Separation of sky and background pixels in hemispherical forest images

MTAT.03.260 Pattern Recognition and Image Analysis

Dmitri Melnikov

1 Introduction

Hemispherical images are often used to study plant canopy and solar radiation. The images are taken by placing the camera with an extremely wide-angle lens to look upwards such that the viewing angle is nearly 180-degrees [1]. Figure 1 shows an example of a hemispherical image.

Researchers often use these images to determine, for example, the transparency index of the forest – how much sunlight do the plants actually get. To facilitate this process it would be convenient to use a tool that could automatically separate the sky regions from the non-sky regions. The result would then be a binary image where the sky pixels are white and everything else is black. The transparency index can be later easily computed from such binary images.

One possible way of creating such a tool is to use the thresholding approach. In this case it was preferred because the images have a well-defined format (hemispherical) and thresholding is a fast and simple operation to perform even on very high resolution images.

Another goal was to make sky separation as automatic as possible, meaning that ideally the user has to only specify the input image without worrying about tuning any parameters.

Figure 1: Example of a hemispherical image

2 Thresholding

It was found that the thresholding approach indeed works well for this type of task, however due to the fact that the images are taken during different time of the day and under different weather conditions, it is not surprising that one manually chosen threshold does not fit all images.

To automate the process, an algorithm for finding the global threshold has to be used. There are a number of different algorithms that can do this, but in this work we shall focus on 3 of them: *iterative method, minimum method* and *edge neighbourhood method*. Figures 3-5 show the resulting images obtained by using these algorithms on the original image in figure 2.

Also, a single global threshold still seems to ignore many small local details such as branches and leaves, to try to resolve this issue we combined automatic methods with *adaptive thresholding*.

2.1 Iterative method

The following iterative method was used to select the threshold automatically [2].

1. Select an initial threshold *T*. In our case, the mean intensity (computed from the image histogram) was used.
2. Image pixels are partitioned into two sets: pixels with intensities below T and pixels with intensities equal and above *T*. Lets call these two sets *G1* and *G2*.
3. The mean intensity of the set *G1* and the mean intensity of the set *G2* is computed. Lets call these *m1* and *m2*.
4. The initial threshold *T* is set to the average of *m1* and *m2*: $T = (m1 + m2)/2$.
5. The process is repeated from step 2 with the updated threshold *T* until it does not change any more.

The average gray level as an initial value for *T* is a good estimate when the background area does not dominate in the image. Since the sky (the foreground) usually has a relatively large area, this initial value for *T* is suitable in practice.

The algorithm does not take many steps to converge, usually only 4-6 iteration are required.

2.2 Minimum method

This approach [4] finds the threshold in the valley (minimum) between two peaks (maxima) in the histogram. This method is not expected to work for all images since some histograms do not have clearly defined valley points.

2.3 Edge neighbourhood method

This method requires running an edge detection algorithm such as Canny's algorithm first. After that the threshold is computed as an average of intensities around the found edges. The idea behind this is that the edges usually separate the regions of different gray levels. In this work the neighbourhood window size of 3x3 pixels was found to be sufficient, larger windows do not produce significantly different results.

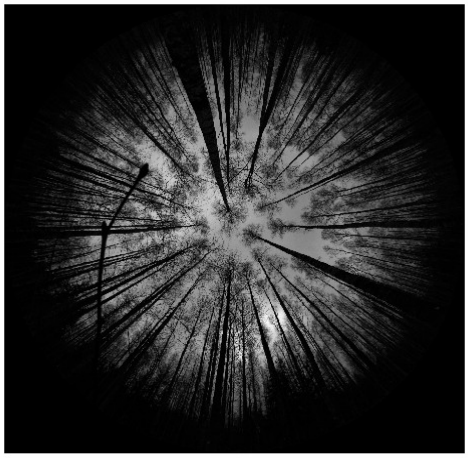

Figure 2: Original image

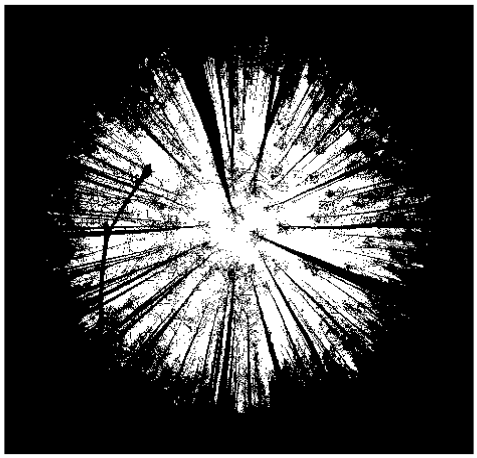

Figure 3: Iterative method

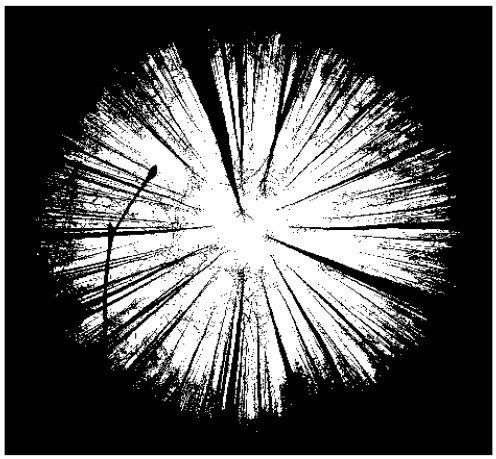

Figure 4: Minimum method

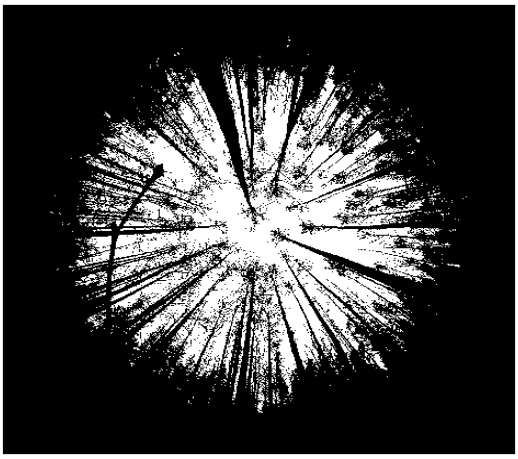

Figure 5: Edge neighbourhood method

2.4 Adaptive thresholding

Global thresholding algorithms produce a single value and all pixels are expected to be either below or above this value. Sometimes, however, this is not flexible enough. Instead we might be interested in knowing only how different is the pixel from its local neighbourhood.

Adaptive thresholding [3] (also called local or dynamic) calculates the threshold for each pixel in the image taking into account only its surrounding neighbourhood. A number of different functions can be chosen to calculate the local threshold for the centre pixel, such as the mean, the median or the average of the minimum and maximum values of the neighbourhood. In this work the mean was used with the neighbourhood window size of 11x11 pixels.

When using adaptive thresholding one must keep in mind that in areas where the neighbourhood has nearly constant intensity the mean is not a suitable threshold. This can be improved by subtracting a small constant C from the mean. In the current work 0.025 was a suitable value for C (when image intensities were normalized to [0, 1]).

The main advantage of using adaptive thresholding is that it can reveal small details such as leaves and branches that disappear when only the global thresholding is used.

2.5 Combination of automatic and adaptive thresholding

Using the mean-C as a function for adaptive thresholding implies that the constant intensity areas are marked white even though it might be a constant area of the trees in the image and should be black. Thus it would be fair to expect the best results to be achieved by combining the output from automatic and adaptive algorithms. Since their output is binary, a *logical AND* operation can be used for this. We test all 3 automatic algorithms with and without adaptive thresholding.

The result of combining automatic and adaptive thresholding is shown in figures 6-8.

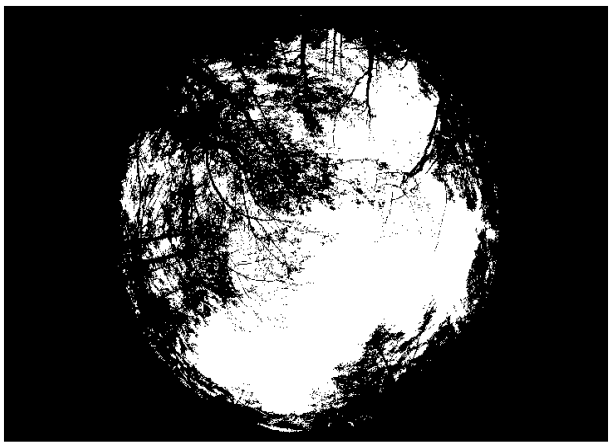

Figure 6: Thresholded image

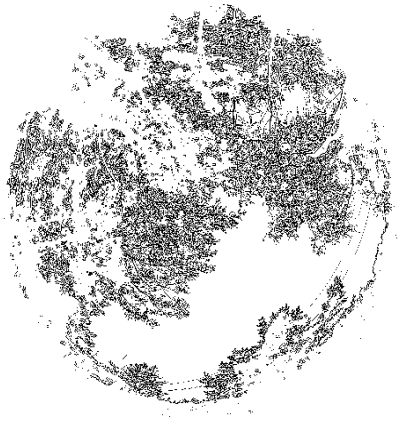

Figure 7: Adaptive thresholding

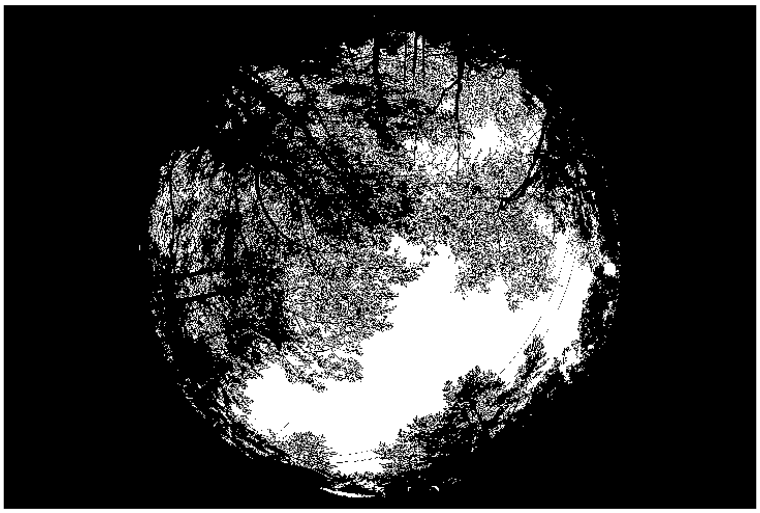

Figure 8: Combined image

3 Measuring quality

Quality was measured using the provided data of transparency estimates. This estimate data is not fully correct and may contain some errors, but it still allows to approximately measure the quality of the automatic thresholding algorithm.

First, transparency data was divided into 16 non-overlapping parts around the centre. Then the average for each part was found. This average was compared with the ratio of white/all pixels of the same part in the binary output of the algorithm. The differences of these two values were used to calculate the variance. Naturally, the smaller the variance, the closer an automatic thresholding is to the transparency estimate data, the higher is the quality.

4 Results

Below are the tables that summarize the results for each of the 3 algorithms.

Image file	Threshold	Time (s)	Variance / Variance with adaptive
FE_IMG_1115.rst	191	0.023257	0.0013 / 0.0019
FE_IMG_0186.rst	424	0.016717	0.0153 / 0.0040
FE__MG_1824.rst	89	0.018446	0.0001354 / 0.0001874
IFE__MG_1380.rst	5455	0.028833	0.0091 / 0.0100

Table 1: Results for iterative method

Image file	Threshold	Time (s)	Variance / Variance with adaptive
FE_IMG_1115.rst	243	0.178239	0.0037 / 0.0049
FE_IMG_0186.rst	227	0.153303	0.0398 / 0.0034
FE__MG_1824.rst	163	0.156888	0.0018 / 0.0018
IFE__MG_1380.rst	217	0.181914	0.1199 / 0.0381

Table 2: Results for minimum method

Image file	Threshold	Time (s)	Variance / Variance with adaptive
FE_IMG_1115.rst	117	2.652710	0.0040 / 0.00035692
FE_IMG_0186.rst	479	3.341964	0.0117 / 0.0059
FE__MG_1824.rst	40	1.808587	0.0011 / 0.00048987
IFE__MG_1380.rst	3200	2.365197	0.00090082 / 0.0011

Table 3: Results for edge neighbourhood method

As seen from the tables 1-3, iterative method and edge neighbourhood method produce the best results, however the first one is significantly faster than the second one. Perhaps a better implementation of edge neighbourhood method can be made to run faster, but it is still unlikely that it would outperform the iterative method since we first need to run the edge detection algorithm. Also, increasing the window size would slow it down even more.

Including adaptive thresholding does not seem to increase the quality as much as expected. The variance may even become slightly larger after adding adaptive thresholding. With iterative method and the given quality measure, we cannot say that using adaptive thresholding gives any advantage.

5 Conclusion

Sky separation is a well studied problem and the thresholding approach is just one of many possible ways to solve this problem. Although simple, thresholding can give acceptable results for many applications where extreme precision is not required.

The great advantage of automatic thresholding algorithms is that they require no parameter tuning from the user thus making them very simple to use. There are a number of such algorithms, three of which were compared in this work.

Iterative method proved to be the most efficient in both speed and accuracy for most images. Whether combining automatic with adaptive thresholding can improve sky separation quality is still not clear and requires further study.

6 References

[1] http://en.wikipedia.org/wiki/Hemispherical_photography

[2] Rafael Gonzalez, Digital Image Processing, 2nd edition, p. 599.

[3] http://homepages.inf.ed.ac.uk/rbf/HIPR2/adpthrsh.htm

[4] C. A. Glasbey, "An analysis of histogram-based thresholding algorithms", 1993