Charles CAGNON
Nathan HUE

mardi 6 juin 2017

Projet Real Time Blurring

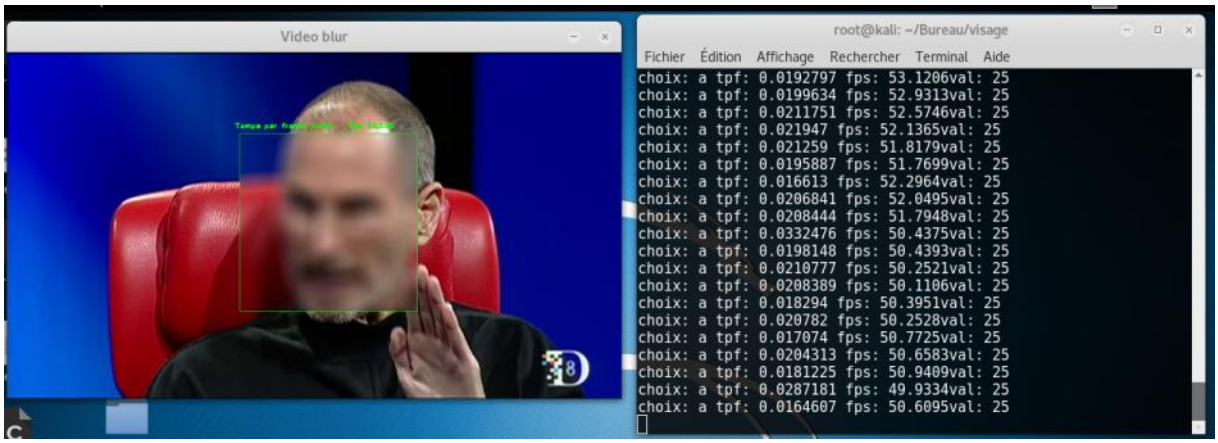
Les objectifs

Le but de ce projet est de pouvoir réaliser la méthode du blurring à partir d'un fichier video contenant des visages. Nous devrons utiliser les librairies python "openCV" et "Numpy", dans un premier temps afin de réaliser les différentes méthodes proposées par OpenCv, et enfin si nous avons le courage basculer plutot sur un programme codé en C++ pour des soucis de performances et d'integration dans des applications tierces telles qu'un environnement graphique cross platform Qt.

Notre cheminement

Au début nous voulions réaliser un deblurring en temps réel sur des vidéos trouvées sur internet. Nous avons testé différents logiciels de deblurring tels que "Smart deblur" et "Photoshop" pour nous rendre compte des méthodes de fonctionnement. Finalement les résultats obtenus étaient légèrement décevant.

Nous avons donc opté pour un autre projet. Cette fois ci du blurring en temps réel.

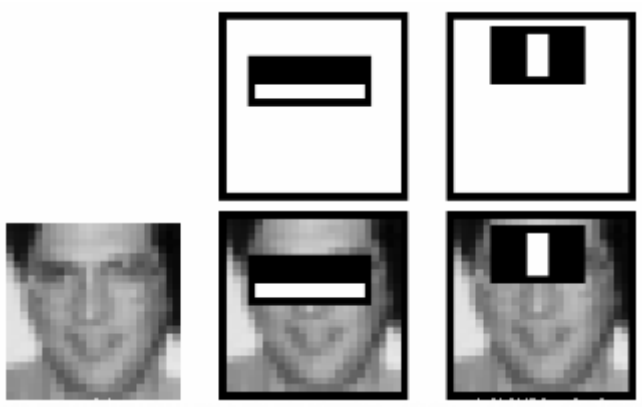


Charles CAGNON
Nathan HUE
mardi 6 juin 2017

Le principe de la détection de visages

La technique la plus simple consiste a vérifier chaque pixel de l'image et déduire une concordance avec un motif de type visage.
Cela fait énormément de calculs, ne serai-ce que pour une image de 24x24 pixels on atteindrai l'équivalant de 160000 opérations.
Pour des raison pratiques, deux ingénieurs "Paul Viola" et "Michael Jones" ont mit au point une stratégie de détection de zones.
Ainsi par exemple pour un visage on vient vérifier la luminosité de différentes zones logiques, les yeux apparaissent de manière foncés a coté du front et du nez et les cotés apparaissent plus clair que le nez par exemple.

Voici ce que cela donne en image.



Ce principe de reconnaissance faciale a donnée naissance a de nombreux générateur de fichiers cascades prenant en comptes les positions les plus probables de luminosités forte et de luminosité faible que l'on rencontre sur un visage. Il est même possible de créer son propre fichier cascade afin d'être sur d'être reconnu en un rien de temps.

```
<opencv_storage>
<haarcascade_frontalface_alt type_id="opencv-haar-classifier">
  <size>20 20</size>
  <stages>
    <_>
      <!-- stage 0 -->
      <trees>
        <_>
          <!-- tree 0 -->
          <_>
            <!-- root node -->
            <feature>
              <rects>
                <_>3 7 14 4 -1.</_>
                <_>3 9 14 2 2.</_></rects>
              <tilted>0</tilted></feature>
            <threshold>4.0141958743333817e-003</threshold>
            <left_val>0.0337941907346249</left_val>
            <right_val>0.8378106951713562</right_val></_></_>
        <_>
          <!-- tree 1 -->
```

Charles CAGNON
Nathan HUE

mardi 6 juin 2017

Les grand principes du bluring

L'API Opencv nous permet d'ajuster différents paramètres afin d'obtenir un blurring personnalisé. Voici une brève présentation des quelque types que nous avons utilisés pour notre projet.

Le principe de Bluring homogène ou Lissage normalisé

Le principe du bluring homogène reprend le principe de convolution 2D et consiste à recouper des images avec différents filtres passe bas, puis de soustraire les filtres de convolution personnalisés aux images.

- Ce filtre est le plus simple de tous! Chaque pixel de sortie est la moyenne de ses voisins du noyau (tous contribuent avec des poids égaux)
- Le noyau est présenté ci-dessous:

$$K = \frac{1}{K_{width} \cdot K_{height}} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \\ . & . & . & \dots & 1 \\ . & . & . & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}$$

Du coté de la méthode voici ce que l'on peut trouver:

```
blur( src, dst, Size( i, i ), Point(-1,-1) );
```

- **src** – image source;
- **dst** – image de sortie.
- **ksize** – taille du noyau de blurring.
- **anchor** – point d'ancrage;
- **borderType** – Type de bordure.

Charles CAGNON
Nathan HUE

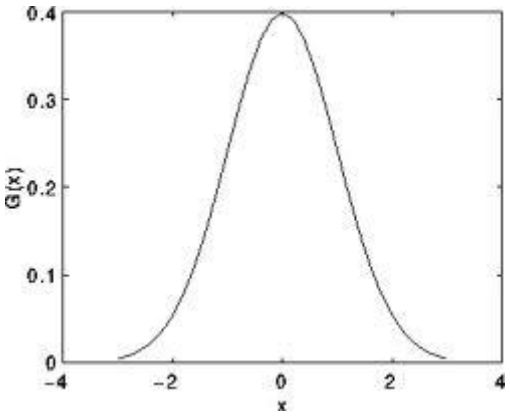
mardi 6 juin 2017

Le principe du Blur Gaussien

L'effet visuel de cette technique de flou est un flou lisse ressemblant à celui de la visualisation de l'image à travers un écran translucide. Mathématiquement, appliquer un flou gaussien à une image revient a faire une convolution de l'image avec une fonction Gaussienne .Elle est également connu comme une transformation bidimensionnelle de Weierstrass. L'application d'un flou gaussien a pour effet de réduire les composants haute fréquence de l'image. Un flou gaussien est donc un filtre passe-bas .

- Probablement le filtre le plus utile (mais pas le plus rapide). Le filtrage gaussien se fait en convertissant chaque point du tableau d'entrée avec un *noyau gaussien* et en les additionnant tous pour produire le tableau de sortie.

- Juste pour rendre l'image plus claire, n'oubliez pas comment ressemble un noyau Gaussian 1D?



En supposant qu'une image est 1D, on peut remarquer que le pixel situé au milieu aurait le plus grand poids. Le poids de ses voisins diminue à mesure que la distance spatiale entre eux et le pixel central augmente.

Du coté de la méthode voici ce que l'on peut trouver:

```
GaussianBlur( src, dst, Size( i, i ), 0, 0 );
```

- **src** – image source;
- **dst** – image de sortie.
- **ksize** – taille du noyau de blurring.
- **SigmaX -** écart type du noyau gaussien en direction X.
- **SigmaY -** écart type du noyau gaussien dans la direction Y
- **borderType** – Type de bordure.

Charles CAGNON
Nathan HUE
mardi 6 juin 2017

Le principe du blur Median

Le filtre médian traverse chaque élément de l'image et remplace chaque pixel par la médiane de ses pixels voisins (situé dans un quartier carré autour du pixel évalué).

Du coté de la méthode voici ce que l'on peut trouver:

```
medianBlur ( src, dst, i );
```

- **Src :** image source
- **Dst :** l'image de destination, doit être du même type que src
- **I :** Taille du noyau (un seul parce que nous utilisons une fenêtre carrée). Doit être impair.

Le principe du radius smooth

Le principe du radius smooth consite a dupliquer de manière multiple une image puis de les faire tourner par rapport a son image de base. La superposition des images permet de donner un effet de flou. Malheureusement nous n'avons pas eu le temps de mettre en place cette solution, c'est pourquoi peu d'information sont explicitées dans cette partie.

Le principe bi-lateral

Jusqu'à présent, nous avons expliqué certains filtres dont l'objectif principal est de lisser une image d'entrée. Cependant, parfois, les filtres ne dissolvent pas seulement le bruit, mais aussi lissent les bords . Pour éviter cela nous pouvons utiliser un filtre bilatéral.

D'une manière analogue à celle du filtre gaussien, le filtre bilatéral considère également les pixels voisins avec des poids attribués à chacun d'eux. Ces poids ont deux composantes, dont la première est la même pondération utilisée par le filtre gaussien. La deuxième composante prend en compte la différence d'intensité entre les pixels voisins et l'évaluation.

Du coté de la méthode voici ce que l'on peut trouver:

```
bilateralFilter  (  src ,  dst ,  i ,  i * 2 ,  i / 2  );
```

- **Src :** image source
- **Dst :** image de destination
- **D :** Le diamètre de chaque quartier de pixels.
- σ_{Color}**:** Écart type dans l'espace colorimétrique.
- σ_{Space}**:** Écart type dans l'espace de coordonnées (en termes de pixels)

Charles CAGNON
Nathan HUE

mardi 6 juin 2017

Nos tests

Les premiers test ont été effectué avec python2.7 et les librairies Opencv et numpy.

test 1 : Découpe de visage

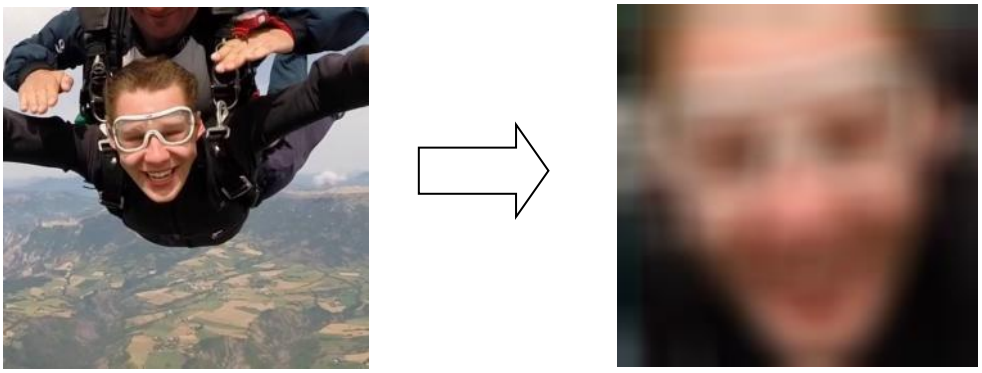
Le tout premier programme consistait à faire de la détection de visage en important un fichier image et de découper ce dernier pour l'enregistrer dans un fichier clone recadré.

```
img = cv2.imread(image)
face_model = cv2.CascadeClassifier("haarcascade_frontalface_alt2.xml")
faces = face_model.detectMultiScale(img)
print ("nombre de visages", len(faces), "dimension de l'image", img.shape, "image", image)
for face in faces:

    cv2.rectangle(img, (cv2.blur(face[0],(5,5)), face[1]), (face[0] + face[2], face[0] + face[3]), (255, 0, 0), 3)

cv2.imwrite(image_out, img)
if show :
    cv2.imshow("visage",img)
```

test 2 : Premiers pas avec le Blurring

Ensuite nous avons appris à positionner des rectangles sur la zone qui délimite la détection du visage, et de rendre floue cette zone rectangle.

```
image = cv2.imread("./jb.jpg")
result_image = image.copy()

face_cascade_name = "./haarcascade_frontalface_alt.xml"
face_cascade = cv2.CascadeClassifier()
face_cascade.load(face_cascade_name)

#Preprocess de l'image
grayimg = cv2.cvtColor(image, cv2.cv.CV_BGR2GRAY)
grayimg = cv2.equalizeHist(grayimg)

#Utilisation des classifiers
faces = face_cascade.detectMultiScale(grayimg, 1.1, 2, 0|cv2.cv.CV_HAAR_SCALE_IMAGE, (30, 30))
print "Visage detectés"

if len(faces) != 0:         # Si il y a des visage dans l'image
    for f in faces:         # Pour chacun d'entre eux

        x, y, w, h = [ v for v in f ]

        cv2.rectangle(image, (x,y), (x+w,y+h), (255,255,0), 0)
        sub_face = image[y:y+h, x:x+w]
        # On applique un blur gaussien
        sub_face = cv2.GaussianBlur(sub_face,(23, 23), 30)
        # On place ce rectangle blurré sur l'image
        result_image[y:y+sub_face.shape[0], x:x+sub_face.shape[1]] = sub_face
        face_file_name = "./face_" + str(y) + ".jpg"
        cv2.imwrite(face_file_name, sub_face)
```



Charles CAGNON
Nathan HUE

Test 3 : Lecture video

En combinant ces deux premiers test nous avons écrit un script permettant de délimiter les visages en temps réel sur une vidéo.

```
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

#cap = cv2.VideoCapture(0)          #webcam
cap = cv2.VideoCapture("out.mp4") #video

while cap.isOpened():
    ret, img = cap.read()

    if ret:
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(gray, 1.3, 5)

    for (x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]

    cv2.imshow('img',img)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

puis flouter ces images en les plaçant directement sur la lecture de la vidéo.

Malheureusement nous avons été confronté a des bugs sur python (erreur 215), et notamment sur le post process de l'image en BGR2GRAY donc nous somme passé sur un programme codé en c++, qui sera présenté plus en détail ci dessous.

Charles CAGNON
Nathan HUE
mardi 6 juin 2017

Programme final

Notre programme est capable de détecter les visages sur une vidéo en temps réel, puis d'appliquer un flou de différents type. Dans le premier cas, on lit la sortie webcam, sinon on lit une video importée à la racine de notre dossier.

Pour garantir la meilleure expérience possible, nous avons optimisé les paramètres de manière a obtenir la meilleure fluidité possible.

En effet le principe de détection de visage est très gourmand en calcul. C'est pourquoi nous nous sommes concentrés tout particulièrement sur la méthode "detecMultiscale" du "CascadeClassifier" et avons réajusté la taille de la zone de détection dynamiquement en fonction du nombre de colonnes et de lignes de la frame. De plus nous limitons a 3 le nombre de voisins maximum détectables pour gagner en vitesse de traitement.
Pour ce qui est du fichier XML importé nous en avons testé de multiples mais haarcascade_frontalface_alt.xml semble le meilleur.

```
//haar_cascade.load("haarcascade_frontalface_default.xml"); //Cascade peu performante
haar_cascade.load("haarcascade_frontalface_alt.xml");
```

Voici un récapitulatif détaillé des paramètre présents dans la méthode DetectMultiscale(..)

- **Image** - Matrice du type `CV_8U` contenant une image où des objets sont détectés.
- **Objets** - Vector de rectangles où chaque rectangle contient l'objet détecté.
- **ScaleFactor** - Paramètre spécifiant combien la taille de l'image est réduite à chaque échelle d'image.
- **MinNeighbors** - Paramètre précisant combien de voisins chaque rectangle candidat devrait avoir à le retenir.
- **Drapeaux** - Paramètre ayant la même signification pour une ancienne cascade que dans la fonction `cvHaarDetectObjects`. Il n'est pas utilisé pour une nouvelle cascade.
- **Taille** minimale possible. Des objets plus petits que ceux qui sont ignorés.
- **MaxSize** - Taille maximale de l'objet possible. Des objets plus importants que ceux qui sont ignorés.

Et voici les paramètres que nous avons appliqués.

```
//Trouver des visages dans la video a partir d'un fichier cascade
//contenant les positions probables des traits du visages
vector< Rect_<int> > faces;
float facteur_echelle = 1.1;
int voisins_minimum = 3;

//haar_cascade.detectMultiScale(gray, faces); //qualité
haar_cascade.detectMultiScale(gray, faces,    //performance
facteur_echelle,
voisins_minimum,
0|CV_HAAR_SCALE_IMAGE,
Size(frame.rows / 5, frame.rows / 5),
Size(frame.rows * 2 / 3, frame.rows * 2 / 3));
```

Charles CAGNON
Nathan HUE
mardi 6 juin 2017

Pour ce qui est de la navigation dans le programme. L'utilisateur à la possibilité de choisir un type de filtrage :

Et d'ajuster le niveau d'intensité de blurring à l'aide des boutons "o" et "p" pour monter ou descendre la valeur des variables associées à:
- A la taille du noyau dans le cas des filtrages "Du lissage normalisé" et du "Blur Median".
- Aux écarts type des distances en X,Y au noyau dans le cas du "filtrage Gaussien".

Ceci grâce a la méthode getVal() qui renvoi nécessairement une valeur impaire pour les méthodes des blur gaussien et medians, sans quoi le calcul est faux et notre programme crashe.

```
int val=7;

// modification smoothing Temps Reel
int getVal(char msg){
   int val_int;
   switch(msg){
        case 'a':  val_int = val;                                    break; //blur homogene
        case 'b':  val_int = val; if (val_int%2==0){val_int++;}    break; //blur gaussien
        case 'c':  val_int = val; if (val_int%2==0){val_int++;}    break; //blur median
   }

   if (val>0 && val<50){
      return val_int;
   } else {
      val=9;
      return val;
   }
}
```

Charles CAGNON
Nathan HUE

mardi 6 juin 2017

Les valeurs des filtres sont modifiées en temps réel.

```
//Pour chaque visages detectés
for(int i = 0; i < faces.size(); i++) {

    Rect face_i = faces[i];
    Mat face = gray(face_i); //rogne le visgae

        //Application du type de smoothing
        switch(msg[0]){
            case 'a':
                blur(original(face_i),original(face_i),Size(getVal(msg[0]),getVal(msg[0])));
                break;
            case 'b':
                GaussianBlur(original(face_i),original(face_i), Size(getVal(msg[0]),getVal(msg[0])), 0, 0 );
                break;
            case 'c':
                medianBlur(original(face_i),original(face_i), getVal(msg[0]) );
                break;
            case 'd':
                cout << "Cette methode n'est toujours pas au point\n";
                //bilateralFilter(original(face_i),original(face_i), 0.5 ,0.5*2, 0.5/2 );
                break;
            default:  break;
        }
```

Pour vérifier nos performances on a placé une Textbox au dessus des visages. Cette dernière contient les valeurs des temps de rafraichissement par frame, et la valeur en fps.

```
//Rectangle vert d'epaisseur 1
rectangle(original, face_i, CV_RGB(0, 255,0), 1);

//Label au dessus des tetes
string box_text = format("Temps par frame: %3.3f    fps: %3.3f", temps_par_frame,fps);
int pos_x = std::max(face_i.tl().x - 10, 0);
int pos_y = std::max(face_i.tl().y - 10, 0);
putText(original, box_text, Point(pos_x, pos_y), FONT_HERSHEY_PLAIN, 1.0, CV_RGB(0,255,0), 2.0);
```

Voici un aperçu des différents filtres que nous obtenons sur notre video.

Blur Convolution	Blur Gaussien	Blur Median
		

Charles CAGNON
Nathan HUE
mardi 6 juin 2017

Sources :

http://docs.opencv.org/trunk/d4/d13/tutorial_py_filtering.html

http://docs.opencv.org/2.4/modules/contrib/doc/facerec/tutorial/facerec_video_recognition.html