Documentație la Proiect de detecție și corectare a ochiilor roșii

## Procesare de Imagini

### Proiect realizat de Cărășel Ionuț-Ștefan

# Cuprins:

1. Raționament si taskmanagent
2. Implementare
3. Testare
4. Îmbunătățiri

# 1.Raționament și task management

Ceea ce își propune să rezolve acest proiect, este sa detecteze efectul „ochii roșii” din anumite imagini digitale fără utilizarea inteligenței artificiale. Acest fenomen, cauzat de reflexia luminii asupra retinei, poate afecta calitatea estetică a imaginii.

Se pornește de la ipoteza că putem identifica ochii roșii prin pixelii care au intensitatea canalului roșu prea mare, iar intensitatea celorlalte canale redusă. Prin analiza acestuia, se aplică praguri și operații morfologice pentru a izola zonele afectate.

Pentru detecție, sunt folosite contururi extrase din masca binară obținută prin procesare, iar zonele sunt încadrate cu dreptunghiuri (cv::Rect). Acestea sunt apoi transmise către modulul de corectare, care modifică pixelii roșii pe baza informațiilor din canalele verde și albastru.

La acest proiect ne vom folosi de urmatoarele tehnologii:

* Limbajul C++;
* Biblioteca OpenCV;
* Mediul de dezvoltare: CLion;
* Set de imagini pentru testare.

**Functionalitățiile proiectului:**

* **Încărcarea imaginii:**
* Citirea unei imagini din path-ul din memorie primit in format cv::Mat;
* Probleme posibile: path-ul greșit sau imagine lipsă
* Rezolvare verificarea existenței și validare.
* **Conversia în Spațiul YCrCb:**
  + Conversie în spațiul YCrCb pentru separarea canalului Cr (roșu);
  + Canale separate pentru analiză;
  + Probleme posibile: Contrast slab între zone;
  + Rezolvare: Ajustarea pragului de binarizare.
* **Generarea de mască:**
  + Se aplică binarizare și operații morfologice pentru curățare;
  + Masca este în forma cv::Mat;
  + Problemă posibilă: Zgomot;
  + Rezolvare: Median Blur.
* **Detecția ochilor:**
  + Se extrag contururi și se delimitează zonele roșii cu cv::Rect;
  + Posibile probleme: Zone lipsă sau greșit încadrate;
  + Rezolvare: Arie minimă de filtrare
* **Corectarea culorii ochilor roșii:**
  + Se înlocuiesc pixelii roșii cu o valoare apropiată de culoarea naturală;
  + Posibile probleme: Neînlocuire completă sau înlocuire prea agresivă;
  + Rezolvare: O conditie pentru un minim de Roșu și un maxim de Verde și Albastru în comparație cu intensitatea de Roșu.
* **Salvarea imaginilor corectate:**
  + Se salvează local imaginea cu funcția cv::imwrite();

# 2. Implementare

Arhitectura proiectului este modulară și se bazează pe patru clase principale: ImageLoader, RedEyeDetector, RedEyeCorrector și RedEyeProcessor. Acestea sunt integrate astfel încât să permită detectarea și corectarea automată a efectului de ochi roșii dintr-o imagine de intrare.

**Modulul ImageLoader**

**Funcționalități:**

* Încărcarea imaginii de intrare folosind cv::imread().
* Salvarea imaginii procesate cu cv::imwrite().

**Alegerea design-ului:**

* Clasa este statică, deoarece nu necesită instanțiere.
* Am separat logica de I/O pentru a respecta principiul responsabilității unice.

**ImageLoader.h:**#ifndef IMAGELOADER\_H  
#define IMAGELOADER\_H  
#include <string>  
#include <opencv2/core/mat.hpp>  
  
  
class ImageLoader {  
 public:  
 static cv::Mat *loadImage*(const std::string& path);  
 static void *saveImage*(const std::string& path, const cv::Mat& image);  
 private:  
 ImageLoader() = delete;  
};  
  
#endif //IMAGELOADER\_H

**ImageLoader.cpp:**

#include "ImageLoader.h"  
#include <opencv2/imgcodecs.hpp>  
  
cv::Mat ImageLoader::*loadImage*(const std::string& path) {  
 return cv::imread(path,cv::IMREAD\_COLOR);  
}  
void ImageLoader::*saveImage*(const std::string& path,const cv::Mat& image) {  
 cv::imwrite(path,image);  
}

**Modulul RedEyeDetector**

**Funcționalități:**

* Convertirea imaginii în spațiul de culoare YCrCb pentru a evidenția componentele roșii (Cr).
* Crearea unei măști binare pe baza pragului aplicat canalului Cr (valori mari ale roșului).
* Aplicarea de filtre (median blur, morfologie de tip „open”) pentru reducerea zgomotului.
* Detectarea contururilor pe mască și generarea unor regiuni cv::Rect extinse în jurul ochilor roșii.

**Decizii de design:**

* Am ales spațiul YCrCb deoarece canalul Cr oferă separare mai bună pentru culoarea roșie față de HSV în cazul ochilor roșii.
* Extinderea dreptunghiurilor de detecție (Rect) s-a realizat pentru a include complet zona afectată.

**RedEyeDetector.h:**

#ifndef REDEYEDETECTOR\_H  
#define REDEYEDETECTOR\_H  
#include <opencv2/core/mat.hpp>  
  
  
class RedEyeDetector {  
public:  
 explicit RedEyeDetector(const cv::Mat& image);  
 std::vector<cv::Rect> detect();  
 cv::Mat getMask() const;  
  
private:  
 cv::Mat originalImage;  
 cv::Mat mask;  
  
 void generateMask();  
};  
  
#endif //REDEYEDETECTOR\_H

**RedEyeDetector.cpp:**

#include "RedEyeDetector.h"  
  
#include <opencv2/imgproc.hpp>  
#include <opencv2/imgcodecs.hpp>  
  
RedEyeDetector::RedEyeDetector(const cv::Mat &image) {  
 originalImage = image.clone();  
 generateMask();  
}  
  
void RedEyeDetector::generateMask() {  
 cv::Mat imgYCrCb;  
 cv::cvtColor(originalImage, imgYCrCb, cv::COLOR\_BGR2YCrCb);  
  
 std::vector<cv::Mat> channels;  
 cv::split(imgYCrCb, channels);  
 cv::threshold(channels[1], mask, 165, 255, cv::THRESH\_BINARY);  
  
 cv::medianBlur(mask, mask, 5);  
 cv::Mat kernel = cv::getStructuringElement(cv::MORPH\_ELLIPSE, cv::Size(5, 5));  
 cv::morphologyEx(mask, mask, cv::MORPH\_OPEN, kernel);  
 cv::imwrite("D:\\PI\\masca\_detectata.jpg", mask);  
}  
  
std::vector<cv::Rect> RedEyeDetector::detect () {  
 std::vector<std::vector<cv::Point>> contours;  
 std::vector<cv::Rect> redEyeRegions;  
  
 cv::findContours(mask, contours, cv::RETR\_EXTERNAL, cv::CHAIN\_APPROX\_SIMPLE);  
  
 for (const auto& contour : contours) {  
 cv::Rect box = cv::boundingRect(contour);  
 if (box.area() > 100) {  
 cv::Rect expandedBox = box;  
 int expand = 5;  
 expandedBox.x = std::max(0, box.x - expand);  
 expandedBox.y = std::max(0, box.y - expand);  
 expandedBox.width = std::min(originalImage.cols - expandedBox.x, box.width + 2 \* expand);  
 expandedBox.height = std::min(originalImage.rows - expandedBox.y, box.height + 2 \* expand);  
 float aspectRatio = static\_cast<float>(expandedBox.width) / expandedBox.height;

if (aspectRatio > 0.75 && aspectRatio <1.65) {  
 redEyeRegions.push\_back(expandedBox);  
}

}  
 }  
  
 cv::Mat debugImage = originalImage.clone();  
 for (const auto& box : redEyeRegions) {  
 cv::rectangle(debugImage, box, cv::Scalar(0, 255, 0), 2);  
 }  
 cv::imwrite("D:\\PI\\detectie\_contururi.jpg", debugImage);  
  
 return redEyeRegions;  
}  
  
cv::Mat RedEyeDetector::getMask() const {  
 return mask.clone();  
}

**Modulul RedEyeCorrector**

**Funcționalități:**

* Parcurge fiecare regiune identificată și verifică fiecare pixel.
* Dacă pixelul este considerat prea roșu (red > green + blue + prag), atunci este corectat prin înlocuirea valorii de roșu cu media canalelor verde și albastru.

**Design:**

* Corectarea este locală, fără a afecta alte regiuni.
* Folosim o corecție simplă, dar eficientă, fără a introduce artefacte vizibile.

**RedEyeCorrector.h:**

#ifndef REDEYECORRECTOR\_H  
#define REDEYECORRECTOR\_H  
#include <opencv2/core/mat.hpp>  
  
class RedEyeCorrector {  
public:  
 static void *correct*(cv::Mat& image, const std::vector<cv::Rect>& regions);  
};  
  
#endif //REDEYECORRECTOR\_H

**RedEyeCorrector.cpp:**

#include "RedEyeCorrector.h"  
#include <opencv2/imgproc.hpp>  
  
void RedEyeCorrector::*correct*(cv::Mat& image, const std::vector<cv::Rect>& regions) {  
 for (const auto& region : regions) {  
 cv::Mat roi = image(region);  
  
 for (int y = 0; y < roi.rows; ++y) {  
 for (int x = 0; x < roi.cols; ++x) {  
 cv::Vec3b& pixel = roi.at<cv::Vec3b>(y, x);  
 uchar blue = pixel[0];  
 uchar green = pixel[1];  
 uchar red = pixel[2];  
  
 if (red > 80 && red > green + 30 && red > blue + 30) {  
 uchar newRed = static\_cast<uchar>((green+blue)/2);  
 pixel[2] = newRed;  
 }  
 }  
 }  
 }  
}

**Modulul RedEyeProcessor**

**Funcționalități:**

* Leagă toate modulele: încarcă imaginea, detectează ochii roșii, aplică corectarea și salvează imaginea rezultată.
* Afișează informații utile în consolă, precum numărul de regiuni detectate.

**Decizii tehnice:**

* Afișarea imaginilor originale și corectate (resize automat dacă imaginea e prea mare).
* Logica principală e concentrată în această clasă pentru a putea fi ușor testată și extinsă.

**RedEyeProcessor.h:**

#ifndef REDEYEDETECTOR\_H  
#define REDEYEDETECTOR\_H  
#include <opencv2/core/mat.hpp>  
  
  
class RedEyeDetector {  
public:  
 explicit RedEyeDetector(const cv::Mat& image);  
 std::vector<cv::Rect> detect();  
 cv::Mat getMask() const;  
  
private:  
 cv::Mat originalImage;  
 cv::Mat mask;  
  
 void generateMask();  
};  
  
#endif //REDEYEDETECTOR\_H

**RedEyeProcessor.cpp:**

#include "RedEyeProcessor.h"  
#include "ImageLoader.h"  
#include "RedEyeDetector.h"  
#include "RedEyeCorrector.h"  
#include <iostream>  
  
#include <opencv2/highgui.hpp>  
#include <opencv2/opencv.hpp>  
  
void RedEyeProcessor::run(const std::string& inputPath, const std::string& outputPath) {  
 cv::Mat image = ImageLoader::*loadImage*(inputPath);  
 if (image.empty()) {  
 std::cerr << "Eroare: imaginea nu a putut fi încărcată de la: " << inputPath << std::endl;  
 return;  
 }  
  
 double scaleFactor = 1.0;  
 int maxWidth = 800;  
 int maxHeight = 600;  
  
 if (image.cols > maxWidth || image.rows > maxHeight) {  
 double scaleX = static\_cast<double>(maxWidth) / image.cols;  
 double scaleY = static\_cast<double>(maxHeight) / image.rows;  
 scaleFactor = std::min(scaleX, scaleY);  
 }  
  
 cv::Mat resizedOriginal;  
 cv::resize(image, resizedOriginal, cv::Size(), scaleFactor, scaleFactor);  
 cv::imshow("Imagine Originala", resizedOriginal);  
  
 RedEyeDetector detector(image);  
 auto regions = detector.detect();  
  
 if (regions.empty()) {  
 std::cout << "Nu au fost detectate zone de ochii rosii.\n";  
 } else {  
 std::cout << "S-au detectat " << regions.size() << " regiuni de ochii rosii.\n";  
 RedEyeCorrector::*correct*(image, regions);  
 }  
  
 cv::Mat resizedCorrected;  
 cv::resize(image,resizedCorrected,cv::Size(),scaleFactor, scaleFactor);  
 cv::imshow("Imagine Corectata", resizedCorrected);  
 ImageLoader::*saveImage*(outputPath, image);  
 std::cout<<"Imaginea a fost salvata la:"<<outputPath<<std::endl;  
 cv::waitKey(0);  
}

# 3. Testare

Pentru a evalua eficiența algoritmului de detectare și corectare a efectului de ochi roșii, am dezvoltat un plan de testare structurat, care include:

* Testare pe caz comun (poză de portret standard)

A close up of a person's eyes

AI-generated content may be incorrect.

* Testare pe cazuri limită (poze de portret cu elemente roșii in fundal, sau ruj)

A collage of a person with red eyes

AI-generated content may be incorrect.

A collage of a person's face

AI-generated content may be incorrect.

* Testare pe imagini care nu au ochii rosii, dar are elemente roșii

A person sitting in front of a fire

AI-generated content may be incorrect.

* Testare pe imagini fără ochii roșii

A collage of a cat

AI-generated content may be incorrect.

**Rezumat testare:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nr Test** | **Scenariu** | **Rezultat așteptat** | **Rezultat obținut** | **Observații** |
| **1** | Portret standard cu ochii roșii | Corectare completă | Corectare completă | Succes ambii ochii detectați |
| **2** | Portret standard cu ochii roșii și ruj | Corectare doar pe ochii | Corectare completă | Ambii ochi detectați, doar culoarea corectata nu este 100% corecta |
| **3** | Portret standard cu ochii roșii și nuanțe roșii în fundal | Corectare doar pe ochii | Corectare completă, dar detecția în plus a unor zone adiționale | Algoritmul a detectat și a corectat în plus anumite zone |
| **4** | Poze fără ochii roșii dar cu zone roșii (poza cu focul de tabără) | Fără corectare falsă | Corectarea falsă a focului de tabara | Algoritmul a detectat si a corectat culoarea focului |
| **5** | Poze fără ochii roșii | Fără corectare falsă | Fără corectare falsă | Algoritmul nu a detectat nimic |

**Metrici de performanță**

Rata de detecție corectă din 5 imagini:

* Corectare completă: 70%
* False positive: 30%

**Concluzie**

Etapa de testare a evidențiat atât punctele forte cât și limitările actualului algoritm. Cu o rată de succes de peste 70% în scenarii standard, soluția este funcțională, dar necesită îmbunătățiri pentru cazurile-limită.