Compte-rendu projet individuel 96 : les yeux nous trahissent ?

Auteurs : Elliot VANEGUE et Gaëtan DEFLANDRE
Encadrant : Marius BILASCO, Benjamin ALLAERT et José MENNESSON

12 mars 2015

Remerciement

Résumé

Abstract

Table des matières

Table des figures

Introduction

Durant nos études de master informatique à l'université de Lille 1, nous avons l'occasion de participer à un projet proposé par une équipe de recherche. Cette expérience a pour but de nous faire découvrir le milieu de la recherche.

Le but du projet est de rendre plus stable la détection des yeux dans une application qui permet de faire de la reconnaissance d'émotions à travers l'interprétation des mouvements et des expressions du visage. Ce type d'application peut être utilisé pour connaître l'intérêt d'une personne pour une publicité ou encore connaître l'attention d'un étudiant dans le cadre du e-learning. Ce genre d'outils existe déjà mais dans de nombreux cas, il utilise des appareils assez intrusifs pour l'étude des yeux.

Pour la réalisation de ce projet, nous travaillons avec l'équipe FOX qui étudie l'analyse du mouvement à partir de vidéos. Plus précisément, leurs recherches portent sur l'extraction du comportement humain depuis les flux vidéo. Leurs travaux sont divisés en quatre grands domaines : le regard, qui est la partie sur laquelle nous travaillons, l'événement, l'émotion et la reconnaissance de personnes. La grande majorité de leurs travaux sont des applications temps réel, ce qui permet d'avoir un niveau de réactivité très élevé. Le projet sur lequel nous avons travaillé est basé sur les travaux d'anciens étudiants qui se sont concentrés sur la détection de visage et des yeux afin d'extraire les émotions d'une personne. Le projet est une application temps réel, dont les flux vidéo peuvent provenir de vidéos enregistrées ou d'une caméra type webcam.

Au début du projet, la détection des yeux été implémentée, cependant, la méthode de détection est encore approximative et les points permettant de localiser les yeux peuvent parfois subir un léger décalage, surtout lorsque la personne ferme les yeux. Le problème étant que les algorithmes de reconnaissance d'émotions, écrits auparavant, se reposent sur cette détection approximative des yeux.

Pour corriger ce défaut, nous cherchons à extraire les yeux du reste du visage, afin de retrouver des points fixes, nous permettant de recentrer les points calculés auparavant. Ensuite, l'objectif est de stabiliser les points pris en compte pour la localisation du visage avec les informations que nous avons récupéré. Une fois que ces régions seront stabilisées, le reste de l'application normalisera le visage et cela permettra d'avoir des résultats beaucoup plus fiables lors de la reconnaissance d'émotions.

1 Application existante

1.1 Architecture

1.2 Reconnaissance du visage : Viola et Jones

L'application est divisée en deux parties. La première recherche le visage grâce à l'algorithme de Viola et Jones et la seconde recherche les yeux dans la région délimitée précédemment.

L'algorithme de Viola et Jones est une méthode qui a été créée pour la reconnaissance de visage dans une image. Cette méthode s'est par la suite généralisée à toutes sortes d'objets. L'algorithme nécessite une base de connaissances composée des caractèristiques de l'objet recherché. L'algorithme de Viola et Jones nécessite un apprentissage supervisé, c'est à dire que l'algorithme a besoin de données représentant l'objet à détecter pour classifier les caractèristiques de celui-ci.

Cette algorithme est basé sur des caractèristiques pseudo-Haar qui crée des masques rectangulaires et adjacentes dans différente zone de l'image. Chaque masque calcule l'intensité des pixels qu'il contient, puis l'algorithme fait la différence entre les masque blanc et les masque noir. Cette méthode va permettre de détecter des contours ou des changements de texture.

FIGURE 1 – Exemple de caractèristiques pseudo-Haar utilisé pour l'algorithme Viola et Jones

Pour améliorer les perfomance de leur algorithme, Viola et Jones utilise la méthode Adaboost. Son principe est de séléctionner les caractèristiques les plus performante pour la détection de l'objet.

1.3 Suivi des yeux

2 Recherche de solution

2.1 L'algorithme de Canny

2.1.1 Version de base

Le filtre de Canny a été créé en 1986 dans le but d'améliorer les résultat du filtre de Sobel. Le principe du filtre est d'utiliser deux filtres, un filtre haut et un filtre bas. L'algorithme commence par séléctionner les pixels supérieur au seuil haut, puis recherche à partir de chaque pixels au dessus du seuil haut les pixels qui sont au dessus du seuil bas. Ainsi on voit que cet algorithme prend en compte deux caractèristiques, l'intensité et la direction des gradients.

2.1.2 Avec égalisation d'histogramme

2.1.3 Avec une moyenne de pixels sur des parties d'image

2.1.4 Avec une médiane sur les valeurs de gris des parties d'image

2.2 L'algorithme de Gabor

3 Implémentation de la solution

Conclusion

Annexes

```
void CannyThreshold(const Mat eyePicture)
{
    //voisinage pris en compte pour l'ouverture
    const Mat element = getStructuringElement(MORPH_RECT, Size(3,3));

    //nombre de division dans l'image
    const unsigned div = 4;

    //taille des partie de l'image
    const unsigned divCol = eyePicture.cols / div;
    const unsigned divRow = eyePicture.rows / div;

    for(unsigned i=0; i<div; i++) {
        for(unsigned j=0; j<div; j++) {

            //r cup ration d'une partie de l'image
            const CvRect part = cvRect(j * divCol, i * divRow, divCol,
    divRow);
            Mat partImage(eyePicture, part);

            //calcul de la moyenne de la partie de l'image
            unsigned moyenne = 0;

            for(unsigned k=0; k<divCol; k++) {
                for(unsigned l=0; l<divRow; l++) {
                    moyenne += partImage.at<uchar>(l,k);
                }
            }

            moyenne /= divCol * divRow;
            Canny( partImage, partImage, 0.5*moyenne, 0.7*moyenne, 3,
    true );

            //ouvertur de l'image pour supprimer les lignes verticales et
            //horizontal, ainsi qu'un peu de bruit
            dilate(partImage,partImage,element);
            erode(partImage,partImage,element);
        }
    }

}
```

Listing 1 – Application du filtre de Canny sur des parties de l'image avec la moyenne des niveaux de gris

```
 void normalize_channel (Mat channel)
{
    double min, max;
    unsigned col = channel.cols;
    unsigned row = channel.rows;

```

```
7       if(channel.channels() > 1) {
8           cerr << "erreur dans le nombre de canaux pour la normalisation d'
        histogramme" << endl;
9       }
10
11      minMaxLoc (channel, &min, &max);
12
13      double scale = 255.0 / (max - min);
14
15      for(unsigned i=0; i<col; i++) {
16          for(unsigned j=0; j<row; j++) {
17              channel.at<uchar>(j,i) -= min;
18              channel.at<uchar>(j,i) *= scale;
19          }
20      }
21
22  }
```

Listing 2 – Normalisation d'une chaine d'une image

Références