# A light-weight text image processing method for handheld embedded cameras

Maurizio Pilu, Stephen Pollard
Hardcopy Technlogy Laboratory
HP Laboratories Bristol
HPL-2002-82
April 12th , 2002*

E-mail: maurizio_pilu@hp.com,, stephen_pollard@hp.com

document imaging, image processing, PDAs, cameras

In the past few years we have witnessed the migration of cheap imaging devices into portable and mobile computing appliances such as PDAs and mobile phones. As these devices become ever more powerful in terms of processor speed and memory, new exciting applications and uses are being developed. A particularly useful one is the casual capture of text images for faxing, note taking, OCR, etcetera. This paper describes the image processing pipeline used to enhance images of text captured by a hand-held low-resolution camera, and a fast text extraction method. The main advantages of the approach are its inherent lightweight structure, speed and relative robustness under poor lighting and focus conditions. The computational efficiency (and a careful implementation) of the approach has allowed its deployment in an interactive-time text capture and foreign-text translation demo on a PDA with a VGA camera attachment.

## 1. Introduction

In the past few years cheap imaging devices are making their way into portable and mobile computing appliances such PDAs and mobile phones, which are in turn becoming more and more powerful in terms of processor speed and memory. Amongst the many exciting new uses that these devices stimulate are interesting applications that live in the intersection between imaging and mobile computing [2][3][7]. One of those is the *casual capture* of text images for, e.g., faxing, note taking, OCR, etcetera.

We have developed a working concept demo of a PDA with a VGA camera attachment that is able capture a text image, process it, detect and locate text, OCR manually selected text lines, and translate the text into another language, all running on a Jornada 568 PDA in just a few seconds[1].

There were several challenges that needed to be addressed. First, embedded or PDA cameras are typically low-resolution (e.g. 640x480) and will continue to be so due to cost pressure. Second, we are in situations where the lighting conditions are poor and can vary wildly across the visual field and soft shadows are also very common (see Figure 5). Hence, we need to make sure that this uneven illumination is dealt with in order to produce a binary text image that is free from large thresholding artifacts and yet is not affected by local edge interaction between adjacent text characters. Third, although mobile computing architectures are becoming ever more powerful, there are still limitations that seriously affect performance, such as a small working RAM, no mass storage, no floating-point coprocessor and limited caching.

A good trade-off between speed an accuracy is therefore necessary for these sort of applications. For instance, in a recent conference paper [3] discussing techniques for a text capture and translation application, most of the image processing was actually done on a powerful server wirelessly connected to the device, because their text processing and extraction method alone took several tens of seconds on a 133 MHz PDA.

This paper describes the image processing pipeline used to process, enhance and binarize raw text images captured by a hand-held camera and an extremely fast text extraction method. The pipeline as described here has been implemented on an actual PDA with camera attachment and has an inherent lightweight structure and relative robustness to poor lighting and focus conditions. However, the deployment of this method is not limited only to PDAs.

[1] Most of which taken by the optical character recognition.

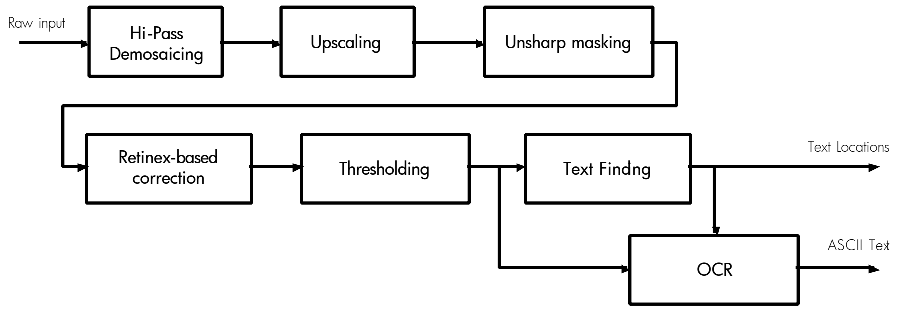

Figure 1. Overview of the text image processing pipeline presented in this paper.

The approach is overviewed in Figure 1. Starting from a raw sensor image, we use a new hi-pass color demosaicing method to reconstruct color planes from raw sensor pixels. Next, we perform simple restoration (to attenuate the degradation due to sub-sampling) by upscaling using a fast implementation of bicubic interpolation, producing an image twice the original resolution. To the upscaled image we apply an unsharp mask filter that enhances the high-frequency edges of the text characters. Next we apply a non-linear retinex-based filter that removes low spatial frequency lighting variations in the image and returns an enhanced image with uniform background, to which thresholding is applied to finally produce a binary text image. Finally we use a variation of [7] for compensating for skew are for locating text in the image[2].

The rest of the paper is structured as follows. Section 2 describes the demosaicing method, Section 3 image upscaling and sharpening, with an eye on some important efficiency issues, Section 4 describes the fast text finding method and finally Section 5 provides some examples and discusses the benefits and limitations of the approach presented.

## 2. HiPass Color Demosaicing

Color cameras in mobile devices typically use a single VGA resolution CCD array employing three or more color filters with each pixel capturing a single color. The most popular arrangement is the Bayer array which has alternating rows of Green-Red and Blue-Green filtered pixels. Hence 50% of the pixels are green, approximating luminance, and 25% each are red and blue reflecting the lower spatial sensitivity to chrominance of the human visual system.

[2] Note that the OCR method used will be not be discussed in this paper. In our prototype running on a PDA we have ported an existing OCR package developed at Hewlett-Packard Laboratories.

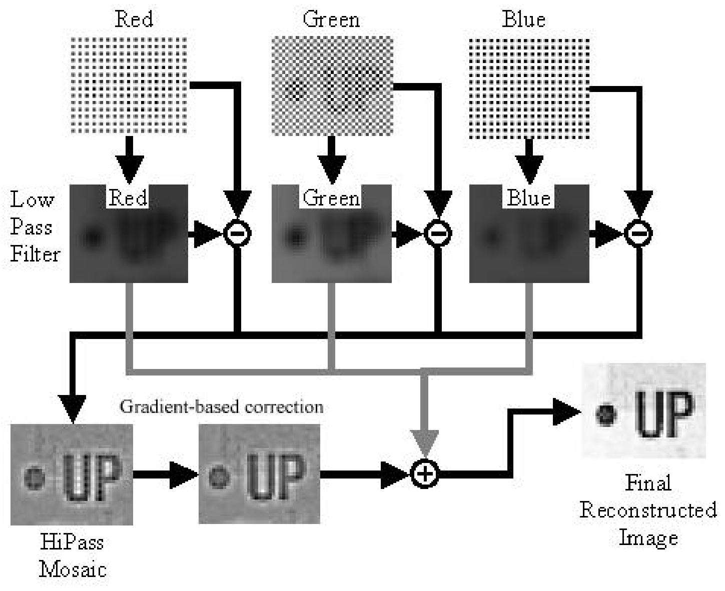

Figure 2. HiPass color demosaicing method.

Full color plane reconstruction, or demosaicing, requires the interpolation of each color plane (see [1] for a review of interpolation schemes). In order to maintain full text resolution we have developed the HiPass demosaicing method [8] outlined in Figure 2. The method attempts to decompose the mosaic into a high frequency monochrome and low frequency RGB data. These can then be combined to approximate the true full color image. While some chrominance edges are attenuated the method provides full resolution monochrome text and avoids most color aliasing artifacts for photographic images.

For each red, green or blue pixel in the raw mosaic (shown expanded into red, green and blue color planes in Figure 2), the corresponding pixel in the HiPass mosaic is constructed by subtracting from it a corresponding low pass red green or blue pixel constructed from the appropriate color plane of the mosaic. This is then corrected to remove zippering artifacts using a variant of the standard gradient-based interpolation schemes outlined in [1]. Each pixel of the HiPass mosaic that is derived from a red or blue pixel is corrected using a 1D quadratic interpolation using the row of 5 neighboring pixels in the vertical or horizontal direction depending which has the lower intensity gradient. The interpolation is of the form $C_0 = (2I_0 - I_{-2} - I_2 + 4I_{-1} + 4I_1)/8$, where I and C are the intensities of the HiPass mosaic and its corrected version, respectively. The corrected HiPass mosaic is then added to each of the low pass color images to generate the fully reconstructed image.

Improved computational effectiveness is achieved by low pass filtering using block averaging over a 7x7 image neighbourhood. By maintaining a row wide table of intermediate sums this can be achieved using 2 adds, 2 subtracts, a multiply and a shift per pixel independent of the block averaging parameter.

## 3. Text image upscaling and sharpening

Although the input gray level image could be left at native resolution, experiments and theory show that with a heavily under-sampled image interpolation kernels such as bicubic interpolation act as true restoration processes [9]. To this end we upscale the image by a factor of two using an efficient separable *bicubic convolution* which requires very few add-multiply operations per pixel.

We use the Keys' interpolating function [4] which approximates the *sinc* function with piecewise third order polynomials:

$$u_a(x) = \begin{cases} (a+2)|x|^3 - (a+3)|x|^2 + 1, & 0 \le |x| < 1 \\ a|x|^3 - 5a|x|^2 + 8a|x| - 4a, & 1 \le |x| < 2 \\ 0 & 2 \le |\mathrm{x}| \end{cases}$$

The value of *a* changes the shape of the kernel and there has been much literature as to how this affects the output. In our implementation we left it as a parameter but $a = -1$ has given the best results in our opinion.
The kernel is separable, that is it can be applied first along the rows to produce an intermediate upscaled image and then along the columns of the intermediate image.

Given that we are upscaling by a factor of two we can easily pre-compute a discrete $4x1$ convolution vector $\mathbf{u}_a$ to be used for each pass. The kernel is different according to whether we are interpolating a pixel in an even or odd (output) position and is given by:

$$\mathbf{u}_a^{even} = [u_a(-1.5) \quad u_a(-0.5) \quad u_a(0.5) \quad u_a(1.5)]$$
$$\mathbf{u}_a^{odd} = [u_a(-1.25) \quad u_a(-0.25) \quad u_a(0.75) \quad u_a(1.75)]$$

After upscaling, we use the well-known unsharp mask filter to boost the image high-frequencies and thereby produce a crispier text image. Unsharp masking works by subtracting from the input image a smoothed version of it, and adding the result weighted by a factor *k* back to the input. In our implementation, extreme care has been taken in optimizing both memory usage and performance. In particular, we create the smoothed image from the input via a very efficient block averaging (see end of Section 2). The block width $W_S$ is configurable but good results for the VGA (640x480) images used were obtained with $W_S=5$. Let us now call by $I_{upscaled}(i,j)$ a pixel of the input image and by $I_{smoothed}^{(W_S)}(k,l)$ a pixel of the block-smoothed image. The formulation of the unsharp mask filter we used is modified from the standard one and substitutes *k* with $k' = \frac{k}{1+k}$, which reduces the unsharp mask equation to:

$$I_{unsharp}(i,j) = \frac{I_{upscaled}(i,j) + k' I_{smoothed}^{(W_S)}(i,j)}{1-k'}$$

where *k'* is a constant factor that controls the added amount of the sharpness and usually ranges between 0.2 and 0.7. In our implementation we have used a fixed $k' = 0.5$.

## 4. Retinex-based intensity correction and thresholding

In order to compensate for illumination variation an safely binarize the image we use a method based on the *retinex* theory [5], a widely studied technique for color and illumination constancy.

The retinex approach is based on a simple model of image formation, $I(x,y) = R(x,y) \cdot L(x,y)$, where $I(x,y)$ is the measured image intensity, $R(x,y)$ is the surface reflectivity and $L(x,y)$ is the illuminant. There are several techniques aimed at using this model in order to recover the reflectivity of the surface $R(x,y)$. The most common ones use the assumption that (the basically unknown) $L(x,y)$ can be approximated by the low-frequency component of the measured image [6]. We have followed this approach but we tailored it to a situation where the desired surface reflectivity is binary-valued. The process is illustrated in Figure 3. First we block-smooth the input image with a large kernel (A). Next we compute the ratio between the input image and the smoothed image (B); this amounts to solving for $R(x,y)$ in $I(x,y) = R(x,y) \cdot L(x,y)$. The ratio is scaled to an 8-bit range to give the illumination corrected image (C). Finally, a threshold is applied to transform thi image into binary (D).
From Figure 3 it is easy to see that since we expect black-on-white text, the ratio should be at most 1 at uniform background pixels and it is relatively small with text characters.
Let us now describe the actual formulation of the filter and its implementation. Let us call by $I_{unsharp}(i,j)$ a pixel of the image as output by the unsharp process of Section 3, and by $I_{smoothed}^{(W_h)}(k,l)$ a pixel of the block-smoothed image. Differently from the smoothed image used for the unsharp masking of Section 3, here the smoothed image should represent the illumination component of the image and as a consequence the smoothing kernel must be rather large; in our implementation we used $W_s = 31$ for 640x480 input images. We used block smoothing again for efficiency reasons, and necessarily so with such a large kernel. We then have:

$$I_{retinex}(i,j) = 255 \cdot \frac{I_{unsharp}(i,j)}{I_{smoothed}^{(W_h)}(i,j)}$$

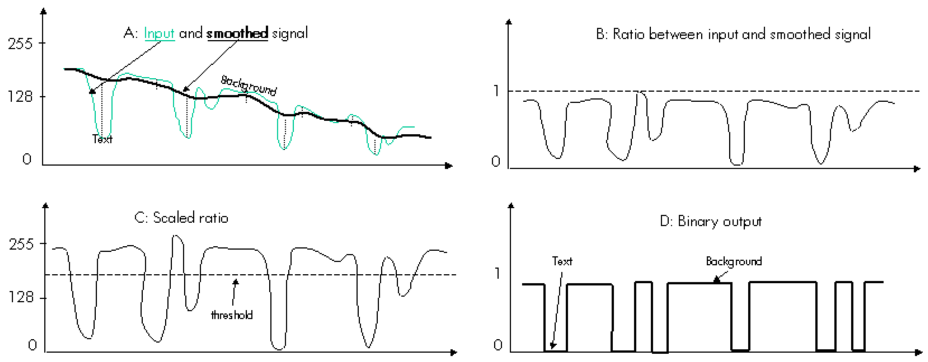

Figure 3. Illustration of the steps involved in the retinex-based filtering to perform illumination correction and finally thresholding.

where $I_{retinex}(i, j)$ is the illumination corrected image shown in Figure 3-C. The results of this illumination correction stage can be also appreciated in the change between Figure 5-C and Figure 5-D.

The final stage in order to produce a binary output image is thresholding. Given the even background of the $I_{retinex}$ thus obtained, this image is amenable to global thresholding. A single threshold $t$ = 200 yielded satisfactory results in a wide range of cases. However the thresholding stage could benefit from a slightly more sophisticated approach that would analyze the histogram of $I_{retinex}$ more locally. For instance, experiments that we carried out with dual-peak thresholding applied to $I_{retinex}$ have shown thinner and more accurate binary characters.

Finally, note that other formulations of the retinex principle can be seen in the literature. A particularly common one is that of using as retinex output the difference of logarithms of the original and smoothed image, which in our case would be $I'_{retinex}(i, j) = \log\left(I_{unsharp}(i, j)\right) - \log\left(I^{(W_h)}_{smoothed}(i, j)\right)$. This formulation is obviously equivalent to the one employing ratio if we threshold $I'_{retinex}$ with $t' = \log(t)$.

## 5. Text finding and localization

As part of the text processing pipeline we have also designed and implemented a lightweight text finding algorithm based on a cut-down version of [7] that analyses a binary image and outputs regions where there is likely to be some text, which can be either presented to the user for selection or automatically used for OCR. It is outside of the scope of this paper to describe this text finding algorithm in detail and we shall only briefly overview it and focus on the relevant modifications.

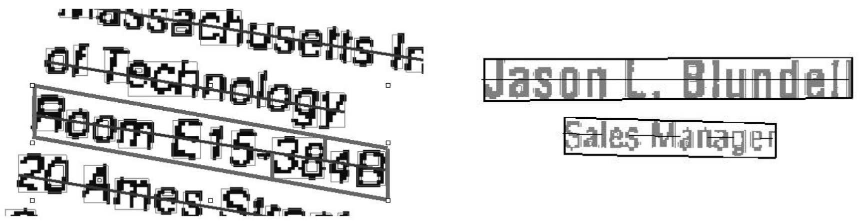

Figure 4. Determination of the bounding box for skewed text (left) and the occasional problems with determining the skew angle merely from the bounding parallelogram (right).

The algorithm starts from the binary image produced by the stage described in Section 4 and assumes that the text lines are within a 20$^{o}$ range from the horizontal (no assumptions of this kind were made in [7]). A fast connected component method labels isolated binary blobs. Blobs that are too large or too small are removed from further consideration. Then the blobs are sorted according to their abscissas to speed up the search. At this stage we start the text-line grouping using loose perceptual similarity, continuation and proximity criteria as detailed in [7]. However, differently from [7] we do not build a probabilistic network of blob associations but using the assumption that the lines must be more or less horizontal we perform direct search and growing of groups as we go along. Starting from the first (leftmost, since we sorted them by abscissas) usable blob, we search for the closest blob to its right and carry on from it until the blobs becomes either too dissimilar in size, are too separated or deviate from the illusory line that the group is forming, all in the same spirit of [7] albeit with a slightly different implementation of the perceptual criteria. We carry on the search for other blobs until all the blobs have been explored and all linear groups are formed. A further higher level pass makes sure that groups are not duplicated and also merges collinear broken groups.

The method is able to group text lines under a considerable amount of rotation, although the performance starts degrading after about 20$^{o}$ skew; the reason for this is mainly due to the way the closest blob is found, that is not by searching for the closest blob contour points as in [7] but by the vastly faster use of the blobs' rectangular bounding boxes (see the blob bounding boxes in Figure 4-left).
Once the groups are formed, we determine the bounding boxes for each group. In Figure 4-left we show the process in the case of skewed text, where a parallelogram is found with vertical lateral sides and with upper and lower sides parallel to the best-fit line to the group and distant enough from it to contain all the connected components' bounding boxes. If any of the boxes are needed for OCR or display, the best-fit line to the connected components or the bounding box itself can be used to de-skew the text portion. Both methods might not always be reliable for short text portion, as shown by the lower bounding box in Figure 4-right.

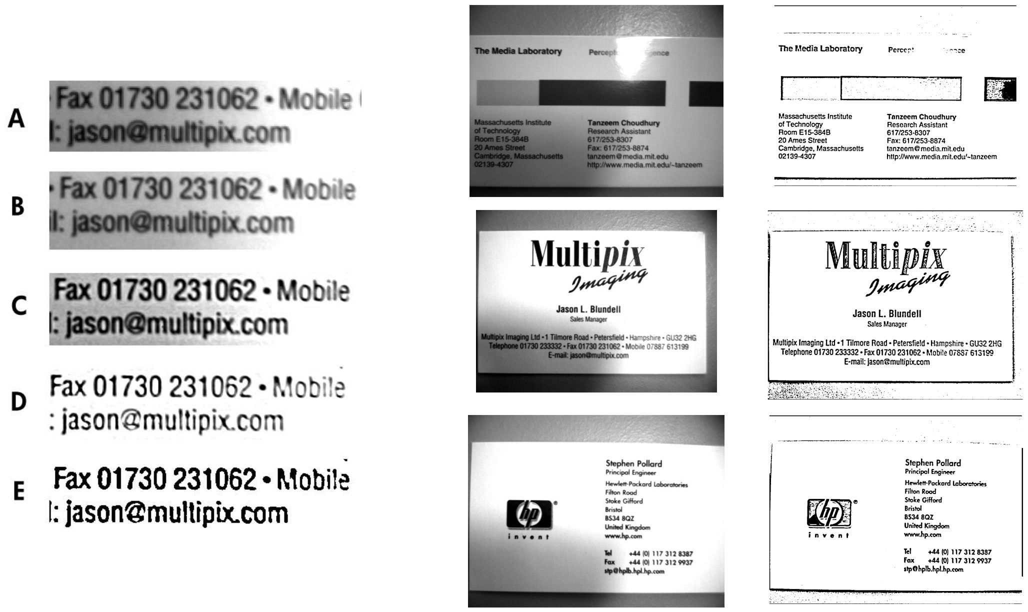

Figure 5. Left: Blowup of image portions showing all the stages that take the raw input image to the binary output. A: Demosaicked image; B: Upscaled image; C: After unsharp mask; D: After retinex-based filter (before thresholding); E: Thresholded binary output. Right: Three full examples showing the input images and the binary result from Section 4.

## 6. Results and Discussion

In this section we present and discuss a series of results of the stages presented in the previous sections.

Figure 5-left shows the results of the three main stages of Section 2, 3 and 4, that is demosaicing (A), upscaling (B), unsharp masking (C) and retinex-based enhancement (D) along with the thresholded output in (E). In Figure 5-right and there are shown four examples where the input images and resulting binary images are shown side by side. It can be noticed that although the method performs well with text of ordinary size it may mistake parts of oversized fonts as background and cannot deal (as is) with reversed text, although improvement in this respect are possible. However, the stages up to binarization proposed here perform satisfactorily in most common situations and cope well with dramatic lighting conditions and shadows. But the primary advantages of the approach are that it is extremely lightweight and memory lean and thus suitable for interactive time processing on low-power architectures. As an example, the whole pipeline runs in a fraction of a second on a current HP Jornada 568 PDA and uses little over one megabyte of working memory.

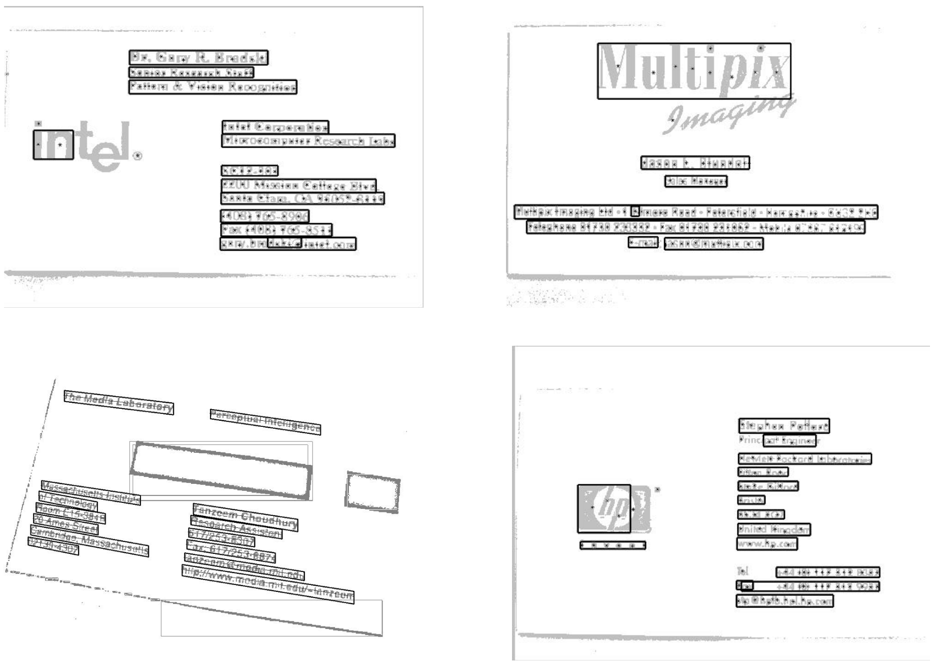

Figure 6. Four examples of text finding using the algorithm described. Note the heavily skewed lower-left example.

Figure 6 shows some results of the text finding stage of Section 5 applied to the some of the binary images of Figure 5.

The results are generally satisfactory in a wide range of cases but some limitations of the method are worth noting. First above all, the method cannot cope with large amount of rotation. It might deal with some *perspective* skew but it can break likewise (but see methods addressing this problem, such as [2][7]). The method is inherently scale independent (see the “Multipix” image in Figure 6-top-right ), but it is strongly based on blob similarity and might not group lines with widely different font sizes. Moreover, being based on the *organization* of text and not on local properties, it cannot group isolated characters, as they are un-organized. Finally, due to that same reason and the fact that we use pre-segmented blobs without intensity information, the method might group blobs that verify organization criteria such as proximity and similarity even if they are not even remotely text-like. In the future we should investigate the abundant literature for a computationally efficient method to detect non-text situations (e.g. using global statistical methods) or assign a probability of being text to each blob in order to avoid grouping so blindly. Having said this, the method has served us well due to its extremely good time-performance trade-off and negligible memory use. It runs in just a few hundreds milliseconds on our prototype running on a current HP Jornada 568 PDA.

## References

[1] Adams, J. E., “Design of practical color filter array interpolation algorithms for digital cameras”, *Proc. SPIE, Real Time Imaging II*, Vol. 3028, 1997.

[2] P. Clark and M. Mirmehdi. “Estimating the orientation and recovery of text planes in a single image”, In *Proceedings of the 12th British Machine Vision Conference*, pp. 421-430. BMVA Press, September 2001.

[3] Haritaoglu, I., “Scene Text Extraction and Translation for Handheld Devices”, *IEEE Computer Vision and Pattern Recognition Conference*, Kauai, HI, December 2001.

[4] Keys, R., "Cubic convolution interpolation for digital image processing”, *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-29, pp. 1153-1160, December 1981.

[5] Land, E. L., "The Retinex Theory of Color Vision," *Scientific American*, Vol. 237, No. 6, pp. 108-128, December 1977.

[6] Land, E. L. , “An alternative technique for the computation of the designator in the retinex theory of computer vision”, *Proc. Nat. Acad. Sc*i., Vol. 83, pp. 3078-3080, 1986.

[7] Pilu, M., “Extraction of illusory linear clues in perspectively skewed documents”, *IEEE Computer Vision and Pattern Recognition Conference*, Kauai, HI, December 2001.

[8] Pollard, S. B., “Image Mosaic Data Reconstruction”, *U.S. Patent Application 09/906,786*, 2001.

[9] Wolberg, G., *Digital Image Warping*, IEEE Computer Society Press, Los Alamitos, CA, 1990.