Rapport de projet : Tableau virtuel interactif

Baptiste Saleil Geoffrey Mélia Julien Pagès Kevin Bollini

12 mai 2012

Table des matières

Table des figures

Remerciements : Nous tenons à remercier tout particulièrement M. William Puech (enseignant chercheur au LIRMM à l'université Montpellier 2, responsable de la formation du Master informatique IMAGINA) sans qui ce projet n'aurait pas pu se faire, pour avoir accepter de nous encadrer, pour son aide et son implication.
Nous souhaitons aussi remercier M. Benoit Lange (doctorant au LIRMM) pour ses conseils et sa présence au cours de ce projet.

Première partie

Rapport de projet

Chapitre 1

Introduction

1.1 Présentation

Les cinq dernières années ont marqué un grand renouveau dans les interfaces entre hommes et terminaux (écrans tactiles, contrôles vocaux, détection de mouvements). Cet état de faits nous a amenés à penser une application mettant en scène l'une de ces nouvelles façons d'interagir.

L'objectif du projet issu de cette réflexion est donc de créer un **tableau virtuel**, avec lequel une ou plusieurs personnes peuvent interagir en réalisant directement des gestes comme sur une toile réelle, par **reconnaissance des mouvements** detectés par une **webcam**.

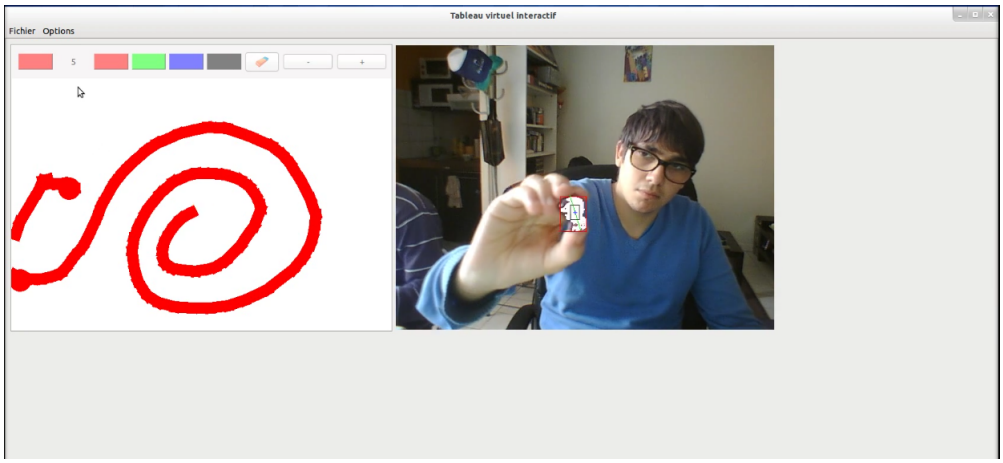

FIGURE 1.1 – Aperçu de l'application

1.2 Contexte

Ce projet est réalisé dans le cadre des TER (Travaux d'Étude et de Recherche) de 1ère année de Master informatique à L'Université Montpellier 2.
Le TER est un projet qui se déroule sur un semestre, généralement choisi par les étudiants parmi une liste de projets qui leurs sont proposés.
Ce genre de projet apporte beaucoup aux étudiants. Il leur permet d'une part de travailler une longue période sur un seul et même sujet afin d'obtenir quelque chose de complet, d'abouti.
D'autre part, le TER (comme son nom l'indique) apporte une bonne expérience dans le domaine de la recherche. En effet, un travail préalable doit être effectué sur l'existant (Est-ce que d'autres projets de ce genre existent ? Si oui qu'ont-ils de différent ?,...), mais également la faisabilité (Est-ce possible à faire ? Quels algorithmes existent pour répondre à tel ou tel besoin ?, ...).

La particularité de ce projet réside dans le fait que nous avons nous-même proposé ce sujet à un enseignant-chercheur.
Nous avons décidé de proposer ce sujet pour diverses raisons, dont voici les pricipales :

- Tout d'abord, c'est un projet que nous avions en tête depuis plus d'un an, qui a donc eu le temps de murîr et d'être au niveau d'un TER.
- Deuxièmement, notre groupe vient de deux formations distinctes : **AIGLE**, formation orientée génie Logiciel ; et **IMAGINA**, spécialisée dans le jeu vidéo et l'image. Ce projet représentait une manière opportune de lier ces deux spécialités
- Enfin, ce projet était pour nous une bonne occasion de réaliser une application fonctionnelle et distribuable, et par conséquent nous permettre d'explorer une autre facette du développement d'applications : la mise en production.

Chapitre 2

Analyse et conception

2.1 Étude de l'existant et faisabilité

Faisabilité

Notre groupe ayant lui-même proposé ce sujet, celui-ci s'accorde parfaitement à nos formations et spécialités. Le choix de ce sujet a donc été réalisé en fonction de nos expériences, savoir-faire et affinités.
Ce projet s'inscrivant dans le cadre d'un TER de notre formation, l'étude de la faisabilité n'incluera pas certains critères comme l'étude de marché, le contexte économique ou le besoin réel. Nous nous sommes concentrés notamment sur les compétences techniques et de gestion. Pour les parties financement et analyse des coûts, il va sans dire que nous ne disposions d'aucun financement et n'avons donc utilisé que des outils gratuits.

Existant

La vision par ordinateur et particulièrement le suivi d'objets, sont des domaines connus et pour lesquels il existe de nombreux travaux. Nous pouvons nous inspirer de certains de ces travaux afin de proposer des fonctionnalités plus pertinentes, éviter certains écueuils ou plus directement utiliser des outils existants comme la bibliothèque OpenCV.

- Exemples dans le jeux vidéo : Kinect, Eye-toy, CamSpace (techniques pour l'IHM, par exemple, mouvements continus ou immobilité sur une zone).
- Exemples issus de thèses et de projets de recherche (techniques de programmations).

FIGURE 2.1 – Exemple du Kinect

2.2 Gestion du projet

2.2.1 Choix stratégiques

Lors de ce projet, nous avons distingué deux parties majeures. La première est une bibliothèque de suivi d'objets. La seconde, une application qui utilise cette bibliothèque pour réaliser un logiciel de dessin interactif.
Ce choix présente plusieurs intérêts :

- Premièrement, ce découpage permet de bien différencier les tâches. Le développement de la bibliothèque requiert des connaissances en traitement d'images pour tracer des couleurs/objets, appliquer des filtres, manipuler les images, etc. Ceci correspond parfaitement à la formation de deux membres du groupe(formation IMAGINA à l'UM2). Le développement de l'application requiert quant à lui des connaissances en génie logiciel, pour l'architecture de l'application, en IHM, pour la réalisation de l'interface (logicielle et gestuelle), ou encore en réseau pour la mise en place de l'architecture client/serveur. Ces différents points correspondent également parfaitement à la formation des deux autres membres du groupe. (Formation AIGLE à l'UM2). Ainsi, les tâches peuvent bien se répartir en fonction des connaissances et spécialités de chacun.

- Deuxièmement, l'intérêt de ce découpage est d'avoir deux projets à la fois complètement indépendants et complémentaires. En effet, la bibliothèque développée offre des fonctionnalités de suivi utilisables par n'importe quelle application. L'application se charge d'utiliser les fonctionnalités de traitement d'images de la bibliothèque en ajoutant une interface graphique, une couche réseau, une connexion aux webcams ou encore la récupération des images. La bibliothèque et l'application sont donc deux projets développés en parallèle et très modulaires. Notre application peut très bien utiliser une autre bibliothèque de traitement d'images, et la bibliothèque peut très bien être utilisée par d'autres applications qui offriraient des fonctionnalités totalement différentes de la nôtre.

2.2.2 Diagramme de Gantt

Étant donné que nous avons proposé notre propre sujet, la première étape a été de rédiger un cahier des charges pour définir ce que nous voulions faire et de quelle manière.
Après ceci, nous avons commencé à effectuer des recherches du côté bibliothèque et application, pour savoir quels outils et techniques utiliser pour arriver à nos objectifs. Nous avons réservé plusieurs semaines pour effectuer des recherches et perfectionner notre idée du projet. Nous avons donc lu des articles de recherches sur des thématiques proches, effectuer des essais d'outils etc. Cette période d'analyse et de conception sert aussi à penser l'application en imaginant des cas d'utilisations (use cases) et en réalisant le diagramme de classes.

Par la suite, la majeure partie du temps est réservée au développement en lui même, en prenant en compte la date de présentation du projet pour avoir une réalisation complète.
Bien entendu ces temps de développement s'accompagnent de réunions avec tout le groupe de projet ainsi que de travail pour lier nos deux sous-parties et les tester ensemble.
Des documents comme le rapport ou la documentation furent produit au fur et à mesure du projet, pour rester en adéquation avec le travail accompli. Voici le rétroplanning qui illustre le déroulement chronologique du projet :

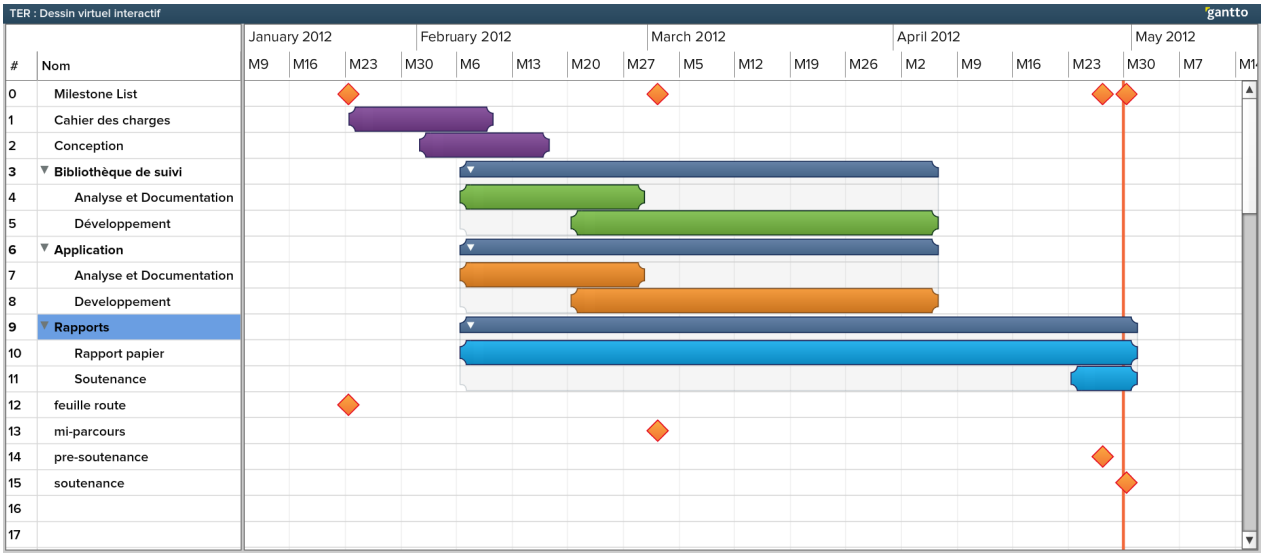

FIGURE 2.2 – Rétroplanning

2.3 Outils utilisés

Durant la période de recherche et au cours du projet nous avons utilisé différent outils (au sens large) pour nous aider dans la réalisation. Nous les détaillons ici dans cette section.

Outils collaboratifs

 Subversion : Un gestionnaire de versions qui nous permettant de synchroniser notre travail, de partager le code source et de s'échanger quelques documents complexes (ce rapport par exemple).

 Gobby : Logiciel libre pour l'édition collaborative, qui permet de développer à distance sur le même document simultanément.

Bien sûr ces outils ne sont pas suffisants, nous avons donc fait des réunions régulières pour organiser le travail et décider des orientations. Par ailleurs, nous avons beaucoup échangé pour que tout le monde soit toujours au courant des différentes informations importantes.

Outils techniques

Langages :

Langage C pour la bibliothèque : ce langage est bien adapté à la réalisation d'une bibliothèque car il permet l'écriture de fonctions. Par ailleurs c'est un langage bien adapté au domaine du traitement de l'image car il est de bas niveau, et très rapide. De plus, nous avons utilisé des bibliothèques (OpenCV par exemple) qui ont été écrites initialement en C.

Langage C++ pour l'application : c'est un langage objet ce qui nous aidait pour la conception. Il est également très rapide et c'est un langage avec lequel nous sommes à l'aise et avons de l'expérience, ce qui nous évitait de devoir apprendre un nouveau langange, et par là de perdre un temps précieux.

Bibliothèques :

 Qt : bibliothèque d'IHM (et bien plus) en C++, nous aide pour faire les interfaces graphiques, pour la partie réseau également où elle offre des abstractions agréables.

 : bibliothèque de traitement de l'image, nous aide à exploiter la webcam et pour le traitement de l'image avec plusieurs fonctionnalités.

2.4 Analyse

2.4.1 Cas d'utilisations

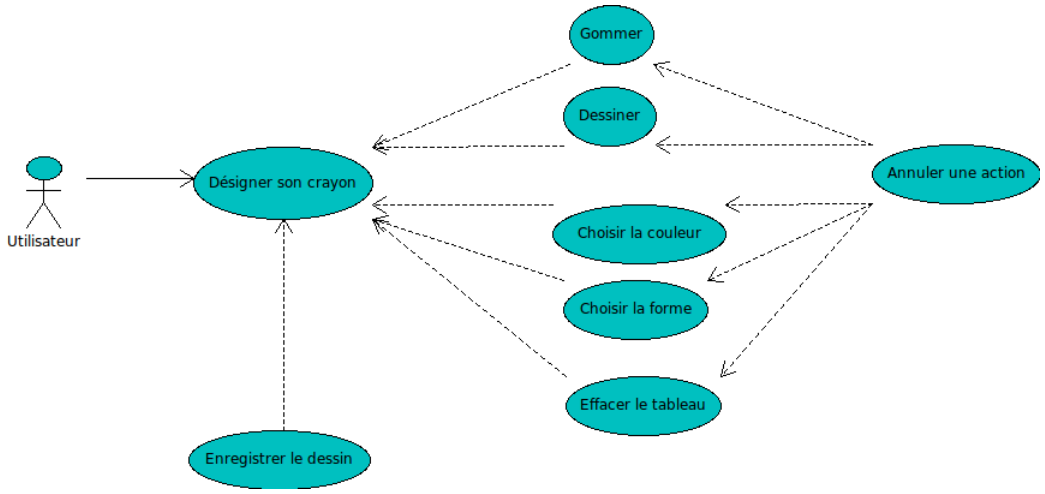

FIGURE 2.3 – Cas d'utilisation dessin

2.4.2 Diagrammes de séquences

Les diagrammes qui suivent expliquent le fonctionnement de l'application pour une utilisation locale, et une utilisation en réseau. On peut voir sur ce diagramme que le fonctionnement en local est assez basique. Dans un premier temps, l'utilisateur

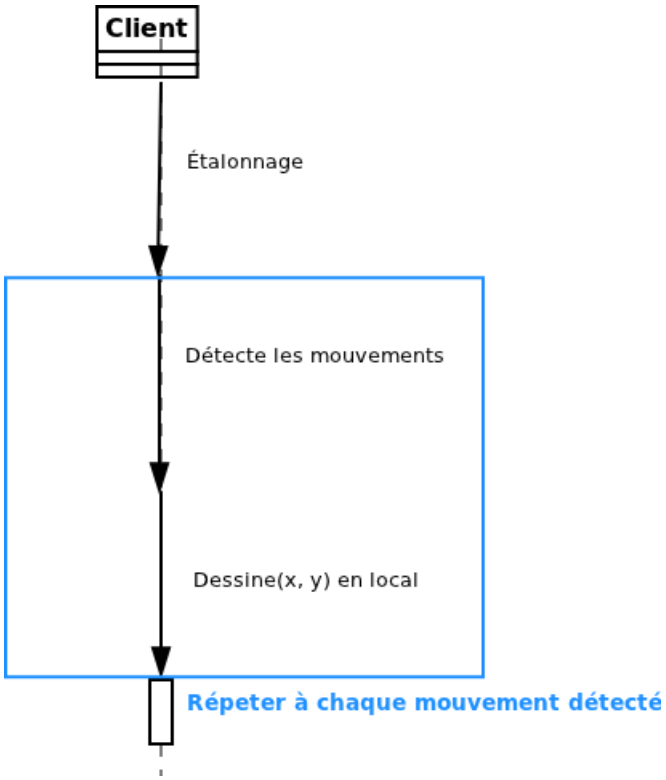

FIGURE 2.4 – Diagramme de séquence local

étalonne l'objet avec lequel il veut dessiner. Ensuite, lorsque l'utilisateur déplace l'objet en question, l'application **détecte** ce déplacement, et **dessine** à la nouvelle position de l'objet. Ce schéma de détection-dessin est répété jusqu'à fermeture de l'application.
Ainsi, l'utilisateur verra se dessiner chacun des mouvements qu'il effectue avec son objet.

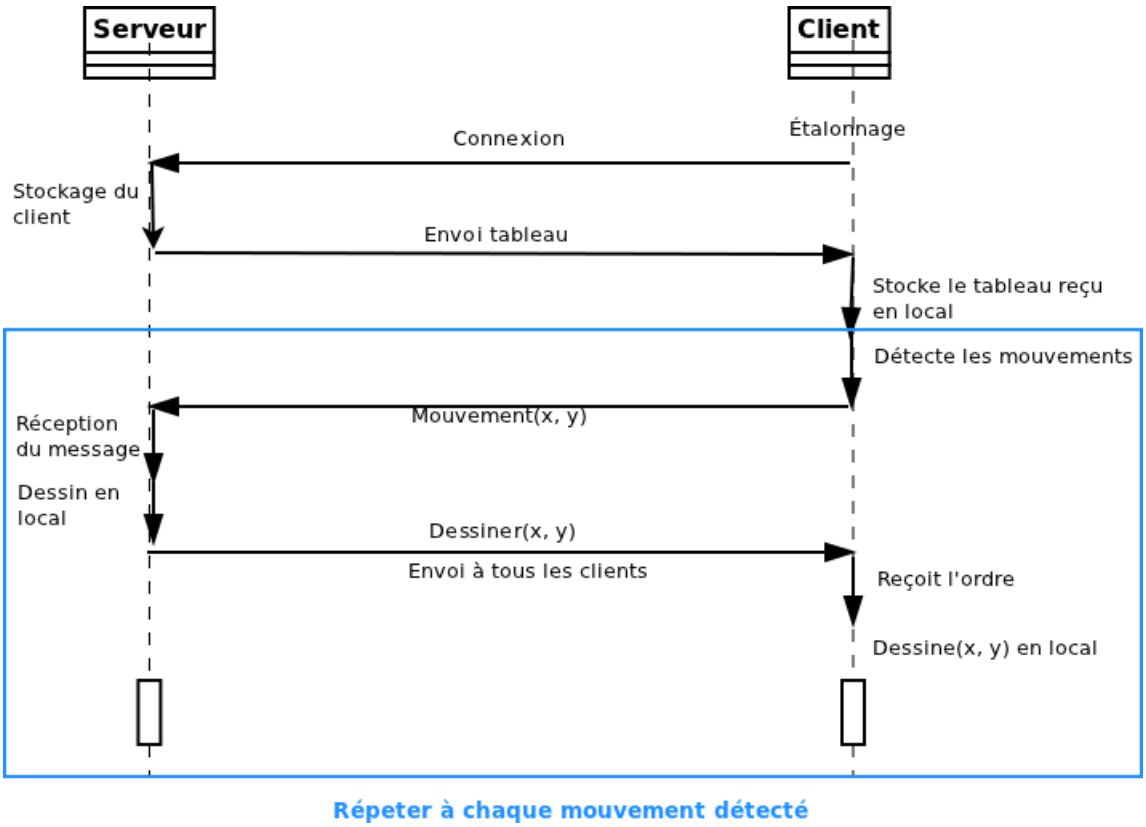

FIGURE 2.5 – Diagramme de séquence réseau

En réseau, l'utilisation suit le même schéma, mais un autre acteur entre en jeu, le **serveur**.
Comme pour une utilisation locale, l'utilisateur comment par **étalonner** l'objet à suivre. Puis, la connexion s'effectue auprès du serveur, qui va renvoyer le dessin courant au nouveau client. Une fois en possession du dessin, le client le stocke et peut rentrer dans la boucle de détection-dessin :
Comme en local, l'application va **détecter** le mouvement de l'objet suivi. La nouvelle position est envoyée au serveur qui va dessiner sur son propre tableau, puis renvoyer à tous les clients l'ordre de **dessiner** à cette même position. Chaque client verra donc ce que les autres clients dessinent.
Ainsi, l'utilisateur verra se dessiner chacun des mouvements qu'il effectue avec son objet, mais également ceux de tous les autres clients.

On peut donc voir que pour l'utilisateur, le fonctionnement est identique qu'il dessine seul ou avec d'autres personnes :
- Il étalonne l'objet
- Il déplace l'objet
- l'objet dessine sur le tableau

Cependant, dans la pratique, le fonctionnement n'est pas le même. En réseau, chaque mouvement passe par le serveur qui se charge de faire le lien avec tous les clients de l'application.

2.4.3 Diagrammes de classes

La bibliothèque fonctionne grâce à une structure de données utilisée par deux fonctions enveloppes.
Le curseur est représenté en mémoire sous la forme d'une strucure de données "Cursor".
Cette structure de données est engendrée par la fonction "calibration" de la bibliothèque. Elle est ensuite utilisée par la fonction "track", qui permet la mise à jour des informations relatives au curseur par rapport à une image.

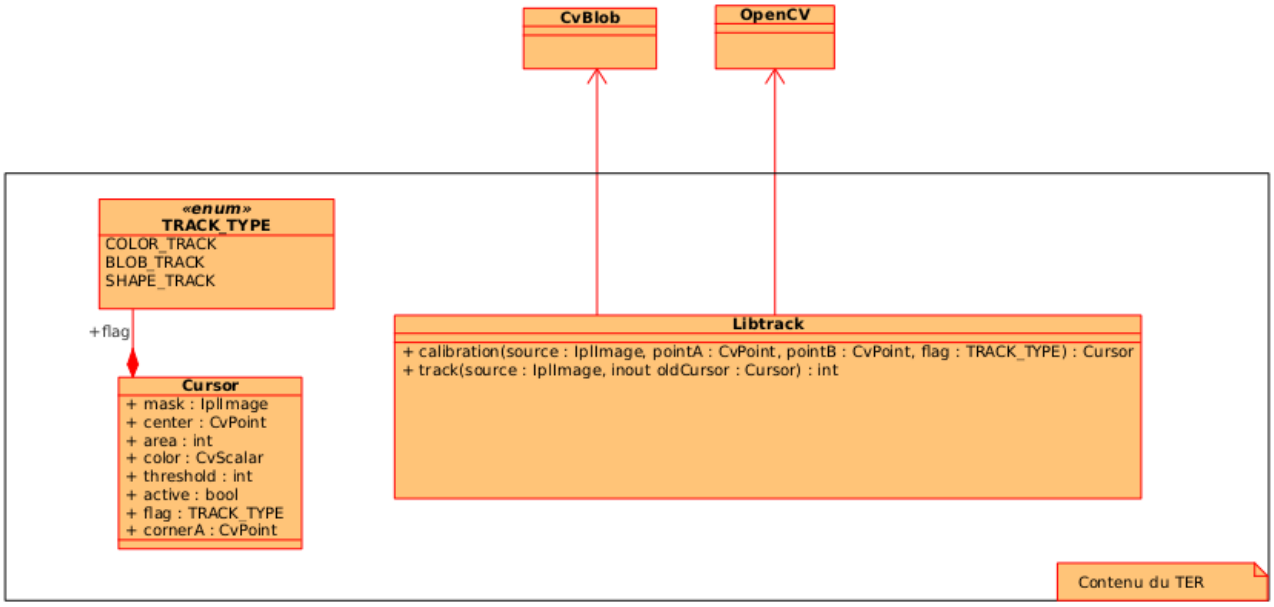

FIGURE 2.6 – Architrecture de la bibliothèque

La bibliothèque dépend de deux bibliothèques externes :
– OpenCV, Spécialisée dans le traitement des images
– CvBlob, bibliothèque issue d'OpenCV utlisée pour la recherche d'objet dans des images binaires

Le diagramme qui suit vous présente l'architecture de notre application.
Avant de commencer l'application, nous avons décidé de passer un long moment sur la réflexion pour avoir une architecture solide et modulable.
Nous avons dans un premier temps identifié les différents modules dont serait composée notre application et quel était précisément le ou les rôles de chacun.
Ensuite, nous avons réfléchi à quel était le meilleur moyen pour que ces différents modules puissent communiquer en évitant toute redondance ou communication superflue.
Une fois cette réflexion faite, nous avons dessiné le diagramme suivant :

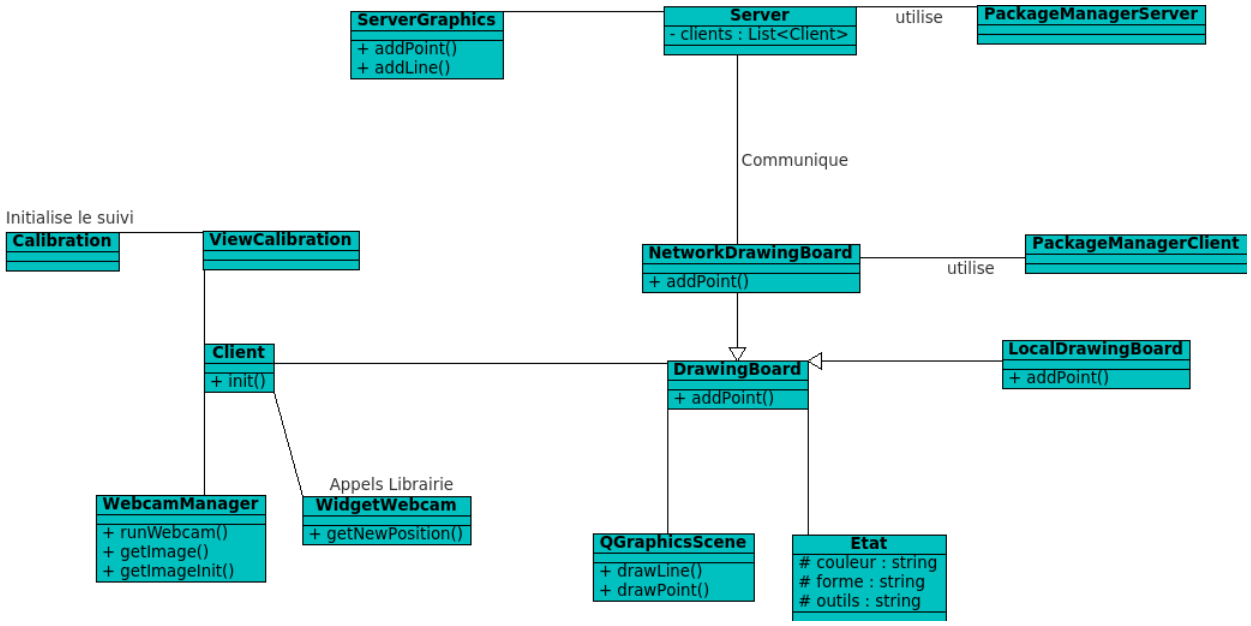

FIGURE 2.7 – Diagramme de classe de l'application

On peut identifier quatre modules principaux :

- **Le module d'étalonnage** (Calibration et ViewCalibration). Ce module sert à tout ce qui touche à l'étalonnage de l'objet. Une fois qu'il est effectué, le module va appeler le module *client* pour lui signaler que l'étalonnage est fait, et lui envoyer les informations qui correspondent (objet à suivre, type de suivi, dessin en local ou réseau, etc...)
- **Le module client** (Client). Ce module est le coeur de l'application, il a principalement deux rôles. Il va dans un premier temps construire toute l'interface graphique (tout ce qui est visuel) qui sera présentée à l'utilisateur. Ensuite, c'est ce module qui va assurer la communication entre tous les modules. Il est le seul intermédiaire entre les différents modules locaux.
- **Le module de dessin** (DrawingBoard, LocalDrawingBoard et NetworkDrawingBoard). Ce module a également deux rôles. Il va s'occuper de toute la partie dessin (Stocke la couleur et taille du pinceau, dessine les points, lignes, etc...). Aussi, pour une utilisation en réseau, c'est lui qui va se charger de communiquer avec l'application serveur.
- **Le module serveur** (Server, ServerGraphics). C'est ce module qui met en relation les différents client, et qui synchronise le dessin. Il reçoit les ordres de dessin, les stocke, et les renvoie à tous les autres clients.

La classe **WebcamManager** est la seule classe qui va communiquer avec le **matériel** (ici les webcams). Lorsque l'on veut une nouvelle image de la webcam, le module client va demander à WebcamManager la nouvelle image, et l'envoyer au module qui en a besoin.
La classe **WidgetWebcam**, elle, est la seule classe qui va communiquer avec la **bibliothèque de suivi**.
Les classes QGraphicsScene et Etat représentent respectivement le dessin et l'état actuel du pinceau.
Les classes PackageManager* servent à assembler et désassembler les paquets qui transitent sur le réseau.

Scénario :
Afin d'illustrer la modularité de notre application, on pourrait imaginer le scénario suivant :

- L'étalonnage s'effectue. Une fois terminé, le module d'étalonnage va prévenir le client, et lui donner les informations.
- Chaque X millisecondes, le client va interroger WebcamManager pour obtenir une nouvelle image.
- Le client va envoyer cette image à WidgetWebcam pour obtenir la nouvelle position. (Grâce à la bibliothèque de suivi)
- Le client va maintenant envoyer cette position à DrawingBoard qui va dessiner le nouveau point en fonction de la couleur la taille, etc...

On voit ici que chaque module rempli bien son rôle et assure le bon fonctionnement général de l'application.

Une fois cette phase de réflexion terminée, toute l'architecture de la bibliothèque, et de l'application était pensée et écrite. Nous pouvions alors débuter la phase de réalisation, programmer et imbriquer les différents modules, et s'attaquer aux problèmes plus techniques.

Chapitre 3

Réalisation

3.1 Bibliothèque de suivi

Cette partie du développement a pour objectif de fournir un ensemble de fonctionnalités permettant la **reconnaissance** et le suivi d'un objet, que nous appellerons **Curseur**, à partir d'images.
L'objectif est de développer notre propre **bibliothèque** pour d'une part avoir un outil remplissant précisément nos objectifs et d'autre part de faire un travail de recherche et développement dans le domaine du traitement de l'image (en particulier le suivi d'objets).
Un certain nombre de bibliothèques existantes proposent des outils de reconnaissance de formes, de suivis de modèle ou d'évaluation de composantes connexes. Le travail consiste donc à tirer partie de ces bibliothèques en les associant de manière à proposer des méthodes efficaces et simples à utiliser. Cela implique une bonne connaissance des outils existants, de leur compréhension et de leur utilisation conjointe dans le but de concevoir notre propre bibliothèque.

3.1.1 Aspect général

La structure générale de la bibliothèque est relativement simple à appréhender, ce qui permet une utilisation aisée sans connaissances particulières en traitement d'images.
Comme expliqué dans la partie d'analyse, le fonctionnement de la bibliothèque de suivi repose sur l'utilisation d'une structure de données créée et manipulée respectivement par les deux fonctions d'étalonnage et de suivi. L'affinage de la recherche et le choix par l'utilisateur du type de suivi à réaliser passe par la manipulation d'une énumération, proposée par la bibliothèque. Aujourd'hui composée de 3 éléments correspondant chacun à un type de suivi particulier, cette énumération est amenée à s'agrandir avec l'ajout de nouvelles méthodes.

Étalonnage

Afin d'effectuer le suivi d'un objet (au sens large du terme) grâce à notre bibliothèque, il convient en premier lieu de faire appel à la fonction "calibration". Réalisant l'étalonnage, elle permet de créer et d'initialiser la struture de suivi, appelée "Cursor", contenant l'intégralité des informations qui vont permettre d'identifier l'objet et de réaliser sa localisation dans les images futures. C'est lors de cette étape que l'utilisateur doit préciser le type de suivi soubaité, par l'intermédiaire d'un simple paramètre.

Fonctionnement

Du point de vue de l'utilisateur, une seule fonction permet de gérer la totalité de l'étalonnage, ce qui permet de simplifier l'utilisation de notre bibliothèque :

*Cursor * calibration(IplImage * source, CvPoint A, CvPoint B, TYPE_TRACK flag) ;*

Cette fonction, créant et initialisant la structure de suivi, n'est en réalité qu'une enveloppe faisant appel à une fonction différente selon le flag envoyé. Ces différences permettent de préparer la structure aux spécifités des fonctions de track et d'optimiser le suivi.

Il suffit donc de lui passer l'image où se trouve l'objet, 2 points pour définir ses contours et enfin un flag déterminant la méthode à employer, parmi les objet de l'énumération suivante :

enum TYPE_TRACK {TRACK_COLOR, TRACK_SHAPE, TRACK_BLOB} ;

Suivi d'un objet

Une fois ces informations enregistrées, il est alors possible de faire appel autant de fois que nécessaire à la fonction "track" de suivi, afin mettre à jour la structure de données avec les modifications inhérantes au changement d'image.

Fonctionnement

3.1.2 Détails techniques

Fonctions d'étalonnage

colorAverage

Détails sur la fonction de colorAverage.

mainColor

Détails sur la fonction de recherche de couleur principale.

Fonctions de suivi

Binarisation

Détails sur la fonction de binarisation.

blobFounding

Détails sur la fonction de blobFounding.

setNewCoord

Détails sur la fonction de setNewCoord.

shapeTrack

Détails sur la fonction de reconnaissance par modèle.

3.2 Application

Application :
L'application doit exploiter le plus efficacement possible la bibliothèque de reconnaissance. Nous voulons donc mettre en place un étalonnage de l'objet à suivre le plus précisément possible. Le projet doit ensuite être exploitable et utile, nous devons donc mettre en place un certain nombre d'outils tels que la possibilité de gommer, changer de forme, de couleur etc. Un module réseau est aussi prévu pour une utilisation concurrente et synchronisée. Et bien sûr la fonctionnalité clé de l'application est le tableau virtuel, qui devra donc être exploitable et le dessin devra être fidèle aux mouvements reconnus.

Chapitre 4

Résultats

Dans cette section nous allons présenter le résultat tangible de ce projet, en mettant en lumière l'utilisation de la bibiothèque de suivi au travers de l'application de dessin.

4.1 Client

Comme dit précedemment, l'application est composée de deux grandes parties, la **calibration** et le **dessin**.

4.1.1 Calibration

La calibration est la première étape dans l'utilisation du logiciel, cette section est étroitement liée à la fonction **calibration** de la bibliothèque. Cette section est composée de quatre interfaces, ou étapes, permettant l'**initialisation** d'un suivi d'objet. Chacune de ces étapes aura pour but d'obtenir les paramètres de la fonction calibration, mais aussi de permettre à l'utilisateur d'agir sur des attributs de réglage du suivi.

Le premier écran de l'application est l'interface de sélection de webcam et de méthode de suivi.

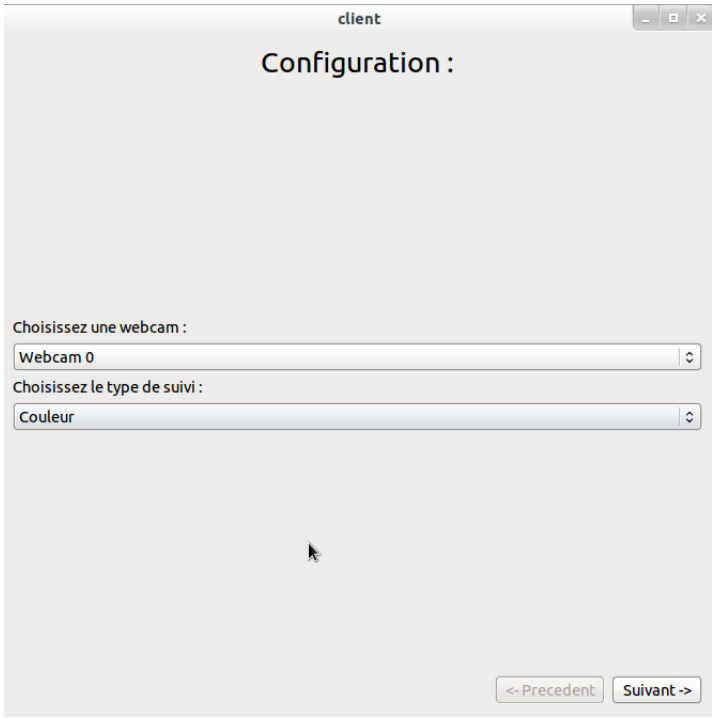

FIGURE 4.1 – Choix webcam/type track

La première liste déroulante permet le choix de la webcam à utiliser pour la capture, la seconde la sélection de la méthode de track à parmi utiliser :

- Suivi par modèle
- Suivi par couleur
- Suivi par couleur + composantes connexes

Au niveau de la bibliothèque de suivi, ces listes servent à définir la sources des images servant au track, et à choisir le drapeau pour la fonction calibration.

L'écran suivant permet quand à lui de sélectionner sur le retour image de la webcam l'objet à suivre, le curseur.

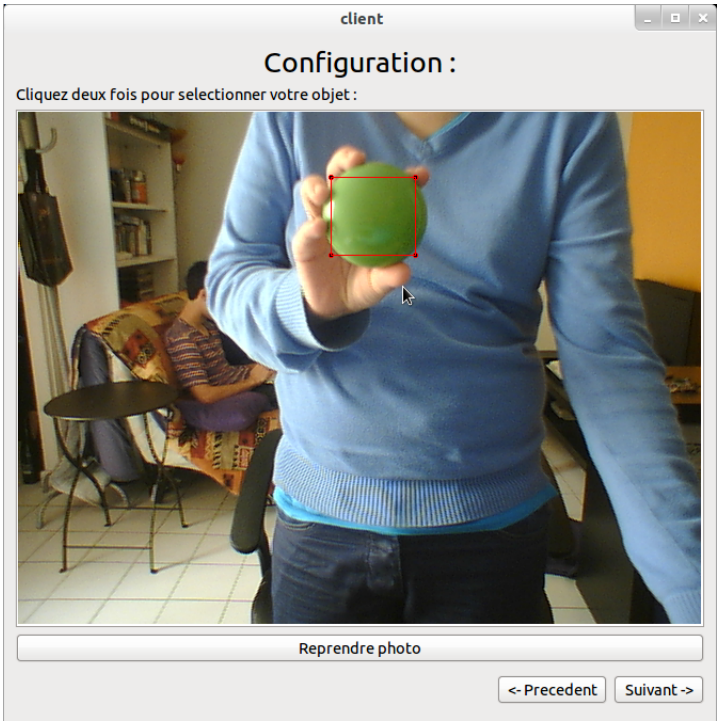

FIGURE 4.2 – Sélection du curseur

Il suffit pour cela, par clics sur l'image, de créer une boîte englobante autour du dit curseur.
Cet écran sert à récuperer les coordonnées de la hit box nécessaire à l'isolement de l'objet pour le suivi par comparaison de modèle ou à l'extraction de la couleur et de l'aire de l'objet pour les suivis par couleur.

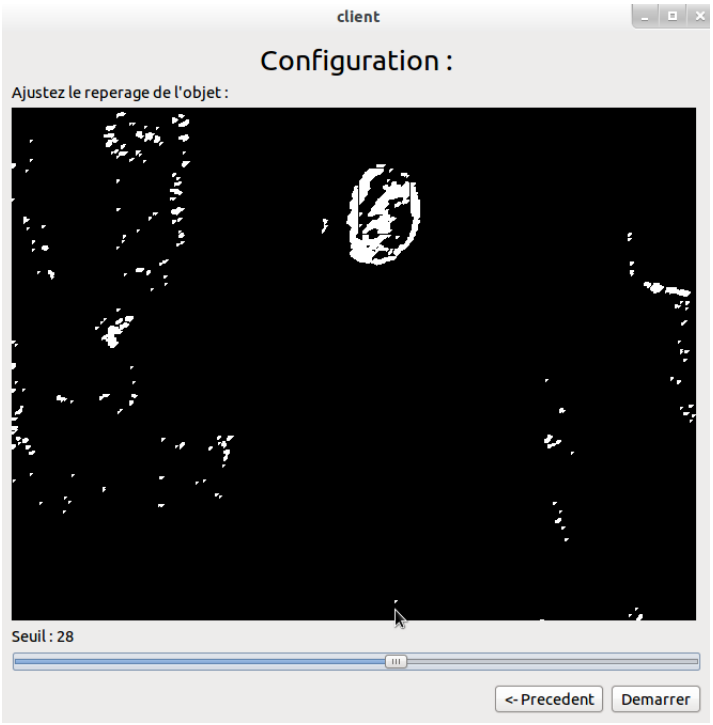

FIGURE 4.3 – Réglage du seuil

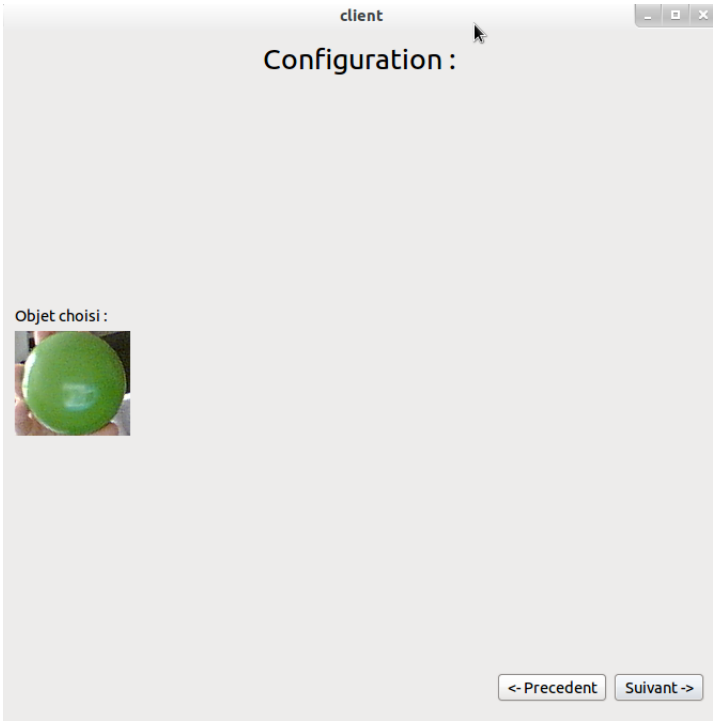

FIGURE 4.4 – Contrôle du modèle

FIGURE 4.5 – Sélecteur Local/Réseau

4.1.2 Dessin

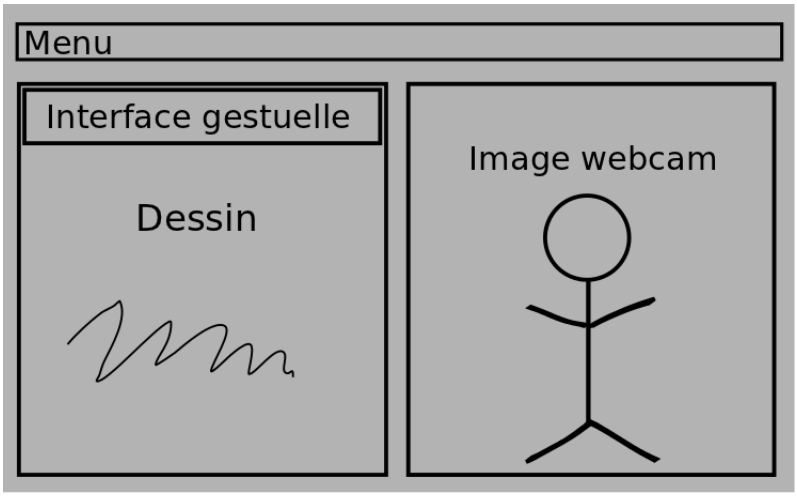

FIGURE 4.6 – Interface de dessin

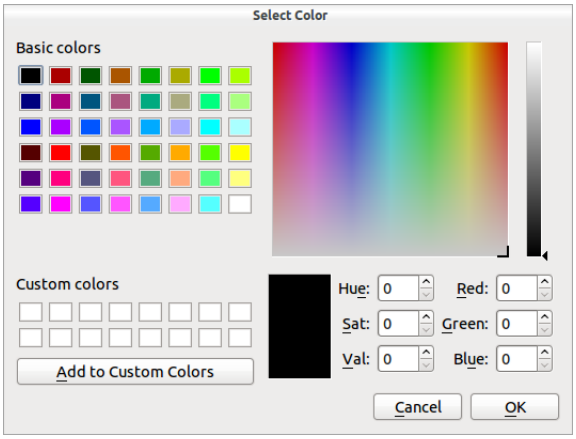

FIGURE 4.7 – Selecteur de couleur

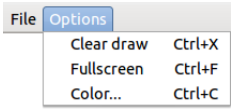

FIGURE 4.8 – Menu

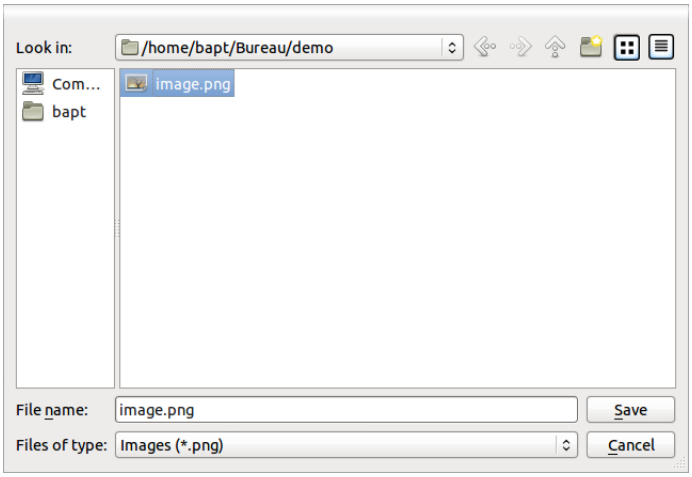

FIGURE 4.9 – Export

4.2 serveur

Chapitre 5

Conclusion

5.1 Difficultés rencontrées

5.2 Perspectives

5.3 Conclusion

Chapitre 6

Références

Deuxième partie

Annexes

Annexe A

Documentation de la bibliothèque