Introduction à la vision par ordinateur
ISIA 2015-2016
TP 1 : Détection de visages

February 24, 2016

1 Avant propos

Les travaux pratiques du cours de Vision par ordinateur consisteront en des mini-projets liés aux différentes notions vues en cours. Pour cela, vous utiliserez principalement la bibliothèque **OpenCV**[1] sous l'environnement de votre choix (préférence: linux). Pour chaque TP, vous devrez remettre un rapport et votre code. Le rapport expliquera votre démarche et présentera les différents résultats.

Il est fortement conseillé de faciliter la clarté du code, notamment en le structurant correctement, en incluant des commentaires appropriés et en choisissant des noms de variable qui fassent sens. Par exemple, une variable contenant une image peut être nommée `image`, `image_in`, `img` ou éventuellement `I` mais plus improbablement `i` (indice), `aa`, `toto`, `mat`...

OpenCV a des interfaces C++, C, Python and Java et vous pouvez donc choisir l'interface qui vous convient le mieux (préférence: python ou C++).

2 Installation et prise en main de OpenCV

Pour commencer, vous devez installer la bibliothèque OpenCV sur votre ordinateur. De nombreuses instructions sont disponibles pour l'installation sur le site officiel.

[1] `http://opencv.org/`

- C++ : `http://opencv.org/`, rubrique Documentation - Tutorials ou Quick Start.
- Python : `https://opencv-python-tutroals.readthedocs.org/en/latest/`

Pour les utilisateurs de mac, homebrew peut aussi être utiliser pour l'installation[2].

A l'issue de l'installation, vous devez réussir à compiler et à exécuter un programme consistant à ouvrir une image, à l'afficher et à fermer la fenêtre d'affichage. Une image est représentée comme une matrice dans OpenCV : structure `Mat`. Une explication assez détaillée de la structure de données est disponible dans le tutorial[3].

```
// g++ compile with:
// OpenCV 2.4.9  -lopencv_highgui -lopencv_core
// OpenCV 3.0.0  -lopencv_highgui -lopencv_core -lopencv_imgcodecs
#include <opencv2/highgui/highgui_c.h>
#include <opencv2/highgui/highgui.hpp>

void display(const cv::Mat & img, const std::string message)
{
    if (img.data) {
        cv::namedWindow( message.c_str(), CV_WINDOW_AUTOSIZE );
        cv::imshow(message.c_str(), img);
        cv::waitKey(0);
    }
}

int main(int argc, char** argv)
{
  cv::Mat img = cv::imread(argv[1]);
  display(img, "Resulting image...");
  return 0;
}
```

[2] `https://jjyap.wordpress.com/2014/05/24/installing-opencv-2-4-9-on-mac-osx-with-python-suppor` ou `http://www.pyimagesearch.com/2015/06/29/install-opencv-3-0-and-python-3-4-on-osx/`

[3] `http://docs.opencv.org/2.4/doc/tutorials/core/mat_the_basic_image_container/mat_the_basic_image_container.html#matthebasicimagecontainer`

```
import numpy as np
from matplotlib import pyplot as plt
import cv2


def loadImage(src):
    img=cv2.imread(src,1)
    cv2.imshow('image',img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

if __name__ == '__main__':
    loadImge('oscar.jpg')
```

Voici les liens vers les tutoriels concernant la lecture, l'affichage et la sauvegarde d'une image :

- C++ : `http://docs.opencv.org/doc/tutorials/introduction/display_image/display_image.html#display-image` ou `http://docs.opencv.org/doc/tutorials/introduction/load_save_image/load_save_image.html#load-save-image`
- Python : `http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_gui/py_image_display/py_image_display.html#display-image`

D'autres exemples sont donnés en annexe de ce document.

Documentation et aide pour OpenCV

- Sites officiels :
  - C++ : `http://opencv.org/documentation.html`
- Tutoriaux :
  - Global : `http://docs.opencv.org/2.4/doc/tutorials/tutorials.html`
  - Python : `http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_tutorials.html`
  - ...

3 Detection de visages

L'objectif de ce projet est de développer et de programmer des algorithmes permettant de détecter des visages dans des images. La *détection* Pour cela, on testera plusieurs approches :

- Une approche à partir de la detection de la peau.
- Une approche à l'aide de l'algorithme de Viola Jones.
- Une approche à partir d'algorithmes de segmentation en regions (optionnel).

4 Approche 1 : Detection de la peau

L'objectif est de detecter des visages par la détection de la peau humaine dans des images. Un premier travail est donc de chercher et d'étudier rapidement quelques articles scientifiques traitant de ce sujet et disponibles sur Claroline Par exemple :

- *A Survey on Pixel-Based Skin Color Detection Techniques*, Vezhnevets V., Sazonov V., Andreeva A, IN PROC. GRAPHICON-2003, 2003 (disponible sur Claroline).
- *Statistical Color Models with Application to Skin Detection*, Michael J. Jones, James M. Rehg, In Int. J. Comput. Vision 46, 1 (January 2002), 81-96 (disponible sur Claroline).
- *A survey of skin-color modeling and detection methods*, P. Kakumanu, S. Makrogiannis, N. Bourbakis. In Pattern Recogn. 40, 3 (March 2007), 1106-1122 (disponible sur Claroline).

4.1 Avant de commencer

1. Que pensez vous de ce type d'approches, que l'on pourra qualifier d'approches colorimétriques, pour la detection de visages ? La détection de la peau est-elle suffisante ? Quelles en sont les limites ?
2. Dans la mise en œuvre de ce type d'approches, quelles sont les principales questions à se poser ?

4.2 A vous de jouer !

L'article suivant : *A Survey on Pixel-Based Skin Color Detection Techniques*, [Vezhnevets V., Sazonov V., Andreeva A], IN PROC. GRAPHICON-2003, 2003, disponible sur Claroline, listant et décrivant les approches connues servira de base pour le développement de votre application de detection de la peau. Le principe général va consister en la construction d'une base d'exemples de *peau* et d'une base d'exemples de *non-peau* qui vont nous permettre d'apprendre la couleur de la peau et d'en construire un modèle. Comme mentionné dans l'article, plusieurs questions se posent :

- Le choix de l'espace de représentation de la couleur, notamment pour garantir une bonne séparation entre la peau et la non peau.
- La manière de construire le modèle de peau (non-paramétrique ou paramétrique).

4.2.1 Base d'images

Un premier travail est de construire une base d' images contenant des personnes (mais pas seulement), afin d'avoir des échantillons contenant de la peau et d'autres choses (non-peau). Pour cela, vous pouvez utiliser des jeux de données existants comme la **UCD Colour Face Image Database** disponible sur Claroline ou encore le Pratheepan Dataset (`http://web.fsktm.um.edu.my/~cschan/downloads_skin_dataset.html`). Choisissez des images modèles pour construire votre modèle de la peau. Le choix d'images représentatives pour la peau mais aussi pour la non-peau est important ici car il influencera la qualité de vos résultats. Vous pouvez par exemple sélectionner les images segmentées (visages ou autres) de la base de données UCD ou de l'autre base pour la peau. Vous n'avez pas besoin de prendre tous les pixels des images. Prenez des échantillons d'images. Par contre, faites attention à la qualité de vos échantillons. Prenez des exemples de peau dans différentes conditions (différentes personnes, différents éclairages, ...). La quantité et la qualité de vos exemples ici influencera directement la qualité de vos résultats finaux.

4.2.2 Une première méthode simple

La première méthode décrite dans l'article de Vezhnevets *et al.* consiste à utiliser une règle de classification simple. **Un pixel de composante (RGB)**

est classifié comme pixel de peau si :

- $R > 95$
- $G > 40$
- $B > 20$
- $\max\{R, G, B\} - \min\{R, G, B\} > 15$
- $|R - G| > 15$
- $R > G$
- $R > B$

Mettre en œuvre cette méthode et tester la sur quelques images de votre base, des images de peau et des images de non-peau. Commenter vos résultats.

La mise en œuvre de cette méthode vous permettra de prendre en main les principales fonctions de manipulations d'images (parcours d'image, accès à un pixel, cueillage). Par défaut, les images couleur sont en BGR et la première composante d'un pixel correspond donc au canal bleu et non rouge.

Documentation utile :

- Python :
  - `http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_core/py_basic_ops/py_basic_ops.html#basic-ops`
  - `http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html#thresholding`
- C++
  - `http://docs.opencv.org/doc/tutorials/core/how_to_scan_images/how_to_scan_images.html#howtoscanimagesopencv`
  - `http://docs.opencv.org/doc/tutorials/imgproc/threshold/threshold.html#basic-threshold`

4.2.3 Approche non paramétrique : Histogramme et modèle de peau

Plusieurs espaces couleurs sont possibles pour détecter les zones de peau dans des images. Dans ce TP, un objectif est de comparer entre différents types d'espaces couleur. Vous comparerez notamment les espaces RGB, HSV et Lab mais vous pouvez aussi en tester et en comparer d'autres.

Conversion d'espaces couleurs Un premier travail est d'écrire un programme permettant de transformer une image d'un espace couleur à un autre. Pour cela, il faudra faire appel à la fonction `cvtColor` d'OpenCV.

Construction d'histogrammes 2D Dans un premier temps, on pourra travailler avec l'espace Lab. Vous n'utiliserez que les composantes a et b (axes de couleur) et pas L (axe de luminance). Le travail consiste à construire deux **histogrammes 2D** de couleur, un histogramme *peau* et un histogramme *non-peau*. Il s'agit d'un tableau à 2 dimensions (a et b pour Lab, r et g pour rgb et h et s pour hsv). Pour chaque dimension on réduira l'échelle des valeurs de 256 à 32 (réduction de la quantification) et pour chaque case du tableau, il suffit de compter le nombre de pixels ayant comme valeur le couple (a,b) donné. Tous les pixels *peau* de toutes les images se retrouvent dans le même histogramme et réciproquement pour les pixels *non-peau*. Chaque histogramme sera aussi normalisé en divisant chaque valeur par le nombre total de pixels. Vous pouvez pour cela faire appel à la fonction `calcHist` d'OpenCV[4].

Vous devez donc construire deux histogrammes pour chaque espace de couleur choisi. Ces histogrammes pourront dans la suite être considérés comme un modèle de peau et un modèle de non-peau. Ces histogrammes modèle HistogrammePeau(a,b) et HistogrammeNonPeau(a,b) seront inclus dans le rapport sous forme d'images (ou de graphiques).

Detection de la peau dans les images à partir des histogrammes Plusieurs approches sont possibles pour détecter la peau dans les images (c.f. articles fournis sur Claroline). Une approche simple consiste à calculer, pour un pixel p d'une image, sa probabilité de peau et de non-peau à partir de sa couleur c = (a,b)[5] :

- $p(peau|c) = p(c|peau) = HistoPeau(a, b)$
- $p(\neg peau|c) = p(c|\neg peau) = HistoNonPeau(a, b)$

[4] `http://docs.opencv.org/modules/imgproc/doc/histograms.html`, `http://docs.opencv.org/doc/tutorials/imgproc/histograms/histogram_calculation/histogram_calculation.html#histogram-calculation`, `https://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_imgproc/py_histograms/py_2d_histogram/py_2d_histogram.html#twod-histogram`

[5] rappel : seules deux composantes sont utilisées

La probabilité la plus élevée permet de décider si le pixel est un pixel de peau ou non.

Mettre en oeuvre et tester cette méthode.

Detection de la peau dans les images : méthode de Bayes Cette décision peut être raffinée en utilisant la méthode de Bayes :

$$p(peau|c) = \frac{p(c|peau)p(peau)}{p(c|peau)p(peau) + p(c|\neg peau)p(\neg peau)}$$

avec $p(c|peau)$ et $p(c|\neg peau)$ données par les histogrammes et $p(peau)$ et $p(\neg peau)$ sont les pourcentages de pixels peau et non-peau dans votre base d'apprentissage. Une règle de classification peut alors être de classifier les pixels comme pixels de peau si $p(peau|c) > \Theta$ ou Θ est un seuil à déterminer.

Mettre en oeuvre et tester cette méthode.

4.2.4 Evaluation

En utilisant la même base d'images que pour votre apprentissage, évaluez le pourcentage de pixels peau que vous réussissez à bien détecter. Lévaluation quantitative sera exprimée en terme de taux de bonne detection (TP: *true positive*) et de mauvaise detection (FP: *false positive*)). Est-ce que cela vous semble correct ?

Evaluez maintenant vos différentes approches sur une autre base pour laquelle vous avez la vérité terrain mais qui n'a pas été utilisée pour construire vos modèles de peau. Analysez vos résultats. En particulier, vous devrez montrer dans le rapport des images où la détection fonctionne bien et d'autres où cela fonctionne moins bien, en expliquant pourquoi selon vous.

4.2.5 Discussion

Comment peut-on améliorer les résultats que vous avez obtenus ? Donnez des idées. Que faudrait-il faire pour détecter les visages ? N'hésitez pas à suggérer et à tester de nouvelles idées ici. Plusieurs variantes améliorant les résultats sont possibles.

5 Detection de visages par l'approche de Viola Jones

La méthode de Viola Jones est une méthode de détection d'objets dans des images proposée par Paul Viola et Michael Jones en 2001 très largement utilisée pour la detection de visages. Il s'agit d'une méthode de detection par apprentissage d'un classifieur.

Le premier travail consiste à vous documenter sur cette approche à partir de :

- Wikipedia : Méthode de Viola et Jones.
- L'article fondateur disponible sur Claroline : Paul Viola and Michael J. Jones. 2004. Robust Real-Time Face Detection. Int. J. Comput. Vision 57, 2 (May 2004), 137-154

5.1 Etude de la méthode

Une implémentation de cette approche est disponible dans OpenCV[6]. Le travail consistera donc en :

1. La reprise de cette implémentation et son utilisation pour la detection de visages sur une des bases de visages fournies. Vous pourrez pour ce travail utiliser les classifieurs pré-entraînés fournis dans openCV (opencv/data/haarcascades/).
2. Son evaluation qualitative et quantitative.

5.2 Discussion: pour aller plus loin!

Faire un rapide état des lieux concernant la detection de visages, notamment en vous intéressant aux approches actuelles. Que sait-on faire actuellement et avec quelles types d'approches ?

[6] `http://docs.opencv.org/master/d7/d8b/tutorial_py_face_detection.html#gsc.tab=0`

6 Detection de visages par des algorithmes de segmentation (OPTIONNEL)

Cette partie du projet ne devra être traitée qu'après le cours sur la segmentation. L'objectif ici est juste d'appliquer certains des algorithmes vus en cours au problème de la detection de visage et d'étudier leur comportement. En particulier vous pourrez tester l'algorithme *meanshift* et l'algorithme *grabcut* dont des implémentations sont disponibles dans OpenCV. Il s'agit ici principalement d'étudier et de reprendre le code existant et de l'adapter à la problématique de la detection de visages. Comme pour l'approche précédente, une evaluation qualitative et quantitative de la detection devra être réalisée.

7 Annexe: tutorial openCV en C++

Cette section contient des exemples de manipulation des images avec OpenCV en C++

7.1 Accès à un pixel

Deux méthodes sont données ci-dessous pour accéder à un pixel donné d'une image. Pour plus de détails, consultez la documentation à `http://docs.opencv.org/2.4/doc/tutorials/core/how_to_scan_images/how_to_scan_images.html#howtoscanimagesopencv`.

```
// g++ compile with:
// OpenCV 2.4.9  -lopencv_highgui -lopencv_core
// OpenCV 3.0.0  -lopencv_highgui -lopencv_core -lopencv_imgcodecs

#include <iostream>
#include <opencv2/highgui/highgui.hpp>

using std::cout;
using std::endl;

int main(int argc, char **argv)
{
  cv::Mat A = cv::imread(argv[1]);
```

```
  // verifie que le 'type' est bien 'char'
  CV_Assert(A.depth() == CV_8U);

  // verifie que c'est une image couleur
  CV_Assert(A.channels() == 3);

  // Affichage taille de l'image
  int h = A.rows;
  int l = A.cols;
  cout << "-- image de taille " << h << "x" << l << endl;

  // Affichage des valeurs (b,g,r) du pixel (lig,col) = (34,122)
  unsigned int y= 34;
  unsigned int x= 122;

  // methode efficace (pour scanner tous les pixels)
  // NB: le cast en int est pour l'affichage uniquement
  uchar* ptr =  A.ptr<uchar>(y);
  cout << "-- b=" << (int)(ptr[3*x+0])
       << " g=" << (int)(ptr[3*x+1])
       << " r=" << (int)(ptr[3*x+2]) << endl;

  // methode plus lisible
  cout << "-- b=" << (int)(A.at<cv::Vec3b>(y, x)[0])
       << " g=" << (int)(A.at<cv::Vec3b>(y, x)[1])
       << " r=" << (int)(A.at<cv::Vec3b>(y, x)[2]) << endl;

  return 0;
}
```