Image Disorientation Auto-Recovery

Jia Wang, Yin Zhang, Zhiyuan Chen, and Yu Zhang
Department of Computer Science
Cornell University
Ithaca, NY 14853
jiawang@cs.cornell.edu, *yzhang@cs.cornell.edu*,
zhychen@cs.cornell.edu, *yuzhang@cs.cornell.edu*

Abstract

Automatically detecting and correcting disoriented image frames in a content-related image sequence is a problem that must be addressed in many image and video applications. In this paper, we give a robust feature-based algorithm to efficiently detect and adjust the disorientation of images.

Keyword image orientation, crop, Robert edge detection, line feature, Hough transformation.

1 Introduction

In image and video applications, the image input sequence which we get in the real world may have some of the image frames severely disoriented (e.g. upside down or rotate by around 90degree) due to the arbitrary position and rotation of the camera. To facilitate the subsequent image processing in these applications (severely disoriented image may be regarded as invalid input), we must "normalize" the input images, i.e. automatically detect and adjust the directions of the "outlier" image frames. Though simple at the first glance, it is in effect tricky because no effective image operations so far is suitable to serve as the "one-hit" solution. Based on comparison and experimentation on several possible methods, we end with a robust and efficient approach.

Detecting and correcting any slightly tilting and rotation of the image is a very hard problem and in itself a topic for more ambitious and intensive research. At the same time, narrowing the general disorientation down can still meet the practical requirements of many of the applications. So we focused our efforts on attacking severe image disorientation. We made several assumptions about the scenarios in which our algorithm can best be deployed. First, we assume that the image may have one of the four possible orientations respective to 0, 90, 180, and 270 degree clockwise rotations of the original image: UP, $RIGHT$, $DOWN$, and $LEFT$ (see Figure 1). Secondly, we assume that the scenes in image frames have related content hence similar features. For instance, the image set could be some pictures we take in an indoor environment, or image frames digitized from video we shot at a BBQ party. Finally, we assume that there are some reference(well-oriented) images available which specify the correct orientation for the whole image collection. With the last two assumptions, detecting possible disorientation of a target image can be done by comparing its features with the features in the reference images.

The paper is organized as follows. Section 2 briefly describes several image features that we visited and experimented, including histogram, straight lines, and surface planes. We will point out both successful cases and the lessons we learned in unsuccessful cases. The detailed algorithm of extracting line features and using them to detect image disorientation is discussed in Section 3 and Section 4 respectively. Experimental results are shown in Section 5. Finally, we draw some conclusions and discuss future works.

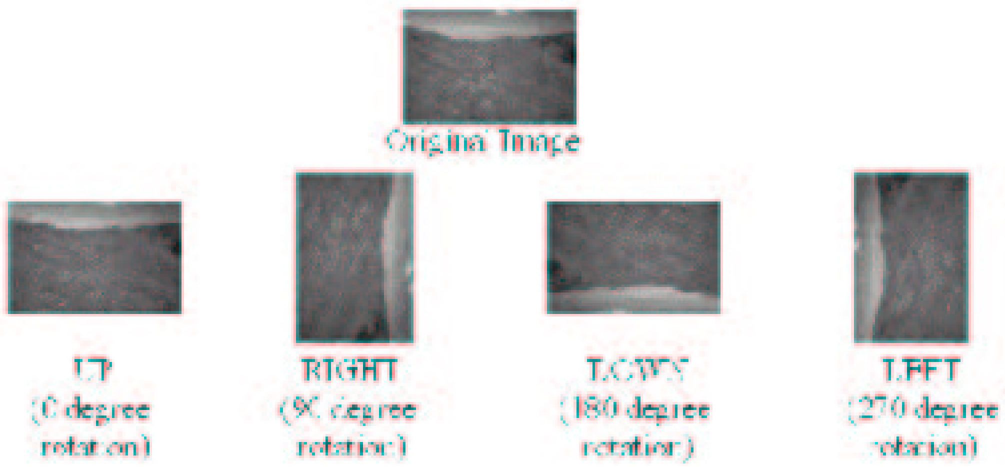

Figure 1: Four possible image orientations

2 Image Features

To detect the orientation of target image according to reference images, we need extract features from both of them and compare these features. The correctness and efficiency of such an approach highly depend on what features are used and how they are extracted from image and used in the detection procedure. Several image features have been considered. One of them is *histogram*. Different parts of an image may show a different histogram pattern, which could potentially provide some clues of the orientation of the image.

The second feature that we could take advantage of is the *straight line* feature of the edges in the image. If there are significant amount of lines aligned on one particular direction or the densities of the straight line segments at different part of the image are different, it would be exploited to identify the correct orientation of the image.

The third option is the direction of the *surfaces* in the image. The direction and density of the normal vectors of such surface planes are some features that could indicate the orientation of the image.

2.1 Histogram

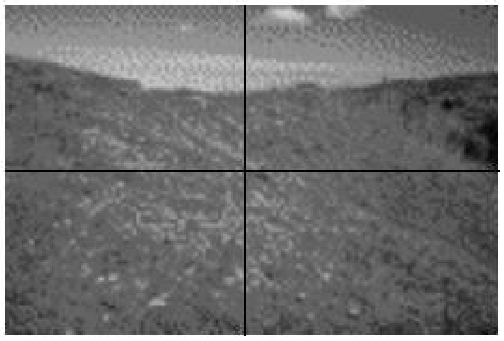
Figure 2: Four partitions of the image

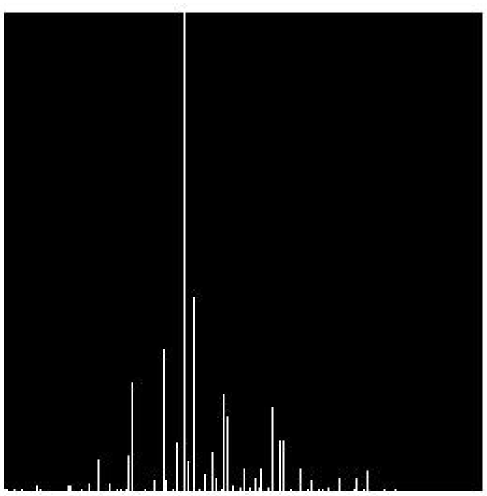
Figure 3: Histogram of the image

At first glance, utilizing histogram to determine image orientation seems very promising. The basic approach could be to divide the entire image into 2×2 four subimages as shown in Figure 2 and compute the histogram of each of the four subimages. If the image is rotated by 90, 180, or 270 degree, the subimages will just be exchanged with each other, so do their histograms.

And different rotation angle corresponds to different exchanging patterns. If the target image and reference images display similar feature in terms of their four subimage histograms, by analyzing the exchanging pattern of the subimage histograms, we can decide the rotation angle, detecting and correcting disorientation. Figure 3 gives the histogram of the entire image and the histogram of four subimages are shown in Figure 4.

Unfortunately, our conjecture is proved to be uncorrect. Though images in the sequence are content-related, the histogram of their subimages can exhibit features quite distinct from each other – in other words, histogram are somehow sensitive to modest change between images and random noises, and cannot serve as a stable feature we could compare with and utilize.

2.2 Straight Lines

Another feature coming naturally into our mind is straight lines in the images. And experiments applying this approach display its suitability in serving our purpose. Two kinds of straight lines are used: "horizontal" and "vertical". For pratical purpose, "horizontal" ("vertical") lines actually refer to those straight lines which lie within the range [-10, 10] degree of angle to the abstract horizontal (vertical) axis (see Figure 5). Generally, objects in the natural scene have most of their edges lying approximately in the horizontal or vetical directions, and edges lying in other directions are not very helpful to detect the image orientation. So we didn't take other directions of straight lines into account.

We use Robert edge detection and Hough transformation to extract straight lines. It turns out that the straight line could be a very powerful feature to determine the the image orientation. The detailed feature extraction procedure will be discussed in Section 3. (Note: The major reason that we choose Robert edge detector is that it's fast.)

3 Feature Extraction

In this section, we describe our straight line extraction algorithm in detail. As we mentioned in Section 1, only horizontal lines and vertical lines are

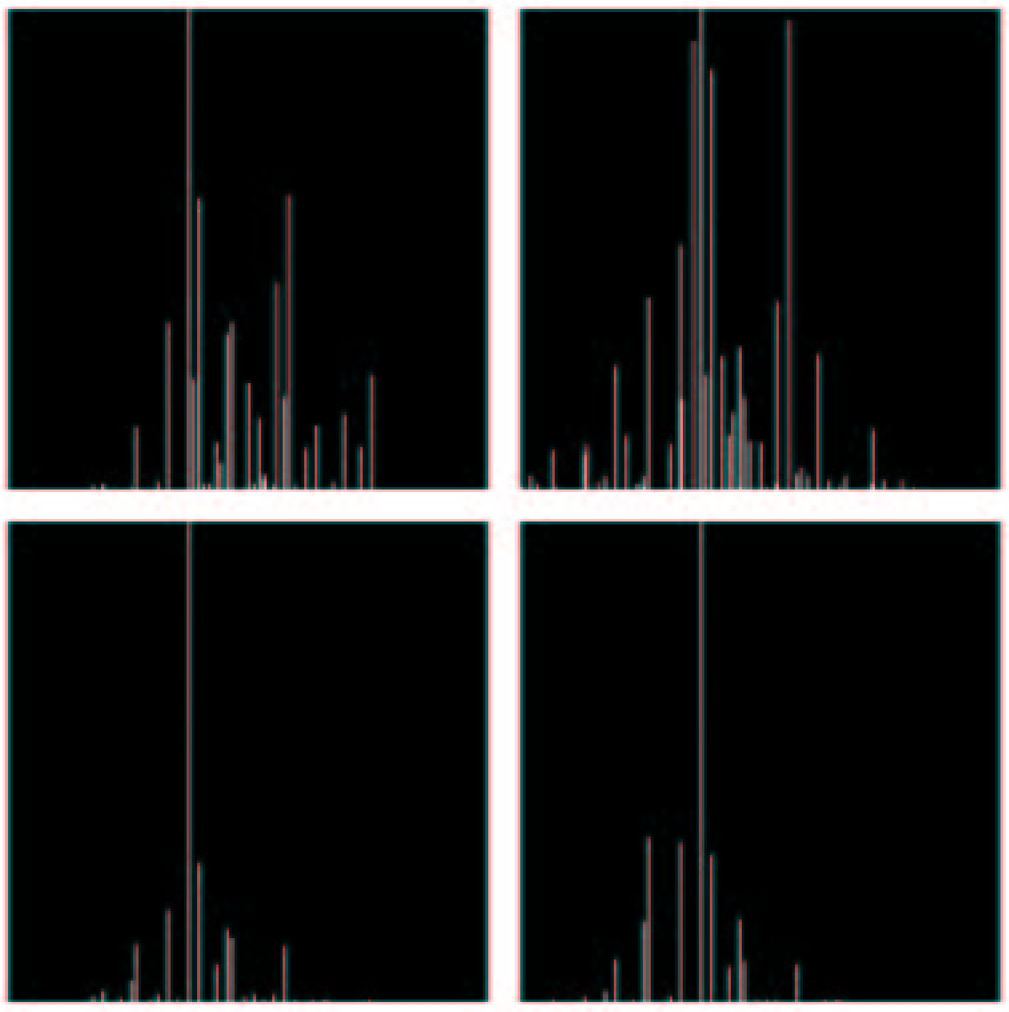
Figure 4: Histograms of four subimages

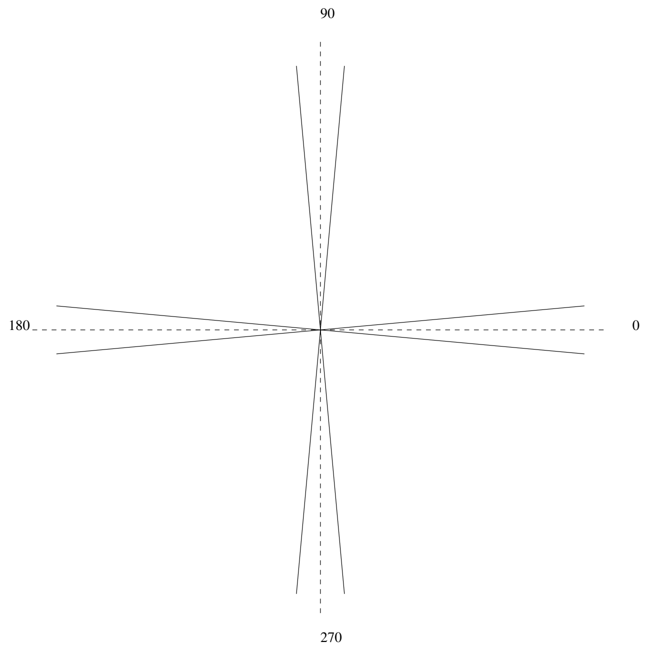

Figure 5: Horizontal lines and vertical lines

considered. There are three main steps in the line feature extraction procedure: noise filtering, edge detection, and straight line extraction.

3.1 Noise Filtering

To make our algorithm more robust, we need some technique to get rid of the noise in the image as much as possible. Two of such techniques are *cropping* and *thresholding*. We first apply cropping operation to cut off the part of the image close to the boundary (see Figure 6), in order to eliminate possible noise on the image boundary. How many pixels close to the boundary are cropped depends on the size of the image to aviod undue loss of information.

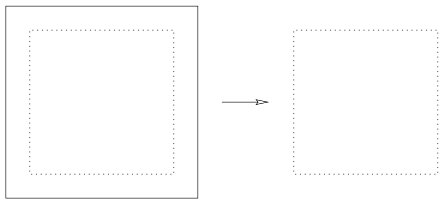

Figure 6: Crop operation

Next, the cropped image is thresholded. The simplest way to do it is to choose a specific value as the threshold H, say $H = 128$, and apply the following operation on each pixel in the image.

$$I'(i,j) = \begin{cases} 0 & I(i,j) < H \\ 255 & I(i,j) \geq H \end{cases}$$

But too much information will be "filtered out" too using this naive method. Therefore, we adopt a more sophisticated method. First, we compute the histogram of the cropped image and find the peaks in the histogram diagram (see Figure 7). The thresholding is performed by replacing the intensity value of each pixel in the image with its closest peak value which is smaller than or equal to itself. An example is shown in Figure 8. The image on the right side is the thresholded result of the image on the left side. You can see clearly from the image that most noise is filtered out.

Now, the result image only has discrete peak intensity values. We call this image *clean image*. Noise has been eliminated while enough informa-

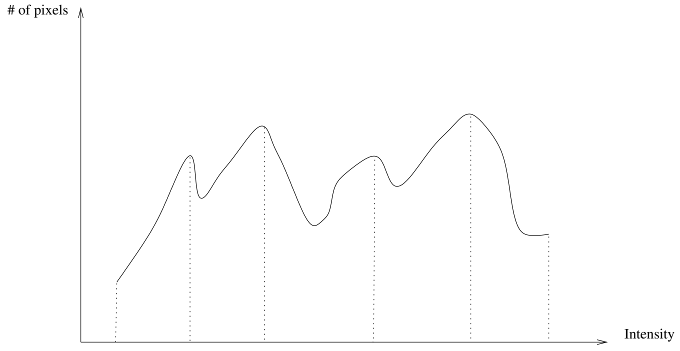

Figure 7: Peak values of the image histogram

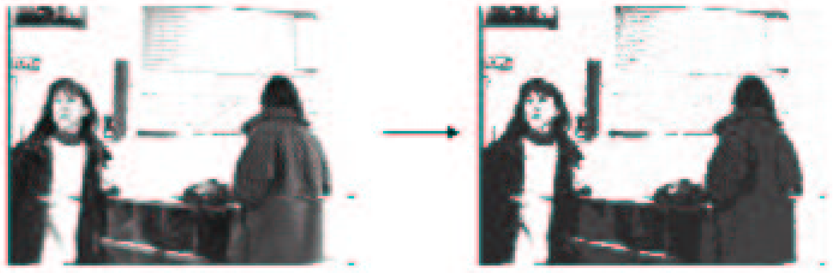

Figure 8: Thresholding the image

tion has been kept for the edge detection described in the following section.

3.2 Edge Detection

In this stage, we apply the Robert edge detection to the clean image obtained from Section 3.1. There are various kinds of edge detectors available. The correctness of our image orienation detection doesn't depend heavily on which edge detector is used. The reason we choose Robert edge detection is its efficiency. Note that pixels in the result edge image of Robert edge detection can have multiple intensity values (not only 0 and 255). To get an edge image with only two possible intensity values, 0 and 255, we apply thresholding operation on each pixel in the edge image with threshold k (see Figure 9).

$$I'(i,j) = \begin{cases} 0 & I(i,j) < k \\ 255 & I(i,j) \geq k \end{cases}$$

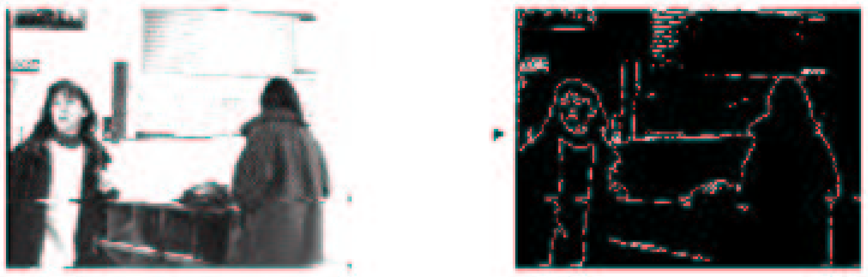

Figure 9: Edge detection and thresholding

The value of k is chosen adaptively. The motivation behind is that we want the edge image to have proper amount of edges. If k is too small, there are too many edges in the result edge image. On the other hand, excessively large k will leave us too few edges to perform subsequent line extraction. The optimal value of k is determined by binary search. Figure 10is the flow chart of this searching procedure. The final edge image, called *clean edge image* is ready for straight line extraction.

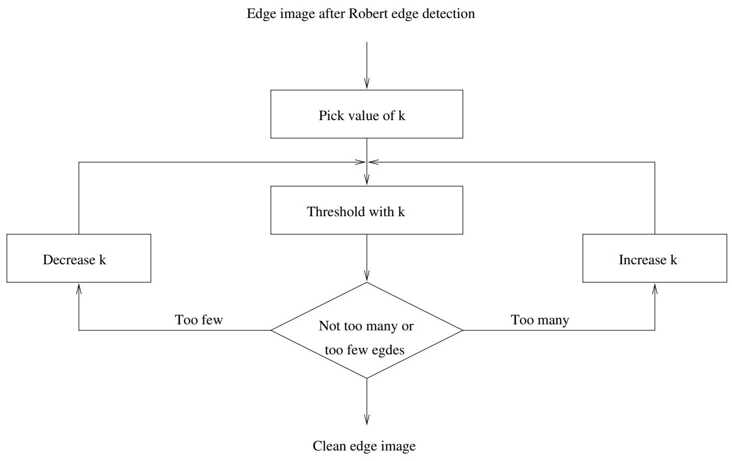

Figure 10: Adaptively choose value of threshold k

3.3 Line Extraction

The final step of feature extraction is to extract straight lines from the clean edge image. Several straight line features helpful to determine the image orientation have been extracted. We first extract the horizontal lines and the vertical lines in the entire clean edge image by applying Hough transformation on it twice (one for horizontal lines and the other for vertival lines). The result are shown in Figure 11 and Figure 12. Similarly, the threshold value used in Hough transformation is chosen adaptively using binary search.

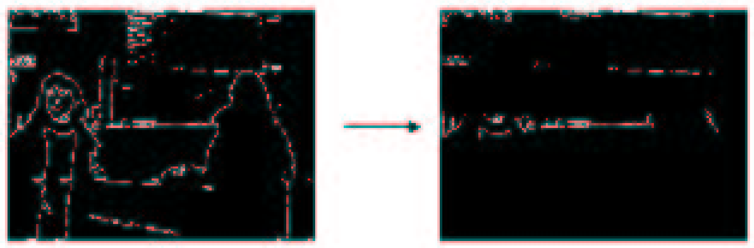

Figure 11: Horizontal lines extracted using Hough transformation

Next, to exploit the line feature even further, we extract line features from different parts of the image to find out some patterns of straight line distribution. The clean edge image is divided into three parts both horizon-

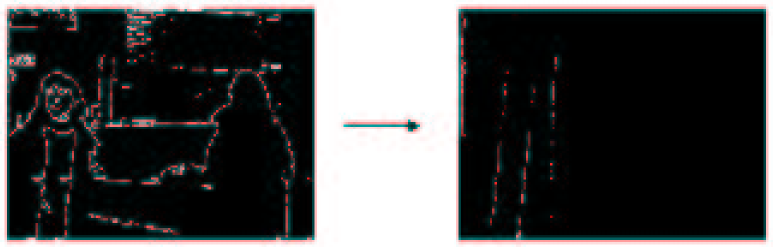

Figure 12: Vertical lines extracted using Hough transformation

tally and vertically as shown in Figure 13. The horizontal lines in the upper and lower parts and the vertical lines in the left and right parts are extracted by Hough transformation. If patterns such as ”the upper part has more horizontal lines than the lower part” or ”the left part has more vertical lines than the right part” are found in the reference images (thus is potentially a pattern occurring in most of the images), they can be utilized to detect the oriention of the target image. The middle part of the image in both horizontal and vertical cases is ignored because our test shows that seldom obvious pattern appears in the middle of the image, thus information there should be discarded to enhance the contrast in the upper-lower or left-right pattern.

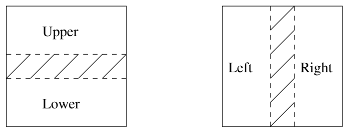

Figure 13: Horizontal and vertical partitions of the image

So far, we have the information of the image line feature, including the total amount of horizontal and vertical lines of the entire image, the amount of horizontal lines in the upper and lower parts, and the amount of vertical lines in the left and right parts. Below we will show how all this information is exploited to effectively determine the orientation of the image.

4 Detect and Correct Target Image Orientation

Now we are well on the way toward the detection of target image orientation. Recall that we have several reference images with one or more target images. Our detection algorithm is outlined below.

1. get features from reference images

   for each reference image:

   - *extract straight line features*;
   - *compute the statistics of those line features.*

We use two groups of statistics here. The first group includes ratio of the reference images in which most lines are horizontal, and ratio of the images in which most lines are vertical. They are used to decide the "big" orientation of the image (see Figure 14). The second group includes ratio of the reference images in which most of the horizontal lines are in the upper(lower) part, and ratio of the reference images in which most of the vertical lines are in the left(right) part. They are used to refine the "big" orientation to one of the two specific orientation.

1. detect and correct the orientation of the target images

   for each target image:

   - extract straight line features;
   - determine the image orientation;
   - if disoriented, rotate the image to the correct orientation.

Here, when extracting the line features, we also compute two group of ratios. First, the ratio of the straight lines in horizontal direction and the ratio of the straight lines in vertical direction. Second, the ratio of the horizontal lines in the upper (lower) part and the ratio of the veritical lines in the left (right) part. These ratios are exactly corresponding to the statistics of the reference images and serve as the basis of the comparison below.

To determine the target image orientation, we compare the ratios computed from the target image with the statistics computed from the reference images. The outline of the algorithm is listed below.

1. determine "big" orientation of the target image by comparing the first group ratios of the target image and the first group statistics of the reference images. If they are consistent in the sense that lines of most reference images and of the target image are both horizontal (or vertical), the "big direction" is UP_DOWN (rotating by 0 or 180 degree), otherwise $LEFT\_RIGHT$ (rotating by 90 or 270 degree).

2. determine specific orientation of the target image by comparing the second group of ratios of the target image and the second group of statistics of reference images. If big direction is UP_DOWN, and most of the reference images and the target image have the same pattern (more horizontal lines on the upper part than the lower part or vice versa, or more vertical lines in the left part than the right part or vice versa), the direction is UP (rotating by 0 degree), otherwise the direction is DOWN (rotating by 180); If big direction is $LEFT\_RIGHT$, and rotating the pattern of most of the reference images by 90 degree ends with the pattern of the target image, the direction is LEFT, otherwise RIGHT.

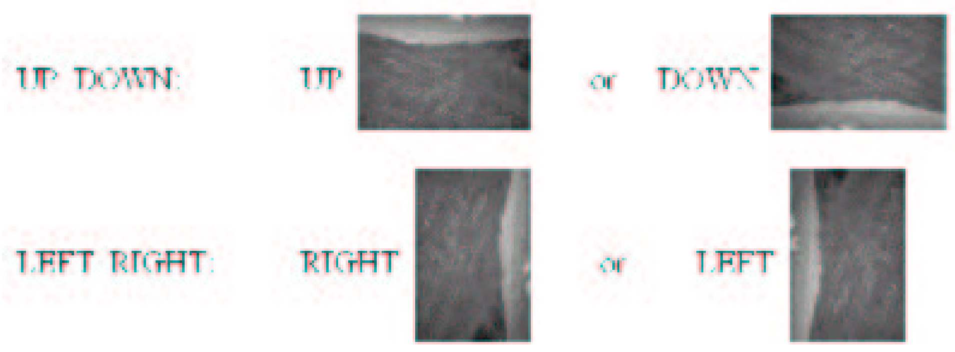

Figure 14: Big-orientation of image

5 Experimental Results

We have implemented our image orientation detection algorithm in VC++. To test its accuracy and speed, we run the algorithm on images of both outdoor scenes and indoor scenes. The results are shown in Table 1.

Image	Ourdoor Scene	Indoor Scene
Number of image frames	80	130
Number of references	15	20
Number of fails	14	11

Table 1. Experimental results

The first image set shows ourdoor scene. The common characteristic of these 80 image frames is that they all have the mountain, the trees, the grass, the sky in the image. We pick first 15 frames as the reference images to indicate the correct orientation. After extracting features from the 15 reference images, we randomly rotate each frame by one of 0, 90, 180, 270 degree, and run the algorithm on the entire image set. Among these 80 images, our algorithm successfully detected the orientation of 66 of them and fails on 14 of them.

The second set of images is indoor scene image frames digitized from a piece of video shot in the Cornell Campus Store. It consists of 130 frames. Since the camera poistion doesn't move, the background of these images remains the same. However, since there are a lot of people walking around, some part of the background is occluded by persons and other foreground objects. Therefore, considerable differences exist between any two images. In this example, we pick first 20 images as references. The final result shows that our algorithm is able to detect the orientation of 119 frames but fails on 11 frames.

We should point out that the algorithm doesn't require the reference images to be chosen as the first several images. Randomly choosing from the image set is another approach we could adopt. The way to choose reference image doesn't affect much on the accuracy of results of our algorithm. As long as the majority of the references have the correct orientation, our detection algorithm will still win.

Also, an interesting fact we observed is that our algorithm is very robust on determining the "big" orientation of the images.

6 Conclusions and Future Works

Detecting and correcting heavily disoriented images in a content-related image sequence is a challenging problem reoccuring in many image and video applications. In this paper, a robust algorithm to solve this problem is provided. Based on straight line features in the edge image, the algorithm exploits the horizontal and vertical line information presented in almost all natural scenes. Sophisticated noise filtering technique using histogram information makes our algorithm relatively insensitive to environment noise. Using Robert edge detection and Hough transformation for straight line extraction as building blocks enhances efficiency.Moreover, we propose the method of adaptive thresholding and straight line distribution pattern matching to bring the simple straight line feature into full play. The result of our experiments on both indoor scene images and outdoor scene images are quite promising. It has vividly shown that in some circumstances exploiting the potentials of a simple approach can achieve some goal which seems need more complicated techniques at the first sight.

Currently the disorientation detection and correction is only for grayscale images. In case of color images, we can take advantage of the feature related to color information to possibly provide more powerful functioinality. In addition, some kind of optimization could be done to improve the efficiency.

Acknowledgement We wish to thank Professor Ramin Zabih for his original suggestion of the topic and valuable advice and help all through our work. Also most of our image processing algorithms are implemented using the CVIPtools library from the EE department of South Illinois University Edwardsville Campus (SIUE).

Reference

1. CVIPtools from SIUE, http://www.ee.siue.edu/CVIPtools/