Rapport projet PJE

Elliot VANEGUE, Gaëtan DEFLANDRE et Samuel GRANDSIR

6 mai 2015

Sommaire

Table des figures

Introduction

Dans le cadre de notre formation en licence informatique à l'université de Lille 1, nous nous sommes orienté vers le projet permettant de découvrir l'une des spécialités de master. Nous avons sélectionné le projet "Interface Multi-Touch" qui correspond à la spécialité IVI [1]. Ce projet a pour but de réaliser une interface tactile pour interagir avec un système permettant de manipuler des images sur un ordinateur. Ce rapport reprend l'ensemble des connaissances acquises durant la première partie du projet qui correspond à des travaux pratiques manipulant la caméra utilisée pour l'interface multi-touch. Durant ce rapport, nous allons expliquer :

— comment fonctionne la formation d'une image,
— comment la caméra peut acquérir de l'information et la numériser,
— comment récupérer des informations sur les doigts de l'utilisateur,
— et omment différencier les différents doigts de l'utilisateur.

Pour répondre à ces questions, nous allons d'abord détailler les éléments d'optiques mis en oeuvre dans ce projet. Puis, nous allons utiliser les bibliothèques d'Opencv pour découvrir comment récupérer et manipuler des images provenant de la caméra. Ensuite, nous verrons comment appliquer des filtres à des images. Et enfin, nous détaillerons le programme permettant de récupérer des informations sur les doigts de l'utilisateur.

1. Image Vision Interaction

1 Elément d'optique

1.1 Dioptre plan

1.1.1 Rayon linéaire

Le dioptre est une surface qui sépare deux milieux transparents homogènes et isotropes ayant un indice de réfraction différent. Pour connaître l'indice de réfraction le plus important, il suffit de déterminer quel est l'angle le plus grand entre l'angle d'incidence et l'angle de réfraction. Puis, avec la loi de **Snell-Descartes**, il est possible de retrouver l'indice de réfraction du deuxième milieu (n_2) :

$$n_2.sin(i_2) = n_1.sin(i_1) \Rightarrow n_2 = \frac{n1 * sin(i_1)}{sin(i_2)}$$

A partir de cette formule, on peut voir que si l'angle i1 est plus grand que l'angle i2, alors l'indice de réfraction le plus petit est n1.

1.1.2 Source ponctuelle

Un système optique est dit stigmate s'il dévie les rayons d'un point source de telle sorte qu'il soit à nouveau concourant en un point en sortant du système.

La simulation 2 que nous étudions ne possède pas un système optique stigmate car si on prolonge les rayons du milieu 2 dans le milieu 1, ceux-ci ne convergent plus en un seul point. Il semble que plus la source de lumière est éloignée et moins elle est étalée, plus le dioptre adopte un comportement stigmate. Lorsque le système se rapproche de ce comportement, on peut dire que l'image est virtuelle car le point de "convergence" des rayons lumineux se situe derrière le système optique.

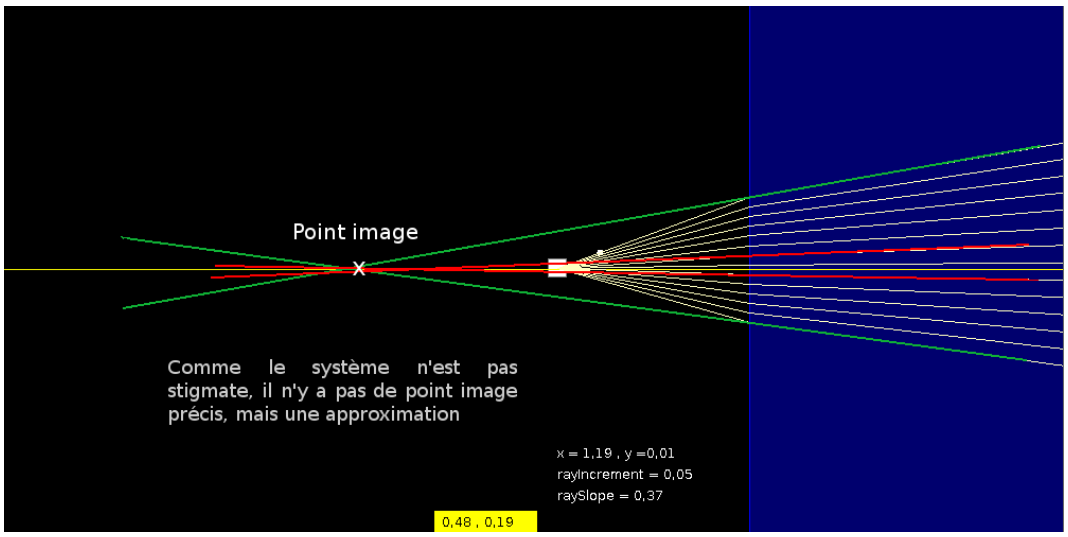

FIGURE 1 – Source ponctuelle

La réaction de ce système fait qu'un observateur qui est à droite du dioptre de la simulation, verra l'objet plus éloigné qu'il ne l'est vraiment. Comme le système n'est pas stigmate, l'image aura une légère distorsion aux yeux de l'observateur. Ces phénomènes sont expliqués par le fait

que si on prolonge les rayons du milieu 2 dans le milieu 1, l'ensemble des prolongements coupe l'axe optique [2] derrière l'objet (source de lumière) et ces prolongements ne sont pas concourants.

Lorsque l'on inverse les deux milieux, on voit que les angles et les propriétés du dioptre sont inversés. Le système reste donc astigmate et l'image est toujours virtuelle. Pour l'observateur se trouvant à droite du système optique, l'objet apparaîtra plus gros, et il semblera plus proche toujours avec une distorsion.

1.2 Réflexion totale

La réflexion totale est un phénomène qui fait qu'à partir d'un certain angle d'incidence un rayon lumineux ne quitte plus son milieu. Nous allons voir ce phénomène dans la seconde simulation qui représente une plaque à faces parallèles.

Lorsque la source lumineuse est dans le milieu un, on voit qu'il y a un léger décalage entre les rayons entrants et sortants de la plaque. Cependant, ces rayons restent parallèles entre eux. Les effets des deux dioptres s'annulent étant donné que le milieu à gauche du premier dioptre est le même que le milieu à droite du second dioptre.

On constate que lorsque la source lumineuse se retrouve entre les deux dioptres, les rayons restents à l'intérieur de la plaque. Cet effet est annulé lorsque l'angle d'incidence est inférieur ou égale à 0,47 radian soit environ 27 degrès. Cette limite nous permet de déterminer l'indice de réfraction du milieu situé entre les deux dioptres grâce au calcul suivant :

$$n_2 = \frac{1 * sin(27)}{sin(79)} = 0,46$$

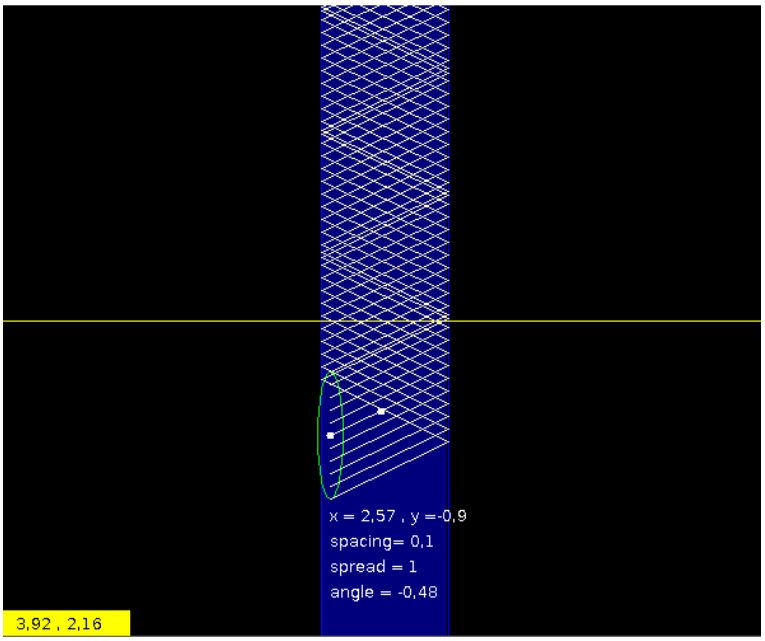

FIGURE 2 – Illustration de l'effet FTIR avec un angle de 28 degrès

A partir de cet angle, il est possible de retrouver le demi-angle de la diode. En effet, le dioptre

2. la normale du dioptre

à la surface du plexiglas et les dioptres sur les arêtes du plexiglas sont différents. Il faut donc faire correspondre l'angle trouvé auparavant avec l'angle de réfraction de la diode placée en face de l'arête du plexiglas, c'est le demi-angle de cette diode qui est réfracté en passant par l'arête du plexiglas, puis se réfléchit sur la surface du plexiglas, à l'intérieur de celui-ci. Ce phénomène est appelé réflexion totale ou FTIR [3].

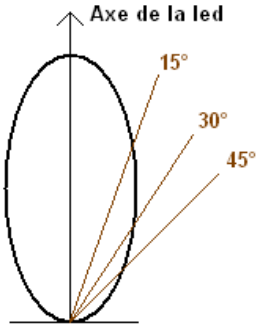

FIGURE 3 – Schéma des différents demis-angles d'une LED

1.3 Objectif et mise au point

La troisième simulation contient deux lentilles, dont une mobile, par laquelle passe des rayons lumineux.

Sans modifier la position de la source lumineuse, l'image devient nette quand la position x de la lentille mobile atteint 4,19. A ce moment, les rayons sont concourants sur le capteur.

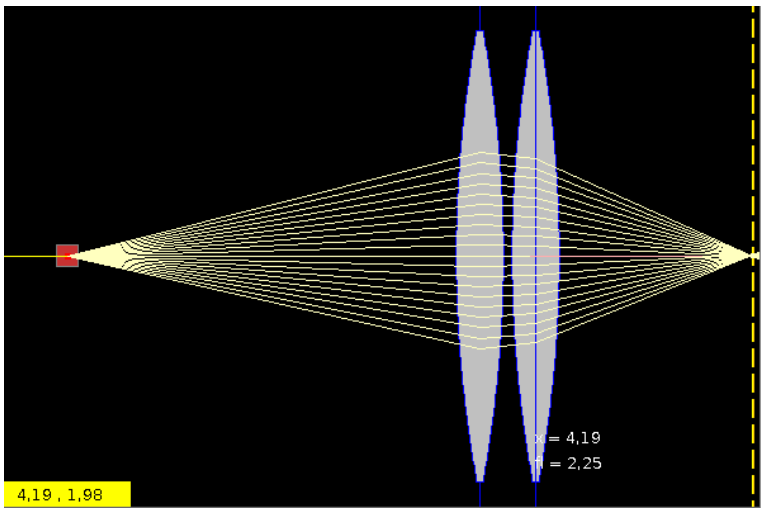

FIGURE 4 – Illustration de la mise au point

En modifiant la position de la source lumineuse, on peut voir que l'image reste nette tant que la source se déplace sur un axe vertical. En effet, la mise au point dépend de la distance de l'objet par rapport à l'objectif. Si l'on bouge l'objet sur un axe horizontal, la distance entre l'objet et l'objectif change et donc l'image de l'objet n'est plus nette. Il faut alors bouger la lentille mobile pour faire la mise au point. On remarque que quand on approche l'objet de l'objectif, à partir d'un certain point l'image de l'objet devient virtuelle.

3. Frustrated Total Internal Reflection

Pour obtenir la distance minimale entre l'objet et l'objectif, il faut coller la lentille mobile à la lentille fixe.

Lorsque la source lumineuse est placée à l'infini, on voit que la lentille mobile doit se situer très près du capteur pour avoir une image nette. En effet, il a été vu précédemment que plus la source lumineuse était proche, plus il faut que la lentille mobile soit proche de la lentille fixe. Ici, c'est le contraire.

La distance focale est la distance entre le centre optique de la lentille et son foyer. Quand l'objet se situe à l'infini, le foyer est confondu avec la surface du capteur, la distance focale f est donc égale à la distance de l'image b.

1.4 Diaphragme et profondeur de champ

Lorsque la mise au point est correctement effectuée sur l'objet et qu'on change l'ouverture du diaphragme, la mise au point reste correcte sur l'objet, les rayons lumineux convergent toujours sur la surface du capteur. En effet, la distance objet et la distance focale ne change pas, donc la distance image ne change pas non plus.

Par mesure, la DMO [4] est de 3.5. on voit que les calculs donnent le même resultat :

$g = 3.5$
$b = b_{max} = 2.17$

$m = \frac{b}{g} = \frac{2.17}{3.5}$
$f = \frac{b}{1+m}$

$DMO = \frac{f * b_{max}}{b_{max} - f} = 3.5$

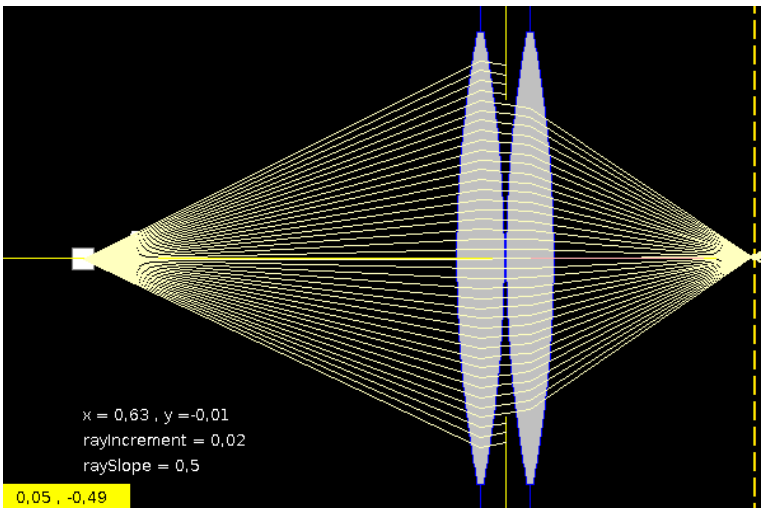

FIGURE 5 – Illustration de la DMO

Lorsque l'ouverture du diaphragme est réglée à 0.5 la DMO reste identique, mais la profondeur de champ est plus grande, donc moins de flou de profondeur de champ.

Avec une source de lumière à une distance infinie, la lentille de mise au point doit forcément être au plus proche du capteur. Le diaphragme garde le même effet de limiter la quantité de lumière entrante en réduisant l'ouverture. L'image apparaîtra donc plus sombre avec légèrement moins de flou de profondeur de champ.

4. Distance minimale de l'objet

2 Acquisition vidéo

2.1 Cadence et réglage du grain de l'image

L'horloge pixel est ce qui permet de définir la cadence de balayage du capteur, c'est donc la période qui sépare deux balayages de la matrice de pixels. Cette horloge pixel a des valeurs bornées avec un minimum de 8 MHz et un maximum de 40MHz. Après avoir fait varier cette valeur, nous avons vu que l'horloge n'avait pas d'impact direct sur l'image. En revanche, l'horloge pixel a un impact sur d'autres paramètres de l'image. Elle plafonne les valeurs de la cadence d'acquisition et du temps d'intégration.

On peut voir que lorsqu'on augmente la cadence d'acquisition et que le temps d'intégration est très bas, l'image clignote quand la pièce est éclairée avec de la lumière artificielle. Ce phénomène est dû à un décalage de phase entre la caméra et le courant alternatif alimentant la lumière.

Le temps d'intégration est le temps durant lequel le capteur reçoit les photons permettant de constituer l'image. Lorsque ce temps est trop bas, on remarque que la luminosité de l'image baisse également. En effet, plus ce temps est bas, moins le capteur reçoit d'énergie et l'image s'assombrit. À l'inverse, plus cette période est longue, plus le nombre de photons reçus par le capteur est grand et plus l'image est lumineuse. Cependant, cela peut entrainer un effet de bougé, car si l'objet bouge, le capteur va récupérer de l'énergie de ses différentes positions.

FIGURE 6 – Illustration du flou de bougé

Le gain est une sorte d'amplificateur du signal. Plus le gain est grand, plus l'image sera lumineuse. En revanche si le gain est trop grand, l'image va perdre de sa qualité et risque de contenir du grain. En effet, le gain amplifie également le bruit du signal, ce qui donne ce grain très caractéristique.

2.2 Taille de l'image

A partir du code fourni, on peut voir qu'il est possible de récupérer certaines informations concernant la caméra. On peut, par exemple, récupérer la résolution maximale du capteur de la caméra qui est de 752x400.

Lorsque l'on réduit la taille de la zone d'intérêt, on voit que l'image qui est récupérée cor-

respond au coin supérieur gauche de l'image. Cependant, c'est le coin inférieur droit du capteur qui est balayé par l'électronique de la caméra et qui reçoit la lumière de la zone d'intérêt.

On voit que lorsque l'on divise la taille de l'image, il est possible de déplacer celle-ci dans l'ensemble de la zone délimitée par la définition maximale du capteur. On peut donc arriver au calcul suivant :

$$iPosx + iSizeX <= iMaxSizeX$$
$$iPosx <= iMaxSizeX - iSizeX$$

Le binning est le fait de regrouper des pixels en utilisant l'ensemble de la surface du capteur. Ceci a pour conséquence de réduire l'image que l'on voit sur l'ordinateur. Si on modifie le binning X, l'image se réduit horizontalement et si on modifie le binning Y, l'image se réduit verticalement.

2.3 Images N et B et couleur

Pour obtenir une image en noir et blanc (i.e. en niveaux de gris) à partir d'une image couleur, il n'est pas nécessaire de modifier le nombre de bits, ni le nombre de canaux car les niveaux de gris sont codés sur 8 bits et n'ont besoin que d'un seul canal. A l'inverse, pour passer d'une image couleur à une image en niveaux de gris, il faut passer à trois canaux (rouge, vert, bleu) et modifier le nombre de bits à 24, 8 bits pour chaque canal.

Avec un arrangement des pixels de type Bayer, et en regroupant les pixels par 2 à la verticale, on obtient une image qui ne tient plus compte des nuances de vert car les pixels correspondant ont été regroupés avec les autres du fait de l'alternance des pixels verts et des autres.

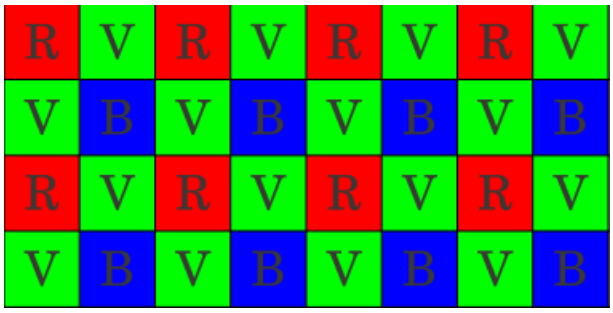

FIGURE 7 – Arrangement de type Bayer

Nous remarquons que si nous mettons le même gain sur chaque couleur, l'image paraît plus verte. En effet, l'homme est capable de voir plus de variantes de la couleur verte que des autres couleurs et le capteur de la caméra comporte plus de pixels associés à la couleur verte. C'est pourquoi il faut ajuster les gains de chaque couleur pour observer une image qui paraît bien proportionnée au niveau des couleurs.

3 Introduction à OpenCV

3.1 Lecture et affichage d'une image

Pour cette introduction à OpenCV, nous allons d'abord commencer par voir comment récupérer certaines informations d'une image. L'exemple suivant nous permet d'afficher dans l'ordre la définition, la taille en octets, la profondeur en bits et le nombre de canaux d'une image.

```
1  // Afficher des proprietes de l'image sur la console
2  printf( "Infos sur l'image %s\n", nomImage.data() );
3  // Definition (largeur x hauteur)
4  printf( "Definition\t: %dx%d pixels\n", image->width, image->height );
5  printf( "Taille \t\t: %d octets\n", image->imageSize );
6  printf( "Profondeur \t: %d bits\n", image->depth );
7  printf( "Nb de canaux \t: %d\n", image->nChannels );
8  printf( "Appuyer sur une touche pour terminer.\n" );
```

Listing 1 – Afficher les champs de la structure IplImage

Suite à l'affichage des quatre images, on peut voir que la taille dépend de la définition, du nombre de canaux et de la profondeur de l'image. En revanche, ces différents paramètres sont indépendants entre eux et leur variation n'est pas toujours visible directement sur l'image.

$$taille = largeur * hauteur * nb\_canaux * \frac{profondeur}{8}$$

La profondeur est divisée par 8 pour passer d'un nombre en bits vers un nombre en octets.

Lorsque l'on charge une image avec la fonction **cvLoadImage** d'OpenCV, il existe plusieurs modes de chargement, les plus fréquents sont :

— **CV_LOAD_IMAGE_UNCHANGED**, l'image est chargée avec les caractéristiques données lors de l'enregistrement de cette image. C'est-à-dire, si cette image a une profondeur de 16 bits alors elle aura une profondeur de 16 bits sous OpenCV.

— **CV_LOAD_IMAGE_COLOR**, l'image est convertie vers une image de 24 bits par pixel, 3 canaux de 8 bits, un canal pour chaque couleur (rouge, vert et bleu). Pour les images en niveau de gris, pour un pixel x et y, les 3 canaux ont la même intensité que le pixel en x et y de l'image chargée. La valeur du canal en niveaux de gris est recopiée dans les canaux de chaque couleur.

— **CV_LOAD_IMAGE_GRAYSCALE**, l'image est convertie vers une image de 8 bits par pixel, avec un canal, le niveau de gris. Lorsque l'image source est en couleur, le niveau de gris résultant est la somme des canaux rouge, vert et bleu auxquels on a appliqué un coefficient, en général de $\frac{1}{3}$.

3.2 Transformations ponctuelles d'une image

Afin d'analyser les variations de gris de l'image, nous avons affiché l'histogramme de celle-ci via l'outil Gimp.

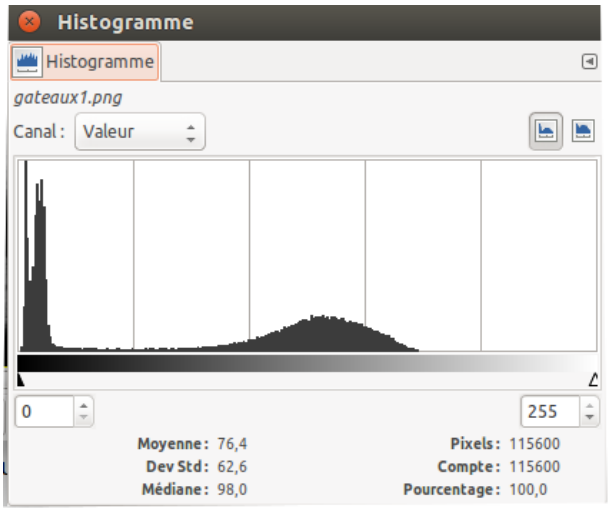

FIGURE 8 – Histogramme de l'image gateau1.png vu avec l'outil Gimp

Ce graphique montre qu'il y a de nombreux pixels proches du noir (au début de l'histogramme) qui correspondent au fond de l'image. Puis les pixels se concentrent dans une zone de gris qui représente les couleurs des gâteaux. À partir de ce graphique, on peut déterminer approximativement la dynamique de l'image. La dynamique est la différence entre la plus grande valeur de niveau de gris de l'image et la plus petite. Sur l'histogramme que nous étudions, on peut voir que la dynamique est approximativement de 177. Cette valeur ne correspond pas à la valeur que peut nous fournir la fonction **cvMinMaxLoc** qui est plus précise que ce que nous pouvons voir avec Gimp. Avec cette fonction, nous avons une dynamique de 201.

La binarisation va nous permettre de séparer clairement les gâteaux du fond. En utilisant l'outil Gimp, nous avons déterminé un seuil à 30 que l'on peut ensuite utiliser dans la fonction **cvThreshold**.

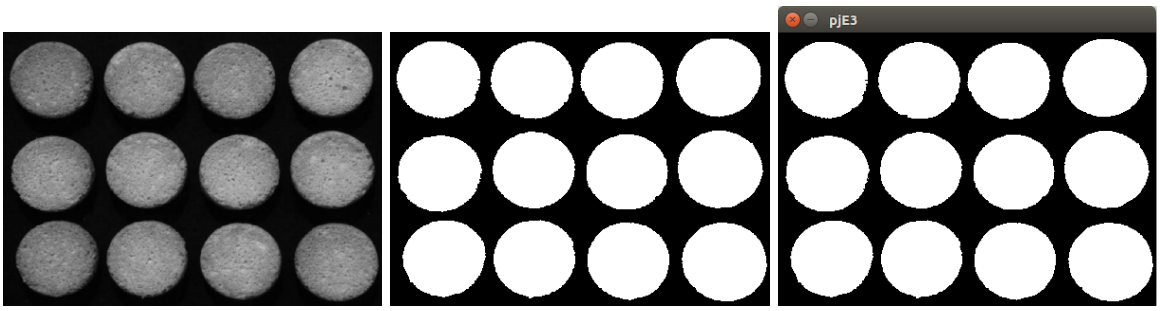

FIGURE 9 – Binarisation des gâteaux. À gauche, l'image source. Au centre, seuillage à 30 avec gimp. À droit, idem avec OpenCV.

La fonction **cvCmp** d'OpenCV permet de comparer deux images, les pixels identiques entre les deux images sont représentés en noir. Nous allons utiliser une fonction pour comparer le seuillage d'OpenCV et de Gimp. La première fois avec un seuil à 30 sur OpenCV et Gimp, voir l'image de gauche ci-dessous. On note quelques différences entre les deux fonctions du fait des inclusions et exclusions des pixels qui ont la valeur du seuil. La seconde fois, la comparaison est effectuée avec un seuil à 30 sous OpenCV et un seuil à 60 (deux fois plus grand) avec Gimp, voir l'image de droite ci-dessous. La différence se trouve sur les bords des gâteaux. En effet, avec un seuil à 60, les bords des gateaux, qui sont plus sombre, sont considérés comme faisant partie du fond.

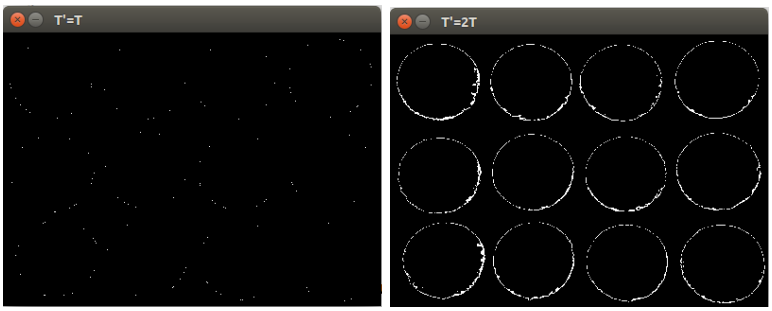

FIGURE 10 – Comparaison des seuillages avec OpenCV et Gimp

Ensuite, l'utilisation de **cvThreshold** avec les flags **CV_THRESH_BINARY** et **CV_THRESH_OTSU**, permet de binariser une image grâce à la méthode d'Otsu. Cette méthode détermine le seuil en prenant compte de l'histogramme. Ceci peut être pratique, mais ne donne pas toujours un meilleur résultat qu'avec un choix de seuil "à la main".

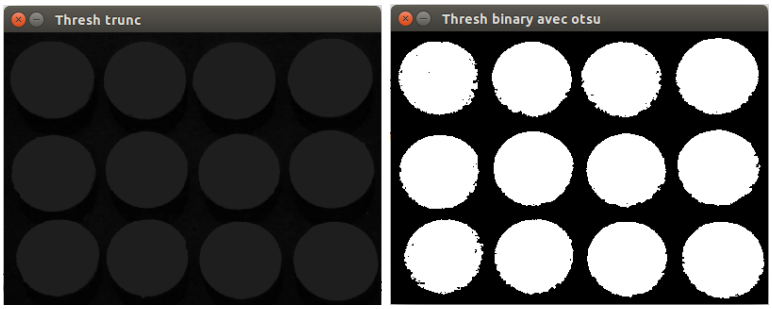

FIGURE 11 – Autres méthodes de seuillage, CV_TRUNC à gauche, CV_BINARY avec Otsu à droite

3.3 Parcours d'une image

Avec OpenCV, les données d'une image sont changées dans la mémoire. L'adresse de début des données de l'image se trouve dans le pointeur d'octets **imageData** de la structure **IplImage**. C'est un vecteur unidimensionnel de valeurs des pixels. Les valeurs sont chargées de manière à ce que les lignes de l'image se suivent.

Exemple avec une image de de 3x3 :

— Valeurs des pixels dans l'image :

10	24	156
13	47	184
28	95	230

— Valeurs des pixels dans imageData :

10	24	156	13	47	184	28	95	230

Pour récupérer les valeurs de chaque pixel de l'image et appliquer un seuillage, il faut parcourir l'image de la manière suivante :

```
1 void seuiller(IplImage* imageSrc, IplImage* imageDst, uchar seuil){
2   int row, col;
3   uchar niveau;
4
5   /* verification du format des images source et destination */
6   if(imageSrc->imageSize!=imageDst->imageSize ||
```

```
7       imageSrc->depth!=8 || imageDst->depth!=8 ||
8       imageSrc->nChannels!=1 || imageDst->nChannels!=1){
9         imageDst = NULL;
10        return;
11    }
12    /* parcours des pixels de l'image source */
13    for(col=0; col<imageSrc->height; col++){
14      for(row=0; row<imageSrc->width; row++){
15        niveau = (uchar)(imageSrc->imageData[ col*imageSrc->widthStep +
      row ]);
16
17        /* application du seuil */
18        if(niveau>seuil){
19          imageDst->imageData[ col*imageDst->widthStep + row ] = 255;
20        } else {
21          imageDst->imageData[ col*imageDst->widthStep + row ] = 0;
22        }
23      }
24    }
25  }
```

Listing 2 – Fonction seuiller avec OpenCV

Le parcours d'une image couleur se fait un peu différemment du parcours d'une image en niveaux de gris. En effet, la valeur des trois canaux se trouve à la suite dans le pointeur **imageData** d'**IplImage**. Il faut donc avoir un pas de trois octets pour aller au pixel suivant. Les canaux sont rangés, par défaut, avec le bleu en premier puis le vert puis le rouge.

```
1   void getCanal(IplImage* imageSrc, IplImage* imageDst, uchar canal){
2     int row, col;
3
4     // la position dans le tableau imageData
5     // la position se caracterise par une ligne, une colonne et un canal
6     int position;
7
8     if(imageSrc->imageSize!=imageDst->imageSize ||
9        imageSrc->depth!=8 || imageDst->depth!=8 ||
10       imageSrc->nChannels!=3 || imageDst->nChannels!=3)
11    {
12      imageDst = NULL;
13      return;
14    }
15    cvSet(imageDst, CV_RGB(0,0,0));
16
17    for(col=0; col<imageSrc->height; col++){
18      // Le pas est de 3 octets
19      // La largeur de l'image est bien de imageSrc->width pixels,
20      // mais il y a 3 fois plus d'octets (les 3 canaux)
21      for(row=0; row<imageSrc->width*3; row=row+3){
22        position = col*imageSrc->widthStep + row + canal;
23        imageDst->imageData[position] = imageSrc->imageData[position];
24      }
25    }
26  }
```

Listing 3 – Fonction getCanal avec OpenCV

4 Filtrage et compléments sous OpenCV

4.1 Etude de quelques filtres usuels

4.1.1 Filtre moyenneur

Un filtre effectue une opération mathématique, appelé convolution, sur un ensemble de données en entrée afin de former une image de sortie différente. Nous avons appliqué un premier filtre linéaire sur l'image phare.png dont le masque est $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$.

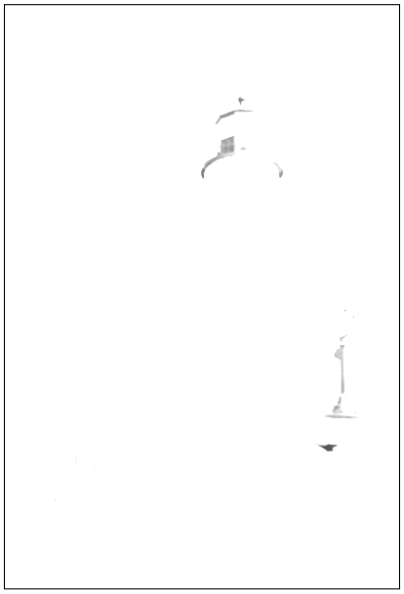

FIGURE 12 – Résultat du premier filtre appliqué sur l'image phare.png

On peut voir que le résutlat est une image presque totalement blanche. En effet, chaque pixel aura comme valeur la somme des huit pixels autour de lui. Presque tous les pixels ont alors une valeur supérieure à 255.

La normalisation fait en sorte que la somme des valeurs de la matrice fournie soit égale à 1. Nous avons appliqué une normalisation sur la matrice précédente, ce qui nous donne une image floutée par rapport au phare.png. Ce flou est encore plus important lorsque la normalisation se fait sur une matrice de plus grande taille.

FIGURE 13 – Résultat de la normalisation avec une matrice 5x5 sur l'image phare.png

4.1.2 Filtre médian

Le filtre médian consiste à prendre la valeur médiane parmi l'ensemble des huit valeurs autour du pixel traité. Cela permet, entre autres, d'améliorer la qualité d'une image comportant un bruit assez important. Nous avons testé ce filtre sur une image avec un bruit de type "poivre et sel".

FIGURE 14 – Image avant et après l'application du filtre médian

Lorsque nous appliquons le filtre moyenneur sur l'image contenant le bruit "poivre et sel", nous pouvons constater que l'image est floutée. Cependant, elle ne contient plus de pixel dont la valeur est très différente de celles des autres pixels qui l'entourent, contrairement au filtre médian.

4.1.3 Filtre dérivateur

Nous avons testé ce type de filtre sur une image contrastée sans bruit avec la matrice suivante : $1/2 \begin{pmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$. Ce filtre a pour effet de rendre l'image noire avec des traces blanches correspondant aux endroits où il y a une grosse variation de couleur, soulignant ainsi certains contours.

FIGURE 15 – Image avant et après l'application du premier filtre dérivateur

Pour obtenir une image qui souligne l'ensemble des contours il nous faut utiliser la matrice suivante : $1/4 \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$. Cependant, ce filtre va souligner les contours même des plus petits objets comme l'herbe.

Alors que la matrice $1/8 \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$ permet de souligner les contours des objets les plus important.

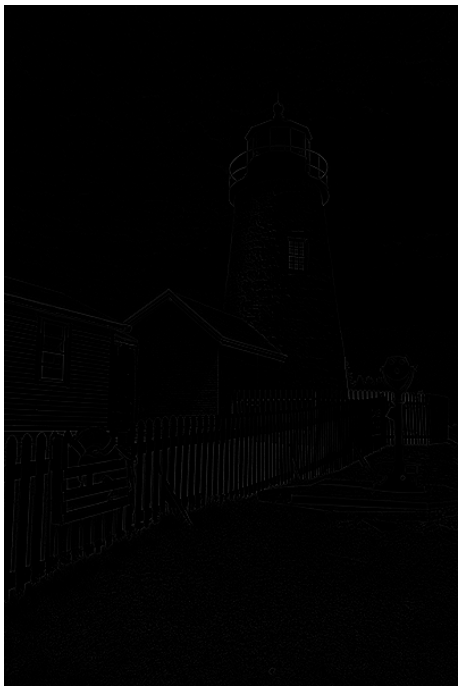

FIGURE 16 – Résultat de la matrice soulignant les contours des objets les plus importants

4.2 Définition et représentation d'histogrammes

4.2.1 Exemple d'histogramme 1D avec l'image du phare

Afin de récupérer davantage d'information sur l'image phare.png, nous avons affiché l'histogramme de l'image à l'aide d'OpenCV.

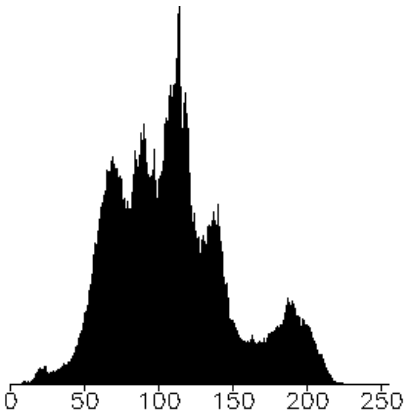

FIGURE 17 – Histogramme de l'image phare.png

Les deux premiers pics correspondent au gris présent dans les toits et l'herbe. Le pic le plus important de l'histogramme représente le ciel qui est en gris et représente une grande partie de l'image. Et le dernier pic aux environs de 200 correspond au phare qui est majoritairement blanc.

4.2.2 Utilisation de l'histogramme sur une image prise par l'interface multitouch

Nous avons maintenant récupéré l'histogramme d'une image prise avec l'interface multitouch afin de déterminer quelle méthode de binarisation est la plus approriée pour notre application.

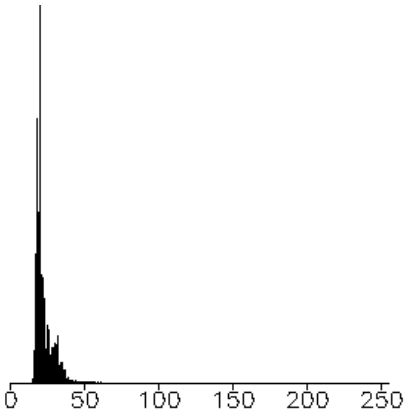

FIGURE 18 – Histogramme d'une image de la surface multitouch

En analysant l'histogramme de cette image on peut voir que l'on ne peut pas déterminer un seuil de binarisation car le fond est entièrement noir et il est impossible de distinguer les niveaux de gris des doigts. Il faudrait donc récupérer l'histogramme d'une ROI [5] restreinte aux alentours d'un doigt de l'image pour pouvoir déterminer un seuil.

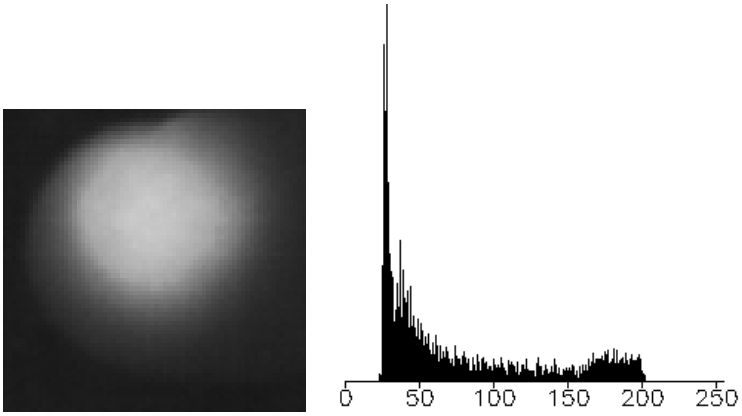

FIGURE 19 – Image et histogramme de la ROI d'un doigt

On peut voir un pic en debut de l'histogramme qui correspond aux pixels du fond. Mais maintenant l'histogramme fait apparaitre les pixels du doigt entre les niveaux de gris 160 et 200 (et du bruit entre les deux zones). En utilisant 160 en seuil de binarisation avec la méthode **cvThreshold** nous obtenons une image binarisée permettant de bien distinguer les doigts dans l'image.

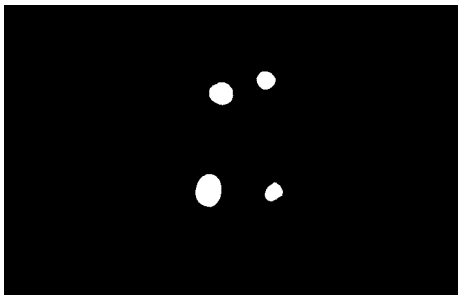

FIGURE 20 – Binarisation d'une image provenant de l'interface multitouch

5. Region Of Interest

En applicant l'option THRESH_OTSU sur l'image des doigts on obtient une binarisation très satisfesante que l'on considère l'image entière ou bien seulement une région restreinte. Les résultats sont très proches. On peut conclure que la méthode d'OTSU est un algorithme qui convient à nos besoins et est moins laborieuse qu'une méthode empirique "à la main".

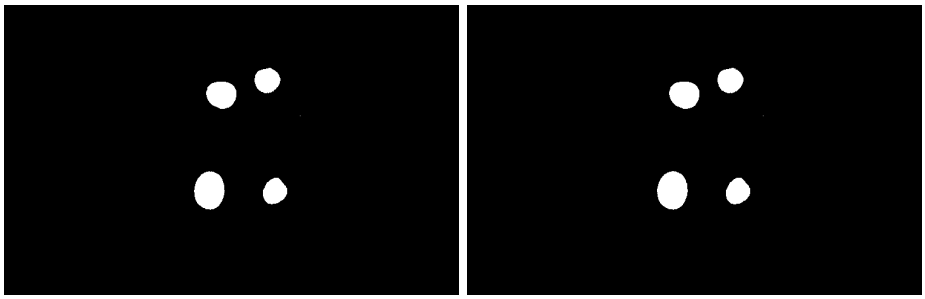

FIGURE 21 – Comparaison de la binarisation avec la méthode d'OTSU en considérant une ROI restreinte (à gauche) et l'image entière (à droite)

5 Détection d'objet

5.1 Blobs d'une image de synthèse

Nous créons une image binaire de synthèse afin d'en retirer des informations sur les blobs de cette image grâce à l'objet CBlobResult.

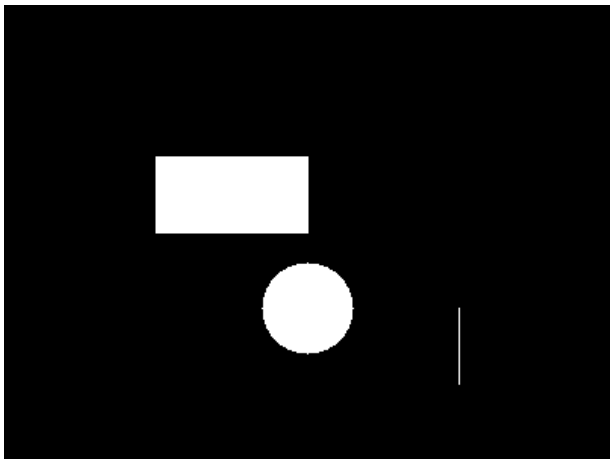

FIGURE 22 – Image synthétique utilisée pour la detection de blobs

Nous pouvons voir, lorsque nous affichons les informations de l'image, que la lecture de celle-ci se fait de haut en bas et de gauche à droite. Les pixels de l'image sont des pixels carrés et un pixel constitue l'unité atomique d'espace de l'image. Par conséquent, les caractéristiques du cercle, telles qu'elles sont représentées dans le blob, ne correspondent pas aux caractéristiques théoriques du cercle qui a été créé dans l'image de synthèse en fournissant son centre et son rayon.

Il est possible avec l'objet CBlobResult de filtrer les blobs que l'on souhaite garder. Pour cela, il faut utiliser la méthode Filter qui prend en paramètre des conditions permettant de sélectionner les blobs souhaités. Nous avons par exemple retiré tous les blobs ayant une surface inférieur à 1000 pixels, ce qui a eu pour conséquence de supprimer le segment de l'objet CBlobResult.

5.2 Blobs issus de la détection des doigts

Nous allons maintenant voir quels sont les réglages optimaux pour obtenir une image binaire des doigts sur la vitre en plexiglas. Etant donné que nous travaillons avec des infrarouges, nous avons choisi d'utiliser le mode couleur IS_CM_SENSOR_RAW8. Nous definissons les autres paramètres avec l'outil uEyeDemo qui nous fournit un fichier d'initialisation à charger dans notre programme.

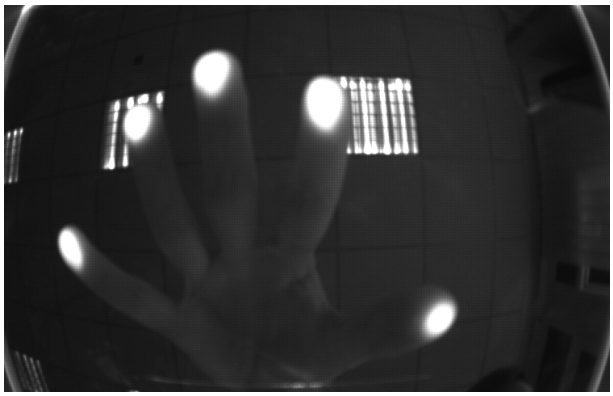

FIGURE 23 – Image de la caméra sans traitement

Pour binariser notre image, nous utilisons la méthode d'Otsu. Cependant, nous avons détecté un problème lorsque la luminosité du soleil est trop importante à l'acquisition de la première image : toutes les images suivantes sont presque totalement blanches. En effet, lorsque la luminosité du soleil est très présente dans l'image, le seuil calculé par cette méthode est faussé sur les images suivantes. Nous avons donc plutôt utilisé la méthode cvThreshold avec le paramètre CV_THRESH_BINARY et un seuil de 34.

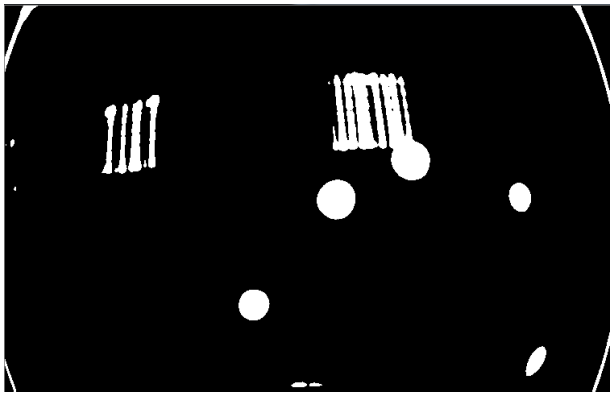

FIGURE 24 – Image de la caméra après binarisation

Maintenant que nous avons une image binarisée, nous devons garder seulement les doigts sur l'image. Pour cela, nous utilisons la méthode Filter de la classe CBlobResult pour supprimer les néons et autres objets inutiles. Nous avons utilisé le code suivant :

```
1 /* filtre sur l'aire des objets */
2 sNewBlobs.Filter(sNewBlobs, B_INCLUDE, CBlobGetArea(), B_INSIDE, 400,
    4750);
3 /* suppression des formes allonges */
4 sNewBlobs.Filter(sNewBlobs, B_INCLUDE, CBlobGetAxisRatio(), B_INSIDE,
    0.5, 1);
5 /* suppression des objets non compacte */
6 sNewBlobs.Filter(sNewBlobs, B_EXCLUDE, CBlobGetCompactness(), B_GREATER
    , 5.0);
```

Listing 4 – Filtrage des blobs via la fonction Filter

5.3 Etiquetage des blobs

Pour pouvoir identifier chaque doigt sur l'image, nous devons passer sur une image couleur via la fonction cvMerge. Puis, via un tableau de couleurs nous faisons correspondre une couleur à un numéro de blob du CBlobResult, ce qui nous donne le résultat suivant.

FIGURE 25 – Image de la caméra après indexage

Le problème avec cette méthode c'est qu'avec le sens de lecture des blobs, il est possible que lorsque nous effectuons une rotation avec deux doigts sur la plaque de plexiglas, les couleurs des doigts changent durant le mouvement. Puisque la première couleur est utilisée pour le premier blob, si l'image lit le deuxième doigt en premier alors sa couleur change, le sens de lecture étant de gauche à droite et de haut en bas.

6 Suivi des objets

Nous avons vu précédemment qu'étant donné que les blobs sont détectés de la gauche vers la droite et de haut en bas, lorsqu'on effectue une rotation de deux doigts, la détection des blobs s'inverse quand le doigt en dessous de l'autre passe au-dessus. Or, pour n'importe quelle application utilisant une interface tactile, il est indispensable de détecter et de suivre un doigt donné sans qu'il soit inversé ou confondu avec un autre. En somme, nous avons répondu à la question « comment détecter un input de type touch ? », mais pas à la question « comment effectuer le suivi de ces input ?».

6.1 Calcul des distances généralisées

La méthode employée pour effectuer un tel suivi consiste à comparer les blobs de deux images consécutives. De cette comparaison va résulter une matrice dont les lignes (resp. les colonnes) correspondent aux blobs détectés de la première image (resp. de l'image suivante). Les valeurs contenues dans cette matrice sont les distances relatives entre chaque blob de la première image et chaque blob de l'image suivante. Ensuite, en traitant ces informations d'une manière que nous verrons plus tard, il est possible d'attribuer à un ancien blob un des nouveaux blobs qui lui correspond le mieux, mais de détecter aussi l'apparition ou la disparition d'un blob.

Pour le calcul des distances, il vient naturellement à l'esprit d'utiliser une distance euclidienne. En effet, entre deux images, un doigt ne peut bouger que très peu et donc les blobs qui lui correspondent sur les deux images sont probablement très proches (à condition bien sûr d'avoir une cadence d'acquisition raisonnablement élevée). De plus, si un des nouveaux blobs se trouve loin de tous les anciens blobs, on peut en déduire qu'il s'agit d'une apparition d'un blob. Inversement, si un ancien blob se trouve loin de tous les nouveaux, il correspond probablement à un doigt qui n'est plus en contact avec la surface tactile. Cependant, il est possible d'améliorer encore le suivi des objets en faisant intervenir d'autres paramètres dans cette distance ; on obtient alors une distance généralisée. Comme autres paramètres, nous pouvons prendre par exemple la surface, l'orientation ou encore la forme. En utilisant une telle distance plus précise, on obtient de ce fait un meilleur suivi moins sujet aux confusions.

Cependant, dès lors que l'on fait intervenir plusieurs paramètres dans la distance, un problème se pose : comment s'assurer de l'équivalence entre l'impact des variations de chaque paramètre ? Prenons comme exemple une distance généralisée dans laquelle interviendraient la distance euclidienne et la surface que l'on additionnerait sans coefficients :

$$(d(blob1, blob2) = (positionblob1 - positionblob2) + (surfaceblob1 - surfaceblob2)$$

La distance étant exprimée en mètre et la surface en mètre carré, les variations de ces deux paramètres n'ont pas du tout le même poids. Une variation de surface (en termes d'unité) entraînerait une plus grande variation de la distance généralisée qu'une variation de la distance euclidienne ; c'est pourquoi nous prenons la racine carrée de la surface plutôt que la surface elle-même. Par ce procédé, les deux paramètres ont un impact uniforme sur la distance généralisée.

Revenons à notre matrice des distances généralisées dont les lignes représentent les blobs de l'image précédente et les colonnes les blobs de l'image courante. Une fois construite, trois cas se présentent :

— il y a plus d'anciens blobs que de nouveaux : on cherche la distance minimum pour chaque

nouveau blob (pour chaque colonne). En procédant ainsi, des anciens blobs (des lignes) ne seront déterminés proches d'aucun nouveau blob. On éliminera plus tard ces blobs.

$$\begin{pmatrix} 2 & 9 \\ 5 & 0.5 \\ 7 & 1 \end{pmatrix}$$

— Il y a plus de nouveaux blobs que d'anciens : inverse du premier cas

$$\begin{pmatrix} 2 & 9 & 3 \\ 5 & 0.5 & 7 \end{pmatrix}$$

— Il y a autant d'anciens que de nouveaux blobs : pour chaque ligne et pour chaque colonne, un minimum sera trouvé. A priori, à chaque ancien blob sera assigné un nouveau.

$$\begin{pmatrix} 2 & 9 & 34 \\ 57 & 124 & 29 \\ 7 & 3 & 41 \end{pmatrix}$$

Une fois que l'on a effectué ce premier traitement, on regarde les valeurs minimums trouvées et on élimine les valeurs supérieures à un seuil que l'on aura préalablement déterminé. En effet, si un couple ancien blob – nouveau blob a été trouvé, qui nous dit qu'il s'agit réellement du même objet ? Si leur distance est supérieure à ce seuil, c'est que l'on est face probablement à la disparition et à l'apparition simultanées de blobs qui n'ont rien à voir entre eux. La matrice résultante contient donc les valeurs retenues et des valeurs sentinelles lorsqu'une valeur a été écartée.

$$\begin{pmatrix} 2 & -1 & -1 \\ -1 & -1 & 29 \\ -1 & 3 & -1 \end{pmatrix} => \begin{pmatrix} 2 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & 3 & -1 \end{pmatrix}$$

6.2 Etiquetage des blobs

Arrivé à ce stade du traitement, il s'agit d'analyser les valeurs calculées et retenues pour assigner les étiquettes des anciens blobs aux nouveaux blobs correspondants et de traiter la disparition ou l'apparition éventuelles de blobs.

Que nous dit la matrice ? Si pour une ligne, aucune valeur n'a été retenue, c'est qu'aucun blob de l'image courante n'a été trouvé pour correspondre au blob de l'image précédente de cette ligne ; c'est donc qu'il a disparu. On supprime donc son étiquette. Ensuite, si pour une colonne, aucune valeur n'a été retenue, c'est qu'aucun blob de l'image précédente n'a été trouvé pour correspondre au blob de l'image courante de cette colonne ; c'est donc que ce blob vient d'apparaître. On lui attribue donc une nouvelle étiquette. Enfin, pour les valeurs retenues, on récupère le couple ligne – colonne et on attribue l'étiquette de l'ancien blob de cette ligne au nouveau blob de cette colonne. On a réussi à effectuer un suivi des objets détectés.

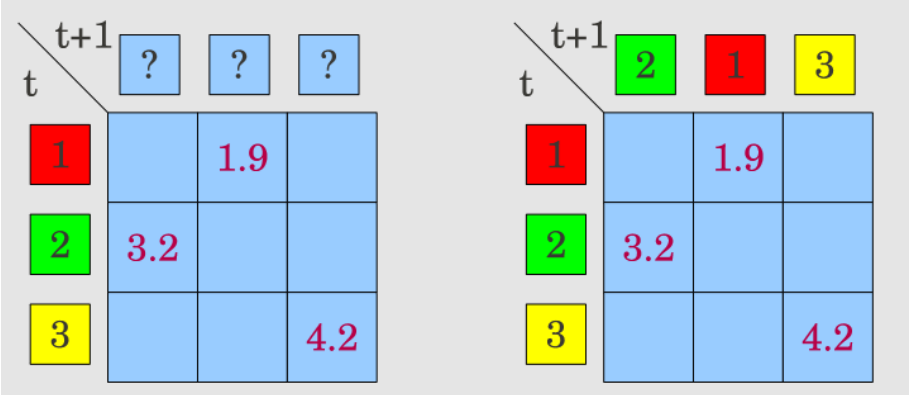

FIGURE 26 – Attribution des étiquettes

Il va de soi qu'un tel système n'est pas infaillible. Des confusions sont encore possibles, mais cela constitue une méthode très performante en matière de suivi d'objets. Il est possible, en ajoutant d'autres paramètres, d'obtenir un système plus minutieux dans l'attribution d'étiquettes. Cependant, il s'agit d'obtenir un compromis raisonnable entre robustesse et flexibilité.

6.3 Envoi de messages par TUIO

Un serveur TUIO fournit une interface permettant de gérer un suivi des doigts qui sont définis en tant que curseurs. A chaque image, l'application soumet le résultat du traitement vu précédemment au serveur qui se charge de garder en mémoire les curseurs avec des informations de position ou encore de datation. Cette interface permet d'envoyer des messages relatifs à chaque image qui informent de l'apparition, de la disparition et de la mise à jour de curseurs. Ce serveur peut ainsi notifier un client des événements impliquant un changement dans les contacts entre les objets et la surface tactile. L'application utilisant une interface multi-touch que nous allons programmer pour la suite de ce cours sera ce client et utilisera les informations du serveur TUIO afin de remplir les tâches qui lui incombent.

Conclusion

Cette première moitié de semestre dans ce projet fut passionnante pour nous trois. Il nous a été enseigné de nombreuses notions et nous avons appris à utiliser de nombreux outils qui vont nous permettre de réaliser l'interface multi-touch dans la seconde moitié de ce semestre.

En effet, nous avons d'abord acquis des connaissances théoriques en termes d'optique et leurs applications techniques dans le domaine de l'acquisition vidéo et du traitement d'image. Nous comprenons désormais comment fonctionne une caméra — comment fonctionnent les éléments qui la composent (capteur, objectif, diaphragme), quels paramètres entrent en jeu (cadence d'acquisition, horloge pixel, temps d'intégration), etc... Nous avons vu comment il était possible d'analyser et de traiter numériquement les images produites par cette caméra. Tous ces aspects interviennent directement dans l'élaboration d'une interface tactile multi-touch et cela constitue un bagage de connaissances et d'expérimentations indispensable pour la suite du projet que nous avons hâte de poursuivre.