Travaux d'Études et de Recherche

Glove Control

Encadrement : Mr Philippe MARTIN

BIGER Kevin *MARGUET Quentin*

MULLER Thomas *VINATIER Frédéric*

Université de Nantes

Juin 2008

UNIVERSITÉ DE NANTES

Remerciements à :

Philippe Martin	Notre encadrant qui nous a fait confiance et qui a accepté de se lancer avec nous dans cette aventure
Gerson Sunyé	Pour nous avoir consacré du temps pour le problème des lecteurs / écrivains
Jean-Marie Normand Thomas Douillard Ricardo Soto Nicolas Berger	Étudiants thésards à la faculté de Nantes qui nous ont aidé à déboguer le programme
Marguet Steven	Enseignant chercheur qui nous a guidés pour la conception de l'algorithme génétique
ASCII	Pour nous avoir soutenu tout au long du projet
Bauny Matthieu	Un personnage unique

Table des matières

1. Problématique

Construire une nouvelle interface homme-machine, constituée d'un gant passif et d'un système d'acquisition vidéo multi-plateforme, permettant de prendre le contrôle de la souris et d'effectuer les fonctions principales implémentées par une souris.

2. Un court historique sur les IHM.

Il est fascinant de replonger dans l'évolution des IHM (interface homme-machine) au cours des cinquante dernières années. C'est de ce goût pour les innovations technologiques qu'est née notre envie de travailler sur les IHM dans le cadre de ce TER. Reprenons, un cours instant, les concepts technologiques mis en œuvre dans ce domaine pour bien comprendre quelles furent nos motivations.

Notre premier constat fut que les interfaces s'éloignent de plus en plus de l'implémentation des mécanismes contrôlés.
On peut dénombrer trois types de paradigmes d'interfaces applicables au domaine des IHM d'acquisition : [Al.Cooper'95]

- le paradigme **technologique** : l'interface reflète la manière dont le mécanisme contrôlé est construit. Cela conduit à des outils très puissants mais destinés à des spécialistes qui savent comment fonctionne la machine à piloter. Exemple : panneaux de contrôle, traitement par lots...

- le paradigme de la **métaphore** qui permet de mimer le comportement de l'interface sur celui d'un objet de la vie courante et donc déjà maîtrisé par l'utilisateur. Exemple : la notion de document.

- le paradigme **idiomatique** qui utilise des éléments d'interface au comportement stéréotypé, cohérent et donc simple à apprendre mais pas nécessairement calqué sur des objets de la vie réelle.

Les IHM utilisant le paradigme de la métaphore :

1957 : Le premier **photostyle** ou **crayon optique** (de l'anglais light pen) fut utilisé sur l'ordinateur Lincoln TX-0 du MIT Lincoln Laboratory.
Ce mécanisme se sert du temps de déplacement du canon à électrons, par rapport au signal de synchronisation, pour déterminer la position du stylet sur l'écran. Pour cette raison, le crayon optique a une assez bonne précision verticale, et médiocre en horizontal.
De plus, à l'époque où ce type d'outil était utilisé, les écrans étaient beaucoup plus bombés qu'aujourd'hui. Cela posait de gros problèmes de précision, et il n'était pas

rare qu'en cliquant au milieu de l'écran dans un logiciel de dessin, par exemple, un trait traverse l'écran en diagonale, du point sélectionné précédemment, jusqu'à un des bords de l'écran.
Cette IHM fut très populaire jusque dans les années 80 (commercialisée avec les Thomson TO9, TO7,TO7/70, MO5). Mais ses problèmes d'ergonomie (garder son bras levé pour contrôler les objets en les pointant explicitement apparut pénible) eurent raison de ce procédé et l'idée de contrôler un curseur de cette façon fut progressivement dépréciée.

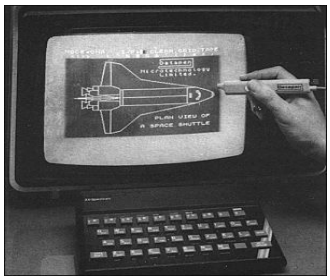


Illustration 1: Le photostyle

1964 : La première **tablette graphique** pouvant s'apparenter à celles actuelles fut la Rand Tablet, connue aussi sous le nom de la Grafacon (venant du mot anglais graphic converter). La surface plate de la tablette était constituée d'une grille qui offrait un quadrillage parfait avec un repérage des coordonnées horizontales et verticales. Chaque coordonnée produisait un signal magnétique que le stylet repérait en tant que récepteur et renvoyait pour signaler sa position.
Dans sa forme actuelle, elle se compose d'une surface plane active, la tablette, et d'un stylet apparenté à un crayon traditionnel, permettant d'interagir avec l'ordinateur. Cette interface est une implémentation directe du paradigme métaphore puisqu'il met en œuvre le type de comportement qu'un utilisateur peut avoir face à une feuille de papier.
Cette IHM fut ensuite améliorée et elle constitue toujours un élément majeur dans l'interaction homme-machine lors d'utilisations telles que la CAO, le design, le graphisme, etc...

Illustration 2: La première tablette Grafacon

1980-1990 : Les **gants à capteurs** (data-gloves, power gloves ou encore cyber-gloves) furent introduits.
Ils ont l'énorme avantage d'être très intuitifs d'utilisation : il suffit de bouger sa main pour les faire fonctionner.

Les mouvements, souvent simples comme ouvrir - fermer la main, déclenchent des capteurs qui envoient les instructions au système.
Nintendo a tenté de distribuer largement des gants de ce type (les Power Gloves), mais leur prix prohibitif pour l'époque (500 Frcs) les a empêchés de connaître un véritable succès. Ils n'ont à l'époque été construits que pour manipuler des objets d'un contexte prédéfini, et souvent de grosse taille, n'ont pas été d'une ergonomie adéquate.


Illustration 3: Les datagloves

Les IHM utilisant le paradigme idiomatique :

1952 : La première **trackball («** boule de commande ») fut inventée par Tom Cranston et Fred Longstaff comme partie du Royal Canadian Navy's DATAR (radar/système militaire canadien [onze ans avant la première souris]). Elle était constituée d'une boule de bowling de 5 pouces.
Ce concept fut ensuite généralisé et miniaturisé. Ainsi une trackball n'est rien d'autre qu'une bille fixée à un socle, qui détecte la rotation de celle-ci suivant l'axe horizontal et l'axe vertical. Elles furent installées sur les ordinateurs portables jusqu'à l'avènement des pavés tactiles.

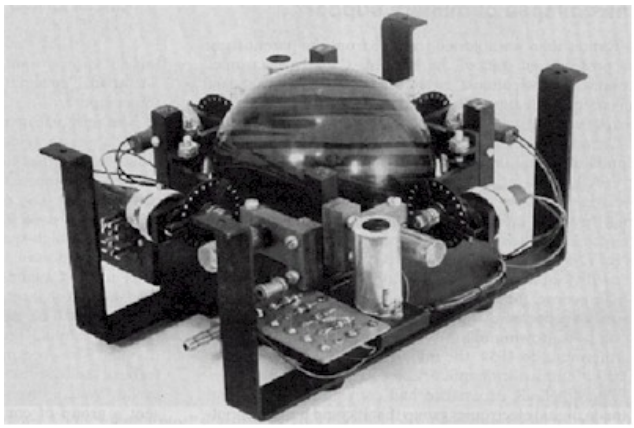


Illustration 4: La première trackball

Les trackballs à bille de grosse taille sont encore utilisées à l'heure actuelle dans le domaine de la CAO car reconnues pour leur grande précision.

1963 : **La première souris** a été inventée et mise au point par Douglas Carle Engelbart du Stanford Research Institute (SRI) : il s'agissait d'une souris en bois contenant deux disques perpendiculaires et reliée à l'ordinateur par une paire de fils torsadés.
Bien que le concept de la souris ait été présenté en 1968, il ne s’est pas imposé immédiatement. Jusqu’au début des années 80, les techniciens ont dédaigné la souris, lui préférant le photostyle (auparavant présenté).
Tout le monde connaît ensuite les évolutions de cet interface tant elle s'est répandue et avérée utile aux côtés de nos machines.

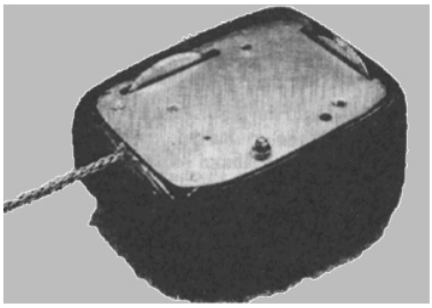


Illustration 5: La première souris

1994 : le **pavé tactile** (en anglais trackpad) fut commercialisé sur les PowerBook 500 d'Apple. Il est resté principalement utilisé dans le domaine des ordinateurs portables. Ils utilisent en majorité le repérage par changement de capacité entre un maillage d'électrodes conductives et la main de l'utilisateur étant elle aussi conductive, mais parfois également le repérage par zone de pression.

Illustration 6: Le PowerBook 500 et son pavé tactile

Un mot sur les IHM atypiques :
On a vu également naître au cours des années, des IHM dites atypiques. Elles ont en fait été construites dans des buts précis. Souvent basées sur le paradigme de métaphore, elles servent à prolonger les mécanismes de contrôle dans des contextes de simulation, très souvent dérivant du but premier du programme pour lesquels elles opèrent.

On peut par exemple, penser au joystick (ou manche à balai) rappelant le manche de contrôle d'un avion, ou encore au couple pédales+volant permettant de contrôler la direction dans une simulation de conduite.
Il existe de très nombreuses IHM atypiques, la plupart construites uniquement dans un contexte de simulation donc très ciblé et inadapté au contrôle des applications dans un sens générique.

Les IHM idiomatiques et liaison avec **GloveControl**:

Selon Alan Cooper [Al.Cooper'95], l'avenir réside dans les IHM idiomatiques. Il est clair qu'en observant le comportement des utilisateurs, on ne peut qu'adhérer à cette idée. Prenons l'exemple de la souris. Cette interface est appréhendée idiomatiquement. Il n'y a rien dans une souris qui permette en l'observant, de dire *a priori* quel est son but ni comment l'utiliser, ni quelque objet de la vie courante qui puisse, par appel à notre expérience, permettre de savoir comment s'en servir. Néanmoins, apprendre à pointer des objets, grâce à une souris, est d'une facilité déconcertante. Au départ, il a dû suffire d'une démonstration de quelques secondes pour qu'un novice puisse maîtriser cette IHM. Nous ne nous soucions pas plus du fonctionnement ou de l'implémentation d'une souris dans le système, mais l'utilisons tout simplement. Et c'est là, la définition même de l'apprentissage idiomatique.

Notre IHM, que nous avons baptisée « GloveControl » fut pensée en ce sens. Elle n'est pas d'un apprentissage inné bien qu'elle soit basée sur la métaphore de la main puisqu'elle le couple au paradigme idiomatique introduit dans une IHM souris.

Son utilisation réutilise donc quelques aspects positifs de la métaphore de la main, ne nécessitant aucun apprentissage, mais également un héritage des fonctions basiques d'une souris.
On est donc en présence d'un « exo-organe», car c'est simplement un prolongement virtuel des capacités d'une main humaine, projetées dans le contexte du contrôle d'un curseur, normalisé en terme d'interaction par la plupart des systèmes d'exploitation courants.

3. L'évolution conceptuelle de Glove Control

Voyons ensemble l'évolution du système telle que nous l'avons pensée.

Nous avons, au début, calqué notre IHM sur le paradigme bureau.

Notre première idée aurait été donc de concevoir une sorte de plan de travail translucide, sur lequel on puisse poser les mains, les points lumineux étant capturés par une webcam placée en dessous.

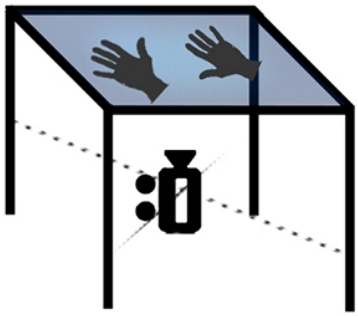


Illustration 7: Notre première idée d'IHM

Cette IHM avait l'avantage de pouvoir reposer ses mains comme sur un bureau , mais la mise en place d'une telle IHM aurait été délicate, et difficile à imaginer lors du passage à l'échelle, puisqu'elle imposait de réserver une place spécifique dans l'espace de travail.
Nous n'avons donc pour ces raisons pas envisagé de la développer.

Notre deuxième idée, basée elle aussi sur le principe du paradigme bureau, aurait consisté à réutiliser l'espace de travail traditionnel, le bureau, sur lequel aurait été posée une webcam visant par le dessus. Les mouvements auraient donc été traduits par des allumages de diodes suivant la pression des doigts sur la surface de travail.

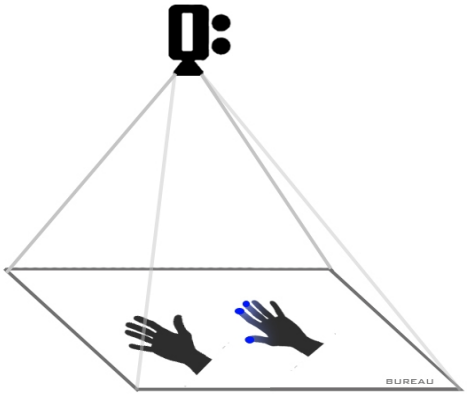


Illustration 8: Notre deuxième idée d'IHM

C'est à cette étape de développement, et sur cette idée d'IHM, que nous avons mené nos premiers tests de faisabilité, permettant d'assurer le fait que nous puissions nous engager sans risque majeur dans un projet tel que celui-ci. Nous avons alors réalisé un programme capturant la lumière d'un gant construit pour ces tests.

3.1. Premier prototype : Glove1

GLOVE1, la première étape de notre recherche :

Le premier gant était constitué d'interrupteurs permettant d'allumer ou non les diodes placées sur le dessus, en fonction des pressions sur le bureau. Nous avons développé en parallèle un programme permettant d'acquérir les images, et de les analyser avec des filtres de bases.
Ce programme utilisait déjà un outil majeur que nous avons par la suite conservé : la librairie OpenCV (INTEL). Cette librairie conçue par INTEL est une librairie de traitement d'image avancée et libre (open-source BSD). Cette librairie (Framework) propose des fonctionnalités très puissantes et relativement optimisées pour l'utilisation que nous en faisons. Cela nous a donc évité de réimplémenter la plupart de celles-ci ou de perdre du temps en redescendant à un niveau d'abstraction plus en dessous. Cela nous a, de plus, appris de nombreuses choses, concernant l'utilisation de frameworks dans le développement logiciel; chose que nous avions jusqu'alors à peine abordée dans notre formation ou notre expérience personnelle.

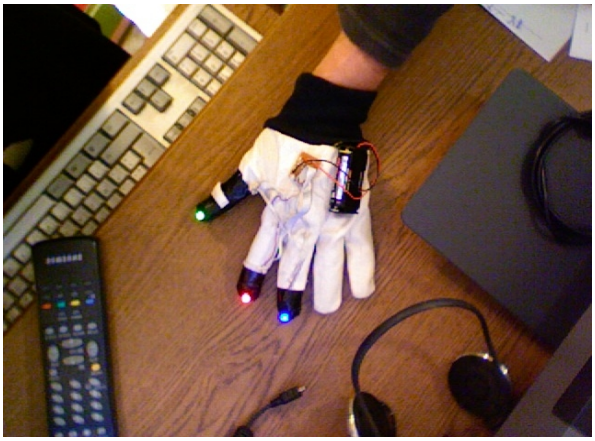


Illustration 9: Le premier prototype de gant

Nous avons ainsi capturé un certain nombre de photos de test (plus de 50). Celles ci ont mis en évidence certaines propriétés que nous avons dû accepter , et qui n'étaient pas forcément évidentes.

La première constatation concernait les couleurs des diodes.
Nous pensions jusqu'alors que les diodes émettaient des signaux de couleur unie. Or , d'après nos tests, nous avons directement pu conclure que ce n'était pas le cas.
La couleur est en réalité présente sur le contour (plastique) de la diode, mais en son centre, la lumière émise est si forte, ou sature à tel point que ce centre paraît blanc, ou très proche du blanc.

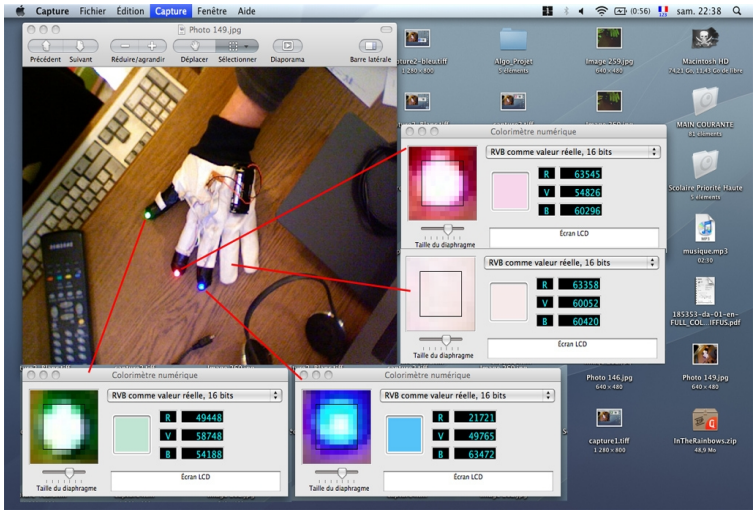


Illustration 10: Le centre des diodes est blanc

Le deuxième constat que nous avons pu faire, concerne la couleur même des diodes. En effet, il fut impressionnant de constater par séparation des canaux d'une image, que les couleurs ambiantes ne sont pas uniformément représentées. Cela paraît normal du fait de la longueur d'onde de celle-ci et donc de leur capacité à rayonner, mais nous ne l'imaginions pas dans ces proportions.

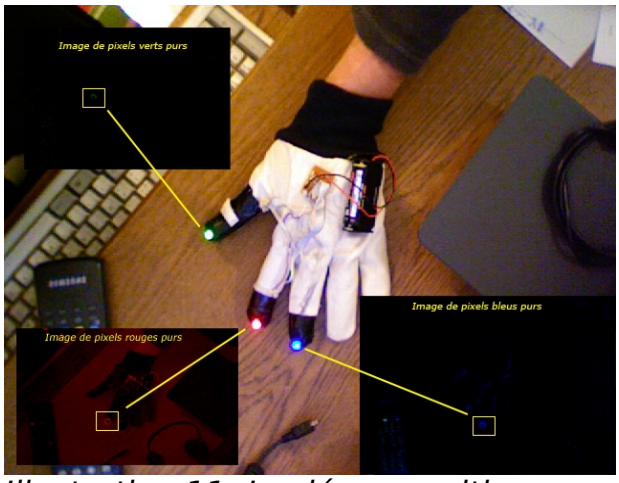


Illustration 11: La décomposition en canaux RVB

Sur l'image ci-dessus, nous avons regroupé les 3 composantes RVB (rouge, vert, bleu) qui composent notre image source. On y voit très distinctement qu'une très grande quantité de rouge y est présente, dans le décor , en dehors du fait que notre diode rouge soit allumée.
Nous avons donc conclu qu'il était impossible de se baser sur des diodes rouges pour représenter des actions utilisateur tant cette lumière peut se retrouver de façon diffuse dans une capture.
A l'inverse, les couleurs vertes et bleues sont agréablement rarement présentes dans une image capturée. On ne retrouve qu'un nombre très limité de parasites dans ces

couleurs. (Cela n'est pas sans rappeler les fonds verts ou bleus du cinéma).
Nous avons donc décidé de ne rester que sur le bleu dans la suite de notre travail.

Enfin, la dernière constatation que nous ayons faite, a remis en cause toute l'idée de l'IHM. En effet, après capture de vidéo de test et traitement par un filtre primaire sur les couleurs, nous avons constaté que le fait d'allumer ou d'éteindre une diode, de façon plus ou moins rapide, ne peut être correctement capturé par le matériel, du fait sans doute de la trop faible fréquence d'échantillonnage proposée par une webcam (en général inférieure ou égale à 30fps). Ainsi , lors de clics, ou double clics, nous ne voyons pas du tout la diode s'allumer , ou inversement, s'éteindre.

Il a fallu alors remettre en cause complètement l'agencement de notre IHM. Et c'est après concertation et longues discussions entre nous que nous avons pensé à l'idée actuelle (celle que nous avons implémentée) que nous allons maintenant exposer.

GloveControl : le prototype du gant, et l'application de contrôle

Après considération de tous ces éléments nouveaux, nous avons repensé entièrement notre IHM, dans un but essentiel : une simplicité plus grande du matériel mis en œuvre pour plus de facilité lors du déploiement de la technologie, et moins de défauts dûs à l'aspect « prototype » du système.

Le système est donc composé d'un gant et d'une application comme auparavant mais son fonctionnement est différent.
Dans GloveControl, l'utilisateur est devant sa caméra, et la caméra est (comme à l'habitude) posée sur l'écran , ou même incorporée dans l'écran.

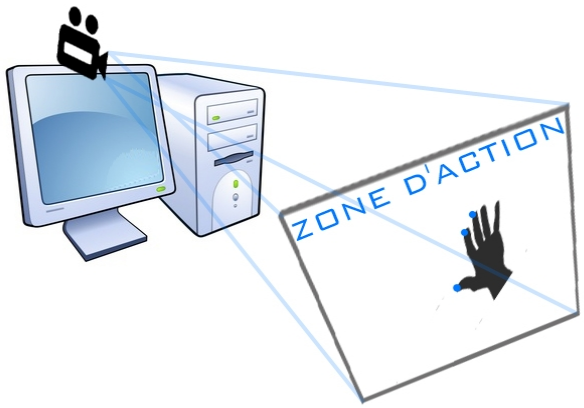


Illustration 12: L'idée finale

On reprend ici le principe du photostyle certes, mais nous prévoyons une utilisation en mode « souris », c'est à dire dans un contexte absolu et non relatif comme celui d'un photostyle.

Ainsi, les mouvements sont simples et plus confortables pour l'utilisateur car ils ne l'obligent pas à pointer spécifiquement un objet de façon aussi précise qu'avec un dispositif de pointage tel qu'un photostyle.

Cela présente l'avantage de ne pas avoir besoin de modifier la place de la webcam sur le bureau, ni de rajouter du matériel pour fixer un dispositif comme celui de notre idée précédente. On y gagne en espace puisque l'espace est pris sur l'espace d'évolution de l'utilisateur et non sur la surface du bureau qui reste libre. Et enfin, elle permet de réduire le coût du système, puisqu'il en est ramené au simple coût du gant, qui du fait de sa très grande simplicité peut être fabriqué par tout un chacun pour un prix en composants d'environ 10€. (le prix du gant varie suivant le type de gant et son tissu)

3.2. Second prototype : le Glove2

Le gant utilisé a donc sensiblement changé depuis le début de notre projet, puisque d'une part , seules des diodes bleues y sont incorporées, et qu'elles sont placées à l'extrémité des doigts (le pointage se faisant face à la caméra).

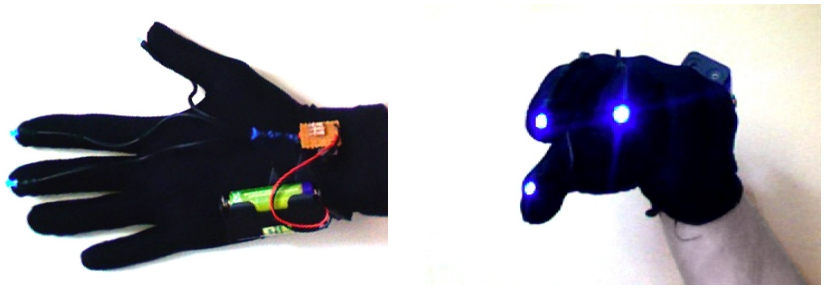


Illustration 13: Vue du dessus, et de face du Glove2

L'idée de trois diodes fut simplement l'idée de pouvoir coder 3^2 événements (une diode étant en fait un bit -> on peut l'éteindre en la cachant ou non).

Sa conception est d'une simplicité déconcertante. Il suffit de trois diodes de 3 millimètres, pour leur petite taille, bleues ou vertes, et d'un système d'alimentation (ici 2 piles AA délivrant 3V et 3 résistances de protection).
Voici donc pour la partie passive permettant d'effectuer des gestes qui seront interprétés par l'application. (Voir le Glove3, le prototype miniaturisé).
Passons maintenant à l'application GloveControl (qu'on appelle parfois GC).

3.3. L'application GloveControl

Notre application réalise donc l'interprétation des mouvements réalisés à la main en face d'une caméra.

GloveControl est une application multi-thread et multi-plateforme. Elle a été développée au départ pour fonctionner sur plateforme mac et pc, mais pourrait également être portée sous plateforme Unix à ceci près que la première considération, que nous avons citée plus haut (la webcam est configurée pour fonctionner sur le système) soit assurée.

Cependant, l'application n'a pas été conçue directement sous sa forme actuelle, et elle a sans aucun doute fait appel à une réflexion poussée sur la conception logicielle.

Voici un schéma temporel représentant l'évolution du programme.

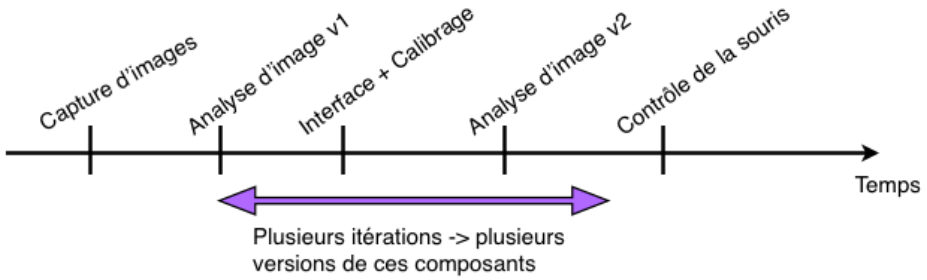


Illustration 14: Evolution du développement

Dans un premier temps , nous avons cherché à développer la capture des images de la webcam et l'analyse de celles-ci. Puis compte tenu des problèmes liés à cette analyse (incapacité de trouver un réglage fixe permettant de traiter les images dans des contextes d'éclairage différents), nous avons développé un mécanisme de calibrage des couleurs à échantillonner. Et en parallèle , nous avons adapté l'analyseur capable de retrouver l'information dans l'image capturée à partir du seuil calculé. Cette itération a été réalisée plusieurs fois dans le cycle de vie de GloveControl.

Nous avons donc essayé de décomposer le programme en une sorte d'assemblage de composants spécifiques représentés sur la figure ci-dessous. On peut noter que cette méthode de développement fut nouvelle pour nous, dans le sens où les versions se sont croisées, chacun implémentant des évolutions de classes différentes. Nous avons eu un certain nombre de problèmes dûs uniquement à ce mode de développement réparti.

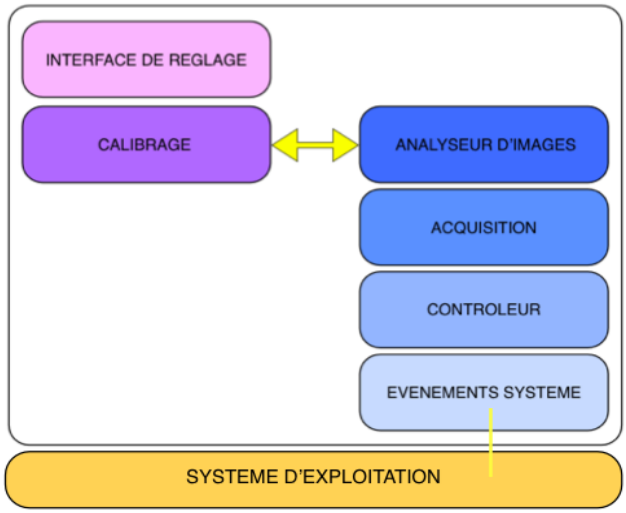


Illustration 15: Les couches de l'application

Le programme GloveControl se décompose donc de la façon suivante.

Une partie conçue pour calibrer le système avant l'utilisation

• une interface graphique : cette interface est réalisée grâce à la librairie Qt de la société Trolltech. Qt est un framework dit cross-plateforme (portable) conçu pour le développement d'interfaces graphiques en C++ composé d'un grand nombre de classes C++ réutilisables et regroupées sous la forme d'une riche bibliothèque.
Cette interface sert à calibrer le système avant de pouvoir s'en servir.

• un module de calibrage, qui réalise les tests pour trouver le meilleur seuillage possible. Il implémente l'algorithme génétique et calcule le seuillage (filtrage) à utiliser dans la situation donnée et la taille d'écartement des diodes que l'utilisateur veut employer lors de l'utilisation,Nous y reviendrons en détail dans un chapitre ultérieur.

Une partie conçue pour utiliser le système

• un thread d'acquisition d'image, il récupère les mouvements et donc les informations lumineuses générées par l'utilisation du gant,

• un thread d'analyse d'image qui se base sur le seuil calculé dans la phase de calibrage et permettant d'extraire les informations utiles des images capturées,

• un thread de contrôle, qui provoque des événements système suivant le type d'événement détecté par l'analyseur d'image. Il utilise la couche événement qui est la seule partie de notre projet qui soit dépendante de la plateforme utilisée.

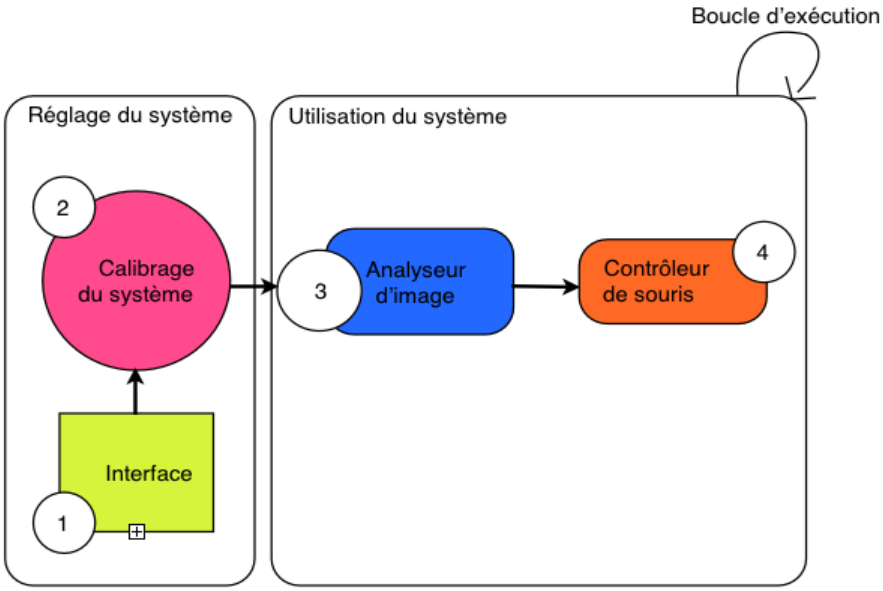


Illustration 16: L'ordre d'exécution du programme

Le schéma ci-dessus résume la chronologie d'une exécution de GloveControl. Dans un premier temps, à l'aide de l'interface, l'utilisateur calibre le système en faisant appel au module de calibrage. Une fois cette étape de réglage effectuée, l'utilisateur peut utiliser le système. Dans une boucle continue : l'analyseur utilise les images capturées en leur appliquant un filtre pour extraire les informations et les renvoie au thread contrôleur qui provoque les événements souris.

4. Interface graphique

4.1. Introduction

L'interface graphique occupe une place importante dans la conception d'un programme. Elle permet de faire le lien entre l'utilisateur et les fonctionnalités de l'application. Cependant, elle est trop souvent délaissée par les programmeurs. En conséquence, nombre d'applications de qualité demeurent inconnues ou inutilisables. Nous avons dès le départ souhaité faire une application utilisable par le plus grand nombre. La conception d'une interface graphique un minimum intuitive était donc nécessaire à notre projet. Cette partie ne va pas rentrer dans le détail du code. C'est plus un aperçu de notre interface graphique. Nous allons ensuite mettre en évidence les difficultés que nous avons rencontrées.

4.2. Les différents modules

Plusieurs modules composent l'interface graphique. Le premier module englobe tous les autres. C'est le menu qui permet d'accéder aux différentes fonctionnalités de l'application. Ce menu est obtenu en cliquant sur l'icône de GloveControl dans la zone de notification.
Il est alors possible de :

- quitter l'application ;
- accéder à l'aide ;
- procéder au calibrage,
- lancer le traitement.

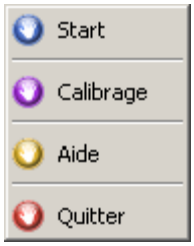
4.2.1. Les icônes

Voici les différentes icônes utilisées dans notre application.

Elles ont été faites par le talentueux designer de notre équipe.

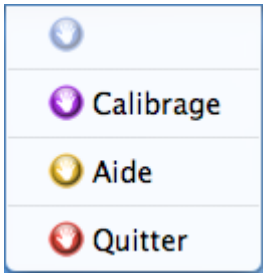
4.2.2. Le menu

Voici des screenshots du menu de GloveControl.



Le menu (sous Windows XP)

Dans cette image, toutes les icônes sont visibles, l'utilisateur peut donc utiliser le programme.



Le menu (sous Mac OS X Leopard)

Sur cette seconde image, l'icône en haut est désactivée et le texte n'apparaît pas. L'utilisateur ne peut pas encore utiliser le logiciel. Il doit préalablement procéder au calibrage de l'application.

4.2.3. L'aide

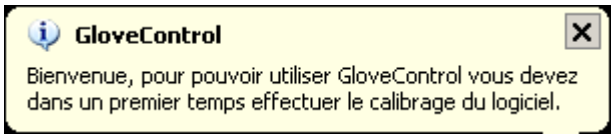
L'aide n'est pas encore implémentée mais à l'avenir, cela ouvrira une page web ou un document ".pdf".

4.2.4. Le traitement

Le lancement du traitement ne fait rien de visible, il permet d'utiliser le gant comme une souris. Cette opération n'est possible qu'une fois le calibrage effectué entièrement. Le programme vérifie que les fichiers de calibrage sont bien présents avant de permettre l'utilisation de cette fonctionnalité. L'utilisateur peut arrêter le traitement en recliquant sur le bouton.

4.2.5. ***Les info-bulles***

Des messages (bulles d'informations) permettent d'informer l'utilisateur sur d'éventuels problèmes.



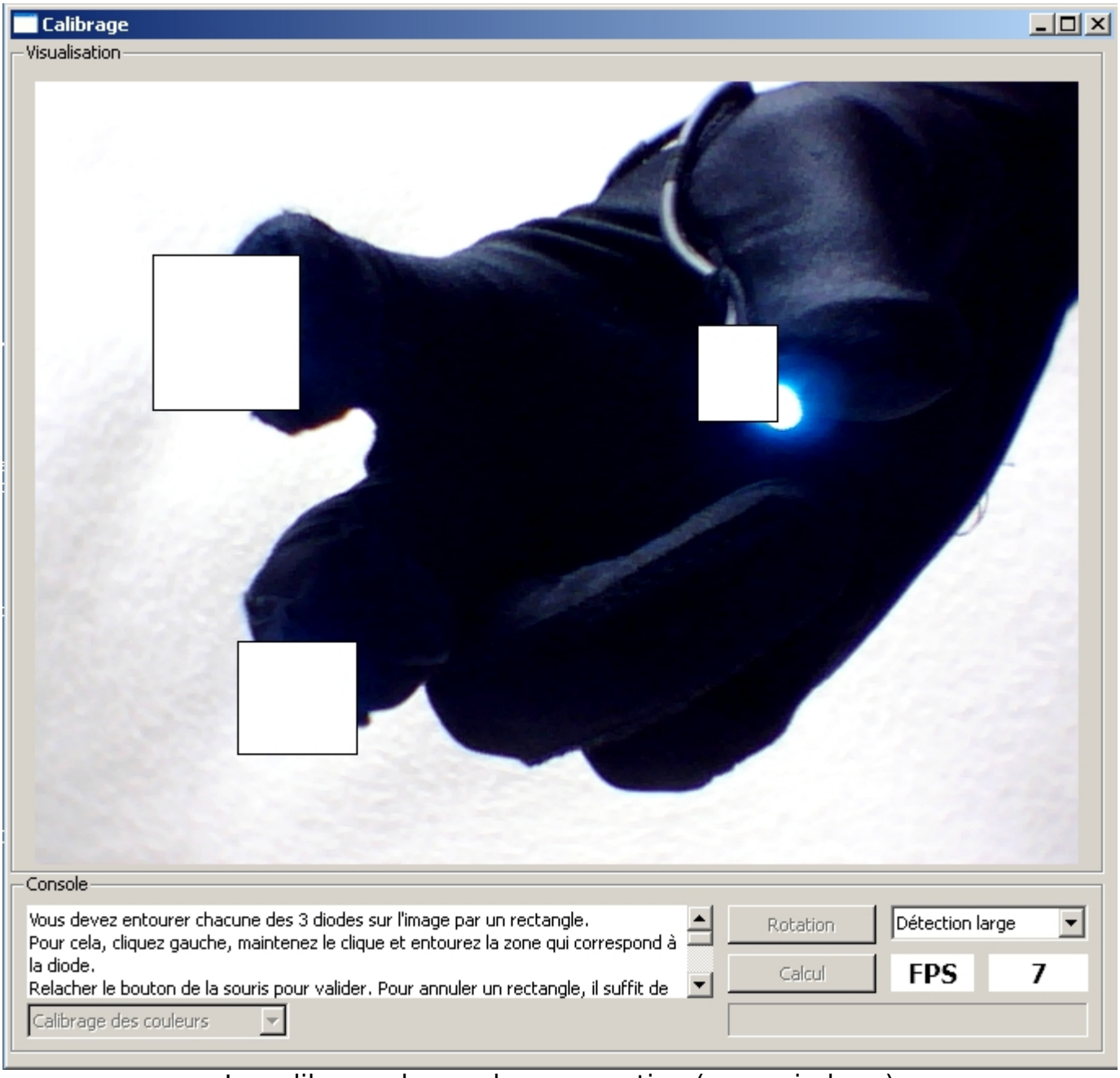
Une info-bulle (sous Windows XP)

Une info-bulle (sous Mac OS X Leopard)

Ces deux info-bulles invitent l'utilisateur à effectuer le calibrage.

4.2.6. ***Le calibrage***

C'est la partie la plus complexe de l'interface graphique. L'idée est de faire un système étape par étape pour l'utilisateur en procédant à une restriction sur ses choix d'actions. Des messages l'informent de ce qu'il doit faire. Il a juste à suivre les instructions.



Le calibrage des couleurs en action (sous windows)

4.3. Difficultés rencontrées

L'interface graphique peut paraître comme la partie facile et agréable du projet. Cependant, sa conception est extrêmement complexe. Il y a énormément de parallélisme et en conséquence beaucoup de zones à protéger par mutex. Par exemple, lors de l'appui sur le bouton qui permet de stopper la webcam, il faut modifier une variable et cette variable doit-être protégée. L'affichage des images de la webcam est dans un thread séparé. Ce sont des exemples parmi tant d'autres.

Il faut bien avoir présent à l'esprit que l'ensemble doit être cohérent.
Quand un utilisateur passe une étape, nous devons tenir compte de chaque bouton en présence :

- devons-nous activer/désactiver ces boutons ?
- un bouton doit-il être relié à un nouveau signal/slot ?
- Le texte sur les boutons doit-il être modifié ?

Les signaux et les slots sont une propriété propre à Qt. L'appui sur un bouton provoque un signal. Quand l'application récupère ce signal (slot) une opération est effectuée (appel d'une méthode). Les signaux et slots peuvent être utilisé pour effectuer nombres d'opérations.

L'affichage de l'image de la webcam dans l'interface graphique n'est pas non plus une opération des plus faciles. Les images de le webcam sont obtenues en utilisant OpenCV, hors l'interface utilise Qt et ne peut pas afficher les images fournies par OpenCV. Il faut donc préalablement convertir cette image dans le format supporté par Qt avant affichage. Le problème est que cette conversion coûte du temps de calcul et que l'affichage d'image sous Qt coûte du temps de calcul. En conséquence, le rafraichissement n'est pas des plus rapides.
Dans les versions futures, l'affichage se fera en utilisant une fenêtre OpenGL (QGLWidget). Ainsi, c'est la carte graphique qui s'occupera du rafraichissement, nous gagnerons donc en rapidité. La conversion des images d'un format à l'autre peut également être encore améliorée en copiant des zones mémoires.

4.4. Conclusion

Comme nous avons pu le voir la conception d'une interface graphique n'est pas simple. Cependant, les résultats sans être exceptionnels sont plus que corrects. Il est à noter en plus que l'interface est plutôt stable sous Mac OS X Leopard et totalement stable sous environnement Windows XP (de très nombreux tests nous permettent de le dire).
Sans être spécialement attractif, l'interface du fait de son aspect unidirectionnel demeure intuitive, ergonomique et donc agréable à utiliser.

5. Calibrage

5.1. Introduction

La récupération de la position des diodes sur une image n'est pas triviale. Il faut nécessairement tenir compte de différents paramètres :

- rendu de la webcam :
  - saturation, luminosité, balance des couleurs, contraste... ;
  - stabilité des couleurs d'une image à l'autre ;
  - résolution (largeur * hauteur) ;
  - nombre d'images par seconde,
- couleurs des diodes ;
- couleurs de l'environnement,
- distance entre les diodes et la webcam.

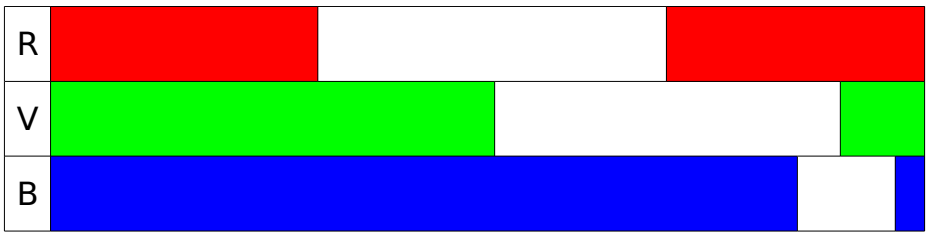
Ces différentes contraintes permettent de jauger de la viabilité des méthodes que nous avons pensé à utiliser. Une méthode qui est capable de tenir compte de toutes ces contraintes sera théoriquement très performante. Après la récupération de la position des diodes, il nous faut également récupérer des informations afin de pouvoir traiter les événements souris. La position des diodes quand l'utilisateur souhaite déplacer la souris est une des informations que nous devons obtenir avant que l'utilisateur puisse utiliser GloveControl. Pour commencer, nous allons d'abord nous intéresser à la détection de la position des diodes dans l'image. Avec ce que nous avons défini comme paramètres, il est dorénavant possible de répertorier différentes méthodes de localisation.

5.2. Méthodes pour récupérer la position des diodes

5.2.1. Principe de base

Le rendu de la webcam dépend entièrement du matériel utilisé ce qui nous oblige à faire un programme adaptable. Il existe différents algorithmes pour localiser des formes, des couleurs sur une image mais ils reposent en général sur une base commune.

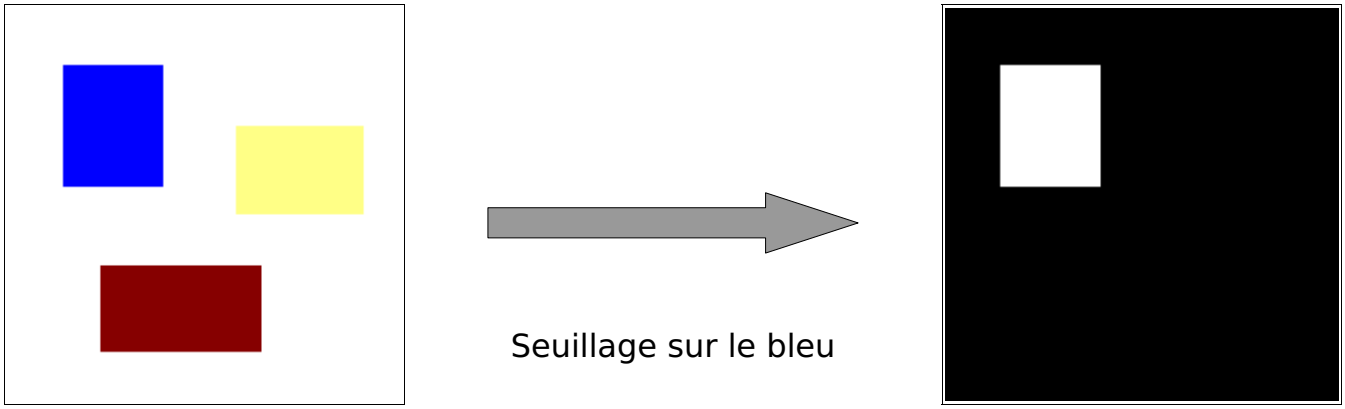
Il y a d'abord application d'un filtre (seuil) sur l'image, ensuite un autre filtre (passe-haut) est appliqué. Le résultat est une image en noir et blanc.
Le seuil permet de filtrer certaines plages de couleurs. Dans notre cas, nous devons filtrer les couleurs dîtes "bleues".



En blanc, ce sont les valeurs autorisées par le seuil. Chaque composante a une borne inférieure et supérieure.

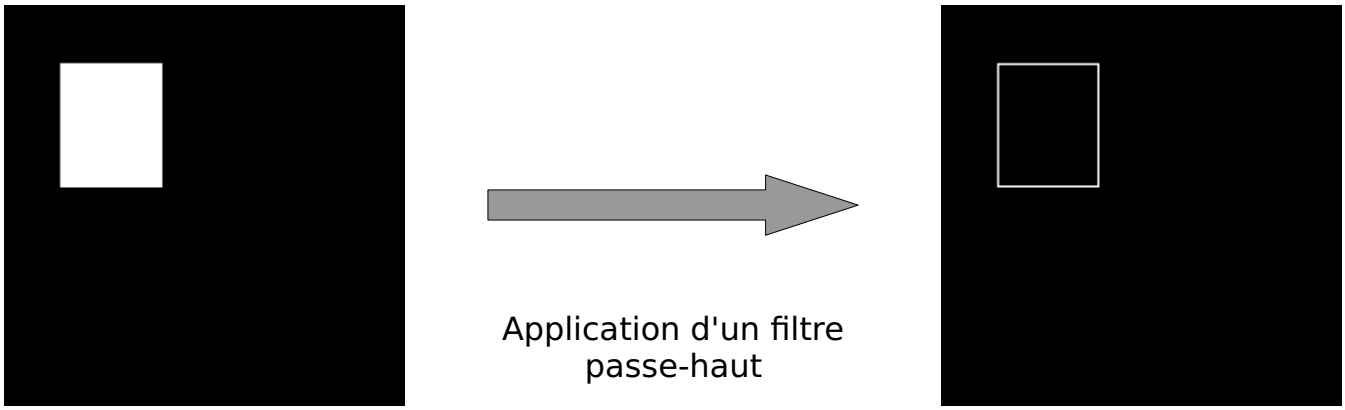
Nous utilisons un seuil qui se base sur le système RVB (Rouge Vert Bleu) avec chaque composante codée sur 1 octet. Les valeurs admissibles vont donc de 0 à 255.

Le seuillage d'une couleur correspond au schéma ci-dessous :



Seuillage sur le bleu

L'image résultante est en noir et blanc. Cela nous permet de réduire considérablement le nombre d'informations importantes.
Le filtre passe-haut permet ensuite de mettre en évidence les fortes variations de couleurs :



Application d'un filtre passe-haut

Le résultat nous donne un contour de la zone concernée. Cependant, il est aisément concevable que les formes que nous allons obtenir seront beaucoup moins plaisantes à traiter. La librairie OpenCV permet d'extracter des informations et cela même sur des contours non-cohérents (disjoints, nuage de points).

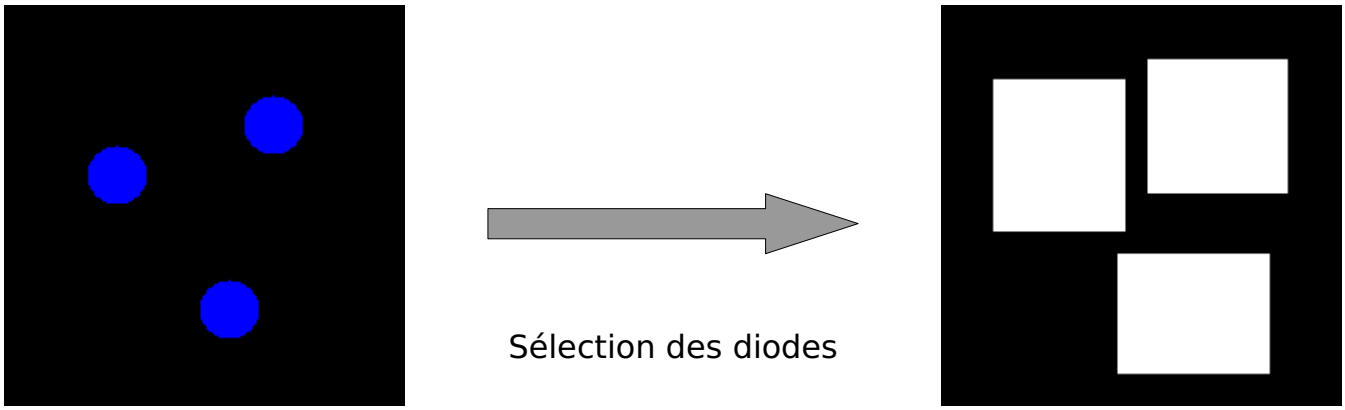
5.2.2. Filtrage

Nous avons vu que nous devions filtrer le bleu, cependant, cette couleur ne correspond pas à une valeur unique mais à un ensemble de valeurs. Il faut donc déterminer cet ensemble de valeurs. Déterminer ces valeurs est possible en effectuant différents tests sur une image de référence mais le filtre obtenu ne sera viable que pour un certain type de caméra et pour un certain type d'environnement. L'éclairage d'une pièce, les éléments qui composent le décor des images renvoyées par la webcam peuvent modifier considérablement ce filtrage.
Plutôt que d'utiliser des valeurs que nous aurions calculées à un instant t pour l'utilisation du programme, nous étions dans l'obligation de faire un système qui calcule automatiquement le filtre pour une machine, une webcam et un environnement donné sinon le programme n'aurait pas été efficace.
Nous allons maintenant voir différents algorithmes qui permettent de faire du filtrage.

5.2.3. Algorithmes de filtrage

Avant de définir des algorithmes, nous devons mettre en évidence les informations que nous possédons :
- images d'une webcam ;
- résolution de cette webcam,
- nombre d'images par seconde.

Ces trois informations triviales nous permettent déjà intuitivement de prédire que les contraintes vues en introduction pourront être respectées parce que l'algorithme effectuera le calibrage sur l'image d'une webcam donnée.
Les trois premières contraintes sont dépendantes des images de la webcam.
La dernière contrainte de distance est dépendante de la résolution de la webcam. Le calcul de distance avec une seule webcam ne peut être fait qu'en considérant une densité de points.
Ces informations sont cependant insuffisantes pour procéder au calibrage. Nous avons besoin de l'intervention de l'utilisateur pour nous spécifier la localisation des diodes. Pour ce faire, nous avons construit une interface graphique intuitive.
L'utilisateur doit faire apparaître les trois diodes sur l'écran et appuyer sur une touche pour stopper la capture des images. La dernière image capturée servira de référence pour l'algorithme de filtrage. Il doit ensuite sélectionner les zones où les diodes apparaissent.



Cette dernière étape va nous apporter une information indispensable. Nous savons désormais approximativement où se situent nos trois diodes.

5.2.3.1. Objectif / Problème à résoudre

Avec toutes les informations que nous avons accumulées, nous pouvons définir la finalité de notre algorithme. Il doit créer un filtre type seuil qui englobe le plus d'éléments possibles (pixels de couleurs) dans les zones et le moins possible à l'extérieur de celles-ci. Le filtre passe-haut ne nécessite pas de calibrage préalable de notre part car il va traiter des images binaires (noir et blanc).

5.2.3.2. Algorithmes basiques

Algorithme itératif

Le premier algorithme que nous avons testé ne nous a pas donné de résultat. Il nous aurait probablement fallu une bonne semaine avant d'obtenir une valeur de seuil. L'avantage de cet algorithme est que le seuil qu'il donne comme résultat est théoriquement le meilleur possible.
L'idée est de générer tous les seuils possibles. Pour chaque seuil ainsi créé, l'algorithme va parcourir l'image et déterminer le nombre de pixels qui respectent le seuil et qui sont hors des trois zones et ceux qui respectent le seuil mais qui sont dans les trois zones. Afin de déterminer le meilleur seuil, une heuristique très simple permet de juger de la qualité du seuil et de lui affecter une valeur.

L'heuristique est simpliste mais efficace :
- (nombre de pixels dans zone) / (nombre de pixels hors zones + 1).

Dès que des pixels sont détectés hors des zones sélectionnées par l'utilisateur, l'heuristique donne des valeurs très basses. Le "+1" permet d'éviter au programme de faire une division par 0.

```
1  SEUIL <- seuil(0,0,0,0,0,0)
2  VALEUR_SEUIL <- -(IMAGE->HAUTEUR * IMAGE->LARGEUR)
3  POUR R1 DE 0 A 255 FAIRE { POUR R2 DE R1 A 255 FAIRE {
4  POUR G1 DE 0 A 255 FAIRE { POUR G2 DE G1 A 255 FAIRE {
5  POUR B1 DE 0 A 255 FAIRE { POUR B2 DE B1 A 255 FAIRE {
6    PIXEL_DANS_ZONE <- 0
7    PIXEL_HORS_ZONE <- 1
8    POUR Y DE 0 A IMAGE->HAUTEUR FAIRE {
9    POUR X DE 0 A IMAGE->LARGEUR FAIRE {
10     PIXEL <- IMAGE->PIXEL(X,Y)
11     SI (R1<=PIXEL<=R2 ET G1<=PIXEL<=G2 ET B1<=PIXEL<=B2) ALORS {
12       SI dansZone(IMAGE->PIXEL(X,Y)) ALORS {
13         PIXEL_DANS_ZONE <- PIXEL_DANS_ZONE + 1
14       } SINON {
15         PIXEL_HORS_ZONE <- PIXEL_HORS_ZONE + 1
16       }
17     }
18   }}
19   SI VALEUR_SEUIL<(PIXEL_DANS_ZONE/PIXEL_HORS_ZONE) ALORS {
20     VALEUR_SEUIL <- (PIXEL_DANS_ZONE/PIXEL_HORS_ZONE + 1)
21     SEUIL <- seuil(R1,R2,G1,G2,B1,B2)
22   }
23 }}}}}}
```

Algorithme utilisant la programmation par contraintes

Le précédant algorithme étant inefficace, nous avons considéré le problème avec une autre logique. Nous l'avons abordé comme un problème de contraintes avec des variables, des domaines et justement des contraintes. Cependant, l'analyse nous a amenés à conclure que le temps de calcul resterait quasiment le même que pour l'algorithme basique. L'espace à explorer est beaucoup trop important. Sur ce problème, même des techniques de filtrage par backtracking avec cohérence de nœuds, d'arcs ou de bornes se révèleraient insuffisantes pour nous permettre d'obtenir des résultats rapidement. Cependant, il y aurait un gain non négligeable par rapport à la version basique, car le programme n'aurait pas systématiquement à instancier toutes les composantes RVB d'un seuil.
Une autre difficulté vient du fait que nous aurions dû utiliser un langage de contraintes comme realpaver ou prolog et il aurait été compliqué d'interfacer ces langages avec le C++.

Algorithme génétique

Utiliser un algorithme génétique pour résoudre ce problème peut sembler surprenant. Cet algorithme a été conçu dans le but d'en comprendre le fonctionnement. Appliqué à notre problème, il s'est révélé être très rapidement d'une redoutable efficacité. Il ne peut pas traiter du temps réel et n'est peut-être pas le meilleur choix dans l'absolu mais il répond à nos attentes en nous fournissant une solution de très bonne qualité en un temps record. Nous ne pouvons pas affirmer que cet algorithme génétique nous donne la meilleure solution, en revanche, tous les tests montrent qu'il en donne d'excellentes.
Nous allons voir par la suite le détail du fonctionnement de cet algorithme appliqué à notre problème.

5.2.4. Algorithme génétique

5.2.4.1. Mise en place des outils

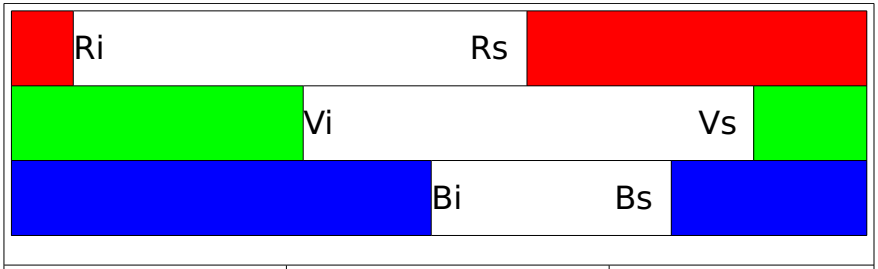
Les mathématiciens n'ont pas réussi à démontrer pourquoi les algorithmes génétiques pouvaient résoudre des problèmes. Seule l'intuition et la logique permettent actuellement de comprendre pourquoi de tels algorithmes sont capables de donner des solutions viables. Ils permettent dans certains cas de réduire considérablement le temps de calcul (notamment pour des problèmes NP). Pour résoudre notre problème, nous avons essayé de coller au mieux aux concepts imposés par les algorithmes génétiques.

Voici les notions qui vont nous intéresser :
- population ;
- individu ;
- gène ;
- mutation ;
- croisement ;
- sélection ;
- élite ;
- tri ;
- qualité d'un individu,
- aléatoire.

Nous allons essayer de caractériser, de donner un sens à chacun de ces mots.
Pour commencer, nous allons spécifier la structure d'un individu.

L'individu

C'est l'unité de base de l'algorithme génétique. Il est défini par ses gènes qui lui confèrent une certaine qualité vis à vis d'un objectif à atteindre. Dans notre cas, un individu est assimilé à un seuil défini par six paramètres, ses gènes : les bornes inférieure et supérieure des trois identifiants de couleurs : R, V et B. Il est de bonne qualité s'il permet de distinguer les diodes dans une image quelconque.



Individu	Valeur : 0,0	État : Normal

Représentation schématique de la structure d'un individu. Ri, Rs, Vi, Vs, Bi et Bs correspondent respectivement aux bornes inférieures et supérieures des composantes RVB. Ce sont les gènes de l'individu.

L'individu est donc représenté par une structure possédant 6 valeurs qui correspondent aux bornes supérieures et inférieures de nos composantes RVB. Ces valeurs vont de 0 à 255. À cela, et pour des besoins de développement, s'ajoute une variable purement informative qui permet de savoir de quoi est issu cet individu (mutation, croisement, élite).

Population

La population correspond à un groupe d'individus dont les bornes supérieures et inférieures des trois composantes RVB, c'est à dire les gènes, ont été tirées initialement aléatoirement entre 0 et 255 (cet intervalle définissant l'espace admissible des paramètres). Cette population va évoluer au cours des itérations.

Sélection

La population est amenée à évoluer. La sélection a pour objectif de permettre aux meilleurs individus de rester présents (dans la population) d'une itération à l'autre. Pour faire cela, une heuristique va donner une valeur aux différents individus. C'est l'indicateur de qualité, la fonction coût ou objectif. Avec ces valeurs, nous allons pouvoir trier notre population et garder les meilleurs éléments intacts.

Qualité d'un individu

Chaque individu est caractérisé par un indicateur de qualité (fonction coût) qui détermine s'il est « intéressant » ou non. L'objectif de l'algorithme génétique est d'obtenir le meilleur individu possible vis à vis de l'objectif fixé.

Tri

Après avoir affecter une valeur aux individus, il est nécessaire de les trier des meilleurs aux moins bons. Nous avons choisi le tri-rapide car il fait parti des algorithmes les plus constants et les plus rapides en terme de complexité $\Theta(n \times (\log(n)))$ pour trier des données.

4	0	0	0	10	6	9	0	0	2

Le tri rapide opère sur la valeur des gènes.

10	9	6	4	2	0	0	0	0	0

Élite

Les individus élites correspondent aux individus qui après le tri et la sélection font parti des 10% les mieux placés (meilleures valeurs).

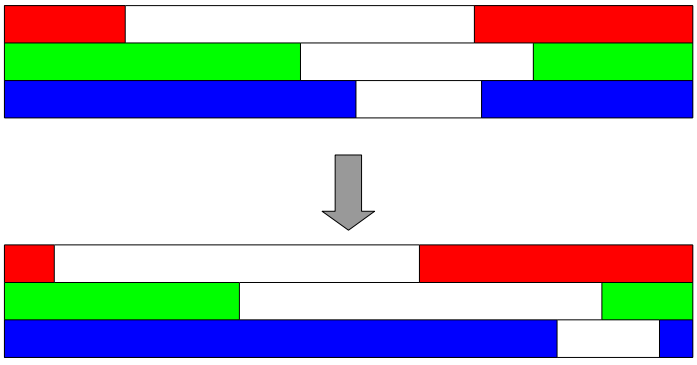
10	9	6	4	2	0	0	0	0	0

Les élites sont gardés dans la prochaine population. Ici, un seul individu est gardé (10% de la population).

10									

Mutation

La mutation est une étape importante dans l'algorithme génétique. Elle permet de renouveler la population et d'éviter la consanguinité (uniformisation de la population). Dans notre cas, la mutation consiste à modifier artificiellement les bornes inférieures et supérieures des composantes RVB de nos gènes. 30% de la population est ainsi mutée. Les individus mutés sont pris aléatoirement. Un même individu peut-être pris plusieurs fois et subir des mutations différentes.



Ri = 30, Rs = 170
Vi = 70, Vs = 200
Bi = 140, Bs = 175

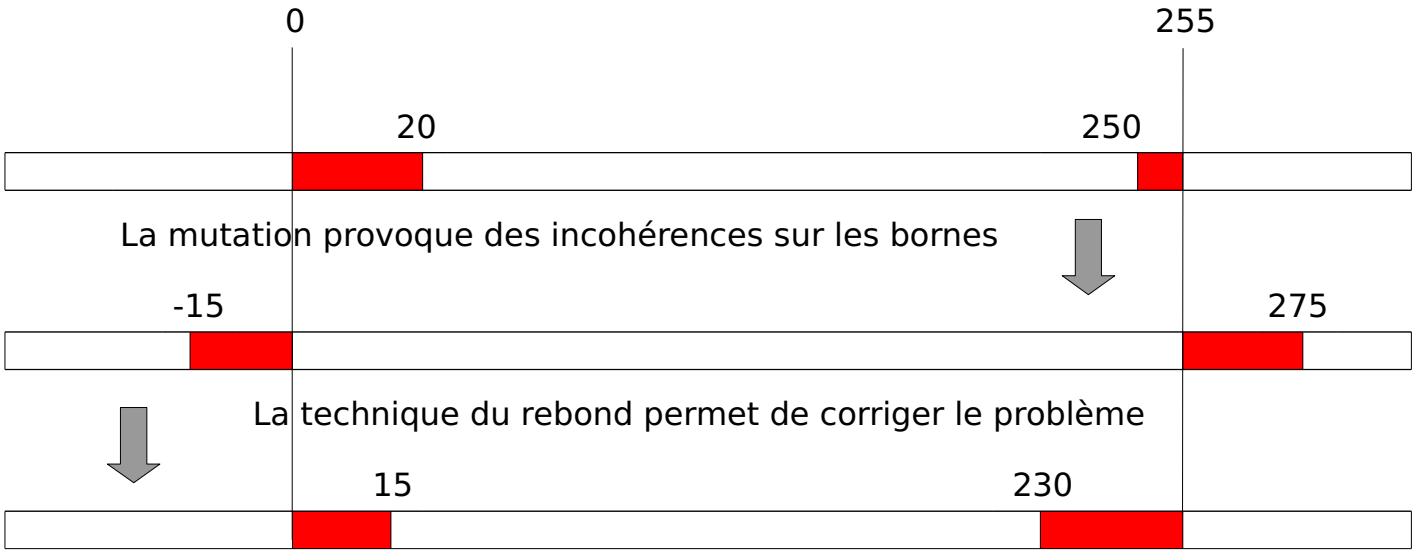
La mutation de l'individu modifie les composantes RVB

Ri = 10, Rs = 150
Vi = 50, Vs = 230
Bi = 210, Bs = 250

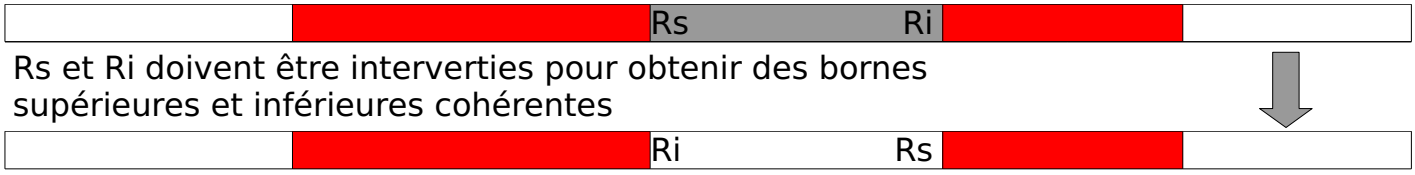
Cette opération nécessite cependant de faire attention aux bornes. En effet, pour effectuer la mutation, nous choisissons aléatoirement une valeur dans un intervalle. Par exemple entre -40 et +40. Cette valeur est ensuite ajoutée à la valeur courante de la borne.

Différents cas d'erreurs peuvent se produire :
- la borne devient négative ;
- la borne dépasse 255,
- la borne inférieure devient plus grande que la borne supérieure.

Pour corriger les deux premiers problèmes, nous utilisons une technique de rebond. C'est-à-dire que si la valeur franchit une borne, nous nous arrangeons pour faire revenir la valeur dans l'intervalle admissible.

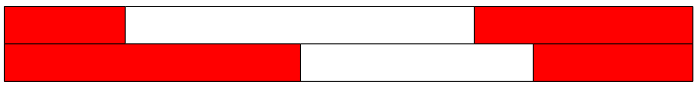


Pour corriger le dernier problème, nous effectuons la permutation des bornes.



Croisement

Le croisement correspond à la fusion de plusieurs individus. Dans notre cas, le croisement s'effectue toujours entre deux individus. L'idée de base est de comparer une à une les composantes RVB des gènes de nos individus et de faire la moyenne entre les différentes bornes. Les individus obtenus par croisement vont compter pour 60% de la population. Comme pour la mutation, les gènes de nos individus sont choisis totalement aléatoirement et un même individu peut intervenir dans plusieurs croisements.



Gènes 1 : Ri =30, Rs = 170
Gènes 2 : Ri = 70, Rs = 200

Le croisement des quatre gènes génère deux nouveaux gènes

Gènes résultant :
Ri = 50, Rs = 185

Nous allons voir par la suite que la solution de la moyenne peut-être améliorée.

Maintenant que nous avons défini tous nos outils, nous pouvons présenter le fonctionnement de l'algorithme.

5.2.4.2. L'algorithme

L'algorithme génétique garde beaucoup d'éléments de l'algorithme basique que nous avions étudié en début. Les boucles qui permettaient de générer l'ensemble des seuils vont disparaître. Cela va avoir pour conséquence de diminuer fortement la complexité de l'ensemble. Afin de faciliter la compréhension, nous allons décomposer l'algorithme en différentes étapes. Les valeurs données sont les valeurs définies actuellement dans le programme mais elles sont modifiables à souhait.

Étape 1 -> Initialisation

Pour commencer, Il nous faut créer une population. Par défaut, nous la peuplons d'une cinquantaine d'individus. Comme nous avons pu le voir, les gènes de nos individus sont tirés aléatoirement dans un intervalle compris entre 0 et 255. L'état d'un individu est défini comme "Normal" s'il ne fait parti ni des élites, ni des individus ayant été mutés ou croisés. La valeur des différents individus est mise à 0.

Étape 2 -> Détermination de la qualité des gènes

L'algorithme va ensuite entrer dans une grande boucle. Cette boucle va s'arrêter au bout d'un certain nombre d'itérations. Par défaut, la boucle fait 250 itérations.
À l'intérieur de cette boucle, nous allons parcourir tous les individus de notre population. Pour chacun de ces individus, nous allons appliquer le seuil défini par leurs gènes sur l'image.
Grâce à cela, nous allons récupérer comme informations :

- le nombre de pixels dans les zones qui respectent le seuil ;
- le nombre de pixels hors des zones qui respectent également le seuil.

Nous allons ensuite appliquer une heuristique (la même que pour l'algorithme basique). Cela va nous donner une valeur qui détermine la qualité d'un individu, la

fonction objectif à améliorer.
Une fois tous les individus analysés, nous allons les trier avec le tri-rapide du meilleur au plus mauvais.

Étape 3 -> Opération sur les individus

L'algorithme est encore à l'intérieur de la grande boucle. Il s'agit maintenant de créer la nouvelle population qui servira pour la prochaine itération. La sélection commence ici.
Il faut tout d'abord sauvegarder les meilleurs éléments de l'ancienne population. Les 10% les meilleurs vont être mis d'office dans la future population.
À cela, va s'ajouter 30% d'individus de l'ancienne population qui auront mutés. Puis afin d'obtenir une taille de population équivalente à l'ancienne, les places restantes seront peuplées par de nouveaux individus obtenus par croisements.

Étape 4 -> Nouveau cycle

La grande boucle est incrémentée et le processus recommence avec la nouvelle population.

5.2.4.3. Analyse

Il n'est pas possible de prouver la réussite (relative) de cet algorithme. En raisonnant par la logique, il apparaît que la population évolue dans le bon sens. Dans cette évolution, il y a systématiquement une phase de sélection qui va avoir tendance à privilégier les meilleurs éléments (en les gardant). Au fur et à mesure, les croisements et la mutation vont permettent à la fois d'affiner, d'améliorer les meilleurs éléments et de renouveler la population. Nous pourrions supposer qu'un mauvais individu ne sert à rien mais c'est tout le contraire. Un mauvais individu croisé ou muté peut donner un excellent individu. Il est donc important de ne pas les supprimer et de les garder pour générer les nouvelles populations. Cela permet finalement d'explorer l'espace des solutions un peu partout et donc d'augmenter les chances d'obtenir un très bon seuil.

Cependant, l'algorithme ne s'arrête pas là. Nous allons voir les évolutions et optimisations qui ont été produites à partir de cette base.

5.2.4.4. Améliorations et optimisations de l'algorithme

Afin de gagner en vitesse et en cohérence, l'algorithme a été optimisé au niveau des techniques de programmation et amélioré pour certaines opérations.

Parcours de l'image

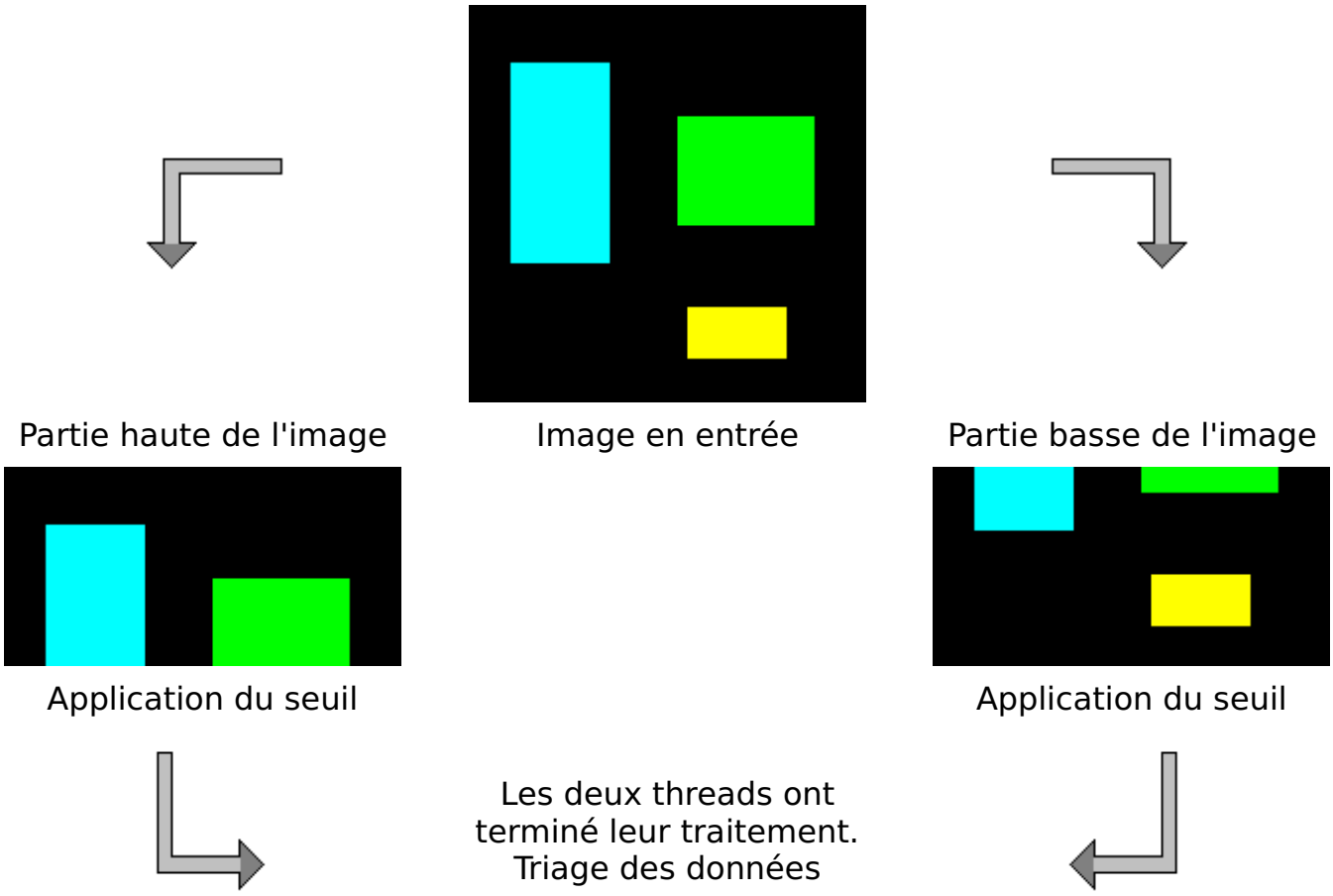
Le parcours des images coûtent beaucoup de temps de calcul. En effet, pour chaque individu l'image complète est parcourue. Afin d'améliorer la vitesse d'exécution de l'algorithme, nous avons cherché des nouvelles techniques pour extraire les informations dont nous avions besoin.
Pour cela, nous avons regardé du côté de la librairie d'OpenCV et nous avons trouvé ce qui nous intéresse.

Une des fonctions de la librairie permet d'appliquer un seuil sur une image. L'image résultante est en noir et blanc. Une autre fonction permet de déterminer à partir de cette image combien de pixels sont blancs. Cela permet donc de dire combien de pixels respectent le seuil. Jusqu'à là, c'est à peu près l'équivalent de ce que nous faisions.
Cependant, OpenCV considère l'image comme une zone mémoire et effectue des opérations binaires de style "ou exclusif" pour appliquer le seuil. En conséquence, le temps de calcul est considérablement diminué (gain de 1 minute sur les 250 itérations).

Un des défauts de cette technique est qu'il faut reconstruire à l'initialisation des images qui correspondent respectivement aux pixels qui sont hors des zones et aux pixels qui sont dans les zones.
Nous aurions pu également paralléliser le parcourt de l'image.

Par exemple :
- un thread s'occupe des pixels qui vont de 0 à 319 en largeur et 0 à 480 en hauteur ;
- un autre thread traite les pixels allant de 320 à 640 en largeur et 0 à 480 en hauteur.

Dès que les deux threads ont terminé leurs opérations, le calcul de l'heuristique est rendu possible. Cette idée est généralisable à n threads mais si l'ordinateur n'est pas multicore, l'utilité d'une telle technique est considérablement réduite.



L'intervalle des composantes RVB

Dans toute notre analyse, nous avons considéré que l'intervalle de chacune des composantes RVB allait de 0 à 255. Cependant, une image ne contient pas forcement des couleurs utilisant toute cette plage.
Avec l'optimisation du parcours de l'image, nous avons une boucle qui effectue un parcours préalable de l'image pour créer les zones mémoires contenant les pixels respectivement hors zone et dans les zones. Nous avons donc ajouté, dans cette boucle, des opérations qui permettent de définir les bornes minimales et maximales des composantes RVB.
Dans le reste de l'algorithme génétique ces nouvelles bornes inférieures et supérieures remplacent les valeurs par défaut 0 et 255.
Concrètement, cela évite à l'algorithme de générer des gènes possédant un seuil dont certaines composantes utilisent des plages de couleurs qui n'existent pas dans l'image.
Cette petite optimisation de l'algorithme génétique a cependant un défaut. La technique du rebond n'assure plus de rester dans l'intervalle admissible car la mutation peut utiliser des valeurs trop grandes.
Afin de corriger ce problème, nous avons basé l'intervalle de mutation sur les intervalles des différentes composantes. La mutation dépend donc dorénavant de l'image d'entrée.

L'amélioration des croisements

Le fait de faire la moyenne des bornes composantes par composantes correspond bien à une opération de croisement, cependant, cela ne tient pas compte de la qualité relative des individus en présence. Il est préférable, lors d'un croisement, de déplacer les bornes inférieures et supérieures en donnant un léger avantage au meilleur des deux individus.

L'idée est de donner une valeur à la nouvelle borne qui soit la plus proche possible de la borne de l'individu ayant la meilleur valeur et la plus loin possible de la borne de l'individu ayant une mauvaise valeur.
En conséquence, cela nous donne :

- (meilleure borne) * coefficient + (moins bonne borne) * (1-coefficient).

Pour le coefficient, il nous a été conseillé d'utiliser la valeur "1,2". Ainsi, la nouvelle borne ne sera plus entre les deux bornes des deux individus mais en dehors et proche de la borne appartenant au meilleur des deux individus. L'utilisation de cette technique a considérablement amélioré l'algorithme génétique. Il trouve beaucoup plus rapidement des solutions de qualités.

La terminaison de l'algorithme

Nous avons choisi d'arrêter l'algorithme après un certain nombre d'itérations. Nous sommes conscients que ce choix est contestable.
Nous avons envisagé d'autres techniques d'arrêt mais nous ne les avons pas mises en pratique car l'avantage du système par itération est qu'il permet de faire une jauge d'avancement qui a un sens pour l'utilisateur.

Pour arrêter l'algorithme, nous avons pensé à faire un système qui stocke dans une variable la valeur du meilleur individu. Si cette valeur n'est pas modifiée au bout d'un certain nombres d'itérations, c'est qu'à priori l'algorithme ne trouvera pas de meilleure solution, il s'arrête donc.

Une autre technique sur le même principe consiste à rebooter l'algorithme si la meilleure valeur n'évolue plus. Rebooter ne signifie pas l'algorithme doit repartir d'une population initiale. La population courante doit être bruitée. C'est à dire que l'algorithme modifie aléatoirement les individus (mutation généralisée). Cela permet parfois de trouver une solution qui n'a pas été trouvée initialement car l'algorithme peut avoir tendance à privilégier un certain type de chemin mais qui au final ne sera pas optimal. Enfin pour notre problème, le seuil obtenu au final n'est pas vraiment dépendant du chemin qui a permis sa création. Même si il peut y avoir des seuils proches qui donnent d'excellents résultats, ce n'est pas forcément un problème de ne pas obtenir le meilleur.

La nouvelle heuristique

Même si l'ancienne heuristique a fait ses preuves, elle ne tient pas compte de certaines particularités.
Imaginons trois zones comme présenté ci-dessous :


Zone 1


Zone 2


Zone 3

Si nous appliquons l'algorithme génétique et que les couleurs présentes dans ces zones ne sont pas présentes à l'extérieure, nous allons obtenir un seuil qui filtre le rouge. Hors le rouge n'est disponible que dans la zone 2 et la zone 3. Il serait préférable pour nous d'obtenir la couleur bleue qui est présente dans les trois zones.

Pour traiter ce problème, nous avons conçu une nouvelle heuristique dont l'objectif est de faire comme précédemment, c'est-à-dire maximiser le nombre de pixels qui respecteront le seuil et qui seront dans les zones en minimisant les pixels qui respectent le seuil mais qui sont à l'extérieur mais en tenant compte aussi des zones. Il faut que la densité de pixels respectant le seuillage dans chaque zone soit la plus importante possible.
Pour résoudre le problème, nous avons utiliser la formule de l'écart type.

La nouvelle heuristique nous donne :

- (nombre de pixels dans zone) / (nombre de pixels hors zones + 1) * [1 /
- √(1/3 *
- [(nombre de pixels dans zone 1)/(taille zone 1)]² +
- [(nombre de pixels dans zone 2)/(taille zone 2)]² +
- [(nombre de pixels dans zone 3)/(taille zone 3)]²) -
- (1/3 *
- [(nombre de pixels dans zone 1)/(taille zone 1)] +
- [(nombre de pixels dans zone 2)/(taille zone 2)] +
- [(nombre de pixels dans zone 3)/(taille zone 3)])²]

Cette formule définit la qualité de l'individu, c'est la fonction objectif à maximiser. En rouge, nous retrouvons notre première heuristique. Le reste correspond à l'écart type. La formule va permettre de donner des meilleures valeurs au seuil qui englobe des pixels dans chacune des zones.
Cependant, la formule posée ainsi ne produit pas de très bons résultats. Nous n'avons pas eu le temps de la calibrer mais il faudrait augmenter artificiellement (exponentielle, puissance de n) la valeur de l'heuristique en rouge. Avec l'écart type, l'algorithme a tendance à chercher des seuils dont le pourcentage de pixels présents dans chaque zones demeure à peu près équivalent ce qui a pour effet d'enlever des

bonnes solutions qui donnent plus de pixels respectant le seuil mais avec des densités plus faible dans certaines zones.

Conséquence des diverses optimisations

L'algorithme basique laissait supposer qu'il faudrait au moins une semaine pour obtenir un résultat, ce qui se comprend quand nous regardons de plus près sa complexité. Les premières versions de l'algorithme génétique nécessitaient à peu près cinq minutes pour offrir une réponse satisfaisante. L'algorithme a ensuite été réglé définitivement pour utiliser une population de cinquante individus et effectuer deux-cent cinquante itérations. Les premières améliorations ont permis de réduire ce temps sous la minute. Actuellement, avec nos dernières optimisations, il faut seulement une dizaine de secondes pour obtenir un résultat de très bonne qualité...

5.2.5. Efficacité de l'algorithme génétique

Dans cette partie, nous nous intéressons à l'efficacité de l'algorithme par rapport à l'analyse que nous avions faite en tout début de ce chapitre, c'est-à-dire par rapport à l'application.

La saturation, la luminosité, la balance des couleurs, le contraste, la résolution de l'image, tout cela est parfaitement géré par l'algorithme génétique. Il permet même une certaine malléabilité car la couleur des diodes n'importe plus. Enfin le bleu reste conseillé car c'est la couleur la moins présente en général dans nos environnements.
Cependant, il ne corrige pas certains aspects. Par exemple, si la webcam est instable. C'est-à-dire que sur un plan fixe, il est possible que la webcam observe des variations de couleurs. La calibration sera valable à un instant t mais pas forcement à t+1. Avec l'algorithme génétique, les seuils fournis sont cependant suffisamment restrictifs pour permettre une utilisation sans modification du calibrage.

Une des évolutions du programme pourrait être de procéder au calibrage sur un groupe d'images et non plus sur une unique image. Par exemple sur un échantillon d'une dizaine d'images. Le seuil le plus restrictif obtenu par l'algorithme génétique serait conservé.

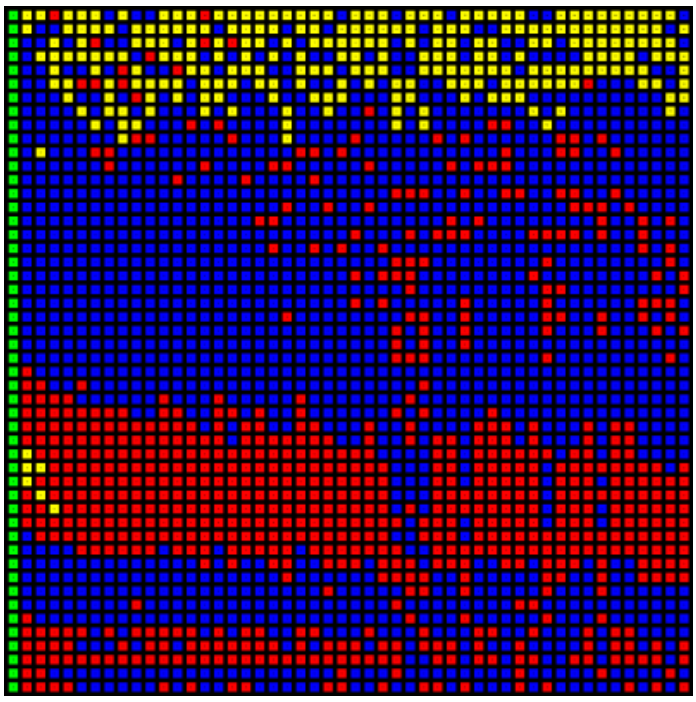
Une autre possibilité est de faire un calibrage sur une unique image et pendant l'exécution du programme, un calibrage utilisant l'algorithme génétique se ferait à intervalles réguliers en arrière plan. Par exemple, si la luminosité de l'image augmente fortement et que nous détectons tout de même la présence des trois diodes, l'algorithme principal génèrerait une zone entourant les trois diodes et appliquerait l'algorithme génétique sur l'image. Le nouveau seuil serait gardé en mémoire puis testé en parallèle pour voir s'il est suffisamment fiable pour prendre la place de l'ancien seuil.
Avant de terminer sur cette partie, voici un schéma qui montre l'évolution des gènes en fonction du temps.

Chaque case représente un individu. Plus la valeur de l'individu est élevée et plus la qualité de l'individu est bonne.

3	3	3	3	3	6	6	8	8	8
1	1	3	0	1	3	3	8	4	8
0	0	0	0	0	0	2	6	2	8
0	0	0	0	0	0	0	1	2	4
0	0	0	0	0	0	0	1	2	2
0	0	0	0	0	0	0	1	1	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

itérations	1	2	3	4	5	6	7	8	9	10

Les gènes verts sont les gènes issus de la population de départ, les bleus sont des gènes résultants de croisements, les jaunes correspondent aux élites et les rouges indiquent les gènes qui ont subi une mutation. L'affichage est volontairement décalé, c'est-à-dire qu'un gène qui apparaît comme, par exemple, un gène d'élite l'était à l'itération d'avant, c'est pourquoi les élites ne sont pas systématiquement toujours les plus en haut dans le tableau.



Cette image montre l'évolution d'une population d'une cinquantaine d'individus sur cinquante itérations. De gauche à droite, il y a l'axe du temps. Et du bas vers le haut, c'est l'axe indiquant la qualité des individus (des plus mauvais au meilleurs).

Lors du développement de l'algorithme, nous avons conçu un système capable de représenter l'évolution d'une population au cours du temps (le dessin d'au dessus est une image résultante de notre programme). Cela permet ensuite de calibrer, d'affiner les paramètres qui gèrent les mutations et les croisements.

La partie sur l'algorithme génétique est maintenant terminée. Nous allons poursuivre avec le calibrage du mouvement.

5.3. Calibrage des interactions utilisateurs / souris

Pour produire les évènements clavier et souris, il est nécessaire de procéder à certains réglages. Nous avons besoin de connaître la distance des diodes par rapport à la webcam et nous devons identifier les différents états (position des diodes) qui font le lien entre l'utilisateur et les évènements souris.

5.3.1. Distance diodes / webcam

Pour calculer la distance des diodes et donc de la main de l'utilisateur à la webcam, il y a différentes possibilités :

- l'utilisateur spécifie à quelle distance se trouve sa main par rapport à la webcam ;
- une seconde webcam, placée sur le côté, nous donne cette information directement,
- la densité de pixels des diodes permet d'estimer la distance.

La première méthode n'est pas envisageable. Cela serait trop compliqué et trop laborieux pour l'utilisateur. Il lui faudrait utiliser un mètre afin de mesurer la distance qui le sépare de la webcam ce qui paraît comme absurde pour une application de nos jours.
La seconde méthode est de loin la plus performante cependant nous voulons que notre programme n'utilise qu'une seule webcam sinon la facilité d'utilisation du logiciel et son accessibilité ne feront plus partie des qualités de notre application.
La dernière méthode apparaît comme la plus logique dans notre cas. Cependant, obtenir la densité des pixels des diodes n'est d'une part pas évident et varie d'autre part selon la distance utilisateur à la webcam, selon la résolution de la webcam et selon la position des diodes sur l'écran. Si les doigts de l'utilisateur sont inclinés, les diodes seront en conséquence moins visibles.

Par la suite, nous allons voir qu'il est nécessaire de définir une distance dite de déplacement. Cette position nécessite d'être calibrée et est basée sur un calcul de distance entre les diodes.
Si nous définissons un critère de distance, il faudra également mettre à jour ces informations en temps réel ce qui n'est pas trivial.

Le calcul de la distance entre les diodes et la webcam est donc loin d'être simple. Il est cependant concevable d'obtenir une approximation. Pour cela, il faudrait tenir compte de nombreux critères et faire des suppositions sur le comportement de l'utilisateur. Par exemple, si après capture d'une cinquantaine d'images, la taille des diodes varie très peu, alors nous pouvons supposer que l'utilisateur se tient à une distance stable et donc tenter de calculer cette distance pour en déduire la nouvelle distance de déplacement.

5.3.2. Distance de déplacement

Pour reproduire toutes les interactions de la souris, il est nécessaire d'identifier des configurations dans la position des diodes.
Nous sommes allés au plus simple et au plus efficace pour faire cela. Nous avons imaginé un cercle qui englobe les trois diodes.
Selon la taille de ce cercle, nous en déduisons des actions. Nous rentrerons dans le détail de cette partie par la suite.
Le calibrage permet d'obtenir la configuration de base, c'est-à-dire la position des diodes quand l'utilisateur souhaite procéder à un déplacement de la souris.

C'est par le biais de l'interface graphique que nous récupérons cette information.
L'utilisateur positionne les diodes de telle manière qu'il considère effectuer un déplacement de souris. Plus concrètement cela revient à écarter légèrement les doigts. Il appuie sur un bouton pour capturer l'image.
Ensuite sur cette image, il indique le centre de chaque diode en cliquant dessus. L'algorithme génétique pourra à l'avenir épargner cette étape à l'utilisateur.
Nous obtenons ensuite le centre de chaque diode pour déterminer le centre entre les diodes. Puis nous récupérons la distance du centre à la diode la plus éloignée. Cette distance servira à définir à quoi correspond un déplacement de souris avec un pourcentage qui servira de seuil de tolérance.

5.4. Conclusion

Dans cette partie, nous avons vu les deux étapes qui composent le calibrage :
- la détection des diodes avec l'algorithme génétique,
- le calcul de la distance de déplacement.

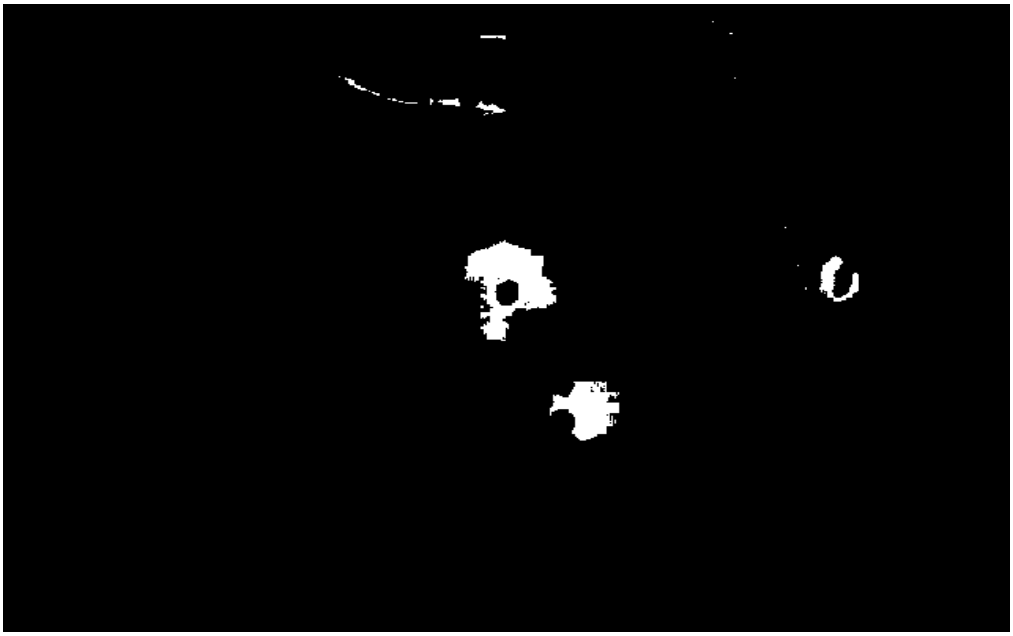
Les informations recueillies nous permettent d'éliminer une grande partie des difficultés qui viennent lors du traitement en temps réel des images mais nous allons voir par la suite que ce n'est pas totalement suffisant.
Le calibrage nécessiterait d'être cependant encore plus complet. En effet, les webcams sont en général réglables. Il est possible de modifier manuellement, la saturation, la balance des couleurs, le contraste, le gamma, l'exposition. La modification de ces paramètres peut permettre de ne détecter, par exemple, que les fortes sources lumineuses avec les couleurs qui sont proches. Le calibrage de la webcam, utilisé conjointement avec l'algorithme génétique, supprime pratiquement tous les défauts présents sur l'image.
La librairie OpenCV ne permet cependant pas encore de modifier ces paramètres et pourtant les fonctions existent mais elles ne sont pas encore implémentées. Ils nous était donc difficile de faire un système automatique de calibration de la webcam. Sous environnement Windows, les librairies natives permettent ce genre de modifications mais nous perdrions l'aspect multiplateforme de notre projet.
Cela dit, il est peut-être possible de faire une librairie multiplateforme qui tiendrait compte de la particularité de chaque système d'exploitation pour fournir à un programmeur plus haut niveau des fonctions de réglages.

6. Seuillage

6.1. Partie analyse

Nous allons vous expliquer dans cette partie, les démarches que nous avons suivies lors de l'analyse des images capturées par la webcam. Cette analyse consiste à faire un seuillage sur l'image afin de récupérer la position des diodes. Il nous a fallu également conserver la position de ces diodes afin que le contrôleur puisse y accéder comme il lui semble.

Dans un premier temps, nous récupérons le seuillage précis que nous a rendu le calibrage. Ces valeurs sont situées dans un fichier nommé « Calibrage ».
Nous appliquons par la suite, la fonction InRangeS qui permet de faire un seuillage sur l'image voulue à l'aide des valeurs de seuil récupérées. Voici ce que nous obtenons :



Nous voyons bien que sur cette image, de nombreux défauts ne nous permettent pas de reconnaître précisément la position des trois diodes. Notre problème a été lié au fait de supprimer ces parasites sans pour autant perdre les pixels correspondants à la position des diodes... Après quelques recherches, nous avons trouvé la fonction de OpenCv nommée cvErode qui permet d'appliquer une érosion sur les pixels.

Principe de l'érosion

Le principe de l'érosion est lié au fait de supprimer des pixels isolés dans l'image. On part de la matrice de pixels correspondant à l'image en noir et blanc du seuillage opéré auparavant.
Pour chaque pixel blanc, si tous ses voisins contigus sont également blancs alors on le laisse en blanc, dans l'autre cas, on le met en noir.

Voici l'érosion sur un exemple :

0	1	0	0	0
0	0	1	1	0
0	1	1	1	1
0	1	1	1	1
0	0	1	1	0

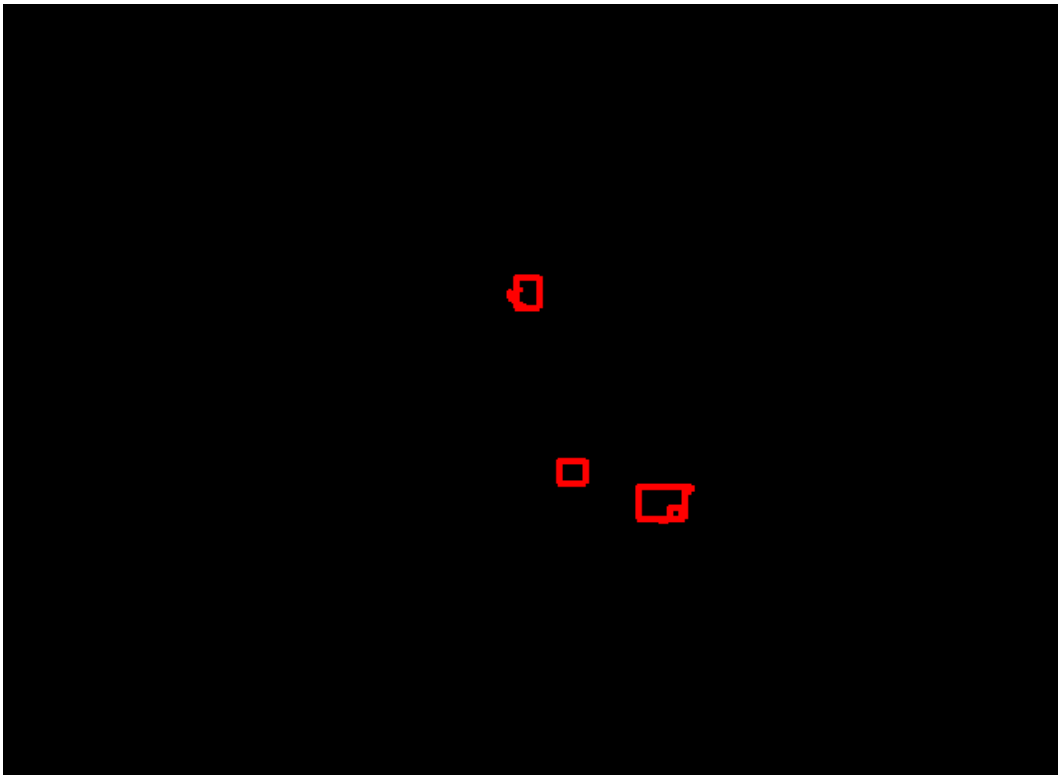
devient :

0	0	0	0	0
0	0	0	0	0
0	0	1	1	0
0	0	1	1	0
0	0	0	0	0

Voici l'image obtenu après érosion :

On remarque sur cette image que des défauts ont été supprimés. Malheureusement, d'autres défauts restent apparents.

On a testé par la suite la fonction cvFindContours qui permet de récupérer les contours dans une image binaire. Elle permet de récupérer les zones unies et blanches d'une image en noir et blanc et de les conserver dans une structure storage. Il nous suffit donc de créer pour chaque zone un rectangle englobant la zone. Le centre du carré nous permettra de savoir la position des diodes dans l'image analysée. Voici le résultat de l'application de la fonction :



On remarque que les résultats ne sont pas intéressants car une diode n'est pas définie par un seul groupement de pixels. Cela est dû au fait qu'on seuille l'image sur le bleu des diodes et étant donné que le centre de la diode est blanc, on ne peut en aucun cas récupérer une zone unie.

Il nous a fallu donc déterminer les zones des diodes de façon plus précise. Pour cela on utilise la fonction cvDilate qui permet d'unifier les groupements de pixels blancs.

Principe de la dilatation

Le principe de la dilatation est lié au fait de mettre des pixels blancs lorsqu'ils sont proches des zones contenant des pixels blancs. On part de la matrice de pixels correspondant à l'image en noir et blanc du seuillage opéré auparavant.
Pour chaque pixel, si au moins un de ses voisins contigus est blanc alors on le met en blanc, dans l'autre cas, on le laisse en noir.

Ainsi, cette matrice :

0	0	0	0	0
0	0	0	1	0
1	0	0	0	1
0	0	0	0	0
0	0	0	1	0

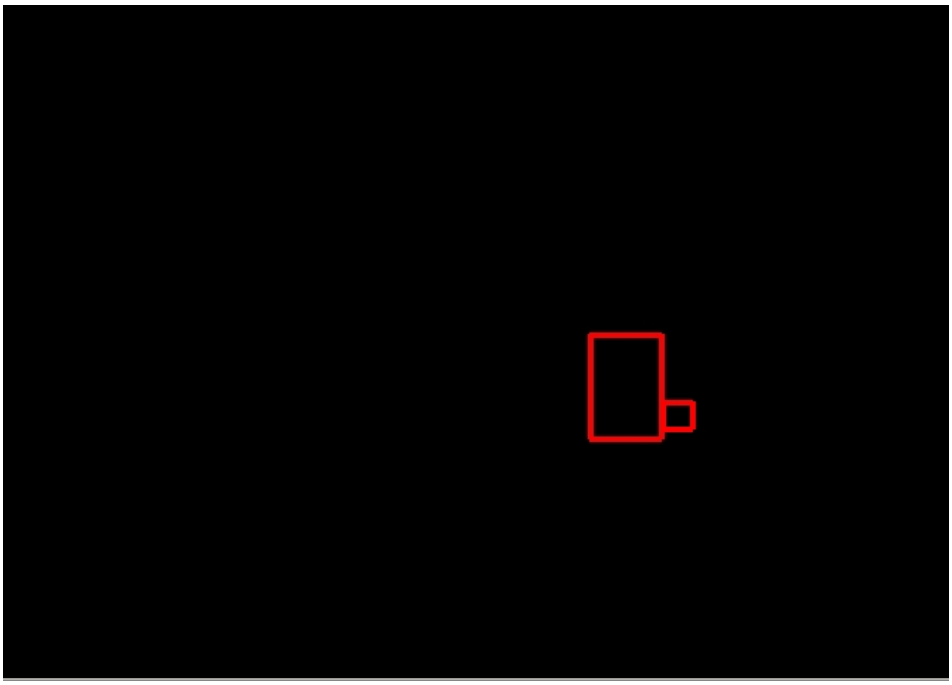
devient :

0	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	0	0	1	1
0	0	1	1	1

On obtient comme image, avec application d'une alternance des fonctions dilate et d'erode, le seuillage suivant :

Ce résultat se rapproche de ce qu'on souhaitait, mais des cas particuliers comme une forte intensité lumineuse, provoquaient encore des défauts et donc une mauvaise détection des diodes. L'autre problème est perceptible au moment où l'on rapproche trop les doigts les uns des autres. En effet, ce rapprochement provoque la détection d'une seule zone de pixels blancs et donc la fonction cvFindContours ne reconnaissait qu'un seul contour, ce qui est préjudiciable.

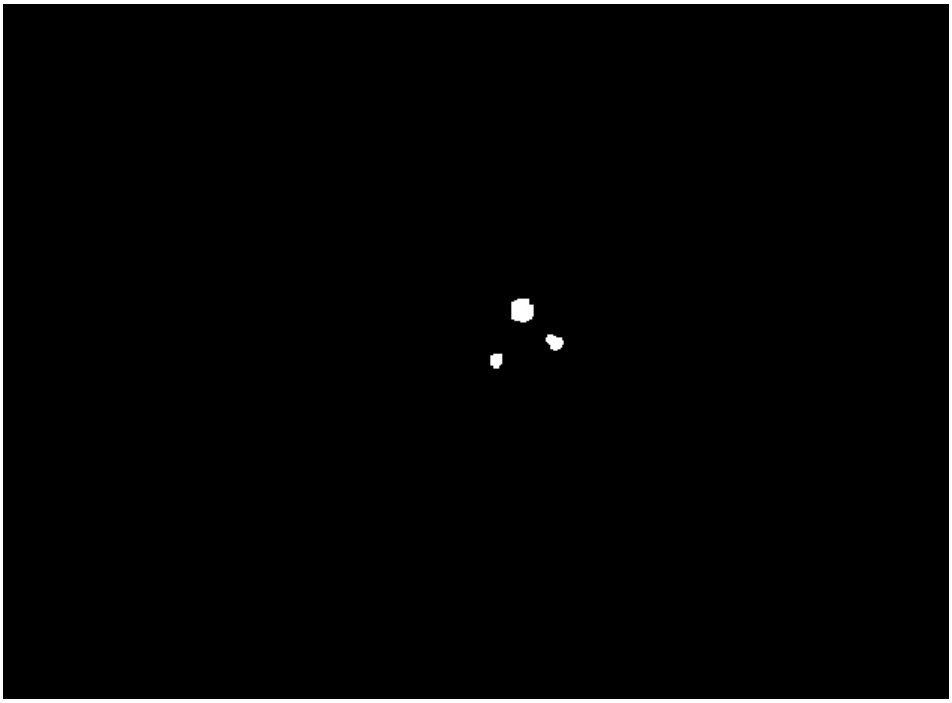
Voici l'image correspondant à ce problème :



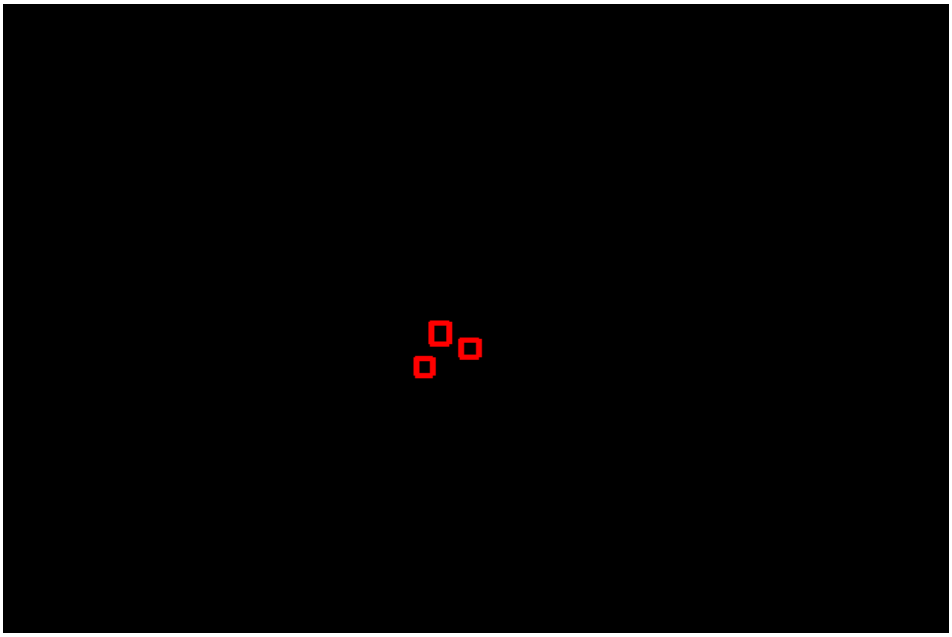
Il nous est venu alors l'idée d'utiliser le fait que le centre des diodes émet une lumière blanche.
Le but de cette idée est de récupérer le blanc de l'image d'origine situé dans les zones des diodes que nous avions trouvées auparavant grâce au seuillage sur le bleu. Cela permettra d'avoir une zone correspondant au blanc de l'intérieur des diodes qui sera plus précise que celle récupérée en faisant le seuillage sur le bleu des diodes. Elle permettra également d'enlever les défauts qui restaient, car les zones correspondant à ces défauts ne contiennent pas de blanc.

Pour opérer cette technique il faut d'abord faire un seuillage sur le blanc. Voici l'image seuillée obtenu :

On remarque que les zones correspondant au blanc sur l'image d'origine sont en noir dans cette image seuillée. Cela permettra d'appliquer l'image seuillée sur le bleu comme masque sur celle-ci. Il suffit donc de faire la soustraction entre l'image seuillée avec le bleu et l'image seuillée sur le blanc à l'aide de la fonction cvSub. Voici le résultat obtenu après la soustraction :

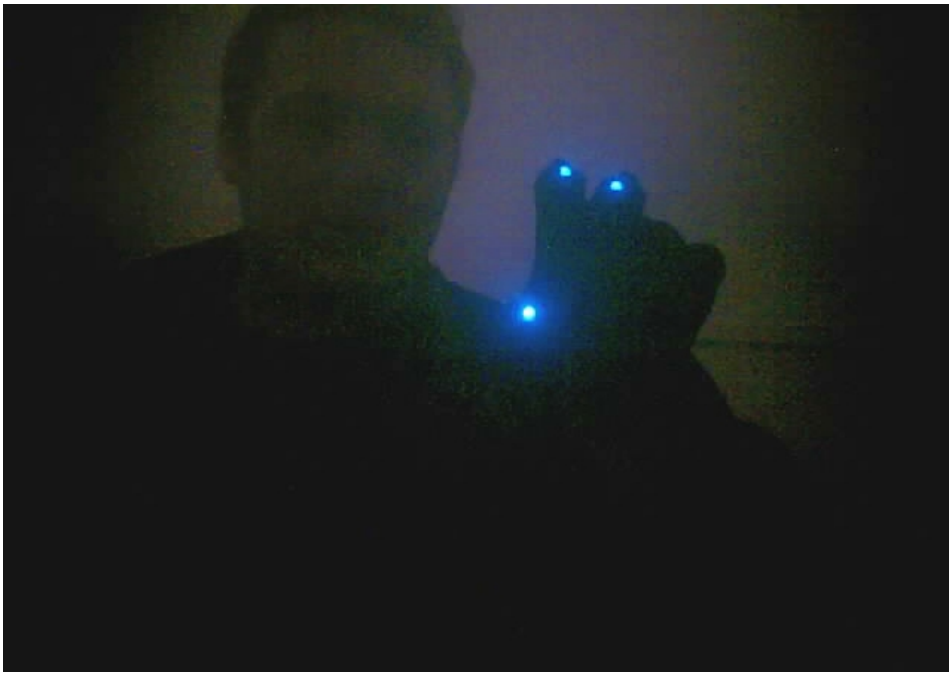


En appliquant la fonction de détection de contours, on obtient donc l'image suivante :



On remarque que la détection des diodes est beaucoup plus fine et nous permet de conserver la position des 3 diodes, même lorsque les doigts sont très resserrés. Nous nous sommes rapprochés de ce que nous espérions dans la reconnaissance de la position des diodes. Au fur et à mesure des tests, nous avons remarqué qu'il pouvait dans certains cas, y avoir des défauts récurrents principalement dus à des changements de luminosité...

Nous nous sommes rendu compte qu'il fallait préalablement appliquer un filtre (physique) sur l'image d'origine afin d'ôter la saturation que peuvent provoquer certaines webcams. Nous avons recherché dans les fonctions prédéfinies d'OpenCv. Malheureusement, ces fonctions ne sont pas encore implémentées. Nous utilisons donc provisoirement un négatif photo apposé à la lentille de la webcam. L'image d'origine possède donc beaucoup moins de saturation. Voici l'image correspondante :

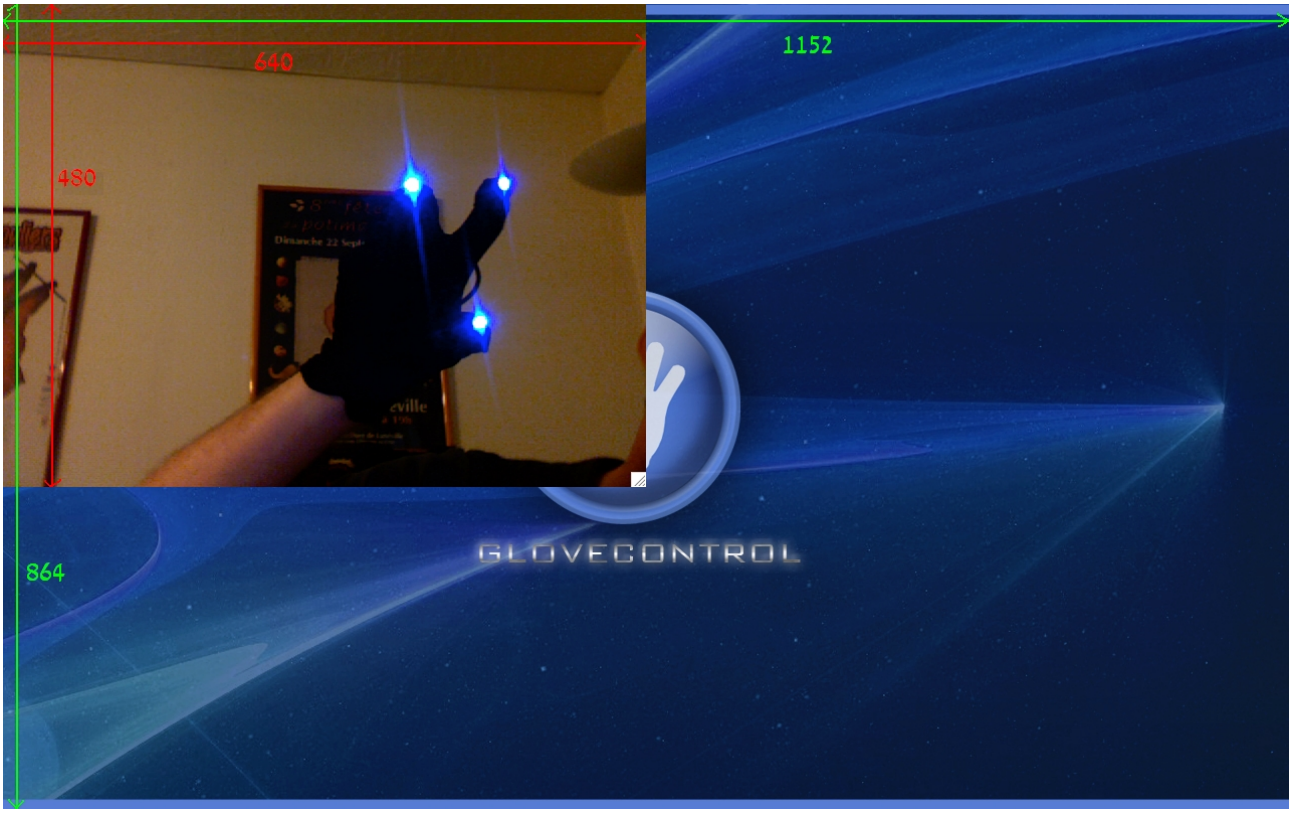


Cette astuce nous permet dès le départ d'enlever la quasi-totalité des défauts, ceux-ci étant principalement dûs à une lumière indirecte, qui se trouve donc éliminée par ce filtre.
Nous avons donc été moins restrictifs au niveau des seuillages afin de ne pas avoir des pertes au niveau des diodes.
Pour conclure, on peut voir que le résultat était, sans l'utilisation du négatif, trop dépendant de la webcam. Le résultat du seuillage pouvait être très bon pour une certaine webcam et assez décevant pour une autre. L'utilisation du négatif permet une certaine homogénéité dans les résultats, ainsi qu'une reconnaissance des diodes satisfaisante.

6.1.1. Gestion de la résolution de la webcam et de l'écran, ou comment atteindre les bords de l'écran

Dans cette partie, nous allons expliquer comment nous avons géré le fait que le déplacement au niveau de la webcam permet de parcourir la surface entière de l'écran. Il y a plusieurs facteurs à prendre en compte comme la résolution de la webcam, celle de l'écran...

La configuration sans utilisation de coefficient est représentable par cette image:



On voit bien sur cette image, qu'il est impossible d'atteindre tous les pixels de l'écran. C'est dû au fait que les positions retournées par l'analyseur renvoient théoriquement, des valeurs de pixels situées entre 0 et la résolution de la webcam.

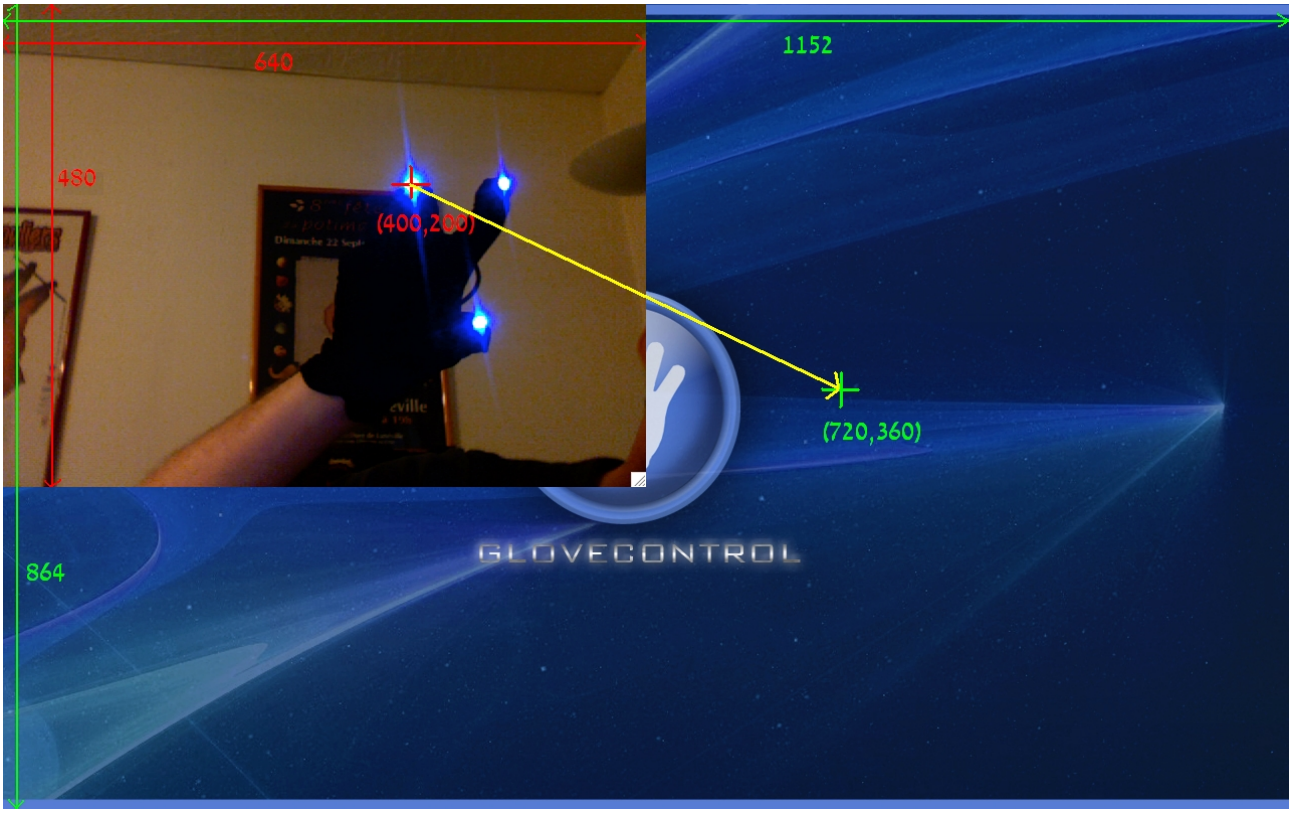
Pour résoudre problème, nous calculons un coefficient correspondant à la résolution de l'écran sur celle de la webcam. Il nous suffit ensuite, de multiplier la valeur des positions obtenues après analyse par ce coefficient pour récupérer la position correspondante sur l'écran. Voici l'image représentant cette opération :

Dans cette image, on a l'illustration de la modification de la position d'un point de la webcam après multiplication par le coefficient. Dans ce cas la résolution de la webcam est 640 par 480, et celle de l'écran est 1162 par 864.
On en déduit le coefficient suivant :

coeffx = 1152/640 = 1,8 et coeffy = 864/480 = 1,8

Il suffit d'appliquer ensuite le coefficient au point voulu. Dans l'exemple, le point analysé est à la position (400,200), on obtient donc le point correspondant :

x = 400*1,8 = 720 et y = 200*1,8 = 360

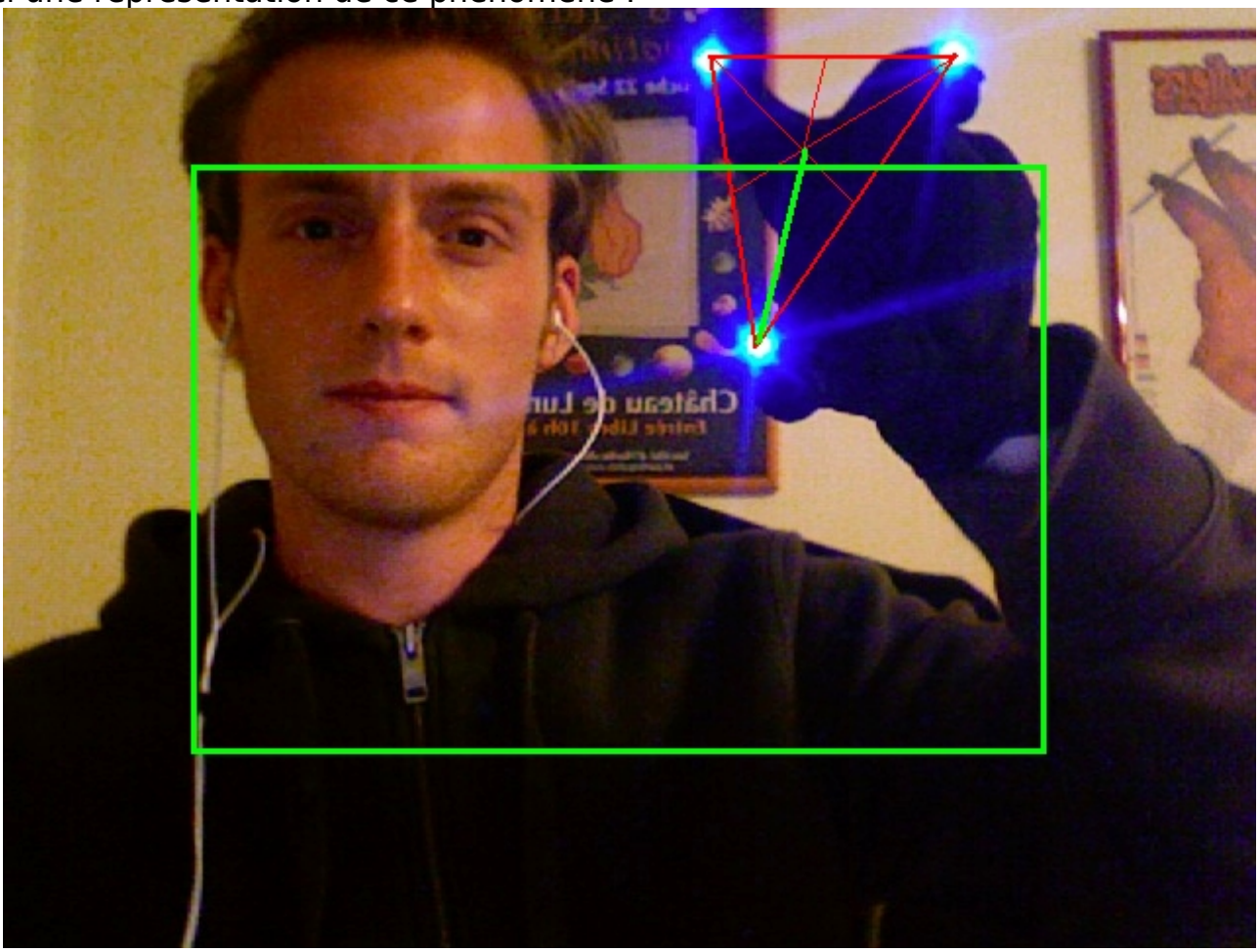


Malgré l'utilisation de ce coefficient, on peut voir qu'il est impossible d'atteindre le contour de l'image. Ce phénomène est dû au fait qu'il est impossible d'obtenir au niveau de la webcam les positions situées à proximité du bord. Le fait de prendre le centre du triangle formé par les trois diodes, ne nous permet pas d'avoir les positions du bord de l'image sans avoir au moins une diode de perdue...

Pour résoudre ce problème nous réduisons la résolution de la webcam à la résolution réellement utilisable. Pour connaître la largeur des zones non utilisables, nous utilisons une technique, qui nous paraît la plus adaptée à ce que l'on veut :

On récupère les distances entre les diodes et le centre de gravité du triangle formé par les trois diodes.
Ces distances ont été enregistrées lors du calibrage du mouvement. On multiplie la plus grande des distances par le coefficient correspondant à l'obtention du rayon pour le clic droit. Cela va nous permettre de connaître la distance maximum entre une diode et le centre de gravité que peut atteindre l'utilisateur. Cette distance correspondra à la zone théorique que ne peut atteindre le gant.

Voici une représentation de ce phénomène :



On voit sur cette photo que l'utilisateur est en mode clic droit et donc à l'écartement le plus grand possible. La distance la plus grande entre le centre de gravité et la diode (représentée en vert sur le schéma) nous permet de définir une zone à ne pas prendre en compte dans la résolution de la webcam. La zone de la webcam utilisable par l'utilisateur est à l'intérieur du carré vert.

Il nous suffit ensuite de décaler sur la gauche et en haut de l'image le pixel du décalage afin que le pixel (0,0) de la nouvelle résolution de la webcam se situe au niveau du pixel (0,0) de l'écran. Il faut ensuite appliquer le coefficient calculé comme auparavant en tenant compte de la nouvelle résolution de la webcam.

Cette technique nous permet d'atteindre vraiment toutes les parties de l'écran, sans avoir la nécessité de trop s'approcher des bords de la webcam où la reconnaissance est moins évidente.

7. Événements

Nous allons ici développer la partie du TER concernant les événements. Cette partie est différente des précédentes sur plusieurs points. En effet, les étapes précédentes ont concerné le traitement des données, tandis qu'ici nous nous basons sur de nouvelles données pour communiquer au système d'exploitation des instructions propres au curseur.
Tout au long de ce travail nous nous sommes servis de la méthode de développement basée sur UML et l'ensemble de ses diagrammes.

7.1. Première analyse du problème

Nous ne nous sommes, dans un premier temps, concentré que sur les besoins relatifs à la souris. Autrement dit nous avons énuméré les capacités de la souris que nous devions reproduire. Cet instrument qui est devenu si commun et paraît si basique est très efficace dans le sens où il remplit parfaitement le rôle qu'on lui accorde en allant à l'essentiel et de manière simple.
Le premier diagramme que nous avons fait ici fut le diagramme de cas d'utilisation. Celui-ci nous a permis d'énumérer les fonctionnalités que nous devions reproduire de manière claire et concise.

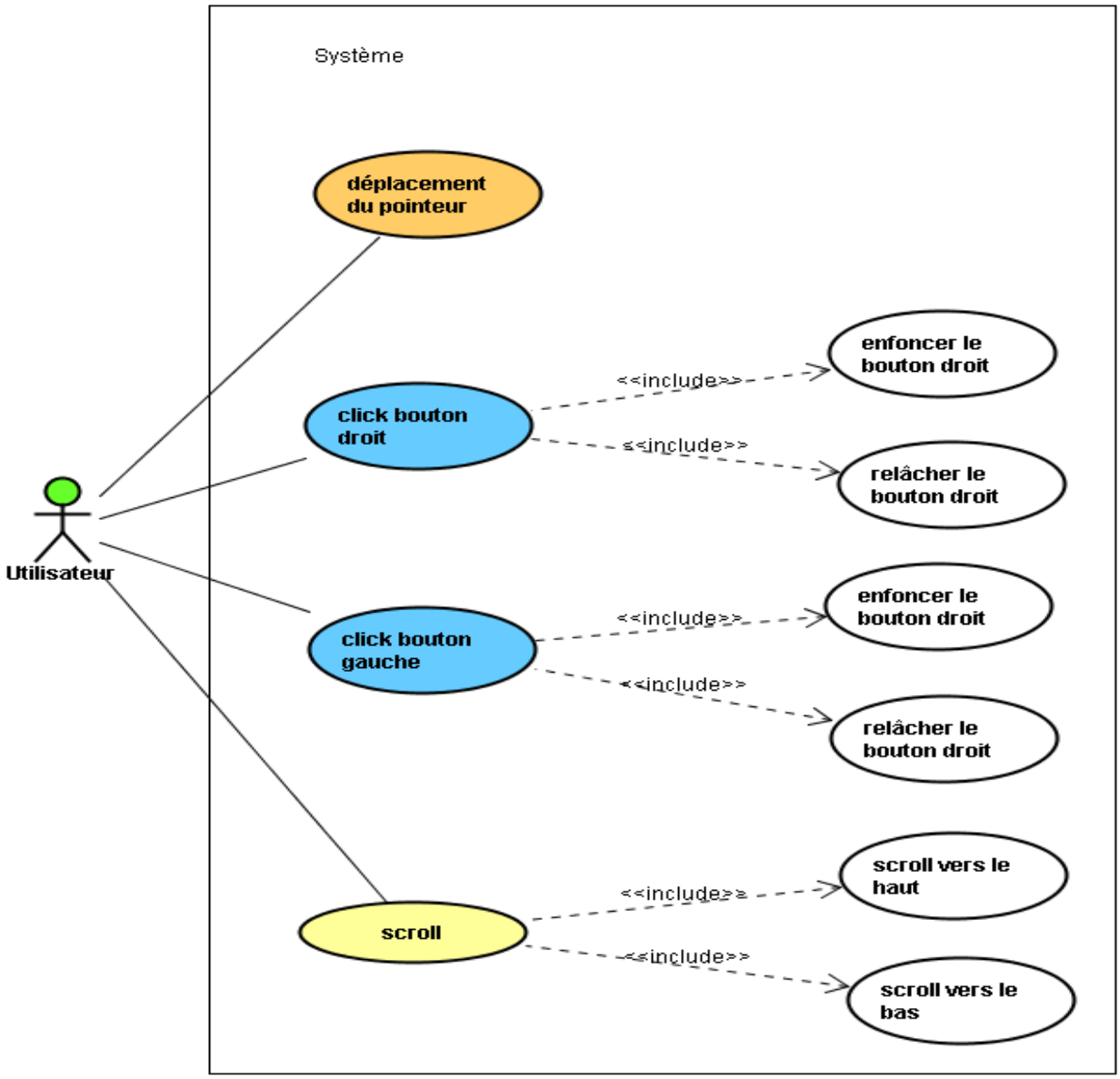


Illustration 17: Diagramme de cas d'utilisation des événements

Nous avions dans un premier temps considéré les événements « clique bouton gauche » et « clique bouton droit » sans y associer la considération d'appui et relâchement sur ces boutons. Ce geste de cliquer étant devenu si naturel il n'était pas, dans nos esprits, décomposé en deux phases distinctes. Les systèmes d'exploitations distinguent bien ces deux phases et lorsque l'on prend en compte l'action du déplacement d'un élément, sur un bureau par exemple, via le « cliquer glisser » cette décomposition des événements paraît d'autant plus évidente.
L'action « cliquer glisser » n'est pas ici présente car elle relève de la combinaison des événements décris auparavant, en effet ce n'est qu'une séquence prédéterminée ayant pour évènements ordonnés « enfoncer le bouton gauche » puis « déplacer le curseur » et enfin « relâcher le bouton gauche ». Nous n'avons ici fait figurer que les évènements atomiques qui, une fois tous mis à disposition et pouvant se combiner, amènent à l'utilisation que tout un chacun a de la souris.

7.2. Analyse approfondie du problème

Une fois la première analyse faite, il a fallu s'atteler à la différenciation de ces évènements suivant les points que l'analyseur nous communiquait. Comment discerner un clique gauche d'un clique droit à partir d'une liste de points? Nous avons pour cela adopté plusieurs choix.

Le premier de ces choix a été pour le clique gauche qui est la base fondamentale de l'interface homme machine que nous avons conçue. Il nous est apparu que la position la plus naturelle des doigts dirigés vers la webcam est celle-ci :

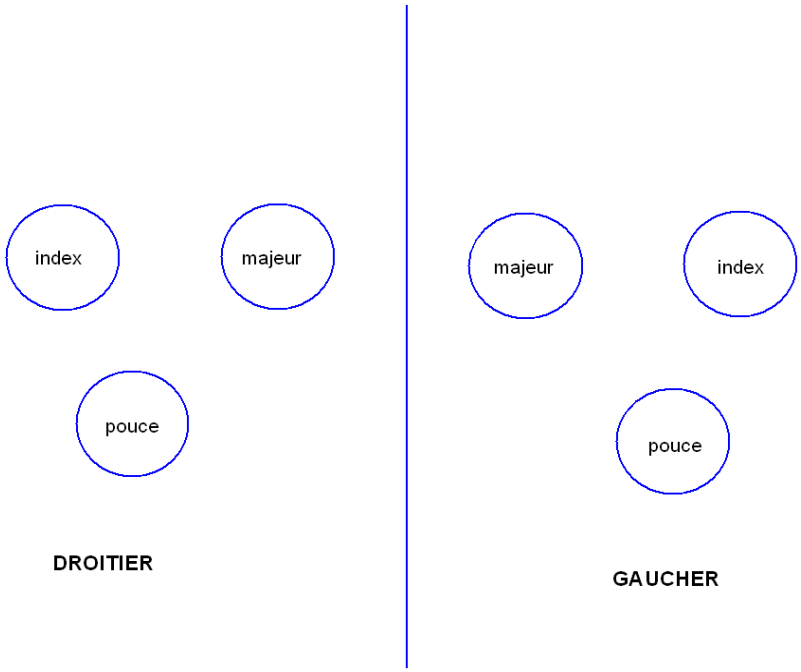


Illustration 18: position des doigts face à la webcam

Partant de cette constatation, nous avons défini le clique gauche comme étant un rapprochement significatif des trois doigts. Autrement dit, l'utilisateur rapproche ses trois doigts pour enfoncer le bouton gauche et les écarte de nouveau pour relâcher ce même bouton gauche. Ce choix nous est venu rapidement et nous a paru à la fois ergonomique et naturel.

Il a fallu ensuite définir une représentation du clique droit. Pour cela, notre première idée a été d'aligner les trois doigts ; l'utilisateur alignait donc ses doigts pour enfoncer le bouton droit puis les remettait dans leur position naturelle pour relâcher le clique. Nous avons pendant longtemps conservé ce choix pour finalement nous raviser, ce point est décrit dans la suite de ce rapport.

Concernant le « scroll », autrement dit l'action de la molette qui est devenue classique sur les souris modernes, nous nous sommes inspirés d'Apple. En effet, depuis

macOSx.4 dénommé « tiger » (et de la révisions b des machines de l'époque) il est possible d'émuler la roulette de la souris sur le trackpad à l'aide de deux doigts. L'utilisateur pose donc son index et son majeur sur le trackpad et les fait descendre ou monter pour respectivement descendre ou monter dans un document par exemple.
Ce concept est très ergonomique et était facilement transposable dans notre contexte.

Une fois ces différentes conventions prises nous avons pu concevoir un premier diagramme d'état-transition.

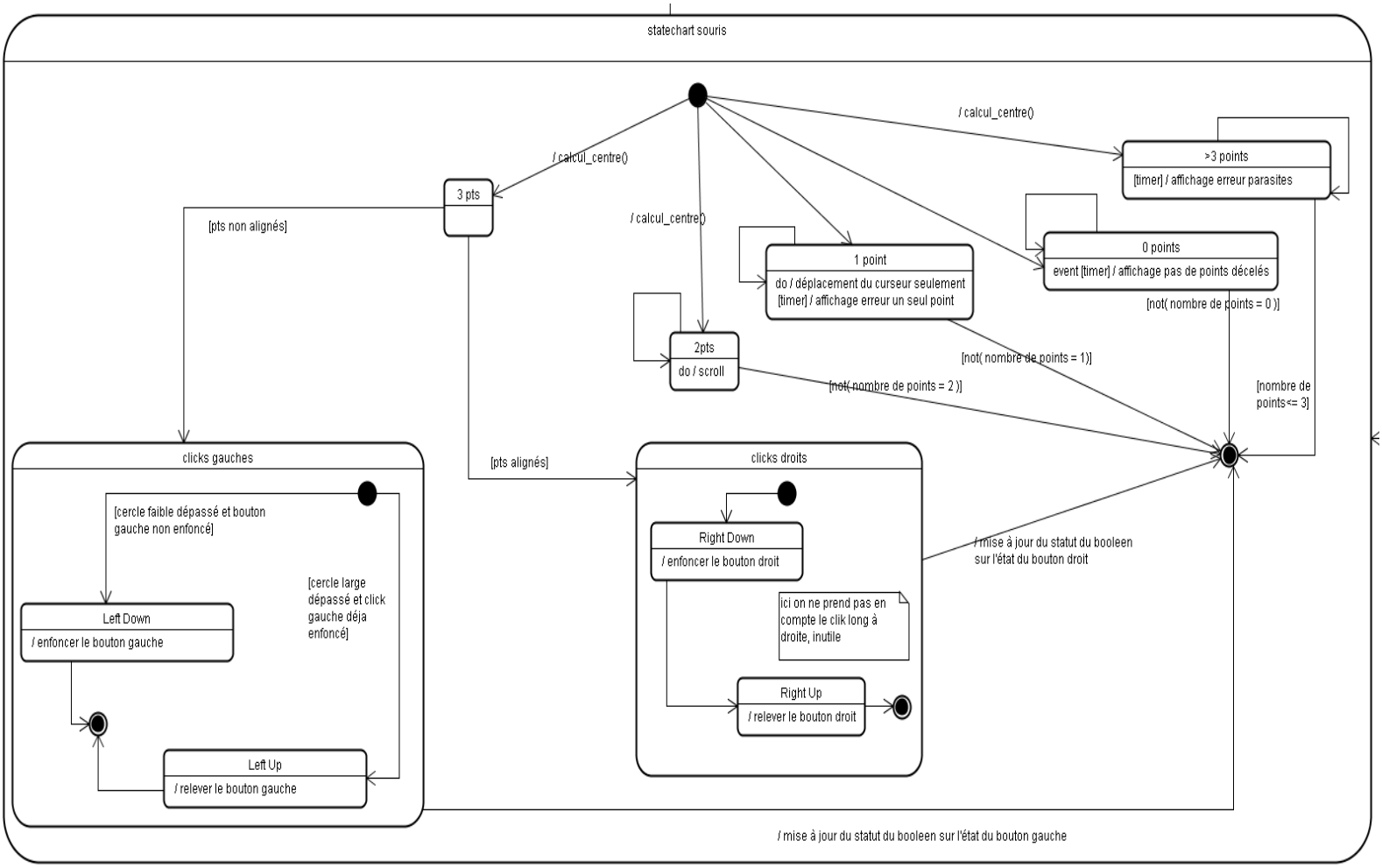


Illustration 19: premier diagramme d'état-transition pour les événements

Ici apparaissent pour la première fois des notions de cercles large et faible. Ceux-ci représentent une sorte de frontière virtuelle que l'utilisateur dépasse en écartant ou en resserrant les doigts, le centre de ce cercle étant au centre des diodes.
L'utilisateur en resserrant ses doigts pour faire un clique gauche va rapprocher les diodes qui pourront alors être comprises dans un cercle restreint, c'est ce cercle qui est appelé ici « faible ».
De la même manière pour relâcher le bouton gauche il est nécessaire d'écarter les doigts et ainsi franchir un cercle virtuel avec les diodes que l'on a nommé ici « cercle large ».

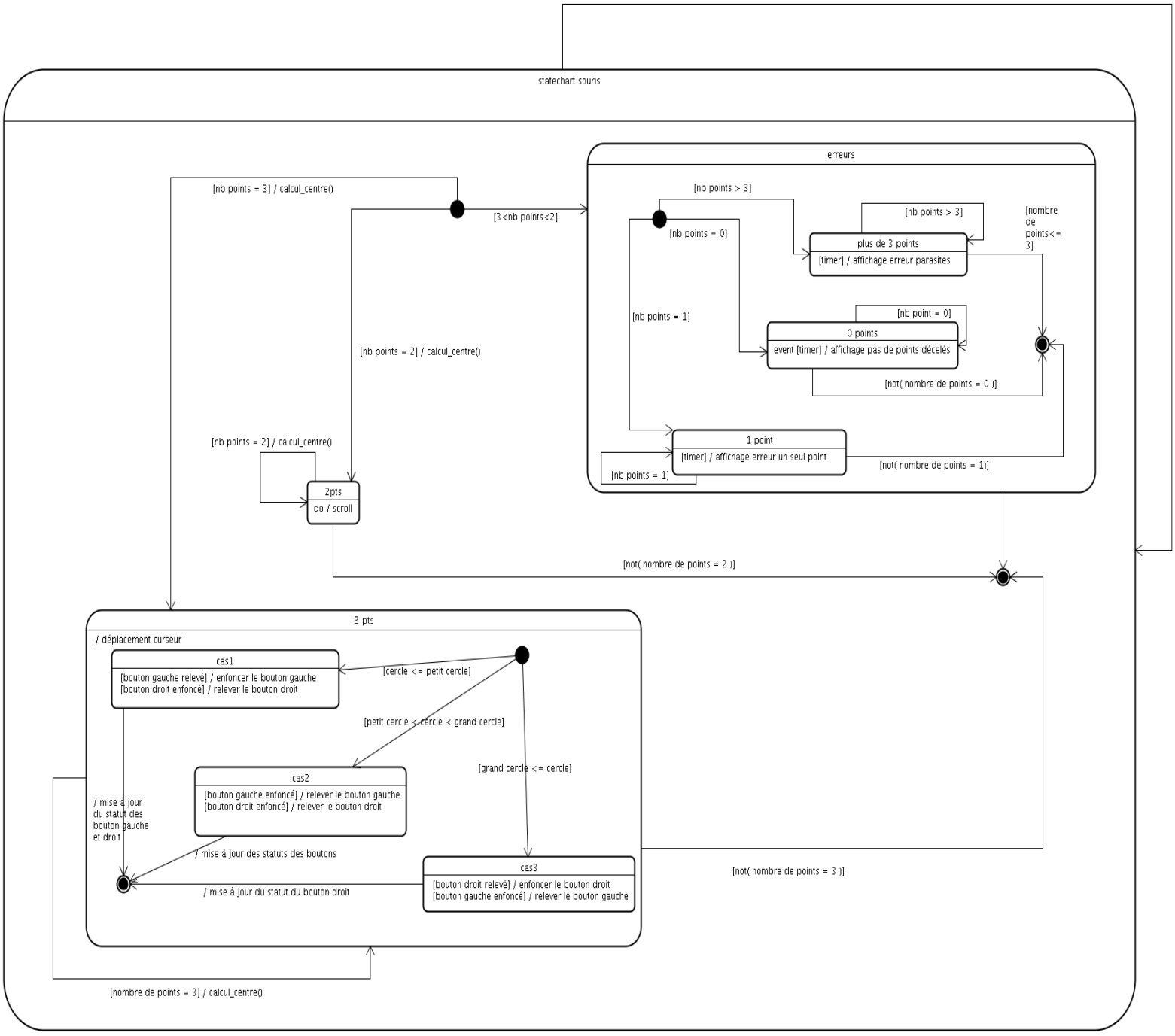
Ces gardes sur les transitions sont couplées à la vérification de l'état des boutons afin d'éviter de multiplier les événements similaires. Si l'utilisateur fait un clique gauche pour saisir une icône et veut la déplacer il risque de rester avec les diodes rapprochées pendant quelques secondes, dès lors il est inutile qu'un événement « clique gauche » soit envoyé en continu au système d'exploitation. Ce dernier gère ce problème bien évidemment, le but de ce test supplémentaire n'est pas d'éviter une erreur à un niveau plus bas mais plutôt d'éviter une surcharge du programme.

Concernant le clique gauche vous pouvez voir qu'ici nous avons donc pris en compte la garde consistant au fait que l'utilisateur aligne ses trois doigts. Cet état n'étant que passager nous avons ici symbolisé l'enchaînement de l'enfoncement et du relâchement du bouton droit. Ce point du diagramme n'est pas bon, nous verrons pourquoi par la suite avec un diagramme plus abouti.

L'état traitant du « scroll » est très basique avec une garde d'entrée et de sortie vérifiant le nombre de points. C'est la partie la plus simple du diagramme.

Nous avons ensuite séparé les différents cas d'erreurs suivant s'il y a zéro, un, ou plus de trois points. Cette partie du diagramme est également perfectible, nous allons voir pourquoi dans la suite.

7.3. Fin de l'analyse du problème



Le premier changement majeur est la nouvelle représentation du clique droit, en effet nous avons abandonné l'idée des doigts alignés. Cette hypothèse amenait à faire une approximation de l'alignement avec une tolérance d'écart qui était soit trop permissive soit trop stricte suivant les cas. De plus le geste associé n'était pas naturel et loin d'être ergonomique.

Nous avons donc opté pour la solution complémentaire au clique gauche. L'utilisateur doit donc écarter les doigts pour enfoncer le clique droit et revenir à la position naturelle pour que le clique droit soit relâché.

Cette nouvelle hypothèse nous a amenés à définir deux cercles concentriques centrés sur le point d'équilibre des diodes. Le plus petit de ces cercles permet de cliquer gauche en faisant passer les diodes à l'intérieur puis à l'extérieur, tandis que le deuxième plus large permet de cliquer droit en faisant passer les diodes cette fois-ci vers l'extérieur puis revenir à l'intérieur.
La position naturelle de l'utilisateur étant bien évidemment avec les diodes placées entre ces deux cercles.

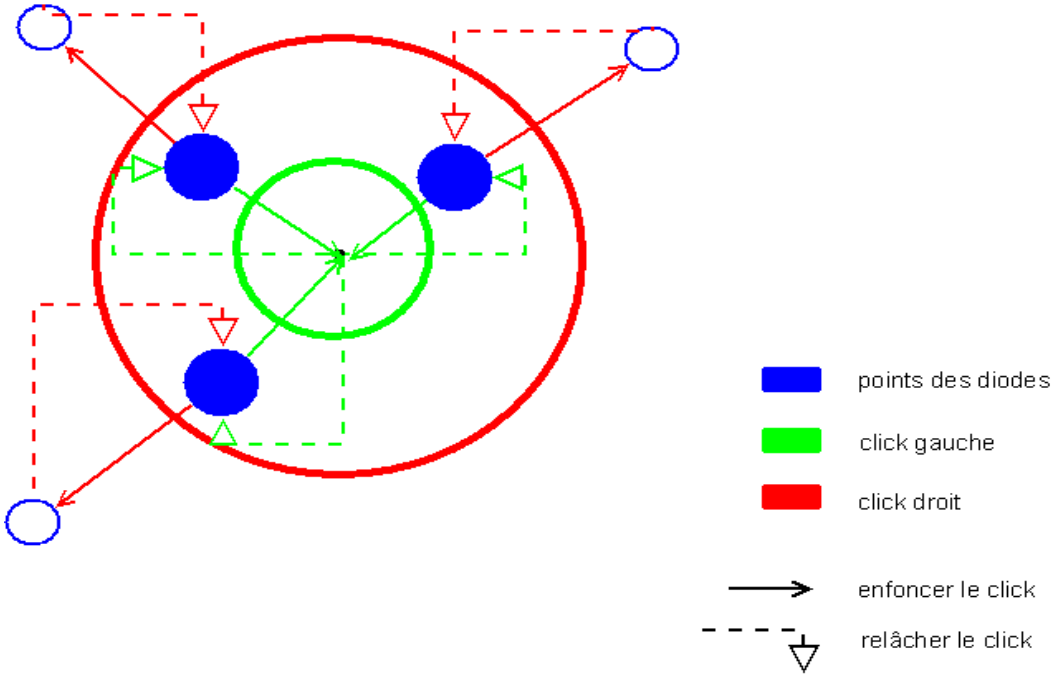


Illustration 20: représentation graphique pour droitier des cercles des événements

Ici sont représentés en gras les cercles dont nous parlions, en vert le cercle servant pour l'événement « clique gauche » et en rouge le cercle pour l'événement « clique droit ».

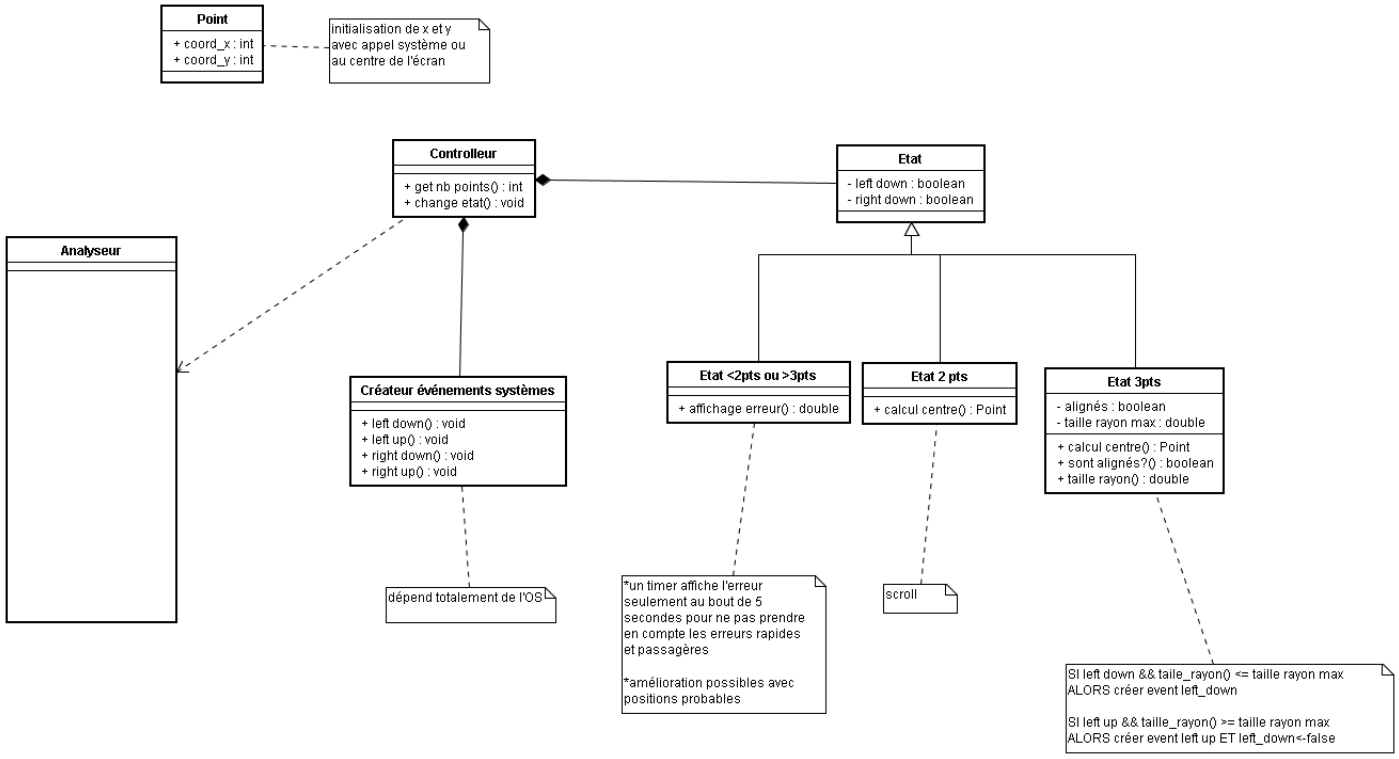
Il est à noter que cette solution apporte quelques cas particuliers qu'il a par la suite fallu gérer lors de la conception, notamment le fait que la webcam ne prenne pas forcément d'image lors d'un passage de la zone clique gauche à la zone clique droit, la mauvaise gestion de cette possibilité peut rendre le système d'exploitation complètement instable avec un bouton de la souris qu'il continue à considérer à tort comme enfoncé (gauche dans cet exemple).
La gestion du double clique qui paraît anodine est ici la partie la plus compliquée, nous la développerons dans la phase de conception qui suit.

7.4. Conception

7.4.1. Premier diagramme de classes

Une fois la phase d'analyse complétée nous nous sommes attelés à passer à la conception. De manière naturelle nous avons commencé à faire quelques ébauches de diagrammes UML de classes.

Ces premiers essais étaient dans leur grande majorité soit trop monolithiques soit trop modulaires, nous avons tenté à plusieurs reprises d'optimiser la granularité de l'application. Il nous est apparu en reconsidérant de manière plus poussée le diagramme d'état-transition que nous étions dans une situation où le « pattern état » représentait la solution la plus adéquate.
En effet l'application change de comportement suivant le nombre de diodes retrouvées dans l'image traitée.



Ici il faut bien différencier l'état du pattern et l'état du système, autrement dit l'état des boutons gauche et droit ne définit pas les différents états qui vont ressortir dans le pattern. Ce dernier doit avoir en état les éléments qui font qu'une entité généralisée agit de manière différente. Dans notre cas c'est bien le nombre de points que l'on a à traiter qui doit modifier le comportement final de l'application.
Les états des boutons gauches et droits sont quant à eux déjà présents au niveau du système d'exploitation mais nous conservons ces informations sous forme de booleens dans le pattern.
La raison en est qu'il est coûteux de faire un appel système à chaque test sur ces valeurs et cela demanderait une gestion de plusieurs plateformes dans ces appels. La

solution des booléens est très économique tant en espace mémoire qu'en temps d'exécution.
Nous avons fait une première version de ce diagramme de classe en nous basant sur la première version du diagramme d'états-transitions, celui-ci est donc erroné mais il permet de voir l'évolution des choix suivant un nouvel angle à travers ce diagramme de classes.

Nous n'allons pas nous attarder sur le défaut de conception au niveau des états qui ne correspondent pas aux différents états qui ressortent du dernier diagramme d'état-transition, ces derniers sont corrigés dans la version finale du diagramme de classes. Ce qui est important à souligner ici est la structure même du pattern état. Nous avons une classe Controller qui communique avec la classe Analyseur qui nous met à disposition les points trouvés. Seule cette classe Controller communique avec le reste de l'application. Cette classe permet également la création des évènements au niveau du système d'exploitation, ces créations seront développées dans la suite du rapport.

La classe Etat est la classe générique du pattern, elle prend le comportement d'une classe fille, celle-ci étant instanciée suivant le contexte adéquat à l'instant T, si le controller a à sa disposition 3 points la classe Etat se comportera suivant les spécifications définies dans la classe fille correspondante qui aura été instanciée.

7.4.2. Un des choix important de conception : l'accès et la modification de la liste des points

Nous nous sommes rendu compte qu'il y avait plusieurs manières de gérer la communication entre l'analyseur qui récupère la position courante entre les points et le contrôleur qui se sert de ces points afin de faire des appels systèmes. La première manière qui nous était venue à l'esprit, était liée au fait que l'analyseur communique l'ensemble des points au contrôleur à chaque analyse d'image. Nous nous sommes vite rendu compte que cette méthode n'était pas très fiable. Par exemple, si l'analyseur envoi plus d'images que le contrôleur ne peut en recevoir (l'utilisation d'une webcam avec un nombre d'images par seconde assez important peut nous causer vite le problème), alors ces images seront soit stockées et l'on risque d'avoir un décalage dans les mouvements de plus en plus grand, soit pas prises en compte et l'on risque d'avoir des résultats peu convaincants au niveau de la fluidité des gestes.

La deuxième méthode que nous avons préférée est définie de la façon suivante :

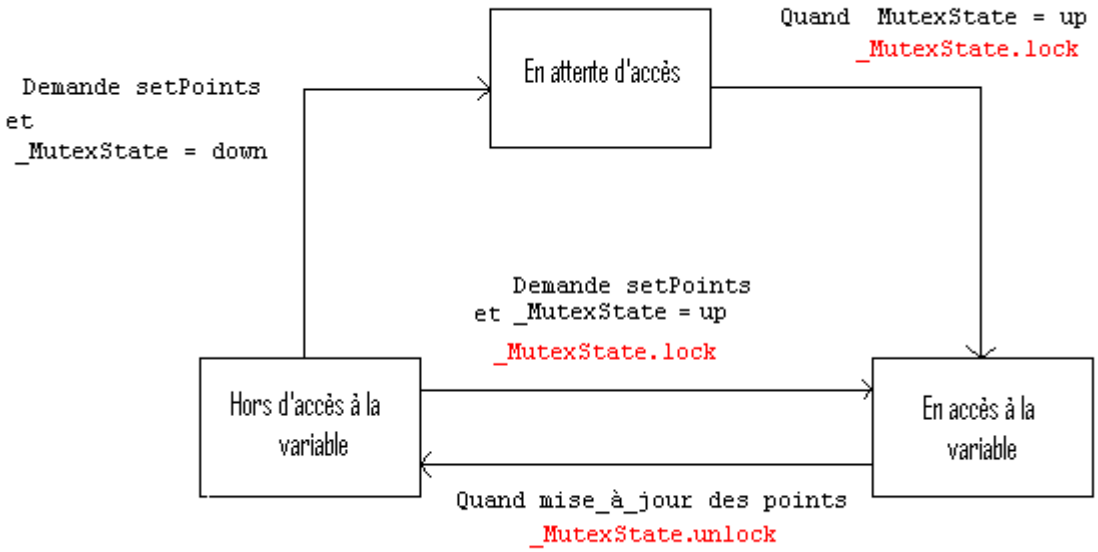
L'analyseur stocke ses nouvelles positions dans un tableau et le contrôleur récupère ces positions quand il le souhaite. Cette technique empêche que le programme ait un retard entre l'analyse des images correspondant à l'utilisation en temps réel du logiciel et la gestion des évènements par le contrôleur correspondant à la retranscription des gestes sur le curseur. Cette solution n'est pas miraculeuse. En effet, si le contrôleur demande plus d'images que l'analyse peut en fournir (Ce cas est possible quand la webcam utilisée est pauvre en images par seconde, le contrôleur récupérera les mêmes points que auparavant. Ce cas est beaucoup moins problématique que celui de la première méthode, car le contrôleur gérera ces points comme les autres comme si l'utilisateur n'avait pas fait de mouvements.

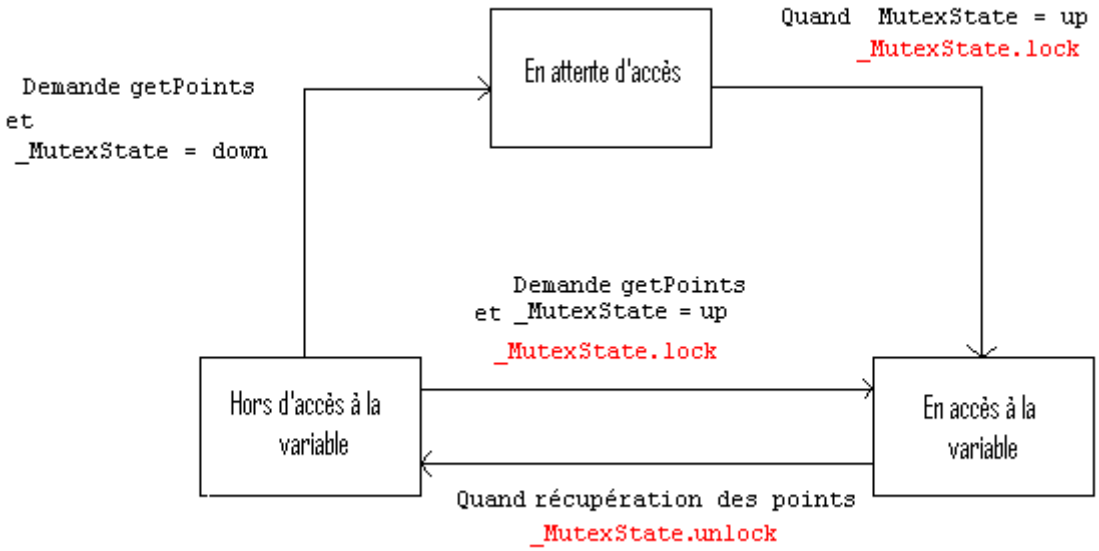
Il est facile de voir, sachant que l'analyseur et le contrôleur sont deux processus séparés, qu'il peut y avoir des problèmes de parallélisme. En effet, si le contrôleur essaye de récupérer la liste des points alors que l'analyseur est en train de la modifier, cela peut engendrer un comportement non voulu sur les appels systèmes et donc sur le bon fonctionnement du logiciel...

Il nous a fallut donc instaurer l'utilisation d'un mutex, avec pour principe celui du lecteur / écrivain. Dans notre cas, l'écrivain est l'analyseur qui, à l'aide de la fonction setPoints, modifie la valeur des points dans la liste et le lecteur est le contrôleur qui, à l'aide de la fonction getPoints, consulte cette liste de points. Nous avons instauré le mutex dans la classe de l'analyseur où est situé la liste des points qui fait l'objet de ces problèmes de parallélisme.

Voici la représentation schématique des processus analyseur et contrôleur :

Analyseur :





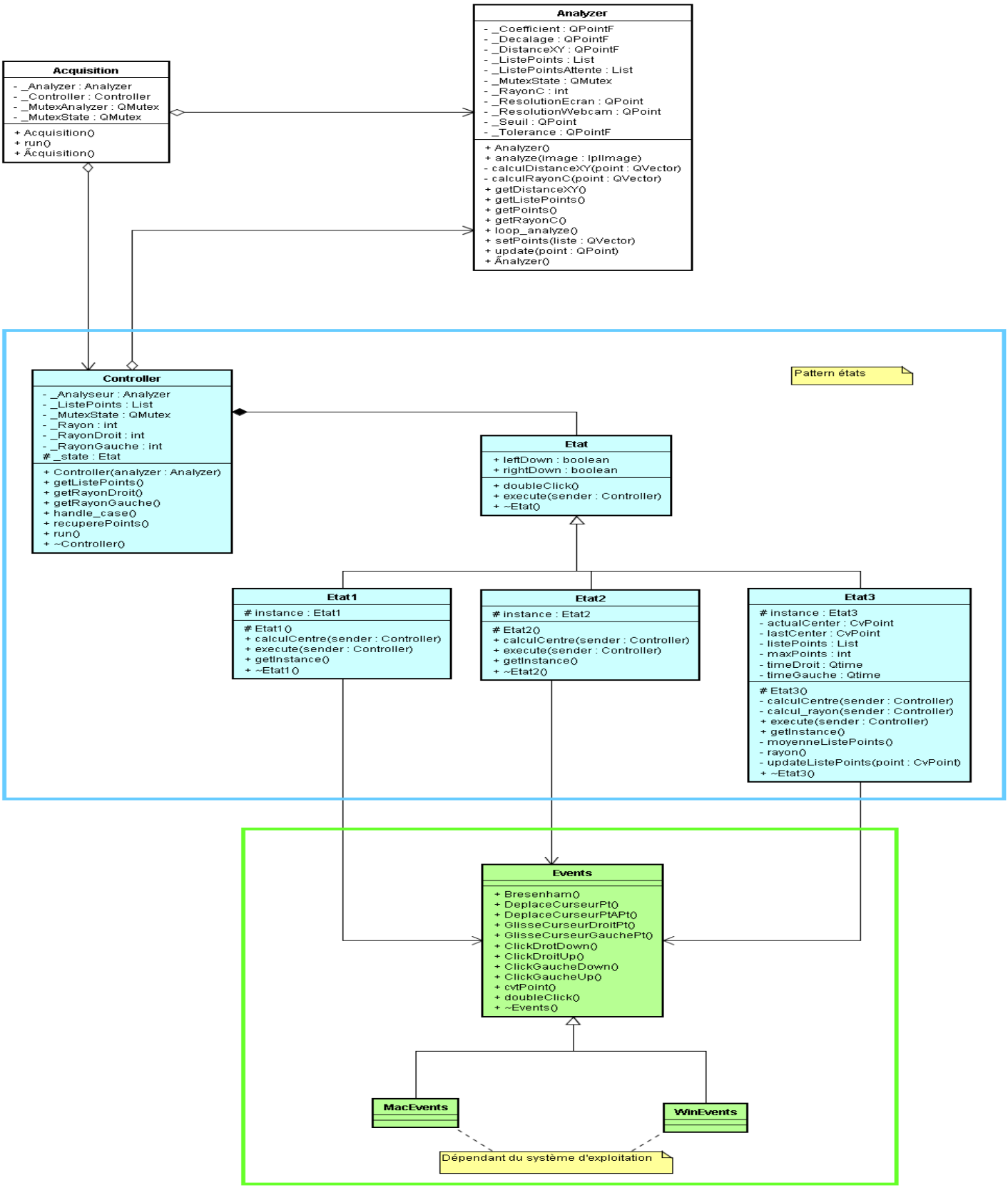
Contrôleur :

On remarque que les schéma sont assez équivalents. Dans les deux cas, si le processus peut faire un lock et que l'autre n'a pas encore lui fait de lock, alors il peut accéder à la variable partagée. Dans le cas contraire, il attend que l'autre processus fasse un unlock pour pouvoir y accéder.

Grâce à ces choix de conception, le lien entre analyseur et contrôleur permet une stabilité du programme, en évitant des temps de latence qui peuvent être préjudiciables pour l'utilisation. L'utilisation du mutex permet quand à lui, de mieux gérer l'accès à la variable partagée qui est la liste des points.

7.4.3. Diagramme de classes final

Le diagramme de classes final correspond au code de l'application. Les détails des classes winEvents et macEvents ne sont pas présents, ces classes contiennent simplement les implémentations pour chaque plateforme des méthodes présentes dans la classe Events qui est abstraite.
Nous avons sur ce schéma fait ressortir le pattern états et la partie concernant les évènements à proprement parler.

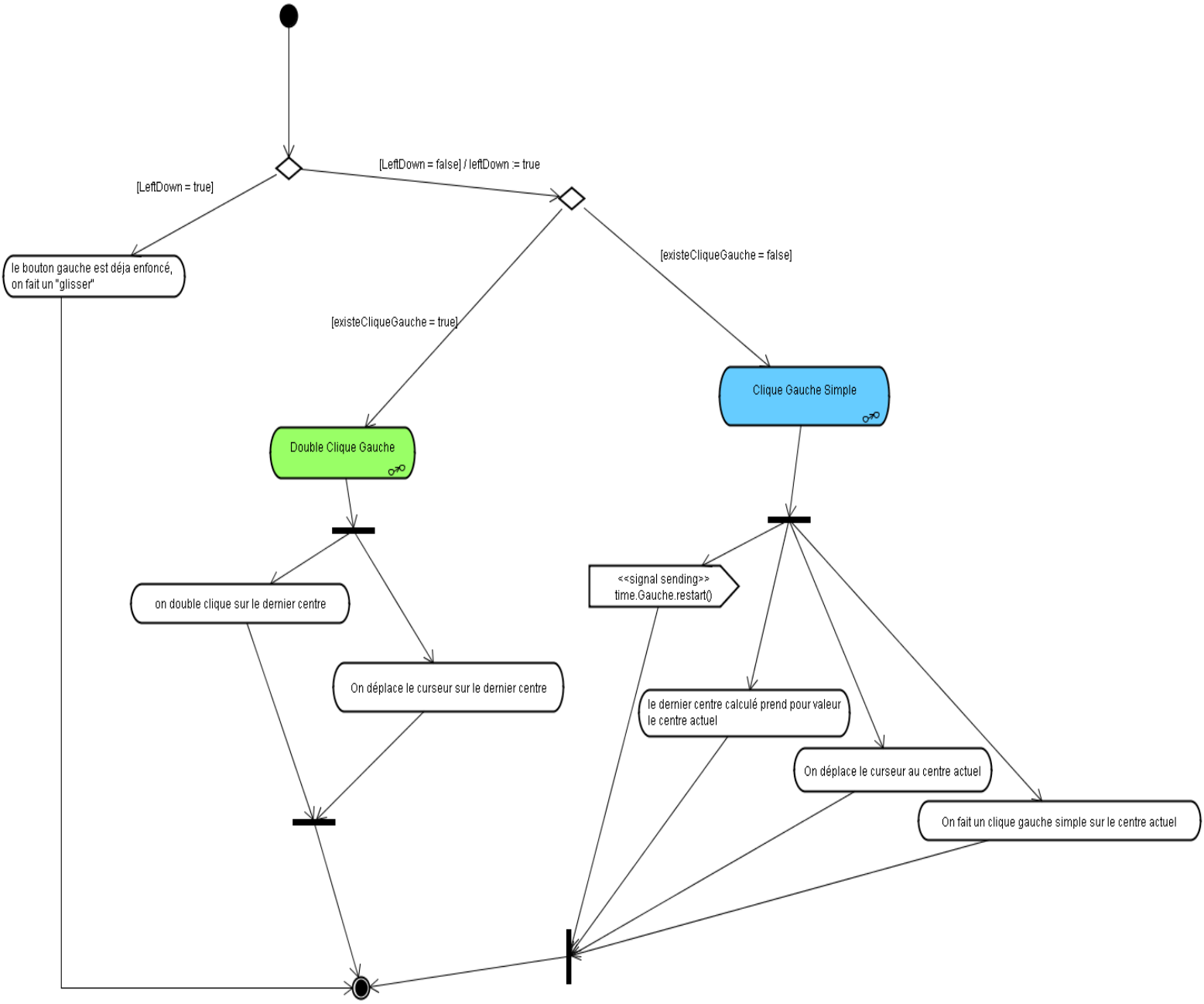

Analyzer
- _Coefficient : QPointF
- _Decalage : QPointF
- _DistanceXY : QPointF
- _ListePoints : List
- _ListePointsAttente : List
- _MutexState : QMutex
- _RayonC : int
- _ResolutionEcran : QPoint
- _ResolutionWebcam : QPoint
- _Seuil : QPoint
- _Tolerance : QPointF
+ Analyzer()
+ analyze(image : IplImage)
- calculDistanceXY(point : QVector)
- calculRayonC(point : QVector)
+ getDistanceXY()
+ getListePoints()
+ getPoints()
+ getRayonC()
+ loop_analyze()
+ setPoints(liste : QVector)
+ update(point : QPoint)
+ ~Analyzer()
Acquisition
- _Analyzer : Analyzer
- _Controller : Controller
- _MutexAnalyzer : QMutex
- _MutexState : QMutex
+ Acquisition()
+ run()
+ ~Acquisition()
Pattern états
Controller
- _Analyseur : Analyzer
- _ListePoints : List
- _MutexState : QMutex
- _Rayon : int
- _RayonDroit : int
- _RayonGauche : int
_state : Etat
+ Controller(analyzer : Analyzer)
+ getListePoints()
+ getRayonDroit()
+ getRayonGauche()
+ handle_case()
+ recuperePoints()
+ run()
+ ~Controller()
Etat
+ leftDown : boolean
+ rightDown : boolean
+ doubleClick()
+ execute(sender : Controller)
+ ~Etat()
Etat1
instance : Etat1
Etat1()
+ calculCentre(sender : Controller)
+ execute(sender : Controller)
+ getInstance()
+ ~Etat1()
Etat2
instance : Etat2
Etat2()
+ calculCentre(sender : Controller)
+ execute(sender : Controller)
+ getInstance()
+ ~Etat2()
Etat3
instance : Etat3
- actualCenter : CvPoint
- lastCenter : CvPoint
- listePoints : List
- maxPoints : int
- timeDroit : Qtime
- timeGauche : Qtime
Etat3()
- calculCentre(sender : Controller)
- calcul_rayon(sender : Controller)
+ execute(sender : Controller)
+ getInstance()
- moyenneListePoints()
- rayon()
- updateListePoints(point : CvPoint)
+ ~Etat3()
Events
+ Bresenham()
+ DeplaceCurseurPt()
+ DeplaceCurseurPtAPt()
+ GlisseCurseurDroitPt()
+ GlisseCurseurGauchePt()
+ ClickDrotDown()
+ ClickDroitUp()
+ ClickGaucheDown()
+ ClickGaucheUp()
+ cvtPoint()
+ doubleClick()
+ ~Events()
MacEvents
WinEvents
Dépendant du système d'exploitation

7.4.4. Un des choix important de conception : le double clique

Un nouveau problème est apparu lors de la phase de tests, il nous était impossible de générer des doubles cliques du fait du temps imparti à cet événement par défaut sur les systèmes d'exploitation. Une solution aurait bien sûr été de régler ce temps mais il aurait agi également sur la souris qui reste manipulable en parallèle.

Il nous a donc fallu trouver une solution pour générer deux cliques gauches dans un faible temps même si l'utilisateur prenait un temps plus long avec le gant. Nous nous sommes basés sur un système avec un timer. Ce dernier est déclenché après un clique gauche puis, si dans un temps que nous fixons l'utilisateur fait un nouveau clique gauche, nous envoyons alors l'équivalent de deux cliques gauches, générant ainsi un double clique.

Ce schéma illustre ce raisonnement :



Ce schéma illustre plus particulièrement le comportement de l'application si l'utilisateur amène les diodes dans l'espace du cercle réduit (dont nous avons parlé précédemment) qui correspond à l'espace du clique gauche.

Le booléen existeCliqueGauche présent sur le schéma est un test sur le timer, il retourne vrai si le timer est inférieur à une valeur temporelle fixée et faux dans le cas contraire.
Si le bouton gauche est déjà enfoncé l'application va opérer un « glisser », l'utilisateur est en train de bouger une icône ou est en train de sélectionner une zone à l'écran.
Dans le cas contraire nous vérifions le booléen existeCliqueGauche, s'il est à vrai nous

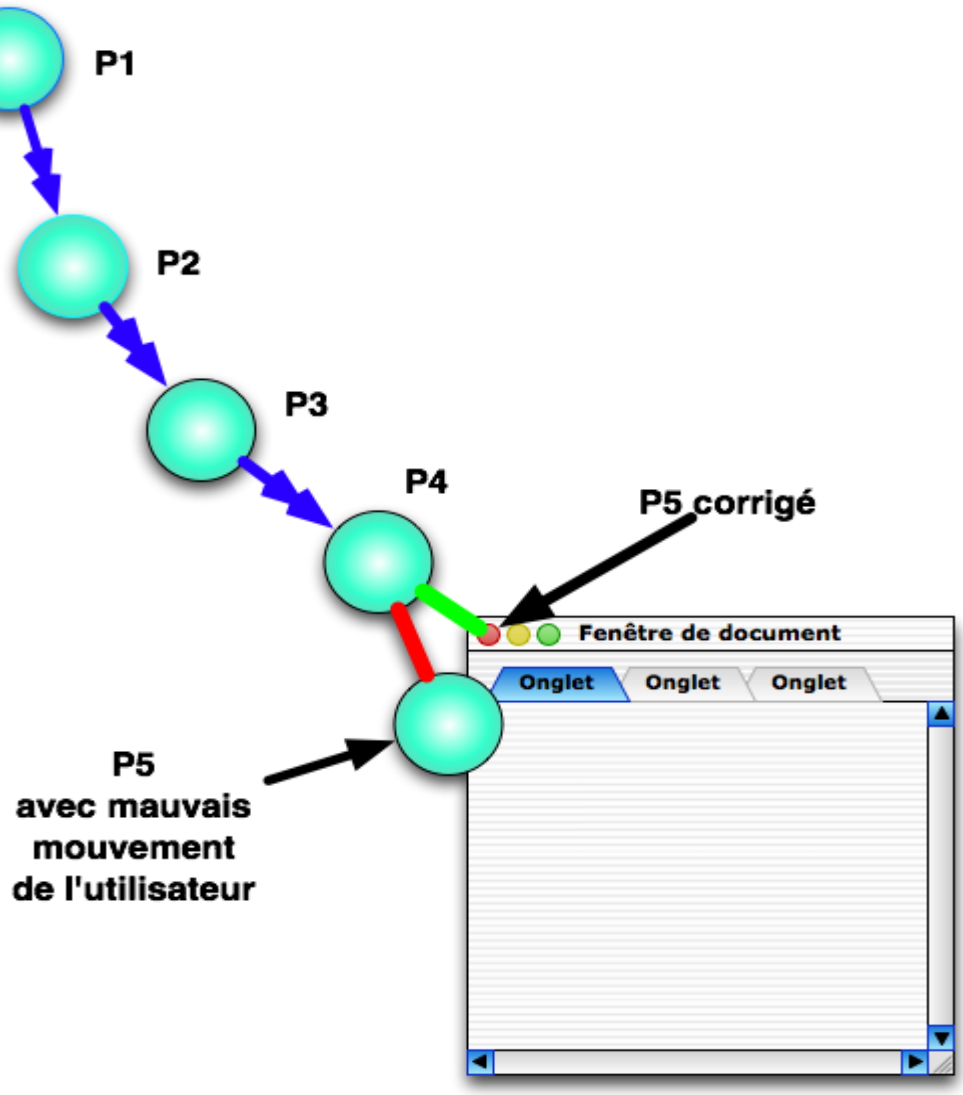
envoyons alors un double clique au système, dans le cas contraire nous envoyons un simple clique et nous réinitialisons le timer (pour pouvoir gérer un possible double clique à suivre).

7.4.5. Stabilisation des mouvements

Lors de l'utilisation de GloveControl nous avons été confrontés au problème de la stabilité, nos mouvements étaient retranscrits très précisément à l'écran et nos tremblements ou hésitations également. Il devenait alors compliqué de cliquer sur une icône avec précision ou de déplacer un élément à un endroit souhaité sans avoir à anticiper sur l'aléatoire de notre mouvement.

Nous avons donc opté pour une pondération des mouvements, chacun des points que nous envoyons au système est pondéré par les points précédents. Ainsi un écart soudain et involontaire de l'utilisateur n'a qu'une incidence faible sur le mouvement final du curseur à l'écran.

Cette pondération se fait sur un nombre donné de points, il va de soit que ce nombre de points est proportionnel au nombre d'images de la webcam que nous pouvons analyser par secondes. Faire une moyenne sur dix points est judicieux avec trente images par seconde mais est un mauvais choix avec cinq images par secondes. Cette valeur est donc fixée suivant les performances de la webcam.



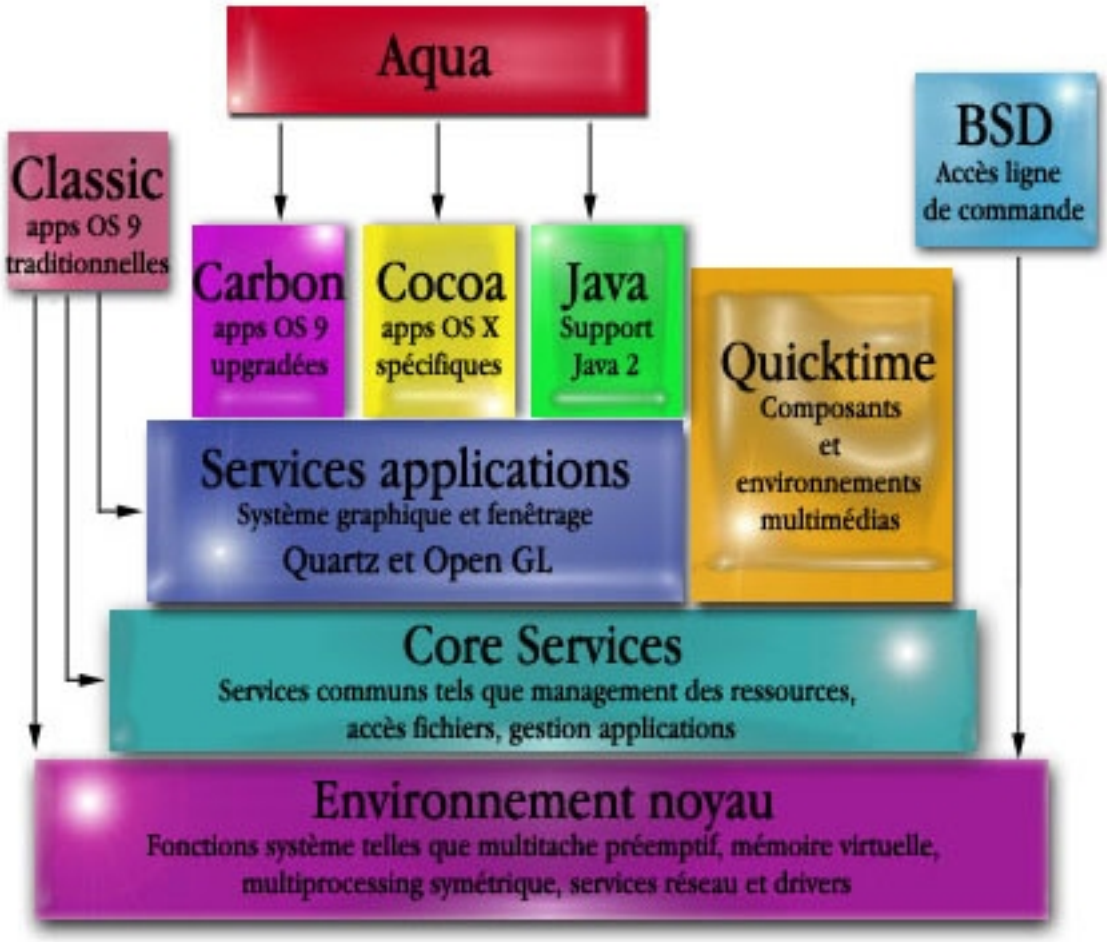
Avec cette méthode nous avons réussi à rendre GloveControl plus agréable à utiliser sans avoir une phase d'adaptation trop longue pour l'utilisateur. Ici nous pouvons voir que l'utilisateur au moment du cinquième point qui est un clic gauche fait un déplacement non voulu qui est corrigé par la moyenne pour l'amener sur le bouton voulu.

7.5. Génération des événements au niveau du système d'exploitation

Nous allons ici développer ce qui a été un des défis lors de la définition de notre sujet, à savoir la génération d'évènements que ce soit sur Microsoft Windows ou bien Mac OSX.
C'était un défi car nous avions à programmer une application de taille conséquente interagissant de manière directe avec le système d'exploitation, fait nouveau pour nous et cette interaction devait pouvoir se faire sur plusieurs plateformes.

Avec du recul cette partie du travail a été relativement rapide au niveau de la programmation, la phase la plus longue la concernant fut sans nul doute le temps de la documentation. Cette dernière est très fournie lorsqu'il s'agit de savoir comment récupérer les états des boutons de la souris mais est beaucoup plus muette sur la génération des évènements souris.

La génération des évènements n'est pas un problème abordé de manière courante dans le domaine de la programmation semble-t-il, du moins pas à un niveau si bas du système. En effet nous avons tout d'abord trouvé de la documentation sur la génération des évènements souris de manière internes aux applications, autrement dit comment forcer un comportement particulier du curseur pour une application donnée.



Ce diagramme représente les différentes couches du système OSX.
Au début de notre travail nous avions des difficultés à savoir où nous devions nous placer. Nous avons tout d'abord regardé du coté de Carbon et Cocoa, il s'est rapidement avéré que ces couches ne permettent que de créer et récupérer des évènements pour une fenêtre ou une application donnée, mais aucunement d'envoyer des évènements directement au système pour qu'ils puissent être considérés à un niveau plus global et soient accessibles par toutes les applications en cours.
Nous avons rapidement abandonné l'idée d'envoyer ces évènements à l'environnement noyau nommé Darwin, nous aurions été confrontés à des effets de bords que nous n'aurions pas pu gérer.
Ces couches du systèmes nous apparaissaient de manière extrêmement floues. Après avoir assisté à une conférence à l'école Polytech' Nantes, conduite par un programmeur Apple et obtenu un entretien avec ce dernier, les choses se sont bien éclaircies.
Nous nous sommes donc penchés sur CoreServices Quartz et CoreGraphic. Les méthodes qui nous intéressent se trouvent dans ApplicationServices.h. Ce fichier se situe dans le framework ApplicationServices disponible dans les frameworks système pour les SDK de MacOSX10.5 ou MacOSX10.4.

Pour Windows la documentation a été plus rapide car cette problématique est plus abordée sur internet. La librairie « windows.h » possède les méthodes qui nous étaient nécessaires, les moyens de créer les évènements et de les poster au système d'exploitation sont similaires dans leur mode de fonctionnement à celles d'Apple nous avons donc pu faire des classes pour chacun des systèmes et une classe abstraite sans problème.

Ces deux systèmes reposent sur un même mode de fonctionnement, les évènements sont créés en renseignant leur type et certains attributs comme le point effectif à l'écran. Ces mêmes événements sont ensuite postés dans une file d'attente du système. Cette file est commune au système entier, en opposition aux files qui peuvent être individuelles à chaque application.

7.6. Détails sur les événements système

Nous avons créé des fichiers d'évènements différents pour les évènements suivant la plate-forme utilisée.
Les clics gauches, double clics, clics droits et le déplacement de la souris sont utilisés de la même manière sur Windows et OSX, les méthodes appelées sont évidemment différentes mais la vision globale de ces évènements est similaire.
Les différences apparaissent avec le cliquer glisser pour lequel windows demande une séquence d'évènements « clic gauche enfoncé », « curseur déplacé », « clic gauche relâché ». OSX quant à lui remplace l'événement curseur déplacé par un événement spécifique « clic gauche glissé ».

Un problème est apparu lors de l'implémentation du double-clic. En effet sous windows il a suffit d'envoyer au système une séquence de la forme clic gauche enfoncé, relâché, enfoncé, relâché. Sur OSX nous avons tout d'abord essayé d'utiliser

la même séquence sans avoir de résultats. Il n'est pas mentionné dans la documentation Apple d'événements spécifiques au double-clique, nous avons donc cherché dans des documentations plus anciennes. Il se trouve qu'en utilisant l'ancien appel d'événements systèmes d'Apple hérité de NextStep nous avons réussi à générer des double-clics effectifs. Cette partie est donc fonctionnelle mais nous attendons une réponse de la part d'Apple pour connaître un moyen plus « propre » de gérer ces événements sans passer par des appels de méthodes censées être non usitées actuellement.

8. Anecdotes

Parmi les problèmes rencontrés, on peut citer par exemple les inclusions conflictuelles :
on a fusionné des bouts de code venant de PC sur mac et les fonctions redéfinissaient de façon conflictuelle les fonctions système de mac, et inversement. Ou encore, de par la complexité de l'utilisation de frameworks, l'intégration ne fut pas évidente car certains frameworks comme blitz++ et l'ApplicationServices de mac entraient en conflit.
On a également perdu du temps, en développement car on a pour la plupart d'entre nous dû nous renseigner en profondeur sur l'utilisation des IDE utilisées (Xcode sous mac et CodeBlocks sous PC). Et on a pu enfin, se rendre compte que le fonctionnement d'un outil sur deux plateformes n'est pas identique : par exemple, la blague du retournement d'image, sans raison apparente suivant que l'on passe de PC à mac.

9. Conclusion

Pour conclure, nous pouvons dire que nous avons été très heureux de pouvoir travailler sur un tel projet, et que nous le sommes d'autant plus, puisque nous avons réussi à répondre à notre problématique de départ, même si notre deuxième objectif d'implémenter le contrôle en mode absolu, n'a pas été atteint.
Du fait du croisement de compétences requises, ce projet nous aura demandé des efforts en conception logicielle, en parallélisme, en algorithmique de haut-niveau (algorithme génétique), en imagerie numérique ou encore en conception IHM.
Ainsi c'est dans tous ces domaines que nous avons progressé.
Enfin, par la mise en place d'un site internet http://glovecontrol.free.fr nous pensons dans un avenir proche diffuser notre travail pour envisager un développement collaboratif.
En effet, nous avons entrepris ce projet, dans le but aussi de pouvoir continuer son développement même après avoir rendu ce présent rapport et passé notre soutenance.
C'est pourquoi, nous pouvons dire, que ce que nous avons ici fait, n'est que le début de l'histoire de cette IHM.

Évolutions déjà pressenties

- nous allons rapidement mettre en place au cours de l'été, une page en anglais, pour faciliter la lecture et la compréhension de notre travail,
- nous pensons d'ici un mois, pouvoir implémenter la fonction souris en mode de fonctionnement absolu, et laisser le choix à l'utilisateur de choisir entre le mode relatif et absolu.

Idées possibles d'intégration

- nous avons été contactés par une entreprise de Genève intéressée par ce type de travail pour une application éventuelle dans le domaine du contrôle d'applications musicales ;
- Kevin a une idée d'évolution pouvant servir à l'apprentissage de la langue de signes sur ordinateur ;
- Thomas pense à une évolution de l'IHM pour son intégration couplant projection sur mur et webcam cachée, en transparence totale avec le lieu dans lequel on se trouve, ainsi qu'un affichage d'un HUD circulaire en temps réel autour du curseur déplacé pour plus de visibilité ;
- Quentin souhaite améliorer l'algorithme génétique et la phase de calibrage afin que les diodes soient identifiées parfaitement. Un système multi-seuil qui s'ajuste dès l'apparition de défauts apparaît comme une très bonne solution.
- Fred veut se servir du fait que le gant soit ludique pour permettre aux enfants une première approche des ordinateurs. Des animations de cliquer déplacer seront sûrement plus abordable avec ce gant. Il suffira à l'enfant de se servir de l'élément voulu et de le relâcher instinctivement.

Bibliographie/bookmarks

1- projets intéressants sur les IHM

Pilotage de la souris sur mac depuis la manette/nunchuk de la Wii :
http://www.wiili.org/index.php/DarwiinRemote
Pilotage de la souris avec les mouvements de la tête face à une webcam :
http://rjcooper.com/smartnav/index.html
http://www.orin.com/access/headmouse/index.htm
http://www.boosttechnology.com/tracer_description.html
http://live.gnome.org/CamTrack#head-2ccca2252515f6a4603b499d63aad2631cb5b0b8
Pilotage de la souris avec le clavier :
http://www.recitadaptscol.qc.ca/spip.php?article255
Pilotage de la souris avec la main :
http://www.viki-project.org/software.php
Pilotage original de la souris en vidéo :
http://www.youtube.com/watch?v=0awjPUkBXOU

2 – Programmation cocoa

http://en.wikibooks.org/wiki/Programming_Mac_OS_X_with_Cocoa_for_beginners/Implementing_Wikidraw
http://www.cocoadev.com/
http://www.stepwise.com/
http://www.mac4ever.com/articles/creation/334/les_bases_de_l_objectivec/
http://mattgemmell.com/
Documentation générale d'Apple:
http://developer.apple.com/samplecode/Cocoa/index.html
Documentation spécifique aux NSEvent :
http://www.cocoadev.com/index.pl?NSEvent

3 – Programmation des événements systèmes d'Apple

Exemple de code pour modifier l'accélération de la souris :
http://developer.apple.com/samplecode/SetMouseAcclSample/index.html
Événements cocoa :
http://developer.apple.com/samplecode/Cocoa/idxEventsOtherInput-date.html
Documentation sur les matériels et les interfaces humaines :
http://developer.apple.com/samplecode/HardwareDrivers/idxHumanInterfaceDeviceForceFeedback-date.html
La liste des méthodes utilisées pour créer et poster les événements Apple :
http://developer.apple.com/documentation/Carbon/Reference/QuartzEventServicesRef/Reference/reference.html

Le site d'un programmeur mac manipulant les événements systèmes d'Apple :
http://www.salling.com/

4 – Programmation des événements systèmes de Microsoft

http://tcharles.developpez.com/simul/

5 – Documentation sur les souris

Article sur la vitesse du curseur et l'accélération de celui-ci :
http://www.codinghorror.com/blog/archives/000977.html

6 – Documentation sur les librairies

OpenCV :
http://www.intel.com/technology/computing/opencv/
Forum http://www.developpez.net/forums/forumdisplay.php?f=739
Qt :
http://trolltech.com/products/qt/
Blitz++ :
http://www.oonumerics.org/blitz/

7 – Historique

[Al.Cooper'95] : Alan Cooper - The Myth of Metaphor – Juin 1995 – originellement publié dans le journal du programmeur Visual Basic.

8 – Algorithme génétique

http://fr.wikipedia.org/wiki/Algorithme_g%C3%A9n%C3%A9tique
http://www.rennard.org/alife/french/gavintr.html
http://khayyam.developpez.com/articles/algo/genetic/