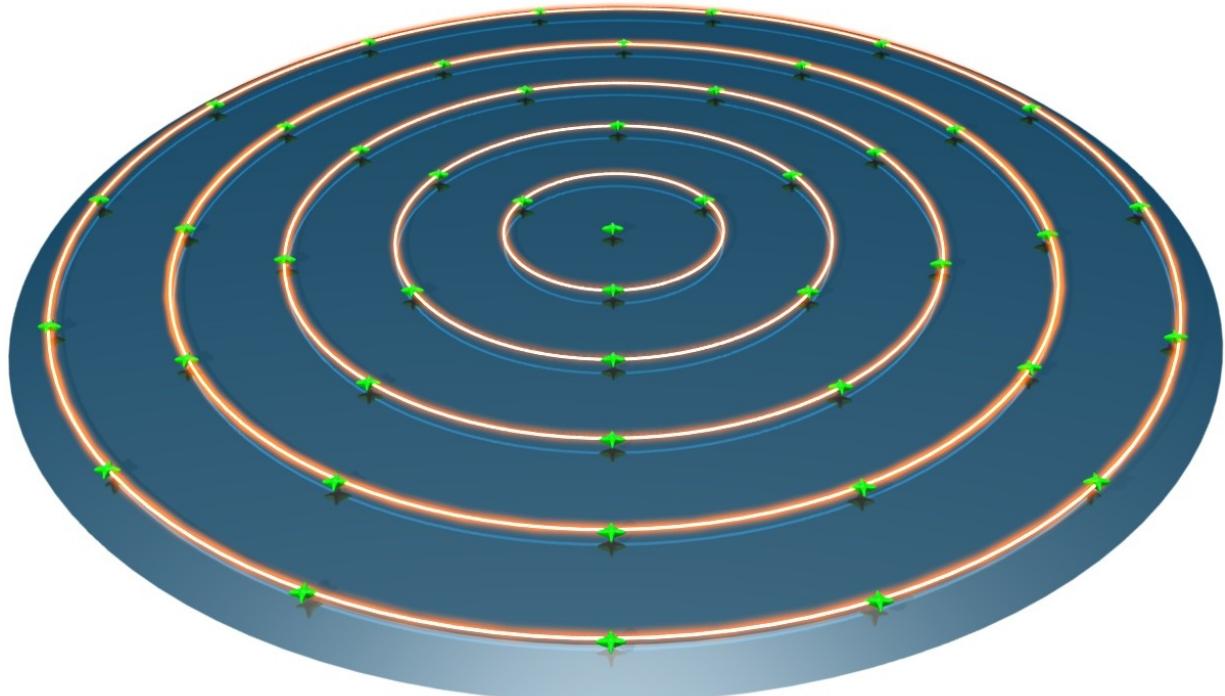


TX : InTACT



DEHORS
Vincent

FOUQUET
Yoann

MARTIN
Christophe

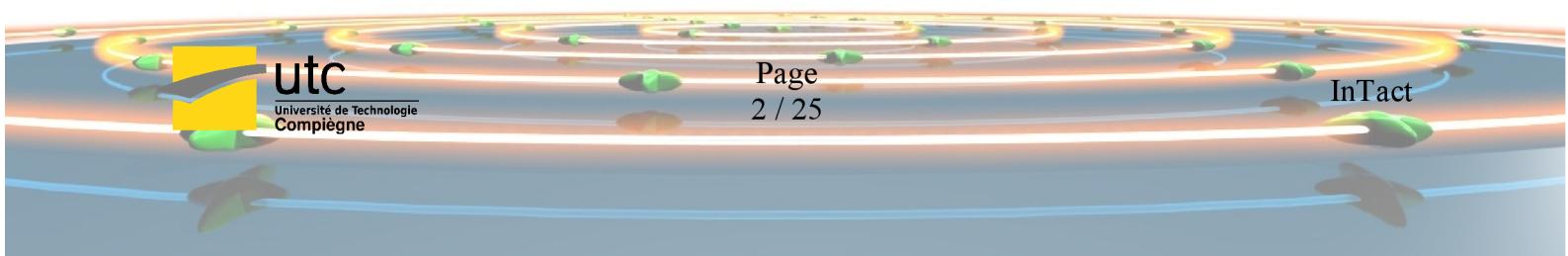
CHATEAU Fabien
FREMONT Vincent

TABLE DES MATIÈRES

| | |
|---|-----------|
| INTRODUCTION..... | 3 |
| I. Cahier des charges..... | 4 |
| 1) Objectifs..... | 4 |
| 2) Cadre pédagogique..... | 4 |
| 3) Public visé..... | 4 |
| 4) État actuel des développements..... | 4 |
| 5) Travail à réaliser..... | 6 |
| 6) Bénéfices attendus..... | 6 |
| 7) Contraintes à prendre en compte..... | 6 |
| 8) Matériel à disposition..... | 6 |
| II. Gestion de projet..... | 7 |
| 1) Organisation générale..... | 7 |
| 2) Répartition chronologique de projet..... | 7 |
| 3) Description des tâches..... | 8 |
| III. Compilation de CCV..... | 11 |
| 1) Qu'est ce que CCV ? | 11 |
| 2) Récupération de CCV..... | 11 |
| 3) Configuration de l'environnement Linux à partir d'une distribution Ubuntu..... | 12 |
| 4) Installation de UNICAP..... | 13 |
| 5) Configuration par défaut..... | 14 |
| IV. Module de calibrage..... | 15 |
| 1) Principe..... | 15 |
| 2) Algorithmes..... | 15 |
| 3) Calibrage de la table..... | 22 |
| 4) Évaluation des résultats..... | 23 |
| 5) Limite..... | 24 |
| CONCLUSION..... | 25 |

INDEX DES ILLUSTRATIONS

| | |
|--|-----------|
| Figure 1 : Diagramme de Gantt..... | 7 |
| Figure 2 : Interface graphique CCV..... | 11 |
| Figure 3 : Positionnement des points sur les cercles..... | 16 |
| Figure 4 : Zone de calibration..... | 17 |
| Figure 5 : Calcul des coordonnées d'un point (cas 1)..... | 17 |
| Figure 6 : Calcul des coordonnées d'un point (cas 2)..... | 18 |
| Figure 7: Inscription de la zone de calibration dans deux triangles..... | 20 |
| Figure 8: Ajout du premier point (erroné)..... | 20 |
| Figure 9: Ajout du premier point (corrigé)..... | 21 |
| Figure 10: Maillage de triangle..... | 21 |
| Figure 11: Test du calibrage actuel..... | 22 |
| Figure 12: Calibration du premier point..... | 22 |
| Figure 13 : Traduction du module de calibrage en anglais et en français..... | 23 |
| Figure 14: Distance entre les points et le centre des cercles (après calcul)..... | 23 |



INTRODUCTION

A l'instar de l'I-phone d'Apple ou de la table Surface de Microsoft, les interfaces tactiles sont de plus en plus visibles dans notre environnement. Le projet "InTact" vise à créer une table tactile ronde et des applications logicielles associées, pour la communication et le divertissement. Lancé en février 2010, le projet InTact est lauréat du concours national d'innovation d'OSEO et du ministère de la recherche et bénéficie du soutien de l'Incubateur Régional Picardie. La technologie Multitouch réunit une communauté de développeurs open-source très active qui développe et implémente des logiciels dédiés au « surface computing » (<http://ccv.nuigroup.com/>). Le logiciel CCV, né de cette communauté, est utilisé sur la table InTact pour le tracking vidéo par caméra infrarouge. Ce logiciel nécessite des ressources importantes en raison de son adaptabilité « multi-devices ». Cela implique un phénomène de latence nuisible pour la fluidité de l'écran tactile.

Dans le cadre de notre TX, nos objectifs étaient multiples. Fabien CHATEAU désirant commercialiser son produit avec Linux, nous devions faire fonctionner CCV sous ce dernier. Une fois cette étape réalisée nous avons cherché à rendre fonctionnelle l'optimisation des calculs par le GPU afin d'améliorer la fluidité du dispositif. Enfin, notre objectif principal était d'implémenter un module de calibrage circulaire adapté à la table tactile. Pour cela, il était aussi nécessaire de modifier l'algorithme se chargeant de corriger les positions détectées.

I. Cahier des charges

1) Objectifs

- Étendre la compatibilité : Rendre le code source de CCV compilable sous Linux (v 2.6)
- Optimiser les performances : Tirer partie de la puissance de calcul du processeur graphique
- Adapter CCV à la table : Implémenter un module de calibrage circulaire
- Adapter la correction : Corriger le positionnement des blobs en prenant en compte le nouveau calibrage
- Analyser le fonctionnement : comprendre comment CCV réalise la reconnaissance de formes

2) Cadre pédagogique

| Fonction | Nom | Prénom | Adresse mail |
|-------------------|---------|---------|----------------------------|
| Porteur du projet | CHATEAU | Fabien | fabien.chateau@utc.fr |
| Responsable | FREMONT | Vincent | vincent.fremont@hds.utc.fr |

3) Public visé

Le travail que nous réalisons ne concerne pas directement un public en particulier. Tout nos développements sont commandés par Fabien CHATEAU qui projète de vendre un produit totalement finalisé fonctionnant sous un OS Linux.

4) État actuel des développements

a) Système d'exploitation

La table tactile est aujourd'hui fonctionnelle sous Windows. Cependant, le système de calibrage n'est pas adapté à la forme circulaire du plateau.

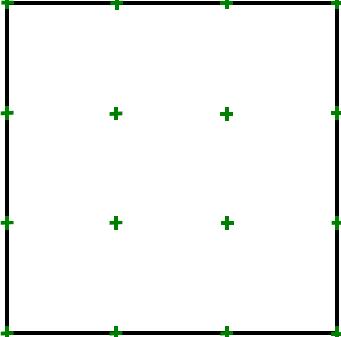
b) Utilisation du processeur graphique

Le GPU n'est pas utilisé.

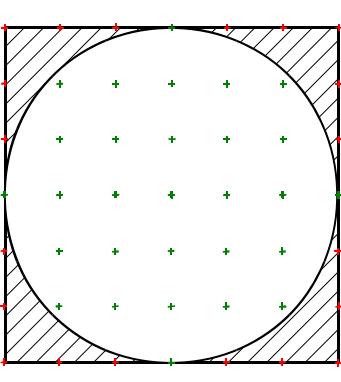
c) Système de calibrage

Le calibrage est une opération qui consiste à créer une correspondance entre les positions théoriques des points répartis sur la surface de la table et leurs positions réellement détectées.

CCV propose un module de calibrage adapté à une surface carrée :

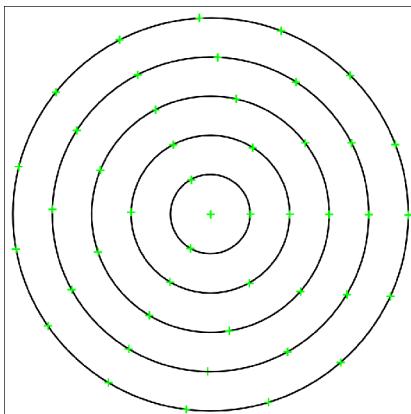
| | |
|---|--|
|  | <p>Une fois le module lancé, des croix vertes sont réparties sur toute la surface de la table.</p> <p>L'utilisateur peut régler le nombre de lignes et de colonnes formées par les croix.</p> <p>Vient alors l'opération de calibrage où l'utilisateur doit appuyer successivement sur chacune des croix.</p> <p>C'est à ce moment que les positions théoriques sont mises en correspondance avec les positions réelles.</p> |
|---|--|

La table tactile ronde est malgré cela fonctionnelle car il est possible de réaliser un calibrage en inscrivant le cercle formé par la table dans le carré de calibrage :

| | |
|---|---|
|  | <p>On constate ici qu'il est possible d'obtenir une définition relativement correcte à l'intérieur du cercle en ajoutant suffisamment de points de calibrages.</p> <p>Cependant, les coordonnées réelles des points situés dans les zones hachurées ne peuvent être données à CCV.</p> <p>Pour palier à ce problème, Fabien a conçu un dispositif qui permet de contourner le problème en donnant de fausses coordonnées pour les points en question. Cela engendre bien évidemment une erreur sur la corrections des points détectées sur la périphérie de la table.</p> |
|---|---|

5) Travail à réaliser

- Télécharger le code source de CCV sur le site officiel (<http://ccv.nuigroup.com/>)
- Installer les librairies unicap sous Linux
- Compiler CCV à l'aide de Codeblocks
- Adapter le module de calibrage de façon à répartir les points de manière circulaire



Pour cela nous aurons à réfléchir à une répartition des points la mieux adaptée pour limiter les erreurs

- Une fois que cette nouvelle répartition sera implémentée, il sera nécessaire d'adapter le système de correction qui était utilisé par CCV.

6) Bénéfices attendus

- Rendre la table tactile InTact compatible à des solutions Open-source telles que Linux permettra de réduire les coûts liés aux achats de licences lors de la commercialisation.
- Avant de commercialiser la table tactile InTact il est nécessaire d'implémenter toutes les fonctionnalités fondamentales :
 - Obtenir une fluidité satisfaisante en exploitant la puissance de calcul du processeur graphique
 - Proposer une interface de calibrage tout à fait adaptée à la forme circulaire de la table

7) Contraintes à prendre en compte

- Nous disposons d'un semestre pour réaliser cette TX.
- Nos développements devront également être compatibles à Windows.

8) Matériel à disposition

Fabien CHATEAU et Vincent FREMONT ont mis à notre disposition :

- Une caméra infrarouge PS3 eye
- Une salle au centre de recherche équipée d'un ordinateur possédant un dual-boot Windows & Ubuntu
- Un accès à la table tactile à partir du mardi 31 Mai.

II. Gestion de projet

1) Organisation générale

Nous avons réalisé ce projet à trois développeurs :

- DEHORS Vincent
- FOUQUET Yoann
- MARTIN Christophe

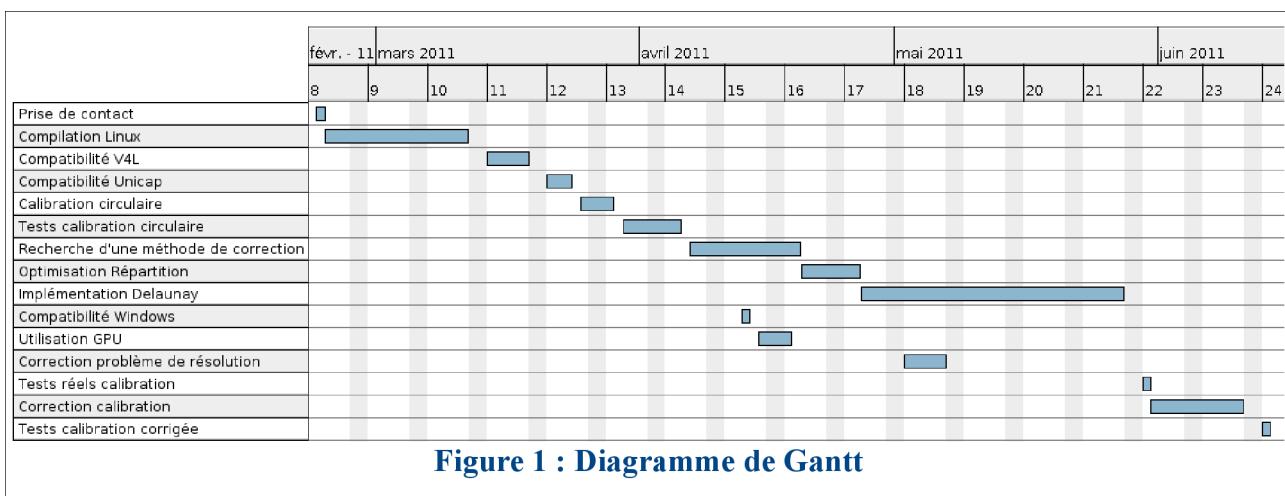
La répartition des tâches n'a pas posé de difficulté particulière puisque toutes nos tâches devaient être développées de manière séquentielle.

Afin d'être tous autonomes nous avons décidé d'installer l'environnement de travail nécessaire sur nos propres machines. Nous avons également convenu d'un rendez-vous hebdomadaire le Mardi à partir de 14h au centre de recherche. C'était le seul créneau où nous étions tous disponibles afin de faire un point chaque semaine.

En ce qui concerne les transferts de données, Fabien nous a demandé d'utiliser un compte Dropbox.

Enfin, nous nous sommes créé un espace brouillon sur Moodle disposant d'un forum et d'un module de gestion de projet. Cela nous a permis de garder une trace des principales décisions prises au cours du semestre.

2) Répartition chronologique de projet



3) Description des tâches

- Prise de contact

- Premier rendez-vous avec Fabien Chateau :
 - Description des objectifs
 - Présentation de la table tactile
 - Signature des clauses de confidentialité

- Compilation Linux

- Téléchargement des sources de CCV 1.3 pour Ubuntu et compilation.
- Principaux problèmes rencontrés : Dépendances Code blocks, Mauvaises librairies fournies (64bit # 32 bits), Problème avec la librairie common.h

- Compatibilité V4L

- Compilation de CCV en utilisant V4L. La Webcam n'étant pas reconnue sous Linux avec Unicap nous avons décidé d'utiliser V4L (Vidéo For Linux).
- Difficulté : Inclusions multiples lors de la compilation résolues en commentant l'inclusion du header de V4L dans CameraUtils.h. Ce header contient le code des fonctions de V4L. Les fonctions étaient définies deux fois malgré le #ifndef qui empêche les inclusions multiples.
- Problème : Défauts de synchronisation de la webcam

- Compatibilité Unicap

- Compilation de CCV en utilisant Unicap. Suite au défaut constaté avec l'utilisation de V4L nous avons décidé de faire fonctionner Unicap sous Linux.
- Difficulté : Installation de Unicap sous Linux.
 - Tuto : <http://kaswy.free.fr/?q=en/node/49>
 - Création d'un script d'installation automatique des librairies

- Calibration circulaire

- Modification de CCV pour permettre une calibration circulaire. Affichage des points de calibration sur des cercles concentriques partant du centre de l'écran.
- Possibilité d'augmenter ou diminuer le nombre de cercles concentriques.
- Possibilité d'augmenter ou diminuer le nombre de points de calibration sur chaque cercle.

- Tests calibration circulaire

- Le module de calibration circulaire étant implémenté, nous avons cherché à déterminer si les corrections sont correctes :
- Les corrections ne se font pas correctement. Le programme CCV utilise des zones triangulaires afin de corriger l'emplacement des blobs. Les zones triangulaires obtenues à partir des points de calibrations disposés en carrés ne sont plus disponibles à partir des points de calibrations disposés en cercles.

- Recherche d'une méthode de correction

- Trouver et implémenter une méthode permettant de corriger l'emplacement des blobs à partir des points de calibrations situés sur des cercles concentriques.

- Optimisation répartition

- Les points de calibration n'étaient pas répartis de manière homogène sur la surface.
- Solution :
 - L'utilisateur choisit le nombre de points de calibration présents sur le premier cercle.
 - Sur les autres cercles les points sont positionnés de manière à toujours obtenir une longueur d'arc séparant deux points égale à celle calculée entre deux points situés sur le premier cercle.

- Implémentation Delaunay

- La solution au problème de correction est d'implémenter l'algorithme de Delaunay permettant d'obtenir des zones triangulaires

- Compatibilité Windows

- Installer l'environnement de développement sous la partition Windows. Vérifier que notre projet est compilable par Visual Studio.

- Utilisation du GPU

- Installation des pilotes graphiques de la carte Nvidia

- Correction problème de résolution

- Le positionnement des points de calibration était dépendant du format de l'écran et pouvaient apparaître sous forme de cercle ou ellipse en fonction du format

- Tests réels calibration

- Tests du module de calibration circulaire et de son algorithme de correction sur la table. La correction n'est pas valide particulièrement sur la périphérie des triangles de Delaunay..

- Correction calibration

- Correction du module de correction de positionnement des blobs.

- Tests calibration corrigée

- Le module de calibration circulaire ainsi que son algorithme de correction sont valides

III. Compilation de CCV

1) Qu'est ce que CCV ?

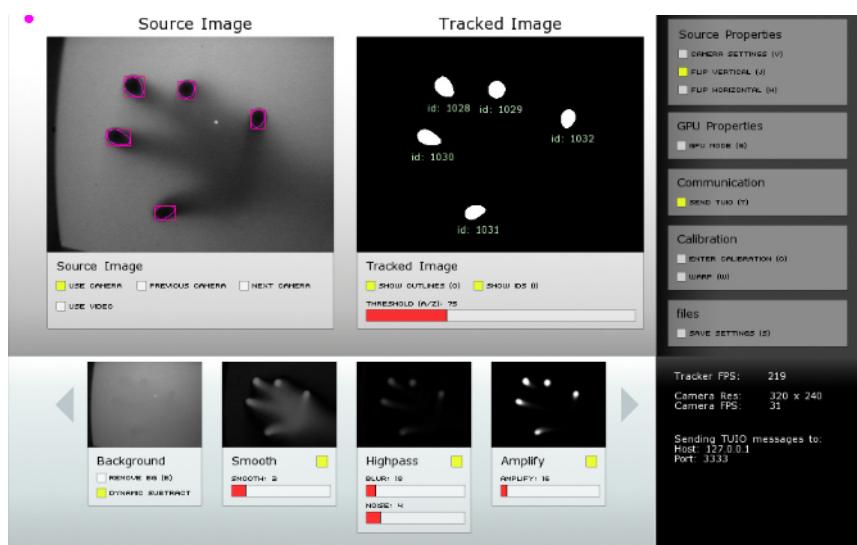


Figure 2 : Interface graphique CCV

Community Core Vision, autrefois appelé tbeta, est un logiciel qui permet de calculer la position de « blobs » et leur orientation à partir d'une image. Il prend en entrée le chemin de la webcam et récupère les images de la webcam à intervalle régulier en appelant les fonctions d'accès au driver (*ioctl* en C). L'utilisateur peut spécifier des filtres à appliquer sur l'image afin que les blobs soient identifiables correctement.

Dans notre cas, on utilise CCV pour détecter des doigts appuyés sur la surface de la table. Les blobs sont donc l'image du bout des doigts des utilisateurs. CCV permet de calculer le centre des blobs, leur direction et de générer des événements (appui, mouvement, disparition,...) qu'il envoie avec le protocole TUIO sur le réseau.

2) Récupération de CCV

CCV est un projet open-source, c'est à dire que le code source du programme est disponible gratuitement et que nous avons le droit de le modifier pour notre usage. L'objectif de notre TX étant de changer le mode de calibration du programme, nous avons nécessairement besoin du code source pour pouvoir le modifier et tout recompiler. Il est aussi possible, de récupérer l'exécutable déjà compilé aussi bien pour Linux que pour Windows sur leur site officiel : <http://ccv.nuigroup.com/>.

Le code est portable, il fonctionne aussi bien sous Windows que sous Linux quelle que soit la distribution. Les développeurs ont mis à disposition un serveur SVN qui nous permet de récupérer l'ensemble du code de CCV avec un client SVN. Nous avons récupéré le contenu de la branche nommée Linux pour la totalité du code source avec le projet CodeBlocks. De même, sous Windows, il y a une branche du même nom qui nous fournit le projet Visual Studio 2008.

```
# Création du dossier pour les sources  
mkdir sources  
  
# Linux source code (SVN Checkout)  
svn checkout http://nuicode.svnrepository.com/svn/tbeta/trunk/tbeta/Linux
```

Récupération du code pour Linux

Bien qu'il s'agisse de deux branches différentes du SVN, le code source est le même, la seule chose qui change est le projet qui permet de tout compiler. Cela veut dire que toutes les modifications que nous faisons sur Linux sont aussi valables sous Windows dans la mesure où notre code est aussi portable.

Sous Linux, l'utilisation de CodeBlocks est conseillé pour compiler le projet car les développeurs nous fournissent le fichier du projet déjà configuré.

3) Configuration de l'environnement Linux à partir d'une distribution Ubuntu

CCV utilise un certain nombre de librairies et de ressources qui doivent être installées préalablement à la compilation sur Ubuntu. Pour Windows, il suffit juste d'installer le pilote de la webcam et Visual Studio 2008.

Tout d'abord, il faut installer le logiciel CodeBlocks s'il n'est pas déjà sur la machine. Pour cela, on peut installer le paquet depuis le gestionnaire de paquet via la commande :

```
apt-get install libcodeblocks0 codeblocks libwxsmithlib0 codeblocks-contrib  
libwxgtk2.8-0
```

Paquets nécessaires pour installer CodeBlocks

Avec CodeBlocks d'installé, on peut maintenant ouvrir le fichier « sources/Linux/apps/addonsexamples/Codeblocks_8_linux/Community Core Vision.cbp ». Cependant la compilation nécessite plusieurs librairies qui peuvent ne pas être installées sur le PC. De la même manière que pour CodeBlocks, nous utilisons un script SH qui les installe toutes.

```
apt-get install libgstreamer0.10-dev libgstreamer-plugins-base0.10-dev  
libavcodec-dev libavformat-dev libavutil-dev libswscale-dev freeglut3-dev  
libasound2-dev libxmu-dev libxxf86vm-dev g++ libgl1-mesa-dev libglu1-mesa-dev  
libraw1394-dev libpoco-dev
```

Paquets nécessaires pour compiler le projet CodeBlocks

Si des paquets sont manquants ou obsolètes, ils seront automatiquement mis à jour. Pour automatiser ce processus sans avoir besoin d'internet, on peut simplement placer les paquets (avec l'extension .deb) dans le dossier de téléchargement du gestionnaire de paquet : « /var/cache/apt/archives ». L'installation se fera alors à partir de ces paquets plutôt qu'avec ceux disponibles sur internet.

A ce stade, la compilation devrait s'effectuer correctement. Nous avons eu plusieurs problèmes lors de la compilation mais ceux ci ne devrait pas apparaître lors de la compilation depuis notre code modifié. Pour résoudre ces problèmes depuis le code du SVN, nous avons suivis les indications des personnes ayant le même soucis sur le forum de CCV :

<http://nuigroup.com/forums/viewthread/8251/>

<http://www.openframeworks.cc/forum/viewtopic.php?f=5&t=1929>

Si le problème "`UINT64_C was not declared in this scope`" apparaît, on peut le résoudre simplement en ajoutant dans le fichier « /usr/include/libavutil/common.h » les lignes suivantes :

```
#ifdef __cplusplus
#define __STDC_CONSTANT_MACROS
#ifndef _STDINT_H
#define _STDINT_H
#endif
#include <stdint.h>
#endif
```

Lignes à ajouter dans le fichier common.h en cas de problème

4) Installation de UNICAP

Bien que le programme compile normalement, il est nécessaire d'installer aussi UNICAP. C'est une librairie C qui permet d'accéder à la webcam depuis le programme C. Normalement, CCV peut fonctionner avec une autre librairie appelé V4L (video for Linux) mais l'installation de UNICAP est quand même requise. En effet, avant d'utiliser UNICAP, nous avions précisé dans le code CCV que nous utilisions V4L mais il y avait un problème d'accès à la webcam qui s'est réglé en installant UNICAP. Nous avons finalement décidé d'utiliser UNICAP sans V4L puisque son installation était obligatoire et que, par défaut, c'est cette librairie qui est utilisée.

Nous avons créé un script SH qui installe la bonne version de UNICAP. Ce script demande à l'utilisateur s'il doit télécharger les fichiers requis depuis internet ou s'il veut les installer depuis des fichiers stockés en local sur le PC. Nous avons fait une copie locale des fichiers afin de pouvoir installer UNICAP sans internet et pour que ce soit toujours possible si le site qui héberge actuellement les fichiers venait à fermer. Grâce à notre script, la machine compile et installe UNICAP sur le système, tout est automatisé.

```
wget http://unicap-imaging.org/downloads/unicap-0.9.5.tar.gz
tar -xvzf unicap-0.9.5.tar.gz
wget http://kaswy.free.fr/sites/default/files/download/ps3eye/unicap/unicap-
gspca.patch
patch -p0 < unicap-gspca.patch
cd unicap-0.9.5
./configure
make
sudo make install
```

Résumé des commandes faites pour installer UNICAP en version 0.9.5

Une fois UNICAP installé, le programme devrait se lancer normalement et la webcam devrait être détectée si les pilotes sont correctement installés. Nous avons testé avec plusieurs webcams dont celle de la PS3 et, à chaque fois, le pilote générique suffisait à son fonctionnement.

5) Configuration par défaut

Lorsque CCV démarre, il lit le fichier « config.xml » situé dans le dossier « data » qui se trouve dans le même dossier que l'exécutable. Ce fichier contient les paramètres utilisés par défaut lors du lancement de l'application. Il est donc très utile de garder une copie de ce fichier lorsque le programme a été paramétré correctement. On peut aussi spécifier dans ce fichier le taux de rafraîchissement de la webcam mais celui-ci ne dépassera pas les limites théoriques imposées par le matériel et le driver. On y définit aussi la résolution de l'image d'entrée. Plus cette résolution est grande, meilleure sera la localisation des appuis de l'utilisateur mais cela augmentera aussi le temps de calcul.

```
<!--// CAMERA SETTINGS // -->
<CAMERA_0>
    <USECAMERA>1</USECAMERA>
    <DEVICE>1</DEVICE>
    <WIDTH>640</WIDTH>
    <HEIGHT>480</HEIGHT>
    <FRAMERATE>48</FRAMERATE>
</CAMERA_0>
```

Paramètres de la webcam dans le fichier de configuration

De même, il existe un fichier qui stocke les valeurs de la calibration. Cela permet de garder la même calibration même lorsqu'on relance l'application. Ce fichier est nommé « calibration.xml » et se trouve dans le même dossier que le fichier de configuration de CCV « config.xml ».

IV. Module de calibrage

Le module de calibrage fait partie intégrante de l'application CCV. Toutefois, ce dernier est destiné au calibrage d'une surface rectangulaire. Le projet *InTact* utilise une surface ronde, aussi il advient nécessaire de modifier l'application en conséquence.

1) Principe

Le principe de la calibration est de faire saisir les coordonnées réelles à l'écran d'un certain nombre de points par un utilisateur. Ensuite, il sera possible de calculer les coordonnées de tous les points en fonction des points calibrés. Pour ce faire, nous allons afficher un ensemble de points à l'écran. Un utilisateur appuiera sur chacun des points et nous aurons donc les coordonnées de chacun des points de calibration pour la caméra, et pour l'écran. Ensuite, pour chaque pixel lisible par la caméra, on calculera ses coordonnées à l'écran en fonction des trois points de calibration les plus adaptés.

2) Algorithmes

L'algorithme permettant le calibrage de la table est le suivant :

1. Initialisation_des_paramètres
2. Initialisation_de_la_fenêtre_de_placement_des_points
3. Initialisation_des_points_de_calibration
4. Initialisation_du_maillage_de_triangles
5. Calcul_de_la_matrice_de_conversion

Nous allons expliquer chaque étape de cet algorithme.

a) Initialisation des paramètres

Fonction(s) : *setGrid(...)*, *loadXMLSettings()*

La première étape consiste à initialiser le nombre de points sur le premier cercle (GRID_X+1) ainsi que le nombre de cercle (GRID_Y+1). Ensuite, nous allons allouer de la mémoire pour les vecteurs contenant les points de la caméra, les points à l'écran ou les maillages de triangles.

b) Création d'une fenêtre pour le placement des points

Fonction(s) : *setScreenBBox(...)*, *calculateBox()*

La première étape de la calibration consiste à afficher ces points à l'écran. Pour cela, il faut calculer leurs coordonnées. Il serait possible de calculer directement les coordonnées des points. Toutefois, il faudrait les recalculer à chaque fois que l'utilisateur déplace ou redimensionne la zone de calibration. Afin d'éviter cela, nous allons créer une fenêtre de dimension 1x1. Chacune des coordonnées sera calculées sera donc dans cet intervalle [0 ; 1]. Lors d'un redimensionnement de la zone de calibration, il sera donc inutile de recalculer les coordonnées des points. En effet, peu importe la taille de cette zone, il suffira de multiplier l'abscisse et l'ordonnée de chaque point de

calibration respectivement par la largeur et la hauteur de cette zone. De même, lors d'un déplacement de la zone de calibration, il suffira d'additionner ou de soustraire un réel aux coordonnées.

c) Calcul des coordonnées des points de calibration

Fonction(s) : *initScreenPoints(), initCameraPoints(...)*

La surface de la table tactile étant ronde, il nous faut placer les points de calibrage sur des cercles. Pour cela, on détermine le centre de l'image capturée par la caméra et nous placerons un point à cet endroit. Ensuite, nous allons un placer un nombre identique de points sur chacun des cercles en utilisant les coordonnées polaires de chaque point comme suit :

```

 $\Delta_{rayon} \leftarrow \min(hauteur, largeur) / (2 * (GRID_Y + 1))$ 
degree  $\leftarrow 0$ 
nb_points  $\leftarrow GRID_X$ 
rayon  $\leftarrow \Delta_{rayon}$ 
conv  $\leftarrow \text{largeur}_{image}/\text{hauteur}_{image}$ 
matrice  $\ll$  centre
pour ( $j \leftarrow 1$  à  $GRID_Y + 1$ )
     $\Delta_{degree} \leftarrow 2\pi/(GRID_X + 1)$ 
    pour ( $i \leftarrow 0$  à nb_points)
        matrice  $\ll$  centre +  $\begin{pmatrix} \text{rayon} + \cos(degree)/\text{conv} \\ \text{rayon} + \sin(degree) \end{pmatrix}$ 
        degree  $\leftarrow degree + \Delta_{degree}$ 
    fpour
    rayon  $\leftarrow rayon + \Delta_{rayon}$ 
    nb_points  $\leftarrow nb\_points + GRID_X$ 
fpour

```

Le résultat est le suivant :

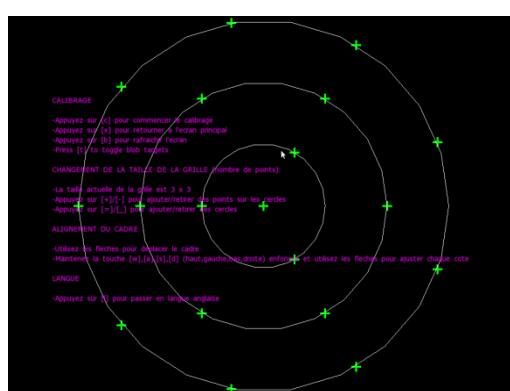


Figure 3 : Positionnement des points sur les cercles

d) Calcul de la matrice de conversion

Fonction(s) : `cameraToScreenSpace(...)`, `inCalibrationSector(...)`, `isOnEdge(...)`, `findTriangleWithin(...)`

Afin d'obtenir rapidement les coordonnées à l'écran de chaque blob lu par la caméra, nous allons créer une matrice de conversion qui, pour chaque pixel de la caméra, contiendra ses coordonnées à l'écran.

La première étape consiste donc à vérifier que le pixel se situe dans la zone de calibration. Pour cela, on extrait les coordonnées du premier point de calibration qui est le centre des cercles, ainsi que les coordonnées du dernier point de calibration qui se situe sur le cercle extérieur (cf. algorithme de placement des points). Enfin, nous allons calculer la distance entre ces deux points. Pour chaque pixel de la caméra, il nous reste donc plus qu'à calculer la distance entre ce pixel et le premier point de calibration :

- Si cette distance est inférieure ou égale au rayon du cercle extérieur, on passe à l'étape suivante.
- Sinon, les coordonnées du point seront (-1 ; -1). Ces coordonnées seront reconnues plus tard comme celle d'un point non valide.

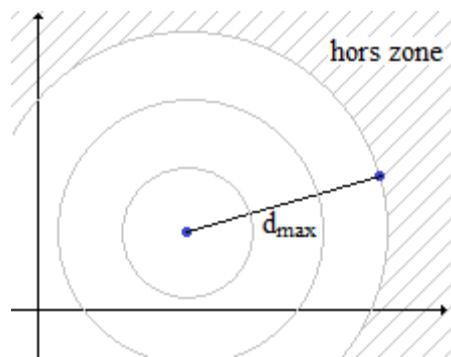


Figure 4 : Zone de calibration

Ensuite, si le point est valide, il faut calculer ses coordonnées à l'écran à partir des points de calibration saisis au préalable. Nous avons d'utiliser uniquement 3 points de calibration. Soit quatre points A, B, C et D dans \mathbb{R}^2 disposé comme suit :

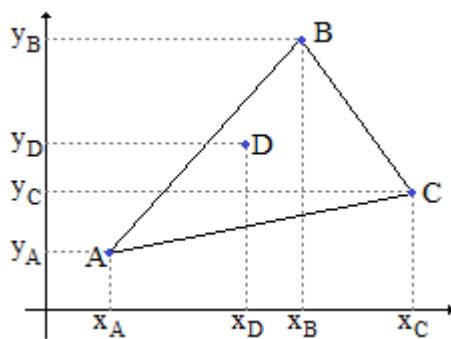


Figure 5 : Calcul des coordonnées d'un point (cas 1)

Il est possible de calculer les coordonnées D à partir de A, B et C en utilisant diverses méthodes. Toutefois, il se peut que le point D soit dans la zone de calibration (à l'intérieur du cercle extérieur), mais à l'extérieur des triangles, comme suit :

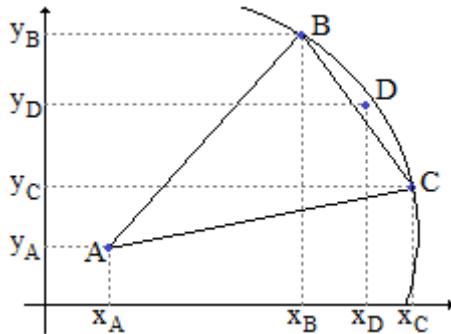


Figure 6 : Calcul des coordonnées d'un point (cas 2)

Nous avons donc choisi de calculer les coordonnées du point D en utilisant la technique du barycentre car elle est indépendante de la position de ce point par rapport au triangle. Les coordonnées de chaque point M de calibrage sont $(x_M \ y_M)^T$ sur l'image de la caméra et $(x'_M \ y'_M)^T$ sur l'image à l'écran. Nous noterons que x_M, y_M, x'_M et $y'_M \in \mathbb{R}^+$. Il existe un unique $(\alpha_1, \alpha_2, \alpha_3)$, à un coefficient multiplicateur près, tel que :

$$\alpha_1 \vec{DA} + \alpha_2 \vec{DB} + \alpha_3 \vec{DC} = \vec{0} \quad (1)$$

On obtient le système suivant :

$$\begin{cases} \alpha_1(x_A - x_D) + \alpha_2(x_B - x_D) + \alpha_3(x_C - x_D) = 0 \\ \alpha_1(y_A - y_D) + \alpha_2(y_B - y_D) + \alpha_3(y_C - y_D) = 0 \end{cases} \quad (2)$$

On considère $\alpha_3=1000$ (choix effectué). Les coordonnées barycentrique de D sont :

$$\alpha_2 = \alpha_3 \frac{(y_C - y_D)(x_A - x_D) + (x_D - x_C)(y_A - y_D)}{(x_B - x_D)(y_A - y_D) + (y_D - y_B)(x_A - x_D)} \quad (3)$$

$$\alpha_1 = \frac{\alpha_2(x_D - x_B) + \alpha_3(x_D - x_C)}{x_A - x_D} \quad (4)$$

Ainsi, les coordonnées « corrigées » de D sont :

$$x'_D = \frac{\alpha_1 x'_A + \alpha_2 x'_B + \alpha_3 x'_C}{\alpha_1 + \alpha_2 + \alpha_3} \quad (5)$$

$$y'_D = \frac{\alpha_1 y'_A + \alpha_2 y'_B + \alpha_3 y'_C}{\alpha_1 + \alpha_2 + \alpha_3} \quad (6)$$

Ainsi, nous pouvons calculer aisément les coordonnées à l'écran de chaque point lu par la caméra. Toutefois, il est nécessaire de prendre quelques précautions avant d'effectuer ce calcul :

- Si D est un point de calibration, on n'effectue aucun calcul. On affecte seulement les coordonnées saisies lors du calibrage.
- Si D est sur une arête du triangle, on calcul la position de D en fonction des deux points de calibration formant cette arête. Cette distinction est obligatoire car nous avons affectés un poids positif à α_3 (=1000) et si D est sur [A,B] alors le résultat serait faux.
- Si D est à l'intérieur du triangle, il faut parfois intervertir les points A, B et C afin que les diviseurs soient non nuls (équations 3 et 4). Cela signifie que le point D ne doit pas être sur la même abscisse que A, ni à la fois sur la même ordonnée que A et B.

Soit D le point dont les coordonnées sont à calculer, l'algorithme de calcul est le suivant :

```

dmax ← norm(center,dernier_point_de_calibration)
dist ← norm(center,D)
si (dist > dmax)alors
|   (x'_D  y'_D) ← (-1  -1)
sinon
|   recherche_des_trois_points_de_calibration_optimals
|   si (D est un point de calibration) alors
|   |   (x'_D  y'_D) ← (x_calibration  y_calibration)
|   sinon si (D est sur une arrête [α,β]) alors
|   |   K ← norm(α,D)/norm(β,D)
|   |   (x'_D  y'_D) ← (K(x_β - x_α) + x_α  K(y_β - y_α) + y_α)
|   sinon si (D est à l'intérieur du triangle) alors
|   |   si (swap nécessaire) alors
|   |   |   interversion_des_points
|   |   fsi
|   |   (x'_D  y'_D) ← résultat_du_calcul_via_barycentre
|   fsi
fsi

```

Cette méthode de calcul remet en cause le positionnement des points. En effet, pour utiliser cette méthode, il nous faut rechercher trois points. Toutefois, pour avoir une erreur similaire pour tous les points de la caméra, la distance entre les points sur chaque cercle doit être identique. De plus, ces points ne doivent pas être alignés (car le calcul de barycentre ne fonctionnerait pas). Les trois points formant un triangle, le calcul est plus précis lorsque les angles de ce triangle sont « grands ». On a donc choisi de récupérer les trois points nécessaires au calcul de chaque pixel en formant un maillage de triangle avec les points calibrés. Pour cela, nous allons utiliser la triangulation de Delaunay.

e) Triangulation de Delaunay

Fonction(s) : `initTriangles()`

La triangulation de Delaunay d'un ensemble P de points du plan est une triangulation DT(P) telle qu'aucun point de P n'est à l'intérieur du cercle circonscrit d'un des triangles de DT(P). Nous avons choisi cette méthode car la triangulation de Delaunay maximise le plus petit angle de l'ensemble des angles des triangles, évitant ainsi les triangles « allongés ». Cela nous permet de réduire l'erreur de calcul obtenu avec la technique du barycentre vue précédemment.

D'après la définition de Delaunay, le cercle circonscrit d'un triangle constitué de trois points de l'ensemble de départ est vide s'il ne contient pas d'autres sommets que les siens. Les autres points sont autorisés sur le périmètre en lui-même mais pas à l'intérieur strict du triangle.

Pour effectuer la triangulation, nous allons commencer par insérer la zone de calibration dans un carré que l'on divisera en deux triangles comme suit :

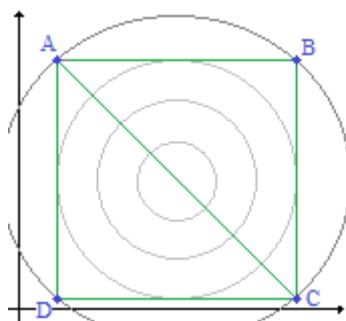


Figure 7: Inscription de la zone de calibration dans deux triangles

Ce maillage est un maillage d'un Delaunay car les cercles circonscrits au cercles \widehat{ABC} et \widehat{ACD} ne contiennent aucun point, hormis sur le cercle.

Ensuite, nous allons ajouter, un par un, les points de calibration au maillage de triangle. Nous noterons que les points seront obligatoirement à l'intérieur ou sur l'arrête d'un triangle. Ainsi, il suffira de « découper » le triangle en trois nouveaux triangles. Si l'on prend l'exemple du premier point de calibration (E, le centre des cercles), on obtient :

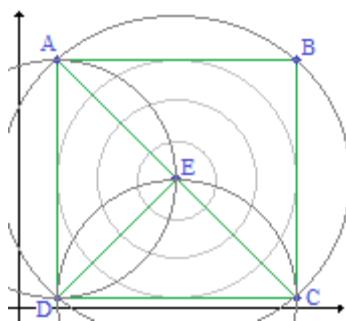


Figure 8: Ajout du premier point (erroné)

On peut remarquer, ici, que le maillage de triangles de correspond plus à une triangulation de Delaunay car le point E est inclus dans le cercle circonscrit au triangle \widehat{ABC} . Pour corriger

cela, nous allons diviser le triangle \widehat{ABC} en deux triangles afin de respecter les conditions de la triangulation. On répète cette division jusqu'à ce que le maillage soit valide, on obtient :

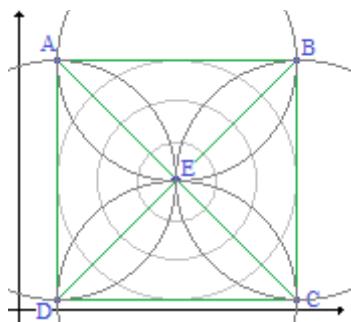


Figure 9: Ajout du premier point (corrigé)

Ensuite, il ne reste plus qu'à répéter ces opérations avec tous les points de calibration restants. Le résultat de la triangulation avec 4 cercles et 5 points sur le premier cercle est le suivant :

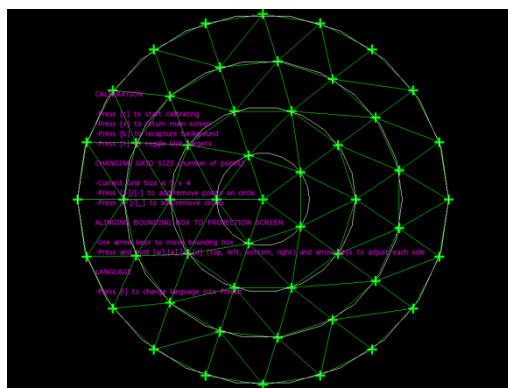


Figure 10: Maillage de triangle

Nous noterons que dans le cas où le point se situe dans la zone de calibration mais en dehors du maillage de triangle, on se contente de rechercher les trois points les plus proches. Par définition, les deux points les plus proches se trouvant sur le cercle, le troisième ne peut pas être aligné avec les deux autres, cela ne gêne donc pas le calcul par la méthode du barycentre.

f) Utilisation

Fonction(s) : *cameraToScreenPosition()*

Une fois les matrices proprement initialisées, pour connaître les coordonnées à l'écran de n'importe quel point de la caméra, il suffit d'extraire les coordonnées de la matrice sans aucun calcul.

3) Calibrage de la table

Pour effectuer la calibration, il suffit de lancer CCV, puis d'appuyer sur la touche  du clavier pour accéder au module de calibration. Il devient ainsi possible de tester le calibrage actuel en appuyant n'importe où sur la table. Il est possible de visualiser l'erreur de calibrage en mesurant la distance entre la croix dessiné sur la table et votre doigt.

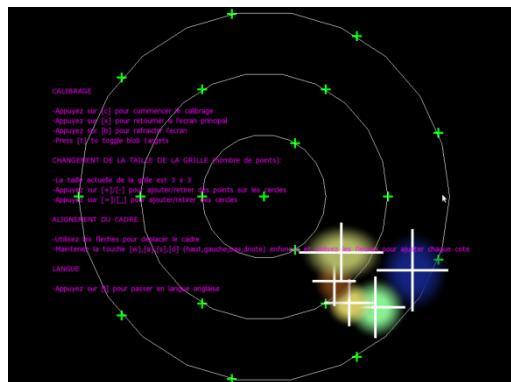


Figure 11: Test du calibrage actuel

Si l'erreur est trop importante, alors lancer la calibration en appuyant de nouveau sur la touche  du clavier. Les points seront, chacun leur tour, entouré d'un cercle rouge. Pour calibrer un point, il faut toucher le centre de la croix avec son doigt et attendre que le cercle devienne blanc.

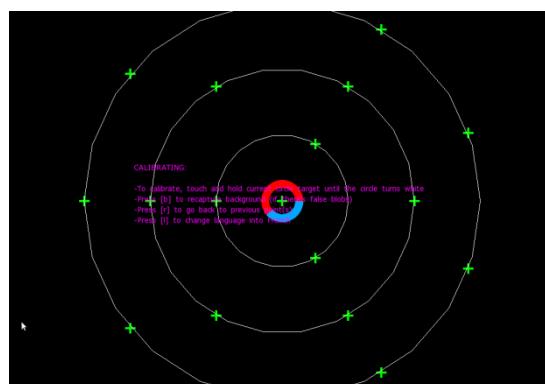


Figure 12: Calibration du premier point

Vous devez réaliser la même opération pour chaque point de la table. Une fois terminé, vous pouvez à nouveau tester votre calibration. Le résultat devrait être bien meilleur.

a) Options

Afin de rendre plus ais e le calibrage de la table, nous avons mis en place un syst me permettant la traduction en langue fran aise des instructions permettant le calibrage.

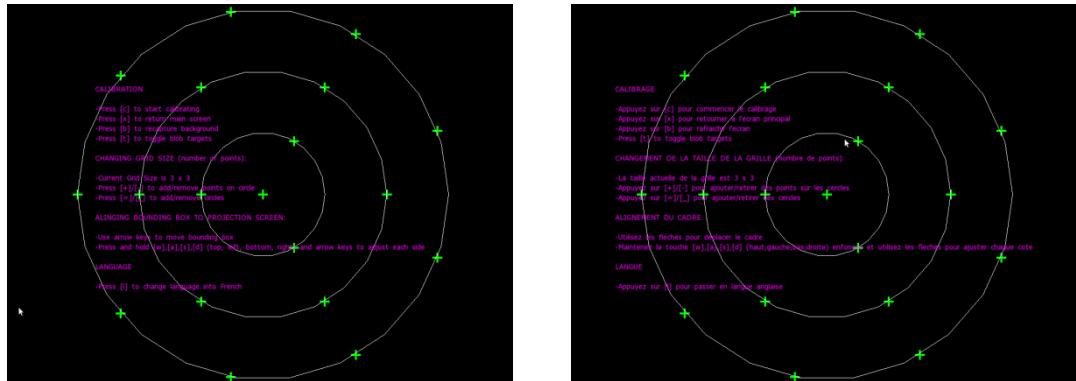


Figure 13 : Traduction du module de calibration en anglais et en fran ais

Aussi, afin d'ajuster parfaitement la zone de calibration   la table, il est possible d'agrandir et de r tr cier la zone ou encore de la d placer. Enfin, pour am liorer la qualit  du calibrage, il est possible d'augmenter ou de diminuer le nombre de cercles (minimum 3) et le nombre de points sur les cercles (minimum 3 points sur le premier cercle).

4) valuation des r sultats

Pour v rifier que notre module de calibrage est correct, nous allons afficher un graphe pr sentant la distance entre les coordonn es   l' cran calcul es pour chaque point de la cam ra et les coordonn es du premier point de calibration (centre des cercles). On obtient :

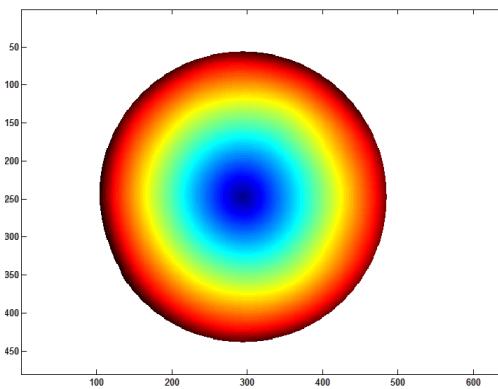


Figure 14: Distance entre les points et le centre des cercles (apr s calcul)

Sur ce graphique, plus la couleur est froide, plus la distance entre le point et le centre est petite. Au contraire, plus la couleur est chaude (jusqu'  rouge) plus la distance entre le point et le centre est grande. On remarque ici que la distance entre les points et le cercle augmente lin airement lorsque l'on s' carte du cercle et que la position d'un point par rapport aux triangles n'interf re pas dans le r sultat du calcul. Ce r sultat s'est v rifi  en testant le module sur la table tactile.

5) Limite

Toutefois, nous noterons une interférence du hardware sur la qualité de la calibration. En effet, lorsqu'un blob est très proche d'un émetteur infrarouge, celui-ci se voit étiré lorsqu'il est lu par la caméra. De part ce fait, le centre du blob est déplacé avant même les calculs de ces coordonnées « corrigées ». Il y a donc un écart de quelques millimètres entre notre doigt et la croix dessinés sur la table (figure 11 p22). Le résultat reste très réaliste dès que l'on s'écarte de quelques centimètres des émetteurs infrarouges.

CONCLUSION

Le projet InTact nous a apporté une expérience intéressante à plusieurs points de vue :

- Nous avons découvert l'évolution d'un projet Open-source et sa communauté
- Nous avons implémenté des algorithmes complexes, en particulier concernant la triangulation de Delaunay

Nous avons réalisé les difficultés intrinsèques à un projet Open-source :

- Une qualité de code hétérogène
- Défauts de compatibilités
- Pertinence des commentaires variable

Enfin, il nous a permis de découvrir un domaine technologique en vogue : celui des médias tactiles. Le projet InTact a en plus le mérite d'apporter une dimension innovante.