



---

## RAPPORT ÉCRIT

### Projet HELP

Hope to Emulate the Life of Paralyzed  
people

---

#### Rédigé par :

Semestre 3	Semestre 4
Hussain Al Othman	Hussain Al Othman
Katleen Blanchet	Katleen Blanchet
Titouan Boulmier	Titouan Boulmier
Laure Dupasquier	Pierre Jacquot
Pierre Jacquot	
Marie-Alice Schweitzer	

**Sous la direction de :**  
Ali Mansour et Olivier Reynet

22/05/2015

## **Remerciements**

Nous tenons à remercier notre encadrant, M. Ali Mansour, pour ses conseils dans la réalisation physique du projet et sa disponibilité.

Nous témoignons également nos remerciements à M. Olivier Reynet pour ses précisions sur l'ingénierie système et pour son accompagnement tout au long du projet.

Nous exprimons notre gratitude à M. Victor Rohu pour ses conseils pour améliorer notre rapport.

Nous remercions aussi les responsables des U.V. 3.4 et 4.4 pour leur présentation des techniques de gestion de projet et la mise en place des ateliers techniques.

Enfin nous tenons à exprimer notre reconnaissance à l'ensemble du personnel de la médiathèque pour leur disponibilité et leurs éclaircissements sur la recherche de documents.

# Sommaire

<b>Remerciements</b>	<b>2</b>
<b>Introduction au projet</b>	<b>6</b>
<b>1 État de l'art</b>	<b>7</b>
1.1 La détection de pupilles . . . . .	7
1.1.1 L'œil . . . . .	7
1.1.2 Caractéristiques d'une image d'un œil acquise par caméra traditionnelle . . . . .	8
1.1.3 Les différentes problématiques de la détection par caméra traditionnelle . . . . .	8
1.1.4 Détection par éclairage infrarouge . . . . .	10
1.1.5 Traitement d'images . . . . .	11
1.2 Méthode de détection de la direction du regard . . . . .	22
1.3 Présentation des technologies existantes . . . . .	23
1.3.1 Systèmes avec contact . . . . .	23
1.3.2 Systèmes sans contact . . . . .	26
1.4 Solutions techniques et méthodes envisagées . . . . .	28
1.4.1 Au préalable, suite à l'état de l'art . . . . .	28
1.4.2 Méthodes choisies lors de la réalisation . . . . .	29
<b>2 Dossier fonctionnel</b>	<b>30</b>
2.1 Ingénierie des exigences . . . . .	30
2.1.1 Approche Top-Down . . . . .	30
2.1.2 Approche Bottom-Up . . . . .	31
2.1.3 Fonctions principales du système . . . . .	32
2.2 Spécification fonctionnelle 3 axes . . . . .	33
2.2.1 Raffinement FAST . . . . .	33
2.2.2 Spécification des données . . . . .	34
2.2.3 Spécification des comportements . . . . .	34
2.3 Architecture fonctionnelle . . . . .	35
<b>3 Implémentation</b>	<b>37</b>
3.1 Architecture physique et interfaces . . . . .	37
3.1.1 Interface d'acquisition . . . . .	38
3.1.2 Interface logicielle . . . . .	38
3.1.3 Interface graphique . . . . .	38

3.2	WBS . . . . .	39
3.3	Matériel utilisé . . . . .	40
3.3.1	Ordinateur . . . . .	40
3.3.2	Caméra . . . . .	41
3.3.3	Mentonnière . . . . .	41
3.4	Plan de validation fonctionnelle . . . . .	42
3.5	Tests unitaires . . . . .	42
3.5.1	Test Unitaire 1 : détection de l'ouverture de l'œil . . . . .	43
3.5.2	Test Unitaire 2 : clic . . . . .	45
<b>4</b>	<b>Organisation</b>	<b>47</b>
4.1	Méthodes de travail . . . . .	47
4.2	Outils pour les échanges . . . . .	47
4.3	Répartition des tâches dans le temps . . . . .	48
<b>5</b>	<b>Partie technique</b>	<b>52</b>
5.1	Le détecteur . . . . .	52
5.1.1	Principe de fonctionnement . . . . .	52
5.1.2	Acquisition du flux vidéo et transformation en images matricielles . . . . .	54
5.1.3	Détection du visage . . . . .	55
5.1.4	Détection des yeux . . . . .	57
5.1.5	Détermination du centre des pupilles . . . . .	57
5.1.6	Déplacement du curseur en fonction de la position des centres	59
5.1.7	Clic . . . . .	60
5.2	Interface graphique . . . . .	61
5.2.1	But et enjeux de l'interface . . . . .	61
5.2.2	Critères d'exigence . . . . .	61
5.2.3	Outils de réalisation . . . . .	63
5.3	Limites du système et problèmes rencontrés . . . . .	63
5.3.1	Problèmes matériels . . . . .	63
5.3.2	Limites du détecteur . . . . .	63
5.3.3	Limites d'utilisation . . . . .	65
<b>Conclusion</b>	<b>67</b>	
<b>Annexes</b>	<b>68</b>	
<b>A Plan de validation fonctionnelle</b>	<b>69</b>	
<b>B Programme permettant d'acquérir un flux vidéo</b>	<b>78</b>	
<b>C Code permettant de récupérer le flux vidéo et de le convertir en images matricielles</b>	<b>80</b>	
<b>D Haar_cascades permettant de détecter le visage et les yeux</b>	<b>81</b>	

<b>E Matrice contenant les rectangles encadrant le visage</b>	<b>82</b>
<b>F Méthode detectMultiScale</b>	<b>83</b>
<b>G Changement de repère des coordonnées du centre des pupilles dans la méthode findEyes</b>	<b>84</b>
<b>H Algorithme du gradient</b>	<b>85</b>
<b>I Fiche de validation fonctionnelle complétée</b>	<b>86</b>
<b>Bibliographie</b>	<b>90</b>

# **Introduction**

## **Contexte**

Dans le cadre des U.V. 3.4 et 4.4, notre équipe a choisi de développer le projet HELP (Hope to Emulate the Life of Paralyzed people) proposé par M. Mansour. Ce projet a pour objectif la réalisation d'un système permettant de remplacer la souris d'ordinateur grâce aux mouvements des yeux. Cette étude semble très intéressante car elle demande une analyse et une compréhension de systèmes complexes d'eye tracking (oculométrie) déjà existants afin de mettre au point une version simplifiée et moins onéreuse. De plus, elle réunit différents aspects du travail d'ingénieur en informatique, tels que le traitement de l'image, l'algorithme et le travail en équipe. Enfin, ce projet peut éventuellement mener à deux finalités différentes : d'abord, l'aide aux personnes tétraplégiques, qui, grâce à ce système, pourraient être moins dépendantes et retrouver un peu de liberté. Puis, ce projet pourrait aussi être utilisé afin d'aider les scientifiques à mettre au point un robot, travaillant en zones hostiles, qui puisse être contrôlé facilement grâce à la détection des mouvements de la tête et des yeux de l'opérateur.

## **Expression initiale du besoin**

Le but premier du projet était le développement d'un système permettant à une personne tétraplégique d'utiliser un ordinateur et surfer sur internet. Cependant, face à l'ampleur de la tâche, et suite à un entretien avec nos encadrants, nous avons décidé de commencer par développer un système permettant à une personne ordinaire de contrôler un ordinateur. Ainsi, le dispositif développé doit permettre à un utilisateur d'exécuter différentes applications sans avoir besoin de toucher une souris ou un clavier. L'usager doit pouvoir effectuer les opérations usuelles en bougeant et clignant des yeux.

# Partie 1

## État de l'art

### 1.1 La détection de pupilles

#### 1.1.1 L'œil

Il est important de commencer par un rappel des caractéristiques biologiques de l'œil. Le lecteur acquerra certaines notions de bases qui lui permettront de mieux appréhender les différentes problématiques de la détection de pupilles. La figure 1.1 nous présente les différentes parties de l'œil.

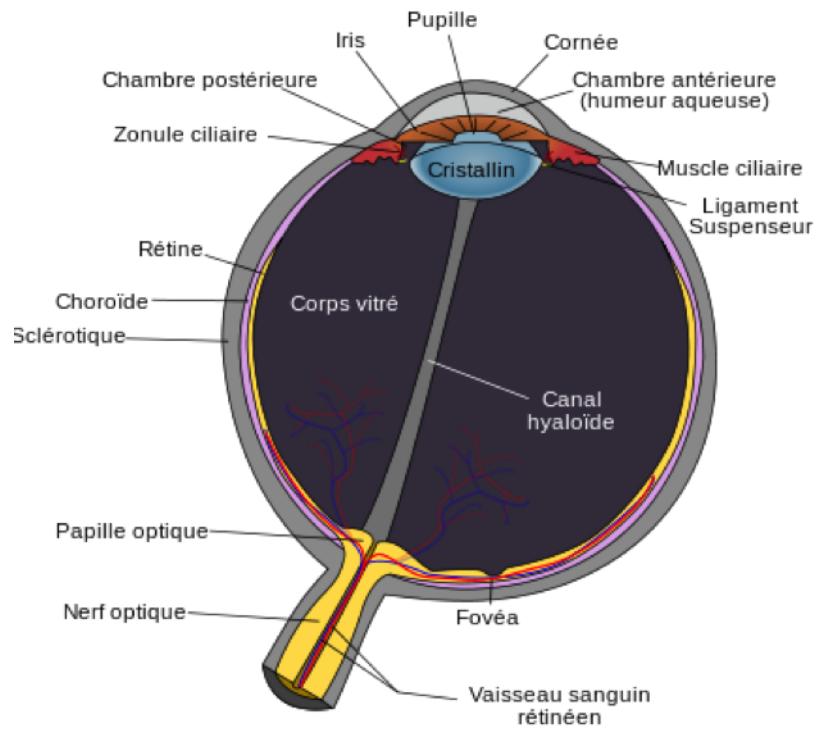


FIGURE 1.1 – Schéma anatomique de l'œil humain

Dans le cadre de la détection de pupilles, nous nous intéresserons plus particulièrement à la partie visible de l'œil. Cette partie externe est composée de 3 zones :

- La partie centrale : la pupille. C'est un orifice noir permettant de laisser passer la lumière.
- La bande colorée entourant la pupille : l'iris. Il permet de plus ou moins dilater la pupille.
- La zone blanche qui recouvre le reste de la partie visible : la sclérotique.

La figure 1.2 présente ces 3 parties.

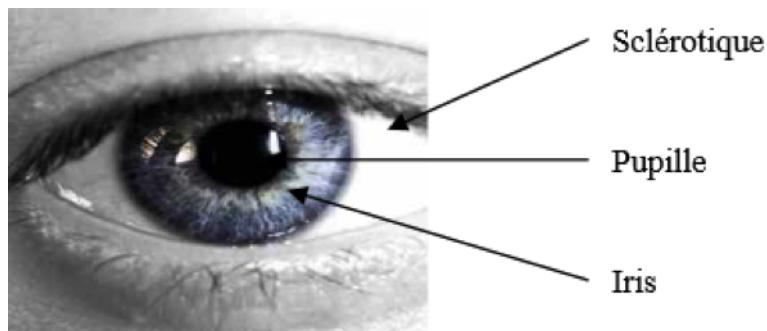


FIGURE 1.2 – Les différentes parties externes de l'œil humain

Nous allons maintenant nous intéresser à la capture d'une image d'un œil, à son traitement puis à la détection de la pupille.

### 1.1.2 Caractéristiques d'une image d'un œil acquise par caméra traditionnelle

Il existe deux types de caméras pour acquérir l'image d'un œil : la caméra traditionnelle et la caméra infrarouge. Alors que la caméra traditionnelle permet de capturer le spectre de couleurs visibles par l'œil humain, la caméra infrarouge permet de capter les ondes infrarouges de la lumière naturelle (voir figure 1.3). Cependant, la lumière naturelle ne comprend que très peu de composantes infrarouges. Ainsi, il est souvent nécessaire de soumettre l'objet d'étude (l'œil pour notre projet) à une source de lumière infrarouge supplémentaire afin d'améliorer la qualité et la luminosité de l'image acquise. La caméra infrarouge permet d'éviter la majorité des problèmes de réflexion rencontrés par une caméra traditionnelle. Nous pouvons voir figure 1.4(a) et figure 1.4(b) les différents types d'image.

### 1.1.3 Les différentes problématiques de la détection par caméra traditionnelle

#### 1.1.3.1 Ombres

La première difficulté rencontrée lors de la détection de pupilles par caméra traditionnelle est la présence de l'ombre des cils (voir figure 1.5). Cet ombrage est

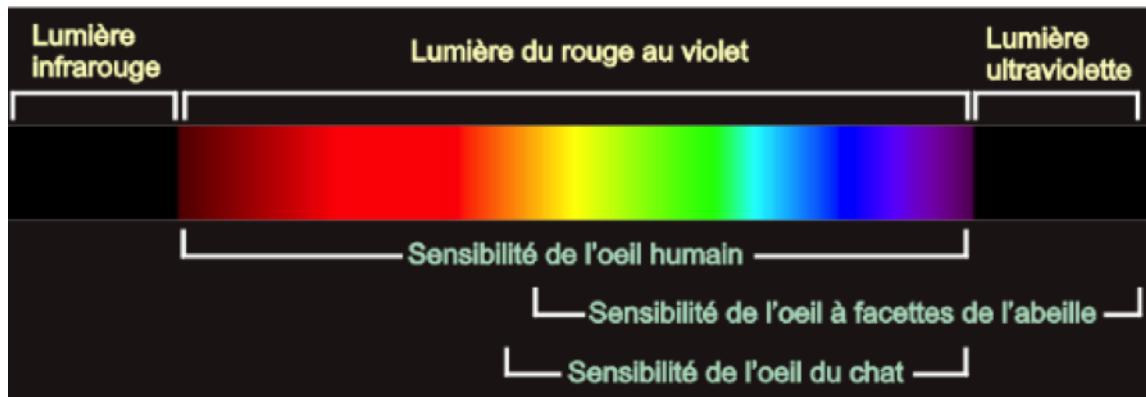


FIGURE 1.3 – Contenu spectral de la lumière



(a) Traditionnelle

(b) Infrarouge

FIGURE 1.4 – Capture d’œil à l’aide d’une caméra

dû à un angle trop grand entre la source de lumière et l’axe de la pupille. Afin de réduire ce phénomène au maximum, une attention particulière doit être portée au positionnement et à l’axe de la source de lumière.

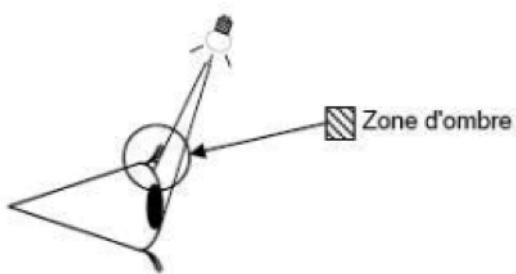


FIGURE 1.5 – Formation d’ombre dans une image d’un œil

### 1.1.3.2 Reflets

La surface de l’œil étant lisse et recouverte par la cornée, structure transparente recouvrant et protégeant l’iris (figure 1.1), les reflets émanant de l’œil sont considérables. L’intensité des reflets sont fonction de l’intensité lumineuse dégagée par la scène. Plus la scène sera lumineuse, plus elle sera reflétée dans l’œil.

Cependant, ces reflets sont gênants pour le traitement de l'image, et plus particulièrement pour la reconnaissance de la pupille. Par exemple, la figure 1.4(a) présente un œil avec un reflet recouvrant une partie de la pupille et une partie de l'iris. Ainsi, la pupille n'est plus tout à fait ronde et cela peut poser problème lors de sa détection.

Une attention particulière devra être portée aux éclairages de la pièce afin d'éviter les réflexions lumineuses ambiantes dues aux fenêtres, lampes et néons.. Cependant, il est important de noter qu'une réduction des sources lumineuses n'est pas une solution car l'image perdrait en clarté, en qualité, et son traitement n'en serait que plus difficile. La solution réside plus dans l'harmonisation de l'éclairage afin de ne pas créer de zone de réflexion.

Plusieurs approches sont possibles pour la résolution de ce problème de réflexion : soit par l'application d'opérateurs morphologiques<sup>1</sup> en dilatant puis érodant la zone atteinte par la réflexion, soit en considérant qu'un reflet possède des caractéristiques contraires à celles de la pupille. Une source lumineuse étant très claire, on peut définir des bornes de niveau de gris à partir de l'histogramme de l'image et effectuer une segmentation (opérations de traitement d'images). Le seuillage de l'image originale de la pupille et l'image arborant les zones de réflexions seront combinés afin de former une image où les trous provoqués par les sources lumineuses seront remplis.

Il faut aussi noter que le port de verres correcteurs ou de lentilles accentue le phénomène de reflets.

#### 1.1.3.3 Pupille

Il faut noter que si l'axe de la source de lumière correspond à celui des pupilles, il en résultera un effet d'yeux rouges qui peut être gênant pour le traitement de l'image et surtout la détection des pupilles. De plus, la taille des pupilles varie en fonction de l'intensité lumineuse présente. Il faudra donc veiller à ne pas imposer une source lumineuse trop importante afin d'avoir une taille de pupille suffisamment grande pour une bonne détection.

#### 1.1.4 Détection par éclairage infrarouge

Si la caméra infrarouge permet d'éviter la majorité des problèmes de réflexion rencontrés par une caméra traditionnelle, sa mise en place est néanmoins plus complexe (utilisation d'émetteurs infrarouges afin d'éclairer l'œil, utilisation de caméras infrarouges, ...) et plus cher. En effet, l'utilisation de caméras infrarouges multiplierait le coût du système par deux ou trois.

La détection de la pupille par illumination de l'œil avec des éclairages infrarouges se décline en deux grandes méthodes : la méthode dite de la pupille lumineuse (bright pupil) et celle de la pupille noire (dark pupil) (voir figure 1.6). Suivant la position de l'éclairage infrarouge la pupille aura des propriétés optiques

---

1. La morphologie mathématique est une théorie et technique mathématique et informatique d'analyse. Son développement est inspiré des problèmes de traitement d'images, domaine qui constitue son principal champ d'application. Elle fournit en particulier des outils de filtrage, segmentation, quantification et modélisation d'images.

différentes. Si l'illumination infrarouge est coaxiale avec l'axe optique de la caméra alors la pupille apparaîtra totalement blanche (bright pupil). Au contraire si la source lumineuse est décalée (toujours par rapport à l'axe optique), alors la pupille apparaîtra noire (dark pupil) et l'on pourra voir distinctement les reflets de la source lumineuse infrarouge sur la cornée du sujet.

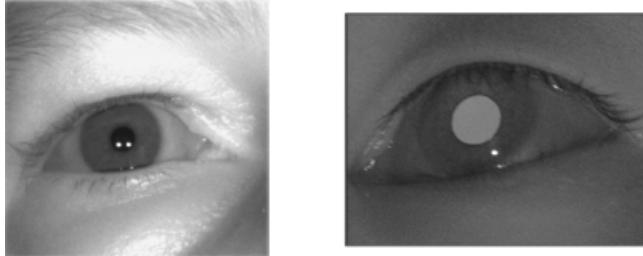


FIGURE 1.6 – Effet black et bright pupil

### 1.1.5 Traitement d'images

#### 1.1.5.1 Segmentation

La segmentation (aussi appelée Clustering) [3] est une étape de base du traitement d'images qui a pour but de séparer différentes zones homogènes d'une image en groupes (clusters) dont les membres ont en commun diverses propriétés (intensité, couleur, texture...). En d'autres mots, cette opération rassemble des pixels entre eux suivant des critères prédéfinis. Les pixels sont ainsi regroupés en régions, qui constituent un pavage ou une partition de l'image. Il peut s'agir par exemple de séparer les objets du fond. On peut regrouper les différentes méthodes de segmentation en deux catégories : la segmentation non supervisée, qui vise à séparer automatiquement l'image en différents clusters naturels (sans aucune connaissance préalable des classes) ; et la segmentation supervisée, qui s'opère à partir de la connaissance de chacune des classes définie. Concernant les méthodes de segmentation non supervisée, nous nous intéresserons à deux méthodes généralement employées pour le traitement d'images : le Fuzzy C-means et le k-means.

#### Description des algorithmes

Les principales étapes d'un algorithme de classification sont :

1. La fixation arbitraire d'une matrice des classes.
2. Le calcul des centroïdes des classes.
3. Le réajustement de la matrice d'appartenance suivant la position des centroïdes.
4. Calcul du critère de minimisation et retour à l'étape 2 s'il y a non convergence de critère.

Fuzzy C-Means (FCM) est un algorithme de classification non supervisée floue. Il introduit la notion d'ensemble flou dans la définition des classes : chaque point

(pixel) dans l'ensemble des données appartient à chaque cluster avec un certain degré d'appartenance, et tous les clusters sont caractérisés par leur centre de gravité. Ainsi, il permet d'obtenir une partition floue de l'image en donnant à chaque pixel un degré d'appartenance compris entre 0 et 1 à un cluster donné. Le cluster auquel est associé un pixel est celui dont le degré d'appartenance est le plus élevé.

La classification floue des objets est décrite par une matrice floue à n lignes et c colonnes dans laquelle n est le nombre d'objets de données et c est le nombre de grappes.  $\mu_{ij}$  est l'élément de la i-ème ligne et la colonne j.  $\mu$ , représente le degré de fonction d'appartenance du i-ème objet avec le pôle j. Les étapes de l'algorithme FCM sont présentées figure 1.7 :

---

**Algorithm 1** Pseudo code de Fuzzy C-Means

---

```

1: for  $t = 1, 2, \dots$  do
2:   Step1 Calcule des centres de clusters  $c_i^{(t)} = \frac{\sum_{j=1}^N (\mu_{ij}^{(t-1)})^m x_j}{\sum_{j=1}^N (\mu_{ij}^{(t-1)})^m}$ 
3:   Step2 Calcule des distances  $D_{ijA}^2$  avec :

$$D_{ijA}^2 = (x_j - c_i)^T A (x_j - c_i), \quad 1 \leq i \leq n_c, 1 \leq j \leq N.$$

4:   Step3 Mise à jour de la matrice des partitions floues :

$$\mu_{ij}^{(t-1)} = \frac{1}{\sum_{k=1}^{n_c} (D_{ikA}/D_{kjA})^{2/(m-1)}}$$

5: end for
6: return  $\|U^{(t)} - U^{(t-1)}\| < \epsilon$ 

```

---

FIGURE 1.7 – Pseudo Code de Fuzzy C-Means

Le FCM est un algorithme très puissant, mais son inconvénient principal réside dans l'initialisation des centres.

L'algorithme k-means est l'algorithme de Clustering le plus connu et le plus utilisé (simple à mettre en œuvre). Il permet de partitionner les pixels d'une image en K clusters. L'algorithme k-means ne crée qu'un seul niveau de clusters. Il renvoie une partition des données, dans laquelle les objets à l'intérieur de chaque cluster sont aussi proches que possible les uns des autres et aussi loin que possible des objets des autres clusters. Chaque cluster de la partition est défini par ses objets et son centroïde. Le k-means est un algorithme itératif qui minimise la somme des distances entre chaque objet et le centroïde de son cluster. La position initiale des centroïdes conditionne le résultat final, de sorte qu'ils doivent être initialement placés le plus loin possible les uns des autres de façon à optimiser l'algorithme. K-means réitère ses opérations jusqu'à ce que la somme ne puisse plus diminuer. Le résultat est un ensemble de clusters compacts et clairement séparés, à condition d'avoir choisi la bonne valeur K du nombre de clusters. Les principales étapes de l'algorithme k-means sont :

1. Choix aléatoire de la position initiale des K clusters.
2. (Ré-)Affecter les objets à un cluster suivant un critère de minimisation des distances (généralement selon une mesure de distance euclidienne).
3. Une fois tous les objets placés, recalculer les K centroïdes.

4. Réitérer les étapes 2 et 3 jusqu'à ce que plus aucune réaffectation ne soit faite.

Statistics Toolbox implémentée sous MatLab 7 contient la fonction kmeans.m, facile à prendre en main et bien documentée et Fuzzy Logic Toolbox contient fcm.m.

## Comparaison

Il apparaît que ces deux algorithmes sont assez efficaces pour des images couleurs et permettent une bonne segmentation. Cependant, lors de la présence de défauts (reflets, ombres...), la segmentation paraît correcte (elle ne permet pas une bonne détection des caractères par Optical Character Recognition OCR, ce qui ne nous intéresse pas). Enfin, il apparaît que ces algorithmes ne sont pas adaptés à des images contenant un grand nombre d'objets (au niveau du temps de calcul). La méthode FCM semble plus efficace pour la haute résolution, et au contraire, k-means convient plus aux images à faible résolution.

### 1.1.5.2 Traitement de l'image pour la détection de pupilles

La détection des pupilles est composée de plusieurs étapes et peut être codée à l'aide de plusieurs méthodes différentes en fonction de la complexité choisie. Certaines méthodes sont plus complexes et meilleures mais plus longues à réaliser (au niveau du temps de calcul). Pour notre projet, nous avons besoin de faire du traitement d'images en temps réel donc nous devrons faire attention à la complexité de notre algorithme. Les différentes étapes de la détection de la pupille sont :

1. Filtrage du bruit (filtre médian...).
2. Prétraitement (égalisation histogramme + seuillage bas pour supprimer le reflet).
3. Extraction de contour (filtre variance, ...).
4. Recherche des cercles/ellipses (Hough).
5. Discrimination du cercle de la pupille (couleur, position, surface, ...).

Une première méthode de détection de la pupille, développée par Jérôme Schmaltz de l'Ecole de Technologie Supérieure de Montréal, repose sur le principe décrit figure 1.8.

## Suppression de surplus de contours, atténuation du bruit

Tout d'abord, il est nécessaire de réduire le bruit occasionné par la caméra, le bruit engendré par le transfert des données et leur compression. Différents filtres [1] peuvent être utilisés, même si leur application altérera les contours de l'image.

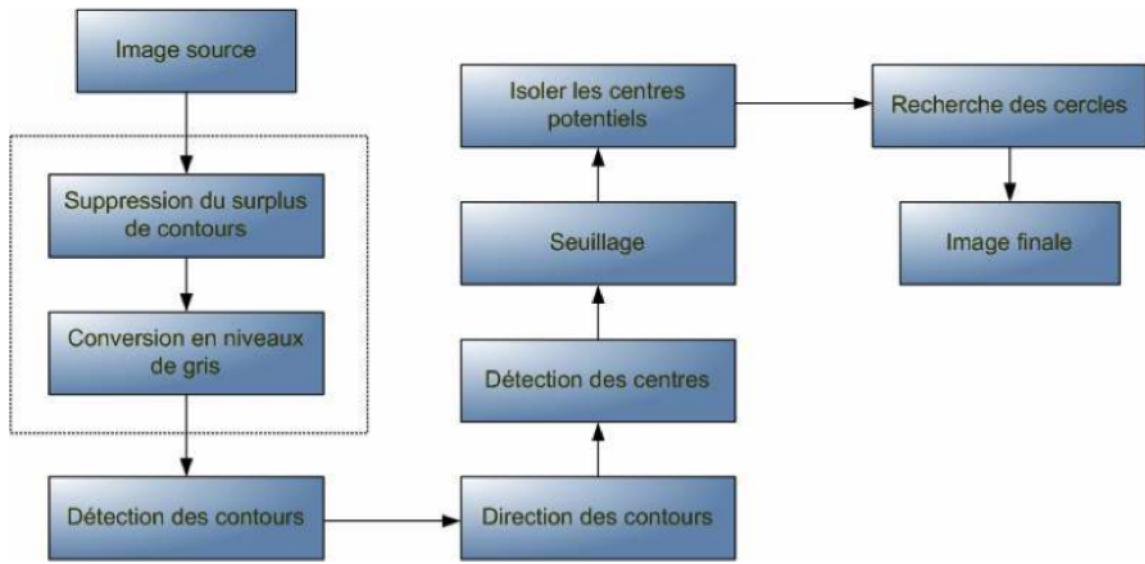


FIGURE 1.8 – Différentes étapes de traitement de l'image afin de détecter une pupille dans une image.

### Filtre Moyenneur

Ce filtre lisseur part du principe que la valeur d'un pixel est relativement similaire à son voisinage. Il fait donc en sorte que chaque pixel soit remplacé par la moyenne pondérée de ses voisins. Si on applique un filtre moyenneur de taille  $\lambda=3$ , cela signifie qu'on additionne la valeur de tous les pixels du voisinage du pixel traité. On obtient ainsi la matrice de convolution suivante :

$$h = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

h s'appelle le masque de convolution. La somme des coefficients du masque valant 1, le lissage préservera toute zone de l'image où le niveau de gris est constant. Ce filtre peut engendrer des phénomènes de fausses couleurs, contrairement au filtre médian, car son efficacité est moindre lorsque les objets présents dans l'image sont de faible dimension par rapport aux dégradations. Ce filtre est isotrope.

Une amélioration du filtre moyenneur consiste à jouer sur les valeurs des coefficients du masque :

$$h = \frac{1}{10} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

### Coupe Médiane

Une coupe médiane permet d'atténuer le bruit d'une image. Le principe est de prendre dans le voisinage la valeur la moins extrême. Pour cela, on crée une

liste des valeurs du voisinage, puis on trie cette liste et on prend la valeur qui se trouve au milieu de la liste. Cette valeur "médiane" est la plus éloignée des deux extrêmes.

## Diffusion

Le principe est d'atténuer les différences d'intensité entre le pixel central et ses voisins. Pour chaque voisin, on calcule la différence d'intensité avec le pixel central. Plus la différence est faible, plus elle est propagée vers le pixel central. Cela permet d'uniformiser les zones d'intensité proche et de conserver les forts contrastes (et donc les contours).

## Filtre Gaussien

Le filtre Gaussien est un filtre isotrope spécial avec des propriétés mathématiques bien précises.  $\sigma$  caractérise l'écart type soit la largeur du filtre : autrement dit la largeur du filtre en partant du point central est égale à  $3\sigma$  arrondi à l'entier supérieur. En deux dimensions, le filtre de Gauss est le produit de deux fonctions gaussiennes, une pour chaque direction :

$$g(x, y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}$$

L'effet de ce filtre sur l'image est assez similaire au filtre moyenieur, mais la moyenne est pondérée : les pixels près du centre ont un poids plus important que les autres. En général, un filtre Gaussien avec  $\sigma < 1$  est utilisé pour réduire le bruit, et si  $\sigma > 1$  c'est dans le but de flouter volontairement l'image. Il faut noter que plus  $\sigma$  est grand, plus la cloche Gaussienne est large, et plus le flou appliqué à l'image sera marqué.

### 1.1.5.3 Conversion en niveau de gris

La seconde étape du prétraitement consiste à convertir l'image RGB en niveau de gris. Cette conversion est nécessaire puisque les différents algorithmes se basent sur des images en niveau de gris. On pourrait d'abord penser que le niveau de gris se calcule comme la somme des 3 pixels divisée par 3 :  $Gris = \frac{R+G+B}{3}$

Cependant, d'après les recommandations de la commission internationale de l'éclairage, il devrait plutôt être de la forme :  $Gris = 0.299R + 0.587G + 0.114B$ . Nous pouvons voir figure 1.9 le rendu de cette conversion.

### 1.1.5.4 Détection des contours

#### Filtre de Canny

Il existe différentes méthodes de détection des contours. Des filtres peuvent être utilisés (voir paragraphe 1.1.5.2). Le choix du filtre doit prendre en compte le fait qu'une réduction trop importante du bruit masque certains contours, mais qu'une réduction trop faible peut laisser apparaître trop de contours non significatifs. Nous nous intéresserons ici au meilleur filtre pour la détection de contours : le

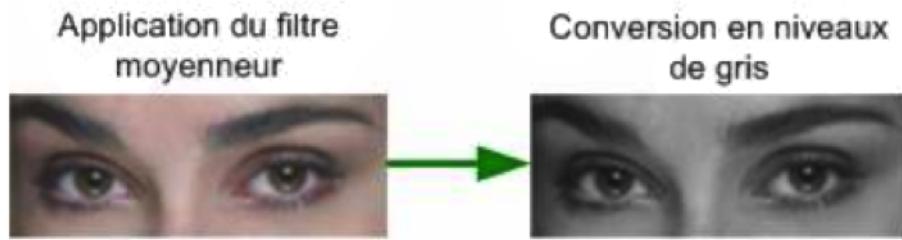


FIGURE 1.9 – Conversion en niveau de gris

filtre de Canny. Il est considéré comme tel car il offre un très bon compromis entre la réduction du bruit et la localisation des contours. Les différentes étapes du filtre de Canny sont :

- Réduction du bruit grâce à la convolution d'un filtre passe-bas Gaussien, ce qui permet de tamiser les contours.
- Eclaircissement des contours grâce à l'utilisation des valeurs des amplitudes des gradients ainsi que leurs directions. Pour cela, nous utilisons les formules (1.1) et (1.2).

$$g = \sqrt{g_x^2 + g_y^2} \quad (1.1)$$

$$\theta = \arctan \frac{g_y}{g_x} \quad (1.2)$$

Nous pouvons voir que l'amplitude est proportionnelle à  $g_x$  et  $g_y$ , ce qui montre qu'elle mesure la force d'un contour indépendamment de sa direction.

- Suppression des non maximaux, cela permet de réduire la largeur des arrêtes détectées à un pixel. La figure 1.10 décrit un algorithme permettant de supprimer ces non maximaux.

```

Soit  $g_s$  une image de mêmes dimensions que l'image source  $g$ .
Pour toutes les coordonnées des pixels  $x$  et  $y$ ,
    Approximer  $\theta$  ( $x, y$ ) par 0, 45, 90 ou 135 degrés.
    Si  $g_s(x, y) < g$  dans un voisinage de direction approximée ou que
         $g_s(x, y) < g$  dans un voisinage de direction  $\theta + 180$  alors
             $g_s(x, y) = 0$ 
    Sinon
         $g_s(x, y) = g(x, y)$ 
    Fin si
Fin pour

```

FIGURE 1.10 – Algorithme permettant de supprimer des non maximaux

- Localisation : elle repose sur l'identification des contours significatifs à partir des amplitudes des gradients précédemment calculés.

La méthode triviale consiste à appliquer un seuillage aux gradients calculés. Cependant, l'application d'un seuil trop faible implique la prise en compte de

tous les contours, y compris les contours non significatifs (prise en compte des gradients maximaux engendrés par le bruit). De plus, l'application d'un seuil trop élevé engendre une fragmentation des chaines de pixels qui forment des contours significatifs de l'image. Ainsi, le seuil par hystérésis offre une bonne solution à ce problème.

La figure 1.11 illustre l'idée de détection des contours grâce au filtre de Canny.

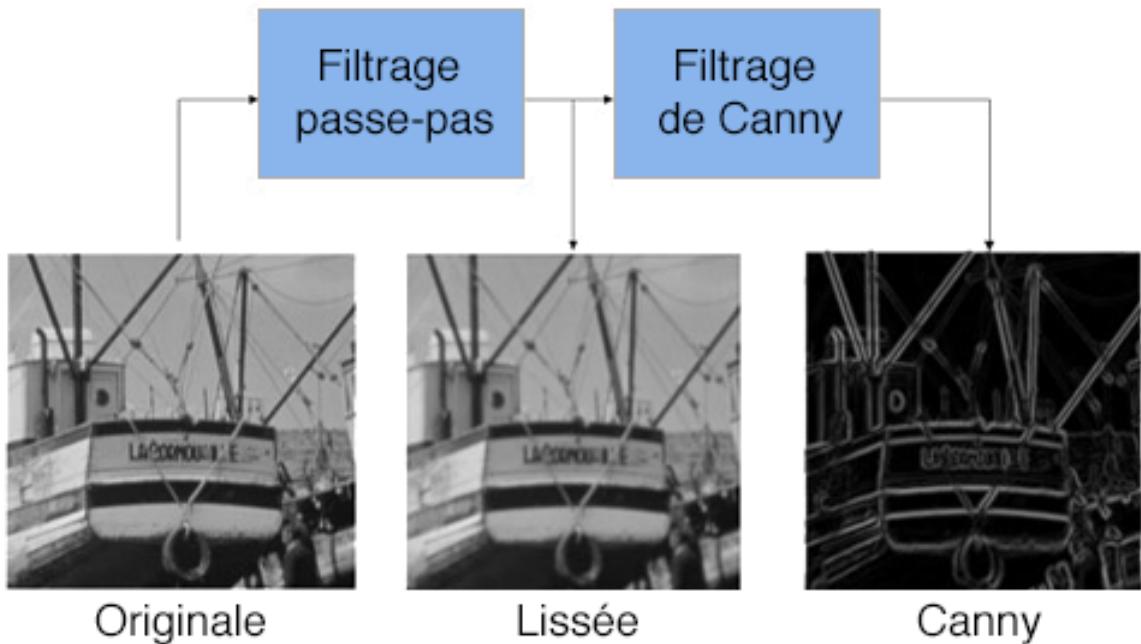


FIGURE 1.11 – Détection des contours grâce au filtre de Canny

### Filtre de Sobel

L'opérateur calcule le gradient de l'intensité de chaque pixel. Celui-ci indique la direction de la plus forte variation (du clair au sombre), ainsi que le taux de changement dans cette direction. On connaît alors les points de changement soudain de luminosité, correspondant probablement à des bords, ainsi que l'orientation de ces bords. L'opérateur utilise des matrices de convolution. La matrice (généralement de taille  $3 \times 3$ ) subit une convolution avec l'image pour calculer des approximations des dérivées horizontale et verticale. Soit  $A$  l'image source,  $G_x$  et  $G_y$  deux images qui en chaque point contiennent des approximations respectivement de la dérivée horizontale et verticale de chaque point. Ces images sont calculées comme suit :

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * A \text{ et } G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

En chaque point, les approximations des gradients horizontaux et verticaux peuvent être combinées comme suit pour obtenir une approximation de la norme du gradient :  $G = \sqrt{G_x^2 + G_y^2}$

On peut également calculer la direction du gradient comme suit :  $\Theta = \text{atan2}(G_y, G_x)$ , et créer une matrice des directions.

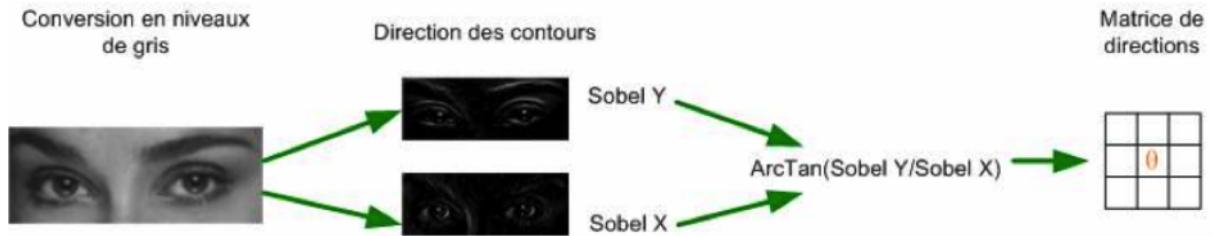


FIGURE 1.12 – Déterminer les directions des gradients à partir de l’application du filtre de Sobel

Enfin, d’autres filtres tels que le filtre de Kirsch, le filtre MDIF, le filtre de Prewitt ou encore celui de Roberts, permettent aussi la détection de contours, mais nous ne les détaillerons pas ici.

#### 1.1.5.5 Détection des centres

Le principe de la détection des centres se base sur les deux étapes précédentes. Pour chaque point des contours de l’image de Canny, on trace une droite en fonction de la direction calculée par le biais de l’application des filtres de Sobel. Ainsi, par l’application de la formule de la droite  $y = m \cdot x + b$ , on trace une droite dans l’accumulateur, matrice de même dimension que l’image traitée contenant des entiers, en fonction des points de contours de l’image de Canny tels qu’exposés dans la figure 1.13.

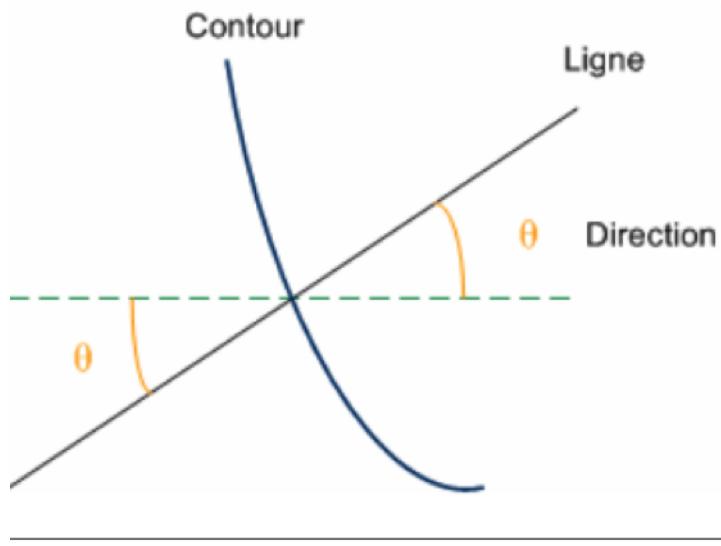


FIGURE 1.13 – Traçage d’une droite en fonction d’un contour

Tracer une droite dans un accumulateur revient à incrémenter chaque cellule d’une matrice aux endroits où elle passe (voir figure 1.14).

Ainsi, suivant ce principe, les valeurs maximales de la matrice indiqueront les différents centres potentiels. La figure 1.15 synthétise la méthode énoncée.

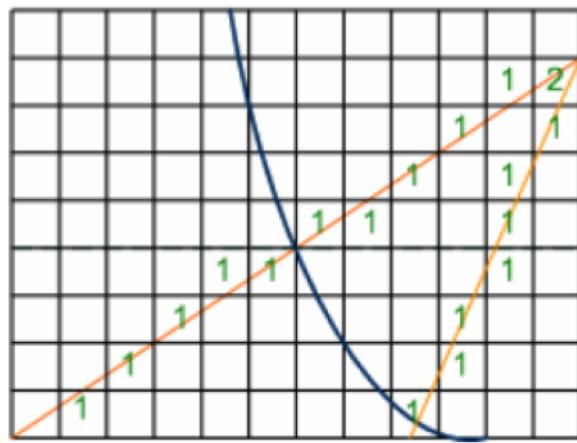


FIGURE 1.14 – Traçage d'une droite dans l'accumulateur

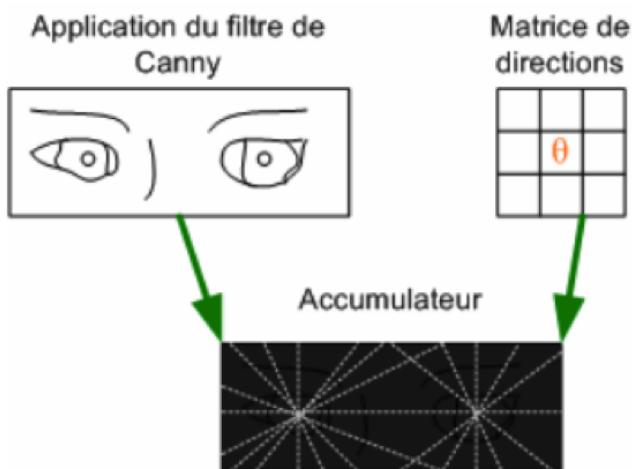


FIGURE 1.15 – Traçage des lignes dans l'accumulateur en fonction de l'image de Canny et de la matrice de directions

### 1.1.5.6 Seuillage

Le seuillage permet de désencombrer la matrice accumulateur des informations superflues. Nous recherchons les centres potentiels et ainsi nous n'avons pas besoin de toutes les valeurs de l'accumulateur. En effet, il suffit d'appliquer un seuil et de garder les pixels ayant une valeur supérieure à celui-ci (les autres pixels seront mis à zéro) car ils représenteront les différents centres potentiels.

Pour ce faire, nous pouvons par exemple utiliser un seuil du type :

valeur pixel actuel  $> 0.5 \cdot \max(\text{valeur pixel image})$ , qui permet de garder les pixels ayant au moins la moitié de la valeur maximale d'une cellule de l'accumulateur.

Nous pouvons voir figure 1.16 le résultat du seuillage.

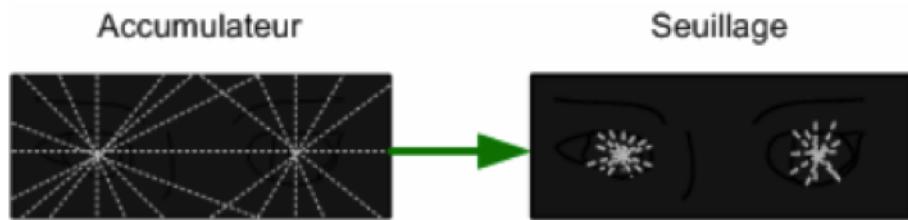


FIGURE 1.16 – Application d'un seuil dans l'accumulateur

#### 1.1.5.7 Isoler les centres potentiels

L'application d'une convolution d'un chapeau mexicain (voir formule (1.3) et figure 1.17) sur l'accumulateur permet d'isoler les points les plus au centre. Un chapeau mexicain est la combinaison d'un filtre Gaussien (voir paragraphe 1.1.5.2) avec un filtre Laplacien.

$$\nabla^2 G_\sigma(r) = \frac{-1}{2\pi\sigma^4} \left(2 - \frac{r^2}{\sigma^2}\right) e^{-\frac{r^2}{2\sigma^2}} \quad (1.3)$$

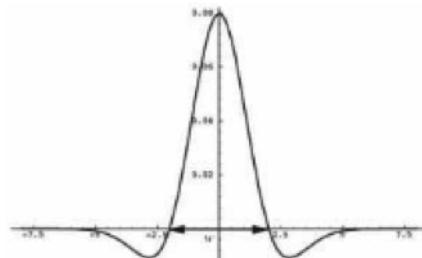


FIGURE 1.17 – Représentation graphique d'un Laplacien de Gaussien

Ainsi, l'application d'un chapeau mexicain sur l'accumulateur permet de conserver les points les plus à même d'être les centres de cercles potentiels en décrémentant les valeurs des cellules entourant les centres. La figure 1.18 présente le résultat de cet isolement des centres.

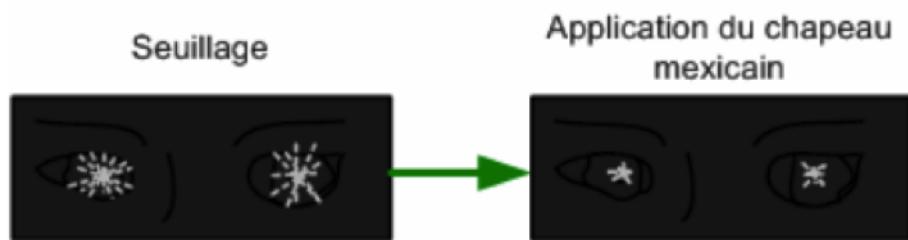


FIGURE 1.18 – Application d'un chapeau mexicain dans l'accumulateur

#### 1.1.5.8 Recherche des cercles

La recherche des cercles s'appuie sur la création d'accumulateurs et la transformée de Hough. En effet, pour un cercle de rayon inconnu, nous utilisons des accumulateurs en 3 dimensions : les deux premières servent à ranger les coordonnées  $x$  et  $y$  du centre du cercle et la troisième permet de ranger le rayon  $r$  du cercle.

1. On initialise les accumulateurs.
2. A partir de l'image des contours de Canny, on trace littéralement des cercles dans un des accumulateurs en faisant coïncider les centres des cercles avec les coordonnées du pixel de contour pour un rayon donné. La méthode de Bresenham pourra être employée afin de tracer plus efficacement les cercles dans l'accumulateur de Hough.
3. Comme on peut le voir figure 1.19, plusieurs cercles sont tracés (dans un accumulateur de rayon  $r$ ) en suivant les contours de l'image de Canny. Nous pouvons voir que les cellules contenant les valeurs maximales coïncident avec des centres de cercles potentiels.

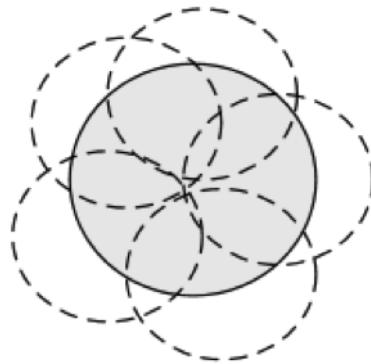


FIGURE 1.19 – Traçage des cercles avec l'image des contours

4. Ensuite, on peut diviser la zone d'intérêt de l'image en deux parties : partie droite et partie gauche du visage. Pour chacune de ces parties, on applique un seuil aux accumulateurs d'Hough permettant de conserver les centres, les coordonnées et rayons de cercles potentiels et on vérifie qu'ils font bien partie de l'accumulateur des cercles potentiels de l'étape de l'isolement des centres potentiels.
5. Pour finir, une moyenne des positions et des rayons des cercles restants permet une identification des cercles entourant l'iris et la pupille (à gauche et à droite). Il ne reste plus qu'à conserver le cercle ayant le rayon le plus petit, correspondant à la pupille.

La figure 1.20 résume cette démarche.

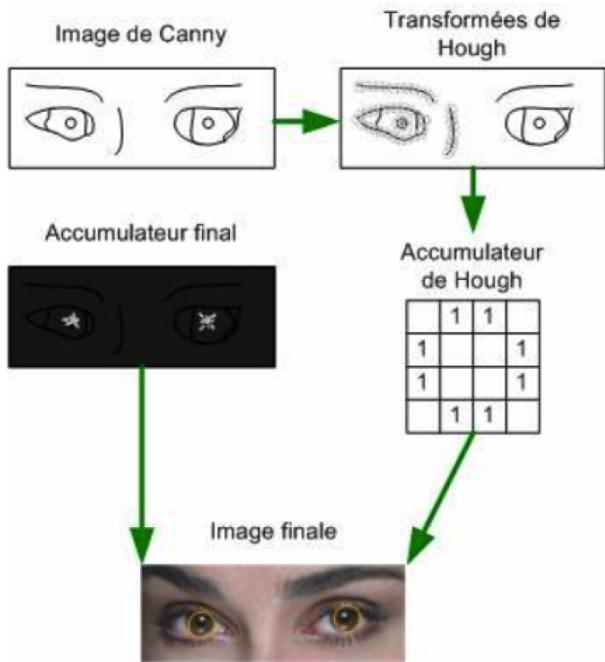


FIGURE 1.20 – Les différentes opérations permettant de localiser les pupilles

## 1.2 Méthode de détection de la direction du regard

La direction du regard d'une personne peut être obtenue à l'aide de deux paramètres : L'orientation du visage et l'orientation oculaire. La détection du visage permet de connaître l'orientation globale de l'utilisateur et de vérifier que ses deux yeux sont bien en face de la caméra. Suivant l'orientation de la tête de l'utilisateur, une approximation de la direction du regard peut aussi être réalisée. La détection effectuée à l'aide de l'orientation oculaire approche la direction du regard à l'aide de propriétés géométriques et physiques de l'iris et de la pupille. Le problème principal réside dans le fait que cette étude est une étude locale, c'est-à-dire que la caméra qui filme l'œil de l'utilisateur doit avoir un fort niveau de zoom, obligeant le sujet à rester immobile afin de ne pas sortir du champ de la caméra.

Les reflets détectés dans l'œil du sujet par la détection de pupille avec infrarouge (voir paragraphe 1.1.4) vont permettre de déterminer la direction du regard de l'utilisateur. En supposant la tête de l'utilisateur immobile, et en prenant comme référence le reflet formé par la lumière infrarouge sur la cornée, on peut déterminer la direction du regard en se servant du vecteur créé entre le reflet et le centre de la pupille (ou l'iris) (voir figure ci-dessous). Cependant cette méthode demande certains prérequis et notamment une calibration et ce avec chaque nouvel utilisateur. Pour effectuer cette calibration, il est demandé au sujet de fixer quatre points (minimum), situés sur l'écran, afin d'obtenir des valeurs étalons (souvent les coins de l'écran), qui permettront de déduire la direction du

regard par la suite.

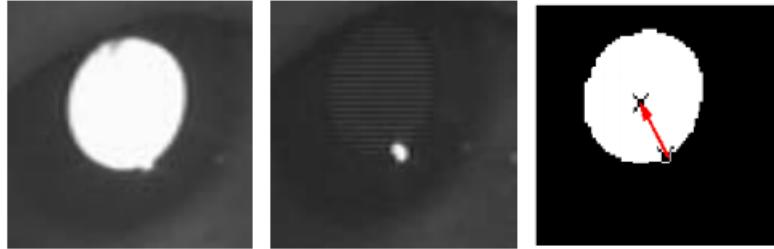


FIGURE 1.21 – Tracé du vecteur entre la réflexion infrarouge dans la cornée et la pupille

Cette méthode présente tout de même deux défauts majeurs : l'impossibilité pour l'utilisateur de bouger la tête ou le corps, sous risque de devoir recommencer la calibration, et l'obligation de re-calibrer le système à chaque changement d'utilisateur.

## 1.3 Présentation des technologies existantes

Le suivi de l'œil (aussi appelé « eye tracking ») a de nombreuses applications, tels que le contrôle d'un système. Une pluralité de dispositifs a déjà été développée dans ce domaine et nous allons tâcher dans cette partie de vous présenter les procédés majeurs. Deux grandes catégories de systèmes existent : avec ou sans contact. Un système est dit sans contact lorsque celui-ci n'est pas attaché ou relié à l'utilisateur. Il a l'avantage d'être plus agréable est facile à utiliser. Le deuxième, avec contact, est monté sur l'usager (comme des lunettes par exemple). Il offre ainsi une plus grande mobilité. Dans notre cas, les deux méthodes sont à envisager.

### 1.3.1 Systèmes avec contact

#### 1.3.1.1 Les Tobii Glasses

Les Tobii Glasses [15] sont des lunettes capables de filmer et d'enregistrer le mouvement des yeux. Elles peuvent ainsi savoir en temps réel ce que l'utilisateur fixe et voit. Ces lunettes sont équipées de (cf. figure 1.22) :

- caméras qui filment directement l'œil
- une caméra qui filme ce que voit l'utilisateur
- plusieurs illuminateurs permettant d'éclairer l'œil
- un capteur infrarouge

Le capteur infrarouge permet au système de connaître la position de la tête de l'utilisateur dans l'espace à l'aide d'émetteurs infrarouges balisant la zone qu'il regarde (cf. figure 1.23).

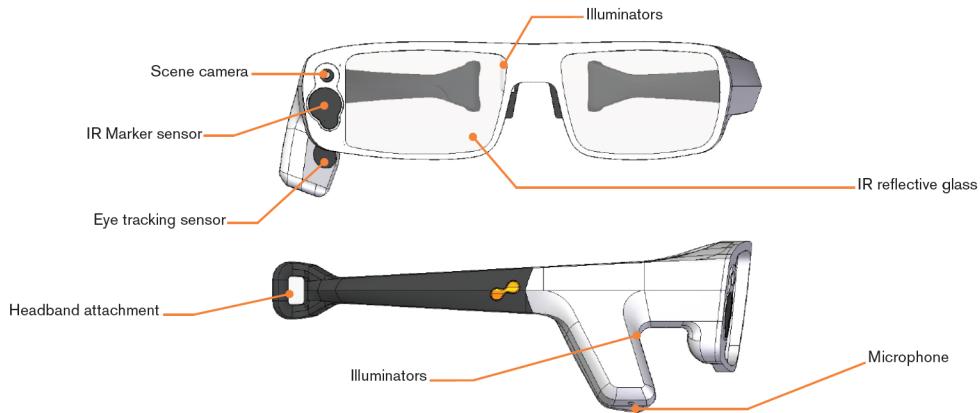


FIGURE 1.22 – Tobii Glasses

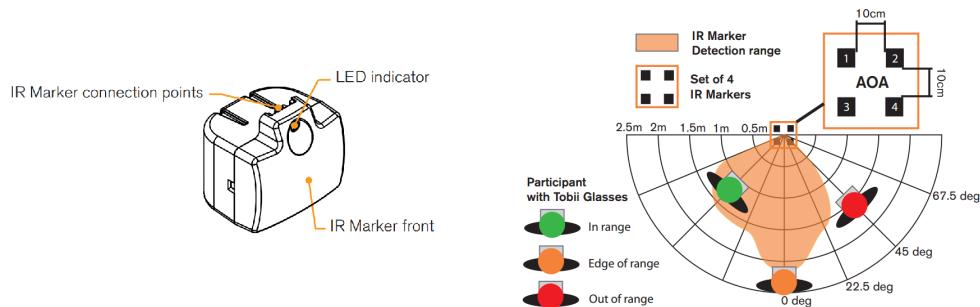


FIGURE 1.23 – Emetteur

Toutes les données sont enregistrées dans un boîtier relié aux lunettes (cf. figure 1.24). Elles peuvent ensuite être récupérées, traitées et analysées sur un ordinateur. La calibration est aussi effectuée via le boîtier.

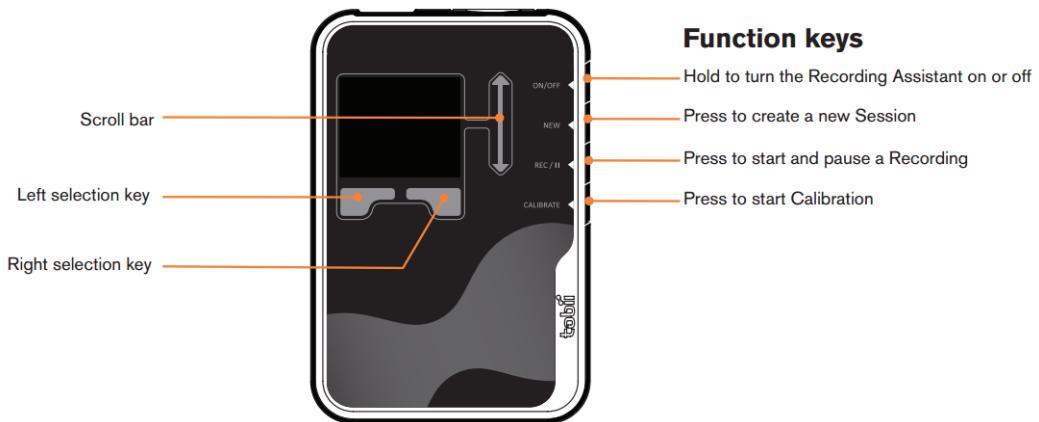


FIGURE 1.24 – Boîtier de contrôle et d'enregistrement

D'un point de vue technique ces lunettes utilisent le Pupil Centre Corneal Reflection (PCCR, cf. figure 1.25). Cette méthode consiste à illuminer l'œil, créant ainsi un reflet sur la pupille et la cornée. Une caméra récupère ensuite une image

de cette réflexion. Le vecteur formé par l'angle entre la cornée et le reflet lumineux sur la pupille est ensuite calculé. La direction correspond alors à la direction du regard de l'utilisateur. Malheureusement l'algorithme de calcul n'est pas donné.

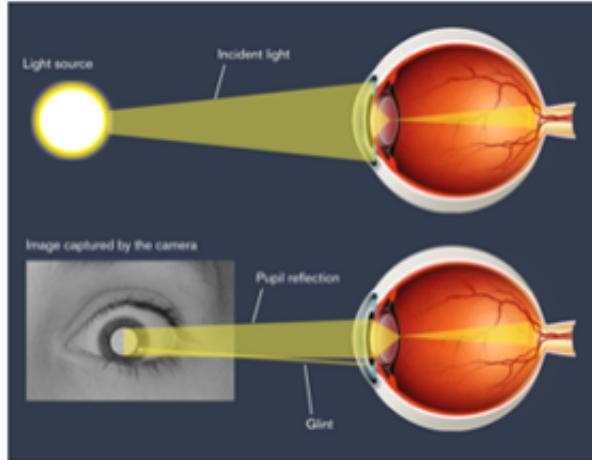


FIGURE 1.25 – Pupil Centre Corneal Reflection PCCR

Le prix n'est pas communiqué sur le site internet, mais il semblerait qu'il soit d'environ 10 000 €.

### 1.3.1.2 PUPIL



FIGURE 1.26 – PUPIL

PUPIL [6] est un projet développé par trois étudiants du MIT. Tout comme les Tobbi Glasses il peut capter et enregistrer les mouvements effectués par l'œil de l'utilisateur.

Le principe de fonctionnement est basé sur deux caméras. La première permet d'enregistrer les mouvements de l'œil, et donc de retrouver par la suite la position de la pupille. La seconde filme ce qu'est censé voir l'utilisateur et permet de connaître la direction du regard de l'utilisateur en utilisant les données de la première caméra. Deux notions sont donc à distinguer : la position de la pupille et

la direction du regard. La première nous aide à déduire la seconde. L'avantage de cette méthode est l'usage d'une caméra classique afin de détecter la direction du regard. Il n'est ici pas obligatoire d'employer la réflexion d'une lumière infra-rouge dans l'œil de l'utilisateur.

D'un point de vue matériel, les deux caméras utilisées sont très différentes. Celle enregistrant le mouvement des yeux est une caméra basse résolution ( $640 \times 480$  à 30 fps) avec un filtre infrarouge (la Microsoft LifeCam HD-6000 est recommandée). La caméra qui filme le monde alentour possède quant à elle une grande résolution ( $1920 \times 1080$ , la Logitech HD 1080p Webcams est recommandée). PU-PIL peut être acheté pour la somme de 380 €.

### 1.3.2 Systèmes sans contact

#### 1.3.2.1 Tobii X2-30&60



FIGURE 1.27 – Tobii X2-30&60

Autre produit créé par Tobii, le X2 [14] rentre dans les systèmes d'oculométrie sans contact. Ne mesurant que 184 mm, il se branche directement en USB. Il est très facile d'utilisation et s'adapte à de nombreux supports (ordinateurs portables, tablettes, télévisions...). Cet outil ne permet pas de contrôler le système sur lequel il est branché mais d'enregistrer ce que l'utilisateur regarde afin d'obtenir des statistiques. La détection de la direction du regard se fait aussi à l'aide de la réflexion causée par des LEDs infrarouges sur la cornée du sujet. Combiné avec la localisation de la pupille, le système peut en déduire la direction du regard de l'utilisateur. Encore une fois l'algorithme n'est malheureusement jamais décrit. Cependant, l'appareil à une précision d'environ  $0.34^\circ$  sur la direction du regard du sujet, la performance usuelle des technologies d'Eye Tracking variant entre  $0.01$  et  $1^\circ$ .

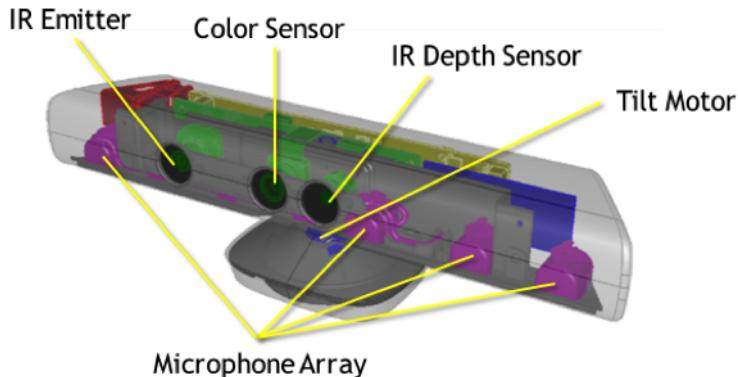


FIGURE 1.28 – EyeCharm

### 1.3.2.2 Eye Charm

Créé par une société allemande (4tiito), EyeCharm [19] est un adaptateur qui se clipse sur la Kinect et exploite sa caméra infrarouge pour suivre le mouvement des yeux. Un logiciel compatible avec Windows 7 & 8 (NUIA) a été développé pour contrôler les principales fonctions d'un ordinateur grâce à ce système, comme le navigateur internet, les jeux vidéo et d'autres applications. L'utilisation d'EyeCharm ne nécessite aucun changement dans le code source des applications. L'algorithme de suivi traite des images. La puissance de calcul nécessaire à son utilisation est ainsi assez importante (exemple : « son algorithme consomme 5 % de la puissance d'un processeur Intel Core i5-3470 cadencé à 3,2 GHz »). Il est recommandé de détenir « un pc équipé d'un processeur AMD ou Intel multicœur et avec au moins 2 Go de mémoire vive ». Le rôle d'EyeCharm est de projeter une lumière infrarouge sur le visage de l'utilisateur. Celle-ci est captée par la caméra de la Kinect (pour Xbox ou Windows, connectée en USB 2.0), ce qui lui permet de suivre le mouvement des yeux. Il est conseillé de se tenir à 75 cm de l'écran pour obtenir de bons résultats.

Le logiciel compte plusieurs extensions pour étendre les possibilités de contrôle par les yeux à :

- plusieurs navigateurs internet (Chrome, Internet Explorer, Firefox)
- Adobe Photoshop
- la suite Office
- les jeux World of Warcraft et Minecraft

Certaines fonctionnalités, comme le zoom ou le retour à la page précédente peuvent nécessiter une action supplémentaire, non réalisée par les yeux. Pour cela, il est possible d'appuyer sur une touche du clavier ou d'utiliser une commande vocale, prise en charge par la Kinect. Un kit de développement a été prévu pour que les utilisateurs puissent développer eux-mêmes des applications à l'aide de Qt Creator, Visual Studio, et les langages C, C++, C# et Java.

### 1.3.2.3 Robust Eye and Pupil Detection Method for Gaze Tracking [4]

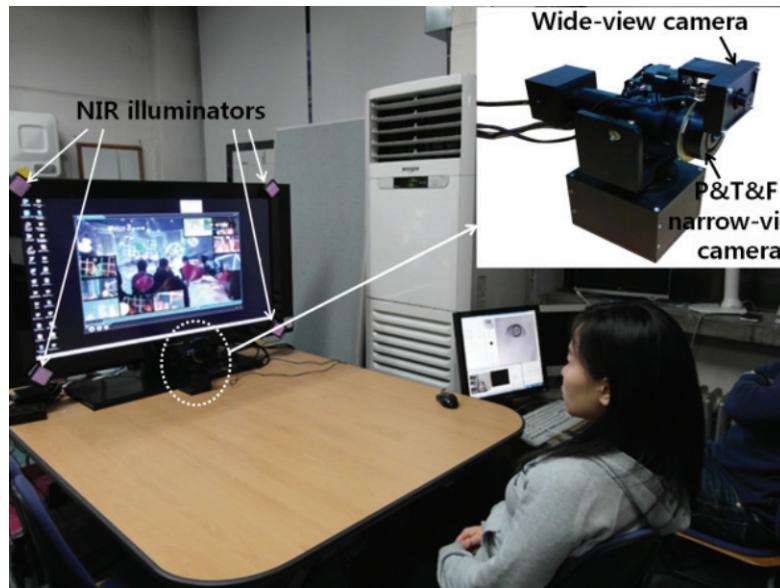


FIGURE 1.29 – Robust Eye and Pupil Detection Method for Gaze Tracking

Ce système est intéressant car il n'est pas directement embarqué sur l'utilisateur. La détection des yeux se fait à l'aide de deux caméras. Ces dernières sont motorisées et peuvent donc tourner sur elle-même ou s'orienter vers le haut ou le bas. Une caméra dite wide-view filme l'utilisateur dans son intégralité. Dans les faits cette caméra permet de repérer son visage, puis la position de son œil grâce à deux algorithmes de reconnaissance faciale (l'Adaboost et le CAMShift). Une fois l'œil détecté, sa position est transmise à la seconde caméra dite narrow-view (d'une résolution de 1600x1200 réduite à du 240x320 pour améliorer le temps de calcul). Elle va pouvoir zoomer sur l'œil et ainsi obtenir un gros plan, tandis que la première ne sert qu'à repérer l'usager. Une fois l'œil dans le champ de vision de la caméra, la direction de celui-ci est calculée à l'aide du centre de la pupille et des réflexions spéculaires créées sur l'œil par quatre « near-infrared illuminators », eux même placés aux quatre coins de l'écran que regarde l'utilisateur. Ici encore l'algorithme de calcul reste flou.

D'un point de vue logiciel, cette méthode emploie du C++ et la librairie OpenCV. Tous les calculs sont effectués directement sur un ordinateur classique équipé d'un processeur Intel Core 2 Quad 2.3 GHz et de 4 GB. N'étant qu'un sujet de thèse, ce montage n'est pas en vente.

## 1.4 Solutions techniques et méthodes envisagées

### 1.4.1 Au préalable, suite à l'état de l'art

Suite à l'état de l'art, la méthode d'eye tracking sans contact a été retenue (pas de système embarqué). Elle est similaire à celle de la partie 1.3.2.3 et semble

plus facilement réalisable qu'un système utilisant des lunettes avec caméras embarquées. Ainsi, dans un premier temps, cette technique sera développée afin d'obtenir un système qui fonctionne correctement. Puis, le système pourra être perfectionné et, s'il reste du temps sur les délais fixés, la méthode utilisant des lunettes sera approfondie.

Pour l'approche sans contact, deux caméras seront utilisées. La première, la caméra dite "grand angle", permettra de filmer la scène dans son ensemble et de localiser le visage ainsi que certains de ses éléments morphologiques tels que les oreilles et la bouche. Elle transmettra ensuite les coordonnées spatiales du visage à la seconde caméra "petit angle". Cette dernière zoomera sur le visage (grâce aux coordonnées fournies) et effectuera un tracking des pupilles.

La détection de la pupille s'effectuera à l'aide de la méthode de la bright et de la black pupille. Les différentes étapes vues dans l'état de l'art seront respectées et des tests seront menés afin de déterminer la meilleure solution pour réaliser chaque étape.

Le calcul de la direction du regard s'appuiera sur la méthode vue dans la partie [1.2](#). Des LEDs infrarouges, placées devant l'utilisateur, illumineront sa cornée, créant une réflexion spéculaire exploitable pour estimer la direction du regard. La caméra "petit angle" devra ainsi être munie d'un filtre infrarouge. Ce filtre pourra être réalisé à l'aide d'une pellicule photo placée devant l'objectif de la caméra.

#### 1.4.2 Méthodes choisies lors de la réalisation

Devant notre effectif réduit lors de la deuxième phase du projet, nous avons décidé de nous concentrer sur l'utilisation d'une seule caméra combinant "grand angle" et "petit angle". En effet, cela nous a permis d'éliminer tout problème de calibrage entre les deux caméras. De plus, nous avons constaté que la résolution de notre webcam était suffisamment élevée pour la détection du centre des pupilles. Ainsi, puisque l'infrarouge, qui permet certes une détection plus précise, est inconfortable et moins robuste à la lumière du jour, nous avons décidé d'abandonner cette technologie dans un premier temps.

# Partie 2

## Dossier fonctionnel

### 2.1 Ingénierie des exigences

#### 2.1.1 Approche Top-Down

Pour notre approche Top-Down, nous sommes revenus à la demande initiale du projet : remplacer la souris d'un ordinateur par le mouvement oculaire de l'utilisateur. La fonction principale du système est donc apparue clairement : permettre à l'utilisateur d'interagir avec une interface via ses yeux.

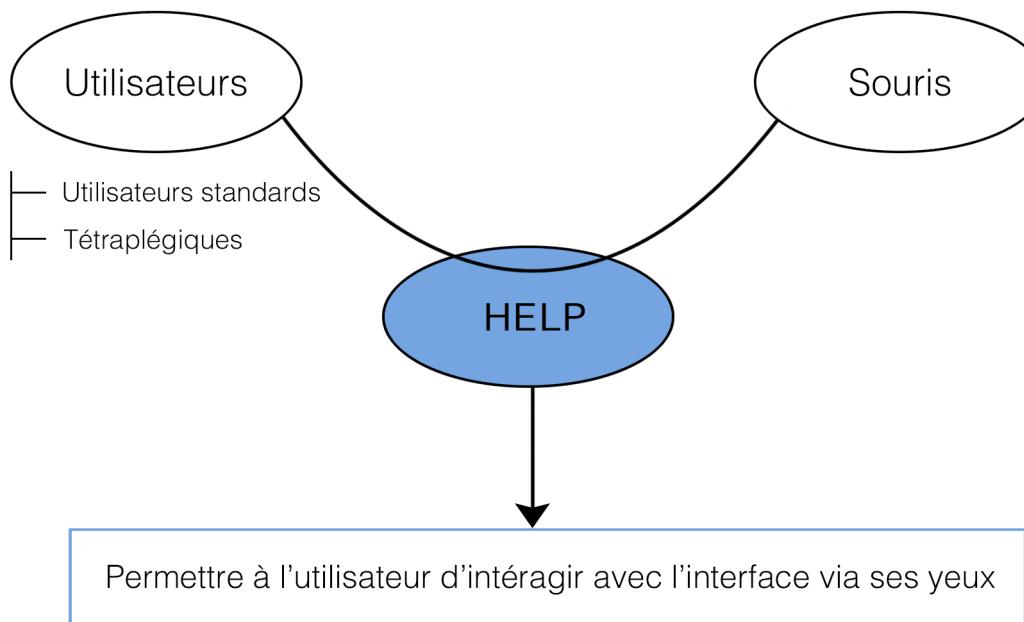
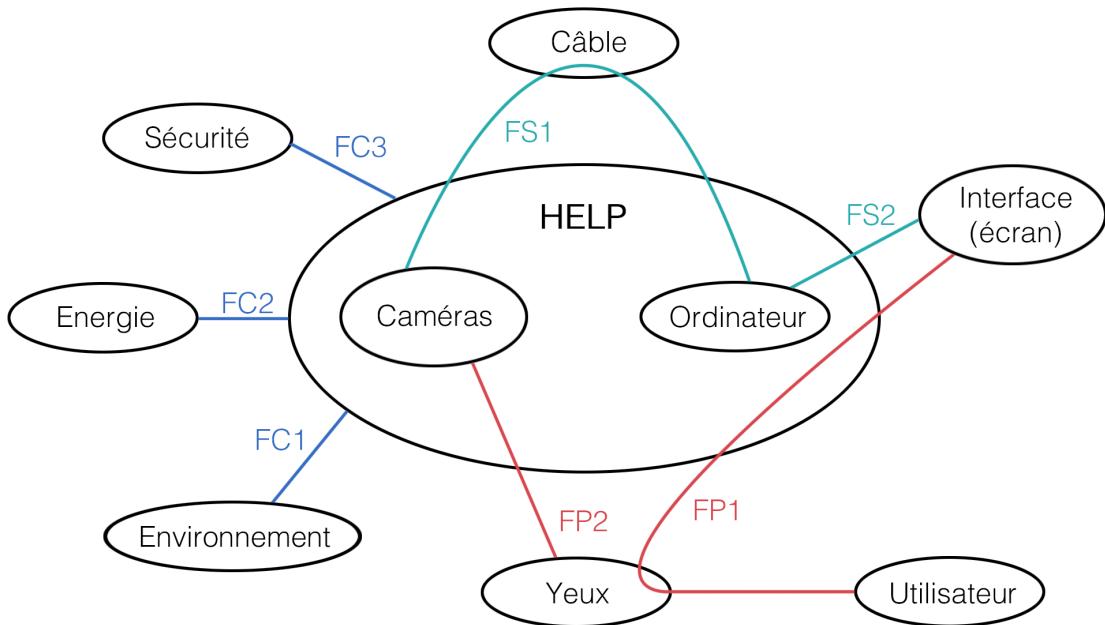


FIGURE 2.1 – Bête à cornes

Dans la suite du dossier fonctionnel, nous avons supposé que nous utilisions deux caméras. Dans les faits, comme nous l'avons précisé dans les méthodes de réalisation choisies (paragraphe 1.4.2), nous n'avons pas eu le temps de configurer les deux webcams. Le système a toutefois gardé la même démarche, une caméra

réalisant à la fois la détection du visage et du centre des pupilles.

D'autre part, dans le diagramme pieuvre (figure 2.2), nous avons défini la fonction de contrainte sur la sécurité de l'utilisateur. En effet, ce critère est important, notamment dans le cas d'essais cliniques sur des personnes tétraplégiques. Notre système n'étant pas encore au point, nous n'avons pas eu l'occasion de nous mettre en relation avec de tels utilisateurs. Néanmoins, nous pensons que les autorités réglementaires concernées valideraient les tests. N'utilisant pas d'infrarouge, le système, composé d'un ordinateur et d'une ou deux caméras nous semble fiable.



#### FONCTIONS PRINCIPALES

**FP1** : Permettre à l'utilisateur d'intégrer avec l'interface via ses yeux

**FP2** : Repérer et filmer les yeux de l'utilisateur

#### FONCTIONS CONTRAINTES

**FC1** : Résister à l'environnement

**FC2** : S'adapter à l'alimentation

**FC3** : Ne pas porter atteinte à la sécurité de l'utilisateur

#### FONCTIONS DE SERVICE

**FS1** : Transmettre des informations à l'ordinateur

**FS2** : Proposer une IHM adaptée

FIGURE 2.2 – Diagramme pieuvre

### 2.1.2 Approche Bottom-Up

Nous avons défini les différentes exigences auxquelles le système devra répondre, indépendamment de nos choix de réalisation technique. Nous les avons alors rassemblées par groupement logique, ce qui nous permettra de définir nos fonctions principales. Les lignes en ■ sont uniquement valables pour la méthode

embarquant un système sur l'utilisateur. Elles sont donc à ignorer dans un premier temps puisque nous avons choisi une approche différente.

<b>Id</b>	<b>Type</b>	<b>Expression</b>	<b>Criticité</b>	<b>Flexibilité</b>	<b>Expression de la fonction</b>
1.1.1	Service	Positionner l'utilisateur pour l'acquisition vidéo	5	5	Acquérir les données nécessaires à la détection du mouvement de l'oeil
1.1.2	Service	Filmer le visage de l'utilisateur	5	4	
1.1.3	Service	Acquérir les données nécessaires à l'estimation de la distance de l'utilisateur à l'interface	5	5	
1.1.4	Contrainte	Etre utilisable à une distance de 50cm à 3m	5	3	
1.1.5	Contrainte	Avoir une autonomie de 3h	4	3	
1.2.1	Service	Relayer les données à la carte de traitement	5	5	Transférer les données enregistrées
1.2.2	Service	Transmettre le signal compressé	5	5	
1.2.3	Contrainte	Effectuer la transmission sans fil	3	3	
1.3.1	Service	Déetecter l'utilisateur et ses mouvements :	5	5	Extraire les repères anatomiques
1.3.1.1	Service	Déetecter le visage de l'utilisateur	5	5	
1.3.1.2	Service	Déetecter les yeux de l'utilisateur	5	5	
1.3.1.3	Service	DéTECTER les pupilles de l'utilisateur	5	5	
1.3.1.4	Service	DéTECTER le mouvement des pupilles de l'utilisateur	5	5	
1.3.1.5	Service	DéTECTER le clignement des yeux de l'utilisateur	4	3	
1.3.2	Service	Calculer la distance entre l'utilisateur et l'interface	5	5	
1.4.1	Service	Analyser le mouvement de la pupille à l'aide de la distance utilisateur-IHM	5	5	Interpréter les données pour déduire l'action à exécuter
1.4.2	Service	Déduire l'action à effectuer	5	5	
1.5.1	Service	Afficher le curseur à l'endroit regardé par l'utilisateur	5	4	Agir sur l'IHM
1.5.2	Contrainte	Actualiser l'affichage en moins de 10ms	5	2	
1.5.3	Service	Permettre le clic gauche / la sélection	5	3	
1.5.4	Service	Permettre à l'utilisateur de désactiver (mettre en pause) le système de détection	3	2	
1.6.1	Service	Permettre à l'utilisateur de paramétrier l'IHM	3	3	Rendre l'IHM ergonomique
1.6.2	Contrainte	Rendre l'IHM adaptée au contrôle via le gaze-tracking	5	3	

FIGURE 2.3 – Cahier des exigences (les critères de criticité et de flexibilité sont sur 5)

### 2.1.3 Fonctions principales du système

Dans le cadre de ce projet, nous cherchons donc à remplacer la souris d'un ordinateur par un système qui suit le mouvement des yeux de l'utilisateur. Pour ce faire, nous avons mis en évidence des groupements logiques d'exigence. Tout d'abord, le système doit acquérir les données nécessaires à la détection du mouvement de l'œil. Ces données devront alors être traitées par l'ordinateur. Ce dernier doit alors interpréter ces données pour en déduire l'action à exécuter. De ces nouvelles informations, l'ordinateur doit pouvoir effectuer l'action que l'utilisateur veut effectuer sur l'IHM, la mettre en place, et montrer que ces modifications ont été exécutées.

- FP1 : Acquérir les données nécessaire à la détection du mouvement de l'œil
- FP2 : Transférer les données enregistrées
- FP3 : Extraire les repères anatomiques
- FP4 : Interpréter les données pour déduire l'action à exécuter
- FP5 : Agir sur l'IHM
- FP6 : Rendre l'IHM ergonomique

## 2.2 Spécification fonctionnelle 3 axes

### 2.2.1 Raffinement FAST

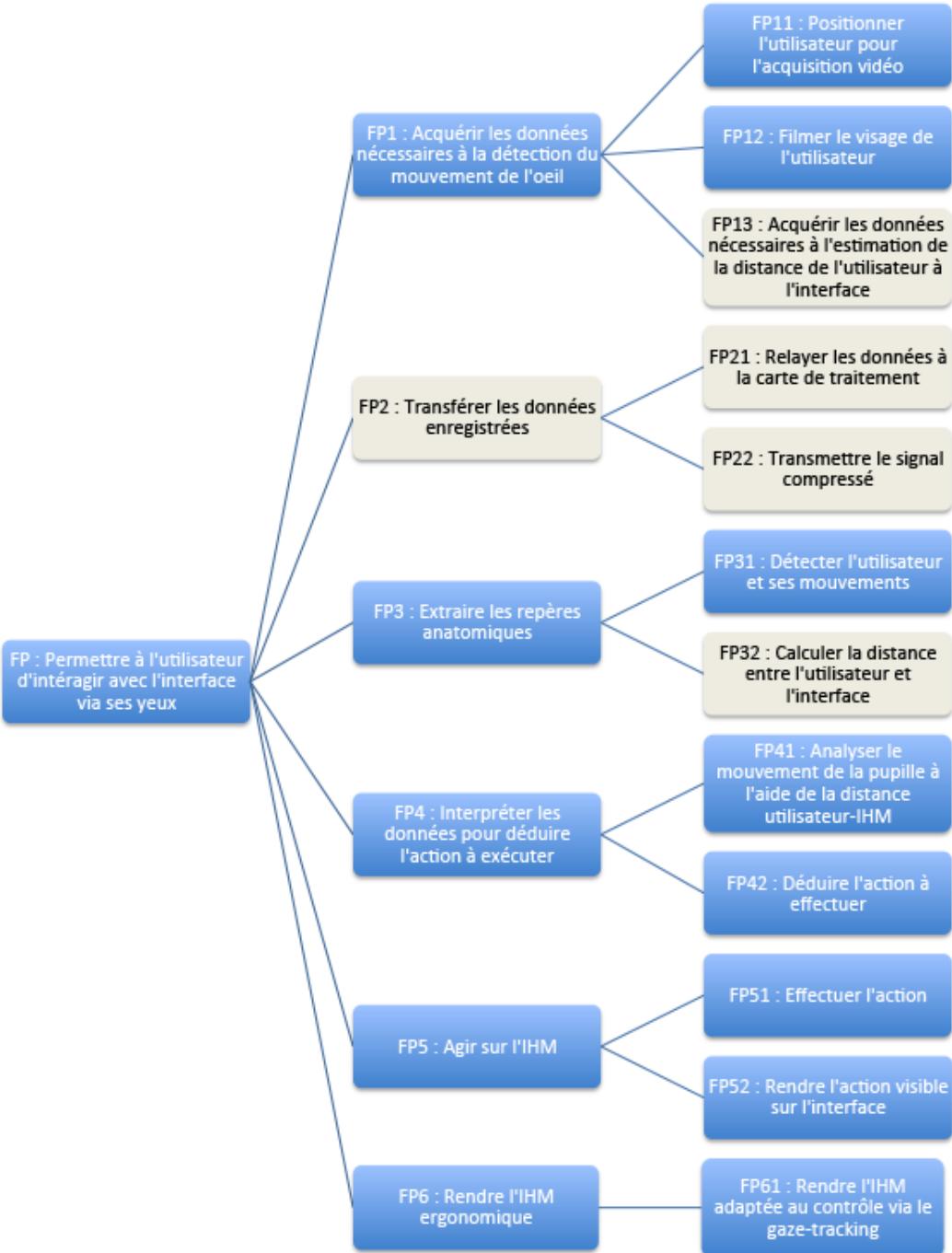


FIGURE 2.4 – FAST raffiné

Le cahier des exigences précédent nous a permis de définir les fonctions principales du système. Nous avons alors cherché à les raffiner par la méthode FAST (cf. figure 2.4) pour obtenir des fonctions plus précises auxquelles nous pourrons

apporter des solutions techniques propres à chacune. Nous pouvons ainsi entrevoir l'architecture fonctionnelle.

### 2.2.2 Spécification des données

La spécification du flux de données va permettre de comprendre les interactions et les échanges entre les différentes parties de notre système. Spécifier ainsi les données nous a permis de mieux comprendre les interactions entre chaque sous-partie du système. Il est ainsi clair que les données que nous allons principalement traiter et transférer seront des flux vidéos et qu'il sera donc important d'optimiser au mieux la rapidité de ces échanges. Au travers de la figure 2.5, nous pouvons aussi voir l'ordre avec lequel les sous-systèmes vont rentrer en jeu. Ainsi la Camera Narrow View sera dépendante du flux envoyé par la Camera Wide View. Un parallélisme de traitement des données est tout de même envisageable sur l'ordinateur pour gérer le flux vidéo de deux caméras.

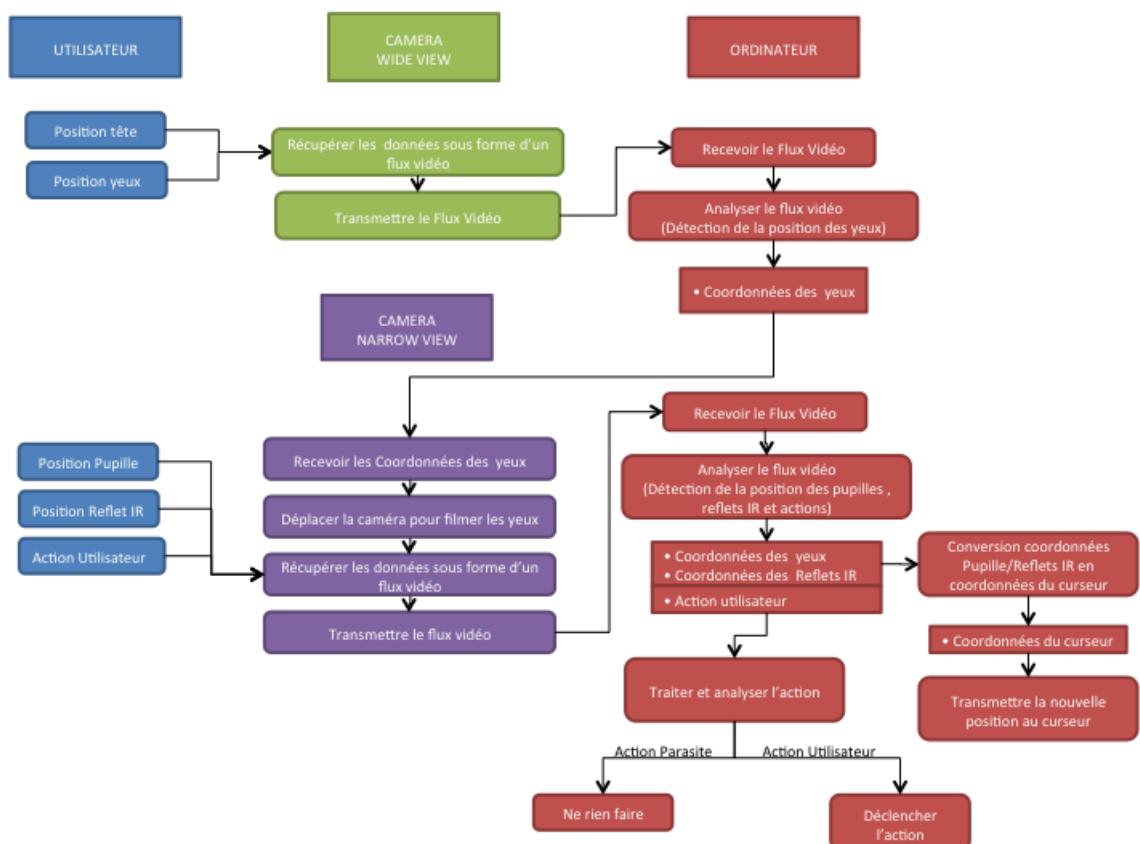


FIGURE 2.5 – Flux de données

### 2.2.3 Spécification des comportements

Le diagramme de séquence de la figure 2.6 illustre le comportement global du système.

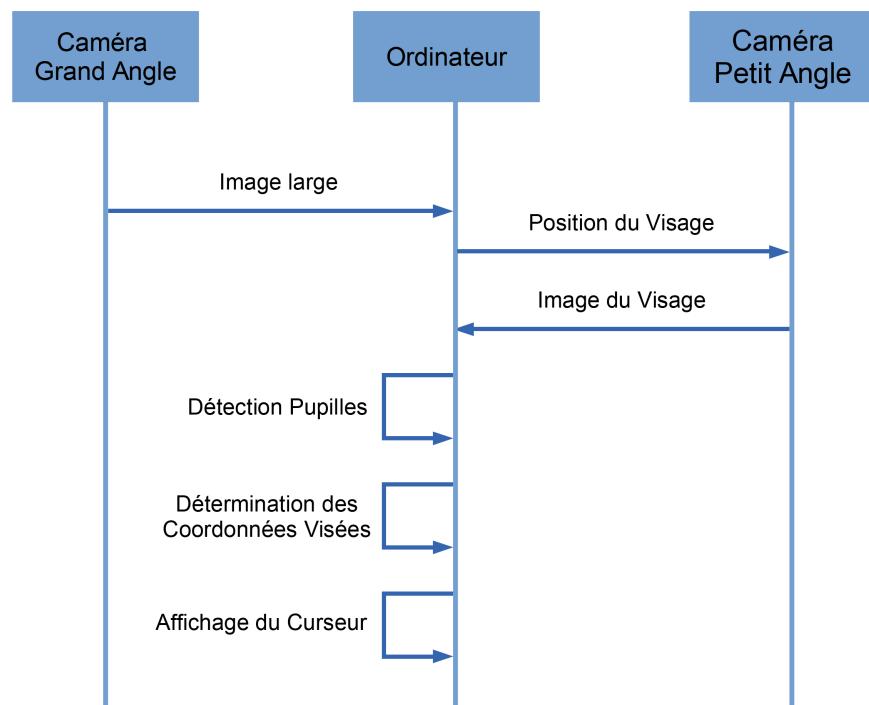


FIGURE 2.6 – Diagramme des spécifications du comportement global

## 2.3 Architecture fonctionnelle

Tout le travail réalisé au préalable sur la spécification fonctionnelle 3 axes permet finalement de proposer une architecture fonctionnelle pour notre système (cf. figure 2.7).

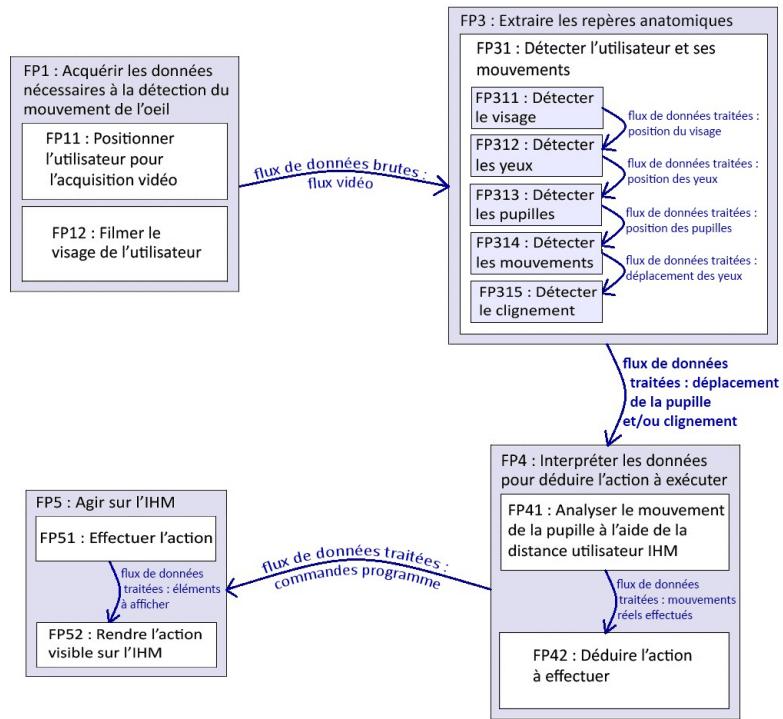


FIGURE 2.7 – Architecture Fonctionnelle

# Partie 3

## Implémentation

### 3.1 Architecture physique et interfaces

Grâce à l'architecture physique (cf figure 3.1), nous avons pu déduire que notre projet se décomposait en trois sous-systèmes : les interfaces d'acquisition, logicielle et graphique.

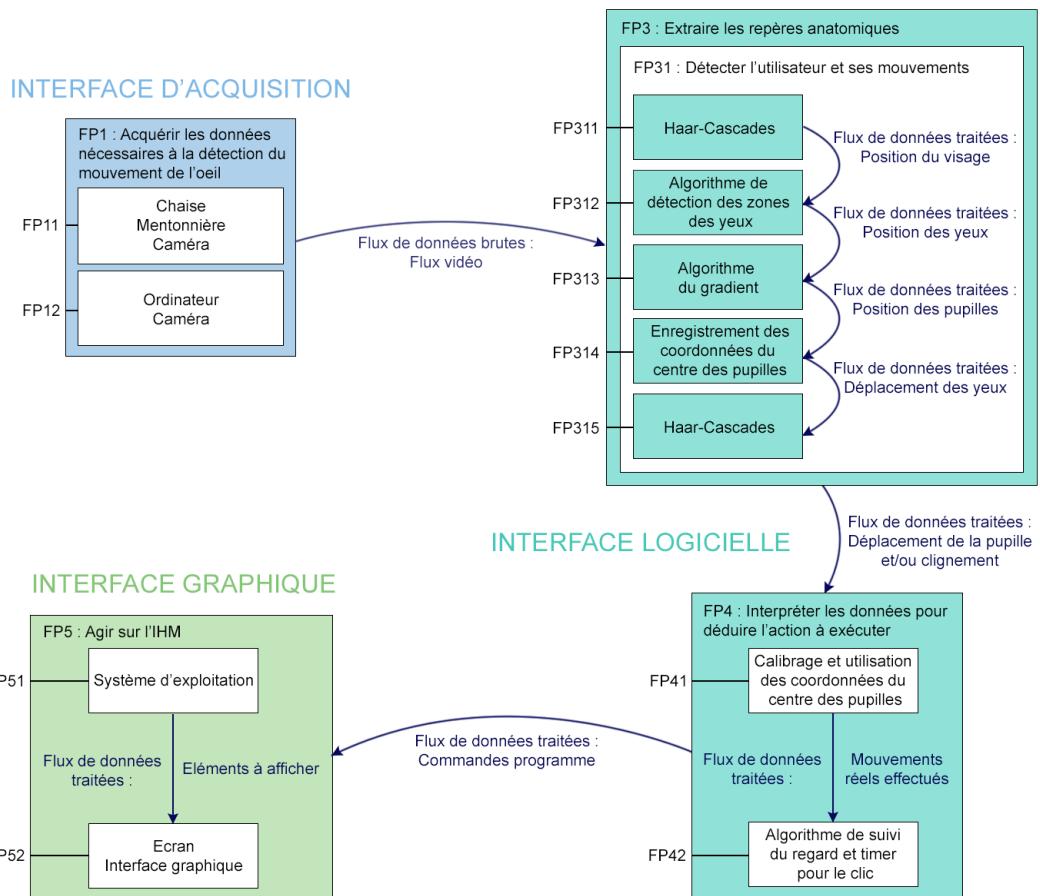


FIGURE 3.1 – Architecture Physique

### 3.1.1 Interface d'acquisition

Cette interface est constituée principalement de la caméra et de la mentonnière (cf figure 3.2). Cette dernière est utilisée pour éviter les mouvements de la tête, non traités par le programme. L'ordinateur occupe également une place secondaire, puisqu'il permet de vérifier que l'utilisateur est convenablement installé.



FIGURE 3.2 – Interface d'acquisition composée de la caméra, fixée à la mentonnière

### 3.1.2 Interface logicielle

L'interface logicielle s'appuie sur le programme contenant des algorithmes développés en C++ avec OpenCV. Ils ont pour but de traiter les images transmises par la webcam afin de détecter le visage, puis les yeux et enfin les pupilles et les clignements pour déduire l'action que l'utilisateur souhaite effectuer. Que ce soit pour le clic ou le mouvement du curseur de la souris, une interaction avec l'OS est nécessaire. Le programme doit donc être adapté suivant le système d'exploitation utilisé.

### 3.1.3 Interface graphique

Enfin, le rôle de l'interface graphique (voir figure 3.3) est de proposer à l'utilisateur un choix de boutons qui lancent des applications. Cette dernière est développée en C++ à l'aide de Qt Creator et ne peut être utilisée que sur Windows car il faudrait changer toutes les commandes d'ouverture d'applications pour exécuter le programme sur un autre système d'exploitation.

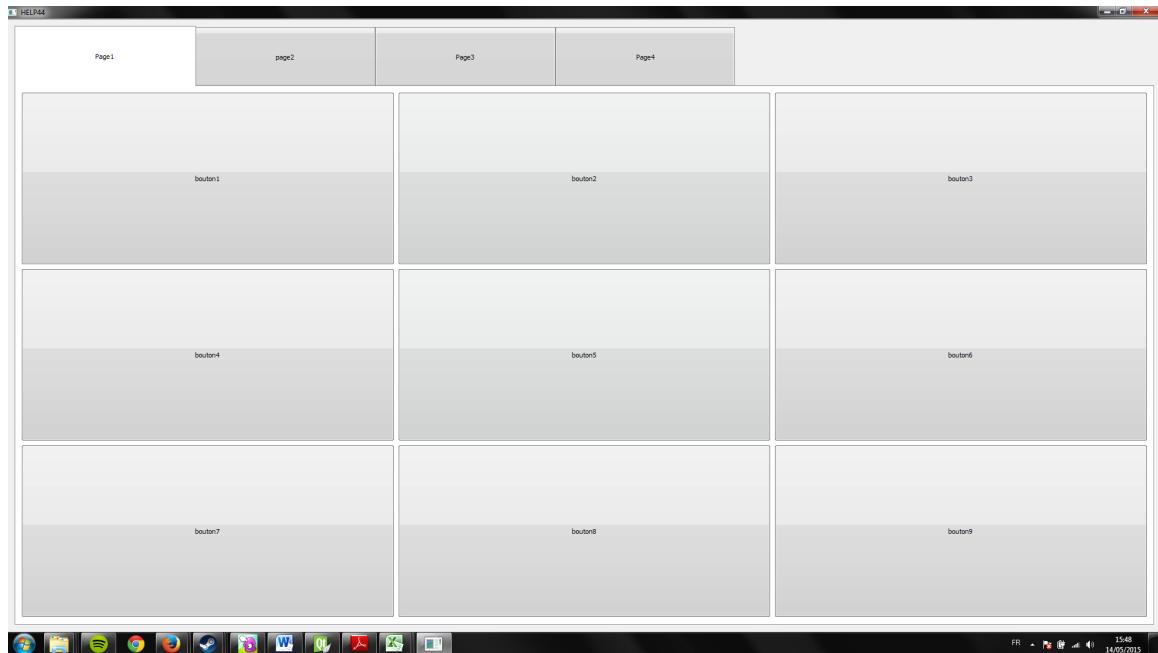


FIGURE 3.3 – Interface graphique 9 boutons

## 3.2 WBS

La structure de découpage du projet découle de l'architecture physique. La figure 3.4 illustre le WBS ainsi établi.

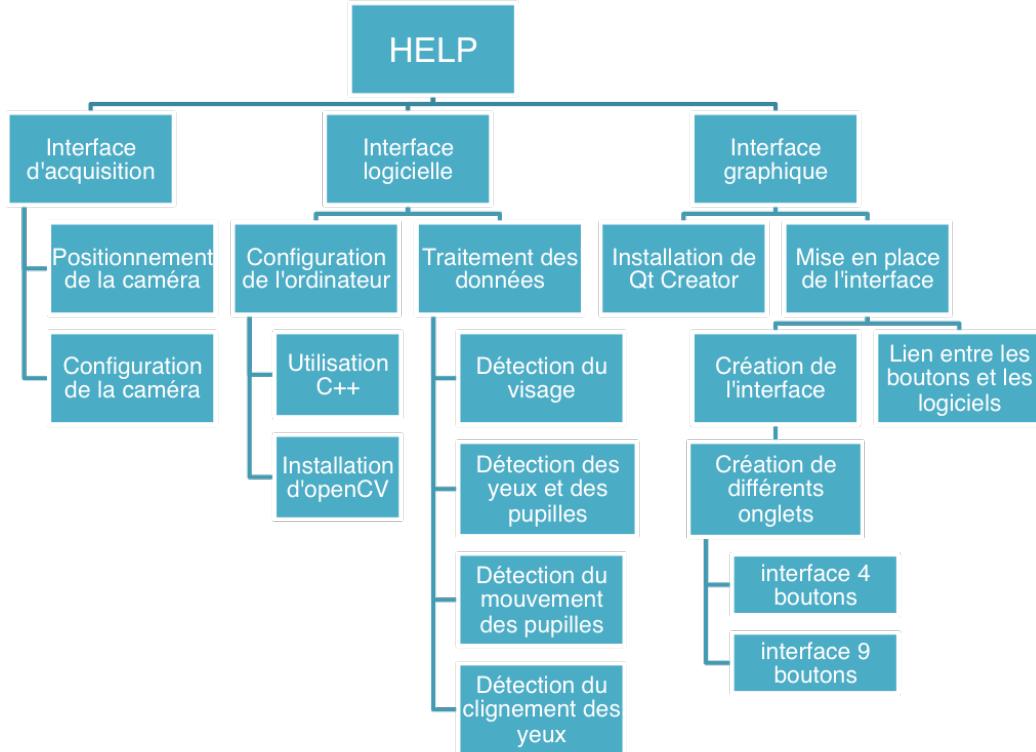


FIGURE 3.4 – WBS

### 3.3 Matériel utilisé

Comme présenté sur l'architecture physique, nous avons besoin de matériel pour réaliser notre système.

#### 3.3.1 Ordinateur

L'ordinateur s'est naturellement imposé comme matériel nécessaire au projet, puisque l'utilisateur devait pouvoir utiliser quelques unes de ses fonctionnalités de base. De plus, pour traiter le flux d'images filmées, nous avions besoin d'une puissance de calcul suffisante. Ainsi, l'ordinateur a un double rôle. Il permet d'afficher l'interface graphique, et également d'obtenir l'action demandé par l'utilisateur,

grâce à des algorithmes écrits en C++, s'appuyant sur OpenCV.

### 3.3.2 Caméra

Si de nombreux types de caméras sont possibles, nous avons retenu la webcam. En effet, la résolution des webcams HD actuelles est largement suffisante pour une détection de pupilles. De plus, elles sont simples d'utilisation, car faciles à installer grâce au port usb d'un ordinateur. Il est aussi aisément de récupérer les flux vidéos. Nous pouvons également noter que les webcams sont relativement peu coûteuses. Parmi celles-ci, nous avons sélectionné la LifeCam Studio HD 1080p (voir figure 3.5), pour son bon rapport qualité prix. En effet, la grande résolution HD 1080p (1920x1080) permet un bon tracking des pupilles.



FIGURE 3.5 – LifeCam Studio HD 1080p  
Source : Microsoft

### 3.3.3 Mentonnière

Afin que les coordonnées du centre des pupilles soient exploitables pour le suivi du regard, nous avions besoin que la tête reste fixe. En effet, lorsque la tête était libre de tout mouvement, les données collectées ne permettait pas de calibrer le programme. La solution de la mentonnière (figure 3.6) a ainsi émergée. De plus, étant donné que le système vise à être utilisé par la suite par des personnes tétraplégiques, l'immobilisation de la tête ne soulève pas d'incohérence. Enfin, nous avons fabriqué une planche de bois pour fixer dans un premier temps la caméra près de l'oeil.



FIGURE 3.6 – Mentonnière

### 3.4 Plan de validation fonctionnelle

Afin de vérifier le bon fonctionnement de notre programme, nous avons établi une stratégie de validation système. L'annexe A spécifie les tests de validation, l'organisation et la logistique de validation, ainsi que le plan documentaire.

### 3.5 Tests unitaires

Des tests unitaires permettent de vérifier que chaque fonction est opérationnelle. Nous ne les détaillerons pas tous. Ainsi, nous nous concentrerons sur le critère 1.5.3 "Permettre le clic gauche / la sélection". Nous décrirons la procédure permettant de vérifier le bon fonctionnement d'une partie précise du logiciel, le clic gauche. Nous testerons ce module, indépendamment du reste du programme, afin de nous assurer qu'il répond aux spécifications fonctionnelles et qu'il fonctionne correctement en toutes circonstances. Notre projet comporte un fichier click.cpp où est implémentée la fonction

```
1 bool closedEyesAndClick(cv::Mat faceROI, CascadeClassifier
eyes_cascade)
```

Cette fonction permet à la fois de détecter si les yeux sont fermés ou ouverts et de cliquer. Elle fonctionne de la façon suivante (figure 3.7) :

- Entrée :
  - Le premier argument est `faceROI`, de type `cv::Mat`. Cet argument est la matrice contenant l'image de l'œil.
  - Le second argument est `eyes_cascade`, de type `CascadeClassifier`. Cet argument est l'`haar_cascade` permettant de détecter un œil.
- Sortie 1 : La sortie est `yeuxFermes`, de type `boolean`. `YeuxFermes` est à `true` lorsque l'œil est fermé (pas de détection possible de l'œil) et à `false` lorsque

l'œil est ouvert. Cette sortie est utilisée par les autres modules de notre partie logicielle.

- Sortie 2 : Une seconde sortie peut être considérée dans cette fonction, le clic qui agit sur l'OS.

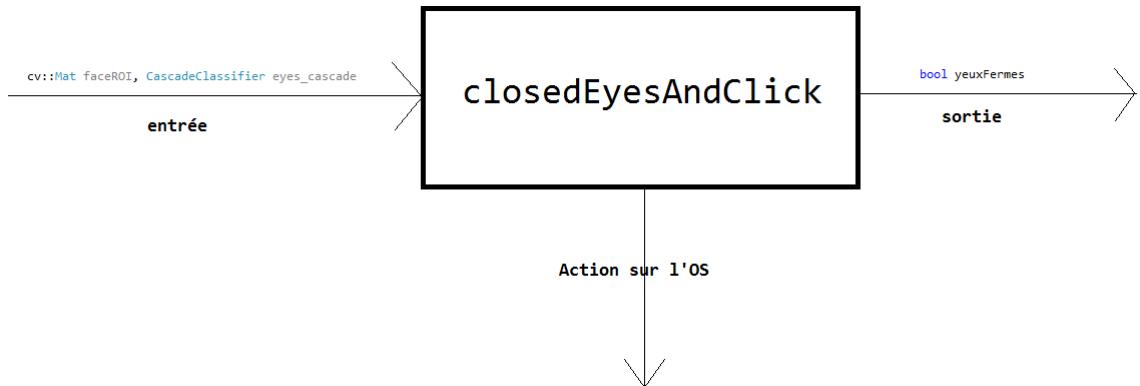


FIGURE 3.7 – Fonction closedEyesAndClick

Nous présenterons donc ici deux tests unitaires, le premier permettant de tester la détection de l'ouverture des yeux et le second permettant de vérifier l'action clic.

### 3.5.1 Test Unitaire 1 : détection de l'ouverture de l'œil

**Les buts/attendus du test** Le but du test est de vérifier que notre fonction renvoie bien le boolean `true` lorsque l'œil est fermé et `false` lorsque l'œil est ouvert.

**La préparation du test (mise en place, outils et instrumentation nécessaires)** Nous avons codé un petit programme (voir annexe B) permettant d'acquérir un flux vidéo et d'appliquer notre fonction à ce flux. Une boucle `while` permet, à chaque itération, de transformer l'image du flux en matrice `cv::Mat`, de récupérer la sortie de la fonction et de l'afficher. De plus, nous avons fait en sorte qu'un rectangle soit tracé autour de l'œil lorsque celui est détecté, afin de mieux visualiser la détection ou non de l'œil.

#### Le déroulement du test

**Conditions initiales (environnement) - Lancement du test** L'utilisateur est assis à 50 cm de l'écran et de la webcam, positionnés face à lui. Le programme permettant d'effectuer le test unitaire est en cours de fonctionnement.

#### Déroulement du test - Fin du test

1. L'utilisateur garde les yeux ouverts. Nous vérifions que `false` s'affiche dans le terminal et qu'un rectangle apparaît bien sur l'image acquise par la webcam.

2. L'utilisateur ferme les yeux. Nous vérifions que `true` s'affiche dans le terminal et que le rectangle ne s'affiche plus sur l'image acquise par la webcam.

Ensuite, nous reprenons cette démarche avec l'utilisateur à 1 m de l'écran et de la webcam.

**Analyse et consignation des résultats du test** Après réalisation des tests, nous pouvons conclure que la méthode `closedEyesAndClick` fonctionne correctement de 50 cm (comme nous pouvons le voir figure 3.8 et figure 3.9) à 1 m. En effet, de 50 cm à 1m, avec un pas de 10 cm, la fonction nous renvoie bien `true` lorsque l'œil est fermé et renvoie `false` lorsque l'œil est ouvert, tout en dessinant un rectangle autour de celui-ci.

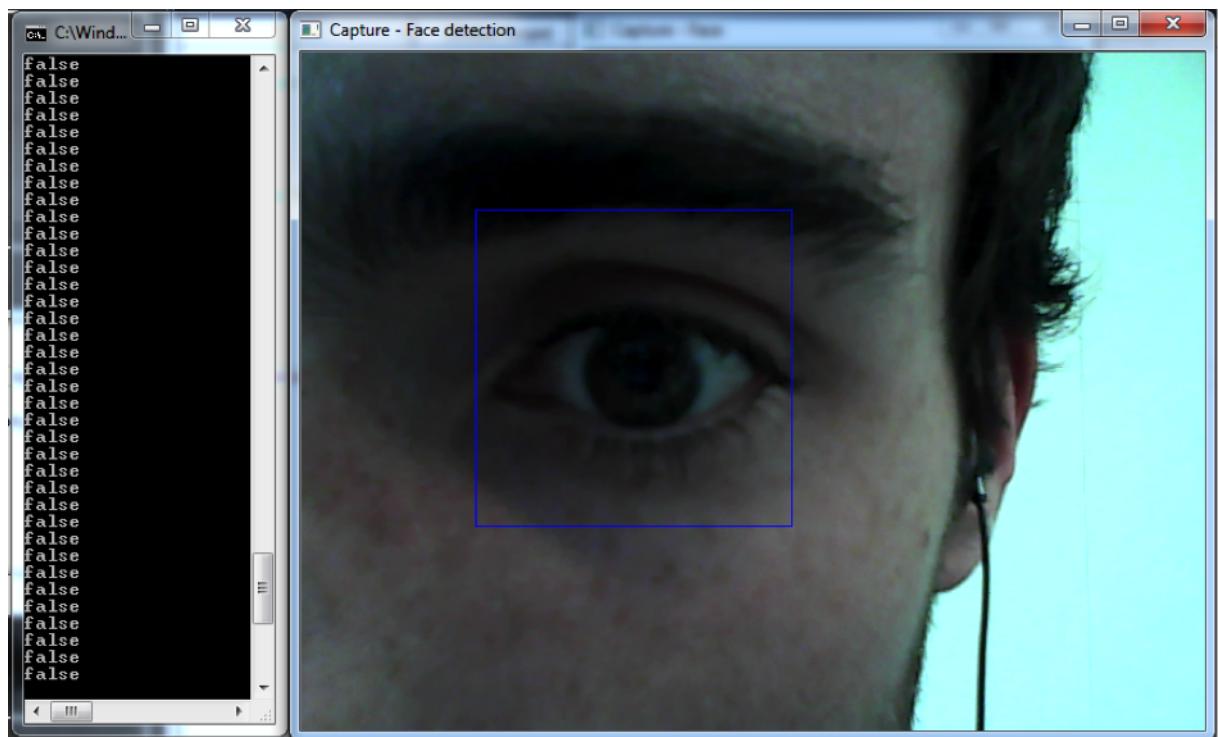


FIGURE 3.8 – Résultat du test unitaire pour la détection de l'ouverture de l'œil – Œil ouvert

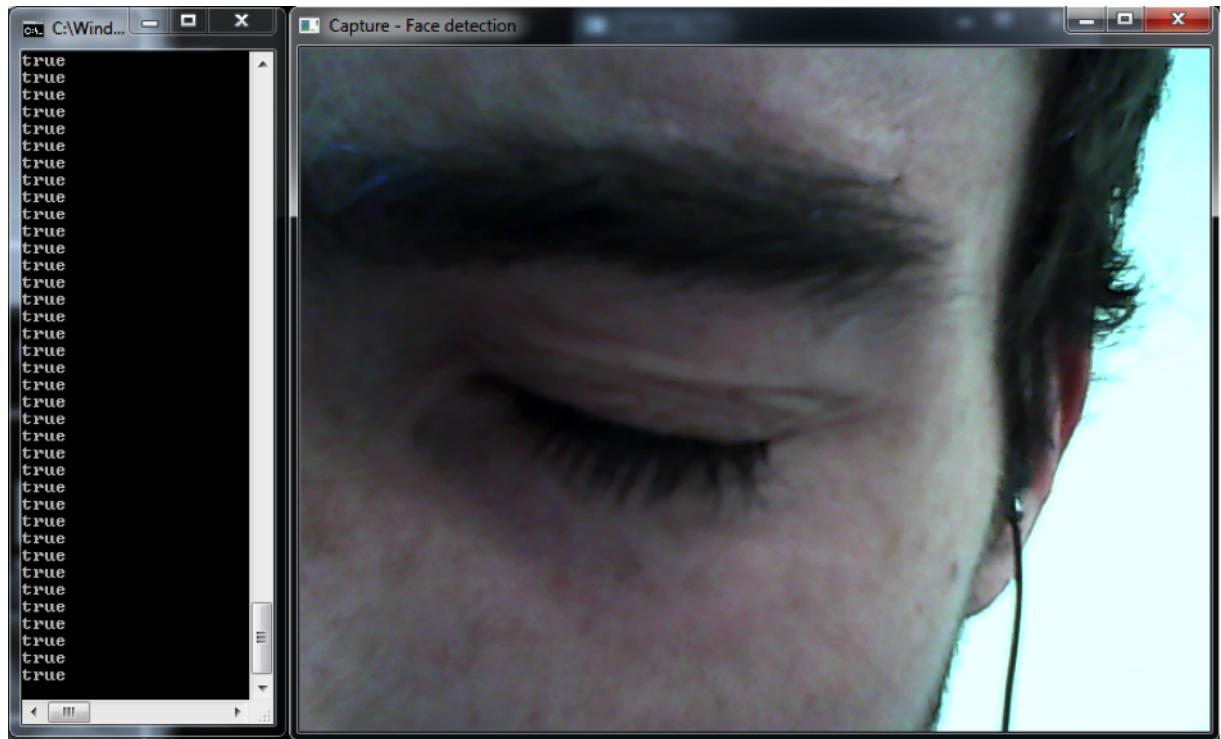


FIGURE 3.9 – Résultat du test unitaire pour la détection de l'ouverture de l'œil – Œil fermé

### 3.5.2 Test Unitaire 2 : clic

**Les buts/attendus du test** Le but du test est de vérifier que notre fonction permet le clic lorsque l'on ferme l'œil.

**La préparation du test (mise en place, outils et instrumentation nécessaires)** Pour préparer ce test unitaire, nous avons repris le programme du test unitaire précédent. Nous avons néanmoins ajouté la ligne `printf("\a")`; à la fonction `closedEyesAndClick` afin de faire en sorte que l'ordinateur émette un bip lorsque le clic est déclenché.

#### Le déroulement du test

**Conditions initiales (environnement) - Lancement du test** L'utilisateur est assis à 50 cm de l'écran et de la webcam, positionnés face à lui. Le programme permettant d'effectuer le test unitaire est en cours de fonctionnement.

#### Déroulement du test - Fin du test

1. L'utilisateur garde les yeux ouverts. Nous vérifions que `false` s'affiche dans le terminal et qu'un rectangle apparaît bien sur l'image acquise par la webcam.
2. Nous plaçons le curseur sur une fenêtre de l'OS se trouvant en arrière plan.
3. L'utilisateur ferme l'œil concerné par le test.

4. Nous vérifions que le rectangle disparait et que `true` s'affiche dans le terminal.
5. Au bout de 1 seconde (minimum), l'utilisateur rouvre l'œil et nous vérifions que le bip sonore est bien émis et que la fenêtre sur laquelle se trouve le curseur passe au premier plan de l'OS. Nous constatons également que `false` s'affiche dans le terminal et qu'un rectangle réapparaît bien autour de l'œil.

Ensuite, nous reprenons cette démarche avec l'utilisateur à 1 m de l'écran et de la webcam.

**Analyse et consignation des résultats du test** Après réalisation des tests, nous pouvons conclure que la méthode `closedEyesAndClick` fonctionne correctement de 50 cm (comme nous pouvons le voir figure 3.8 et figure 3.9) à 1 m. En effet, de 50 cm à 1 m, avec un pas de 10 cm, le programme réagit comme prévu (passage de la fenêtre au premier plan).

# Partie 4

## Organisation

### 4.1 Méthodes de travail

Pour ce deuxième semestre, notre équipe s'est vue réduite de deux membres : Marie-Alice Schweitzer et Laure Dupasquier. Nous avons donc dû totalement remanier notre système de fonctionnement. Notre groupe s'est séparé en deux parties distinctes. D'un côté Katleen Blanchet, Titouan Boulmier et Pierre Jacquot ont travaillé sur la réalisation concrète du projet décrit dans le premier rapport d'avancement. Cela consistait à la création d'un système composé d'une webcam relié à un ordinateur, permettant d'enregistrer la direction du regard de l'utilisateur et de déplacer le curseur sur l'écran en fonction de cette direction.

Et de l'autre Hussain Al Othman a été chargé de réaliser des recherches sur un second système. Ce dernier est directement embarqué sous forme de lunettes sur l'utilisateur. Cette partie viendra compléter notre présentation orale, en donnant une alternative à notre prototype, mais n'est pas réalisée et reste purement théorique. Elle permet d'offrir une alternative à notre projet, si celui-ci est poursuivi par la suite.

Pour ce qui est des réunions, elles étaient conduites tous les mardi matin, afin de se répartir les tâches et aussi de connaître l'avancement de chacun dans son travail. Le rôle de secrétaire, autrefois occupé par Laure Dupasquier a été effectué par Pierre Jacquot afin de garder une trace de chaque réunion et de planifier au mieux nos actions futures.

Katleen Blanchet et Titouan Boulmier ont continué de s'occuper de la rédaction et de la mise en page du rapport ainsi que de la mise à jour de la page GitHub. Les livrables réalisés au cours de ce semestre ont été réalisés en majorité par Pierre Jacquot et relus par le reste de l'équipe.

### 4.2 Outils pour les échanges

Concernant ce semestre les échanges ont été beaucoup plus simples à effectuer. Étant trois à travailler sur la partie principale du projet, la répartition du travail a été rapide et efficace. Le déroulement des séances se faisait le matin à notre arrivée, et nous nous adaptions au fur et à mesure de la journée au travail qu'il restait à faire. La majorité des échanges s'est fait oralement ou via e-mail.

En dehors du projet, pour les questions de logistique (horaires de travail, lieux), nous restions en contact par mail.

### 4.3 Répartition des tâches dans le temps

Afin de visualiser au mieux les tâches à effectuer dans le temps nous avons réalisé un carnet de bord (voir figures [4.1](#) et [4.2](#)) et des diagrammes de Gantt (voir figures [4.3](#), [4.4](#), [4.5](#) et [4.6](#)).

Semaine	Jour	Travail à effectuer durant la séance	Travail réalisé	Tâches à réaliser lors de la prochaine séance
37	lundi 08		Introduction de l'UV et de l'IS	
	jeudi 11		Cours analyse fonctionnelle	
38	lundi 15		Cours fonctions système	
	jeudi 18		Rencontre avec les encadrants pour la présentation des projets	
39	lundi 22		Choix sujet et constitution des groupes + prise de contact avec les membres du groupe (méthodes de travail, rythme...)	
	jeudi 25	Se répartir les tâches pour bien préparer l'interview	Préparation de l'interview avec Mr Mansour	Prévoir le matériel pour l'interview
40	lundi 29	Interview + restitution	Interview	Poster l'interview sur moodle
	jeudi 02	Exigences	Travail en commun pour lister les fonctions, exigences du système	Continuer l'expression des besoins
41	lundi 06	Mise en place de la répartition des activités	Début état de l'art	
	jeudi 09		Etat de l'art	
42	lundi 13		Atelier	
	jeudi 16		Pas cours : journée internationale	
43	lundi 20		Atelier	
	jeudi 23		Etat de l'art	
45	lundi 03		Atelier	
	jeudi 06		Restitution de chaque atelier (explication Github, simulink) à tous les membres du groupe	Faire un point état de l'art
44	vacances	Continuer l'état de l'art chacun de son coté		
46	lundi 10	Pas cours		
	jeudi 13	Point état de l'art	Redistribution état de l'art + raffinement	Continuer le passage de la spécification à l'architecture fonctionnelle
47	lundi 17		Etat de l'art	Commencer la rédaction du rapport
	jeudi 20		Etat de l'art	
48	lundi 24	Répartition des parties du rapport d'avancement	Rapport avancement + état de l'art	
	jeudi 27		Rapport avancement + état de l'art	
49	lundi 1		Rapport avancement	
	jeudi 4		Rapport avancement + choix du matériel	
	vendredi 5		Rapport avancement + commande du matériel	
50	lundi 8		Début des tests de simulations et de codage	
	jeudi 11		Codage et simulation	
51	lundi 15		Codage et simulation	
	jeudi 18	Préparation et organisation de notre présentation projet pour les portes ouvertes		
2	lundi 5		Finalisation du rapport avant de le poster	
	jeudi 8		Atelier II	Poster le rapport avant le dimanche 23
3	lundi 12		Atelier II + évaluation P2P	
	jeudi 15		Atelier II	

FIGURE 4.1 – Carnet de bord semestre 1

5	mardi 27		codage (sur l'existant)	Rédaction livrable sur la réunion de lancement
6	mardi 03		Modification du rapport, codage, livrable	Risque infrarouge
7	mardi 10	Travail sur l'architecture physique/Analyse de codes/Algorithmes existantes	Architecture physique possible établie/Algorithme trouvé	Démarrage Analyse Fonctionnel
8	mardi 17		Travail sur des algorithmes dans différents langages (Matlab, simpleCV, open CV)	Rédaction de l'Analyse Fonctionnel
9	mardi 24		Rédaction de l'Analyse Fonctionnelle/Compréhension de l'algorithme en openCV	
10	mardi 3		Rencontre avec Ali Mansour/Travail sur le code de détection de pupille (travail sur une image plus grosse et suppression de la partie détectant la tête)/ Recherche d'une caméra Narrow View/ TD Gestion de groupe	
11	mardi 10	Trouver une caméra Narrow View	Calcul de la résolution minimale pour la caméra Narrow View/Rencontre avec Mr Reynet afin de commander une caméra	Loi de commande pour servomoteur à déterminer
13	mardi 24	Rédaction livrable 2ème compte-rendu avec l'encadrant	Rencontre avec Ali Mansour, Détection de la direction du regard/Détection du clignement des yeux/Interface Qt	
14	mardi 31		Détection de la direction du regard/Détection du clignement des yeux/Interface Qt	
15	mardi 07		Auto-évaluation de groupe/Détection de la direction du regard/Interface Qt	Avancer les deux parties pendant les vacances
17	mardi 21	Travail sur l'interface Qt /Test de la nouvelle caméra	Avancement de l'interface Qt/Amélioration de la détection avec la nouvelle caméra	
18	mardi 28	Rédaction livrable 3ème compte-rendu avec l'encadrant	Rencontre avec Ali Mansour et M Reynet concernant le projet et la rédaction de la validation fonctionnelle	Finalisation de la validation fonctionnelle/Réalisation Tests Unitaires
19	mardi 05	Centrage de l'image filmée par la caméra/Tests Unitaires / Validation Fonctionnelle	Rédaction de la validation fonctionnelle/Finalisation de l'interface graphique/ Réalisation et Rédaction de tests unitaires	Réalisation de la validation fonctionnelle
20	mardi 12	Avancement du rapport/ Réalisation de tests fonctionnels	Détection de la direction du regard/Interface Qt	Finalisation du rapport/ Crédit de l'eportfolio

FIGURE 4.2 – Carnet de bord semestre 2

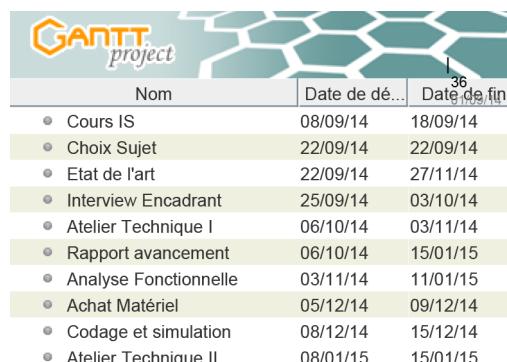


FIGURE 4.3 – Diagramme de Gantt semestre 1

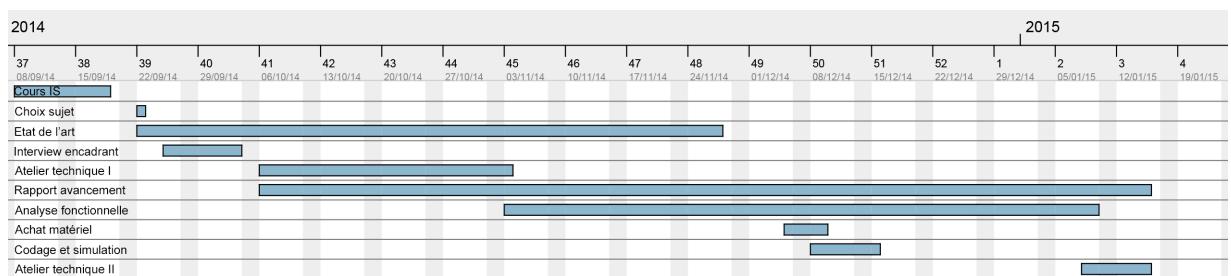


FIGURE 4.4 – Diagramme de Gantt version graphique semestre 1

**GANTT project**

Norm	Date de début	Date de fin
• Algorithme	03/02/15	02/06/15
• Interview Encadrant II	04/05/15	06/05/15
• Interface Graphique	07/04/15	22/05/15
• Validation Fonctionnelle	24/02/15	12/05/15
• Test Unitaire	26/01/15	26/01/15
• Interview Encadrant III	28/04/15	29/04/15
• TD Gestion de Groupe	03/03/15	03/03/15
• e-portfolio	12/05/15	26/05/15
• Poster	26/01/15	26/05/15

FIGURE 4.5 – Diagramme de Gantt semestre 2

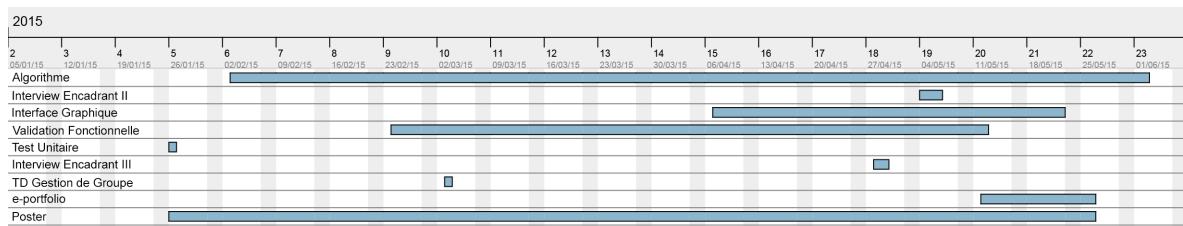


FIGURE 4.6 – Diagramme de Gantt version graphique semestre 2

Ces diagrammes nous ont permis d'anticiper au mieux les échéances, et de commencer à réfléchir aux livrables en avance. En plus de la réunion de planification, nous faisions un point sur l'avancement de chacun tous les matins.

Il est vrai que notre faible nombre a rendu l'organisation compliquée, car l'un de nous devait régulièrement interrompre ses avancées sur la partie technique, qui était pourtant cruciale car non débutée (ou peu) au premier semestre, afin de rédiger des livrables. L'équipe fonctionnait donc à 2/3 de ses possibilités sur la partie technique la plupart du temps au cours du mardi.

# Partie 5

## Partie technique

Après avoir établi l'IS de notre projet, nous nous sommes lancés dans la réalisation technique au deuxième semestre.

La partie programme de notre système se divise en deux parties. La première, le "détecteur", permet de détecter les pupilles, de déplacer le curseur et de cliquer. La seconde partie, l'interface graphique, propose à l'utilisateur une IHM simplifiée, composée des fonctionnalités qui lui seront les plus utiles. Ce sera l'OS, ici Windows ou MAC, qui permet le lien entre ces deux parties du système. En effet, le détecteur permet de déplacer le curseur et de cliquer sur l'interface graphique.

### 5.1 Le détecteur

La partie "détecteur" de notre système permet de déplacer le curseur et de cliquer grâce au mouvement oculaire. Comme nous pouvons le voir figure 5.1, le détecteur prend en entrée un flux vidéo acquis par la webcam et permet d'agir, en déplaçant le curseur et cliquant, sur l'interface graphique.



FIGURE 5.1 – Entrée/Sortie du détecteur

#### 5.1.1 Principe de fonctionnement

Le détecteur est composé de différentes parties que nous pouvons retrouver sous forme de méthodes (voir figure 5.2). Ces différentes parties permettent de :

1. Acquérir le flux vidéo et le transformer en images matricielles
2. Déetecter le visage

3. Déetecter les yeux
4. Déterminer le centre des pupilles
5. Déplacer le curseur en fonction de la position des centres
6. Cliquer

Afin d'expliquer comment sont réalisées ces fonctionnalités, nous allons présenter rapidement différentes méthodes qui composent le détecteur en nous attardant sur les points essentiels vus ci-dessus.

D'autre part, il est important de signaler que nous avons essayé de convertir la position du centre de la pupille en coordonnée de curseur de différentes façons. Ainsi, il existe plusieurs versions informatiques du détecteur. Dans la suite de cette partie du rapport, nous présenterons les méthodes les plus claires afin de faciliter la compréhension du lecteur. Cependant, suivant la version du détecteur, ces méthodes pourront présenter quelques différences.

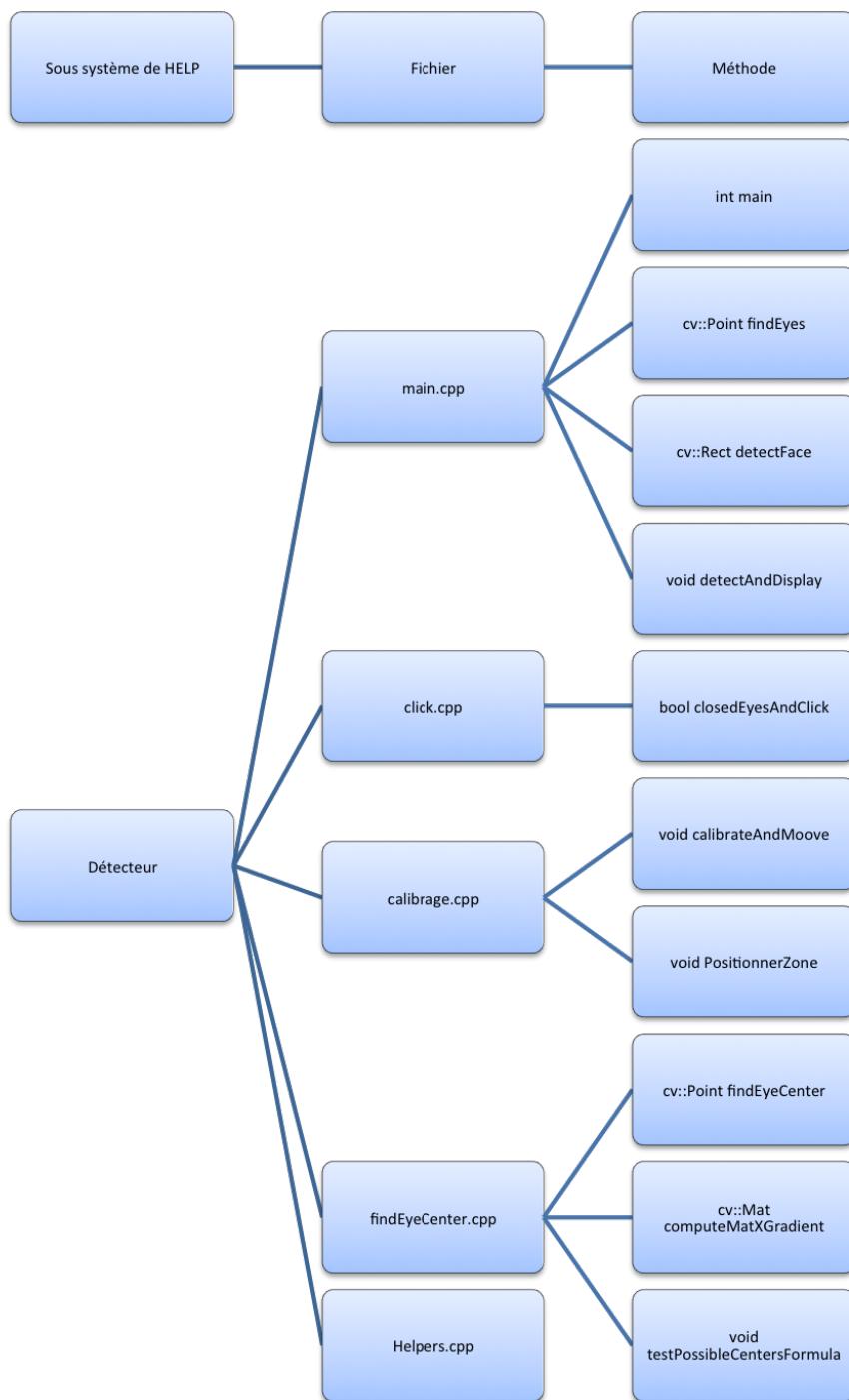


FIGURE 5.2 – Organisation du détecteur

### 5.1.2 Acquisition du flux vidéo et transformation en images matricielles

La méthode `main` permet de récupérer le flux vidéo et de le convertir en images matricielles (voir annexe C). Ces images matricielles seront nécessaires aux méthodes utilisées par le détecteur. La méthode `cvCaptureFromCam(0)` permet de récupérer le flux vidéo de la webcam par défaut. Ensuite, dans une boucle `while`,

l'image matricielle, récupérée grâce à la méthode `cvQueryFrame`, est inversée selon la verticale grâce à la méthode `cv::flip` afin de faciliter l'utilisation des méthodes du détecteur. En effet, sans cette inversion, un mouvement vers la droite de l'utilisateur correspondrait à un mouvement vers la gauche sur l'écran de l'ordinateur. Ainsi, elle facilite par exemple le placement de l'utilisateur.

### 5.1.3 Détection du visage

Tout d'abord, il faut savoir que nous effectuons la détection du visage et des yeux grâce aux `haar_cascades` [8][17].

**Haar cascades / Caractéristiques pseudo-Haar** Les caractéristiques pseudo-Haar (Haar-like features en anglais) sont utilisées en vision par ordinateur pour détecter des objets dans des images numériques. Le principal avantage de celles-ci est la rapidité de calcul. Cependant, cela impose une détection d'objets simples uniquement. Les Haar-like features sont des fenêtres de détection (ou masques) qui délimitent des zones rectangulaires adjacentes. Une caractéristique rectangulaire simple peut être définie comme la différence des sommes de pixels de deux ou plusieurs zones rectangulaires adjacentes. De plus, ce rectangle prend toutes les tailles possibles et se déplace à toutes les positions de l'image à laquelle est appliquée cette caractéristique. Les valeurs indiquent certaines propriétés d'une zone particulière de l'image. **Une caractéristique est donc un nombre réel qui code les variations du contenu pixellique à une position et taille donnée dans la fenêtre de détection.** Chaque type de caractéristique peut indiquer l'existence (ou l'absence) de certaines particularités dans l'image étudiée telles que des contours ou des changements de texture. Par exemple, une caractéristique à 2 rectangles permet d'indiquer où se situe une frontière entre une zone sombre et une zone claire.

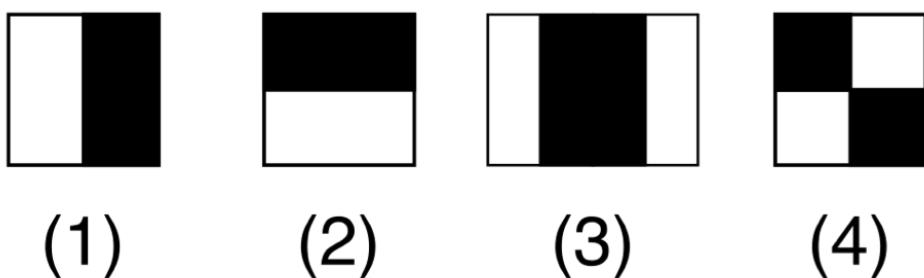


FIGURE 5.3 – Un exemple des premières caractéristiques pseudo-Haar utilisées par Viola et Jones en 2001

Source : Indif - Wikimedia Commons

Ensuite, un classifieur est créé à partir de quelques centaines d'images de référence de l'objet ciblé (dans notre cas, le visage et les yeux). Ces images sont dimensionnées à la même taille et sont en négatif. Une fois le classifieur créé à partir de cette banque d'images, il peut être appliqué à une région d'une image selon la méthode vue précédemment. Ainsi, il sera retourné un nombre réel proche

de 1 lorsque la région ciblée de l'image sera proche du classifieur et 0 lorsqu'elle s'en éloignera. Pour rechercher l'objet dans toute l'image, le classifieur sera déplacé à travers cette dernière. De plus, il sera redimensionné afin de trouver l'objet recherché à différentes tailles (ce qui est plus simple que de redimensionner l'image elle-même).

Le mot "cascade" dans le nom du classifieur signifie que le classifieur résultant se compose de plusieurs classificateurs simples (étapes) qui sont appliqués à la suite à une région d'intérêt jusqu'à ce que, à un moment donné, la région candidate soit rejetée ou ait passé toutes les étapes. Le mot "boosté" signifie que les classificateurs, à chaque étape de la cascade, sont eux-mêmes complexes et qu'ils sont construits sur des classificateurs de base en utilisant l'une des quatre techniques différentes suivantes : Discrete Adaboost, Real Adaboost, Gentle Adaboost and Logitboost. Les caractéristiques pseudo-haar sont l'entrée du classifieur basique.

**Utilisation de ces haar\_cascades** La méthode `detectFace` permet de réaliser la détection du visage grâce à un haar\_cascade. Elle prend en entrée l'image acquise par la webcam. Les haar\_cascades permettant de détecter le visage et les yeux sont d'abord chargés dans le main (voir annexe D) puis, dans la méthode `detectFace`, un vecteur de `cv::Rect` est créé et rempli lors de l'utilisation de l'haar\_cascade. Dans ce vecteur se trouvent donc les rectangles encadrant le visage (voir annexe E). La méthode trace ensuite un rectangle (voir figure 5.4) autour du visage en prenant le premier du vecteur. La méthode `detectFace` retourne ce rectangle.

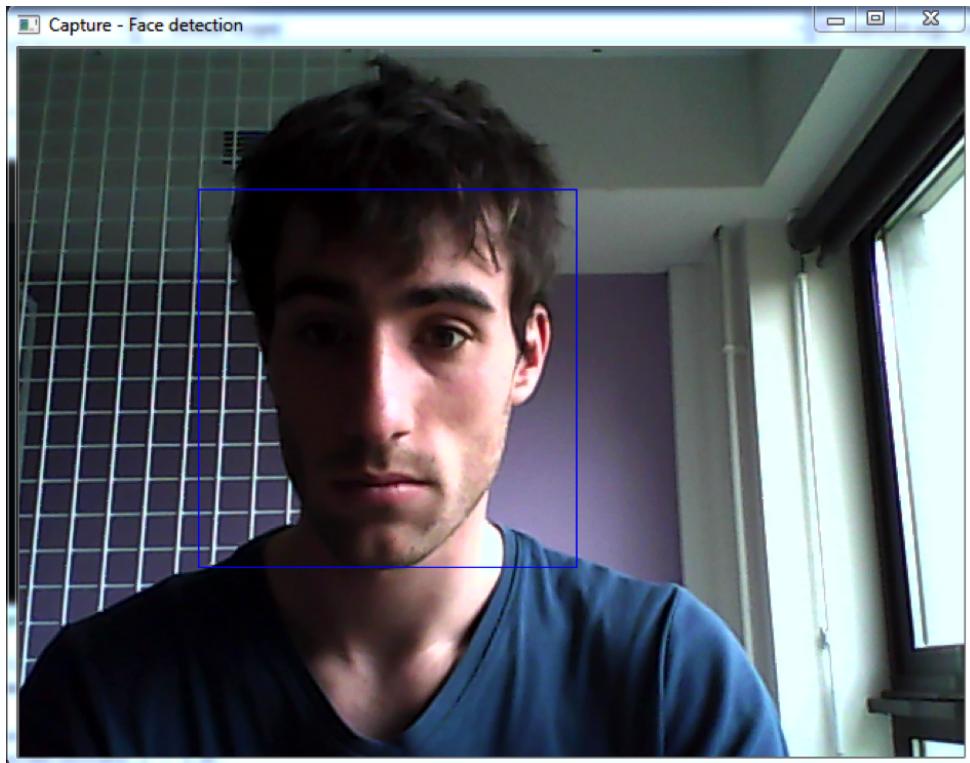


FIGURE 5.4 – Détection du visage

### 5.1.4 Détection des yeux

La méthode `closedEyesAndClick` qui prend en argument l'image du visage ainsi que l'`haar_cascade` permettant de détecter un œil, et qui retourne un booléen en fonction de l'ouverture ou non de l'œil, permet de détecter l'œil. En effet, de la même manière que pour la détection du visage, un vecteur "eyes" contenant les rectangles encadrant les yeux est créé. Ensuite, la méthode `detectMultiScale` permet de remplir ce vecteur grâce à l'image du visage et l'`haar_cascade` (voir annexe F).

### 5.1.5 Détermination du centre des pupilles

Concernant la détection du centre des pupilles, la méthode `findEyeCenter` prend en argument une matrice contenant l'image du visage, un rectangle encadrant l'œil et un nom de fenêtre dans laquelle sera affichée l'image de l'œil traitée par la méthode. Tout d'abord, le rectangle encadrant l'œil appliqué à l'image du visage permet de créer une matrice contenant l'image de l'œil. Ce sera grâce à cette matrice que sera détecté le centre des pupilles. De plus, la méthode retourne les coordonnées, dans l'image de l'œil, du centre des pupilles. Il sera ensuite effectué un changement de repère dans la méthode `findEyes` afin de d'obtenir les coordonnées des pupilles dans l'image du visage (voir figure 5.5 et annexe G).

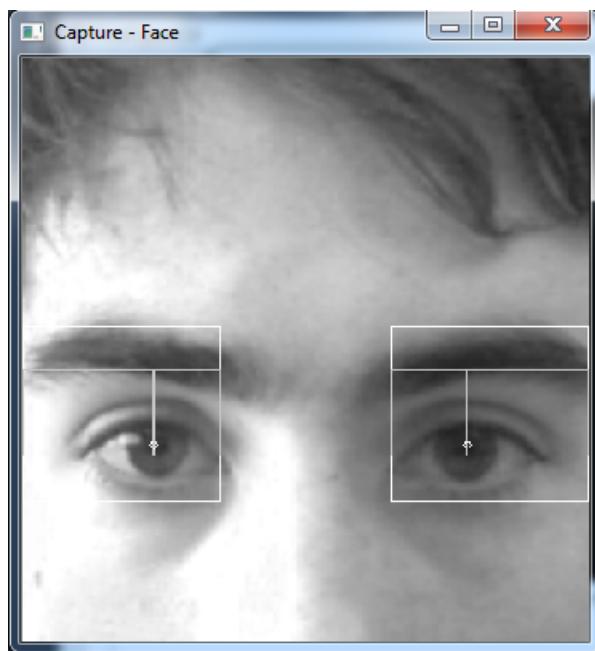


FIGURE 5.5 – Détection du centre des pupilles

**Localisation du centre des pupilles** Afin de localiser le centre des pupilles, nous avons utilisé un code de Tristan Hume [5]. Nous pouvons d'ailleurs noter que ce code sera la seule partie de notre projet que nous avons repris d'un système déjà existant. Il s'appuie sur un article de Fabian Timm [16]. Cet article décrit une méthode s'appuyant sur les gradients. La direction de ces gradients est utilisée afin

de déterminer le centre des pupilles. Cependant, dans l'image de l'œil, plusieurs centres des pupilles sont possibles si l'on considère seulement les gradients. En effet, certains facteurs tels que les sourcils, les paupières, des lunettes, ou tout simplement des reflets peuvent modifier la valeur théorique des gradients.

Considérons un centre possible  $\mathbf{c}$  et un vecteur gradient  $\mathbf{g}_i$  à la position  $\mathbf{x}_i$ . Le vecteur  $\mathbf{d}_i$  de la distance normalisée (entre  $\mathbf{c}$  et  $\mathbf{x}_i$ ) devrait avoir la même orientation que  $\mathbf{g}_i$ . Ainsi, le produit scalaire entre ces deux vecteurs devrait être égale à un. Pour trouver le centre optimal  $\mathbf{c}^*$  des pupilles, il faudrait donc que la somme sur  $i$  des produits scalaires entre  $\mathbf{d}_i$  et  $\mathbf{g}_i$  soit maximale. Ainsi,  $c^*$  sera égale à l'argument du maximum sur  $\mathbf{c}$ , noté *arg max*, (l'ensemble des points en lesquels une expression atteint sa valeur maximale) de cette somme.

$$\mathbf{c}^* = \arg \max_{\mathbf{c}} \left( \frac{1}{N} \sum_{i=1}^N (\mathbf{d}_i^T \mathbf{g}_i)^2 \right) \quad (5.1)$$

$$\mathbf{d}_i = \frac{\mathbf{x}_i - \mathbf{c}}{\|\mathbf{x}_i - \mathbf{c}\|_2}, \forall i : \|\mathbf{g}_i\|_2 = 1 \quad (5.2)$$

Nous pouvons voir une illustration de cette méthode figure 5.6.

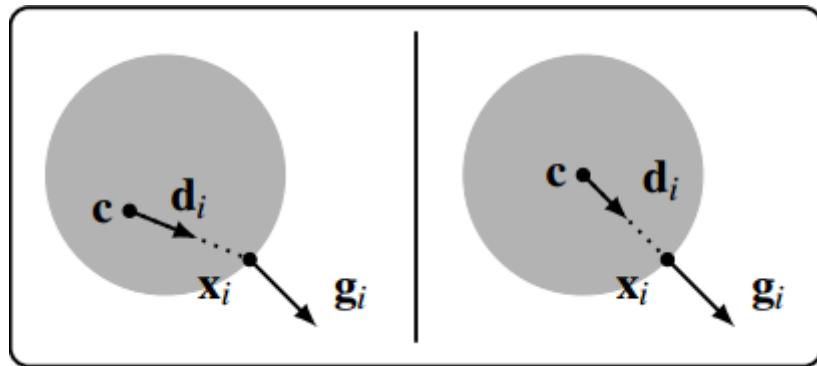


FIGURE 5.6 – Exemple de localisation du centre d'un rond noir sur fond blanc [16]

**Algorithme du gradient** Afin de calculer les gradients dans l'image de l'œil, l'algorithme utilisé par la méthode `findEyeCenter` est celui utilisé par Matlab transformé en C++. En Matlab, cela donne `[x(2)-x(1) (x(3:end)-x(1:end-2))/2 x(end)-x(end-1)]`, avec `x` l'entrée (voir annexe H pour la version en C++ utilisée par `findEyeCenter`). Le gradient X est ainsi obtenu comme suit `cv::Mat gradientX = computeMatXGradient(eyeROI)` et le gradient Y est obtenu en transposant le résultat de la méthode appliquée à la transposée de X : `cv::Mat gradientY = computeMatXGradient(eyeROI.t()).t()`. Ensuite, un seuil est appliqué grâce à la méthode `computeDynamicThreshold` afin de filtrer les gradients obtenus `double gradientThresh = computeDynamicThreshold(mags, kGradientThreshold)` où "mags" est la matrice des magnitudes de tous les gradients calculés.

### 5.1.6 Déplacement du curseur en fonction de la position des centres

Cette fonctionnalité est assurée par la méthode `calibrateAndMoove` qui prend en argument la position du centre d'une pupille dans l'image de l'œil. Cette méthode ne retourne rien, mais permet de déplacer le curseur.

`CalibrateAndMoove` est composée de cinq parties différentes, quatre consacrées à la calibration et une dernière permettant le déplacement du curseur une fois la calibration effectuée. Les trois premières étapes consistent à enregistrer la position du centre de la pupille lorsque l'utilisateur regarde en haut à gauche de l'écran, puis en haut à droite et enfin en bas à gauche. Ces positions sont stockées dans la matrice `pointsMesuresReference[3]`. Ensuite, la quatrième étape permet d'effectuer un changement de repère, la position du centre de la pupille est transformée en position du curseur sur l'écran. Pour cela, nous avons choisi de considérer qu'il s'agit d'un changement de repère affine. Nous pouvons voir figure 5.7 un schéma de l'écran avec les coordonnées de trois coins de l'écran dans le repère de l'écran lui-même et dans le repère de l'image de la pupille.

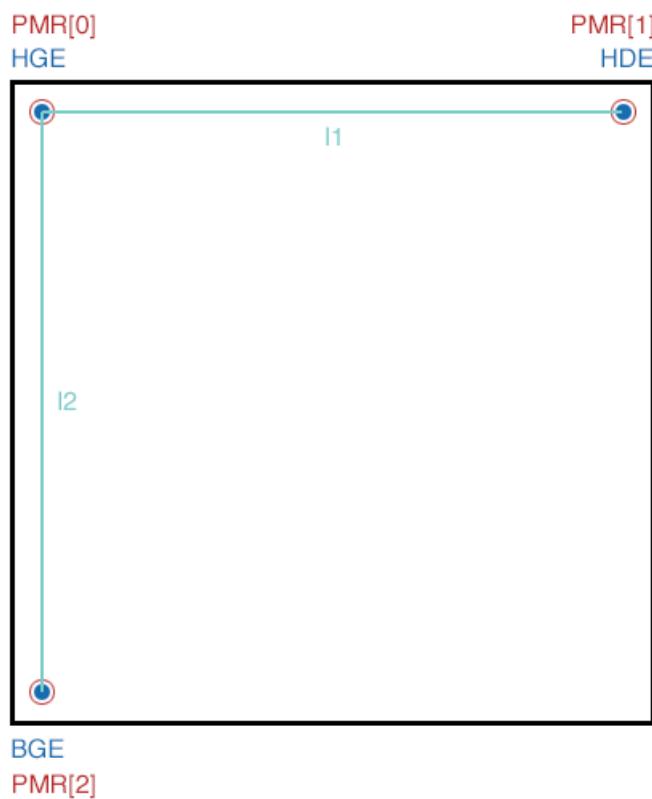


FIGURE 5.7 – Schéma calibration

Avec, entre autre :

- `PMR[0]` = `pointMesureReference[0]` qui correspond aux coordonnées de la pupille lorsque l'utilisateur regarde le coin en haut à gauche.
- `HGE` = `hautGaucheEcran` qui correspond au coordonnées du point en haut à gauche de l'écran (0,0).

Ainsi, la longueur l1 est égale à HGD.x et correspond à (PMR[1]-PMR[0]).x

Deux équations sont donc nécessaires à ce changement de repère (une suivant X et une suivant Y) et les coordonnées du point visé sont calculées de la manière suivante :

```
1 cv::Point pointVise;
2
3 double aH = -(double)hautDroitEcran.x / (double)(
4     pointsMesuresReference[0].x - pointsMesuresReference[1].x)
5 ;
6 double bH = (double)hautGaucheEcran.x - aH*
7     pointsMesuresReference[0].x;
8
9 double aV = -(double)basGaucheEcran.y / (double)(
10    pointsMesuresReference[0].y - pointsMesuresReference[2].y)
11 ;
12 double bV = (double)hautGaucheEcran.y - aV*
13     pointsMesuresReference[0].y;
14
15 pointVise.x = aH*leftPupil.x + bH;
16 pointVise.y = aV*leftPupil.y + bV;
```

Enfin, la cinquième partie permet de bouger le curseur grâce à `SetCursorPos(pointVise.x, pointVise.y)`. De plus, la méthode sera utilisée pour notre démonstrateur afin de découper l'écran en quatre zones (puis neuf pour les utilisateurs les plus à l'aise avec le système). Ainsi, si le point visé se trouve dans la partie supérieure gauche de l'écran, le curseur sera placé au centre de cette zone (de même que pour les trois autres zones).

### 5.1.7 Clic

Le clic est assuré par la méthode `closedEyeAndClick` que nous avons déjà présenté précédemment. Lorsque l'œil n'est plus détecté (œil fermé) (déttection de l'œil grâce à l'`haar_cascade`), un compte à rebours est déclenché. Une fois l'œil rouvert, il est de nouveau détecté et le compte à rebours est stoppé. Si le délai est supérieur à 1 seconde et inférieur à 3 secondes, le clic est déclenché grâce à la fonction `mouse_event`.

```
1 mouse_event(MOUSEEVENTF_LEFTDOWN, 0, 0, 0, 0);
2 mouse_event(MOUSEEVENTF_LEFTUP, 0, 0, 0, 0);
```

## 5.2 Interface graphique

### 5.2.1 But et enjeux de l'interface

Dès le début du projet, l'idée d'une interface graphique permettant de contrôler le PC a été envisagée. En effet, la finalité est d'aider les personnes tétraplégiques, or, l'utilisation de l'interface standard d'un ordinateur reste compliquée. Nous avons donc pensé à créer notre propre outil de communication avec l'utilisateur, qui permettrait de lancer les programmes favoris de ce dernier.

### 5.2.2 Critères d'exigence

Afin de rendre l'IHM la plus ergonomique possible pour l'utilisateur, nous avons décidé de valider deux critères principaux. En premier lieu, l'interface devra être adaptée au contrôle via notre logiciel de gaze-tracking. Celui-ci étant parfois imprécis et pouvant être fatiguant à utiliser sur le long terme, nous avons fait en sorte que les boutons de l'interface occupent le maximum d'espace sur l'écran (voir figure 5.8).

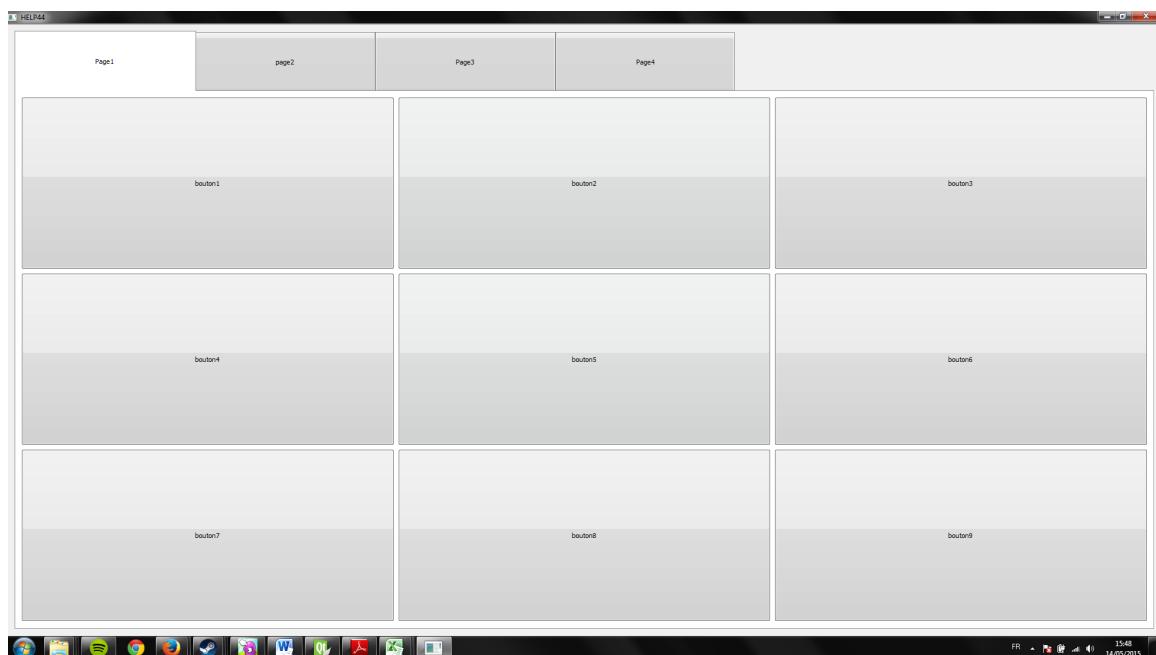


FIGURE 5.8 – Interface graphique 9 boutons

Deux modes de fonctionnement existent pour notre logiciel de gaze-tracking : un mode de fonctionnement libre, où l'utilisateur peut déplacer le curseur où il veut sur l'écran, et un mode assisté où le mouvement est restreint à certaines zones de l'écran. De plus, deux interfaces distinctes ont été développées. La première à neuf boutons (voir figure 5.8), réservée aux utilisateurs les plus à l'aise avec le logiciel de gaze-tracking et la seconde à quatre boutons (voir figure 5.9) pour les utilisateurs débutants. Ces deux interfaces autorisent l'utilisation du mode assisté et du mode libre.

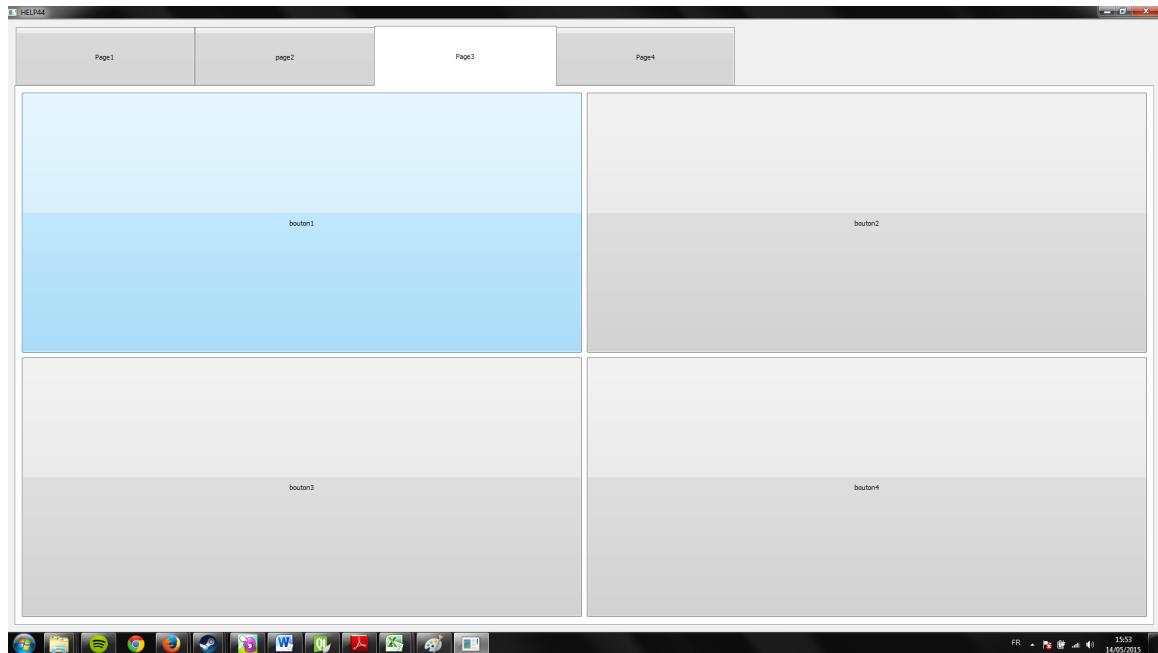


FIGURE 5.9 – Interface graphique 4 boutons

Dans un deuxième temps, l'interface devait être paramétrable afin de pouvoir lancer les logiciels favoris de l'utilisateur, ou encore un film ou un document PDF. Ainsi, le clic sur l'un des boutons lance un logiciel, soit entré par défaut, soit personnalisable par l'utilisateur. La page permettant de paramétriser les boutons est présentée figure 5.10.

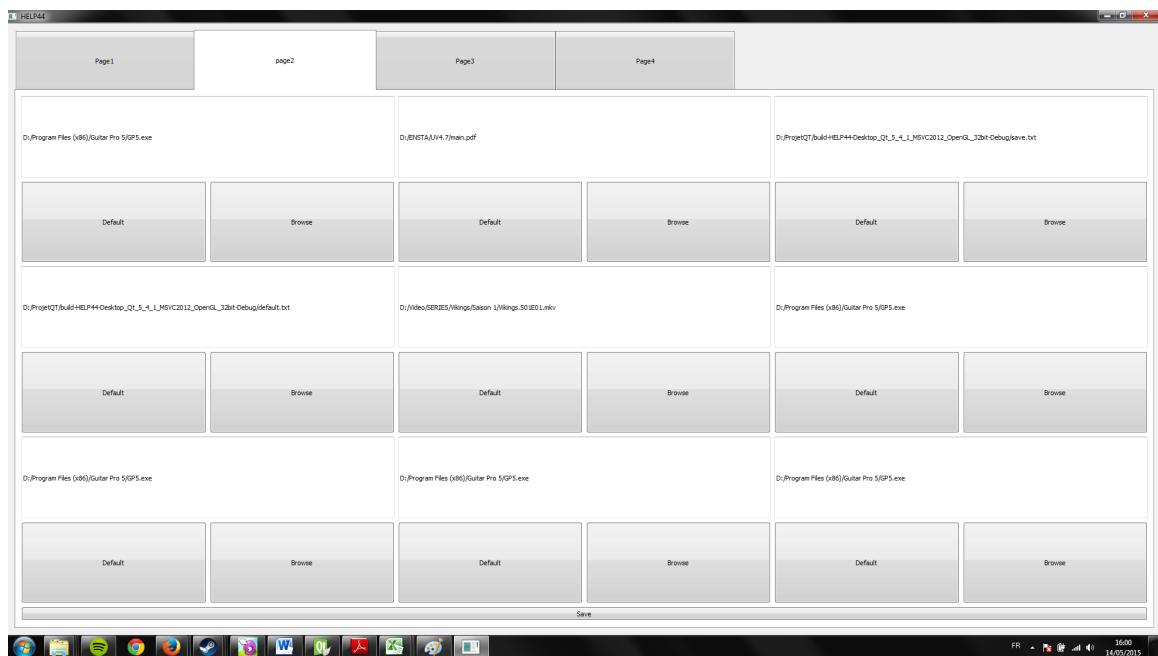


FIGURE 5.10 – Interface de configuration des 9 boutons (existe aussi pour les 4 boutons)

Chaque chemin peut être modifié à l'aide du bouton "Browse", qui permet de changer le logiciel exécuté par le bouton que l'on a choisi. Ces adresses peuvent

ensuite être stockées dans un fichier texte à l'aide du bouton "Save" (le fichier texte se situe dans le dossier contenant l'exécutable). En cas d'erreur ou d'une manipulation non souhaitée, le bouton "Default" permet de lier de nouveau le bouton erroné à un chemin valide. Ces adresses par défaut permettent de lancer les logiciels d'ergonomie de Windows, ainsi que les utilitaires classiques tels que la calculatrice ou Internet Explorer.

### 5.2.3 Outils de réalisation

Comme langage de programmation, nous avons choisi le C++ couplé à Qt Creator, pour réaliser cette interface. En effet, notre programme de gaze-tracking étant codé en C++, il sera plus facile dans le futur de réunir les deux logiciels en un seul (chose qui n'est pas encore faite à ce jour). D'autre part la librairie Qt permet de faire rapidement et simplement des interfaces graphiques, et c'est donc présentée comme une évidence.

## 5.3 Limites du système et problèmes rencontrés

### 5.3.1 Problèmes matériels

L'installation d'OpenCV sur Windows n'est pas aisée et a fait perdre beaucoup de temps à deux membres de l'équipe, déjà fortement réduite.

De plus, ayant besoin de fixer la tête pour que notre calibrage soit efficace, nous avons du trouver un moyen de la maintenir. Nous avons finalement opté pour une mentonnière, qui nous a gentiment été prêtée par un ophtalmologue.

### 5.3.2 Limites du détecteur

Concernant les limites de la partie détecteur de notre système, nous pouvons distinguer trois grandes catégories :

- Limites liées au mouvement de la tête
- Limites liées à la caméra
- Limites liées à la luminosité

#### 5.3.2.1 Limites liées au mouvement de la tête

La principale limite de notre démonstrateur est l'absence de possibilité de mouvement de la tête. En effet, nous devons utiliser une mentonnière afin de maintenir la tête et de réduire les mouvements de celle-ci. Cette restriction de notre démonstrateur peut s'expliquer par trois limites de notre système.

**Limites des haar\_cascades** La détection du centre des pupilles a d'abord été testée en utilisant l'image du visage où étaient récupérées les images des yeux grâce à des constantes liées à la morphologie du visage. Cependant, le rectangle encadrant le visage, renvoyé lors de l'utilisation de l'haar\_cascade permettant la

détection du visage, n'est pas suffisamment précis. En effet, nous avons constaté un léger décalage de quelques pixels au cours du temps de ce rectangle (comme une vibration). Or, les coordonnées du centre des pupilles ayant une variation de seulement quelques pixels, ces décalages du rectangle modifiaient trop les coordonnées du centre pour avoir une estimation correcte du point visé sur l'écran. Pour corriger le problème, nous avons tout d'abord fixé le cadre de la tête lorsque sa variation restait faible. Néanmoins, même si c'était mieux, un très petit mouvement de la tête obligeait à recalibrer le système. L'idée de mettre une gommette de couleur vive sur le visage de l'utilisateur a alors été émise. En effet, il est facile de filtrer une couleur avec OpenCV, ce qui nous donnait un point fixe qui servait de repère. Cette solution n'a pas non plus abouti. Nous avons finalement essayé de récupérer l'image des yeux non plus grâce à des constantes, mais en utilisant un haar\_cascade permettant de récupérer le rectangle encadrant les yeux. Cependant, encore une fois, la position de ce rectangle variait trop au cours du temps. C'est pourquoi notre démonstrateur n'effectue pas cette détection du visage mais seulement celle de l'oeil et filme directement celui-ci avec une tête immobile.

**Orientation de la tête** Ensuite, une deuxième limite à notre système est qu'il ne prend pas en compte l'orientation de la tête. Si le déplacement du visage dans le plan était pris en compte, un mouvement circulaire de la tête vers la gauche ou la droite provoquerait un dérèglement du calibrage. En effet, lorsqu'un utilisateur fixe un point de l'écran en tournant la tête ou non, cela change les coordonnées du centre des pupilles. Cependant, nous pouvons penser qu'un utilisateur tétraplégique ne bougera pas sa tête, mais il ne faudra pas non plus que sa tête glisse.

**Tête verticale seulement** La dernière limite quant au mouvement de la tête est que l'haar\_cascade ne permet de détecter qu'un visage plus ou moins vertical. Empiriquement, nous avons vu qu'à partir d'un angle d'environ 10 degrés, l'haar\_cascade ne permettait plus la détection du visage.

### 5.3.2.2 Limites liées à la caméra

Pour notre démonstrateur, nous utilisons une mentonnière afin de limiter les mouvements du visage qui induiraient de mauvaises valeurs pour les points visés. En effet, un décalage du visage par rapport à la caméra induirait un étalonnage obsolète. Ainsi, le principe du casque / lunettes étudié lors de la phase bibliographique aurait peut-être permis de pallier ce problème.

### 5.3.2.3 Limites liées à la luminosité

Une des principales limites de l'utilisation de la méthode permettant la détection du centre des pupilles est qu'elle s'appuie sur les gradients. Or, ces gradients peuvent être plus ou moins en accord avec leur valeur théorique selon la luminosité. En effet, un simple reflet dans la partie de l'image de la pupille peut avoir une influence assez conséquente sur la détermination de son centre. Si la méthode utilisée permet de réduire les défauts liés à ces aléas, les centres calculés restent parfois imprécis (hésitation entre deux positions par exemple). De plus, de part

l'utilisation de la méthode des gradients, la méthode de détection du centre des pupilles est assez robuste avec des yeux clairs (yeux bleus), mais reste moins efficace avec les yeux sombres (yeux marrons).

### 5.3.3 Limites d'utilisation

#### 5.3.3.1 Limites d'utilisation pour l'utilisateur

Les tests de validation fonctionnelle (un exemple de fiche complétée est donné en annexe I) nous ont permis de mettre au clair plusieurs limites concernant l'utilisateur de notre système. Tout d'abord, notre système reste encore trop fatigant lorsqu'il est utilisé en mode libre. En effet, les utilisateurs se sont vite fatigués à essayer de naviguer sur l'écran lorsque celui-ci n'est pas pré découpé en différentes zones. La navigation est encore difficile car trop imprécise.

D'un point de vue du confort, l'immobilité de la tête due à la mentonnière est aussi un point à améliorer. Cette immobilité forcée empêche l'utilisateur de parler ou de bouger la tête sous peine de devoir refaire une calibration. Le fait de fixer la mentonnière et l'ordinateur à la table pourrait aider à améliorer cette situation en offrant des repères parfaitement fixes.

Le champ de vision de l'utilisateur pose aussi un problème. Le fait de se concentrer sur une zone particulière de l'écran afin de cliquer dessus empêche la vision globale de l'écran.

La morphologie de l'utilisateur paraît enfin très déterminante sur les résultats obtenus. Les yeux plus clairs semblent permettre un meilleur contrôle de la souris en comparaison avec les yeux foncés. Certaines morphologies du visage rendent aussi la détection des yeux difficiles, notamment lorsque l'utilisateur observe les angles du bas de l'écran.

#### 5.3.3.2 Limites d'utilisation de l'interface graphique

Concernant l'interface graphique, la principale limite vient du fait qu'elle n'est pas encore totalement adaptée à un contrôle par notre logiciel. En effet le découpage de l'écran en différentes zones distinctes oblige à sectionner chaque onglet de façon différente. L'idéal serait donc de pouvoir faire abstraction de ce découpage et laisser un total contrôle à l'utilisateur. Cependant, la précision de notre logiciel ne le permet pas encore.

Une seconde limite est liée à la personnalisation de notre interface. L'utilisateur ne peut pas configurer lui-même le nombre de boutons ou encore les placer où il veut sur l'écran. De plus, dans le cas où l'utilisateur est tétraplégique, la personnalisation du logiciel lancé par un bouton donné devra forcément être faite par une tierce personne.

A l'heure actuelle, notre interface ne permet pas de lancer deux logiciels en même temps, ce qui pose problème pour le multi-tasking. Cependant, ce soucis pourra facilement être corrigé dans les versions futures.

La dernière limite vient du fait que même si notre interface permet de lancer un logiciel facilement, celui-ci doit aussi être utilisable à l'aide du gaze-tracking. Or la majorité des logiciels les plus courants ne possèdent pas de telles options.

Il faudrait donc créer un lot de logiciels adaptés à cette utilisation et notamment dans le cas où l'utilisateur est tétraplégique.

## Conclusion

La première partie du projet HELP (partie IS - Ingénierie Système) nous a permis de comprendre les étapes qui mènent à l'aboutissement d'un projet. En effet, afin de se faire une idée de ce qui est réalisable, nous avons tout d'abord procédé à un état de l'art. Cette étude préalable nous a permis de comprendre que de nombreuses technologies existent déjà afin d'effectuer de l'eye tracking. A la fin de cette première phase, notre choix s'est orienté vers un système composé de deux caméras. Cependant lors de la réalisation de ce prototype, nous avons rencontré certaines difficultés qui nous ont emmenés à redéfinir et simplifier notre système. Nous nous sommes ainsi rendus compte de l'importance des phases d'expérimentation et de conception, notamment pour un projet d'une telle envergure.

De plus, si la première partie du projet est importante et que la partie sur l'état de l'art permet d'étudier un panel de solution envisageable à notre système, celui-ci n'a été réellement défini que lors de l'étape de conception. En effet, cette phase nous a permis de tester certaines idées, de faire des choix plus adaptés à nos compétences et d'améliorer, raffiner la partie IS du projet.

Enfin, si nous avons revus nos exigences à la baisse durant ce projet, nous avons tout de même un démonstrateur fonctionnel et relativement robuste. De plus, la réalisation de celui-ci nous a plongés dans le domaine du gaze-tracking, encore peu exploré à ce jour. Cela nous a également permis de développer nos compétences en programmation, notamment en openCV et en C++, d'apprendre à travailler en équipe et de gagner en autonomie, grâce à la grande liberté de choix qui nous a été donnée par nos encadrants. Désormais, il serait intéressant d'embarquer notre système sur un casque ou des lunettes, solution envisagée mais que nous n'avons pas eu le temps de réaliser. Cela permettrait de résoudre les problèmes de mouvement de la tête.

# **Annexes**

## **Annexe A**

### **Plan de validation fonctionnelle**

## 1°) Chaines fonctionnelles :

Les chaines fonctionnelles que nous devrons réaliser afin de valider notre système sont les suivantes :

- Acquérir les données nécessaires à la détection du mouvement de l'œil
- Traiter les données enregistrées
- Interpréter les données pour déduire l'action à exécuter
- Agir sur l'IHM
- Rendre l'IHM ergonomique

Expression de la fonction	Id	Expression
Acquérir les données nécessaires à la détection du mouvement de l'oeil	1.1.1	Positionner l'utilisateur pour l'acquisition vidéo
	1.1.2	Filmer le visage de l'utilisateur
	1.1.4	Etre utilisable à une distance de 50cm à 3m
	1.1.5	Avoir une autonomie de 3h
Traiter les données enregistrées	1.3.1	Déetecter l'utilisateur et ses mouvements :
	1.3.1.1	DéTECTER le visage de l'utilisateur
	1.3.1.2	DéTECTER les yeux de l'utilisateur
	1.3.1.3	DéTECTER les pupilles de l'utilisateur
	1.3.1.4	DéTECTER le mouvement des pupilles de l'utilisateur
Interpréter les données pour déduire l'action à exécuter	1.4.1	Analyser le mouvement de la pupille à l'aide de la distance utilisateur-IHM
	1.4.2	Déduire l'action à effectuer
Agir sur l'IHM	1.5.1	Afficher le curseur à l'endroit regardé par l'utilisateur
	1.5.2	Actualiser l'affichage en moins de 10ms
	1.5.3	Permettre le clic gauche / la sélection
	1.5.4	Permettre à l'utilisateur de désactiver (mettre en pause) le système de détection
Rendre l'IHM ergonomique	1.6.1	Permettre à l'utilisateur de paramétrier l'IHM
	1.6.2	Rendre l'IHM adaptée au contrôle via le gaze-tracking

Ces chaines fonctionnelles devront être validées dans l'ordre vu précédemment, chaque chaîne étant dépendante de la chaîne précédente. Une chaîne fonctionnelle ne pourra être validée que lorsque chaque exigence définissant la chaîne aura passé avec succès le test de validation lui étant associé. Les scénarios liés à ces tests de validation seront décrits dans les parties suivantes.

## 2°) Tests de validation :

### 2°) 1°) Acquérir les données nécessaires à la détection du mouvement de l'œil :

#### *Exigences 1.1.1 et 1.1.2 : Positionner l'utilisateur pour l'acquisition vidéo et Filmer le visage de l'utilisateur*

**Attentes :** Ce test a pour but de connaître la position optimale pour assurer à la fois le confort de l'utilisateur et garantir le bon fonctionnement de notre système.

#### Déroulement :

- *Conditions initiales* : Laisser un sujet se placer de la façon qu'il juge la plus agréable devant notre système avec la contrainte de devoir être en position assise ou debout devant le système.
- *Lancement* : Après ce positionnement, vérifier que la caméra est capable de filmer l'intégralité du visage. Pour cela, lancer la webcam avec un logiciel tel que YouCam et vérifier que l'intégralité du visage se trouve dans la fenêtre.
- *Déroulement du test* : Vérifier aussi que les yeux du sujet sont filmés avec une résolution suffisante pour effectuer le traitement. Pour cela, faire un zoom sur l'œil et vérifier que la définition de l'image est d'au moins 18\*9 pixels.
- *Fin du test* : Vérification de la taille de l'image effectuée

**Logistique :** L'idéal serait d'avoir une dizaine de sujets à tester et de préférence de morphologies différentes. D'un point de vue matériel, nous aurons besoin d'une chaise, d'un ordinateur et de notre caméra.

#### *Sous test 1 : Vérifier le fonctionnement de la webcam.*

**Attentes :** Vérifier que la webcam est détectée par l'ordinateur et qu'elle transmet correctement les informations enregistrées.

#### Déroulement :

- *Conditions initiales* : Se munir de la webcam et allumer l'ordinateur
- *Lancement* : Brancher la webcam et attendre un éventuel son de l'ordinateur indiquant la détection de la webcam. Attendre que les périphériques soient installés.
- *Déroulement du test* : Lancer un logiciel tel que YouCam et vérifier que les images filmées par la webcam s'affichent correctement à l'écran.
- *Fin du test* : la vidéo s'affiche correctement à l'écran.

**Logistique :** Le matériel nécessaire se limite à un ordinateur Windows ou un ordinateur Mac et de la webcam.

#### **Exigence 1.1.4 : Etre utilisable à une distance de 50cm à 1m**

**Attentes :** Vérifier la robustesse de notre algorithme à des distances variables et connaître de façon précise la distance d'utilisation limite.

#### **Déroulement :**

- *Conditions initiales* : Placer le sujet à une distance de 50cm devant l'écran en position assise ou debout.
- *Lancement* : Affichage de quatre zones de couleur à l'écran que l'utilisateur devra parcourir en suivant les flèches (voir figure 2). Lancement de notre logiciel de gaze-tracking.
- *Déroulement du test* : L'utilisateur devra parcourir les zones en suivant les flèches à la distance de 50cm, et changera de zone à notre signal. Puis l'utilisateur effectuera la même opération 10cm plus loin et ce jusqu'à une distance de 1m. Les coordonnées du pointeur seront sauvegardées afin de vérifier qu'il se situe bien dans la zone que nous lui avons indiquée et ainsi estimer la robustesse de notre système.
- *Fin du test* : L'utilisateur a terminé le dernier test à une distance de 1m.

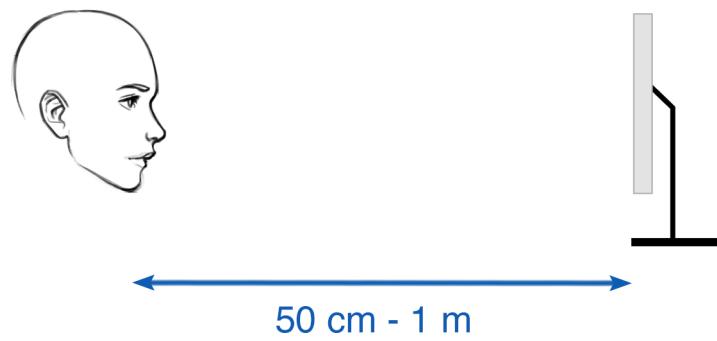


Figure 2 : Distance utilisateur-écran

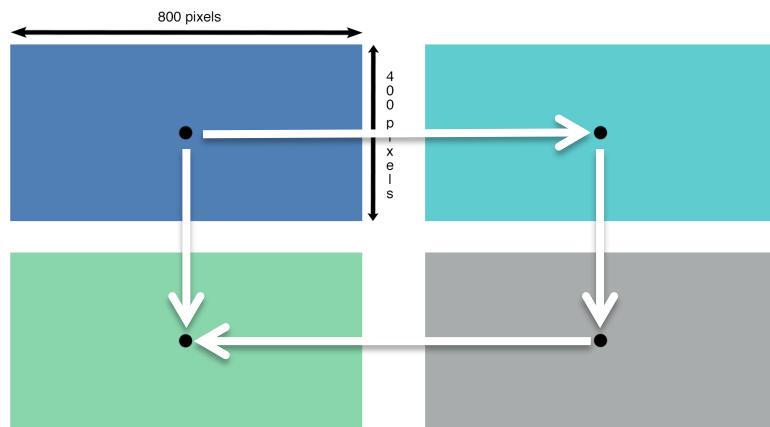


Figure 1 - Chemin à suivre

**Logistique :** Ce test devra être effectué plusieurs fois afin d'avoir un maximum de données à comparer et rendre notre étude statistique la plus fiable possible. D'un point de vue matériel nous aurons besoin d'un ordinateur et de notre caméra. L'ordinateur devra être équipé de notre logiciel

de gaze-tracking. La mise en place d'un système fiable pour mesurer la distance écran utilisateur devra être effectuée (utilisation d'un mètre).

## 2°) 2°) Traiter les données enregistrées

### Exigence 1.3.1.i : Déetecter l'utilisateur et ses mouvements

**Attentes :** Connaitre la robustesse de notre algorithme à différents niveaux (détection du visage, des yeux, d'un œil, de la pupille et des mouvements).

**Déroulement :** La validation de ce test va se faire principalement de façon visuelle. Nous allons commencer par vérifier la bonne détection du visage, puis des yeux, puis d'un œil, puis des pupilles (en incluant le suivi des mouvements de la pupille) et enfin la détection des mouvements parasites ou volontaires du sujet (par exemple les clignements).

- *Conditions initiales* : Laisser un sujet se placer de la façon qu'il juge la plus agréable devant notre système avec la contrainte de devoir être en position assise ou debout devant le système.
- *Lancement* : Lancement de notre logiciel de gaze-tracking.
- *Déroulement du test* : L'utilisateur devra simplement se placer en face de la caméra. Pour la détection du visage, un cadre devra être tracé autour du visage de l'utilisateur (voir figure 3 et 4) tant que le système parvient à le détecter.  
Des tests similaires seront réalisés pour les yeux et les pupilles. Les régions des yeux ainsi que les pupilles seront identifiées sur une image zoomée du visage. A chaque fois que notre système perdra sa cible, une valeur sera incrémentée. Cela nous permettra de faire par la suite des études statistiques en connaissant le nombre de mesures effectuées par rapport au nombre de mesures ratées.  
En ce qui concerne les clignements des yeux, il sera demandé à l'utilisateur de fermer les yeux de quelques millisecondes, ce qui correspond aux clignements normaux des yeux jusqu'à une seconde, temps choisi pour le clic.
- *Fin du test* : La détection du visage, des yeux et de la pupille a été testée.

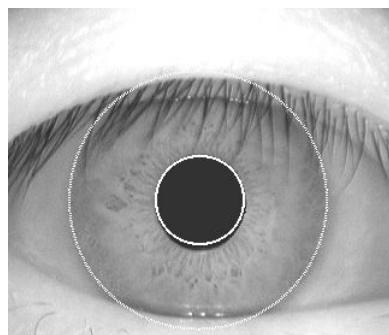


Figure 4 : Détection de la pupille



Figure 3 : Détection du visage

**Logistique :** D'un point de vue matériel nous aurons besoin d'un ordinateur, de notre caméra et d'un chronomètre. L'ordinateur devra être équipé de notre logiciel de gaze-tracking. Le chronomètre permettra de vérifier que le temps de clignement des yeux correspond au temps renvoyé par le

programme. Différents sujets pourront aussi passer le test, afin de savoir si la couleur de la peau ou des yeux peut influencer notre détection. Chacun de ces tests devra être effectué à distance variable. Des tests avec et sans lunettes (ou lentilles de contact) sont à envisager.

## 2°) 3°) Interpréter les données pour déduire l'action à exécuter

**Attentes :** Vérifier que notre logiciel détecte la fermeture des yeux et effectue les actions désirées suivant le temps de fermeture de l'œil.

**Déroulement :**

- *Conditions initiales* : Laisser un sujet se placer de la façon qu'il juge la plus agréable devant notre système avec la contrainte de devoir être en position assise ou debout devant le système.
- *Lancement* : Lancer notre logiciel de gaze-tracking, ainsi qu'une interface composée d'un bouton occupant l'écran.
- *Déroulement du test* : L'utilisateur devra fermer les yeux pendant 1 seconde afin d'effectuer le clic gauche. Le temps de fermeture des yeux de l'utilisateur sera chronométré afin de vérifier que l'action se déroule bien après 1 seconde. La même manipulation sera effectuée mais cette fois-ci pour un temps de 3 secondes afin de vérifier que le système se met bien en pause.
- *Fin du test* : la vidéo s'affiche correctement à l'écran.

**Logistique :** Nous aurons besoin d'un ordinateur et de notre caméra. L'ordinateur devra être équipé de notre logiciel de gaze-tracking.

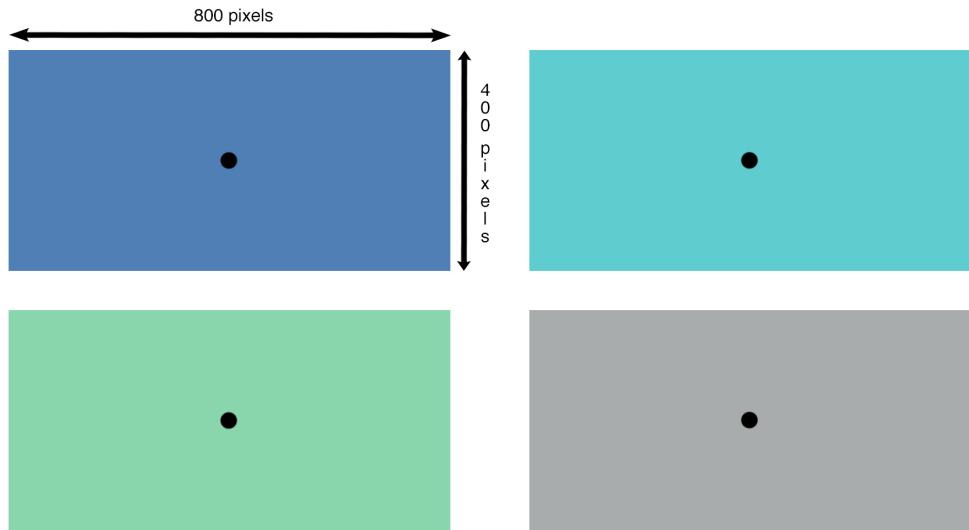
## 2°) 4°) Agir sur l'IHM

### Exigence 1.5.1 Afficher le curseur à l'endroit regardé par l'utilisateur

**Attentes :** Vérifier la précision de notre système de placement du curseur.

**Déroulement :**

- *Conditions initiales* : Laisser un sujet se placer de la façon qu'il juge la plus agréable devant notre système avec la contrainte de devoir être en position assise ou debout devant le système.
- *Lancement* : Lancer notre logiciel de gaze-tracking, ainsi qu'un schéma composé de plusieurs rectangles (voir figure ci-dessous).
- *Déroulement du test* : L'utilisateur devra regarder le centre de chaque rectangle présent sur le schéma, sans chercher à placer obligatoirement le curseur sur celui-ci. Connaissant la position exacte de chaque rectangle sur l'écran, ainsi que la position du curseur, nous serons alors en mesure de quantifier l'erreur commise par notre algorithme de détection. Cette expérience devra aussi être effectuée plusieurs fois afin de d'obtenir des résultats les plus fiables possibles.
- *Fin du test* : L'expérience a été effectuée un nombre de fois satisfaisant.



**Figure 5 : Test de positionnement du curseur**

**Logistique :** Nous aurons besoin d'un ordinateur et de notre caméra. L'ordinateur devra être équipé de notre logiciel de gaze-tracking.

#### **Exigence 1.5.2 Actualiser l'affichage en moins de 10ms :**

**Attentes :** Ce test a pour but de valider l'exigence 1.5.2 et de connaître le temps de réponse minimal.

#### **Déroulement :**

- *Conditions initiales* : Laisser un sujet se placer de la façon qu'il juge la plus agréable devant notre système avec la contrainte de devoir être en position assise ou debout devant le système.
- *Lancement* : Lancement de notre logiciel de gaze-tracking. Placer dans le code une fonction permettant de calculer le temps de calcul et d'acquisition avant le réaffichage du curseur.
- *Déroulement du test* : Faire bouger le curseur à l'aide de notre fonction et enregistrer le temps de réaffichage
- *Fin du test* : L'expérience a été effectuée un nombre de fois satisfaisant.

**Logistique :** Nous aurons besoin d'un ordinateur et de notre caméra. L'ordinateur devra être équipé de notre logiciel de gaze-tracking.

#### **Exigence 1.5.3 Permettre le clic gauche / la sélection**

**Attentes :** Ce test a pour but de vérifier qu'effectuer un clic gauche est réalisable facilement. Il permettra aussi de vérifier que le clic dans une zone particulière de l'écran est faisable.

#### **Déroulement :**

- *Conditions initiales* : Laisser un sujet se placer de la façon qu'il juge la plus agréable devant notre système avec la contrainte de devoir être en position assise ou debout devant le système.
- *Lancement* : Lancement de notre logiciel de gaze-tracking. Ainsi que de notre interface graphique composée d'un bouton.

- *Déroulement du test* : Pour effectuer ce test, nous allons placer l'utilisateur face à une interface graphique composée de quatre boutons. Il sera demandé à l'utilisateur de cliquer sur les boutons que nous lui indiquerons. Le nombre de secondes sera modifié plusieurs fois afin de voir avec l'utilisateur quelle est la période de temps la plus confortable pour naviguer. Durant ce test notre logiciel sera en mode assisté dans un premier temps, puis en mode libre. Par la suite nous placerons l'utilisateur face à une interface graphique composée cette fois-ci de neuf boutons boutons (voir figure ci-dessous). La même manipulation sera demandée à l'utilisateur pour vérifier que le clic gauche est toujours réalisable précisément avec plus de boutons.
- *Fin du test* : Le clic gauche a été testé avec un et quatre boutons.



Figure 6 - Test de clic

**Logistique :** Nous aurons besoin d'un ordinateur et de notre caméra. L'ordinateur devra être équipé de notre logiciel de gaze-tracking.

#### Exigence 1.5.4 Permettre à l'utilisateur de mettre le système en pause

**Attentes :** Vérifier que la mise en pause effectuée par notre logiciel est effective.

#### Déroulement :

- *Conditions initiales* : Laisser un sujet se placer de la façon qu'il juge la plus agréable devant notre système avec la contrainte de devoir être en position assise ou debout devant le système.
- *Lancement* : Lancement de notre logiciel de gaze-tracking. Ainsi que de notre interface graphique composée d'un bouton.
- *Déroulement du test* : Il sera demandé à l'utilisateur de fermer les yeux pendant 3 secondes et ensuite de vérifier que le clic de la souris a bien été désactivé. Un bip sonore indiquera la prise en compte du mode pause. Le nombre de secondes sera déterminé pendant le test par l'utilisateur pour garantir un confort d'utilisation optimal.
- *Fin du test* : La pause a été testée avec succès

**Logistique :** Nous aurons besoin d'un ordinateur et de notre caméra. L'ordinateur devra être équipé de notre logiciel de gaze-tracking.

## FICHE DE TEST

Exigence n°=

<b>Logistique</b>	
Personnes présentes :	
Matériel :	
Objectif(s) :	

<b>Déroulement</b>	
Prérequis :	
Résultat souhaité	Résultat obtenu

<b>Bilan</b>	
Commentaires	

## Annexe B

# Programme permettant d'acquérir un flux vidéo

Des variables globales telles que eyes\_cascade\_name ne sont pas indiquées ici. Elle n'aident pas à la compréhension du code et nécessitent certaines informations propres à l'ordinateur utilisé (chemin de répertoire par exemple).

```
1 int main(int argc, const char** argv) {
2
3     CvCapture* capture;
4     cv::Mat frame;
5
6     // Load the cascade
7     if (!eyes_cascade.load(eyes_cascade_name)){ printf(" --(!)
8         Error loading\n"); return -1; };
9     capture = cvCaptureFromCAM(0);
10    if (capture) {
11        while (true) {
12            frame = cvQueryFrame(capture);
13            // mirror it
14            cv::flip(frame, frame, 1);
15            frame.copyTo(debugImage);
16
17            // closedEyesAndClick function
18            if (!frame.empty()) {
19                bool res = closedEyesAndClick(frame, eyes_cascade);
20                printf("%s\n", res ? "true" : "false");
21            }
22            else {
23                printf(" --(!) No captured frame -- Break!");
24                break;
25            }
26            imshow(main_window_name, debugImage);
27            int c = cv::waitKey(10);
28            if ((char)c == 'c') { break; }
29            if ((char)c == 'f') {
30                imwrite("frame.png", frame);
```

```
30         }
31
32     }
33 }
34
35     releaseCornerKernels();
36
37     return 0;
38 }
```

## Annexe C

### Code permettant de récupérer le flux vidéo et de le convertir en images matricielles

```
1 CvCapture* capture;
2 cv::Mat frame;
3 capture = cvCaptureFromCAM(0);
4 if (capture) {
5     while (true) {
6         frame = cvQueryFrame(capture);
7         // mirror it
8         cv::flip(frame, frame, 1);
9         frame.copyTo(debugImage);
```

## Annexe D

### Haar\_cascades permettant de détecter le visage et les yeux

```
1 cv::String eyes_cascade_name = "CheminDossier/
    haarcascade_eye_tree_eyeglasses.xml";
2 cv::CascadeClassifier eyes_cascade;
3 cv::String face_cascade_name = " CheminDossier/
    haarcascade_frontalface_alt.xml";
4 cv::CascadeClassifier face_cascade;
5 // Load the cascades
6 if (!face_cascade.load(face_cascade_name)){ printf(" --(!)
    Error loading face cascade, please change
    face_cascade_name in source code.\n"); return -1; };
7 if (!eyes_cascade.load(eyes_cascade_name)){ printf(" --(!)
    Error loading eyes cascade, please change
    eyes_cascade_name in source code.\n"); return -1; };
```

## Annexe E

### Matrice contenant les rectangles encadrant le visage

```
1 cv::Rect detectFace(cv::Mat frame) {
2
3     std::vector<cv::Rect> faces; //vecteur contenant les
4         rectangles encadrant le visage
5
6     std::vector<cv::Mat> rgbChannels(3);
7     cv::split(frame, rgbChannels);
8     cv::Mat frame_gray = rgbChannels[2];
9     //-- Detect faces
10    face_cascade.detectMultiScale(frame_gray, faces, 1.1, 2, 0
11        | CV_HAAR_SCALE_IMAGE | CV_HAAR_FIND_BIGGEST_OBJECT, cv::
12        Size(150, 150)); //utilisation de l'haar_cascade afin de
13        remplir faces
14
15    if (faces.size() > 0) {
16        rectangle(debugImage, faces[0], 1234); // on trace un
17        rectangle autour du visage
18    }
19    else faces.push_back(cv::Rect(0, 0, 0, 0));
20
21    return faces[0];
22
23 }
```

## Annexe F

### Méthode detectMultiScale

```
1 void closedEyesAndClick(cv::Mat faceROI, CascadeClassifier
2   eyes_cascade)
3 {
4   std::vector<cv::Rect> eyes;
5   eyes_cascade.detectMultiScale(faceROI, eyes, 1.1, 2, 0 |
6     CV_HAAR_SCALE_IMAGE, cv::Size(150, 150)); // detection
7   yeux
```

## Annexe G

### Changement de repère des coordonnées du centre des pupilles dans la méthode findEyes

```
1 // change eye centers to face coordinates
2 rightPupil.x += rightEyeRegion.x;
3 rightPupil.y += rightEyeRegion.y;
4 leftPupil.x += leftEyeRegion.x;
5 leftPupil.y += leftEyeRegion.y;
```

# Annexe H

## Algorithme du gradient

```
1 cv::Mat computeMatXGradient(const cv::Mat &mat) {
2     cv::Mat out(mat.rows, mat.cols, CV_64F);
3
4     for (int y = 0; y < mat.rows; ++y) {
5         const uchar *Mr = mat.ptr<uchar>(y);
6         double *Or = out.ptr<double>(y);
7
8         Or[0] = Mr[1] - Mr[0];
9         for (int x = 1; x < mat.cols - 1; ++x) {
10             Or[x] = (Mr[x + 1] - Mr[x - 1]) / 2.0;
11         }
12         Or[mat.cols - 1] = Mr[mat.cols - 1] - Mr[mat.cols - 2];
13     }
14
15     return out;
16 }
```

## **Annexe I**

### **Fiche de validation fonctionnelle complétée**

<b>Logistique</b>
Personnes présentes :
Astrid Béranger-Fenouillet, Thomas Boulier, Benoît Raymond
Matériel :
Mentonnière, logiciel de gaze-tracking, caméra
Objectif(s) :
Vérifier que l'interface est facilement utilisable et navigable

<b>Déroulement</b>	
Prérequis :	
Fixation de la caméra sur la mentonnière, positionnement du sujet et explication du fonctionnement de notre logiciel et interface.	
Résultat souhaité	Résultat obtenu
<ul style="list-style-type: none"> <li>- Navigation possible en mode assisté avec 4 et 9 boutons</li> <li>- Navigation possible en mode non-assisté</li> <li>- Clic facile à utiliser</li> <li>- Paramétrage du temps de clic</li> <li>- Savoir si l'utilisation de notre logiciel génère de la fatigue chez le sujet</li> </ul>	<ul style="list-style-type: none"> <li>- Problème au centre avec 9 boutons</li> <li>- Bonne navigation avec 4 boutons</li> <li>- Clic pas toujours évident : fermeture des yeux nécessite trop de crispation, temps parfois trop long (à calibrer au préalable)</li> <li>- Navigation non assistée peu précise et vite fatigante (en moins de 5 min)</li> <li>- Clic facilement réalisable. Le temps pourrait même être diminué</li> <li>- Navigation parfois difficile car le curseur oscille d'un bouton à l'autre (problème de calibration). Elle reste cependant parfaitement réalisable (4 boutons)</li> <li>- L'utilisation en mode assisté n'est pas fatigante.</li> <li>- Toujours des problèmes de détection au niveau du bouton central avec les 9 boutons</li> <li>- Temps de clic un peu moins long</li> <li>- Navigation facile avec 4 boutons</li> <li>- Clic facile (mais temps trop long) avec 4 boutons</li> <li>- Pas de fatigue (4 boutons)</li> <li>- Navigation facile avec 9 boutons</li> <li>- Navigation réussie avec le mode libre, cependant c'est plus fatiguant et moins précis.</li> </ul>

**Bilan**

L'interface à 4 boutons est dans l'ensemble parfaitement utilisable.

Celle de 9 boutons reste plus difficile d'accès notamment pour les boutons centraux (ce problème peut être lié au calibrage).

Le temps du clic doit être configuré suivant l'utilisateur, certains le trouvant parfois trop long et d'autres trop court.

La navigation en mode non assisté reste difficile pour certain, mais est parfois accessible.

L'exigence est donc parfaitement validée.

**Commentaires**

Problèmes de détection des clics liés aux cils.

Le port des lunettes ne semble pas être un problème

Ajout de pastille pour faciliter la calibration

Les yeux clairs semblent donner de meilleurs résultats.

L'impossibilité de bouger la tête rend la mentonnière peu confortable et empêche de parler.

# Références bibliographiques

- [1] Maitine BERGOUNIOUX : Quelques méthodes de filtrage en traitement d'image. 2010.
- [2] Chul Woo CHO, Ji Woo LEE, Kwang Yong SHIN, Eui Chul LEE, Kang Ryoung PARK, Heekyung LEE et Jihun CHA : Gaze detection by wearable eye-tracking and nir led-based head-tracking device based on svr. *ETRI Journal*, 34(4):542–552, 2012.
- [3] El Khayati Brahim D'HONDT FRÉDÉRIC : Etude de méthodes de clustering pour la segmentation d'images en couleurs. *Faculté Polytechnique de Mons, 5ème Electricité, Certificat Applicatifs Multimédia*, 2004.
- [4] Su Yeong GWON, Chul Woo CHO, Hyeon Chang LEE, Won Oh LEE et Kang Ryoung PARK : Robust eye and pupil detection method for gaze tracking. *Int J Adv Robotic Sy*, 10(98), 2013.
- [5] Tristan HUME : eyelike. <https://github.com/trishume/eyeLike>, 2012. Accédé le 19 mai 2015.
- [6] Pupil LABS : Pupil v0.3.6 - marker tracking. <http://www.pupil-labs.com/blog/2013/12/036-release.html>, 2013. Accédé le 15 janvier 2015.
- [7] Hyeon Chang LEE, Won Oh LEE, Chul Woo CHO, Su Yeong GWON, Kang Ryoung PARK, Heekyung LEE et Jihun CHA : Remote gaze tracking system on a large display. *Sensors*, 13(10):13439–13463, 2013.
- [8] Rainer LIENHART et Jochen MAYDT : An extended set of haar-like features for rapid object detection. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 1, pages I–900. IEEE, 2002.
- [9] Noël-Arnaud MAGUIS : Rédigez des documents de qualité avec latex. <http://fr.openclassrooms.com/informatique/cours/redigez-des-documents-de-qualite-avec-latex>, 2013. Accédé le 30 août 2014.
- [10] Angèle MASSONNEAU et Nicolas BIARD : Etat de l'art des différents systèmes de pointages à l'oeil. *Dossier de la Plate-Forme Nouvelles Technologies, Assistance Publique Hôpitaux de Paris*, 2013.
- [11] Benjamin RAYNAL et Venceslas BIRI : Détection de pupille par combinaison des critères morphologiques et colorimétriques.
- [12] Jérôme SCHMALTZ : Détection des contours de la pupille à l'aide des transformées de hough.

- [13] Tobii TECHNOLOGY : An introduction to eye tracking and tobii eye trackers. <http://www.tobii.com/en/eye-tracking-research/global/library/white-papers/tobii-eye-tracking-white-paper/>, 2010. Accédé le 15 janvier 2015.
- [14] Tobii TECHNOLOGY : Tobiix2-30eyetracker. *User Manual*, 2014.
- [15] Tobii® TECHNOLOGY : Tobii glasses eye tracker. *User Manual*, 2011.
- [16] Fabian TIMM et Erhardt BARTH : Accurate eye centre localisation by means of gradients. In *VISAPP*, pages 125–130, 2011.
- [17] Paul VIOLA et Michael JONES : Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.
- [18] ZAFERSAVAS : Trackeye : Real-time tracking of human eyes using a webcam. <http://www.codeproject.com/Articles/26897/TrackEye-Real-Time-Tracking-Of-Human-Eyes-Using-a>, 2008. Accédé le 15 janvier 2015.
- [19] Marc ZAFFAGNI : Eyecharm : contrôler son pc du regard avec un capteur kinect. <http://www.futura-sciences.com/magazines/high-tech/infos/actu/d/informatique-eyecharm-controler-son-pc-regard-capteur-kinect-45381/>, 2013. Accédé le 15 janvier 2015.