

Rapport Projet

Université de Bordeaux

De l'oeil à la souris
OSS 117 : Eye Origin



Master 1 Bioinformatique
Théo Falgarone, Alexis Hubert, Tristan Maunier,
Christian Té
Date de remise : 11/05/17

Clients :

Adrien BOUSSICAULT
Pierre LACROIX

Nous remercions vivement l'ensemble des personnes ayant participé, de près ou de loin, à ce projet. Tout d'abord nous adressons nos remerciements à nos référents Adrien Boussicault et Pierre Lacroix pour leur temps, leur aide précieuse et le soutien qu'ils nous ont apporté. Nous remercions également l'Eirlab[1] qui nous a accueilli et ses membres pour leur support technique et intellectuel.

Lien vers le dépôt GitHub : <https://github.com/theofalga/OSS117EyeOrigin>

Table des matières

Introduction	1
1 Analyse du sujet	2
1.1 Contexte	2
1.2 Etat de l'art	3
1.2.1 Librairie de traitement d'images	3
1.2.2 Logiciel de traitement d'images	3
1.2.3 Logiciels de détection des yeux	4
1.2.4 Gestionnaires de fenêtres	6
1.3 Bilan	8
1.4 Besoins fonctionnels du logiciel	9
2 Conception	10
2.1 Création d'une collection d'images	10
2.1.1 Dispositif d'acquisition	10
2.1.2 Protocole d'utilisation	10
2.1.3 Logiciel	10
2.2 Récupération coordonnées de références	11
2.3 Test d'un algorithme de détection d'iris	11
2.4 Interprétation	11
2.5 Architecture	12
3 Réalisation	13
3.1 Création d'une collection d'images	13
3.1.1 Dispositif d'acquisition	13
3.1.2 Protocole d'utilisation	14
3.1.2.1 Installation du dispositif de mesure	14
3.1.2.2 Définition des points de repères sur l'écran	15
3.1.2.3 Positionnement de la caméra	16
3.1.2.4 Capture des photographies	17
3.1.3 Logiciel	17
3.2 Récupération des coordonnées de références	18
3.3 Test d'un algorithme de détection d'iris	19
3.3.1 Fonctionnement général	19
3.3.2 L'exemple de Eyelike	20
3.4 Interprétation	23
3.5 Architecture	24
4 Perspectives	25
Conclusion	26

Introduction

De nos jours l'informatique est un domaine omniprésent au sein de la société, ainsi l'accès à un matériel informatique se doit d'être aisément pour quiconque en a le besoin ou la volonté. Nous définissons l'accessibilité numérique comme étant l'accès à toutes les ressources numériques pour tous les individus quels que soient leur géolocalisation, leur langue, leur matériel ou bien leurs aptitudes physiques et mentales. Ces dernières années un mouvement s'est mis en place afin d'étendre cette accessibilité numérique au plus grand nombre, mais malgré cela l'accès à un ordinateur reste compliqué.

L'OMS[2] définit l'handicap comme étant une limitation des possibilités d'interaction avec l'environnement, cela étant dû à une déficience physiologique. Cette même limitation empêche un nombre élevé de personnes à mobilité réduite d'utiliser un ordinateur. Nous avons dû nous familiariser avec la notion d'handicap afin de pouvoir identifier avec précision les difficultés qu'implique l'utilisation d'un ordinateur par une personne à mobilité réduite et tenter d'y remédier.

Depuis 1992 la cellule PHASE[3] aide et accompagne les étudiants ayant des besoins spécifiques dans leur cursus universitaire à Bordeaux. Néanmoins cette aide, bien que très importante, implique l'intervention d'une tierce personne ce qui peut s'avérer gênant ou inadapté dans certains cas.

C'est dans ce contexte que le projet OSS 117[4] a été lancé en 2016, dans le but d'atteindre l'objectif primordial qu'est l'autonomie pour tous. Ce projet a pour but de permettre à l'utilisateur d'un ordinateur de diriger un curseur avec les yeux. Il permet également l'accès à un clavier virtuel réactif à des commandes gérées avec les yeux dans l'optique de s'affranchir de l'utilisation, parfois contraignante, d'un clavier classique.

Chapitre 1

Analyse du sujet

1.1 Contexte

Ce projet étant un projet d'ampleur, des groupes d'étudiants se relayent années après années afin de mettre en place les différentes parties qui composeront le logiciel à son état final. Nos précédents camarades ont été les premiers à travailler sur le sujet, leurs travaux constituent une base solide pour ce projet et c'est cette même base qui nous permet de prendre conscience du travail qu'il reste à faire dans les différents domaines. Il s'agit alors pour notre groupe d'identifier une problématique de travail et de faire avancer OSS 117 à notre tour.

Le bon fonctionnement d'OSS 117 passe tout d'abord par une détection précise de l'iris des yeux de l'utilisateur. Pour ce faire, en plus d'un algorithme fonctionnel, il est nécessaire d'avoir un dispositif de capture vidéo approprié. Le logiciel OSS 117 a également besoin d'interpréter les mouvements des yeux de l'utilisateur afin de pouvoir déplacer un curseur. Le dernier point essentiel d'OSS 117 est l'utilisation d'un gestionnaire de fenêtre simple d'utilisation capable d'afficher un clavier virtuel à l'écran.

1.2 Etat de l'art

Avant de pouvoir définir correctement une problématique il nous a fallu faire une étude des logiciels et librairies existants qui seraient susceptible de répondre aux attentes du projet.

1.2.1 Librairie de traitement d'images

OpenCV[5] :

Open Computer Vison est une bibliothèque graphique libre, qui permet le traitement d'images en temps réel. Elle possède des interfaces C, C++, Java et Python et est utilisable sous tous les principaux systèmes d'exploitations existants (MacOS, Windows et Linux). OpenCV est la librairie choisie par nos prédecesseurs, il semble donc logique que nous l'utilisions à notre tour.

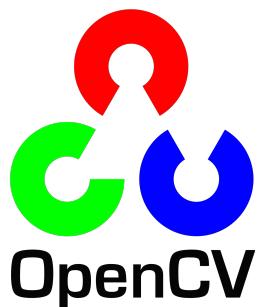


FIGURE 1.1 – OpenCV

1.2.2 Logiciel de traitement d'images

ImageJ[6] :

ImageJ est un logiciel open source de traitement et d'analyse d'images développé par National Institutes of Health. Il est écrit en Java et permet l'ajout de nouvelles fonctionnalités via des plugins et macros. ImageJ est exécuté comme une application téléchargeable sur n'importe quel ordinateur disposant d'une machine virtuelle Java 5 ou ultérieure. Des distributions pour Microsoft Windows, Mac OS X, Linux, et Zaurus sont disponibles en téléchargement.



FIGURE 1.2 – Imagej

Inkscape[7] :

Inkscape est un logiciel libre de dessin vectoriel sous licence GNU GPL. Il gère des fichiers conformes aux formats standards XML et SVG. Il est écrit en C++/C et GTK+/GTKmm.



FIGURE 1.3 – Inkscape

1.2.3 Logiciels de détection des yeux

Tobii eyeX[8] :

Ce logiciel est fourni avec une barre à placer en dessous de l'écran, dotée de deux caméras et de trois led InfraRouge permettant le suivi des yeux et de déplacer la souris à l'écran. Il est utilisé pour la navigation sur l'ordinateur et les jeux vidéos. Le logiciel est propriétaire.



FIGURE 1.4 – Tobii eyeX ~200\$

Smart Eye Pro 5.0 [9] :

Nécessite 2 à 8 caméras dotées de led InfraRouge. Conçu dans un but de recherche, il permet un suivi des yeux précis afin d'identifier les zones d'intérêts de l'utilisateur et autres informations concernant son regard.



FIGURE 1.5 – Smart Eye

Imotions[10]

Logiciel de détection des yeux qui permet de faire une cartographie des régions d'intérêt de l'utilisateur, de quantifier l'attention visuel (par un décompte de temps) ainsi que de mesurer la dilatation des pupilles. Il a pour but d'identifier les zones d'intérêts de sites internets ou dans les rayons d'un supermarché, ce logiciel a été élaboré pour un but commercial.

EyeLike[11]

EyeLike est un logiciel permettant la détection de la position de l'iris de l'utilisateur ainsi que son mouvement au cours du temps via un flux vidéo. Ce logiciel ne nécessite qu'une simple webcam pour fonctionner et son code est open source.

EyeWriter[12]

Ce projet a été réalisé dans le but de permettre à des personnes handicapées moteur de pouvoir dessiner à l'aide de leurs yeux. Le dispositif de suivi de l'oeil est composé d'une caméra et de LEDInfraRouge qui peuvent être montés soit sur une barre à placer sous l'écran, soit directement sur des lunettes. Le logiciel est divisé en deux parties, le Eye-Tracking qui s'occupe de la détection de l'oeil et le Eye-Drawing qui permet de traduire les mouvements des yeux en différents outils de dessin. Ce projet a été conçu par des informaticiens indépendants, il est facilement reproductible et représente un coût de fabrication relativement faible. Les logiciels sont open source en General Public License, ils sont donc disponibles sur le site de EyeWriter à l'instar des instructions de fabrication. Tout le projet est libre de droits.

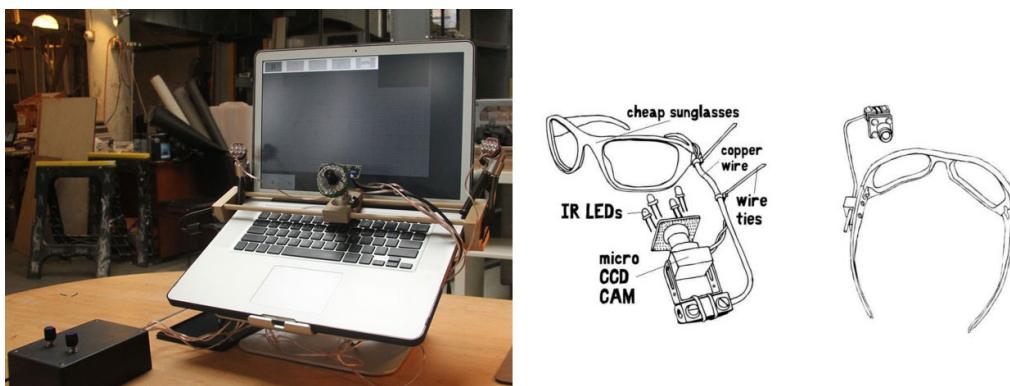


FIGURE 1.6 – A gauche : EyeWriter 2.0, A droite : Eyewriter 1.0

INSA-Strasbourg[13]

La publication date du 23 janvier 2017. L'équipe de génie électrique de l'INSA de Strasbourg a mis au point un programme sous Matlab qui effectue le déplacement du curseur de la souris grâce au suivi des yeux. Le programme effectue dans un premier temps la capture d'images via une webcam, vient ensuite la transformation de celles-ci en niveau de gris et puis la détection du visage. Par la suite il y a la détection des yeux dont ils déterminent la position, celle-ci permettant de transcrire la direction que doit prendre le curseur in fine. Leur programme étant codé sous Matlab, il ne peut être utiliser en dehors de son environnement. En effet toute interaction avec l'ensemble de l'espace de travail de l'ordinateur est impossible. Tous droits réservés INSA 2017.

1.2.4 Gestionnaires de fenêtres

Unity[14] :

Unity est l'interface système développée pour l'environnement Gnome, c'est l'interface de base pour le système d'exploitation Ubuntu.



FIGURE 1.7 – Unity

DWM - Dynamic Window Manager[15] :

DWM est un gestionnaire de fenêtres dynamique qui peut gérer les fenêtres selon trois agencements différents : tuilage (fenêtres réparties entre zone principale et zone d'empilement), flottant (position et dimension des fenêtres choisis librement par l'utilisateur) et monocle (toutes les fenêtres font la taille de l'écran). DWM est très léger (2000 lignes de codes seulement) et est facilement personnalisable grâce à la présence d'un fichier config.



FIGURE 1.8 – DWM

Gnome[16] :

Gnome est un gestionnaire de fenêtres très populaire chez les utilisateurs de système UNIX. Complet, stylisé et coloré, ce gestionnaire de fenêtre est nettement plus optimisé que ceux évoqués précédemment. Néanmoins le code est très lourd ainsi que difficilement configurable.



FIGURE 1.9 – Gnome

Xfce-Desktop Environment[17] :

Xfce est un environnement de travail que l'on retrouve sous les systèmes Unix, il est réputé pour être rapide et demande peu de ressources au système. Graphiquement simpliste, Xfce reste facile d'utilisation.



FIGURE 1.10 – Xfce

Kde[18] :

KDE est un projet de logiciel libre historiquement centré autour d'un environnement de bureau pour systèmes UNIX. L'ensemble est utilisé principalement avec les systèmes d'exploitation Linux et BSD. KDE est avec GNOME la principale alternative libre et grand public aux interfaces des systèmes d'exploitation plus répandus (c'est-à-dire Windows et Mac OS X). Ses logiciels sont généralement publiés sous la licence GNU GPL, et ses bibliothèques sous la GNU LGPL.



FIGURE 1.11 – Kde

1.3 Bilan

Suite à une réflexion collective basée sur l'analyse approfondie du sujet et des bases du projet OSS 117, nous avons décidé d'orienter notre travail autour de la détection de l'iris. Il nous semble essentiel de mettre l'accent sur cette partie, celle-ci étant la première étape cruciale au bon fonctionnement du logiciel final.

Plusieurs algorithmes existants permettent d'ores et déjà de repérer le centre de l'oeil dans une image, parmi eux nous avons étudié ceux utilisés par les logiciels EyeLike, EyeWriter et OSS 117 (qui est un algorithme issue de celui d'EyeLike).

Suite à l'apprentissage et l'étude de ces logiciels, nous avons pris conscience de l'importance de l'algorithme utilisé pour détecter l'iris de l'utilisateur de notre logiciel.

Naturellement avec l'accord de nos clients nous avons donc précisé les objectifs de notre projet. Il est indispensable de procéder à un test approfondi de plusieurs algorithmes en amont d'un choix définitif, le but de notre projet est alors de mettre au point un logiciel indépendant du logiciel final permettant de tester différents algorithmes de détection de l'iris, ce logiciel permettra à terme de définir l'algorithme qui sera le plus adapté aux demandes du projet OSS 117.

Et ainsi naquit OSS 117 : Eye Origin.

1.4 Besoins fonctionnels du logiciel

Au vu du travail effectué par nos prédécesseurs et du choix de notre problématique, nous avons pu identifier les besoins fonctionnels liés au logiciel OSS 117 Eye Origin :

1. Créer une collection d'images de visages recueillies selon un protocol prédefini. Les images doivent être utilisables pour les algorithmes qui sont testés
2. Etablir un protocole de collecte des images pouvant être reproductible. Ce protocole doit respecter les besoins suivants :
 - (a) Définir différentes positions pour la caméra
 - (b) Définir différents points de repère sur l'écran.
 - (c) Définir différents angles (verticaux et horizontaux) pour la caméra.
 - (d) Construire un dispositif permettant de placer la caméra aux différentes positions et angles prédefinis.
 - (e) Définir la position de l'utilisateur vis à vis de la caméra et de l'écran.
3. Concevoir un programme permettant à l'utilisateur, via un événement clic souris, de renvoyer les coordonnées des centres des iris de chaque image dans un fichier.
4. Concevoir un programme qui intègre un algorithme de détection de l'iris et qui retourne les coordonnées des centres des iris dans un fichier. Le programme doit être facilement modifiable pour pouvoir y integrer différents algorithmes
5. Interpréter les deux jeux de données afin de déterminer si l'algorithme fonctionne et dans quelles conditions (positions de caméra, angles). Cette interprétation est effectuée par le biai de tests statistiques propres aux traitements d'images. Et les critères d'évaluation devront pouvoir être modifiés facilement.
6. Déterminer une corrélation entre les différentes positions de la caméra (définies dans le protocole) et la bonne détection du centre de l'iris dans les images

Chapitre 2

Conception

2.1 Creation d'une collection d'images

2.1.1 Dispositif d'acquisition

La conception d'un dispositif d'acquisition est necessaire dans le but de determiner une position ideale d'une camera ou une combinaison de plusieurs cameras adaptee `a une detection permanente de l'iris d'un utilisateur. Ce dispositif doit positionner une camera, capable de prendre des photos de bonne qualite, `a differentes positions dans l'espace de travail de l'utilisateur.

2.1.2 Protocole d'utilisation

Une fois le dispositif d'acquisition cre nous devons definir une serie de parametres experimentaux qui regroupent un panel suffisamment large de conditions d'utilisation dans lesquelles l'utilisateur serait amene `a utiliser le logiciel. L'objectif est donc de realiser un protocole experimental facilement reproductible permettant d'obtenir une collection d'images consante et qualitative. Ce protocole fait `a la fois office de manuel d'utilisation du dispositif cre et de plan d'experience consacre `a la construction de la collection d'images.

2.1.3 Logiciel

Au vu du nombre important d'images que l'utilisateur doit capturer pour realiser sa collection, il est indispensable de concevoir un logiciel capable de gérer la capture des images et de les ordonner selon les differents parametres utilises.

2.2 Récupération coordonnées de références

Afin de pouvoir tester la bonne détection des iris via un algorithme, il faut dans un premier temps obtenir les coordonnées de références correspondantes à la réalité terrain, auxquelles on peut comparer par la suite les coordonnées récupérées par l'algorithme. Pour cela la collection d'images doit être traitée par un programme associé à un logiciel de traitement d'images. Ce programme doit permettre à l'utilisateur de relever les coordonnées réelles des centres des deux iris, sur chaque image, grâce à deux clics souris. En cas d'erreur ou de problème, le logiciel doit permettre à l'utilisateur de revenir en arrière lors de la sélection des centres des iris. Une fois les coordonnées récupérées, elles doivent être écrites et sauvegardées dans un fichier au format .txt . Ce fichier servira par la suite pour l'interprétation des résultats obtenus.

2.3 Test d'un algorithme de détection d'iris

Il existe de nombreux algorithmes de détection de l'iris, dont certains sont open-source, fonctionnant pour la majorité avec un flux vidéo. Or pour pouvoir être compatible avec notre collection d'images, notre programme se doit d'exécuter un algorithme à partir d'images fixes et non un flux continu d'images. Il s'agit lors de l'élaboration du programme de faire en sorte de prendre des photos à un instant donné via le flux vidéo pour répondre à cette contrainte. Les images sont analysées les unes après les autres et les coordonnées x et y de chaque œil sont enregistrées dans un fichier texte. Le programme sépare la partie de détection de l'iris de la partie d'écriture des coordonnées, ce qui lui permet d'être modifiable pour pouvoir y intégrer un maximum d'algorithmes potentiels. Les coordonnées recueillies par ce programme sont ensuite comparées avec les données de référence lors de l'interprétation.

2.4 Interprétation

Afin de déterminer si un algorithme de détection de l'iris remplit sa fonction et surtout dans quelles conditions, il est indispensable de prévoir une partie d'analyse des données décrites précédemment en partie 2.2 et 2.3. Cette analyse compare les deux jeux de données recueillis à l'aide de tests statistiques et retourne les conditions pour lesquelles la détection de l'iris est optimale et les conditions pour lesquelles elle est compromise voir impossible. Elle listera ainsi les meilleures conditions d'utilisation (positions et angles de caméra) de l'algorithme testé.

2.5 Architecture

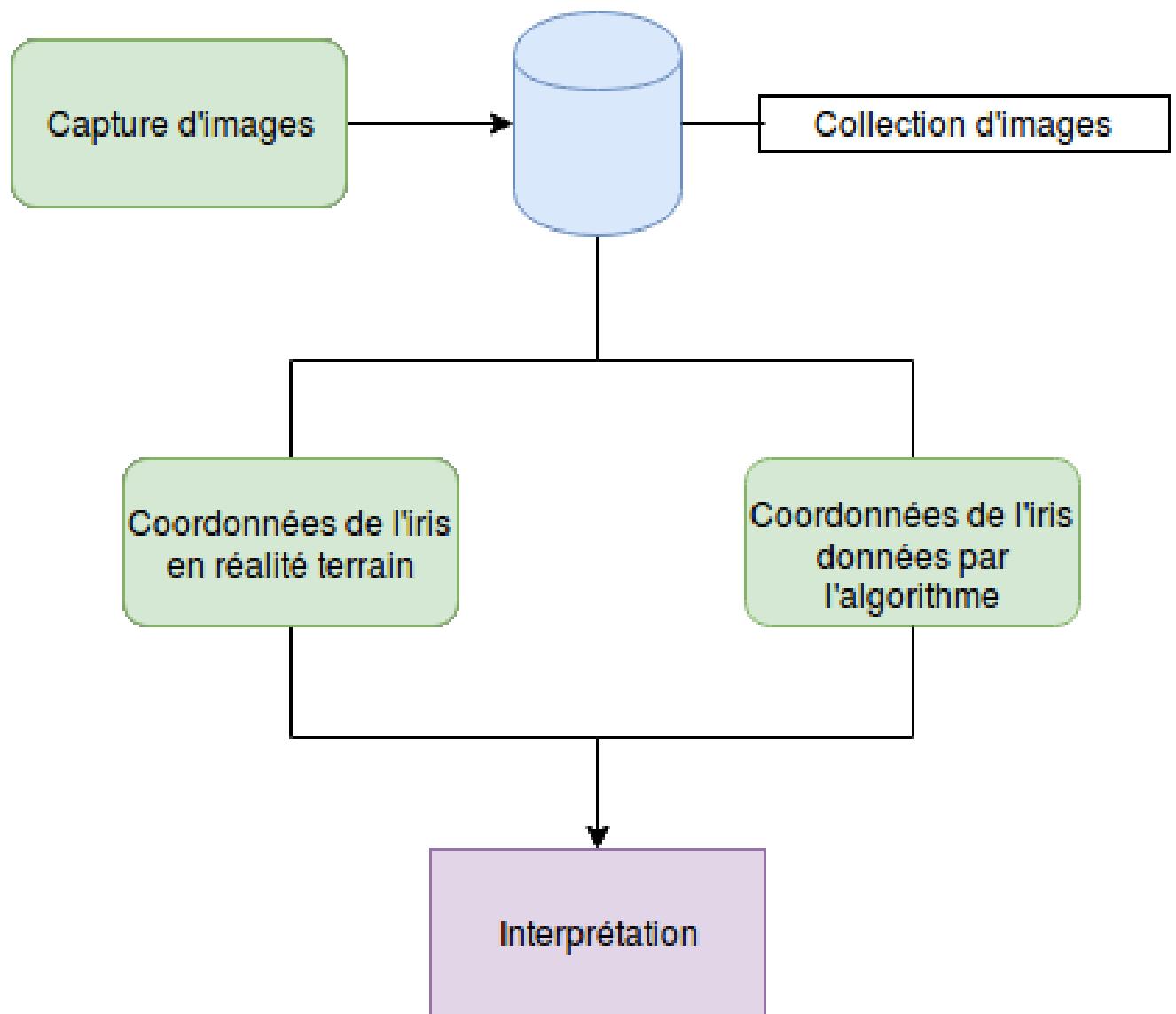


FIGURE 2.1 – Ebauche d'architecture du logiciel OSS 117 : Eye Origin

Chapitre 3

Réalisation

Pour faciliter le développement du logiciel, nous avons ajouté un fichier header protocol_constant.h contenant les variables de paramètres déterminant le protocole, à savoir le nombre de position de la caméra, le nombre d'angles verticaux, le nombre d'angles horizontaux et le nombre de points repère à l'écran.

3.1 Création d'une collection d'images

3.1.1 Dispositif d'acquisition

La réalisation d'un tel dispositif nécessite une importante réflexion préliminaire, en effet de nombreuses contraintes et paramètres sont à prendre en compte, notamment en ce qui concerne la forme du montage, le choix des matériaux ou bien encore les dimensions des différentes pièces. Chaque partie du montage final a été pensée de manière à ce que le dispositif soit le plus précis et le plus fonctionnel possible.

Nous avons eu la chance d'avoir accès à l'Eirlab[1] qui est un atelier de fabrication numérique proposant à ses membres des outils de prototypage rapide comme l'impression 3D, la découpe et gravure laser ou la gravure de circuits électroniques pour transformer une idée en prototype concret.

Eirlab est une association hébergée dans les locaux de l'ENSEIRB-MATMECA à Talence et s'adresse autant aux étudiants qu'au grand public qui partagent un espace ouvert de 400m², des machines-outils et un esprit communautaire. Il s'agit d'un lieu d'innovation et de partage où chacun peut mettre au point son idée, rejoindre un projet existant, se former puis former à son tour.

C'est grâce à cet environnement, autant via le matériel mis à notre disposition que via les personnes nous ayant conseillé et formé pour son utilisation que nous avons pu mettre au point un tel dispositif d'acquisition.

Nous avons choisi d'utiliser du bois comme matériau principal, la découpe des différents éléments à été réalisé grâce à une découpeuse laser de la marque Trotec (modèle Speedy 400 Flexx). Cette machine nous a également permis de graver des graduations précise dans le bois afin de positionner les éléments à différentes hauteurs et de les incliner de différents angles.

Le logiciel Inkscape nous a permis de dessiner les plans de chaque pièce au format .svg qui signifie Scalable Vector Graphics (ou graphique vectoriel adaptable en français). Le format svg est utilisé comme format d'entrée pour la découpeuse laser, celui ci étant basé sur des vecteurs le laser se sert des coordonnées vectorielles pour se diriger. Inkscape présente des outils de mesure précis avec lesquels nous avons pu élaborer les dessins au dizième de millimètre près et les gravures au pixel près.

Le dispositif final présente différents éléments indépendants des autres :

- une tour qui est l'élément principal du dispositif, elle est composée d'un boitier contenant une caméra et de différents mécanismes pouvant orienter ce boitier comme l'utilisateur le désire.
- un plateau servant de support à un ordinateur portable, il est assorti d'un repère afin de situer l'écran de l'ordinateur
- une planche intermédiaire marquées qui permet de placer tous les éléments les uns par rapport aux autres
- un support sur lequel l'utilisateur devra poser son menton lors de l'acquisition des photos

3.1.2 Protocole d'utilisation

Ce protocole permet d'effectuer une série d'acquisition de photos du visage de l'utilisateur d'un ordinateur. Les photographies seront prises pour 7 positions différentes de la caméra autour de l'écran, 9 points de repères sur l'écran, 3 angles verticaux et 5 angles horizontaux d'inclinaison de la caméra pour chaque utilisateur. Nous avons choisi ces paramètres afin d'obtenir une collection d'images suffisamment dense et diversifiée pour exposer des résultats les plus significatifs possibles. Ce protocole amène donc à l'acquisition d'un total de 945 photographies par utilisateur.

3.1.2.1 Installation du dispositif de mesure

Pour l'acquisition des images nous préconisons d'utiliser un plan plat et droit (une table par exemple) pour disposer les différents éléments (plateau, planche, support tête et tour) du dispositif. Le dispositif doit être installé de la manière démontré sur la Figure 3.1 ci-dessous. La planche se cale avec les pieds du plateau et les zones d'emplacement de la tour et du support tête sont délimitées par des marques sur la planche. L'utilisateur placera la tour à l'une des trois positions proposées en fonction de la position de la caméra demandée par le logiciel. Une fois les éléments convenablement disposés sur la table, placer l'ordinateur portable au centre du plateau prévu à cet effet, une ligne rouge symbolise l'endroit où l'écran de l'ordinateur doit être situé. Notez que le plateau en question permet de surélever l'ordinateur de 20cm par rapport à la table.

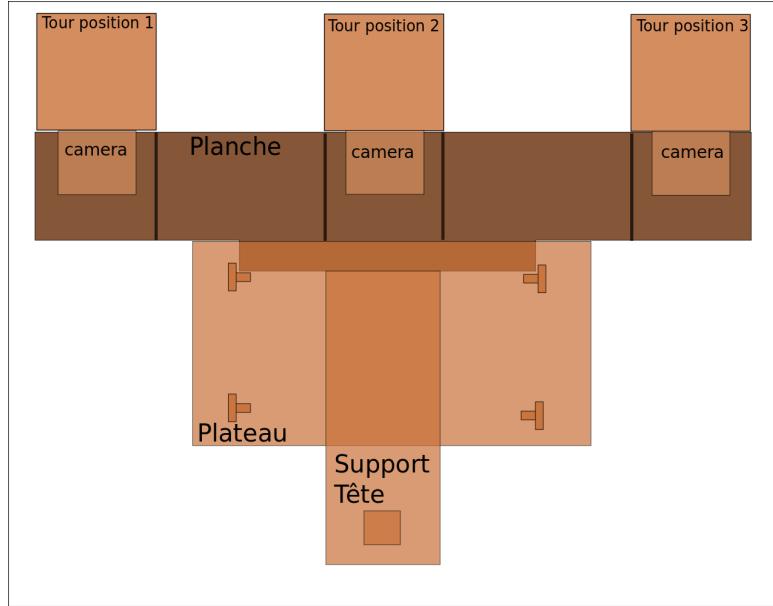


FIGURE 3.1 – Disposition des éléments du dispositif

3.1.2.2 Définition des points de repères sur l'écran

Neuf points de repères ont été placés sur une image, ils correspondent à neuf points distincts sur l'écran disposés de la manière suivante (Figure 3.2). Vous trouverez l'image suivante sous le nom points_reperes_num.png .

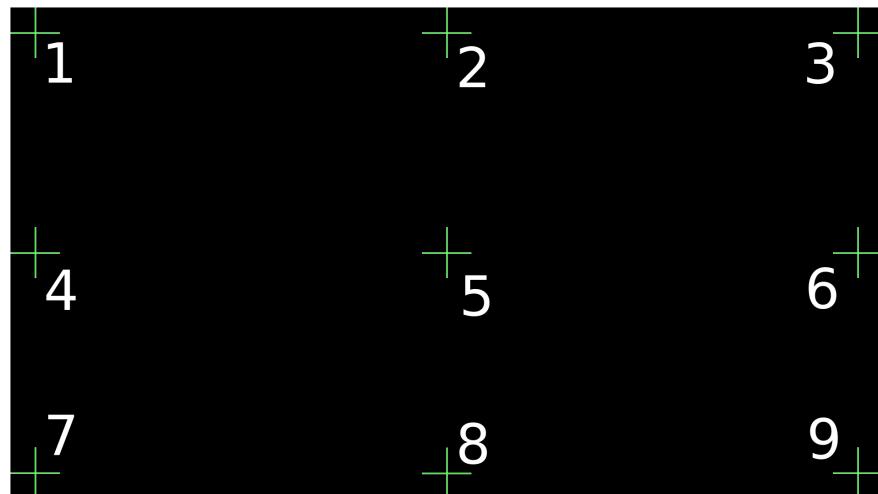


FIGURE 3.2 – Points de repères numérotés sur l'écran

3.1.2.3 Positionnement de la caméra

Sept positions de la caméra autour de l'écran de l'ordinateur ont été définies selon le schéma suivant :

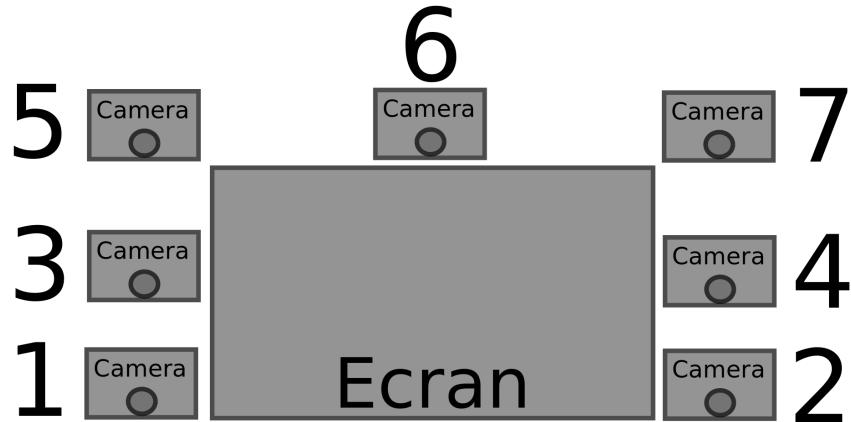


FIGURE 3.3 – Positions de la caméra autour de l'écran

Pour chaque position de la caméra , différents angles d'inclinaison horizontaux et verticaux seront testés. L'ensemble des configurations de la caméra sont répertoriées dans le tableau ci-dessous :

Position Caméra	1	2	3	4	5	6	7
Angles verticaux (degrés)	70	70	70	70	85	85	85
	75	75	75	75	90	90	90
	80	80	80	80	95	95	95
Angles horizontaux (degrés)	-20	20	-20	20	-20	-10	20
	-25	25	-25	25	-25	-5	25
	-30	30	-30	30	-30	0	30
	-35	35	-35	35	-35	5	35
	-40	40	-40	40	-40	10	40
Hauteur de la caméra sur la tour (cm)	10	10	20.5	20.5	35	35	35

TABLE 3.1 – Critères de mesure

Nous avons défini pour chaque position de caméra un angle vertical et un angle horizontal de référence. Pour chaque angle vertical de référence, on ajoute 5 puis 10 degrès et pour chaque angle horizontal de référence on enlève 5 puis 10 degrès et on ajoute 5 puis 10 degrès. Si on définit r comme angle de référence on teste donc 3 angles verticaux $r, r+5, r+10$ et 5 angles horizontaux $r-10, r-5, r, r+5$ et $r+10$.

Nous avons choisi de procéder de la manière suivante pour l'acquisition des images : angle horizontal 1 /angle vertical 1, angle horizontal 1 /angle vertical 2, angle horizontal 1 /angle vertical 3, angle horizontal 2 / angle vertical 1, etc ... et ceci pour toutes les positions de la caméra.

3.1.2.4 Capture des photographies

L'utilisateur doit se tenir assis en face de l'écran à une distance de 50cm, il doit avoir une posture la plus droite possible, le menton posé sur le support prévu à cet effet (30cm de hauteur) et ce tout le long de l'acquisition. Evitez au maximum de bouger et de faire bouger le dispositif de capture. Le logiciel indique pour chaque photo dans quelle position la camera se trouve, les angles utilisés ainsi que le point de repère à regarder. La photographie est prise suite à un événement clavier enclenché par l'utilisateur. Remarque : Compter quelques dizaines de secondes pour changer la position de la caméra entre deux séries de photos (exemple : passage de la position 2 à la position 3 autour de l'écran).

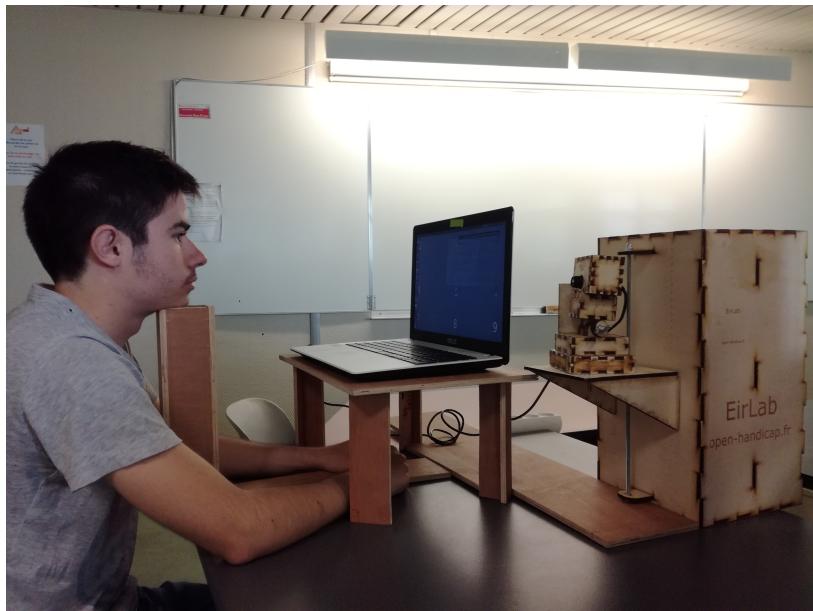


FIGURE 3.4 – Photographie du dispositif et de l'utilisateur

3.1.3 Logiciel

Pour enregistrer la capture d'images selon le protocole décrit précédemment un court programme écrit en C++[19] nommé Protocol.cpp a été conçu. En début de programme l'utilisateur doit renseigner le nom de la personne photographiée, ce qui va créer un répertoire unique portant ce nom, ce répertoire est généré par la commande bash mkdir. Chacun de ces répertoires contient les 945 photographies correspondant à l'exécution du protocole complet. Le programme génère un nom d'image exclusif via quatre boucles imbriquées ayant pour paramètres de fin de boucles respectivement le nombre de position de la caméra, le nombre d'angles horizontaux, le nombre d'angles verticaux et le nombre de points de repère à l'écran. Par exemple l'image prise à la position 3, à l'angle horizontal 2, à l'angle vertical 1 et au point de repère 9 aura pour nom 3219.png. L'image est capturée à la suite d'un évènement clavier via la fonction Videocapture de la librairie OpenCV. En cas d'erreur de capture de l'image un message d'erreur s'affiche dans le terminal et permet de recommencer la capture de celle-ci. Le logiciel affichera à l'écran pour chaque image, la position de la caméra, les angles horizontaux et verticaux ainsi que le point de repère que l'utilisateur doit fixer avant de réaliser l'événement clavier.

3.2 Récupération des coordonnées de références

Le programme permettant de récupérer les coordonnées de références des iris de la collection d'images se présente sous la forme d'un plugin ImageJ codé en Java[20]. Les images de la collection sont ouvertes dans ImageJ sous la forme d'une pile d'images (appelée stack) et les coordonnées sont écrites dans un fichier texte formaté, qui est créé dans le même dossier que la collection d'images prise en entrée. Chaque image est définie par quatre paramètres, la position de la caméra, l'angle horizontal, l'angle vertical et le point de repère correspondant. Dans le fichier texte, pour chaque image, la première ligne correspond au point de repère allant de 1 à 9. La deuxième ligne correspond au sha (identifiant unique) généré de l'image ainsi que le chemin absolu. La troisième ligne correspond aux coordonnées X et Y du centre de l'iris de l'œil gauche de l'image, puis au coordonnées X et Y du centre de l'iris de l'œil droit. Lorsque le point de repère est 1, on ajoute avant la première ligne la position de la caméra allant de 1 à 7, l'angle horizontal de 1 à 5, et l'angle vertical de 1 à 3.

Le plugin fonctionne par étapes qui sont répétées pour chaque image du stack. Tout d'abord il récupère le nom de l'image et écrit dans le fichier texte appelé "positionReference.txt" le dernier caractère de ce même nom, qui est le chiffre correspondant au point de repère. Dans le cas où il est égale à 1, il récupère le reste du nom de l'image et l'écrit avant, sachant que le premier chiffre correspond à la position, le deuxième l'angle horizontal, et le troisième l'angle vertical.

Il récupère ensuite le chemin absolu, génère le sha de l'image et les écrit dans le fichier texte. Ensuite la fonction mousePressed permet à l'utilisateur de récupérer les coordonnées du centre des deux iris en cliquant dessus. Ces coordonnées sont écrites dans le fichier positionReference.txt. Chaque clic souris ajoute 1 à un compteur initialisé à 0. Lorsque le compteur est égale à 1, il écrit dans le fichier les coordonnées du centre de l'iris gauche à l'aide de la fonction `save`. Lors du second clic, le compteur passe à 2, il écrit les coordonnées du centre du deuxième iris grâce à `save`, engendre le passage à l'image suivante du stack ainsi que le retour à 0 du compteur et l'incrémentation de la variable `frame`.

Le plugin permet également à l'utilisateur de revenir à l'image précédente, dans le cas où il se serait trompé. En effet un clic du milieu(centrale ou roulette) de la souris de l'utilisateur permet le passage à l'image précédente et appelle la fonction `deleteLine`. Cette fonction supprime les trois dernières lignes du fichier en copiant le fichier dans une `arraylist` et réécrivant le fichier avec le contenu de l'`arraylist` moins les trois derniers éléments qui correspondent aux trois dernières lignes du fichier d'origine.

3.3 Test d'un algorithme de détection d'iris

3.3.1 Fonctionnement général

Le programme permettant d'analyser une collection d'images avec un algorithme de détection de l'iris a été conçu de manière modulable, on distingue deux parties fonctionnant de concert. La première partie écrite dans un fichier de sauvegarde au format .txt, la deuxième est la partie propre à l'algorithme testé qui retourne les coordonnées x et y du centre de l'iris de chaque oeil.

La collection d'image est répertoriée selon le nom de la personne ayant effectué le protocole de capture des images. En début de programme l'utilisateur doit renseigner le nom de la personne dont il souhaite utiliser les images qui seront traitées par l'algorithme. Le programme se charge de générer le chemin absolu pour retrouver la collection d'images via le nom renseigné par l'utilisateur et l'exécution de la commande bash `pwd`. Le nom des images est généré par une succession de quatre boucles `for` reprenant comme paramètres de fin de boucles respectivement le nombre de positions de la caméra, le nombre d'angles horizontaux, le nombre d'angles verticaux et le nombre de points de repère affichés à l'écran. Par exemple l'image prise à la position 3, à l'angle horizontal 2, à l'angle vertical 1 et au point de repère 9 aura pour nom `3219.png`. Le nom de l'image est ensuite additionné au chemin absolu pour permettre l'ouverture de celle-ci via la fonction `imread` de la librairie openCV créant ainsi une matrice. Cette matrice est intégrée en paramètre dans la partie de détection de l'iris qui retourne les coordonnées des iris, cette partie étant propre à chaque algorithme de détection nous ne rentrerons pas dans les détails, nous donnerons tout de même un exemple dans la sous-partie suivante.

L'écriture dans le fichier suit la même logique tout du long, premièrement on écrit sur une première ligne la position de la caméra, suivi de l'angle horizontal, puis de l'angle vertical. Sur la ligne suivante on écrit le point à l'écran puis sur une troisième ligne le chemin absolu de l'image et le sha1, celui-ci étant généré par la commande bash `sha1sum`. Sur une quatrième ligne on inscrit les coordonnées x et y des iris de l'oeil gauche puis de l'oeil droit. L'opération est répétée à chaque changement de position de caméra et/ou d'angles horizontal et/ou vertical.

3.3.2 L'exemple de Eyelike

Afin d'illustrer et de tester le bon fonctionnement de l'ensemble de notre architecture, nous avons choisi d'implémenter un programme de détection de l'iris, Eyelike. Ce programme est celui utilisé par nos prédecesseurs dans le projet OSS 117, il nous a donc semblé pertinent pour l'avenir de ce projet de tester cet algorithme en premier.

Eyelike est un code développé par Tristan Hume[22], écrit en C++, open source, qui permet la détection de l'iris utilisant la librairie OpenCV. Initialement, ce programme traite une image issue d'un flux vidéo, nous avons dû modifier le média d'entrée pour que ce soit une image fixe provenant de notre collection d'images et non plus un flux vidéo.

En premier lieu, l'image est transformée par l'algorithme des "descripteurs de Haar", cette méthode de détection d'objet dans une image numérique a été proposée par Paul Viola et Michael Jones dans leur article "Rapid object detection using a boosted cascade of simple features"[21]. Dans le cas où un visage est bel et bien présent dans l'image, un carré délimite ce dernier. Ensuite, une étape de recherche des yeux à l'intérieur de ce carré est effectuée. Il existe de nombreux classificateurs des yeux qui permettent de réaliser ces étapes mais une méthode plus rapide a été mise au point dans Eyelike. En effet les yeux sont positionnés dans la même zone que le visage, on détermine alors une zone ayant la plus forte probabilité de contenir les yeux. Pour ce faire un carré est dessiné en se basant sur un pourcentage de largeur et hauteur du visage.

La dernière étape de Eyelike est de déterminer le centre de l'iris, pour ce faire est utilisée la méthode des gradients de Fabian Timm décrite dans son article "Accurate eye center localisation by means of gradients"[23]. Cet algorithme définit le centre de l'iris (pupille) comme la position où la majorité des vecteurs de gradients de l'image s'entrecoupent. Le gradient d'une image est le changement directionnel d'intensité des pixels dans une image. Pour illustrer cela sur la figure ci-dessous, les vecteurs de gradients sont représentés par des flèches bleues pointant vers la zone la plus sombre, c'est à dire ceux avec la plus haute intensité. Ce schéma illustre ce qui se produit avec la pupille d'un œil.

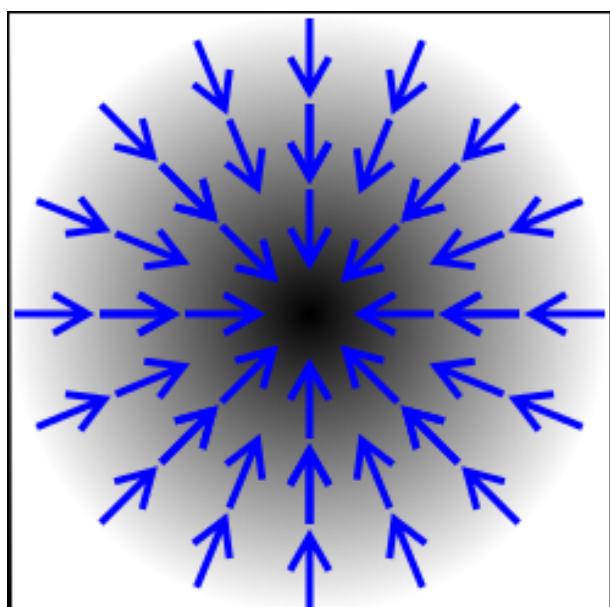


FIGURE 3.5 – Schéma des gradients

La première étape de l'algorithme de Fabian Timm est le calcul des gradients en x et en y de chaque pixel de l'image avec les formules suivantes :

$$\begin{aligned}\Delta_x f &= \frac{(x+1,y) - f(x-1,y)}{2} \\ \Delta_y f &= \frac{(x,y+1) - f(x,y-1)}{2}\end{aligned}$$

La norme de chaque gradient est calculée avec la formule suivante :

$$\sqrt{gX^2 + gY^2}$$

où gX et gY sont les calculs de gradients en X et en Y. Les gradients avec une norme significative sont sélectionnés pour la suite pour diminuer la complexité de calcul, c'est à dire en ignorant les gradients des régions homogènes de la sclère.

Ensuite, l'algorithme applique une fonction mathématique tel que décrite par Fabian Timm, qui calcule la probabilité c^* qu'un point de l'image soit le centre optimal de l'iris :

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

FIGURE 3.6 – Formule pour calculer le centre optimal de l'iris

Cette fonction va tester chaque point de l'image comme un centre possible et vérifier si son vecteur distance (d) (entre le centre testé et la position de départ du gradient) est aligné avec les vecteurs gradients (g) de l'image comme illustré sur l'image ci-dessous :

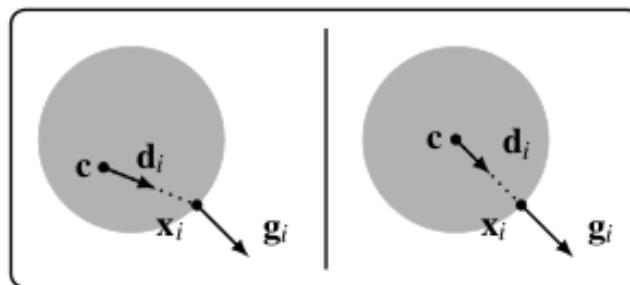


FIGURE 3.7 – Schéma représentant l'iris en gris et la sclère en blanc. A gauche le vecteur de déplacement d_i et le vecteur gradient g_i n'ont pas la même orientation, alors qu'à droite ils sont alignés.

L'alignement entre les deux vecteurs est calculé en effectuant le produit scalaire de ces deux vecteurs. Les normes des vecteurs gradients et des vecteurs distances sont toutes normalisées afin que la valeur résultante du produit scalaire dépende uniquement de l'alignement de ces deux vecteurs. Plus il y a de gradients alignés avec le centre testé, plus sa puissance c^* augmente et donc plus la probabilité que ce centre soit le centre optimal de l'oeil est forte. Enfin, puisque la pupille est en général plus sombre que la sclère et la peau, un poids est appliqué pour chaque centre possible tel que les centres sombres aient plus de chance d'être le centre optimal de l'iris que les centres clairs. Ceci permet d'éviter que le centre de l'iris soit confondu avec des zones sombres proche de l'iris, comme les cils ou les sourcils. En intégrant ce poids à la fonction précédente, on obtient la fonction suivante :

$$\arg \max_{\mathbf{c}} \frac{1}{N} \sum_{i=1}^N w_{\mathbf{c}} (\mathbf{d}_i^T \mathbf{g}_i)^2$$

FIGURE 3.8 – Formule pour calculer le centre optimal de l'iris en intégrant le poids

Après application de cette formule chaque point de la zone défini comme étant celle de l'oeil obtiendra donc une valeur c^* représentant sa probabilité d'être le centre optimal de l'iris. Les coordonnées du point possédant la plus grande valeur c^* seront les coordonnées du centre de l'iris.

3.4 Interprétation

Une fois les coordonnées des iris sauvegardées dans des fichiers .txt, une étape d'interprétation des données obtenus par le plugin ImageJ et l'algorithme testé est nécessaire. Pour ce faire nous avons conçu un programme en C++ dont le but est de parcourir les deux fichiers .txt, pour ce faire nous utilisons la méthode ifstream présente sous C++. Afin de faciliter l'analyse des résultats, une classe "camera" a été implémentée. Cette classe est définie par une position de caméra, un angle horizontal, un angle vertical et deux tableaux de booléen (un pour chaque œil) renvoyant true si l'iris est correctement détecté et false pour chaque point de repère à l'écran, il renvoie false sinon. On considère que le centre de l'iris est correctement détecté si la différence entre les coordonnées x et y retournées par l'algorithme testé et les coordonnées x et y de références est inférieur à 3 pixels. Au fur et à mesure que les fichiers sont parcourus, des objets camera sont créés et stockés dans un vecteur.

Malheureusement, à cause d'un trop faible nombre d'utilisateur et par manque de temps nous n'avons pas pu développer une analyse statistique poussée et correct. Ceci pourra être poursuivi à l'avenir et constitue une analyse plus fine que celle existante, il sera alors possible de déterminer les conditions optimales pour la détection de l'iris. En revanche, le programme actuel permet d'afficher l'ensemble des caméras avec le pourcentage de détection correcte pour les deux yeux. Il est aussi possible de n'afficher que les caméras ayant 100% de détection correcte aux deux yeux. Enfin, une dernière fonction retourne les couples de caméra compatible. Un couple de caméra est défini comme compatible, si pour chaque point de repère à l'écran au moins une des deux caméras a une détection correcte, pour l'œil gauche et l'œil droit.

3.5 Architecture

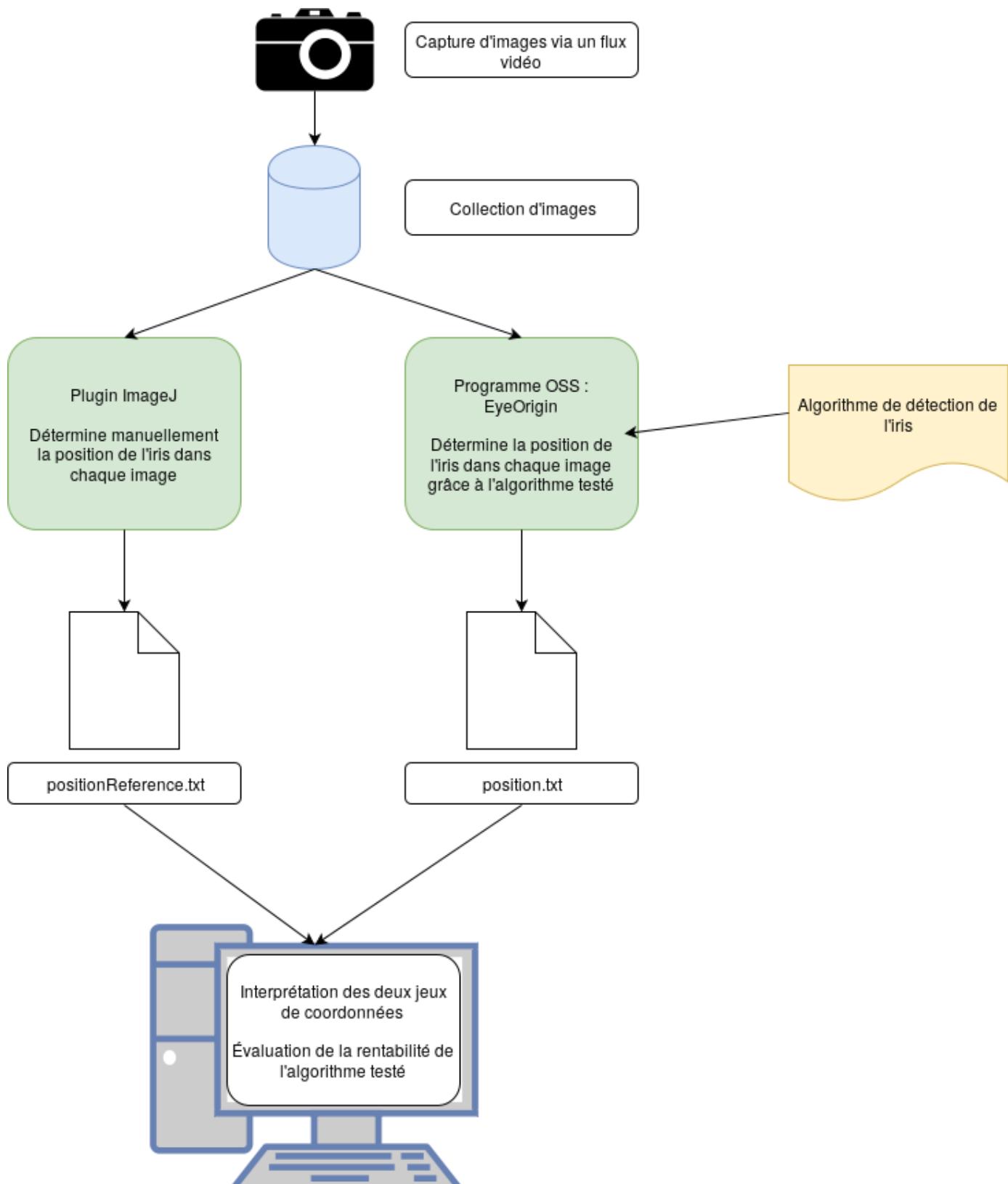


FIGURE 3.9 – Architecture finale du logiciel OSS 117 : Eye Origin

Chapitre 4

Perspectives

Le projet OSS 117 : Eye Origin bien que fonctionnel en l'état, possède quelques points qui pourraient être optimisés. Tout d'abord il est nécessaire de tester des algorithmes de détection d'iris autre qu'Eyelike, ce qui est tout à fait possible dans l'état actuel du code mais qui n'a pas pu être fait par manque de temps. Ensuite l'écriture des coordonnées pourrait être effectuée dans des fichiers au format xml en lieu et place des fichiers texte actuels. Cela permettrait de rendre le programme plus modulable, dans le but de pouvoir rajouter différents paramètres et/ou angles facilement. Il serait alors également plus aisément d'exploiter les résultats avec des logiciels différents.

D'autres paramètres d'acquisition des images peuvent également être ajoutés tels que, la distance entre l'écran et l'utilisateur, l'utilisation d'un écran plus grand ou la luminosité (intensité et direction d'éclairage). L'utilisation de LED infrarouge et donc d'un dispositif d'acquisition capable de capter l'infrarouge muni d'un filtre de la lumière visible pourrait permettre une meilleure distinction de l'iris via un contraste augmenté. La qualité de la vidéo influe directement sur les résultats obtenus, en effet meilleure la qualité sera, plus fiables seront les coordonnées obtenues par l'algorithme testé.

Il semble important pour confirmer la robustesse des résultats issues de l'interprétation d'effectuer une étude statistique autour de ces résultats. Pour se faire, il faudrait augmenter le nombre d'utilisateurs et ainsi augmenter le nombre d'images traitées, afin d'obtenir une répétabilité de l'expérience suffisamment élevée. De plus l'étape de récupération des coordonnées de référence doit être réalisée par différentes personnes pour nullifier l'impact de l'erreur humaine. Une fois les biais statistiques écartés il sera possible de comparer l'efficacité des différents algorithmes en fonction des différents paramètres testés. On pourrait alors déterminer quel algorithme utiliser pour le projet OSS 117, si une ou plusieurs caméras doivent être utilisées et dans quelles disposition.

Une fois l'ensemble de ces pré-requis mis en place, la détection de l'iris du logiciel OSS 117 sera parfaitement fonctionnelle, il s'agira ensuite pour les groupes suivants de développer le reste du programme capable d'interpréter les mouvement de l'iris et de pouvoir utiliser le clavier virtuel grâce à un gestionnaire de fenêtre adapté.

Conclusion

OSS 117 est un projet immense qui implique que différents groupes d'étudiants se succèdent années après années, chaque groupe apportant sa pierre à l'édifice afin de faire avancer le projet jusqu'à son terme. Une longue étape d'analyse du sujet, de compréhension du travail déjà effectué et de maîtrise des différents logiciels, librairies et outils utilisés a été nécessaire avant de pouvoir dégager une problématique de travail claire et précise.

Au cours de ce travail de longue haleine il nous a fallu mettre en place une répartition des tâches, une gestion du temps et une communication indispensable au bon déroulement de celui-ci. Au vu de la taille conséquente du projet, nos superviseurs nous ont laissé le choix de la thématique sur laquelle on voulait travailler. Malgré la difficulté que cela représentait, nous avons fait le choix de se pencher sur la détection de l'iris. Ce choix nous a amené à explorer de très nombreux domaines liés de près ou de loin à l'informatique, parmi ceux là on retrouve l'optique, l'électronique, l'ingénierie ou encore la programmation orientée objet.

Ce projet aura été une expérience difficile mais très enrichissante, nous avons réussi à remplir la majorité des objectifs que nous nous étions fixés et nous nous sommes construit une véritable cohésion de groupe. Nous en garderons un souvenir positif et nous encourageons les prochains étudiants à poursuivre le projet OSS 117.

Bibliographie

- [1] EirLab, le FabLab High-Tech de l'ENSEIRB-MATMECA - Bordeaux INP
- [2] OMS, Handicaps, <http://www.who.int/topics/disabilities/fr/>.
- [3] Karine Verdeau, Cellule phase,<http://www.u-bordeaux.fr/Universite/Organisation/Administration/>
Formation-Insertion-Professionnelle-et-Vie-Universitaire/Direction-de-la-vie-universitaire/Service-PHASE.
- [4] Auteurs : Julien Estebeteguy, Lisa Perus, Alexandre Petit, Thomas Riquelme
- [5] OpenCV, <http://opencv.org/>.
- [6] ImageJ, <https://imagej.nih.gov/ij/>.
- [7] Inkscape, <https://inkscape.org/fr/>.
- [8] tobii, <http://www.tobii.com/tech/products/>
- [9] Smart Eye, <http://smarteye.se/>
- [10] imotions, <https://imotions.com/>
- [11] Tristan Hume. eyelike : A webcam based pupil tracking implementation. <https://github.com/trishume/eyeLike>.
- [12] EyeWriter, <http://www.eyewriter.org/>
- [13] INSA-Strasbourg, <http://genie-electrique.insa-strasbourg.fr/projet-ge-alternance-eye-tracking/>, 21 janvier 2017
- [14] Unity. <https://doc.ubuntu-fr.org/unity>.
- [15] Suckless. Dwm. <http://dwm.suckless.org/>.
- [16] Gnome. Gnome. <https://www.gnome.org/>.
- [17] Xfce, Olivier Fourdan , 1996
- [18] Kde, Projet KDE, 1996
- [19] C++,Bjarne Stroustrup,1983
- [20] Java, <https://www.oracle.com/fr/java/index.html>.
- [21] Michael Jones Paul Viola. Rapid object detection using a boosted cascade of simple features. In Accepted conference on computer vision and pattern recognition, 2001.
- [22] Tristan Hume. eyelike : A webcam based pupil tracking implementation. <https://github.com/trishume/eyeLike>.
- [23] Erhardt Barth Fabian Timm. Accurate eye center localisation by means of gradients. In International Conference on Computer Vision Theory and Applications, 2011.