ABSTRACT
While the problem of Content Based Image Retrieval (CBIR) and automated image indexing systems has been widely studied in the past years they still represent a challenging research field. Indeed capturing high level semantics from digital images basing on low level basic descriptors remains an issue. A review of existing systems shows that edge descriptors are among the most popular features. While color features have led to extensive work, edge features haven’t produced such active research and most current systems rather rely on completing basic edge information with other, more computationally expensive features such as texture. In this paper we propose to work on a more accurate edge feature while keeping a relatively low computation cost. We will begin with a review of common edge features used in CBIR and automated indexing systems, we will then explain our Enhanced Fast Hough Transform algorithm and the edge descriptor we derived from it. Through a study of computational complexity, we will explain that the computational burden is kept minimal and experimental results using a sample automated indexing system will show that our new edge feature significantly improves over more traditional descriptors.

KEY WORDS
Image Processing, Low level features, Hough Transform

1 Introduction
Edge Features are a very important component in human vision, and this explains the popularity of this kind of features in CBIR and automated indexing systems. Indeed they are, along with color and texture features, the most widely used descriptor for image retrieval and indexing. Furthermore, this simple combination of features has already proven effective for image retrieval purposes [6].

Information extracted from pixels or structures close to the size of a pixel are the most reliable in an image, and more global features, such as texture, are clearly very informative but also very hard and costly to characterize efficiently. If we look at color and edge information, on the other hand, both these features can be quite easily extracted and share a high level of confidence in the data they provide which make them a valuable basic tool to extract more complex data. However, precisely because they are extracted from small structures, they are difficult to localize and while spatial and structural information among these features are significant, these become difficult to extract. We propose to increase the amount of information provided by edge descriptors by extracting line segments rather than local edges thus providing structural information.

In this paper we first introduce our Enhanced Fast Hough transform which is a reliable and computationally efficient way of extracting line segments from an edge image. We will then propose edge feature types drawn from this basis with their respective qualities and drawbacks. Finally we will evaluate various algorithms with a simple color/edge based classifier using a corpus of 600 images from the internet selected to encompass as much diversity as possible. Evaluation shows that our proposed features lead to a significant improvement in classification performance for classes where edge features intuitively seem to play a significant role (forests, cities, ...).

2 Background and Related Work
All algorithms require some initial edge information to be drawn from the image. It usually is extracted through filtering, most of the time through differential operators (as per [11]), the most commonly used being Sobel and Kirsch operators eventually completed by some processing as per the well known Canny Edge Detector [2]. We may also mention the MPEG-7 standard which specifies a set of fast and simple filters for extracting specific edge directions, as cited in [10].

Having extracted base edge information we then move to feature representation. We will not consider systems that intend to seek and extract specific shape information (using active contours or any other shape characterization method such as in the SQUID cbir system [9]). Indeed those systems work only in very specific conditions (i.e. limited number of objects, fixed background, need of closed boundaries, etc.).

A very popular edge feature is the edge histogram descriptor (EHD) introduced as a classification tool in the work by Jain and Vailaya [12, 6]. It computes edge orientation from a basic filter and uses Canny edge detector to determine significant edges. Edge orientations are then put into a histogram of chosen dimensions. Indeed, histograms are a interesting way of capturing image data as they are
invariance. Although this basic feature performs quite well, it suffers from several drawbacks. As noted in [8] the lack of local neighborhood makes the edge information inaccurate (no structure information) and we may add that it also makes it very sensitive to noise. Therefore, spatial localization of features, be it relative (neighborhood) or absolute (position within the image) is the main problem. Global histograms simply ignore this problem, but we can also include neighborhood information (cooccurrence matrices, also used for color features, are a popular way of achieving this [8, 1]) as well as perform some kind of image decomposition.

Cooccurrence matrices reflect local neighbourhood of edge points, it is thus very informative because it translates local structure within the selected neighborhood. However it does not take into account edge point connectivity though the diagonal elements of the cooccurrence matrix provide information about collinearity. It also requires grouping edge orientations using coarse intervals as its dimension is given by the squared number of intervals.

Because image features related segmentation is quite difficult to achieve and is quite taxing on computer resources, it is, actually, always performed as a separate task (in order to localize all features, which is outside the scope of this paper). Therefore the only kind of segmentation which can be acceptably performed for a single feature is using a fixed grid as in [10]. This process, however, limits both translation invariance and scale invariance which only exist within a region. Region size also becomes a concern: on the one hand the bigger the regions, the bigger scale changes you will be able to process. On the other hand the bigger the region, the least meaningful spatial localization will be.

We will now describe our new edge feature that overcomes the shortcomings of the widespread edge histogram descriptor (EHD) by providing structure information through edge connectivity while still keeping a low computational complexity. We extract segment data using our proposed variant of Hough Transform (Enhanced Fast Hough Transform) and we then use segment information into for three edge features: global histogram using segment length information, segment cooccurrence matrix within a fixed neighborhood and features localized by a grid. Those features will be evaluated, compared to each other as well as compared to classical EHD and correlogram features.

3 Proposed Hough Transform

Hough transform is a well known and quite efficient tool used to extract shapes from images. We will use the classical Hough transform [4] which requires the sought features to be described in some parametric form. It is indeed convenient for shapes such as lines, ellipses or circles (a generalized Hough transform also exists for more complex patterns).

3.1 Hough Transform

Basic Hough transform is based on “accumulator cells” which record all possible sought shapes passing through each extracted edge point. This method shows several flaws: the influence of isolated pixels as well as the computing time requirements being among the biggest noticeable problems. Processing image blocks and choosing random edge points were among the proposed variants (as referenced in surveys by Leavers [7] and Illingworth [5]) in order to use Hough transform to perform efficient line segment detection. However, in terms of speed and robustness to noise, these methods remain unsatisfactory.

Our method uses Hough space to determine local connectivity rather than using it as an accumulator space, we process only edge points using precomputed trigonometric values which considerably reduce computational burden. Processing segments one after the other allows filtering segments regarding their length which reduces the impact of noise. We will now explain these processes more in detail.

3.2 Refinement and Optimization

As for any other Hough transform we start from an edge map of the processed image. Because we wish to avoid problems related to edge thickness, we use a Canny Edge Detector [2] to process our image in order to ensure a one pixel thickness for our edge map.

The base principle is as follows: for each edge point we explore each direction ranging from 0 to \( \pi \) until no edge point is found. We will store the longest line segment characterized by its length and orientation. Because we operate in parameterized space, we compute the next point in a given direction which requires trigonometrical computations that could prove costly because they are performed for each edge point and for each direction. As a first optimization, we choose to precompute those trigonometrical values. Our basic operation will be: “starting from some origin edge point, we wish to check the \( r^{th} \) point in the direction \( \theta \)”. In terms of image coordinates in a discrete space, this translates into the following cases according to the value of \( \theta \).

For \( \theta \) values between 0 and \( \frac{\pi}{2} \), \( x \) coordinates increase faster than \( y \), we thus have:

\[
x = r \cos \theta, \quad y = r \sin \theta
\]

For \( \theta \) values between \( \frac{\pi}{4} \) and \( \frac{3\pi}{4} \), \( y \) coordinates increase faster than \( |x| \), we have:

\[
x = E \left( \frac{r}{\tan \theta} \right), \quad y = r \text{ with } x = 0 \text{ and } y = r \text{ for } \theta = \frac{\pi}{2}
\]

For \( \theta \) values between \( \frac{3\pi}{4} \) and \( \frac{5\pi}{4} \), \( |x| \) coordinates increase faster than \( y \), which leads to:

\[
x = -r \cos \theta, \quad y = E (r / \tan \theta)
\]
Choosing a precision of 1°, this leads to computing $x$ and $y$ values for $\theta$ ranging from $0^\circ$ to $180^\circ$ with a $1^\circ$ step and $r = 1$. Though less critical we can then avoid multiplication operations by pre-computing values of $x$ and $y$ for various $r$ values. We then obtain a table containing the relative coordinates of the point to check according to direction and distance to the origin.

To avoid processing the same line segment twice, we remove the longest segment from the edge image after an edge point is processed. In order not to lose edge data in the process (we need it to determine neighbouring points) we also keep a copy of the original edge image. To perform some noise filtering, we will not keep information of line segments below a certain length threshold. Though information can be drawn from having a lot of short segments, we choose to ignore it in our descriptor because it is quite difficult to differentiate this kind of features from noise. Finally we can increase robustness by allowing our exploration process to check points between neighbouring values of $\theta$ or allowing some gaps within an edge segment. We thus obtain a robust and efficient segment extraction scheme. We illustrate the base extraction results with figure 1: it shows a hand drawn rectangle whose edges are not perfectly straight (left) then the extraction of our EFHT segment features with no gap or angle tolerance settings (right), then the features extracted using an angle tolerance of 2 degrees and a gap tolerance of 1 pixel (below).

### 3.3 Complexity Analysis

Algorithm complexity is quite difficult to evaluate accurately as it depends on edge point concentration. Basically we will check every point of the edge image and for each edge point we will explore its surroundings in 180 directions. Although in most cases most directions will be discarded after a few iterations, a concentration of edge points can lead to edge pixels being processed several times. It actually happens when several line segments share a common origin and line segments intersect at the same point. If we consider the average case, the complexity is linear regarding the number of edge points and in this regard the algorithm is fairly efficient. Processing times are given in the performance evaluation section. Further testing also reveals that our transform performs about 30 times faster than a randomized Hough transform on a sample set of pictures.

### 4 EFHT Features

From our EFHT, we designed three features derived from their pixel-based counterparts (i.e: edge histogram, cooccurrence matrix and spatially localized features). We provide sample feature vector values for each one, extracted from the images in figure 2.

The first feature we design is a simple normalized histogram of segment “energies”: we fix an angle interval for a specified bin and a segment adds its squared length to the corresponding bin. Indeed, simply adding segment length would provide similar results whether the segment points are connected or not. We thus decided to add squared length in order to make longer segments have greater impact on feature value. The feature construction process is straightforward: we apply a Canny Edge detector to the greyscale image, we then execute our EFHT algorithm to extract line segments with their respective length and orientations. We then choose an angle interval to group segments with close orientations thus determining histogram binning. Figure 3 illustrates features vectors obtained in this way (horizontal axis represents the histogram bins (from 0 to 36) while the vertical axis represents the cumulated squared population of the corresponding bin. We can see a distinct signature for the cityscape image with strong vertical and horizontal edges, while the mountain picture has a more uniform input but with some important diagonal edges.

The second proposed feature is there to include cooccurrence information while preserving the benefit of having edge connectivity information. The feature is built as follows: we extract line segments as specified above, although
we use coarser grouping of segmentations as it would dramatically increase feature size. We therefore choose 6 orientations for angles between 0° and 179° using the same orientation values to represent angles from 0° to 15° and 165° to 179° to emphasize that those orientations, although fairly different, represent an almost horizontal line segment. We then build a typical cooccurrence matrix by checking neighbourhood within a specified radius to see if we find a point from another line segment. We then fill the cooccurrence matrix with the segment length thus giving more importance to longer segments. Figure 4 illustrates the features values for our sample images. Because of the strong presence of collinear segments, diagonal elements were too important and affected example clarity. Diagonal elements have therefore been divided to be able to see other values more easily. Proportions among diagonal elements remain unchanged. As we can see the city image produces a quite characteristic pattern with a “+” shaped set of high values for the orientation value of 4 which represents vertical orientations and another set of high values for horizontal orientations (1). The highest values are encountered for the combinations of those orientations. The mountain image produces a more evenly distributed descriptor, but we once again remark some strong diagonal and horizontal orientations.

Figure 4. EFHT cooccurrence matrix features values for the two sample images - diagonal values were altered for the sake of clarity

The third feature is a “Grid Hough Transform” which divides the image in n*n areas and computes mean angle as well as orientation and variance for each area, leading to a $2n^2$ feature size. Because we did not want long segments to have too much weight, we chose that segment length would represent the length within the current region rather than the total length. Nevertheless segment length does have an impact on feature value as we use it as a weight when computing the average angle and the variance. Of course we have to switch to circular statistics formulae in order to compute the mean angle [3], we adapt the formulae to obtain a weighted mean of angle values ranging from 0° to 179° rather than 0° to 359°. Those trigonometric calculations could make the operation quite costly, however, sine and cosine values can once again be precomputed as we use discrete angle values, we will thus only need to compute an arctangent for each area. We also evaluated other variants of spatial features (local histograms on fewer regions, averages on more regions ...) but this one gave the best results on our testing set. We may also note that this feature is slightly smaller than the others with a size of 32 (2 features for each of the 4*4 sections). The results on the two sample images are presented on figure 5, features are numbered from 0 to 16 and represented from left to right, top to bottom. As we can see the city image still shows a lot of horizontal edges, vertical edges being quite short they shift a little bit the orientation towards 90° and produce a variance peek but are not directly visible. On the mountain image, we see a 0 value corresponding to the sky area, then strong diagonal edges with little variance on the crest line, then very noisy data in the lower part of the image. While this describes the image quite well, we can doubt the efficiency of such features for automated indexing.

Figure 5. Weighted average and variance of segment orientations in the first (left) and second (right) images

We will now evaluate those three types of features along with common edge features in a sample automated indexing task.

5 Experimental Results

Regarding our image database, we collected images from 6 categories with 100 images for each category. Those images were collected from the internet and we specifically focused on selecting images representing as much diversity as possible (in terms of point of view, scale, lighting) in order to produce a challenging automated indexing task. The chosen categories are: sunrise/sunset, seascape, mountain, beach/desert, forests/greenery, cityscape. To ensure decent base performance, we will complete our classifier with a very basic color feature: a 12 bin histogram for each of the R, G, B channels giving a feature of a size of 36.

5.1 Experiments and comments

In order to have results independent from the classification system we will evaluate the performances of various
Table 1. Comparison of classifier error rates

<table>
<thead>
<tr>
<th></th>
<th>SVM</th>
<th>MLP</th>
<th>KNN</th>
<th>C4.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>base EHD</td>
<td>34.47</td>
<td>37.77</td>
<td>43.4</td>
<td>50.07</td>
</tr>
<tr>
<td>Cooc</td>
<td>32.93</td>
<td>35.4</td>
<td>38.07</td>
<td>45.43</td>
</tr>
<tr>
<td>HHD5</td>
<td>32.27</td>
<td>36.23</td>
<td>39.4</td>
<td>48.4</td>
</tr>
<tr>
<td>HHD10</td>
<td>34.33</td>
<td>37.6</td>
<td>39.53</td>
<td>48.93</td>
</tr>
<tr>
<td>CHD</td>
<td>27.93</td>
<td>33.5</td>
<td>37.13</td>
<td>46.63</td>
</tr>
<tr>
<td>GHD</td>
<td>38.53</td>
<td>45.7</td>
<td>50.87</td>
<td>57.23</td>
</tr>
</tbody>
</table>

algorithms with 4 different supervised learning methods: support vector machines for multiple classes (using the C-SVM algorithm from the LIBSVM library; it is noted “SVM” in figures), multi-layer perceptron (noted “MLP”), K-Nearest Neighbours (KNN), C4.5 decision tree (C4.5).

Performance is evaluated using 5 successive ten-fold cross validation tests, the results obtained are shown in table 1. In this first evaluation we mention the “error rate”, which represents the number of misclassified samples divided by the total number of samples (600) averaged on the 5 successive experiments and written as a percentage value.

Regarding the segment length threshold used for Hough features we have to make a call between robustness to noise and resolution. We tested various thresholds by training our different Hough descriptors for our classification problem. It revealed that our images are not noisy enough to show a big performance difference when changing it. However classes are impacted differently: while classes that commonly show small segments (e.g. small buildings or windows in cities, grass or small trees in forests) perform better with lower thresholds, classes that are more characterized by long segments (e.g. crestlines from mountains) perform better with higher thresholds. Using the result of this experiment we set our classifiers to discard segments with a length inferior to 4 pixels as it gave the best average results.

Using these various learning methods, we will evaluate 6 different features. As a comparison basis, our first descriptor will be a basic edge histogram as per [6] calculated from edge information extracted using a canny edge detector (noted “base EHD”) and 36 edge features corresponding to angles from 0° to 180° with 5° intervals. We will evaluate an edge cooccurrence matrix as per [1] (noted “Cooc”) using 6 orientation bins and cooccurrence being detected in a 5 pixel radius. We will then use our Hough features, first a simple histogram with 5° intervals (noted “HHD5”), we then explore a different angle interval with a base Hough histogram feature using 10° intervals (noted “HHD10”). Next we use our Cooccurrence Hough Descriptor (noted CHD) with also 6 orientations (leading to a feature vector dimension of 36) and a 5 pixel radius. Our last descriptor will be the spatially localized version of our base Hough feature (noted GHD) to assess the eventual benefits of spatial localization.

It is noteworthy that a study of relative feature importance shows that edge features are the most important features in classification. We also note that among the five experiments performed for a same classifier/feature couple, the error rate showed very few variations (within +/- 0.7%).

From this first study we can draw several global observations. First of all our versions of the various classifiers bring noticeable improvements over their counterparts. Regarding the features themselves we note that cooccurrence features which translate not only edge connectivity but also local structure shown to be the most efficient. Our evaluation set being a mix of classes more or less characterized by their edge, we will then complete this first impression by studying results for each class. Regarding classifiers, we note that SVM and MLP behave similarly while KNN loses accuracy for Mountain, Beach and Seascape classes and C4.5 seems to rather impair Cityscapes, Forest and Sunrise/sunset classes more. This global behaviour is constant whichever the feature vector but some classes are slightly more impacted by some feature types rather than others. This explains the case of the “Cooc” feature and we can witness the amount of deformation is small. Finally spatial localisation performs poorly. With such generic categories we may indeed argue that the position of components matters little.

We now perform a specific study for EHD, Cooc, HHD5, CHD and GHD classifiers. We will provide precision/recall values for each class to evaluate more accurately how these features fared. As we said, overall relative performance is not really affected by the nature of the classifier type, this time we only provide data obtained using support vector machines which produced the best results. The result of the class-wise analysis is shown on table 2 (our data is expressed as percentage values).

This study backs intuitive assertions regarding the chosen classes. Indeed our features actually mostly improve for classes for which edge information seems to be important: “Cityscape”, “Forest/greenery” and, to a lesser extent, “Mountain” classes. This also confirms that cooccurrence features are the most efficient in this area. We also see that the bad results of the GHD features, while globally poor, seem to be particularly caused by “cityscape” and “Forest/greenery” categories where feature location actually matters little. Interestingly enough we find that the use of efficient edge descriptors actually lowers performance from some classes for which edge information seems to be almost useless (the “sunrise/sunset” class is a typical example of this). This seems to be due to the non mutual-exclusivity of the classes and that there actually are pictures within this class that contain elements from another class that could justify its classification in it. However it also points out that these descriptors can create noise in classes where they aren’t actually needed and the eventual necessity of adapting the classifier to take this into account.
Table 2. Precision/recall rates

<table>
<thead>
<tr>
<th></th>
<th>EHD precision</th>
<th>EHD recall</th>
<th>HHD5 precision</th>
<th>HHD5 recall</th>
<th>Cooc precision</th>
<th>Cooc recall</th>
<th>CHD precision</th>
<th>CHD recall</th>
<th>GHD precision</th>
<th>GHD recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunrise/Sunset</td>
<td>87.31</td>
<td>70.2</td>
<td>76.96</td>
<td>72.8</td>
<td>74.95</td>
<td>68.8</td>
<td>83.47</td>
<td>79.8</td>
<td>78.79</td>
<td>67.6</td>
</tr>
<tr>
<td>Seascape</td>
<td>72.26</td>
<td>67.2</td>
<td>71.81</td>
<td>64.2</td>
<td>63.53</td>
<td>55.4</td>
<td>72.75</td>
<td>64.6</td>
<td>69.85</td>
<td>55.6</td>
</tr>
<tr>
<td>Mountains</td>
<td>53.86</td>
<td>62.8</td>
<td>56.17</td>
<td>62.8</td>
<td>54.63</td>
<td>60.2</td>
<td>55.2</td>
<td>64.8</td>
<td>54.18</td>
<td>54.4</td>
</tr>
<tr>
<td>Beach/desert</td>
<td>65.7</td>
<td>63.2</td>
<td>64.57</td>
<td>59.4</td>
<td>57.97</td>
<td>53.8</td>
<td>67.63</td>
<td>60.6</td>
<td>58.78</td>
<td>61.6</td>
</tr>
<tr>
<td>Forest/greenery</td>
<td>61.79</td>
<td>65.0</td>
<td>70.66</td>
<td>76.6</td>
<td>71.77</td>
<td>78.8</td>
<td>74.85</td>
<td>77.4</td>
<td>65.74</td>
<td>66.4</td>
</tr>
<tr>
<td>Cityscape</td>
<td>59.67</td>
<td>64.8</td>
<td>68.02</td>
<td>70.6</td>
<td>78.93</td>
<td>85.4</td>
<td>80.99</td>
<td>85.2</td>
<td>49.22</td>
<td>63.2</td>
</tr>
</tbody>
</table>

5.2 performance

All tests were performed on a laptop with an intel core duo at 1.9 ghz with 2gb of ram; implementation was made using C# code. Preliminary computations (which we can consider as an offline task) take 31 milliseconds for a maximum segment length of 3000 pixels. EFHT in itself processes a 640*480 image in an average 358ms while a 1286*864 image is processed in 1057ms. Sample images were chosen in the “cityscape” category and contain a higher number of line segments. It shows that processing time stays linear regarding image size. Regarding descriptors they add some processing as such, even if most of the feature collection and organization job is done during the segment extraction process. Indeed histogram is processed directly, correlation matrix requires a new pass on extracted segments from which we explore local neighbourhood which can become costly if we increase the explored radius. If we repeat our tests we see that the correlation matrix features roughly doubles extraction time. Finally the spatially localized features add two passes among the line segments in each region (one to compute the mean and another for the variance). Processing times are also a little bit more than doubled which reveals that the load increase is significant but complexity stays linear. Better optimization of the code (using c++, optimizing it for multi-core processors, etc.) should improve these results.

6 Conclusion

In this paper we propose a new variant of the Hough transform (EFHT) which extracts line segments efficiently and robustly. We used this extracted data to improve over classical edge descriptors. We also proposed a version of those descriptors which uses data extracted through our proposed EFHT and tested them on a sample automated indexing task. It appears that our algorithms globally outperform their counterparts. We now intend to work on the classification task and on the automated indexing problem using our Hough Transform descriptor and combining with other efficient features. We noticed that we could need to adapt our descriptors to the probable outcome of the classification and therefore intend to work on this area. We can also further improve our descriptors, for instance by adapting the length thresholds.

References