

**Image Processing**

**Project Documentation**

**Red Eye Detection and Removal From**

**Digital Images**

**Student:** Anca Mihai

**Group:** 30432

**Assistant Teacher:** Horațiu Florea

**Academic Year:** 2023-2024

**1. Introduction**

In dimly lit environments, individuals often utilize flash photography to illuminate subjects. However, the use of a flash can lead to an occurrence known as the *red-eye* effect. This phenomenon is caused by the flash reflecting off the blood vessels behind enlarged pupils and subsequently reaching the camera lens, resulting in the eyes appearing red in photographs.

Red-eye effect will never occur if *we do not use flash light* when we take a picture. Another light source can be used to control brightness instead of flash light. But it is *impractical* to carry another lighting device and most of people use flash light when it is dark.

Red-eye removal methods are categorized into two classes: *physics-based* and *software-based*. A *physics-based* method is to *prevent* red-eye effect: the distance between *flash and lens* can be increased so that the lens is located outside the red-eye beacon. However, the size of a camera *should be large enough*. Or the size of pupil can be made *small* by using *pre-exposure* flash. Camera will consume much power and people feel annoyed when seeing the pre-exposure flash.

A *software-based* method post-processes digital photographs using algorithms to remove existing red-eye in them. Algorithms for red-eye removal are researched by many corporations and laboratories: a lot of image editing software tools offer the function of red-eye removal and some companies developed the software for red-eye removal to apply to their products. However, a lot of these tools are implemented with *automatic* algorithms, that have *poor pupil segmentation* which leads to unnatural red eye correction: sometimes they correct red eye pixels too aggressively, *darkening eyelid areas*; or too conservatively, leaving *many red eye pixels uncorrected*.

In general, red-eye removal algorithms are composed of two parts: red-eye detection and red-eye correction.

1. *Red eye detection*:

Red eye detection strategies can be broadly divided into more classes. One of them assumes that *candidate eye regions* are somehow *identiﬁed*, either manually or automatically. In my solution, it is assumed there is a rectangle selection of the area on the image where the eyes are placed.

In most approaches the color portion of the image, the candidate to contain a pupil, is converted into a new image. It is typically a *gray scale* one, usually deﬁned as the *redness map*, and different transformations can be adopted to generate it. The candidate red pupils are usually located binarizing the redness map using, for instance, empirically deﬁned *thresholds*. Morphological ﬁlters or geometric constraints and other considerations described are then usually adopted to discriminate from true red pupils and other red spots.

As examples, according to *Benati [5]*, to identify red eye pixels, ﬁrst a threshold in the HLS color space is applied, then the pixels are grouped into spatially contiguous regions, and a score is assigned to each region based on size, shape, color and brightness. The region with the highest score corresponds to the pupil to be corrected.

Furthermore, the algorithm developed by *Gasparini and Schettini [3]* looks for red eyes in the regions with high value of redness, deﬁned as:

*Redness = (max (0, (2R-(G+B))/R))^2*

Then, to limit the number of false hits the algorithm exploits some geometric constraints: in particular, the *percentage ratio* between the *area of the candidate red* eye and the *whole face*, the *red pixel spatial distribution* and the *roundness of the region* considered.

Based on the methods presented above, my main solution is the following:

* redness: in HSV space, red has value 0 for *Hue*; for this reason, taking into account also the variations of red, the target regions would have the Hue either slightly smaller than 359 or slightly bigger than 0
* geometric shape: for detecting the pupil, from the red regions detected above will be considered the first/two biggest ones who have the shape similar to the one of a circle (such that to not consider other red elements which could be characteristic for a face – ex: red skin tone); for validating the geometric constraints (roundness) *Hough algorithm* or the *thinness ratio* can be used

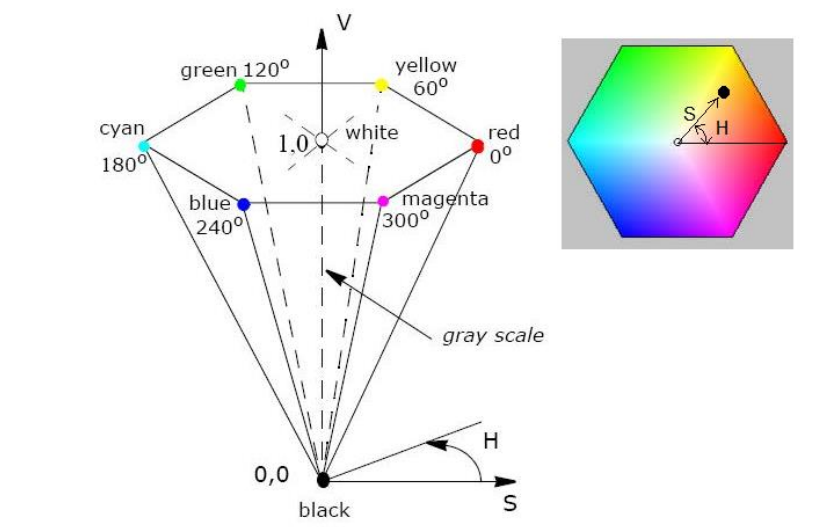


Figure 1: HSV Color Space

2. *Red eye correction*:

Most red-eye correction algorithms *desaturate* *red color* component from red-eyes. Of course, they distinguished their algorithms from other algorithms by proposing the method that makes corrected red-eyes more natural.

*Patti[6]* proposed a simple red eye color correction where all the detected red eye pixels are replaced by a gray value of 0.8 of their *luminance value*. This factor is experimentally determined that yields a natural correction of the defective pixels. Before applying this color correction, a morphological pruning is performed on the mask, to avoid the correction of non-pupil regions, such as eyelids.

Other approaches adopt very simple corrections such as *Wu [7]*, where the red color of the defected eyes are simply substituted by black. Corrections of this type could be very dangerous leading to a processed image which is even worse than the defective original.

Moreover, *Gaubatz and Ulichney[8]* desaturated red color in proportion to redness in order to soften the boundary of red-eye.

Taking into consideration all the above, a method for red eye correction would be to desaturate the affected area by reducing the intensity, which can be done in a straightforward manner, by making all RGB channels have the value of the average between G and B channels.

**2. Dataset**

The dataset I choose to use in developing this project consists of 10 images containing instance of red-eye occurrences in humans. These images were obtained from publicly accessible repositories, some of them even can be found in the examples offered by the research papers mentioned in the previous chapter. The images from this set were selected because they offer an accurate representation of a red eye generated by a digital camera and because they provide a wide range of scenarios (backgrounds included) and lightning environments: in this way they support thorough examination and assessment of the algorithm I propose for red-eye detection and correction.

C++ programming language along with the OpenCV library will be used for the implementation, which will be tested with the dataset mentioned right before. The powerful capabilities of OpenCV in image processing and computer vision make it suitable for effectively perform tasks related to detecting and correcting red-eye issues.

**References**

**[1]** Seunghwan Yoo and Rae-Hong Park. Red-eye detection and correction using inpainting in digital photographs. *Consumer Electronics, IEEE Transactions on*, 55:1006 – 1014, 09 2009

**[2]** Francesca Gasparini. *Automatic red-eye removal for digital photography*. 09 2008

**[3]** F. Gasparini and R. Schettini, *“Automatic redeye removal for smart enhancement of photos of unknown origin,”* *Lecture Notes in Computer Sciences*, vol. 3736, pp. 226–233, 2006.

**[4]** Sepideh Samadzadegan. *Automatic and Adaptive Red Eye Detection and Removal - Investigation and Implementation*. 05 2012

**[5]** P. Benati, R. Gray, and P. Cosgrove, *“Automated detection and correction of eye color defects due to ﬂashillumination,”* US Patent 5748764, May 1998.

**[6]** A. Patti, K. Konstantinides, D. tretter and Q. Lin, *"Automatic digital redeye reduction," Image Processing*, vol. 3, pp. 55-59, 1998.

**[7]** D. Wu, *“Automatic red eye removal,”* US Patent 0232481, October 2005.

**[8]** M. Gaubatz and R. Ulichney*, “Automatic red-eye detection and correction,”* in Proc. IEEE Int. Conf. Image Processing, vol. 1, pp. 804–807, Rochester, NY, Sep. 2002.