

Détection et correction automatique des yeux rouges

Réalisé par :

Dorra Larnaout [larnaout.dorra@yahoo.fr]

Mohamed Amine Salem [medsalem@gmail.com]

1 Table des matières

2	Abstract	3
3	Introduction.....	4
4	Phénomène des yeux rouges	5
5	Choix de l'environnement de développement.....	5
6	Procédure de détection	6
6.1	Détection des visages.....	6
6.2	Détection des yeux	8
7	Correction des yeux rouges.....	9
7.1	Problématique	9
7.2	Approche mise en œuvre	10
7.2.1	Modèle HSV	10
7.2.2	Création du masque	11
7.2.3	Correction	12
8	Résultat : Avantages et limites	14
8.1	Avantages	14
8.2	Limites	15
9	Idées d'amélioration	15
10	Conclusion.....	17
11	Bibliographie.....	18

2 Abstract

Automatic detection of objects and correction of images is a main interest in Computer Vision. Among problems faced in photography, we can find the red-eye effect which is the common appearance of red pupils in color photographs of eyes. In order to overcome such a problem, new cameras have anti-red eyes systems; a “preflash” that will cause a contraction of pupils and reduce the transmission of light. In some cases, these systems do not have their expected effect, therefore many algorithms and methods were proposed for reducing eye coloration artifacts in an image.

In this paper we present a simple method for red-eye detection and correction in color images. First of all we try to detect faces, and then we try to detect the eyes in each face. Finally, using both the RGB and the HSV color spaces, the red pupils are detected and corrected. The results highly depend of chosen thresholds.

Keywords: red-eye, yeux rouges, face, visage, detection, détection, correction, OpenCV

3 Introduction

Le traitement d'image est un domaine en plein essor, dans lequel il s'agit d'améliorer des images ou bien d'en tirer des informations utiles. Dans ce contexte, la détection d'objets et de formes suscite l'intérêt de plus en plus de chercheurs qui proposent des algorithmes de plus en plus efficaces et innovants permettant de remplir des fonctionnalités de manière aussi performante, voire la plupart du temps plus performante que l'être humain.

N'échappant pas à la règle, les photos prises par des appareils à flash nécessitent un traitement particulier pour remédier à un problème très répandu et qui est celui des yeux rouges. Pour cela, les appareils photos récents intègrent un système de préflash qui permet d'éliminer cet effet indésirable et inesthétique.

Cependant, l'amélioration matérielle a des limites qui sont impossibles à franchir, il est alors inévitable de se tourner vers des solutions logicielles qui permettent de réduire le parasite des yeux rouges. Et justement, dans ce projet il était question de développer un outil permettant d'atteindre ce résultat de façon automatisée.

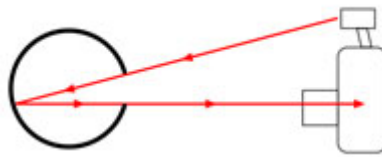
Dans le cadre de la filière Imagerie 3D et Environnements Virtuels du master 2 Science Informatique de l'Université de Marne-la-Vallée, il nous a été demandé développer un outil de détection et de correction automatique des yeux rouges dans une image.

On se propose dans ce rapport de présenter dans un premier temps le phénomène d'un point de vue scientifique. On passera ensuite aux différentes étapes suivies pour détecter les zones d'intérêt. Suite à quoi on expliquera les différents traitements apportés à l'image afin d'éliminer le parasite des yeux rouges. On finira ensuite en analysant les résultats obtenus.

4 Phénomène des yeux rouges

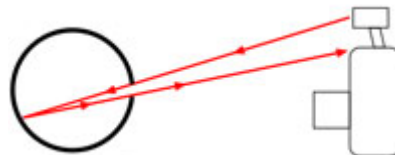
Ce phénomène arrive la plupart du temps lors de la prise de photos dans des endroits mal éclairés. En effet, la pupille de l'œil se dilate dans le noir pour capter le maximum de lumière. Mais lors de la prise de vue et face à la forte lumière du flash, l'iris n'a pas le temps de se contracter et donc la lumière éclaire le fond de l'œil qui est richement vascularisé.

A cause de la forme sphérique de l'œil, la réflexion a une forme de cône comme le montre le schéma ci-dessous :



Lorsque le flash et l'objectif sont assez proches l'un de l'autre, la lumière est comme émise par l'objectif lui-même et aura tendance à se refléter et revenir vers ce dernier.

Comme énoncé dans l'introduction, les nouveaux appareils photos sont équipés d'un système permettant d'éviter ce phénomène. Ce système consiste à envoyer une série de plusieurs flashes de courte durée avant le flash final. Ainsi, lors de la prise de vue proprement dite, la pupille a une taille suffisante pour empêcher la lumière d'aller éclairer la rétine et revenir vers l'objectif, et ce comme le montre le schéma suivant :



5 Choix de l'environnement de développement

Etant donné que c'est la principale étape de notre projet, on a dû passer beaucoup de temps à étudier l'état de l'art dans le but de trouver la meilleure façon de parvenir à détecter les yeux rouges, ce qui facilitera ultérieurement la correction de la teinte rouge.

On a choisi d'utiliser OpenCV pour développer notre outil et ce pour maintes raisons :

- C'est une bibliothèque « Open Source »
- C'est une bibliothèque de traitement d'images et de vision par ordinateur en langage C/C++, optimisée, proposée par Intel pour Windows et Linux
- Elle offre un très grand nombre d'opérateurs "classiques" et d'outils pour traiter les images et faciliter la reconnaissance d'objets et de formes dans des contextes

et des applications simples sans forcément connaître les algorithmes qui se cachent derrière.

L'installation de cette bibliothèque sur Visual Studio C++ (Windows) est exclusivement manuelle et est très pénible. Pour ce faire, on a suivi un tutoriel [<http://www.robotika-vision.com/instalasi.pdf>] plutôt bien détaillé, ce qui ne nous a pas épargné le passage par des bricolages potentiellement dangereux, notamment la modification des noms de plusieurs fichiers dll, etc...

6 Procédure de détection

Pour espérer corriger automatiquement l'effet des yeux rouges dans les photographies, il est indispensable de commencer par la détection des visages.

6.1 Détection des visages

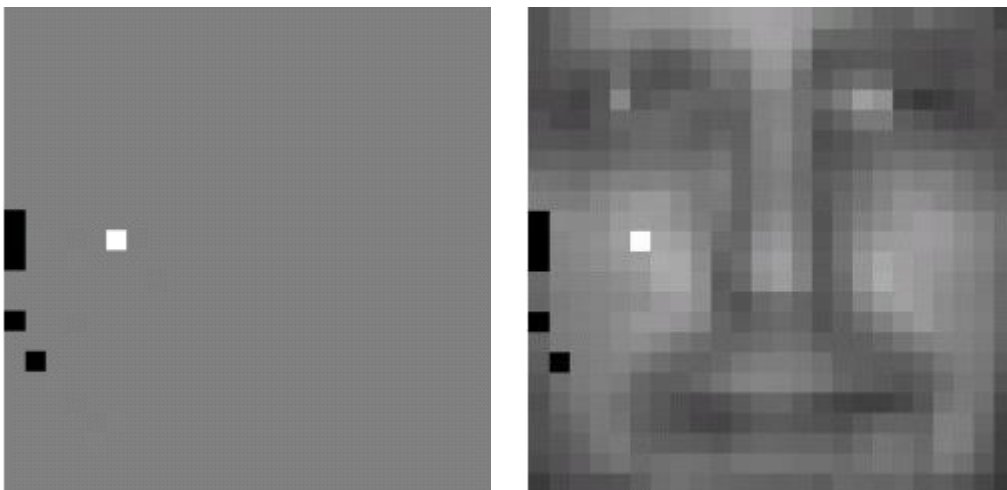
Détecter un visage signifie le localiser avec une précision suffisante dans l'espace, en donnant par exemple un cadre dans lequel le visage se situe.

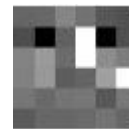
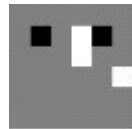
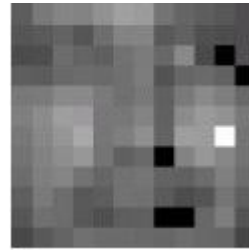
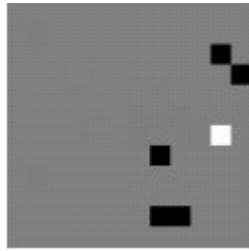
Sous OpenCV, cette étape est assurée par une simple procédure qui se base sur l'algorithme d'Adaboost en utilisant les « classifieurs ».

Pour cela, le programme se sert de plusieurs données d'apprentissage préétablies. Ces données ont permis construire une sorte de catalogue des motifs à repérer, des motifs caractérisant des visages, dans notre cas.

Qu'est ce qu'un motif ?

Le motif dont il est question ici, est composé d'un ensemble de points (de pixels) dits « de contrôle » et répartis en deux groupes ; appliquer un masque sur une image consiste à vérifier le fait que tous les points du premier groupe soient plus lumineux que chacun des points du deuxième groupe (exemple : voir l'ensemble des images qui suivent) :





Exemples de masques contenant des points de contrôle

Exemples de masques appliqués à des images

Les motifs trouvés suite à l'apprentissage sont utilisés pour construire une pyramide, qui est composée de plusieurs étages, chacun contenant un ou plusieurs motifs. Ces derniers (les motifs) sont classés par ordre de discrimination ; les moins discriminants étant ceux qui génèrent le plus de fausses détections tout en conservant le plus de vrais visages. Pris séparément, les motifs sont des critères « faibles » de discrimination, dans le sens où un seul d'entre eux ne permet pas de faire la différence entre un visage et autre chose. Mais utilisés ensemble, ils créent un critère « fort », qui ne laisse passer presque aucune fausse détection tout en conservant au mieux tous les vrais visages. En résumé, chaque motif pris tout seul donne plusieurs « faux positifs », mais utilisés ensemble, les motifs donnent presque 100% de vrais positifs, et très peu de faux positifs.

L'objectif de cette structure pyramidale est de faire en sorte que les images traitées et qui ne comportent pas de visage soient rejetées le plus vite possible, sans générer trop de calculs inutiles. Le premier étage est constitué de masques très simples et peu nombreux, qui permettent de rejeter un nombre moyen d'images sans visages. Plus on descend dans la pyramide et plus les étages comportent de masques et, par conséquent, plus ils sont discriminants. De façon générale, chaque étage de la pyramide est constitué de plusieurs masques et est associé à un seuil. Chaque masque qui le compose est appliqué à l'image et renvoie une valeur : la somme des valeurs retournées doit être supérieure ou égale au seuil de l'étage pour que l'image puisse continuer le parcours dans la pyramide.

La figure suivante illustre un exemple réussi de détection de visages:



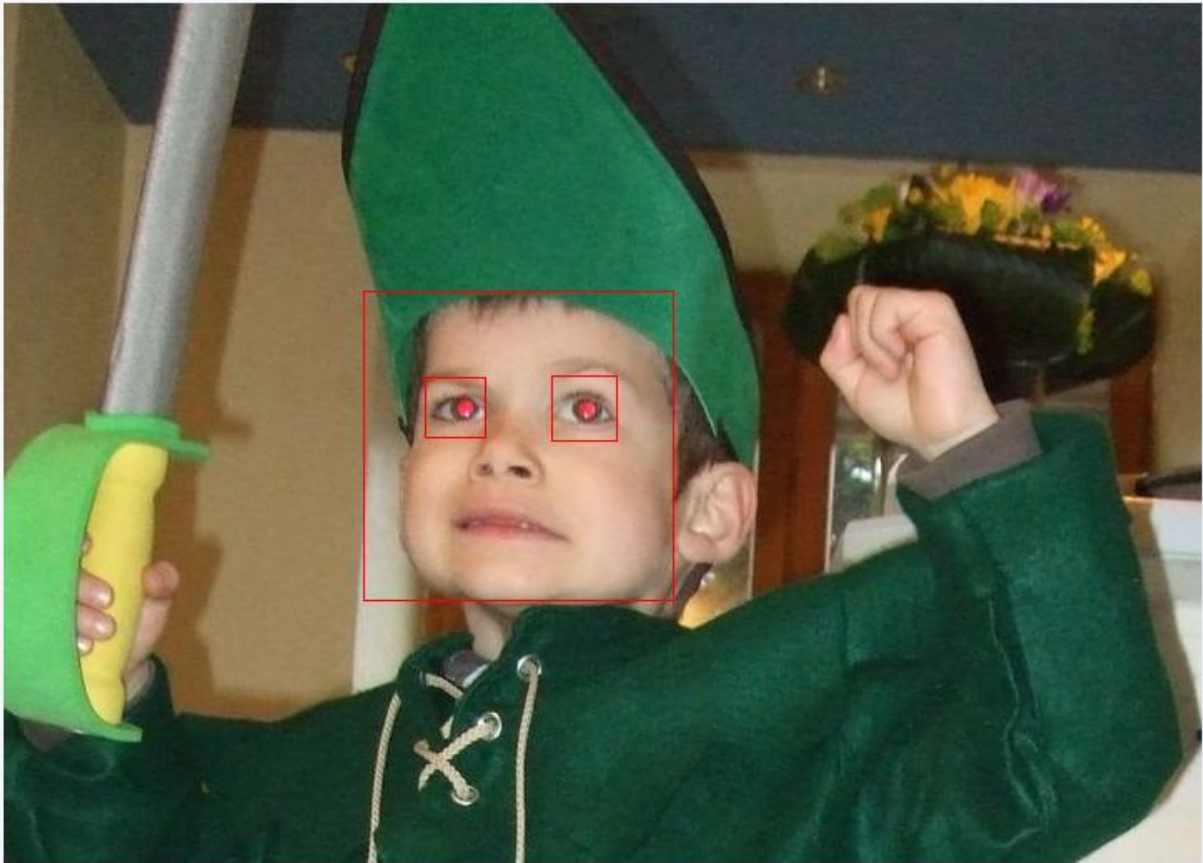
6.2 Détection des yeux

Pour effectuer cette étape il existe également une fonction sous OpenCV qui permet d'avoir le résultat attendu avec le minimum de contraintes. En effet, cette méthode est basée sur le même principe que celui de la détection des visages.

L'intérêt d'une telle méthode réside dans le fait qu'il est tout à fait possible de caractériser d'autres objets que des visages, et donc de parvenir à détecter d'autres formes intéressantes, en l'occurrence les yeux.

Cette fonction nous permet également de récupérer les coordonnées et les dimensions d'un rectangle contenant l'œil détecté, ce qui nous facilitera ultérieurement le repérage de l'artéfact des yeux rouges et sa correction.

La figure suivante illustre un exemple réussi de détection des yeux:



7 Correction des yeux rouges

Une fois la phase de détection est achevée, on peut entamer l'étape de correction qui consiste tout simplement à colorer en noir tous les pixels rouges représentant la pupille.

7.1 Problématique

A fin d'assurer une correction exacte, il indispensable de trouver un critère **robuste** nous permettant de **délimiter exactement la zone de l'œil à corriger**. En effet, le critère le plus intuitif est sans aucun doute la couleur rouge marquante qui caractérise la pupille en question. Toutefois, si on se contente du codage RGB (rouge, vert, bleu), il est très difficile de trouver un seuil générique qu'on peut utiliser pour différentes photos étant donné que le niveau de la couleur rouge change d'une image à une autre (voir images ci-dessous)





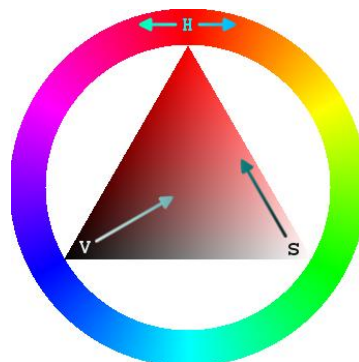
Cette constatation implique qu'il faut trouver un autre espace colorimétrique qui facilitera notre travail et nous garantira des bons résultats.

Dans ce qui suit, on exposera l'approche qu'on a adoptée pour la correction des yeux rouges. Tout d'abord, on va essayer de trouver un intervalle contenant les différentes valeurs de rouge caractérisant les pupilles en question. Ensuite, on créera un masque délimitant les pupilles en se basant sur l'intervalle déjà trouvé. Enfin, on utilisera ce masque pour effectuer la correction.

7.2 Approche mise en œuvre

7.2.1 Modèle HSV

L'espace colorimétrique HSV ou encore TSV est celui qui s'approche le plus à la façon dont les êtres humains perçoivent les couleurs ce qui n'est pas toujours le cas avec le modèle RGB. Cette importante propriété peut être exploitée dans notre cas afin de distinguer exactement la couleur et la luminosité du rouge dans les zones d'intérêt obtenues à l'issue de l'étape de détection. En effet, la couleur est déterminée d'abord en sélectionnant la Teinte puis la Saturation et la Valeur.



Vu que le rouge des pupilles varie selon l'orientation du visage par rapport à l'appareil et diffère d'une image à une autre, on a choisi de le caractériser par les valeurs TSV suivantes :

- Une Teinte inférieure à 3 ou supérieure à 357
- Une Saturation quelconque.
- Une Valeur quelconque.

C'est cette plage de valeurs qui va nous permettre de créer le masque à utiliser par la suite.

7.2.2 Création du masque

Le but de cette partie est de créer une image binaire contenant le moins de bruit possible et dans laquelle les pixels ayant une Teinte appartenant à l'intervalle précisé dans la section précédente auront une valeur de « 255 ». Les autres pixels auront comme valeur « 0 ». Par conséquent, les pupilles seront représentées par des tâches blanches dans l'image binaire.

Voici alors l'algorithme utilisé :

Données : image codé en HSV : « `imgHSV` », intervalle de couleurs : « `interval_` »

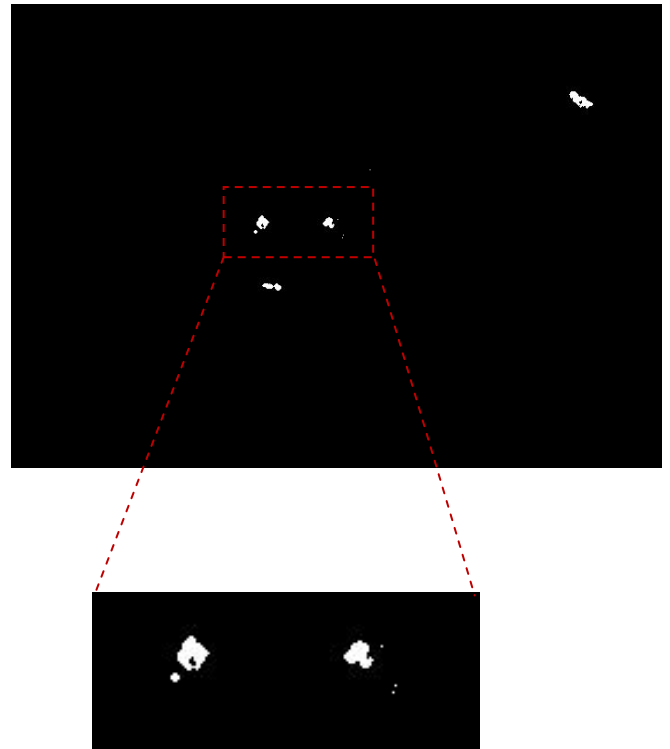
Sortie : image binaire « `masque` »

Pour tout pixel « `p` » appartenant à `imgHSV`

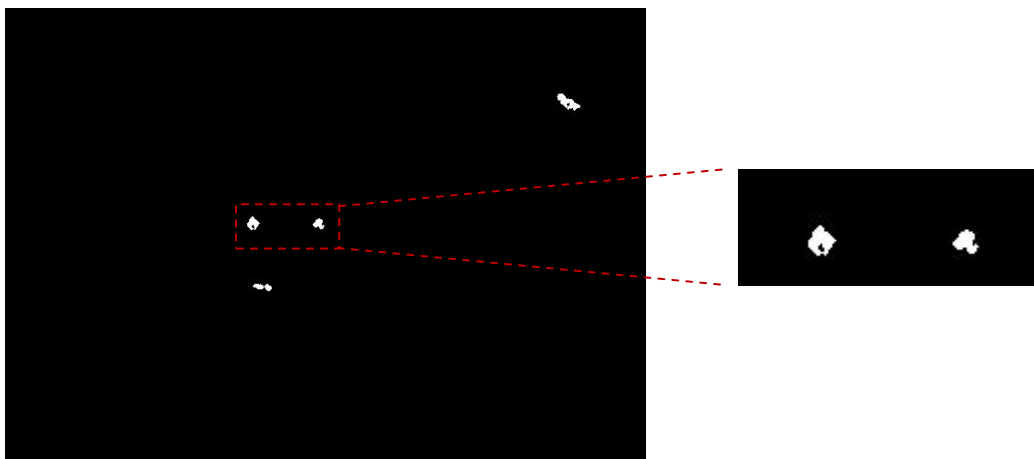
```
{
    Si (la Teinte au niveau du pixel « p » appartient à interval_) alors
    {
        masque [p]=255 ;
    }
    Sinon
    {
        masque [p]=0 ;
    }
}
```

Retourner `masque` ;

Ci-dessous une image couleur ainsi que le masque obtenu :



On remarque bien que le résultat est bruité. Ainsi pour réduire ces bruits, notamment ceux qui existent dans la région des yeux, on a eu recours à des fonctions morphologiques à savoir l'érosion et la dilatation. En effet, on commence par appliquer une érosion qui permettra d'éliminer les petites tâches. Puis, on dilate le résultat obtenu par le même élément structurant afin de retrouver la forme initiale des tâches. Voici l'image binaire après cette opération :



En réduisant l'image à notre zone d'intérêt, on retrouve bien le masque désiré.

7.2.3 Correction

Une fois le masque est créé, tout ce qu'il reste à faire c'est la phase de correction. Le principe de cette correction se résume aux traitements suivants :

- Parcourir pixel par pixel la région des yeux de l'image initiale
- Si la valeur du pixel courant au niveau du masque vaut 255 alors on change la couleur du pixel au niveau de l'image initiale.

Ceci est équivalent à l'algorithme ci-dessous :

Données : image codé en RGB : « *imgRGB* », image binaire « *masque* »

Sortie : image codé en RGB : « *imgRGB* »

Pour tout pixel « *p* » appartenant à *imgRGB*

```
{
    Si (masque [p]=255 ) alors
    {
        imgRGB [p]=noir ;
    }
}
```

Retourner *imgRGB* ;

En appliquant cet algorithme sur la même image de départ que précédemment, on obtient la photo suivante :



Autres résultats :



8 Résultat : Avantages et limites

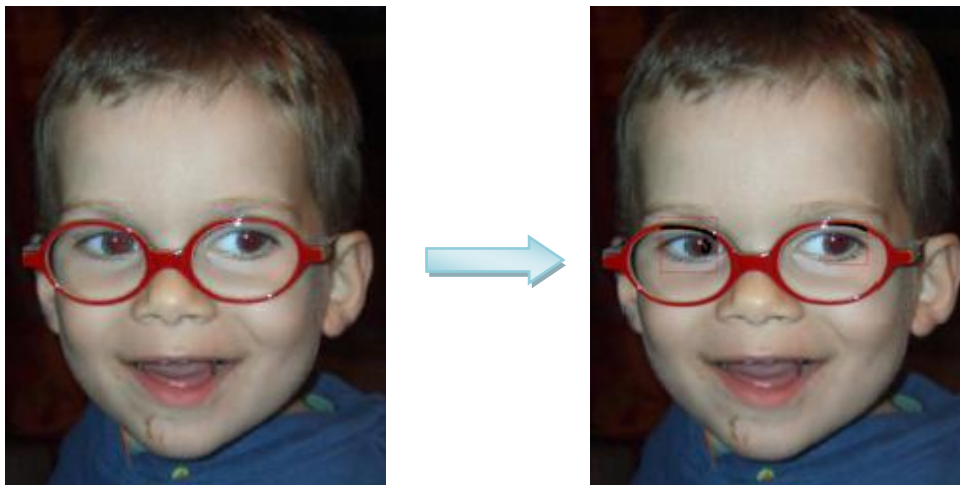
8.1 Avantages

- La méthode mise en œuvre dans notre programme constitue une approche précise robuste et rapide de détection des visages.
- La localisation exacte des faces garantit une détection meilleure des régions des yeux, même pour des orientations différentes des visages.
- La délimitation des régions contenant les yeux nous a permis de réduire l'espace de travail. Ceci nous permet de gagner en terme de temps d'exécution d'une part et d'augmenter le taux d'exactitude de notre correction d'autre part.
- L'algorithme de correction utilisé est très facile à mettre en œuvre et s'applique uniquement au niveau des régions des yeux.

8.2 Limites

Malgré la robustesse de l'algorithme de détection des visages, on constate que ce dernier est incapable de reconnaître les visages inclinés au delà d'un certain degré et ceci est dû aux différents masques évoqués dans la section 6.1.

Bien que la partie de détection nous garantissons des bons résultats, l'algorithme de correction fonctionne moins bien. En effet, on obtient les meilleurs résultats dans le cas le plus classiques c'est-à-dire quand la luminosité dans le rouge est très marquée. Toutefois si le rouge est peu intense, la correction apportée par l'algorithme est partielle. De plus, si un objet rouge autre que les pupilles à corriger se trouve dans la région d'intérêt, il sera aussi coloré en noir. (Voir photo ci-dessous)



9 Idées d'amélioration

Pour améliorer la première partie à savoir la détection des visages, on peut augmenter le nombre de masques sur lesquels on effectue l'apprentissage en ajoutant des modèles de visages inclinés selon des degrés différents.

En observant les résultats de notre algorithme, on constate bien que la forme noire retrouvée au niveau de la pupille après la phase de correction n'est pas souvent elliptique. En se basant sur cette remarque, on a eu l'idée de remplacer cette tâche noire par une ellipse qui vérifie la propriété suivante :

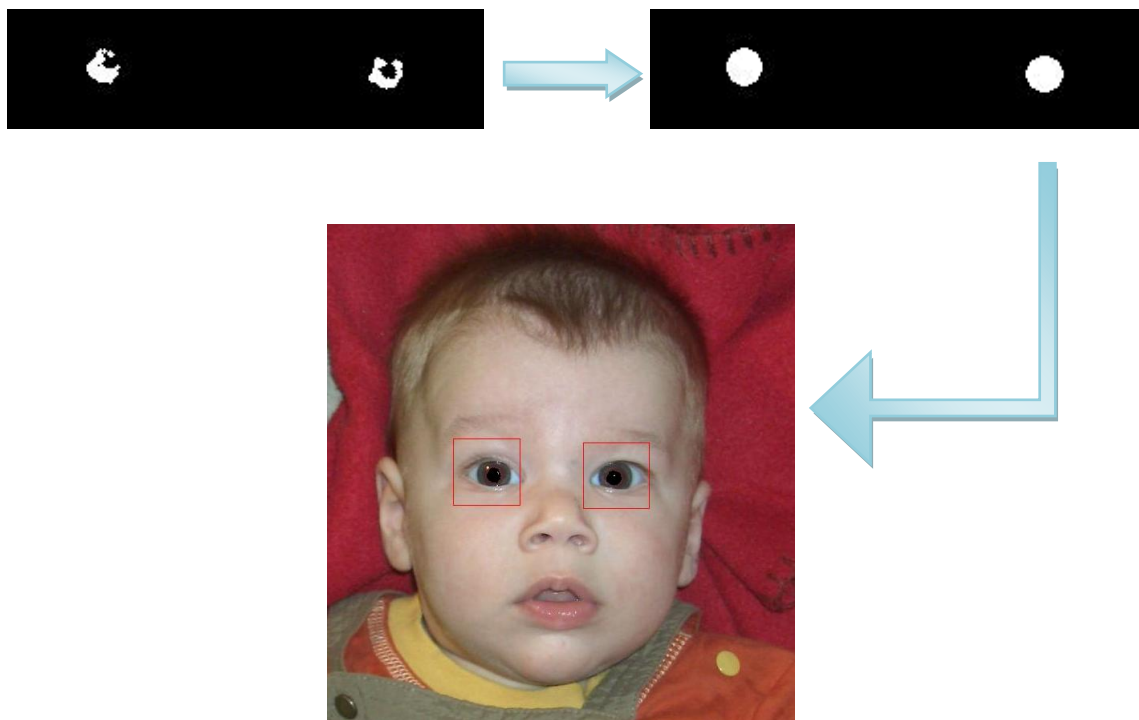
L'ellipse à dessiner sera la plus petite ellipse ayant le même air que la tâche et l'englobe le mieux.

Cette modification au niveau du masque à utiliser pourrait améliorer les résultats en éliminant les débordements d'une part et en perfectionnant la forme des pupilles.

Voici à quoi doit ressembler le résultat après amélioration :



Autre exemple :



10 Conclusion

Ces travaux ont été pour nous l'occasion de découvrir et de comprendre le fonctionnement de la détection d'objets (ici des visages et des yeux), mais aussi de comprendre un algorithme moderne, efficace et très présent dans des logiciels du commerce: la détection automatique et l'élimination des yeux rouges.

Dans ce rapport, nous avons essayé d'expliquer l'approche que nous avons adoptée pour traiter le problème des yeux rouges en s'inspirant de plusieurs études (voir références) et certains programmes existants sur internet.

Certes l'algorithme que nous avons proposé traite bien le cas le plus courant des yeux rouges. Néanmoins, on pense que cette méthode peut être améliorée en appliquant les idées suggérées dans la section « Idées d'amélioration ». Ainsi, on espère que notre travail servira aux autres étudiants afin de réussir à réaliser un outil complet robuste et rapide.

11 Bibliographie

Step by Step Installation of OpenCV 2.0, <http://www.robotika-vision.com/instalasi.pdf>

Detecting Colors using RGB Color Space,
<http://myopencv.wordpress.com/2009/06/13/detecting-colors-using-rgb-color-space/>

Evaluation of Red-eye Correction Tools, Tessera Application Note, July 2009.

An Efficient Automatic Redeye Detection and Correction Algorithm, Huitao Luo, Jonathan Yen and Dan Tretter, Hewlett-Packard Labs.

Etude de la pertinence topologique des descripteurs d' images utilisés dans les algorithmes de détection de visages par apprentissage, Pierre Lemaire.

FIN