Crisan Adrian
30433

Enhancing readability of scanned picture books

Finding the text areas

1. Proposed problem

The problem that I had to solve was to get the text areas from a picture in which text can be found, for example a scanned page of a book. The algorithm determines which pixels of an image correspond to text and using a couple of algorithms, it highlights the location of text by forming text blocks.

2. Text location algorithm

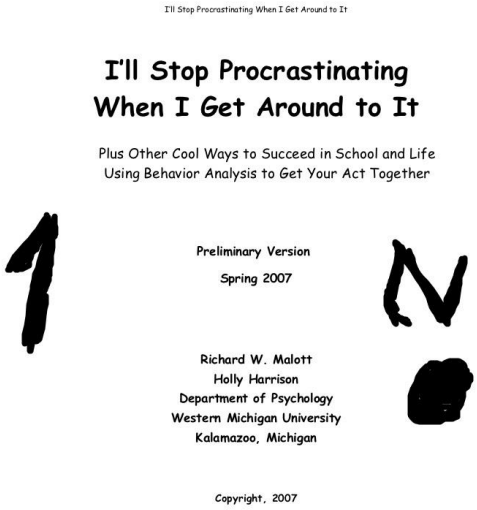
The algorithm that finds the location of the text in the picture consists of 5 steps:

- Color thresholding
- Finding the connected components
- Cleaning the connected components
- Dilation of the result
- Finding the text areas

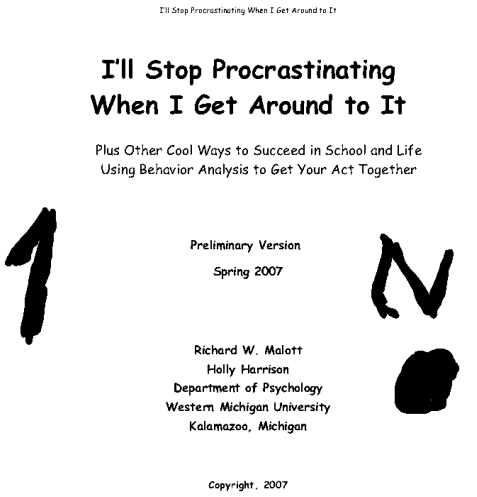
2.1. **Color thresholding**

This steps consists in taking the initial photo and transform it into a binary photo, with 0 (black) corresponding to the darker areas of the picture, the text, and 255 (white) corresponding to anything else.

For example, for the initial photo:



The thresholding will result in:



Which is pretty much the same photo because scanned text pictures tend to be black and white.

The core of the algorithm:

```
if (image.at<uchar>(i, j) < 135) {
	thresholded.at<uchar>(i, j) = 0;
}
else {
	thresholded.at<uchar>(i, j) = 255;
}
```

2.2. Finding the connected components

In order to find the connected components we apply a BFS search on the thresholded image and we search for grouped pixels and give them an unique label. As a result we have a matrix of the image size containing all the connected components.

The core of the algorithm, using a 4-neighborhood:

```
int di[4] = { -1,0,1,0 };
int dj[4] = { 0,-1,0,1 };
if (image.at<uchar>(i, j) == 0 && labels.at<float>(i, j) == 0) {
	std::queue <Point2i> Q;
	label++;
	labels.at<float>(i, j) = label;
	Q.push(Point2i(i, j));
	while (!Q.empty()) {
		Point2i q = Q.front();
		Q.pop();
		for (int k = 0; k < 4; k++) {
			if (q.x + di[k] < 0 || q.y + dj[k] < 0 || q.x + di[k] > image.rows
|| q.y + dj[k] > image.cols) {
			continue;
			}
			if (image.at<uchar>(q.x + di[k], q.y + dj[k]) == 0 &&
labels.at<float>(q.x + di[k], q.y + dj[k]) == 0) {
				labels.at<float>(q.x + di[k], q.y + dj[k]) = label;
				Q.push(Point2i(q.x + di[k], q.y + dj[k]));

				}
			}
		}
	}
```

2.3. Cleaning the connected components

This step is necessary in order to eliminate all the connected components found that are not text. The algorithm removes the connected components that have a size larger that any known font.

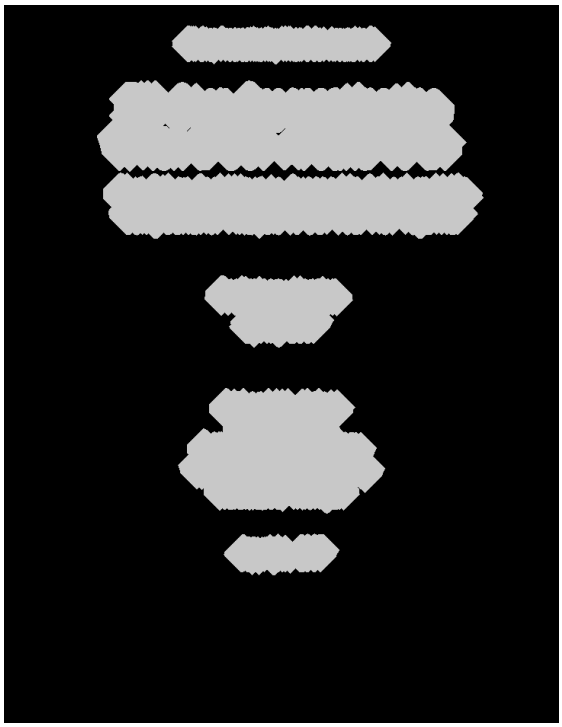
The algorithm stores in an array at the index equal to the label the sum of all the pixels equal to the label:

```
for (int i = 0; i < image.rows - 1; i++) {
	for (int j = 0; j < image.cols - 1; j++) {
		arr[(int)image.at<float>(i, j)] += 1;
	}
}
for (int i = 1; i < 3000; i++) {
	if (arr[i] > 1000) {
		for (int m = 0; m < image.rows - 1; m++) {
			for (int n = 0; n < image.cols - 1; n++) {
				if (image.at<float>(m, n) == i) {
					result.at<float>(m, n) = 0;
				}
			}
		}
	}
}
```

2.4. **Dilation of the resulted cleaned connected components**

This step is basically the formation of the text blocks. It dilates the resulted cleaned connected components by 20 pixels in order to obtain a continuous block of pixels representing the area in which text can be found.

The result of the dilation algorithm on the above picture:



As we can observe, the algorithm detected only the text areas and left the residual connected components out.

The algorithm for dilation, applied 20 consecutive times:

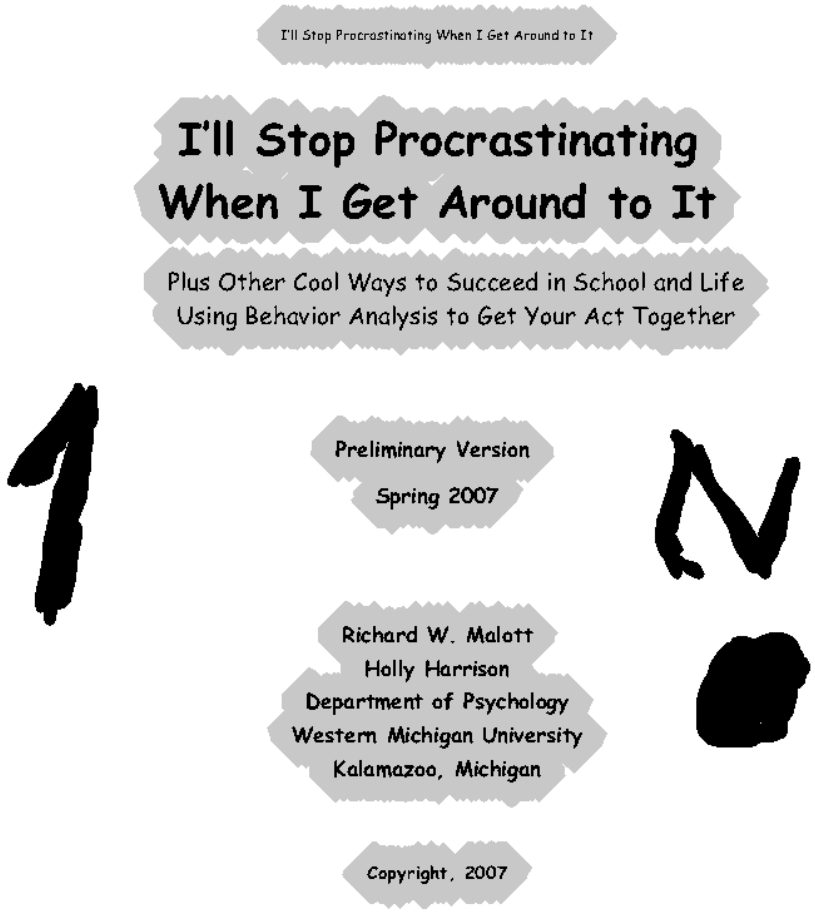
```
for (int k = 0; k < n; k++) {
	for (int i = 1; i < height - 1; i++) {
		for (int j = 1; j < width - 1; j++) {
			if (src.at<uchar>(i, j) != 0) {
				dst.at<uchar>(i, j) = src.at<uchar>(i,j);
				dst.at<uchar>(i - 1, j) = src.at<uchar>(i, j);
				dst.at<uchar>(i, j - 1) = src.at<uchar>(i, j);
				dst.at<uchar>(i + 1, j) = src.at<uchar>(i, j);
				dst.at<uchar>(i, j + 1) = src.at<uchar>(i, j);
			}
		}
	}

}
```

2.5. **Finding the text areas**

This is the final step of the algorithm and it stores in a result image the initial image with the text areas overlapped. It compares the result of the dilation with the initial picture and on the initial picture it highlights the areas covered by the dilation.

I have also used a helper function that converts a float Mat image to an uchar Mat image in order to process it correctly after the labeling was done.

The final result of the text areas detection algorithm on the above image:



3. Future improvements

As future improvements, the algorithm could be improved in terms of memory managements, because it uses many new matrices instead of reusing old ones. Also, the algorithm could be improved by also sharpening the text in order to improve its readability, by applying some filters on the initial tresholded image.

4. References

[1] Enhancing Readability of Scanned Picture Books - http://www.cs.umd.edu/hcil/trs/2008-09/2008-09.pdf

[2] Image Processing Laboratory Works - http://users.utcluj.ro/~igiosan/teaching_ip.html