Automatic Red-Eye Removal based on Sclera and Skin Tone Detection

Flavien Volken, Johann Terrier, Patrick Vandewalle; School of Computer and Communication Sciences, Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Abstract

It is well-known that taking portrait photographs with a built in camera may create a red-eye effect. This effect is caused by the light entering the subject's eye through the pupil and reflecting from the retina back to the sensor. These red eyes are probably one of the most important types of artifacts in portrait pictures. Many different techniques exist for removing these artifacts digitally after image capture. In most of the existing software tools, the user has to select the zone in which the red eye is located. The aim of our method is to automatically detect and correct the red eyes. Our algorithm detects the eye itself by finding the appropriate colors and shapes without input from the user. We use the basic knowledge that an eye is characterized by its shape and the white color of the sclera. Combining this intuitive approach with the detection of "skin" around the eye, we obtain a higher success rate than most of the tools we tested. Moreover, our algorithm works for any type of skin tone. The main goal of this algorithm is to accurately remove red eyes from a picture, while avoiding false positives completely, which is the biggest problem of camera integrated algorithms or distributed software tools. At the same time, we want to keep the false negative rate as low as possible. We implemented this algorithm in a web-based application to allow people to correct their images online.

Introduction

The red-eye effect, which is one of the most common artifacts in amateur photographs, is very disturbing. Most amateurs and many professionals currently use digital still cameras. This evolution has made the post-processing of the image easier. In particular, many digital camera manufacturers provide a red eye removal method in their integrated software. We have to distinguish between manual processing methods, where the user has to select the red eye in the picture, and automatic processing [1, 2, 3, 4, 5, 6, 7], where the eye is detected automatically. The algorithms used to perform this task are often hidden in a "black box," but in most cases this processing is based on an autocorrelation of the image (edge detection) or on face detection techniques [8] (widely used in biometry) for the automatic removal. Unfortunately, the result is not always perfect or not the one expected.

We use a very intuitive approach to red-eye removal, which is not based on face recognition. First of all, we detect red zones in the image. Then we calculate the roundness of each zone and the amount of white around this red zone (detection of the sclera). We also calculate the amount of "skin" around this red zone. Once these parameters are computed, we can estimate the probability for the considered red zone to be part of an eye. Finally, we correct red eyes using a Gaussian filter.

This paper first introduces the state of the art in red-eye removal, and then our approach is introduced. Next, we describe further specificities of our web-based application. Finally we present results and a comparison with the most widespread tools.

State of the art

Most of the currently available tools for red-eye removal are manual. This is also the case for the software included with most digital cameras. For instance, the red-eye removal algorithm in the *Picture Project* application (*Nikon*) is fully manual and not very powerful. The zone of the image that is selected by the user is "automatically" blackened and blurred without any detection of where in that zone the eye is located. A more powerful method is included in *Zoom Browser* (*Canon*): the corrected region looks nicer and the user can choose between automatic and manual mode. Finally, some image processing applications, like *Adobe Photoshop*, *Adobe Paintshop*, *Corel Draw* or *Photopaint* remove the red eyes very nicely, but the detection is done entirely manually. *Adobe Photoshop Elements 2.0* uses an interesting red-eye brush, where the colorization depends on the luminance of the original pixels. Unfortunately it does not detect the eyes automatically.

There are only few automatic red-eye removal tools commercially available. One of the most popular applications is *Stoik RedEye Autofix*, which uses a proprietary algorithm to detect and remove the red eyes from a picture. HP has developed an online tool, RedBot, to remove red eyes from an arbitrary picture [3].

Other approaches for red-eye removal have been proposed recently by Gasparini and Schettini [4, 5]. Their algorithm starts by detecting a face using color information (skin detection). They also use the edges on the intensity channel (detection of the nose or the eyes for instance) to ensure that the detected region is an eye. Edges may also be detected using a convolution operation [1]. Then the algorithm detects the red eye itself using the roundness and the amount of red in a specific region. Our approach starts from the red zones because the face detection is a very difficult task, which additionally increases the sensitivity of the algorithm (if only part of a face was visible, it would not be detected). A too high false positive rate (a zone is wrongly detected as an eye) is the major problem in most of the automatic correction tools available.

Another approach consists in using machine learning techniques by training a classifier with a large collection of images in order to make the detection automatic [6]. For each image, the algorithm improves its model by extracting a feature vector (positive if an image contains a red eye, or negative otherwise). The problem with this kind of technique is that images are generally hard to collect to create a large database. Moreover the machine learning appears again as a "black box". An algorithm that combines the machine learning technique with edge detection and face recognition was presented by Ioffe [7].

Our approach

Our algorithm works as follows: The RGB image is transformed into Lab color space [9] to obtain a more perceptual rep-

resentation for color thresholding [10]. First, bright red pixels in the image are detected using a thresholding operation. This results in a binary mask with all the red pixels. We delete objects smaller than x pixels in order to reduce the computational time. These mask regions are closed by performing a dilation followed by an erosion operation. We thus obtain N smoothened regions α_i $(0 < i \leq N)$.

For each region α_i we calculate its roundness R ($R = 1$ for a perfect circle):

$$R = \frac{4\pi \times A}{P^2}, \tag{1}$$

with A the area and P the perimeter of the region. If the roundness R is smaller than a threshold β, we remove the region from the list of candidate red-eye regions (we assume that the red eye cannot be too far from a perfect circle). We calculate the number W of white pixels in a crop around the red zone and the number S of skin colored pixels in a larger crop around the red zone using again a thresholding operation in Lab color space. Using R, W and S we can determine the probability that the zone is an eye:

$$P(eye) = P(R)P(W)P(S), \tag{2}$$

with R the roundness of the region and W and S the amounts of white and "skin".

We obtain a probability depending on the roundness of the surface and the number of white pixels and "skin" pixels around the red zone. If $P(eye) > \delta$ (δ is a threshold) and $P(W) > \delta_2$ and $P(S) > \delta_3$ (δ_2 and δ_3 are used to avoid the red spot and albino rat problems described in the results section), the region is considered as a red eye. We blacken this region and blur it using a Gaussian filter. The parameters β, δ, δ_2 and δ_3 are chosen experimentally in order to make the false positive and negative rates as small as possible. We have trained an algorithm on a quite large number of sclera and face images in order to statistically fix the region of the Lab color space to be considered.

We will now describe the different steps of our algorithm in more detail. They are illustrated in Figure 1.

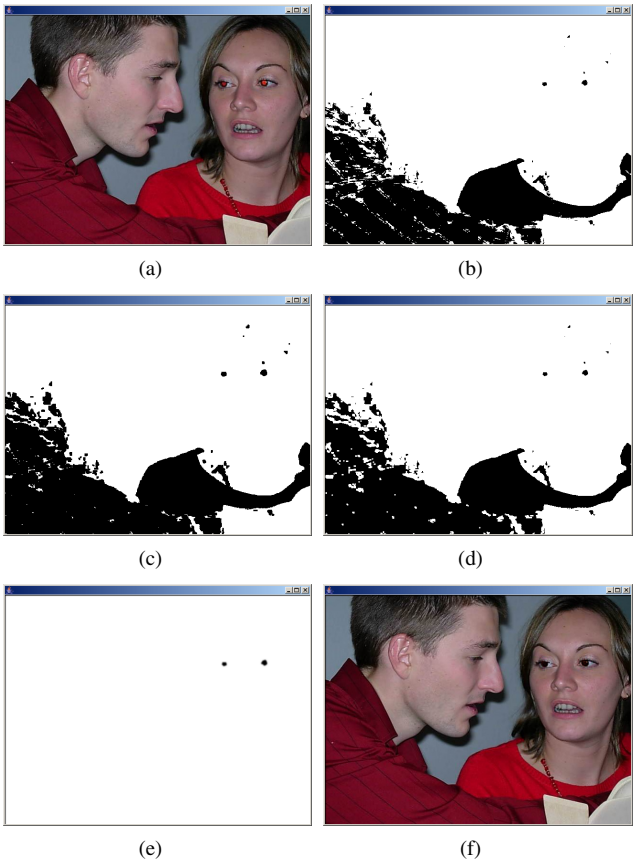
Detection of red, white and "skin" pixels

The RGB image is transformed into Lab color space in order to easily distinguish the red zones using the L, a and b channels. Using a thresholding operation we obtain a binary mask with the red zones. At the same time we check whether each pixel has white or "skin" color and we create two other masks. This means that we go through the whole image only once to obtain three masks.

For the detection of skin tones, we apply a thresholding operation on the L, a and b components. From an analysis of many different skin tone patches, we noticed that the union of all the Lab regions corresponding to the different types of skin tones (Caucasian, dark, etc.) forms a single region in the Lab color space. All skin tones can be detected using a single thresholding operation in Lab space. A pixel is detected as a skin pixel if its probability is larger than a threshold: $P(Skin) > \delta_3$. Figure 2 shows that our algorithm also works for dark skin tones.

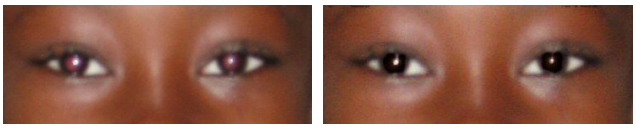
Closing the zones

As the mask is not always perfect and as the original image can contain isolated red pixels, we have to close each zone. The closing operation is a combination of dilation and erosion, two morphological operators. As we can see in Figure 3 the dilation copies a black pixel to its direct neighbors. In contrast, erosion whitens all pixels having at least one neighbor that is not black.



(a) (b) (c) (d) (e) (f)

Figure 1. *Result using our red-eye removal algorithm. (a) Original image. (b) Binary mask with red zones. (c) Binary mask after a dilation operation. (d) Binary mask after an erosion operation. (e) Filtered Binary mask containing only eyes. (f) Corrected image.*



(a) (b)

Figure 2. *Result using our red-eye removal algorithm on a non-Caucasian person. (a) Original image. (b) Corrected image.*

Division in zones

After the closing operation, we can analyze each zone and fill in a table with information related to the zones. We go through the mask line by line and as soon as the algorithm encounters a red pixel it is colored with color i. All the neighbors of the detected pixel are also iteratively colored with the same color if they are part of the same zone (there exists a path between the first encountered pixel and the considered one consisting only of black pixels). When another black pixel is found, i is incremented, allowing us to obtain a mask having one distinct color for each zone. This means that we can support up to $2^{24} - 2$ different zones (from #000001 to #FFFFFE). The result of the separation of the zones is illustrated in Figure 4. As the image is scanned, it is quite easy to count the number of pixels of each zone, to calculate its perimeter (pixels with a neighbor of different color) and to memorize the minimum and maximum coordinates along x and y direction for each zone.

(a) (b) (c) (d)

Figure 3. *Closing a zone. (a) Original image. (b) Red mask. (c) Dilated red mask. (d) Eroded red mask.*

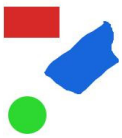


Figure 4. *Colored mask obtained after the separation operation.*

Analysis of each zone

By default each red zone is supposed to be an eye. This means that all the tests performed on these zones aim at rejecting regions that do not satisfy the criteria.

Area and Perimeter

As it was discussed above, the area and the perimeter are obtained after the delimitation of the regions. We consider that a zone formed by less than x pixels (we used $x = 10$ in our application) cannot be an eye or is certainly not visually disturbing. This permits us again to reduce the computational time.

Roundness

Since we have the area and the perimeter at disposal we can calculate the roundness using (1). A zone should have a roundness close to 1 to be an eye.

Amount of skin and white around each zone

We use two different rectangular crops around the red region in which we simply count the number of white and "skin" pixels. These calculations are performed on the Boolean table. We obtain a percentage of white and "skin" in the considered zone by dividing the number of pixels found by the total area of the crops. These percentages must exceed a threshold that is obtained experimentally. If one of the two amounts is below the threshold, the zone is not an eye. We calculate $P(W)$ and $P(S)$ as follows:

$$P(W) = \frac{f(A_r)}{A_w} \sum_i \sum_j white(i,j) \tag{3}$$

$$P(S) = \frac{1}{A_s} \sum_i \sum_j skin(i,j), \tag{4}$$

where A_r is the area of the red zone, A_w and A_s are the areas for the white and the "skin" pixels respectively. A_s is four times bigger than A_w as we double the dimensions along both axes. The function $f(A_r)$ depends on the area of the red zone. This function gives us a coefficient that is proportionally larger when the region is small because in this case the sclera is often hard to detect. When the subject stands in the background, the sclera is not really white. This implies that only few pixels will be detected. This coefficient helps us to reduce the false negative rate. This part of the algorithm removes most of the false positives.

Correction of the original image

After all these steps we know which zones represent red eyes. Zones that are not detected as eyes are removed from the mask. Then we simply apply a Gaussian filter on the black and white mask and replace pixels in the original image by the ones of the filtered mask. We compute the convolution of an NxN Gaussian kernel with the mask to obtain smooth grey levels.

Improvements

We added and tested improvements regarding the roundness, the crop used to detect the sclera and the way the eye is corrected.

The roundness

Using (1) we obtain a good approximation of the roundness. However, if for example an eyelid covers part of the iris, it might be much lower (e.g., less than 0.75). Also, using our approximation the area and the perimeter of a straight line are equal, resulting in a roundness $R > 1$. Thus we add another parameter η that is computed as follows:

$$\gamma = \frac{\Delta x}{\Delta y} = \frac{max(x) - min(x)}{max(y) - min(y)} \tag{5}$$

$$\eta = max\left(\gamma, \frac{1}{\gamma}\right). \tag{6}$$

This formula gives us a parameter $\eta > 1$, that represents the distortion of the zone along the axes. To be an eye, η should be close to 1.

One last case has to be considered. A diagonal line has both a roundness $R > 1$ and an axial distortion $\eta = 1$. Therefore, we add a final parameter calculated as follows:

$$\zeta = \frac{A}{\Delta x \Delta y}. \tag{7}$$

If ζ is close to 1, there are many red pixels in the zone, while if ζ is close to 0, the zone is very thin (a straight line in the worst case).

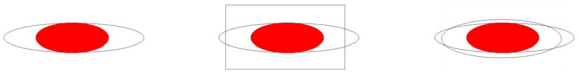
To summarize, to be considered as an eye the zone must verify:

$$\text{red eye} = \begin{cases} R \geq x_1 \\ \eta \leq x_2 \\ \zeta \gg 0, \end{cases} \tag{8}$$

with x_1 and x_2 experimentally determined.

Using a different crop

We examined different crop regions to detect the white around the eye. A good idea might be to use a dilated version of the original zone. For instance if the subject has a half-open eye the percentage of white in a rectangular region around the eye might be very low. In contrast, there is no problem to detect the skin, as the eyelid recovers most of the iris. Using a region that is a dilation of the red zone, we will calculate the amount of white in a region with the same shape. If the eye is half-open, the crop is wide and thin, just like the detected red eye. This would increase the probability to detect the zone as an eye because the threshold becomes easier to determine. From the illustration in Figure 5, we easily see that the proportion of white is more important using this second cropping approach. Results obtained with this crop are very similar to those using the rectangular crop for half-open eyes, because the white pixels are detected in both



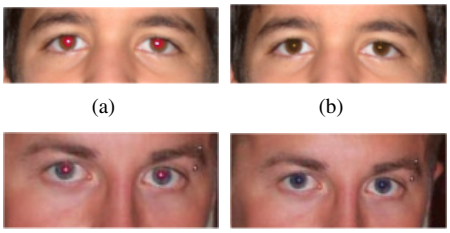
(a) (b) (c)

Figure 5. *Using different crops for the detection of white and "skin" pixels. (a) Original shape. (b) Rectangular crop. (c) Dilated original shape.*

cases. However the results are worse when the eye is open, because not all the white pixels are detected. Moreover, this kind of crop increases the computational time. Therefore, our application uses a rectangular crop.

Natural correction of the eye

Unfortunately, in most cases it is physically impossible to know the original color of the eye with the red shade. It would be possible if part of the eye was not red but that is rarely the case. We tried to colorize the eye by modifying color channels of the mask before it goes through the filter. However, the result is far from perfect because the eye appears to be very unnatural. Figure 6 shows examples where the correction is acceptable. Of course, a grey eye is not very realistic either, but it is visually much less disturbing.



(a) (b)

(c) (d)

Figure 6. *Result using our red-eye removal algorithm for the detection combined with a manual colorization. The corrections shown here are acceptable, but in many cases less natural results are obtained. (a) First input image. (b) Resulting image. (c) Second input image. (d) Resulting image.*

Our application

We wanted to implement a web-based interface for our tool, in order to allow people to correct their images online without installing any software. In order to account for the slowness of Java applications, we reduced the complexity of our algorithm as much as possible. First we minimized the number of times the image has to be scanned. Then we optimized our code using the Shark tool to detect which methods put the most load on the CPU. Using this approach we reduced the processing time by a factor of 4.

As part of the complexity reduction, we fill in a 3-dimensional Boolean table while we transform the image into the Lab color space. For every coordinate (x, y) we set the value to true when a red pixel is detected. This means that, once this work is performed, we have two ways to check if a pixel is red or not (the mask and the Boolean table). At the same time, we check whether each pixel has white or "skin" color and we fill the third dimension of the table for each color for all the pixels. This means that we go through the whole image only once to obtain a Boolean table in which we can check if a pixel has red, white or "skin" color. Later calculations are faster using a Boolean table (1 bit) than a pixel ($3 * 8$ bits). The closing operation is also performed on the Boolean table. Screen shots of the application are shown in Figure 1 and Figure 7. The application is available online at http://ivrgwww.epfl.ch/software/.

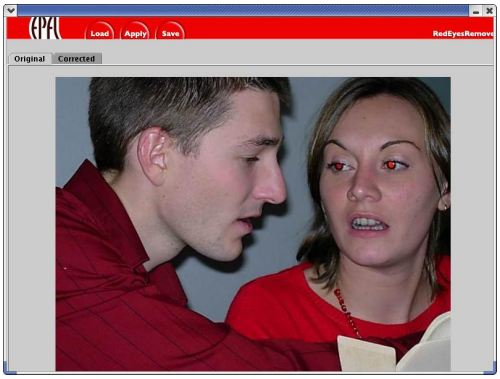
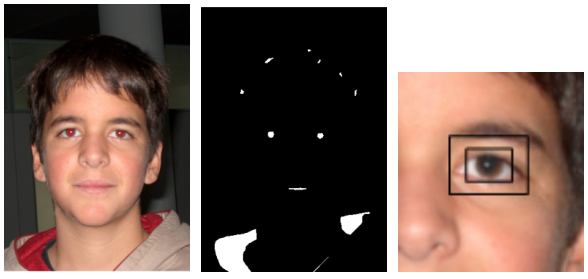


Figure 7. *Java application.*

Results

Our approach was tested on various images using a rectangular crop. The red eyes are easily detected on images where the subjects are not too far from the camera. An example is shown in Figure 8. Moreover, the false positive rate is very low, and for instance albino rats (no skin color) and red spots (no sclera) are not corrected. Our application is not fooled by images similar to those in Figure 9. Our algorithm also has no problem with images in which the red color is predominant.



(a) (b) (c)

Figure 8. *Result using our red-eye removal algorithm. (a) Original image. (b) Binary mask after removing small regions and filling holes. (c) Resulting image after red-eye removal. Rectangles in which the numbers of white and skin pixels are counted are indicated.*

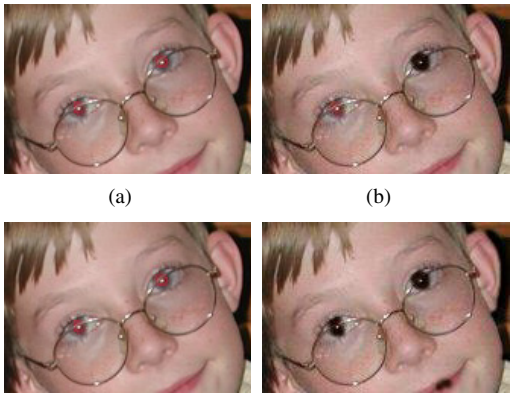
(a) (b)

Figure 9. *(a) Image able to fool the Canon software. (b) Image able to fool the Stoik software.*

Our algorithm successfully corrects red eyes for both Caucasian and non-Caucasian people. As part of our study, we also tested our algorithm on pictures of persons with glasses (see Figure 10). In this image, the frame of the glasses does not divide

(a) (b)

Figure 10. *Result using our red-eye removal algorithm on person with glasses. (a) Original image.(b) Corrected image.*



(a) (b)

(c) (d)

Figure 11. *Result using our red-eye removal algorithm on a person with glasses. (a) Original image. (b) Corrected image. One of the eyes has not been detected because the red zone is split in two by the frame of the glasses. (c) Original image. (d) Corrected image with multiple dilation and erosion operations. Both eyes are now detected, but a part of the lips is also corrected.*

the eye into two parts, and the red eyes are well corrected. If the frame divides the eye, the results depend on the thickness of the dividing line. If the frame is thin enough, the closing operation will merge the 2 zones and the result will be positive. However, if the frame is too large, the eye will not be detected except if the two parts of the eye are detected separately as an eye. As illustrated in Figure 11, the result might be quite arbitrary.

To avoid this problem, we replaced the original closing operation by an extended one. We simply execute the dilation several times, before launching the erosion the same number of times. As a result, the probability for a region to be closed increases. Unfortunately, as we can see in Figure 11, the mouth is also corrected because of this closing operation (two regions, not detected as eyes, are transformed into one single region detected as an eye). That is the reason why this idea has not been implemented in the final application.

Comparison

The different thresholds in our algorithm have been selected manually based on tests on a large set of images. We compared our program with automatic methods available online, *Stoik RedEye Autofix*, *Canon Zoom Browser V5.1* and *Hewlett Packard Redbot*. The test was performed on 100 images of type I and 20 images of type II :

- type I: Images containing red eyes of different kinds (bright or dark, different eye shapes, different skin tones) to test the efficiency of the algorithm.
- type II: Images without an eye to measure the robustness and to try to fool the software.

Results of the different algorithms on 100 test images of type I and 20 images of type II. A false positive (FP) means that a zone is wrongly detected as an eye, and a false negative (FN) represents an eye that is not detected.

	100 type I		20 type II
Methods	FP	FN	FP
Stoik RedEye Autofix	22	70	6
Canon ZoomBrowser	14	60	4
Hewlet Packard Redbot	3	65	1
Our Application	9	52	3

As we see in this table, our algorithm performs better than the other algorithms regarding the false positive rate, except for HP's Redbot. This shows that our program is quite hard to fool with images like those shown in Figure 9. One could argue that a very low false positive rate is not that important as people usually do not try to remove red eyes on a picture without any eye. However, for an automatic algorithm, this is very important because we do not want other regions of the image to get wrongly detected (and corrected) as an eye. For example for the image used in Figure 1, the necklace and the clothes might be blackened.

The false negative rate of our approach is better than all other applications. This shows that all the tests we performed to ensure that a zone is really an eye do not affect our results. This is due to the training of our algorithm on a large set of images.

During this comparison we noticed that the errors made by *Stoik RedEye Autofix* and *Canon Zoom Browser V5.1* occurred on different images. This shows that the embedded algorithms are based on different parameters. *Stoik RedEye Autofix* does not correct the eye if it is too small whithin the image, for example when the person stands in the background. Finally, the red eye is not detected for non-Caucasian people. *Canon Zoom Browser V5.1* also does not correct red eyes for non-Caucasian people and does not detect small red eyes. Corrections are made with a high quality and the false positive rate is lower than the *Stoik RedEye Autofix*.

Albino rats (no skin color) and red spots (no sclera) are not corrected using our algorithm. In contrast, the automatic mode of *Canon ZoomBrowser* does detect the frog eye from Figure 9(a) as a human eye and corrects it. *Stoik RedEye Autofix* also has some problems with more difficult images, like for instance the picture shown in Figure 9(b). Using the Stoik tool, the red dots are wrongly detected as human eyes and corrected. Our algorithm is not fooled by any of these examples because no skin is detected around the red dots. From these experiments, we can deduce that the automatic algorithms from these programs are probably not based on human skin characteristics like our algorithm.

Hewlett Packard Redbot behaved quite differently from the two other tested applications. It is extremely robust and sometimes corrects quite difficult images. However, it does not correct seamingly simple portraits like the image shown in Figure 12. This shows that *Hewlett Packard Redbot* is probably based, as our application, on a large set of quite sensitive tests. The results using *Hewlett Packard Redbot* are very good regarding the false positive rate, and outperform the results obtained with our algorithm. However, the false negative rate is also quite high, meaning that less red eyes are detected than with our algorithm. More-

over the application does not correct red eyes for non-Caucasian people.

Figure 12. *Image not corrected by HP Redbot.*

Our algorithm easily corrects red eyes in portraits and does not make any distinction between Caucasian and non-Caucasian people. Moreover, the false positive rate is very low since a zone has to pass a large set of tests to be considered as an eye. If the sclera cannot be detected, our application cannot correct the eye. According to our tests, our approach is more accurate and robust than most existing applications.

Conclusion

We have presented an intuitive and robust algorithm for the automatic detection and correction of red eyes in digital pictures. Our method is based on the detection of round red regions, in combination with sclera and skin pixels. It corrects red eyes for people with any type of skin tone.

We have tested the accuracy and robustness of our algorithm on a large set of test images, and compared it to three other automatic red-eye removal applications. The results show that our algorithm performs better than other existing algorithms in terms of compromise between false positive and false negative rate. The software is available online at http://ivrgwww.epfl.ch/software/.

Further work is oriented towards improving the overall quality of the correction. It would be interesting to address the problems encountered for people with glasses, and to study more natural correction methods.

Acknowledgment

The authors would like to thank Sabine Süsstrunk from the Images and Visual Representation Group (IVRG) at Ecole Polytechnique Fédérale de Lausanne (EPFL) for her advice and for having contributed to this collaborative effort.

References

[1] B. Smolka, K. Czubin, J. Y. Hardeberg, K. N. Plataniotis, M. Szczepanski, K. Wojciechowski, "Towards Automatic Red Eye Effect Removal", Pattern Recognition Letters, vol. 24, no. 11, pp. 1767-1785, 2003.

[2] P. J. Benati, R. T. Gray and P. A. Cosgrove, "Automated Detection and Correction of Eye Color Defects Due to Flash Illumination", US Patent 5,748,764, 1998.

[3] Redbot, Hewlett-Packard Labs, "RedBot automatic red eye correction", http://redbot.net/.

[4] Raimondo Schettini, Francesca Gasparini and Fadi Chazli, "A modular procedure for automatic red eye correction in digital photos", Proc. SPIE Color Imaging IX: Processing, Hardcopy, and Applications, pp. 139-147, vol. 5293, 2004.

[5] F. Gasparini and R. Schettini, "Automatic Redeye Removal for Smart Enhancement of Photos of Unknown Origin", Proc. 8th International Conference on Visual Information Systems, Lecture Notes in Computer Science, Vol. 3736, pp. 226-233, 2005.

[6] Luo Huitao, J. Yen and D. Tretter, "An efficient automatic redeye detection and correction algorithm", Proc. International Conference on Pattern Recognition, pp. 883-886, vol. 2, 2004.

[7] Sergey Ioffe, "Red Eye Detection with Machine Learning", Proc. IEEE International Conference on Image Processing, pp. 871-874, vol. 2, 2003.

[8] Rein-Lien Hsu, Mohamed Abdel-Mottaleb and Anil K. Jain, "Face Detection in Color Images", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 5, pp. 696-706, 2002.

[9] R. W. G. Hunt, "The Reproduction of Colour in Photography, Printing & Television", Fountain Press, 5th edition, 1995.

[10] Jon Y. Hardeberg, "Red Eye Removal using Digital Color Image Processing", Proc. IS&T Image Processing, Image Quality, Image Capture Systems (PICS), pp. 283-287, 2001.

Author Biography

Flavien Volken received his BS in communication systems in 2004 from the Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland. In July 2006 he will obtain a MS degree from the Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland.

Johann Terrier received his BS in communication systems in 2004 from the Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland. In July 2006 he will obtain a MS degree specialized in information and communication security from the Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland.

Patrick Vandewalle received the MS degree in electrical engineering from Katholieke Universiteit Leuven, Belgium in 2001. From 2001 to 2002, he worked as a research assistant in the Medical Imaging lab at the department of electrical engineering (ESAT), K.U.Leuven. He is currently pursuing a PhD degree in computer, communication and information sciences at the Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland. His research interests are in signal and image processing, sampling, and digital photography.

