed space. The second algorithm is based on the concept of local invariant regions. It attempts to estimate the extrinsic features of a building based on sublimes of images and find a 'normalized' representation of the image in which all transformations are undone. Both algorithms will be described in more detail below.

2.2.1 Normalization after Radon transformation

The main difficulty with the features described earlier is that strictly speaking, one is only interested in the intrinsic properties of an object such as aspect ratio, shape of the windows, whether there are stripes at the left side of the door, etcetera. Other shape properties like translation, scale, rotation, perspective, are extrinsic properties that are involved in the relation between the observer and the target. Ideally, one would like to get rid of these features, since they do not describe the inherent features of the object in which one is interested. The algorithm described in this section is based around the idea that one wishes to correct for the extrinsic variation found in images and compare buildings only on their intrinsic shape properties. The approach we suggest is that one could do this by first applying a Radon transformation on the images in question to obtain a description of the most important lines within the image. A Radon transformation transforms any two dimensional image with lines into a space of possible domain parameters. Within this new space, each line occurring within the image will cause a peak value located at the corresponding line parameters. Parameters that occur more frequently within an image will cause strong peaks within the Radon space, whereas lines that occur only once or twice will be negligible after applying the Radon transform. The Radon transformation is performed by applying parallel projections of the image over varying angles, as has been illustrated in figure 1.

After acquiring the Radon transformation of an image, one might attempt to correct for similarity transformations within this space. In order to do this, one should first notice that the left and right side of a Radon transformed image represent projections of the object from opposite sides: under an angle of $\theta = 0$ and $\theta = 180$ respectively[1]. Therefore the sides of the Radon space can be attached to each other to form a Möbiusring. It is within this Möbius ring of the Radon space

[1]Please note that the remainder of the projections (in the range $\theta = 180$ through $\theta = 360$ are superfluous, as they would yield the same results as the projections performed earlier.

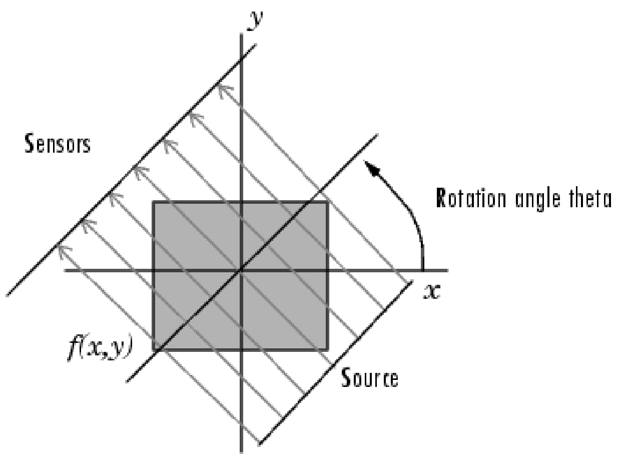


Figure 1: The parallel projections over a varying angle θ which are at the basis of the Möbius Radon transformation

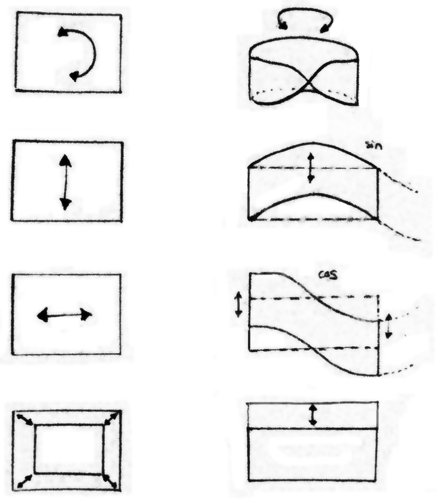


Figure 2: Corrections in the Radon parameter space (connected to form a Möbius ring)(right) for extrinsic variation within the original image (left).

that we believe we could correct for extrinsic properties. In particular, we might begin by shifting the important lines (the ones with strong peaks within the parameter space) to $\theta = 0$. If we do this for all buildings, we are effectively correcting for rotations of the original images. For other corrections, one needs to apply more complicated transformations. In order to correct for translation in the y direction, one will need to transform the Möbius-ring along a sinoid and translate along this transformed ring. For corrections for translation in the x direction, one can do the same but with applying a cosine transformation instead. In order to correct for scaling effects, one should scale the Möbius-ring along its y axis. These corrections are summarized in figure 2.

After applying such normalizations to correct for the transformations in the original image, one might employ any learning algorithm, such as a nearest neighbor approach, to compare and classify images within the parameter space.

2.2.2 Local Invariant Regions

As was touched upon earlier, the main difficulty in classification of buildings is that one is only interested in the intrinsic properties of an object. In this section we suggest a method that should be able to use shape and texture features in a robust manner by applying corrections for extrinsic variation within an image.

The first step in our approach is that an image can be represented by a list of (smaller) image slices, regions, of the original image. Subsequently we can try to estimate the extrinsic properties for these regions. Once we know a region is transformed a certain way we can normalize this local image by undoing this transform. Through this correction, we can represent an image by storing a list of normalized images. A new image from the mobile device's camera can be stored this way, after which a distance measure to known objects can be used to determine the object's identity.

If we look at the entire image of a building it is very hard to determine the discussed transformations and find a canonical representation since different sides of a building are transformed differently. Local regions, where the object is planar, can easily be described in a canonical form, since all points in such a region are subject to the same linear transformation. In the object recognition literature this property of is exploited in so called local region based approaches[11][10][6]. Advantages that are mentioned within the literature are the robustness of these features to occlusion and clutter (due to the fact that they are local), their distinctiveness, as well as the fact that they are computationally inexpensive and extensible.

3 Selected techniques and implementation

In the sections above we have described some techniques that we initially felt to be promising. We have selected a subset of these techniques to be implemented on the mobile platform. We will first give a brief overview of methods that were selected, during which we will elaborate further on some details concerning their implementation. Finally we will discuss why the other techniques were not explored any further.

3.1 Histogram based approaches

We have selected the rgb and HSV based histogram comparison paired with a χ^2 distance measure as a useful initial screening of potential matches. We have chosen the χ^2 measure over other measures such as the Bhattacharyya distance because we like the fact that the χ^2 measure also provides us with a relatively straightforward measure of ambihuity[12]. Given two images, a test image I_t and a model image I_m, the χ^2 distance is defined as:

$$\chi^2(I_t, I_m) = \sum_k \frac{(I_t(k) - I_m(k))^2}{I_t(k) + I_m(k)}$$

Given this distance one can also define a measure of ambiguity which ranges from 0 (in case the test image is very easy to classify) to 1 (in which case the test image is very hard to classify). This ambiguity measure can be defined as:

$$Am = \frac{\chi_1^2}{\frac{1}{n-1}(\sum_i \chi_i^2)}$$

where $i = 2, 3, ..., n$ and χ_i^2 is the ith closest distance of the result. This ambiguity measure might be fruitfully employed in different settings. For example: when combining the histogram based approaches with other classifiers within a decision tree, one might use the ambiguity measure as a cutoff measure for deciding in which cases to prefer other classifiers. In hierarchical approaches one might use it to select the size of the image set to be handed down to more powerful classifiers.

3.2 Color clustering in HVC colorspace

The described HVC color clustering algorithm used in combination with a nearest neighbor technique over the cluster transitions was also selected to be implemented on the mobile phone. The reasons for doing so are twofold. First and foremost, the technique uses a certain amount of spatial information (the ordering of the clusters within the image) which is a clear advantage over the histogram based approaches. Secondly, it can be implemented extremely efficiently on the mobile phone through the use of a look-up table which maps every color from RGB to a certain HVC color cluster.

There are different ways in which one might go about determining the desired color clusters. In their article describing color clustering within the HVC colorspace[5], Li and Shapiro cluster the colors within every picture to detect the different buildings within one image. We however cluster on the colors in all the images of a single building in our complete trainingsset since we need to have the same clusters in the pictures which are taken on the mobile phone and the ones in our trainingsset to be able to compare the two.

On the mobile phone we do not only need to determine the colorclusters, which could be implemented very efficiently using a look-up table as was mentioned earlier, but we also need to determine the *transitions* between the clusters within the query image. As such, we need to have a computationally efficient way to calculate these transitions. In the original article by Li and Shapiro, the authors detect the lines of the building and determine the color transitions across these lines. Since we expect this step to be too computationally expensive, we have opted to simply compare *every* pixel

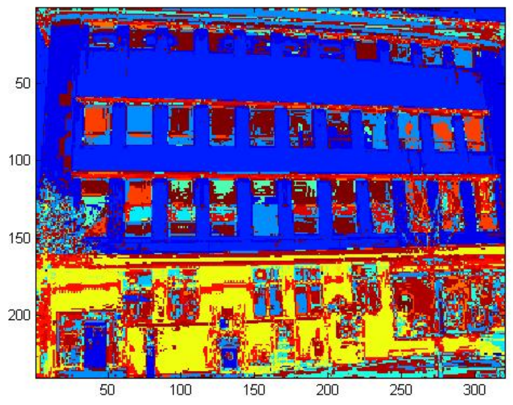


Figure 3: Results of color clustering in HVC space

with the pixel to the east, south east and south of it. In this manner all cluster transitions can be found in only three operations for each pixel, yielding a relatively cheap transition calculation. Subsequently we build a 2 dimensional histogram for every building in the trainingsset in which we count how many transitions occur between every pair of color clusters. Since we want a transitions between C1 and C2 to be counted as the same as a transition between C2 to C1, we always insert a transition as if it is from min(C1,C2) to max(C1,C2) in which the minimum and maximum is defined by the cluster number. This yields an appropriate cluster transition histogram, which can subsequently be used to compare query images and images from the database. For this comparison, we have chosen to use a χ^2 distance measure, for reasons discussed earlier.

3.3 Local invariant regions

The local invariant region approach described earlier was also implemented and tested on the mobile phone. We choose to implement this method because its ability to correct for transformations seems promising and because it could be implemented relatively efficiently on the mobile device. Some details concerning the implementation of this general approach will be discussed below.

As was mentioned earlier, the local invariant regions approach is based on canonical representations of local, planar regions. Based on this basic approach, we will present a new efficient method optimized for mobile building recognition.

Our algorithm does not differ from existing regional methods in the way that it finds characteristic regions, represents them in a invariant way and uses a distance measure in a voting scheme to determine the object's identity. The way we represent regions however, is different from what we have seen before.

Using local regions heavily relies on choosing the same regions independently from each other regardless of image changes. The regions are chosen around key points in the image. Local extrema in the intensity are such stable key points. So under different lighting conditions, viewing angles and scales, dark and light dots in the image remain more or less the same. Intensity extrema which also occur in blurred versions of the image are repeated more often in different views of the object[10]. This makes the selection of key points more robust and reduces the amount of regions found.

From the detected key points regions which automatically adapt the the view point are constructed by probing the intensity changes in different directions. The region boundaries are determined by drastic intensity changes walking along the directions [11].

We have considered other key points found in the literature. Extrema in the difference-of-Gaussian scale space, for example, are much more computationally expensive. Corner points are also more expensive and have a more structural problem: when these points form the centers of a region, we obtain a region which includes different planes in the real world. All of these planes will have their own perspective distortion, making it very hard to estimate the extrinsic variation of the region.

The only transformations we consider for local regions are scaling and vertical skew. By this we assume that the picture is hardly rotated and the skewed rectangle approximates the perspective transformations well enough. This approach has the added advantage

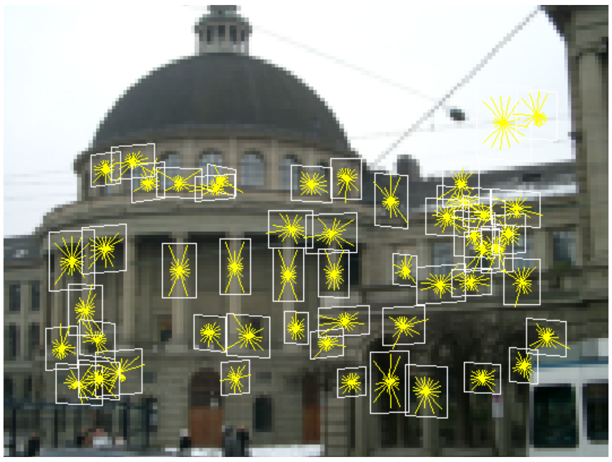


Figure 4: Example of local invariant regions found using parallelogram shaped regions

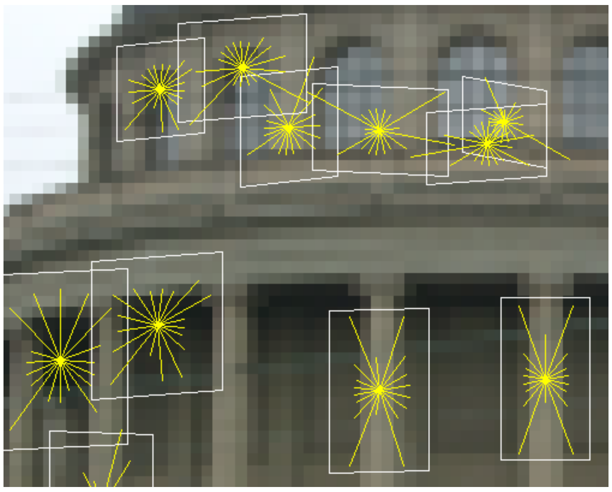


Figure 5: Detail of found regions, clearly showing the intensity extrema used for determining the regions.

of being extremely fast, unlike the approach of using ellipsoidal regions found within the literature[11]. Our approach may not be as generally applicable as the approach using ellipses, but it is very well suited for representing features in man-made structures.

The boundary points of the found regions are used to fit an parallelogram, of which the left and right side are held vertically. To increase the distinctiveness we scale the parallelogram to double it's size.

Unlike others who use ‘Generalized Color Moments’ introduced by Mindru [7] we represent the regions in a more direct way by transforming each region to a fixed size square. This representation is, unlike the ‘Generalized Color Moments’, not invariant to rotations and therefor more descriptive.

We could store all normalized regions directly in a list. Instead of doing this we first do Principal Component Analysis to reduce the amount of information needed to represent each region. By storing the first 30 components we reduce the space required significantly.

To reduce the number of regions stored and thus the amount of computation needed, we discard regions which are not characteristic enough for a certain building and replace similar regions by one prototype. This is achieved by clustering the regions found for one building. Singleton are removed; If a region is characteristic it should occur in more views of the building. This novel addition makes our approach especially tractable for implementation on the mobile phone without loss of performance.

When a new image is captured it will undergo the same process of creating a list of normalized regions from the raw pixel data. Each region found in the query image votes for building belonging to the nearest known neighbor in the principal component space. The weight of the vote equals $1/d$, where d is the Euclidean distance to the nearest neighbor. The query building will be classified as the building with the highest total vote.

3.4 Considerations for rejecting other approaches

The other approaches that were discussed have not been implemented. The reasons for not implementing these techniques vary; they will be elaborated on below.

3.4.1 Shape based approaches

Earlier, we have suggested several potentially interesting shape based features. However, we have decided not to implement any of these features.

The fourier skyline descriptors were deemed to impractical on second thought. Although theoretically one could correct for transformations of the original image, we believe that in practice this would become too computationally expensive and cumbersome. As such we have rejected this technique for the time being, although perhaps it might still be interesting to look into at a later time.

The most promising of the shape based features, line clustering based on orientation, was not implemented due to a lack of proper reference material. We found an article describing the techniques and algorithms we required [], but were unable to acquire the article in time. Since we believed the article in question might arrive at any time, we focused our initial work on the other methods instead of constructing our own algorithm. When the article arrived, however, we found we had too little time left to implement the described technique. Unfortunately we decided in an early stage that the line clustering approach was more promising than the described corner detection approach, so we assigned the latter lower priority - yielding a net result of having implemented neither. This is unfortunate, but such is the fate of science within a two week implementation window.

3.4.2 Normalization after Radon transformation

Normalization after applying a Radon transformation, although very promising, has not been implemented. Initial experimentation with this technique indicated that it would be hard to give the procedure the robustness we require. We tried applying the Radon transformation to edge images, but observed that in general this yields too little information for our normalization purposes (important lines from the building would not occur in the edge images consistently enough). Applying the transformation to the normal images was considered too expensive to be a realistic alternative for our application. Therefore it was decided to invest in implementing the Local Invariant Region approach instead, since this algorithm is inherently robust under affine transformations and much more straightforward to implement.

4 Results

The selected set of techniques have initially been implemented in MATLAB, where they were tested on the 'Zurich Building Database' (ZuBuD)[14]. Subsequently the methods were ported to the mobile phone and tested on a real life database of the 'Roeterseiland complex', a set of buildings belonging to the University of Amsterdam.

Before discussing the results obtained in MATLAB, we will describe the databases that were used for our evaluation purposes. Subsequently we will elaborate briefly on the hardware specifications of the mobile phone used to test our application and the programming language used for the mobile development. Finally we will discuss our current implementation progress on the mobile device.

4.1 Evaluation databases

4.1.1 The Zürich Building Database

The ZuBuD consists of pictures of 201 different buildings taken in the city of Zürich, Germany. For every building there are five different views within the database, each of which are $640x480$ pixels in size. Differences between the views include the angle at which the picture is taken, relatively small scaling effects and occlusion. The pictures look as though they are taken by the same, or at least a very similar, camera. An occasional picture is of considerable lesser quality than the others, but the frequency at which this occurs is negligible. Examples of images from this dataset can be found in figure 6.

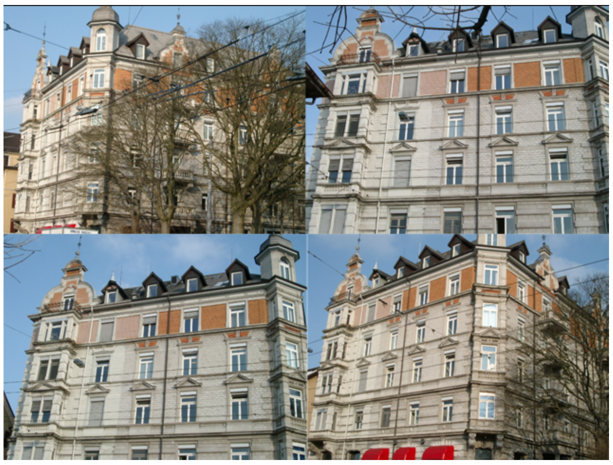


Figure 6: Examples of different views of a single building from the Zürich Building Database.

The ZuBuD comes with a standardized query set, consisting of 115 images of buildings (also with a size of 640x480), all of which occur within the database.

4.1.2 Roeterseiland databases

The Roeterseiland database consist of a set of 6 different buildings. The images used have a size of 1280x960 and were resized from original images shot with a 3.2 megapixel camera. We used two different flavours of this database.

The first Roeterseiland database contains three to five different views of each building (depending on the amount of possible views in real life). Some of the images in this set were rotated (some 90 degrees, some only slightly). Furthermore, we included several images in which occlusion of the buildings by fences or trees occur. Examples of images from this dataset can be found in figure 7.

The second version of this database consists of a set of six different objects, with a slightly higher average number of views per building (four to nine depending on the amount of possible views in real life). The set contains no rotated images, and includes images taken under different lighting conditions. We have explicitly taken care not to include images in which occlusion occurs. Examples of images from this dataset can be found in figure 8.

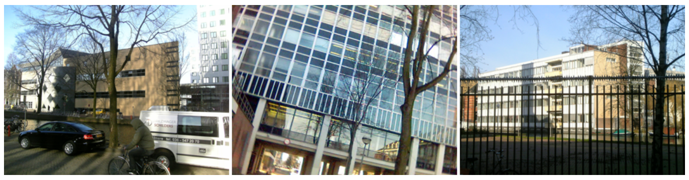


Figure 7: Examples of images from the first 'Roeterseiland' database.

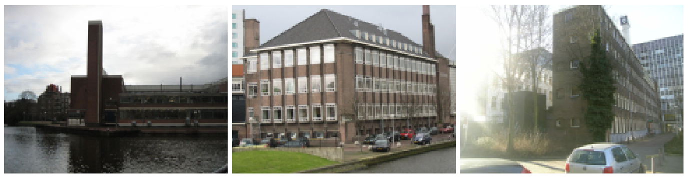


Figure 8: Examples of images from the second 'Roeterseiland' database.

The main reason for evaluating the performance of our methods on two wildly differing databases is to check if our obtained performance is not due to coincidental overfitting to our dataset. The datasets we have constructed are not ideal and could easily be improved to boost the performance (multiple lighting conditions could be included in the dataset seperarely, the dataset could be constructed using a professional camera, etcetera). The results obtained using the Roeterseilandcomplex databases will therefore undoubtedly yield suboptimal results. If we were to build a database for a commercial application, we would make sure that our database is constructed more optimally.

4.2 Evaluation in MATLAB

We compared the implemented classifiers on the datasets described earlier. The obtained results can be found in figure 8.

It's interesting to see that there is no classifier that outperforms all others on all datasets. We obtained a quite high performance for all classifiers, except color

Algorithm	Zubud	RE1	RE2
rgb histograms	79%	58%	7%
hue histograms	94%	34%	23%
Cluster transitions	49%	74%	44%
Local invariant regions	86%	42%	78%

Figure 9: Evaluation results on the different databases. Please note that a single result is still pending at the time of writing of this article.

clustering, on the ZuBuD database. This high performance is against our expectations since we're looking for a simple classifier which works well for a smaller set of buildings. If we decrease the number of buildings in the dataset, to a number which we expect for our target application, the simple classifiers already have a 100% score. The problem is that the conditions under which the ZuBuD queryset and database images were created are quite similar, something we don't expect to be the case in our application. The low score of the color clusterings algorithm can be explained by the fact that the number of buildings in the database is huge and therefore the discriminative power of the transitions drops. For the ZuBuD database we can therefore conclude that there is no need for our more fancy approaches since hue already obtained a good score.

In the Roeterseiland databases, buildings are much more difficult to classify because of the two version of this database include 'noisy' images. As was mentioned before, the first flavour of this database contains some instances of serious occlusion. The second flavour of this database contains images made under varying lighting conditions.

The significant drop in performance for the local invariant region approach on the first Roeterseiland database can be explained in a straightforward manner. Since the images in this database contain quite a few instances of occlusion, this method will have a hard time finding consistent regions across different images of the same building. The histogram based methods suffer less from these circumstances, as can be learned from studying the performance statistics.

The performance drop for the histogram methods on the second Roeterseiland database was also to be expected, since this database includes images with large variation in the lighting conditions, so much so that in some cases we are really pushing the limits of what our method can be expected to handle. The local invariant regions approach suffers less from this type of noise; the local regions that are found do not really suffer from the variation that occurs with the database.

As was mentioned before, the database could and *should* be improved upon for any real life practical application. The obtained performance on the ZuBuD database shows that given a sufficiently stable database, our methods can obtain much better accuracy than they show on the Roeterseiland databases.

It should also be mentioned that before testing the methods seperately, we tried to find a way to combine the three histogram based methods to boost their performance but we were unsuccessful. The biggest problem was that we couldn't find a good certainty measure which predicts wether one of the classifiers is correct or if the right answer will certainly be in it's top N answers. We have tried to use the following measures:

- the size of the bhattacharaya distance;
- the ratio between the smallest distance and the average of the N smallest distances, for both the bhattacharaya and χ^2;

There could be two possible solutions to this problem, either find some similarity measure for histograms which is a metric and computational inexpensive, or find some heuristic like the certainty measure we used for the χ^2 but which actually makes a good prediction.

5 Mobile Phone Development

In this section we will elaborate on some issues concerning mobile phone development. We will discusss the choices one has to make when one decides to develop an application on a mobile phone. We will also describe the development cycle and offer some insight into the performance of a mobile phone in general.

5.1 Platforms

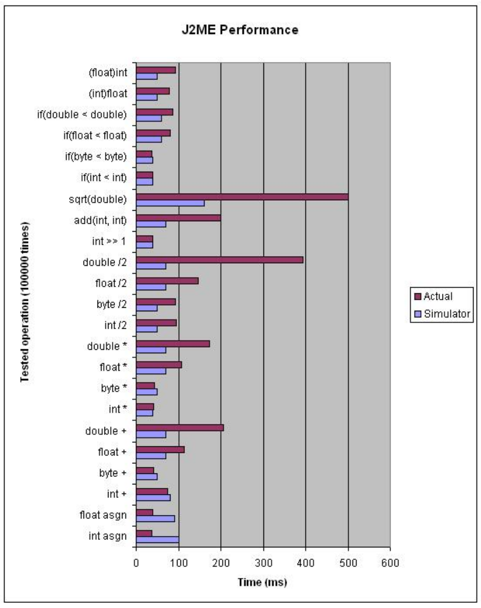
There are two major platforms currently available for mobile phone development: Symbian OS[15] and Java 2 Mobile Edition (J2ME)[16]. From a developer's point of view the major difference between these platforms is that Symbian uses C/C++ whereas J2ME uses a slimmed down version of the Java programming language. Intuitively, this would mean that J2ME is the slower platform as Java compiles to bytecode whereas C/C++ compiles to machine code which can be directly executed by the hardware. Fortunately, there are mobile phones that carry embedded processors (such as the ARM Jazelle[17]) with the capability to execute bytecode directly in hardware, thereby levelling the field as far as performance is concerned. Apart from this technical difference the two platforms also differ in the amount of mobile phones that support the specific platform. At the moment, the J2ME platform is much more widely supported than Symbian OS. We have therefore decided to develop for the J2ME platform as we would like to develop for as big an audience as possible.

5.2 J2ME Development Cycle

J2ME development revolves around the Java Wireless Toolkit[18], which contains the Java compiler, phone emulator, packaging software and software libraries. As the standard J2ME platform offers extremely little functionality one has to select which additional libraries to use. These libraries offer for example the capability to capture video, connect to other devices via bluetooth, support networking (http), access memory cards, play audio files, etc. As not all phones support all these features the use of these libraries further restrict the portability of the code. The toolkit also contains an emulator that greatly shortens the development cycle as one does not need to upload the software to the mobile device in order to test it. For each mobile phone there exists a different mobile phone that not only emulates the functionality of the phone but also, to some extent as we will shortly discuss, the performance of the phone. Instead of offering just the emulator, each mobile phone manufacturer offers their own blend of the Wireless Toolkit as it appears the implementation of the additional libraries is different for each phone. It would appear "Write once, run everywhere" does not hold for mobile phone development. After the software has been written it has to be tested and compiled for each type of mobile phone that one wishes to support as the implementation of the libraries differ for each platform.

5.3 Performance

A mobile phone is computationally challenged w.r.t. a desktop computer. Consider that the mobile phone we have developed on [2] has a heap size of only 1.5MB and it should be obvious that efficiency and speed are indeed an issue. In order to gain an insight into where the bottlenecks of an application might be we have run several benchmarks. We have included a barchart of the performance below. The benchmarks test the following operations: casting from int to float and vice versa, if statements, square root, method call, bit shifting, division, multiplication, addition and array allocation/assignment. We can see that in general the phone



[2]Sony Ericsson K700i

simulator emulates the performance of the phone quite closely except for operations on the double primitive datatype. We must note that the calculation of the square root of a double took 2352ms, this has been plotted as 500ms. in order to keep the rest of the chart readable. We can conclude that we must avoid operations on doubles as these are extremely expensive. An interesting result of the benchmark is that floats and ints appear to be comparable in performance.

In the setting of image processing a lot of operations must be done on two-dimensional arrays. Minimizing the amount of data access turns out to be much more fruitful than minimizing the amount of calculations. This is due to the small heap size of 1.5MB. We refer to the appendix about the optimization of convolution with a Gaussian kernel for a discussion about this type of optimization.

5.4 Implementation of algorithms on the mobile phone

We have succeeded in porting all of the evaluated methods to the mobile phone. The time-performance of running the methods on our Sony Erickson test phone are listed in figure 10. As can be seen, the methods we propose take around 10 seconds to execute on the mobile device. This time encompasses all stages of the classification process: the loading of the database, feature extraction and the comparison of the captured query image with all buildings within the database.

Algorithm	Runtime
rgb, hue and cluster combined	13 seconds
Local invariant regions	10 seconds

Figure 10: Time performance on the mobile phone

6 Discussion

We have suggested several potentially interesting techniques for mobile landmark recognition. We set out to implemented a set of simple, cheap and robust features which might be used as an initial filtering mechanism in classifying buildings and a robust algorithm capable of dealing with similarity and affine transformations. We have selected and implemented two color histogram based approaches, an approach based on histograms on the transitions between HVC colorclusters and have introduced a new algorithm based on local invariant regions. These methods were implemented in MATLAB and tested on three different datasets: the Zurich Building Database (ZuBuD) and two different flavors of a small custom database containing images of buildings on the 'Roeterseiland complex'. The first of these databases was constructed with the expectation that our histogram based approaches would show good performance, while the local invariant region algorithm would show suboptimal performance due to occlusion within the images includes in the database. The second version of the database was constructed with the expectation that the local invariant region algorithm would perform quite well, but that the histogram based approaches would show suboptimal performance due to very strong variation in the lighting conditions between the images includes in the database.

The obtained performance measures showed that the simple histogram based features managed to distinguish rather well between buildings on the ZuBuD database. If subsets of this database were selected of the size that we expect to encounter in our real life application, performance even approached 100%. This performance was quite surprising, as the ZuBuD database is an industrial standard within the computer vision literature. The local invariant regions approach also showed quite good performance on this database, classifying almost 90% of the buildings correctly. On our custom datasets, performance dropped in accordance with our apriori expectations, indicating that the various algorithms we have proposed each have their individual weaknesses. One should take these advantages and disadvantages into account when developing a real life commercial application that requires robust performance and construct the database very carefully.

We believe that there is also still much room for improvement in our current implementation of the described methods.

On the color clustering method a lot of improvement is possible by tuning the parameters of the clusterings algorithm. The biggest boost is probably the variation in the number of clusters. This probably the cause of the bad performance on the ZuBuD database.
Another possible boost could be to only look at transitions of the larger clusters in the picture since the smallest blobs are probably only noise. This could be implemented quite efficiently by first sorting the cluster number on their total size in the query image and then apply a dilation erosion on them to remove the small clusters.

There are also many possible improvements to the method based on local regions. First of all the selection of key points can be improved judging from comparisons between different views of the same building. The more key points are repeated in different views the better. Often key points of some discriminating regions are not detected in every view, because pixels around the intensity extrema often have the same intensity. In such a case none of the pixels are regarded as extrema to avoid too many key points in regions where intensities do not vary. Maybe this can be avoided without discarding useful key points. We could also consider different types of key points.

Different color spaces to represent the regions could be of use to make the representation more invariant to different illumination conditions and thus giving better matches.

Different voting schemes might also boost the performance of the classifier. Wrong matches might be detected. Not just voting for the nearest neighbor but also for a few less near ones might help as well. A problem is that such extensions require setting more parameters. Better is to take many pictures of the buildings to be recognized and use a machine learning scheme to determine how to classify best. Radial basis functions seem like a natural choice for the problem.

7 Conclusion

We have showed that an application for mobile landmark recognition is indeed feasible and that the image processing techniques required to solve the problems faced by such an application can successfully be implemented on a mobile phone to run within acceptable time. We believe that the performance obtained on the test databases shows that the application could be made robust enough for a commercial application, although more work remains to be done before the required level of robustness is obtained. We have suggested several improvements on our current algorithms which might serve to boost the levels of performance and robustness and are convinced that the required performance levels can indeed be obtained following the approaches we have outlined in this paper.

8 Appendices

A HCV colorspace

The transformation from the RGB colorspace to the HCV colorspace is acquired from the analysis of a huge amount of Munsell colors and their RGB value. From this analysis it follows that the RGB to HVC colorspace transformation can be done as follows: $\begin{bmatrix} x_c \\ y \\ z_c] \end{bmatrix} = \begin{bmatrix} 0.620 & 0.178 & 0.204 \\ 0.299 & 0.587 & 0.114 \\ 0.000 & 0.056 & 0.942 \end{bmatrix} = \begin{pmatrix} r \\ g \\ b \end{pmatrix}$
$H_1 = f(x_c) - f(y)$
$H_2 = f(z_c) - f(y)$
$H_3 = f(y)$
$where f(u) = \frac{18.51u}{u+17.58(1+\frac{5.146u}{u+30.07})}$
$M_1 = H_1$
$M_2 = 0.4H_2$
$S_1 = (8.880 + 0.966cos(\theta))M_1$
$S_2 = (8.025 + 2.558cos\theta)M_2$
$where\theta = arctan\frac{M_2}{M_1}$
$H = arctan\frac{S_2}{S_1}$
$V = H_3$
$C = \sqrt{S_1^2 + S_2^2}$

To calculate the distance between two colors we use the National bureau of standards (NBS) color distance instead of the Euclidian distance, since it was found that the NBS distance has a close relation with the difference humans perceive between two colors.

The NBS distance between color (H_1, V_1, C_1) and color (H_2, V_2, C_2) can be calculated with: $color_d is = 1.2\sqrt{(2C_1C_2)(1 - \cos(2\pi\Delta H/100)) + (\Delta C)^2 + (4\Delta V)^2}$ where $\Delta H = |H_1 - H_2|, \Delta V = |V_1 - V_2|, \Delta C = |C_1 - C_2|$

With this formulas it's possible to implement a RGB2HVC method and to build a cluster algorithm which is based on nearest neighboorhood.

B Color clustering

In this appendix a color clusterings method is described which generates a predefined number of clusters in the HVC colorspace based on the trainingsset. The clustering is based on the colorclustering in Gong. The only difference is that we are clustering on multiple objects and multiple pictures. We therefore have to compensate for the different amount of pictures taken of every object and the differences in size between those pictures. It is easy to see that these compensations are needed from the extreme case where we have a lot of photos from one object and only one from another. If would cluster by weighting all pixels the same, the color clusters which are important for the first object will not be created since, they have only a small set of pixel supporting them.

With these additions the final algorithm comes to look like this:

```
for $\forall buildings$ do
    for $\forall views$ do
        Convert the image to HVC.
        Put all the colors of one building in to a temporary histogram.
    end for
    normalize the temporary histogram and sum it to the final histogram.
end for
while $k < T_s andmax(histogram > 0)$ do
    set $seeds = \emptyset andk = 1$
    Find the maximum in the histogram and add it to the seeds if the distance to all other seeds is at least $T_{sd}$, in that case increment k.
end while
Initialize all seeds as a cluster with radius 1;
merge=true.
while merge do
    merge=false.
    Add all colors in the histogram to their closest cluster.
    update the center and radius of every pixel to the weighted average of the colors which are assigned to the cluster.
    Check wether there are two clusters for which holds that their center distance is smaller then $t_{sd}$ and their radius smaller then $T_{cr}$, if so merge the two clusters and set merge to true.
end while
```

C Optimization of convolution with a Gaussian kernel

This appendix describes the process of optimizing the implementation of the convolution operator in J2ME when the kernel is a standard two dimensional Gaussian, with zero mean and standard deviation of one. We will first describe a naive implementation and then elaborate on each optimization step.

C.1 Naive implementation

Our first naive implementation of the convolution operator considers a generic two-dimensional kernel. Application of the convolution operator with kernel K over image F resulting in the image G boils down to the following pseudocode, ignoring border conditions:

```
for x=0:imageWidth
    for y=0:imageHeight
        sum = 0
            for i=-xOrigin:-xOrigin+kernelWidth
                for j=-yOrigin:-yOrigin+kernelHeight
                    sum += F(x+i, y+j)*K(xOrigin+i, yOrigin+j)
        G(x, y) = bound(sum)
```

Note that we consider only greyscale images that are stored in two dimensional arrays of signed integers of 8 bits wide. Therefore we have to keep overflow in mind. This is solved by using a wider data type (32bit integer) for the sum variable, which is then bounded to the allowed values of the narrower data type. Also since the weights of the kernel are floats the multiplication is also done for floats.

In image processing convolution is an essential operation which is performed many times. The canny edge detector, for example, convolves the input image five times using different gaussian derivative kernels. As this operation was deemed necessary for the mobile application it was important to optimize this. Especially since it turned out that the first naive version of the canny edge detector took nearly 12seconds on the mobile phone, most of this time was consumed by the convolution operator.

C.2 Gaussian kernels

Two dimensional Gaussian kernels have the property that they can be decomposed. This entails that convolving an image by a two dimensional kernel can be accomplished by convolving an image first row wise than column wise by a one-dimensional kernel. Which results in a lot less multiplications being done, for example for a 3x3 kernel we can reduce the amount of multiplications from 9 to 6.

Implementing this in J2ME has the consequence that the quadruple for loop reduces into two triple for loops. Unfortunately this resulted in a speed loss instead of a speed gain. This is deemed to be due to the small cache size of the embedded processor which makes data access from the heap an expensive operation. The drop in arithmetic effort unfortunately does not compare to the more frequent data access, w.r.t. run time efficiency. Fortunately further investigation of the Gaussian kernel gives rise to a speedup indeed.

C.3 Bitshifting

Consider the following Gaussian kernel:

$$\frac{1}{8}\begin{pmatrix} 0 & 2 & 0 \\ 2 & 4 & 2 \\ 0 & 2 & 0 \end{pmatrix}$$

As the coefficients are all multiples of two we can implement the convolution of an image by this kernel as a sum of bitshifts. Furthermore since the coefficients sum to unity we do not have to take care of possible overflow. The only care we have take is when the weights of the derivatives of this kernel are negative, in this case we have to negate the result of the bitshift.

Implementing this optimization results in the quadruple for loop but we there are no operations on floating points nor is there an accumulator variable involved. Only bytes are manipulated which as previously discussed in the article, is extremely fast. At this point we have taken the canny edge detector from a runtime of 12seconds to less than 2 seconds.

References

[1] Comaniciu D.,Ramesh V., and Meer P., *Kernel-Based Object Tracking*, IEEE trans. on Pattern Recognition and Machine Intelligence, May 2003, vol. 25, number 5, pp. 564- 578, 2003

[2] Gevers, T. *Color in Image Search Engines Survey on color for image retrieval from Multimedia Search*, ed. M. Lew, Springer Verlag, January, 2001

[3] Gong,Y. *Advancing Content-Based Image Retrieval by Exploiting Image Color and Region Features*, Multimedia Systems, 1999, vol. 7, number 6, pp. 449-457.

[4] Harris, C. and Stephens, M. *A Combined Corner and Edge Detector*. Proc. Alvey Vision Conf., University of Manchester, pp. 147-151, 1988.

[5] Li, Y. and Shapiro, L., *Consistent Line Clusters for Building Recognition in CBIR*,

CBIR International Conference on Pattern Recognition, August 2002. Url = http://citeseer.ist.psu.edu/li02consistent.html

[6] David G. Lowe. *Distinctive image features from scale-invariant keypoints*. International Journal of Computer Vision, 60, 2, 2004, pp. 91-110.

[7] F. Mindru, T. Moons and L. Van Gool. *Recognizing color patterns irrespective of viewpoint and illumination*, IEEE Conference on Computer Vision and Pattern Recognition, vol. 1, pp. 368-373, 1999.

[8] Miyahara, M. and Yoshida, Y., *Math. Transform of (R,G,B) Color Data to Munsell (H,V,C) Color Data*, Vis. Comm. and Image Proc., SPIE, Vol. 1001, pp. 650-657, 1988.

[9] O'Hanlon, K. *Building Recognition using Computer Vision for Urban Ubiquitous Environments*. A dissertation submitted to the University of Dublin, 2005.

[10] Shao H., Svoboda T., Tuytelaars T. and van Gool L.J. *HPAT Indexing for Fast Object/Scene Recognition Based on Local Appearance*. CIVR 2003: 71-80

[11] Tuytelaars T. and van Gool L.J. *Wide baseline stereo based on local affinely invariant regions*. In British Machine Vision Conference, 2000.

[12] Zhang W. and Košecká J., *Localization based on building recognition*, Workshop on Applications for Visually Impaired, IEEE Conference, CVPR, 2005.

[13] Zheng Z., Wang, H. and EKTeoh. *Analysis of Gray Level Corner Detection.*, Pattern Recognition Letters, Vol. 20, pp. 149-162, 1999.

[14] Shao T. S. H. and van Gool L.J. *Zubud-zurich buildings database for image based recognition*. Technique report No. 260, Swiss Federal Institute of Technology, 2003.

[15] Symbian developer site:
http : *//www.symbian.com/developer/index.html*

[16] Java 2 Mobile Edition homepage:
http : *//java.sun.com/j2me/*

[17] Description of ARM Jazelle processor:
http : *//www.arm.com/products/esd/jazelle_home.htm*

[18] Sun Java Wireless Toolkit:
http : *//java.sun.com/products/sjwtoolkit/index.html*