Première année de Master

mention Informatique, Mathématiques et leurs Applications

spécialité Génie Informatique

2004 - 2005

Projet de recherche

Développement d'une application

Où est passé Charlie ?

avec faceLAB™

encadré par Vincent Courboulay

UNIVERSITÉ DE LA ROCHELLE *Antoine Jullien & Gaël Ducerf*

Antoine Jullien & Gaël Ducerf

Projet de recherche

Développement d'une application

« Où est passé Charlie ? »

avec FaceLab™

encadré par Vincent Courboulay

Première année de Master
Mention Informatique, Mathématiques et leurs Applications
Spécialité Génie Informatique

Université de La Rochelle

Année scolaire 2004 - 2005

Remerciements

Nous tenons à adresser nos remerciements aux personnes qui nous ont permis de réaliser ce projet de recherche.

Nous remercions tout particulièrement :

Vincent Courboulay, notre encadrant, pour sa disponibilité et pour nous avoir guidé tout au long de ce projet de recherche.

Les enseignants-chercheurs qui encadraient les autres projets concernant faceLAB™, à savoir, Aymeric Blondel, Michel Ménard, et Bertran Vachon

L'ensemble des étudiants travaillant sur les autres projets faceLAB™ pour leur collaboration.

Résumé

Dans un premier temps, nous avons effectué un travail de recherche en ce qui concerne l'état de l'art du suivi de regard, au niveau de la recherche, des domaines d'application et des solutions proposées sur le marché.

Puis nous nous sommes plus particulièrement penchés sur la prise en main du système de suivi de regard faceLAB™ en réalisant les tutoriaux proposés dans la documentation.

Il nous a ensuite paru nécessaire de réaliser des tests de précision de faceLAB™ dans tous ses modes de fonctionnement. Le développement d'un programme s'est révélé nécessaire à la réalisation de ces tests.

Une fois les résultats de ces tests obtenus, nous les avons exploités pour déduire la meilleur configuration de faceLAB™ pour l'application *Où est passé Charlie ?*.

La partie applicative de nos travaux, à savoir le jeu *Où est passé Charlie ?,* n'a pas été réalisée. Cependant le programme de test pourra facilement être réexploité dans ce but.

Abstract

Initially, we looked for the gaze tracking state in research, application domains and market solutions.

Then we initite ourselves about the faceLAB™ gaze tracking system by carrying out the tutorials proposed in the documentation.

After that, it appeared it was necessary to carry out faceLAB™ precision tests with all its operating modes. The development of a program was necessary to this tests realization.

Once the tests results were obtained, we exploited them to deduce the best configuration from faceLAB™ for the *Où est passé Charlie ? (Where's Waldo ?)* application.

The applicative part of our work *Où est passé Charlie ? (Where's Waldo ?)* was not realized. However the test program could easily be reexploited to this end.

Sommaire

Introduction

Ce rapport est l'aboutissement d'un semestre de recherche effectuée dans le cadre de l'unité d'enseignement *Projet de Recherche* de première année de master.

Nous nous sommes formés à l'utilisation du logiciel de suivi de regard faceLAB pour en exploiter les fonctionnalités.

Nous allons vous conduire - en passant par la présentation du projet *Où est Charlie ?* - au coeur de notre sujet de projet, à savoir, la configuration de faceLAB et la réalisation des tests nécessaires à son exploitation.

Avant tout ceci, nous vous proposons la lecture d'une présentation de faceLAB.

Présentation de faceLAB

Développé par la société Seeing Machines **[1]**, FaceLAB est un système vidéo d'acquisition et de suivi de visages et de regards. Il s'agit d'un système fournissant, en temps réel, des mesures sur :

- la position et l'orientation de la tête
- la direction du regard
- le comportement oculaire

Toutes les mesures sont effectuées depuis deux cameras positionnées face à l'utilisateur.

FaceLab permet ainsi de s'affranchir du mode laboratoire en laissant le sujet se comporter naturellement et avec une grande liberté de mouvement. La robustesse de la capture des informations permet de mettre en œuvre FaceLab dans des conditions d'éclairage et des environnements extrêmement variés : simulateur, véhicules, postes d'observation, surveillance de comportements d'enfants, etc.

Présentation du sujet

Le sujet s'intitule *Développement d'une application « Où est passé Charlie ? » avec faceLAB™*.

Ce sujet de projet de recherche vient s'intégrer dans le cadre actuel des recherches du laboratoire L3i qui s'intéresse particulièrement à l'analyse du comportement. Trois sujets ont été proposés parallèlement à celui-ci à trois autres groupes d'étudiants avec qui nous devions collaborer.

Dans un premier temps nous avons dû prendre en main faceLAB™ afin de comprendre l'ensemble de ses modes de calibration, la signification des paramètres extraits, ainsi que l'environnement virtuel 3D.

Puis, nous avions à tester la précision des données fournies en temps réel par faceLAB™ dans ses différents modes de fonctionnement.

Suite à cela, nous devions numériser des planches de la bande dessinée « Où est Charlie ? » et saisir manuellement les coordonnées de Charlie sur chaque planche afin de pouvoir mettre en oeuvre le jeu *Où est Charlie ?*.

Le jeu consiste à projeter une des planches devant le joueur et à mesurer le temps nécessaire à celui-ci pour y retrouver Charlie. La direction du regard du joueur doit donc être suivie en temps réel et comparée à la position de Charlie. La comparaison doit être suffisamment robuste pour détecter que le joueur s'est bien fixé sur Charlie. Lorsque le joueur a ainsi gagné, l'application communique son temps et une nouvelle planche est proposée.

État de l'art

La première étape de notre travail a consisté à se renseigner sur les différentes techniques existantes pour le suivi de regard, les matériels sur le marché et les logiciels existants. Nous avons par la même occasion pu nous rendre compte de l'étendue des utilisations possibles d'un système de suivi de regard comme faceLAB.

1) Les techniques d'enregistrement du mouvement des yeux pour le suivi de regards [13] [14]

Les systèmes d'enregistrement du suivi du regard ont beaucoup évolué au cours des dernières années. Les systèmes sont ainsi devenus plus souples et surtout moins contraignant et moins invasifs pour les participants. Il existe à l'heure actuelle trois techniques d'enregistrement des mouvements des yeux qui divergent selon le procédé d'enregistrement et par conséquent la précision des mesures et les contraintes assumées par les utilisateurs.

- **La technique électro-oculographique (EOG)** :

  L'EOG est une technique ancienne établie par Fenn & Hursh qui permet de mesurer les différences de potentiels électriques induits par la rotation des yeux. Ces potentiels électriques sont captés par des électrodes placées autour des yeux. Bien que la mesure ait une résolution temporelle correcte, elle permet difficilement de connaître la position réelle du regard et donc de peu d'utilité dans les études ergonomiques.

- **La technique galvanométrique (*Scleral search-coil technique)* :**

  Le principe est de créer un champ magnétique et de repérer à l'intérieur de ce champ les variations d'un signal électrique traversant une lentille spéciale posée sur l'œil du sujet. Le sujet est placé à l'intérieur d'un champ magnétique créé par trois bobines disposées horizontalement, verticalement ou latéralement. La position du regard est donc repérée sur ces trois dimensions. La technique est très précise mais également très contraignante (un ophtalmologiste doit être présent à chaque passation) ce qui limite sévèrement son utilisation à des fins psychologiques ou ergonomiques pour évaluer des interfaces.

- **La technique du reflet cornéen (Pupil-Centre/Corneal Reflection : PPCR, limbus tracking)** :

  Le principe consiste à envoyer au centre de la pupille une lumière infrarouge émise par une diode ou un ensemble de diodes. Le reflet infrarouge renvoyé par la cornée de l'œil est ensuite détecté et ce sont les variations d'intensité de ce reflet qui permettent, après calcul, de repérer le centre de la

pupille et de connaître la position de l'œil. Il existe de nombreux systèmes fondés sur ce principe qui utilisent soit un détecteur optique pour capter le reflet infrarouge, soit une caméra vidéo qui filme l'œil. La technique se prête facilement à des tests ergonomiques car il n'y a aucune contrainte physiologique du sujet (tête libre), la caméra est placée sous l'écran et seulement une courte phase de calibrage est requise. De plus, le système enregistre le diamètre pupillaire qui peut servir d'indicateur pour estimer la charge mentale induit par un écran ou le stress visuel. La technique du reflet cornéen enregistré par une caméra vidéo est celle qui est la plus utilisée car elle fournit des données suffisamment précises pour des contraintes expérimentales réduites.

2) Les utilisations du suivi de regard : [25][26][27]

Elles sont nombreuses, et dans des domaines de recherche très divers, comme par exemple :

- la psychologie **[23] [24]**
- la neurologie
- l'ophtalmologie
- l'ergonomie (notamment du web, de magazines) **[13][14][18] [19] [20]**
- la publicité **[13][14][18] [19] [20]**
- étude de la perception visuelle
- études sur la lecture
- études sur les enfants (notamment autistes : **[16] [17]**)
- études sur l'effet de l'apesanteur sur l'organisme **[28] [29]**
- le regard comme souris **[22]** (utile pour les handicapés)

3) Les matériels sur le marché

- Eyelink Gazetracker (SR Research Ltd., Mississauga, Ontario, Canada) **[4]** :
  Ce système très performant monté sur casque utilise deux caméras haute vitesse pour suivre les deux yeux simultanément. Une troisième caméra suit les 4 marqueurs infrarouges montés sur le dispositif d'affichage des stimuli visuels.

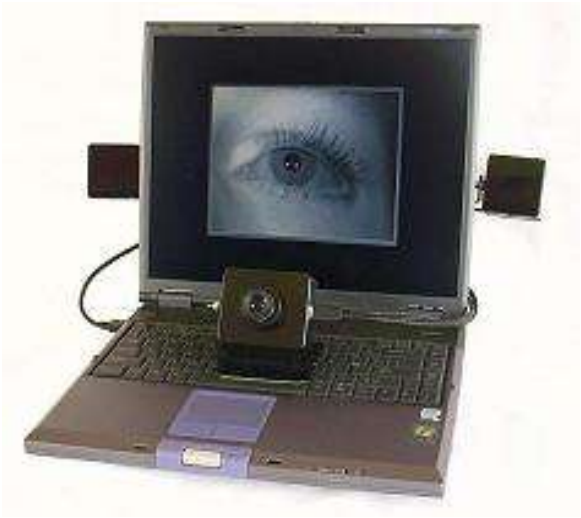
- EyeGaze (LC Technology : Fairfax) :
  Un système simple constitué d'une caméra à placer sous l'écran est facile à utiliser et d'assez bonne qualité. EyeGaze utilise aussi l'infrarouge pour un meilleur suivi.

- Tobii 1750 Eye-Tracker **[2][3]** :
  Développé par la société Tobii Technology, c'est un système de suivi de regard qui n'interfère pas du tout dans l'environnement de l'utilisateur puisque même la caméra et les diodes NIR (Near Infra-

Red) sont intégrés dans un écran plat.

Tobii 1750 ET

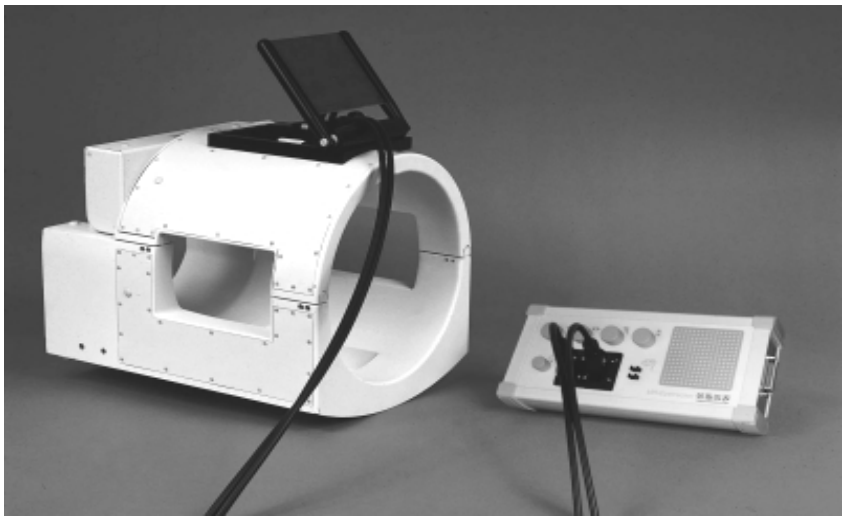
- Quick Glance 2 :
  proposé par la société EyeTech Digital Systems, c'est un système de remplacement de la souris par un dispositif léger, montable sur un écran ou un ordinateur portable. Il utilise des diodes infrarouges.



Quick Glance 2

- MR-Eyetracker **[5][6][7]** :
  C'est un système de suivi de regard spécifiquement conçu pour une utilisation en environnement d'imagerie médicale, et notamment en IRM. Il a été développé par le département de Neurologie de l'Université de Fribourg, en Allemagne. Il n'utilise pas de caméra, mais des diodes infrarouges et des

photodiodes (capteurs). Il peut être utilisé en combinaison avec le *VSG2/5 Visual Stimulus Generator* et le logiciel *vsqEyeTrace* (cf 4. les logiciels existants) pour la stimulation visuelle et l'acquisition de données sur PC.



MR-Eyetracker

- Skalar IRIS **[8]**:
  Ce sytème utilisant l'Infrarouge est conçu spécialement pour une utilisation pour la recherche oculomotrice. Produit et distribué par *Cambridge Research Systems*, il peut mesurer précisément les mouvements horizontaux et verticaux, importants ou très faibles. Il est compatible avec le générateur de stimuli visuels *ViSaGe* et constitue avec *vsgEyeTrace* un système complet.

Skalar IRIS

- ASL (Applied Science Laboratories), Technology and Systems for Eye Tracking : **[9]**
  propose également plusieurs suiveurs de regard. Voir [8].

- Canon Auto Eye-Tracking: **[10]**

propose des systèmes de suivi de regard automatiques et performants pour les ophtalmologues.

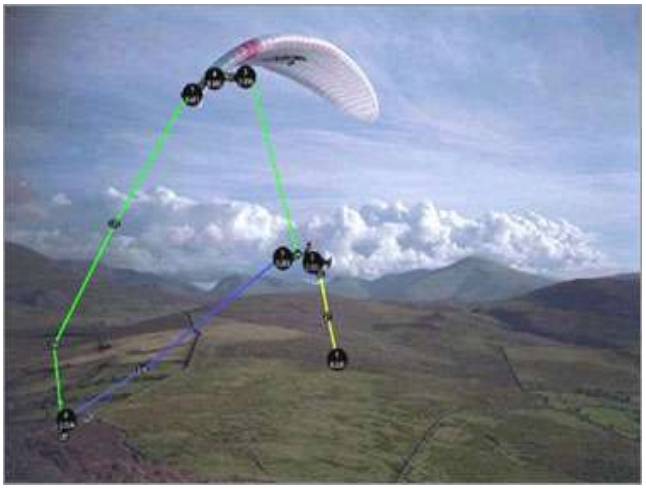
4) Les algorithmes utilisés et les logiciels existants :

La plupart des algorithmes utilisés sont à base de segmentation d'images fournissant des résultats acceptables la plupart du temps lorsque les conditions d'utilisations (par exemple l'illumination) sont idéales. Les diodes infrarouges permettent bien sûr un meilleur suivi car fournit un repérage plus fiable de la pupille des yeux. Pour un système vraiment robuste, pouvant opérer dans des conditions d'illumination très différentes et avec n'importe quel utilisateur, les algorithmes peuvent intégrer et utiliser les connaissances sur les mécanismes de l'oeil, et un historique de la position de l'oeil. L'algorithme peut de plus être adaptif, pour s'ajuster rapidement à chaque utilisateur et aux conditions d'éclairage, sans avoir besoin de réglages coûteux en temps (comme c'est le cas pour les algorithmes les plus simples).

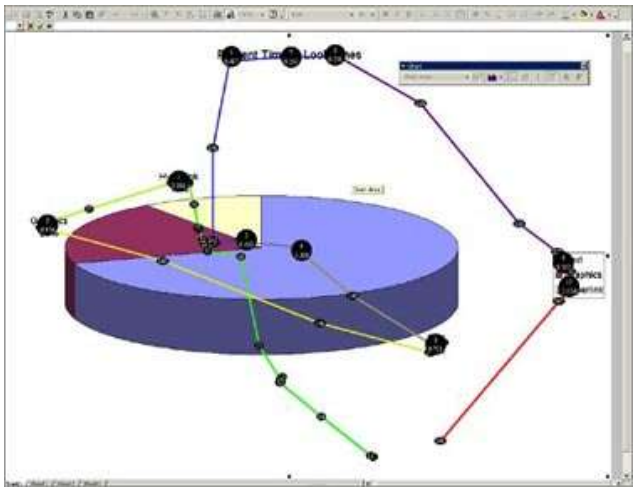
Les logiciels d'acquisition pour les systèmes de suivi de regard sont en général partie intégrante du système complet. Il y a donc peu de logiciels de ce type sur le marché. En revanche, les logiciels d'études du mouvement de l'oeil sont plus nombreux. Ils intègrent des fonctions comme la génération de stimuli visuels, l'analyse de données issues des caméras, la visualisation de ces résultats, etc.

Dans cette catégorie de logiciels, on peut retenir *GazeTracker (a tool for studying eye movement)* **[11]** qui offre 3 fonctions principales :

- La présentation de stimuli / l'enregistrement, incluant :
  - l'analyse d'images fixes
  - l'analyse d'interfaces logicielles (et sites web)
  - analyse de vidéos
- La synchronisation d'informations
- L'analyse et visualisation de données.



GazeTracker : analyse d'image fixe



GazeTracker : analyse d'interface logicielle

Un autre nom revient assez souvent : vsgEyetrace [http://www.crsltd.com/catalog/eyetrace/], qui est un logiciel pour l'acquisition l'enregistrement et de visualisation de données issues de systèmes de suivi de regard. Ce logiciel est souvent associé aux générateurs de stimuli VSG2/5 et viSaGe, qui sont des cartes compatibles avec les systèmes Skalar IRIS et MR-EyeTracker, par exemple.

Tobii propose aussi son logiciel ClearView 2, pour l'analyse du regard. **[12]**

Configuration de faceLAB

La deuxième partie de notre projet a consisté à prendre en main faceLAB, à nous familiariser avec l'environnement 3D proposé, à apprendre à créer des modèles, à effectuer la calibration des caméras.

Pour commencer, nous avons découvert faceLAB à l'aide des tutoriels proposés par Seeingmachines, comme nous l'ont conseillés nos encadrants. Ces tutoriels, ainsi que la documentation dans laquelle ils se trouvent, sont importants pour bien configurer faceLAB. La documentation contient de nombreuses informations utiles pour bien calibrer le système et ainsi avoir les meilleures performances possibles. Les tutoriels permettent de découvrir les différents modes d'utilisations et configurations possibles du système.

Pour une description détaillée de faceLAB, de son utilisation et ses différentes configurations, se reporter au rapport commun.

Etude de la précision

L'objectif de départ de ce projet était de développer une application de jeu "Où est Charlie?", en utilisant un système de vision par ordinateur, en l'occurrence faceLAB. Avant de se lancer dans la programmation de l'application, il était nécessaire de tester les limites de faceLAB, afin s'assurer que faceLAB offre une précision suffisante pour ce type d'application, et de fixer les conditions d'utilisations du jeu. Plusieurs questions se posaient : Quel est le mode d'utilisation adéquat ? Quelle précision du suivi de regard faceLAB offre t-il ? Le joueur doit-il se trouver près des caméras, ou un peu plus éloigné ? Pour tenter de répondre à ces questions, nous avons réalisé une étude de la précision du suivi de regard en deux temps. Premièrement, en utilisant l'outil de calibrage de l'intersection du regard avec l'écran : le Screen Intersection Display, qui offre des tests intéressants pour mesurer la précision du regard sur l'écran. Dans un deuxième temps, en vue de l'objectif de suivre le regard de l'utilisateur sur une planche de "Où est Charlie", nous avons fait des mesures de précision sur le mur, à l'aide d'un programme de tests.

Sur l'écran avec le Screen Intersection Display (SID)

1. Présentation du SID :

Voir le *Récapitulatif des tutoriels concernant la configuration et l'utilisation du système,* à la section 6. *Calibrer l'intersection du regard avec l'écran.*

2. Les paramètres des tests :

L'objectif de ces tests étant de déterminer les modalités d'utilisation adéquates à notre problématique, nous voulions tester l'impact sur la précision du suivi de regard des paramètres suivants :

- configurations des caméras : champ large (*wide field of view*), classique (*classic*), ou regard précis (*precision gaze*)
- type de modèle : tête complète (*full head*) ou seulement de face (*front only*)
- distance entre l'utilisateur et les caméras : 50 cm, 1m, 2m .

Nous avons fixé le mode *IR & visible,* qui utilise à la fois la lumière visible et infrarouge.

3. Les résultats :

Modèle *Front Only :*

- *Mode champ large :*

  Ce mode n'étant pas créé pour le suivi de regard, il n'offre pas de calibrage du regard, et par conséquent, le SID est indisponible. Ce mode n'offrant pas une bonne précision du suivi de regard, il ne convient pas aux exigences de notre problématique. Dans le placement actuel des caméras, le système perd le regard de l'utilisateur dès que ses yeux se trouvent à plus de 60 cm des caméras.

- *Mode classique :*

  Ce mode constitue un compromis entre précision du suivi de regard et robustesse aux mouvements latéraux de la tête. En effet, le mode champ large permet à l'utilisateur une grande liberté de mouvements, alors que le mode précision implique que l'utilisateur reste attentif à ne pas sortir du champ de la caméra. La qualité du suivi de regard est assez bonne lorsque l'utilisateur est proche des caméras. En revanche, si l'on s'éloigne à environ 1 mètre des caméras, les résultats sont très instables ; le regard est détecté assez loin des points affichés sur l'écran, et à des endroits très variables d'une mesure à l'autre. A partir de 1,10 m, le regard est perdu.

- *Mode regard précis :*

  C'est le seul mode où la qualité du suivi de regard peut atteindre le niveau maximum de 3. Les résultats des tests de précision sont donc bons, mais ce mode contraint l'utilisateur à ne pas trop bouger la tête, comme nous l'avons vu précédemment. Puisque dans ce mode, l'une des deux caméras zoome sur le visage de l'utilisateur et détecte les pupilles grâce à l'IR, le regard est plus stable en s'éloignant que dans le mode classique. La détection des pupilles devient instable à partir d'à peu près 1m10, et elles sont perdues à partir d'1m20. Le visage est totalement perdu quand l'utilisateur est à plus de 2m des caméras.

Modèle *Full-Head :*

Par manque de temps, nous n'avons pas pu effectuer les tests nécessaires pour la comparaison avec le mode *front only*. Nous pouvons cependant supposer que les résultats n'auraient pas été très différents, puisque c'est surtout sur la liberté de rotation de la tête que ce mode apporte une amélioration. Notre application ne demandant pas à l'utilisateur de tourner largement la tête, et la configuration d'un mode Full-Head étant beaucoup plus longue et fastidieuse, il nous a semblé plus judicieux de choisir le mode *front only*.

Tableau des mesures effectuées : (voir annexe III)

Vous remarquerez que les distances à laquelle les tests ont été effectués sont différentes pour le mode précision, car le zoom trop important de la caméra ne permettait pas de se trouver aussi près.

Bilan :

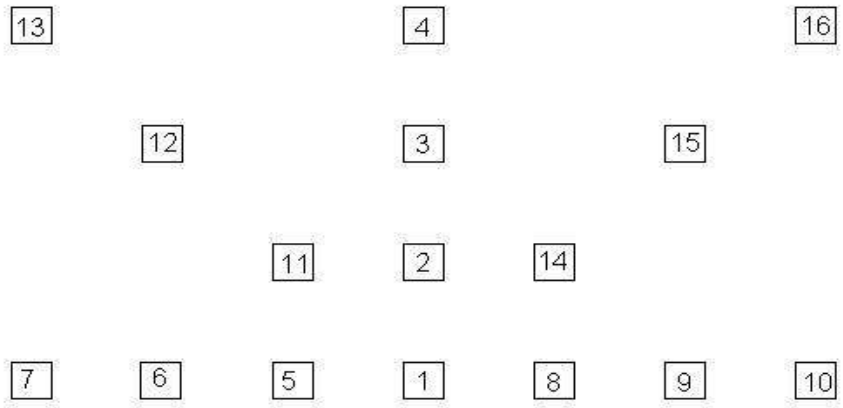
Le mode classique offrant une précision acceptable et étant moins contraignant pour le joueur, semblait à ce moment être le plus approprié.

Sur le mur avec un programme de test

L'objectif du projet étant de suivre le regard d'un joueur sur une planche de "Où est Charlie" projetée, il était nécessaire de tester la précision de faceLAB quand l'utilisateur regarde une zone extérieure à l'écran, par exemple le mur situé derrière.

1. Préparation des tests :

Dans le but de mesurer la qualité du suivi de regard sur le mur, nous avons fixé des carrés de papier numérotés sur le mur à des endroits déterminés, aussi précisément que possible, comme le montre le schéma suivant :



disposition des points tests sur le mur

La disposition de ces points de repères permettra de tester le suivi de regard à la fois horizontalement et verticalement.

Les coordonnées spatiales du point test 1 dans le repère du monde de faceLab sont :

x = 0; y = 50 cm; z = -106cm.

Le z correspond à la distance séparant le mur de l'origine du repère (entre les 2 caméras, au niveau du support). Il est donc égal à 106cm pour tous les points. Les distances horizontale et verticale sont toujours de 25 cm entre les points voisins.

Ainsi, le point 2 a pour coordonnées : x = 0; y = 75cm; z = -106cm,

le point 6 : x = -50cm, y = 50cm , z = -106cm,

le point 14 : x = 25cm, y = 75 cm, z = -106cm. (voir annexe II pour toutes les coordonnées).

En reprenant un peu le principe du SID, nous voulions mesurer l'erreur de distance moyenne et l'écart-type entre la position réelle du point fixé par l'utilisateur et la position du regard donnée par faceLAB. Pour cela, nous avons développé un programme de test

2. Développement des programmes de tests (voir annexes V, VI et VII) :

But :

Le but de ce programme est de pouvoir faire des tests de précision de suivi de regard sur un objet. Ici nous voulons suivre le regard d'une personne sur un mur or l'interface de faceLAB™ ne permet d'effectuer ces opérations que sur un écran.

Principe :

Le principe est, dans un premier temps, d'enregistrer la position du regard donnée par faceLAB™ sur un objet pendant une certaine durée. Dans un second temps on compare l'enregistrement à la position réelle du regard sur l'objet. On peut ainsi obtenir la différence moyenne entre la position réelle du regard et la position donnée par faceLAB™. On calcule également l'écart-type entre les mesures effectuées par faceLAB™ ce qui nous indique leur stabilité.

Réalisation :

Après l'étude de la documentation, nous avons repris le squelette « netclient » fourni dans le « Client Tools SDK », le kit de développement de faceLAB™. Ce squelette en langage C++ est décomposé en trois fonctions. La première est la fonction main() qui appelle en boucle la seconde fonction appelée handleDatagram(). Cette fonction récupère les données extraites et envoyées sur le réseau par faceLAB™ et appelle la troisième fonction handleEngineOutputData() dont le rôle est de traiter les données récupérées. Notre travail a donc été d'écrire cette troisième fonction.

On a donc 5 principales étapes dans notre fonction de traitement :

1 - On récupère le numéro de la frame

```
int frame_num = output_data.frameNum();
```

2 - On récupère le nom de l'objet sur lequel le regard est posé

```
string object_name =
      output_data.worldOutputData()->
      gazeIntersectionOutputData()->
      intersectionObjectName();
```

3 – On récupèration des coordonnées du regard sur l'objet

```
vector<float> intersection =
      output_data.worldOutputData()->
      gazeIntersectionOutputData()->
      gazeObjectIntersection();
```

4 - On décompose les coordonnées en x, y, z

```
float x = intersection[0];
float y = intersection[1];
float z = intersection[2];
```

5 - On formate et affiche les informations

```
printf( "\r%20s -> %10d :  x =%7.3f, y =%7.3f, z =%7.3f ",
      object_name.c_str(), frame_num, x, y, z );
```

Ensuite nous avons ajouté l'enregistrement de ces mesures sur un nombre de points donné. Ce qui nous permet donc de calculer la déviation, la déviation moyenne et la déviation standard.

La déviation est la différence entre la position du regard extraite par faceLAB et la position réelle.

La déviation moyenne est la moyenne des déviations calculées précédemment.

La déviation standard est l'écart-type des déviations.

3. Déroulement des tests et résultats :

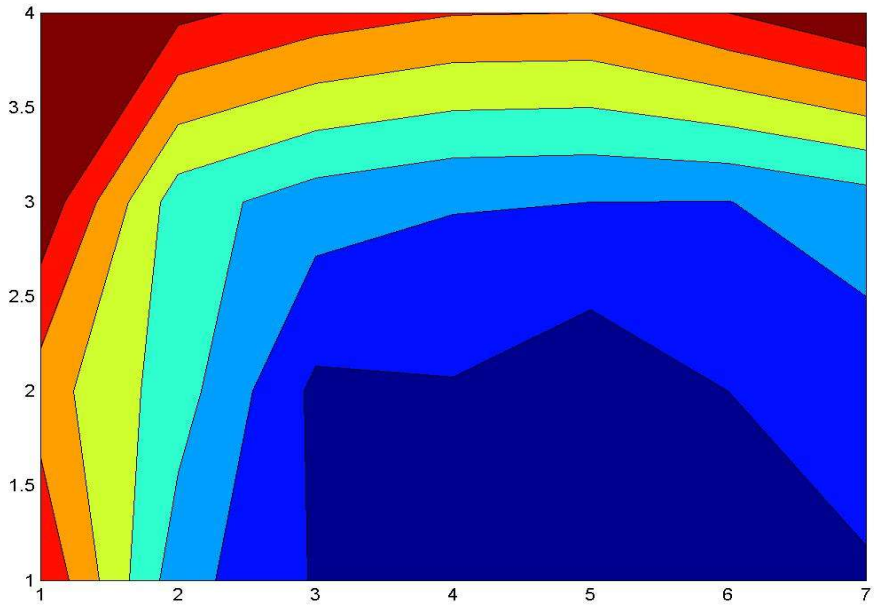
Principe :

L'utilisateur a configuré un modèle front only, et lance le troisième programme de test, qui permet de calculer sur 180 mesures, l'erreur moyenne et l'écart-type de la position de regard quand il regarde un point numéroté sur le mur. L'utilisateur fixe un point choisi sur le mur, et lance le programme avec les coordonnées du point à tester en paramètres.

Résultats (voir annexe IV) :

Lors des premiers tests que nous avons effectués, nous nous sommes rendus compte que le mode classique n'était finalement pas approprié, car dans ce mode, faceLAB ne donne pas les coordonnées du regard sur un objet. Nous avons donc fait les tests en mode précision.

Voici une représentation des résultats de ces mesures d'erreur de distance, faite grâce à la fonction *contourf* de Matlab. Nous avons tout d'abord effectué une moyenne des quelques valeurs pour chaque point, pour n'avoir qu'une valeur par point. Les données que nous avions ne formaient pas une matrice 4*7 complète (4 lignes, 7 colonnes, correspondant à l'emplacement des points sur le mur) car il n'y a que 16 points tests. Pour former une matrice complète, nous avons comblé les "trous" par des valeurs intermédiaires constituant une moyenne des points alentour.



visualisation des erreurs de distance sur le mur

On peut remarquer que les résultats sont plus rapidement mauvais du côté gauche que du côté droit. Nous avions remarqué, lors des tests, que le suivi de l'oeil gauche était plus mauvais que celui de l'oeil droit. FaceLAB perdait la pupille gauche lorsque l'oeil tournait trop à gauche, ce qui entraînait une baisse significative de la qualité du suivi. Nous n'expliquons pas réellement ce phénomène car la configuration du modèle était a priori bien faite, la calibration des caméras aussi. Malheureusement, nous n'avons pas eu le temps de nous attarder sur ce point et d'effectuer d'autres tests pour en comprendre l'origine.

Conclusion :

Ces tests nous ont d'abord permis de déterminer la configuration appropriée (mode précision) pour l'utilisation d'une application de type "Où est Charlie?". La précision obtenue semble acceptable pour ce genre d'application, il faudra penser à définir la taille de la zone autour de Charlie dans laquelle on considère que le joueur a trouvé Charlie. Cette taille pourra être déterminée grâce aux résultats des tests effectués sur le mur, et devra sans doute être choisie en fonction de l'endroit où se trouve Charlie. En effet, si Charlie se trouve sur un bord de l'image, la zone devra être plus grande que s'il se trouve au centre, compte tenu de la baisse de précision lorsque le regard s'éloigne du centre.

Déroulement du projet

Durant le mois de mars, nos encadrants ont suggérés que nous préparions un planning prévisionnel de l'avancement du projet. A cette date, les objectifs du projet et les différentes tâches à accomplir étaient bien définis. Cependant, il était relativement difficile de prévoir la durée à affecter à chaque tâche. De plus, il était prévu au départ que les créneaux horaires pour le projet de recherche soient de plus en plus nombreux au fil du semestre, pour arriver jusqu'à une semaine complète réservée pour ce projet. Finalement, nous n'avons pas eu plus de temps accordé à ce projet en fin d'année qu'en début de semestre. Ceci a donc constitué une difficulté pour suivre le planning de départ que voici, sous forme de diagramme de Gantt :

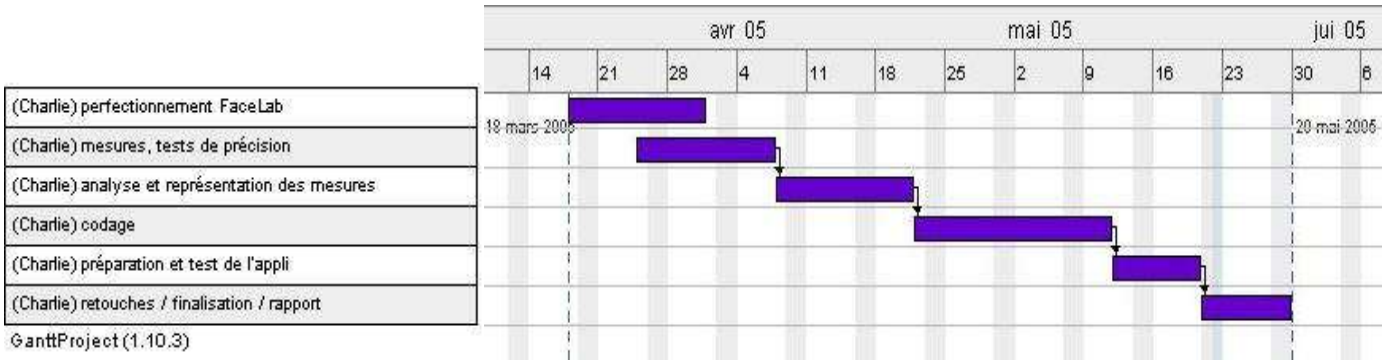


diagramme de Gantt

A la fin du projet, nous avons atteint la plupart des objectifs de départ. Il n'y a pas de jeu "Où est Charlie?", mais le but principal de ce projet était surtout de savoir si faceLAB permettait ce genre d'application, et si oui dans quelles conditions d'utilisations, avec quelle configuration. La partie où nous avons pris du retard est celle des mesures et tests de précision. Elle a finalement duré jusqu'à la fin du projet. Nous n'avions pas prévu qu'elle serait aussi longue à mettre en oeuvre (notamment pour les mesures sur le mur), et nous avons perdu un peu de temps à faire des tests sans résultats concrets (par exemple sur l'affiche située derrière le système faceLAB, et avec le vidéoprojecteur). De plus, par soucis de précision, nous avons effectué de nombreuses calibrations et créations de nouveaux modèles coûteuses en temps. Ceci dit, nous avions prévu un temps plus que nécessaire au codage de l'application, et nous avons donc pu rattraper un peu de notre retard sur cette partie. Enfin, en s'appuyant sur le programme de tests que nous avons développé, il serait possible de réaliser un petit jeu dans le genre de "Où est Charlie?" assez rapidement.

Conclusion

Nous avons donc réalisé notre projet en quatre étapes : la recherche d'information concernant l'état de l'art du suivi de regard, la prise en main de faceLAB™, les tests de robustesse de faceLAB™, et enfin, l'exploitation de ces résultats et déduction du meilleur mode de fonctionnement de faceLAB™.

La partie applicative de nos travaux, à savoir le jeu *Où est passé Charlie ?,* n'a pas été réalisée.

Ce projet a été enrichissant pour plusieurs raisons. Tout d'abord parce qu'il nous a permis de découvrir un système suivi de regard, et l'étendu de ses utilisations possibles. Ensuite, parce qu'il nous a permis de nous initier à la gestion de projet, et au travail collaboratif.

L'avenir du projet

Le programme réalisé pour la mise en place des tests de faceLAB™ pourra facilement être repris et modifié pour la création du jeu "*Où est passé Charlie ?"*.

Ce qu'il reste à faire :

La numérisation des planches de "*Où est passé Charlie ?"*.

La saisie manuelle des coordonnées de Charlie sur chaque planche.

Le système de projection des planches.

La reprise du programme de test pour récupérer la position du regard en temps réel.

Un système de calibration semblable à celui du Screen Intersection Display (SID) est probablement à envisager.

La mise en oeuvre finale de "*Où est passé Charlie ?"* en rassemblant le système de projection, la récupération de la position du regard, la comparaison avec les coordonnées de Charlie sur la planche et le défilement des planches projetées.

Bibliographie

MATERIELS :

Site de Seeing Machines, fabriquant de faceLAB :
[1] http://www.seeingmachines.com

Site de Tobii Technology, qui propose plusieurs systèmes de suivi de regard :
[2] http://www.tobii.se/
Descriptif du Tobii 1750 Eye-tracker :
[3] http://www.tobii.se/downloads/Tobii_1750_PD_8.pdf

Produit par SR Research Ltd., Ontario, Canada, Eyelink est un système de suivi de regard trè performant, et très précis. Le site comprend la description globale, technique du matériel et des applications logicielles fournies.
[4] http://www.eyelinkinfo.com/

Site de Cambridge Research Systems (Tools for vision science). Propose une vue d'ensemble des technologies, matériels, et logiciels permettant le suivi de regard :
[5] http://www.crsltd.com
[6] http://www.crsltd.com/catalog/eye-trackers/
Documentation du MR-eyetracker :
[7] http://www.crsltd.com/support/documentation/mr-eyetracker/UG_v1.pdf
Le système Skalar :
[8] http://www.crsltd.com/catalog/skalar/index.html

Site de l'Applied Science Laboratories, Technology and Systems for Eye Tracking, qui propose plusieurs suiveurs de regard :
[9] http://www.a-s-l.com/

Un système Canon de suivi de regard performant pour les ophtalmologues.
Animation montrant le fonctionnement du système optique L'Auto eye-tracking.
[10] http://www.canon.com/technology/detail/product_tech/eye_tracking/#top

LOGICIELS :

Distributeur de GazeTracker (a tool for studying eye movement)
[11] www.eyeresponse.com

Descriptif du logiciel d'analyse de suivi de regard de l'entreprise Tobii : ClearView 2 .
[12] http://www.tobii.se/downloads/ClearView2_PD_9.pdf

ARTICLES / TECHNIQUES POUR LE SUIVI DE REGARD / SES UTILISATIONS :

Entre autres, description des techniques d'enregistrement du mouvement des yeux :
[13] http://www.unice.fr/LPEQ/doctorants/teresa/Mvts_des_yeux_et_web.pdf

Etude du suivi du regard (eye-tracking) et ergonomie des sites Web.
Introduction simple et claire du sujet avec historique rapide des différentes techniques. Suite orientée sur l'ergonomie.
[14] http://www.lergonome.com/pages/detail_articles.php?indice=13

Site du Electrical, Computer, and Systems Engineering Department du Rensselaer Polytechnic Institute - Troy, New York
sujet : Eye and gaze tracking for interactive graphic display
Résumé : This paper describes a computer vision system based on active IR illumination for real-time gaze tracking for interactive graphic display.
-> 10 pages sur le matériel, les techniques, algorithmes utilisés pour le suivi de regard.
[15] http://www.ecse.rpi.edu/~qji/Papers/gaze.pdf

Sur le site de l'evl (electronic visualization laboratory) **[16]**, on trouve une page titrée «Gaze Tracking Software for Children with Autism» **[17]**. Une page intéressante où on pourra même télécharger le code source du projet. Les nombreuses captures d'écran disponibles permettent d'avoir un aperçu des capacité du logiciel.
[16] evl (electronic visualization laboratory) : http://www.evl.uic.edu
[17] Gaze Tracking Software for Children with Autism : http://www.evl.uic.edu/dmitr/gazetracker/

Site d'une entreprise internationale allemande d'études marketing spécialisée en ergonomie du web, en copy tests et en évaluation de publicités et de linéaires. Leur outil principal est l'eye-tracking (pour les analyses d'ergonomie, etc...) A voir pour un aperçu des possibilités d'utilisation du suivi de regard (dans le domaine du marketing, en tous cas...)
[18] http://www.eye-square.com/francais/
Eye tracking and its application in usability and media research MICHAEL SCHIESSL, SABRINA DUDA, ANDREAS THÃ–LKE & RICO FISCHER
[19] http://www.eye-square.com/documents/EyeTracking-ResearchApplications.pdf

Utilisation du suivi de regard en ergonomie et design de sites web. Explication de ce qu'est le suivi de regard, comment ça marche (matériel, logiciel), enjeux économiques, analyse de sites webs avec exemples et résultats. (32 pages)
[20] http://banners.noticiasdot.com/termometro/boletines/docs/consultoras/nop/2004/nop_webdesigners.pdf

Papier de O K Oyekoya et F W M Stentiford, état de l'art assez complet, explication du fonctionnement, des différentes recherches actuelles sur le sujet, bonne bibliographie.

[21] http://www.ee.ucl.ac.uk/~ooyekoya/BTTJpaper.pdf

Article de la BBC : Replace your mouse with your eye (Monday, 8 July, 2002, 08:19 GMT 09:19 UK)
[22] http://news.bbc.co.uk/1/hi/sci/tech/2098030.stm

Utilisation du suivi de regard et de visage pour l'étude sur les émotions, la cognition :
[23] http://www.psychology.uwa.edu.au/home/research
[24] http://www.psy.uwa.edu.au/user/labs/cogemo/

Article de Andrew T. Duchowski, Department of Computer Science, Clemson University. Ce spécialiste y explique les utilisations du suivi de regard dans tous les domaines. (Bibliographie imposante)
[25] http://andrewd.ces.clemson.edu/research/vislab/docs/BET107cr.pdf

Théorie de l'analyse mouvement de l'oeil et du suivi de regard. Formules, algos, et tests par Andrew Duchowski, Eric Medlin, Nathan Cournia, and Hunter Murphy Department of Computer Science, Clemson University Anand Gramopadhye, Santosh Nair, Jeenal Vorah, and Brian Melloy Department of Industrial Engineering, Clemson University
[26] http://andrewd.ces.clemson.edu/research/vislab/docs/BET102cr.pdf

Les différents documents d'Andrew Duchowski et de son équipe d'où sont tirés les 2 précédents.
[27] http://andrewd.ces.clemson.edu/research/vislab/docs/

L'Eye Tracking Device (ETD) est utilisé par l'European Space Agency (ESA) pour mieux comprendre l'effet de l'apesanteur sur l'organisme, notamment sur l'équilibre, le mal de l'espace.
[28] http://www.spaceflight.esa.int/users/file.cfm?filename=miss-delta-exp-etd
[29] http://www.spaceflight.esa.int/delta/documents/factsheet-delta-hp-etd.pdf

Annexe I

Position des Pods infrarouges et de l'affiche du L3i située sur le mur derrière la machine faceLAB, en 001 :

IR Pod 1

position :

X(m) -0.215

Y(m) +0.325

Z(m) -0.090

IR Pod 2

position :

X(m) +0.215

Y(m) +0.325

Z(m) -0.090

Affiche L3i

position :

X(m) +0.260

Y(m) +1.103

Z(m) -1.060

size :

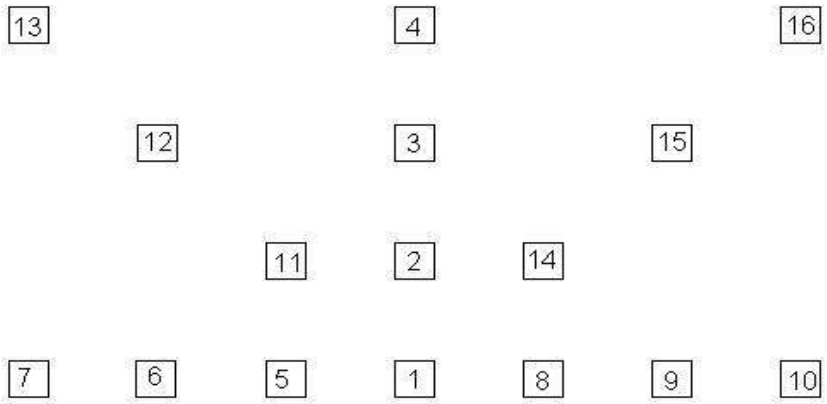
X(m) 1.320

Y(m) 0.915

Ces coordonnées sont exprimées en mètres, dans le repère du monde de faceLAB.

Annexe II

Coordonnées du centre des carrés de papier sur le mur de la salle 001 :



Coordonnées exprimées dans le repère du monde de faceLAB, en cm.

	X	Y	Z
Point 1	0	50	-106
Point 2	0	75	-106
Point 3	0	100	-106
Point 4	0	125	-106
Point 5	-25	50	-106
Point 6	-50	50	-106
Point 7	-75	50	-106
Point 8	25	50	-106
Point 9	50	50	-106
Point 10	75	50	-106
Point 11	-25	75	-106
Point 12	-50	100	-106
Point 13	-75	125	-106
Point 14	25	75	-106
Point 15	50	100	-106
Point 16	75	125	-106

Annexe III

Tableau des mesures effectuées de l'étude de la précision du suivi du regard avec le *SID :*

MDE : Mean Distance Error
MAE : Mean Angular Error
SD : Standard Deviation

			55-60 cm (70-75cm pour mode précision)				1 m (85 cm pour mode précision)				2 m (1m05 pour mode précison)			
			MDE (en mm)	MAE (en deg)	SD distance (en mm)	SD angle (en deg)	MDE (en mm)	MAE (en deg)	SD distance (en mm)	SD angle (en deg)	MDE (en mm)	MAE (en deg)	SD distance (en mm)	SD angle (en deg)
Front Only	Mode 1 (champ large)		Pas de SID				Pas de SID Perte du regard à partir de 60 cm				Pas de SID Regard perdu			
	Mode 2 (classique)		12,1	0,9	5,4	0,4	57,5 *	3 *	41,7 *	2,2 *	Perte du regard à environ 1m10			
	Mode 3 (précision)	Oeil Gauche	9,3	0,6	10	0,6	13,2	0.8	14,2	0.8	21,4	1	22,1	1
		Oeil Droit	9,7	0,6	9,8	0,6	15,4	0,9	16,7	1	21,9	1,1	25	1,2
Full Head	Mode 1 (champ large)		Pas de SID				Pas de SID Perte du regard à partir de 60 cm				Pas de SID Regard perdu			
	Mode 2 (classique)													
	Mode 3 (précision)	Oeil Gauche												
		Oeil Droit												

* : résultats instables (très variables d'une mesure à l'autre)

Annexe IV

Résultat des tests effectués sur le mur :

180 points de mesures (3 secondes)
Position de la tête : environ -0.01 0.28 0.9

-- Point 1 --	coord en x	coord en y	
Target Coordinates (m)	0.0	0.5	
Mean Deviation (mm)			53.3844
Standard Deviation (mm)			2.87167
Mean Deviation (mm)			56.8374
Standard Deviation (mm)			4.56135
Mean Deviation (mm)			51.6922
Standard Deviation (mm)			3.6359

-- Point 2 --			
Target Coordinates (m)	0.0	0.75	
Mean Deviation (mm)			90.4408
Standard Deviation (mm)			4.86085
Mean Deviation (mm)			89.2067
Standard Deviation (mm)			3.98022
Mean Deviation (mm)			79.9512
Standard Deviation (mm)			12.6459

-- Point 3 --			
Target Coordinates (m)	0.0	1.0	
Mean Deviation (mm)			88.7729
Standard Deviation (mm)			2.69231
Mean Deviation (mm)			89.4878
Standard Deviation (mm)			1.91694

Mean Deviation (mm)			90.0619
Standard Deviation (mm)			3.73341
-- Point 4 --			
Target Coordinates (m)	0.0	1.25	
Mean Deviation (mm)			148.062
Standard Deviation (mm)			3.34655
Mean Deviation (mm)			177.307
Standard Deviation (mm)			4.98392
Mean Deviation (mm)			158.6
Standard Deviation (mm)			8.16389
-- Point 5 --			
Target Coordinates (m)	-0.25	0.5	
Mean Deviation (mm)			55.9313
Standard Deviation (mm)			3.58481
Mean Deviation (mm)			58.7287
Standard Deviation (mm)			2.74886
Mean Deviation (mm)			60.5475
Standard Deviation (mm)			2.57972
-- Point 6 --			
Target Coordinates (m)	-0.5	0.5	
Mean Deviation (mm)			85.4667
Standard Deviation (mm)			3.56747
Mean Deviation (mm)			85.4256
Standard Deviation (mm)			6.17249
Mean Deviation (mm)			93.9115
Standard Deviation (mm)			4.88275
-- Point 7 --			
Target Coordinates (m)	-0.75	0.5	
Mean Deviation (mm)			182.633
Standard Deviation (mm)			11.312
Mean Deviation (mm)			171.516

Standard Deviation (mm)			18.5753
Mean Deviation (mm)			182.837
Standard Deviation (mm)			25.4056
-- Point 8 --			
Target Coordinates (m)	0.25	0.5	
Mean Deviation (mm)			42.06
Standard Deviation (mm)			3.04536
Mean Deviation (mm)			42.5864
Standard Deviation (mm)			1.93934
Mean Deviation (mm)			41.5919
Standard Deviation (mm)			1.74052
-- Point 9 --			
Target Coordinates (m)	0.5	0.5	
Mean Deviation (mm)			46.5682
Standard Deviation (mm)			5.31351
Mean Deviation (mm)			48.0703
Standard Deviation (mm)			3.74546
Mean Deviation (mm)			50.8041
Standard Deviation (mm)			9.36786
-- Point 10 --			
Target Coordinates (m)	0.75	0.5	
Mean Deviation (mm)			60.9791
Standard Deviation (mm)			4.34305
Mean Deviation (mm)			54.0521
Standard Deviation (mm)			10.506
Mean Deviation (mm)			58.4976
Standard Deviation (mm)			4.78603
-- Point 11 --			
Target Coordinates (m)	-0.25	0.75	
Mean Deviation (mm)			56.2575
Standard Deviation (mm)			2.18128

Mean Deviation (mm)			53.947
Standard Deviation (mm)			2.27195
Mean Deviation (mm)			55.9
Standard Deviation (mm)			1.93354
-- Point 12 --			
Target Coordinates (m)	-0.5	1.0	
Mean Deviation (mm)			94.8183
Standard Deviation (mm)			21.9525
Mean Deviation (mm)			116.086
Standard Deviation (mm)			7.82988
Mean Deviation (mm)			117.798
Standard Deviation (mm)			4.45631
-- Point 13 --			
Target Coordinates (m)	-0.75	1.25	
Mean Deviation (mm)			1457.74
Standard Deviation (mm)			0.00190735
Mean Deviation (mm)			1457.74
Standard Deviation (mm)			0.00190735
Mean Deviation (mm)			1457.74
Standard Deviation (mm)			0.00190735
-- Point 14 --			
Target Coordinates (m)	0.25	0.75	
Mean Deviation (mm)			45.3435
Standard Deviation (mm)			2.7666
Mean Deviation (mm)			36.6776
Standard Deviation (mm)			4.83587
Mean Deviation (mm)			36.1627
Standard Deviation (mm)			2.86133
-- Point 15 --			
Target Coordinates (m)	0.5	1.0	

Mean Deviation (mm)			74.262
Standard Deviation (mm)			5.6784
Mean Deviation (mm)			78.0998
Standard Deviation (mm)			7.79147
Mean Deviation (mm)			87.0126
Standard Deviation (mm)			3.66456

-- Point 16 --

Target Coordinates (m)	0.75	1.25	
Mean Deviation (mm)			1457.74
Standard Deviation (mm)			0.00190735
Mean Deviation (mm)			1457.74
Standard Deviation (mm)			0.00190735
Mean Deviation (mm)			1457.74
Standard Deviation (mm)			0.00190735

Annexe V

<u>Test v1</u> : Ce programme affiche en temps réel les coordonnées de l'intersection entre le regard et un objet du monde virtuel de faceLAB. Il affiche également le nom de l'objet en question.

Il affiche "nothing" pour le nom de l'objet si aucun objet du monde virtuel n'est regardé.

<u>Utilisation</u> :

./bin/test [numéro du port]

[numéro du port] par défaut le numéro du port est 2001

```
#include <csignal>
#include <string>
#include <vector>
// From SDK
#include <sdk.h>
#include "sdk_worldoutputdata.h"

using namespace std;
using namespace ClientToolsSDK;
using namespace PortNumberValidatorConsts;

DatagramSocket *_input_socket;

void handleEngineOutputData(EngineOutputData& output_data)
{
    // Recuperation des informations
        string  object_name  =  output_data.worldOutputData()->gazeIntersectionOutputData()-
>intersectionObjectName();
     vector<float> intersection = output_data.worldOutputData()->gazeIntersectionOutputData
()->gazeObjectIntersection();
    float x = intersection[0];
    float y = intersection[1];
    float z = intersection[2];

    // Formate et affiche les informations
    printf( "\r%22s -> x =%7.3f, y =%7.3f, z =%7.3f ", object_name.c_str(), x, y, z );

    // Synchronise le tampon utilisé par le flux
    std::cout << std::flush;
}

void handleDatagram(const DatagramSocket *input_socket)
{
    static std::vector<char> buffer;
    static InetAddress from;

    buffer.clear();
    input_socket->receiveDatagram(buffer,from);
    int pos = 0;

    // Extract all the objects from the buffer
```

```
    while (pos < (int)buffer.size())
    {
        Serializable* serializable = SerializableFactory::newObject(buffer, pos);
                        if((serializable   !=   NULL)   &&   (serializable->objectID()   ==
EngineOutputData::ENGINE_OUTPUT_DATA_ID))
            handleEngineOutputData(static_cast<EngineOutputData&>(*serializable));
        else
        {
            std::cout << std::endl << "Unrecognised packet received,  header id: "
            << buffer[0] << std::endl;
        }
        delete serializable;
    }
}

void ctrl_c(int)  {
    // Just in case we're interrupted mid line, go to the next line
    std::cout << std::endl;
    delete _input_socket;
    exit(0);
}

void checkArgs(int argc, char **argv, int &udp_port_number)
{
    if (argc > 1) {
        udp_port_number = atoi(argv[1]); // returns zero if conversion was unsuccessful
         if (udp_port_number < MINIMUM_PORT_NUMBER || udp_port_number > MAXIMUM_PORT_NUMBER)
{
            std::stringstream error_msg;
            error_msg << "UDP port numbers must be between " << MINIMUM_PORT_NUMBER;
            error_msg << " and " << MAXIMUM_PORT_NUMBER;
            throw std::runtime_error(error_msg.str());
        }
    }
}

int main(int argc, char **argv)
{
    int retval = 0;
    int udp_port_number = DEFAULT_PORT_NUMBER;
    try {
        checkArgs(argc, argv, udp_port_number);

        // Construct network object
        _input_socket = new DatagramSocket(udp_port_number);
           std::cout << "Listening on port " << udp_port_number << " (CTRL-C to exit)" <<
std::endl;
            std::cout << std::endl << "intersectionObjectName -> gazeObjectIntersection" <<
std::endl;
        signal(SIGINT, ctrl_c);
        while (true)
            handleDatagram(_input_socket);
    }
    catch (SocketException &e) {
        std::cout << std::endl << "Socket Error: " << e.what() << std::endl;
        retval = -1;
    }
    catch (std::exception &e) {
        std::cout << std::endl << "Exception: " << e.what() << std::endl;
        retval = -1;
    }
    return retval;
}
```

Annexe VI

<u>Test v2</u> : Ce programme récupère 180 mesures de position du regard sur un objet puis calcule la déviation moyenne et l'écart-type (Mean Deviation et Standard Deviation).

Il affiche "nothing" pour le nom de l'objet si aucun objet du monde virtuel n'est regardé.

<u>Utilisation</u> :

./bin/test [numéro du port] [nombre de mesures] [cible en x] [cible en y]

```
#include <csignal>
#include <string>
#include <vector>
#include <cmath>

// From SDK
#include <sdk.h>
#include "sdk_worldoutputdata.h"

// Nombre maximum de mesures pour le mode calcul statistique
#define MAX 1000

using namespace std;
using namespace ClientToolsSDK;
using namespace PortNumberValidatorConsts;

DatagramSocket *_input_socket;

// Compteur d'appels handleEngineOutputData
int ind = 0;
int limit = 0;
// Tableau de mesures
float deviation[3][MAX];
float target_x = 0;
float target_y = 0;

// Calcul de la moyenne
float arrayMean( float *array ) {
    float result = 0;
    int i;

    for( i = 0; i < limit; i++ ) {
        result += array[i];
    }

    return result / limit;
}

// Calcul de l'écart-type
float arrayStandard( float *array ) {
    float result = 0;
    float mean;
    int i;
```

```
    mean = arrayMean( array );

    for( i = 0; i < limit; i++ ) {
        result += pow( array[i] - mean, 2 );
    }

    return sqrt( result / limit );
}

// Fonction qui est appelé pour chaque frame
void handleEngineOutputData( EngineOutputData& output_data ) {
    // Récupération des informations :

    // 1- Récupération du numéro de la frame
    int frame_num = output_data.frameNum();

    // 2- Récupération du nom de l'objet sur lequel le regard est posé
        string  object_name  =  output_data.worldOutputData()->gazeIntersectionOutputData()-
>intersectionObjectName();

    // 3- Récupération des coordonnées du regard sur l'objet
     vector<float> intersection = output_data.worldOutputData()->gazeIntersectionOutputData
()->gazeObjectIntersection();

    // 4- Décomposition des coordonnées en x, y, z
    float x = intersection[0];
    float y = intersection[1];
    float z = intersection[2];

    // Formate et affiche les informations
      printf( "\r%20s -> %10d :   x =%7.3f, y =%7.3f, z =%7.3f ", object_name.c_str(),
frame_num, x, y, z );

    // Synchronise le tampon utilisé par le flux
    std::cout << std::flush;

    // Si on souhaite effectuer des mesures sur lesquelles
    // on effectuera des calculs statistiques par rapport
    // à une cible dont on a déterminé la position en x et y
    if( limit ) {
        // Pendant les mesures...
        if( ind < limit ) {
            // Calcul de la différence entre la position mesurée
            // et la position réelle de la cible en x et en y
            deviation[0][ind] = x - target_x;
            deviation[1][ind] = y - target_y;

            // Calcul de la distance entre la position mesurée
            // et la position réelle de la cible
             deviation[2][ind] = sqrt( pow( deviation[0][ind], 2 ) + pow( deviation[1][ind],
2 ) );

            // Incrémentation du compteur
            ind++;

        // Quand les mesures sont terminées...
        } else {
            // Calcul des moyennes
            float mean_x = arrayMean( deviation[0] );
            float mean_y = arrayMean( deviation[1] );
            float mean_d = arrayMean( deviation[2] );
            // Calcul des écart-types
            float strd_x = arrayStandard( deviation[0] );
            float strd_y = arrayStandard( deviation[1] );
            float strd_d = arrayStandard( deviation[2] );

            // Affichage des résultats
             printf( "\r%20s -> x =%7.1f, y =%7.1f -> %7.1f %-15s", "Mean Deviation", 1000 *
mean_x, 1000 * mean_y, 1000 * mean_d, "(mm)" );
            std::cout << std::endl;
              printf( "\r%20s -> x =%7.1f, y =%7.1f -> %7.1f %-15s", "Standard Deviation",
1000 * strd_x, 1000 * strd_y, 1000 * strd_d, "(mm)" );
```

```
            std::cout << std::endl;

            // Quitte le programme
            delete _input_socket;
            exit( 0 );
        }
    }
}

// Fonction de réception des informations sur chaque frame
void handleDatagram( const DatagramSocket *input_socket ) {
    static std::vector<char> buffer;
    static InetAddress from;

    buffer.clear();
    input_socket->receiveDatagram( buffer, from );
    int pos = 0;

    // Extract all the objects from the buffer
    while( pos < (int)buffer.size() ) {
        Serializable* serializable = SerializableFactory::newObject( buffer, pos );

                        if(   (serializable   !=   NULL)   &&   (serializable->objectID()   ==
EngineOutputData::ENGINE_OUTPUT_DATA_ID) ) {
                handleEngineOutputData( static_cast<EngineOutputData&>(*serializable) );
        } else {
            std::cout << std::endl << "Unrecognised packet received,  header id: "
            << buffer[0] << std::endl;
        }

        delete serializable;
    }
}

void ctrl_c( int )  {
    // Just in case we're interrupted mid line, go to the next line
    std::cout << std::endl;
    delete _input_socket;
    exit( 0 );
}

// Fonction de vérification des arguments
void checkArgs( int argc, char **argv, int &udp_port_number ) {
    if( argc > 1 ) {
        udp_port_number = atoi( argv[1] ); // returns zero if conversion was unsuccessful
        if( udp_port_number < MINIMUM_PORT_NUMBER || udp_port_number > MAXIMUM_PORT_NUMBER )
{
            std::stringstream error_msg;
            error_msg << "UDP port numbers must be between " << MINIMUM_PORT_NUMBER;
            error_msg << " and " << MAXIMUM_PORT_NUMBER;
            throw std::runtime_error( error_msg.str() );
        }
    }

    if( argc > 4 ) {
        if( (limit = atoi( argv[2] )) > MAX ) {
            limit = MAX;
        }
        target_x = atof( argv[3] );
        target_y = atof( argv[4] );
    } else {
        limit = 0;
    }

}

int main( int argc, char *argv[] ) {
    int retval = 0;
    int udp_port_number = DEFAULT_PORT_NUMBER;

    try {
        checkArgs( argc, argv, udp_port_number );
```

```
        // Construct network object
        _input_socket = new DatagramSocket( udp_port_number );
           std::cout << "Listening on port " << udp_port_number << " (CTRL-C to exit)" <<
std::endl;
        signal( SIGINT, ctrl_c );
        while( true )
            handleDatagram( _input_socket );
    }
    catch( SocketException &e ) {
        std::cout << std::endl << "Socket Error: " << e.what() << std::endl;
        retval = -1;
    }
    catch( std::exception &e ) {
        std::cout << std::endl << "Exception: " << e.what() << std::endl;
        retval = -1;
    }

    return retval;
}
```

Annexe VII

<u>Test v3</u> : Ce programme récupère des mesures de position du regard sur un objet puis calcule la déviation moyenne et l'écart-type (Mean Deviation et Standard Deviation). Il enregistre les résultats dans un fichier log si un nom de fichier est spécifié.

Il affiche "nothing" pour le nom de l'objet si aucun objet du monde virtuel n'est regardé.

<u>Utilisation</u> :

./bin/test [numéro du port] [nombre de mesures] [cible en x] [cible en y] [nom du fichier log]

```
#include <csignal>
#include <string>
#include <vector>
#include <cmath>
#include <iostream>
#include <fstream>

// From SDK
#include <sdk.h>
#include "sdk_worldoutputdata.h"

// Nombre maximum de mesures pour le mode calcul statistique
#define MAX 1000

using namespace std;
using namespace ClientToolsSDK;
using namespace PortNumberValidatorConsts;

DatagramSocket *_input_socket;

// Compteur d'appels handleEngineOutputData
int ind = 0;
int limit = 0;
// Tableau de mesures
float deviation[3][MAX];
float target_x = 0;
float target_y = 0;
// nom du fichier de log
string file_name = "";

// Calcul de la moyenne
float arrayMean( float *array ) {
    float result = 0;
    int i;

    for( i = 0; i < limit; i++ ) {
        result += array[i];
    }

    return result / limit;
}
```

```
// Calcul de l'écart-type
float arrayStandard( float *array ) {
    float result = 0;
    float mean;
    int i;

    mean = arrayMean( array );

    for( i = 0; i < limit; i++ ) {
        result += pow( array[i] - mean, 2 );
    }

    return sqrt( result / limit );
}

// Fonction qui est appelé pour chaque frame
void handleEngineOutputData( EngineOutputData& output_data ) {
    // Récupération des informations :

    // 1- Récupération du numéro de la frame
    int frame_num = output_data.frameNum();

    // 2- Récupération du nom de l'objet sur lequel le regard est posé
        string  object_name  =  output_data.worldOutputData()->gazeIntersectionOutputData()-
>intersectionObjectName();

    // 3- Récupération des coordonnées du regard sur l'objet
     vector<float> intersection = output_data.worldOutputData()->gazeIntersectionOutputData
()->gazeObjectIntersection();

    // 4- Décomposition des coordonnées en x, y, z
    float x = intersection[0];
    float y = intersection[1];
    float z = intersection[2];

    // Formate et affiche les informations
      printf( "\r%20s -> %10d :   x =%7.3f, y =%7.3f, z =%7.3f ", object_name.c_str(),
frame_num, x, y, z );

    // Synchronise le tampon utilisé par le flux
    cout << flush;

    // Si on souhaite effectuer des mesures sur lesquelles
    // on effectuera des calculs statistiques par rapport
    // à une cible dont on a déterminé la position en x et y
    if( limit ) {
        // Pendant les mesures...
        if( ind < limit ) {
            // Calcul de la différence entre la position mesurée
            // et la position réelle de la cible en x et en y
            deviation[0][ind] = x - target_x;
            deviation[1][ind] = y - target_y;

            // Calcul de la distance entre la position mesurée
            // et la position réelle de la cible
             deviation[2][ind] = sqrt( pow( deviation[0][ind], 2 ) + pow( deviation[1][ind],
2 ) );

            // Incrémentation du compteur
            ind++;

        // Quand les mesures sont terminées...
        } else {
            // Calcul des moyennes
            float mean_x = arrayMean( deviation[0] );
            float mean_y = arrayMean( deviation[1] );
            float mean_d = arrayMean( deviation[2] );
            // Calcul des écart-types
            float strd_x = arrayStandard( deviation[0] );
            float strd_y = arrayStandard( deviation[1] );
            float strd_d = arrayStandard( deviation[2] );
```

```
            // Affichage des résultats (en mm)
             printf( "\r%20s -> x =%7.1f, y =%7.1f -> %7.1f %-15s", "Mean Deviation", 1000 *
mean_x, 1000 * mean_y, 1000 * mean_d, "(mm)" );
            cout << endl;
              printf( "\r%20s -> x =%7.1f, y =%7.1f -> %7.1f %-15s", "Standard Deviation",
1000 * strd_x, 1000 * strd_y, 1000 * strd_d, "(mm)" );
            cout << endl;
            if( file_name != "" ) {
                ofstream file( file_name.c_str(), ios_base::app );
                if( file.is_open() ) {
                    file << "Target Coordinates (m)" << '\t' << target_x << '\t' << target_y
<< endl;
                    file << "Mean Deviation (mm)" << '\t' << 1000 * mean_x << '\t' << 1000 *
mean_y << '\t' << 1000 * mean_d << endl;
                      file << "Standard Deviation (mm)" << '\t' << 1000 * strd_x << '\t' <<
1000 * strd_y << '\t' << 1000 * strd_d << endl;
                    file << endl;
                    file.close();
                }
            }
            // Quitte le programme
            delete _input_socket;
            exit( 0 );
        }
    }
}

// Fonction de réception des informations sur chaque frame
void handleDatagram( const DatagramSocket *input_socket ) {
    static vector<char> buffer;
    static InetAddress from;

    buffer.clear();
    input_socket->receiveDatagram( buffer, from );
    int pos = 0;

    // Extract all the objects from the buffer
    while( pos < (int)buffer.size() ) {
        Serializable* serializable = SerializableFactory::newObject( buffer, pos );

                      if(   (serializable   !=   NULL)   &&   (serializable->objectID()   ==
EngineOutputData::ENGINE_OUTPUT_DATA_ID) ) {
                handleEngineOutputData( static_cast<EngineOutputData&>(*serializable) );
        } else {
            cout << endl << "Unrecognised packet received,  header id: "
            << buffer[0] << endl;
        }

        delete serializable;
    }
}

void ctrl_c( int )  {
    // Just in case we're interrupted mid line, go to the next line
    cout << endl;
    delete _input_socket;
    exit( 0 );
}

// Fonction de vérification des arguments
void checkArgs( int argc, char **argv, int &udp_port_number ) {
    if( argc > 1 ) {
        udp_port_number = atoi( argv[1] ); // returns zero if conversion was unsuccessful
        if( udp_port_number < MINIMUM_PORT_NUMBER || udp_port_number > MAXIMUM_PORT_NUMBER )
{
            stringstream error_msg;
            error_msg << "UDP port numbers must be between " << MINIMUM_PORT_NUMBER;
            error_msg << " and " << MAXIMUM_PORT_NUMBER;
            throw runtime_error( error_msg.str() );
        }
    }
```

```
    if( argc > 4 ) {
        if( (limit = atoi( argv[2] )) > MAX ) {
            limit = MAX;
        }
        target_x = atof( argv[3] );
        target_y = atof( argv[4] );
    } else {
        limit = 0;
    }

    if( argc > 5 ) {
        file_name = argv[5];
    }

}

int main( int argc, char **argv ) {
    int retval = 0;
    int udp_port_number = DEFAULT_PORT_NUMBER;

    try {
        checkArgs( argc, argv, udp_port_number );

        // Construct network object
        _input_socket = new DatagramSocket( udp_port_number );
        cout << "Listening on port " << udp_port_number << " (CTRL-C to exit)" << endl;
        signal( SIGINT, ctrl_c );
        while( true )
            handleDatagram( _input_socket );
    }
    catch( SocketException &e ) {
        cout << endl << "Socket Error: " << e.what() << endl;
        retval = -1;
    }
    catch( exception &e ) {
        cout << endl << "Exception: " << e.what() << endl;
        retval = -1;
    }

    return retval;
}
```