# Détection de visages et repères

## Image numérique et vision par ordinateur

Alexis GUILLOTEAU

**IMT Lille Douai**
École Mines-Télécom
IMT-Université de Lille

# TP 3 – Détection de visages et repères

## Introduction de CMake

Afin de compiler le package « dlib » nous devons installer CMake.

CMake est un outil logiciel open-source permettant la compilation cross-plateforme. Si vous besoin d'installer un package ou logiciel a partir de source sous linux vous aurez surement besoin d'effectuer une compilation des sources grâce à CMake.

## Introduction de dlib

Dlib est une librairie cross-plateforme à faible impact de ressource, développée pour les réseaux, opération numériques, apprentissage et traitement d'images.

## Introduction aux repères

La détection de repères sur un visage correspond à la prédiction de formes sur des zones d'intérêts. Cette détection se fait en deux étapes :

1 - Localisation du visage complet dans une image

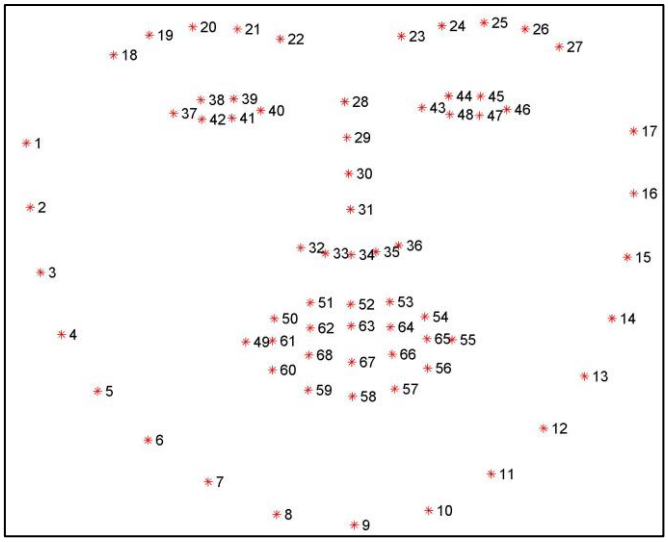
2 - Détection des zones d'intérêts et prédiction grâce à un ou plusieurs modèles pré enregistrées telles que la bouche, les sourcils, les yeux, le nez, la mâchoire…

Dans le cas des librairies « dlib », la détection de repères de visage grâces à deux type d'informations, un dataset d'images avec les coordonnées de la structure du visage associées manuellement (On peut très bien aussi entrainer un réseau pour générer un dataset pour entrainer une autre réseau et enchainer ceci sur plusieurs niveaux et embranchements) ainsi qu'un jeu de probabilité concernant les distances entres les différents repères (œil droit par rapport à œil gauche ou la bouche…). On peut alors utiliser l'intensité des pixels une fois le visage localisé pour essayer de déterminer les différents repères/éléments constituant le visage.

Dlib a été entrainé en utilisant les annotations par 68 points du dataset iBUG 300-W :

https://ibug.doc.ic.ac.uk/resources/facial-point-annotations/

Remarques : Ce dataset ne peut être utilisé que dans de le but de la recherche, dans le cadre d'un projet commercial vous devrez créer votre propre base d'annotation. Tous les datasets que vous trouverez ne sont pas forcements libre de toutes utilisations. Ci-dessous vous trouverez une visualisation de 68 points de repères provenant de ce dataset.




Bien que fortement utilisé, ce dataset n'est pas le seul disponible si vous souhaitez effectuer de la reconnaissance de repères sur visage. Le modèle sur 194 points HELEN peut aussi être utilisé :

http://www.ifp.illinois.edu/~vuongle2/helen/

Pour ce TP nous garderons le modèle sur 68 pour un gain de temps et pour simplifier la seconde partie du TP qui vous demandera de remplacer une partie de l'image grâce à une création manuelle d'un Template sur points.

## Imports nécessaires

```
# Imports nécessaires pour ce TP
from imutils import face_utils
import numpy as np
import argparse
# Bien vérifier que vous avez votre package imutils à jour
import imutils
import dlib
import cv2
```

Porter bien attention à avoir numpy et imutils à jour dans votre environnement de développement.

Dlib met à votre disposition différents prédicteurs de forme, dans ce TP nous allons utiliser le prédicteur pré entrainé « shape_predictor_68_face_landmarks.dat ».

Créez deux nouvelles entrées d'arguments dans votre script : chemins vers le prédicteur et l'image à traiter.

Initialisez votre détecteur dlib :

```
# On initialise un module de detection de visage dlib
# puis on lui assigne notre prédicteur
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(args["shape_predictor"])
```

Chargez votre image, passez la en niveau de gris.

Il est possible que vous deviez redimensionner votre image si celle-ci fait plus de 500 pixels de large, le prédicteur dlib fonctionne mieux avec des imagettes de taille limité.
Dans ce cas effectuer un redimensionnement proportionnel en gardant 500 pixels de largeur (indice : imutils.resize)

```
# Charger l'image
image = cv2.imread(args["image"])
image = imutils.resize(image, width=500)
result = image.copy()
result = result.astype(np.float32) / 255.0
# puis la passer en niveau de gris
# Rappel : Preprocessing en niveau de gris pour adherer au model
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Appliquez votre détecteur à votre image :

```
# On applique la detection de visage sur notre image en niveau de gris
gray_rec = detector(gray, 1)
```

Utilisez la boucle ci-dessous (et compléter là  afin d'afficher les repères du visage et vous aidez à les utiliser plus tard) :

```
# On boucle sur la detection
for (i, rect) in enumerate(gray_rec):
	# Apres avoir determiner les reperes sur le visage
	# On doit convertir les coordonnées des reperes dans
	# un tableau numpy afin de pouvoir les réutilisers pour la partie suivante du TP
	forme = predictor(gray, rect)
	forme = face_utils.shape_to_np(forme)

	# Pour aider à l'affichage on peut ajouter un rectangle
	# sur notre image au niveau de la zone de detection du visage
	(x, y, w, h) = face_utils.rect_to_bb(rect)
	# Il faut alors récupérer les coordonnées de la fonction rectangle
	# de dlib pour la convertir en rectangle au format opencv
	...

	# Dans le cas on nous ajoutons une règle de traitement relatif aux différents visages
	# nous pouvons les annoter si plusieurs visages
	...

	# Enfin on boucle sur tout nos coordonnées de points d'interets
	# et on les ajoute à notre image de base avec un petit cercle pour être visible
	j = 0

	landmarks = [None] * 100
	for (x, y) in forme:
		print(j)
		landmarks[j] = (x, y)
		...
		j=j+1

	# On enregistre notre résultat de détection
	cv2.imwrite("result.jpg",image)
```

Vous pouvez afficher votre résultat afin de confirmer votre boucle.

Définissiez votre tableau de points correspondant à l'espace que vous souhaitez remplacer (ci-dessous la bouche comme exemple) :

```
dst_pts = np.array(
      [
            forme[48],
            forme[49],
            forme[50],
            forme[51],
            forme[52],
            forme[53],
            forme[54],
            forme[55],
            forme[56],
            forme[57],
            forme[58],
            forme[59],
      ],
      dtype="float32",
)
```

Nous allons utiliser cette liste afin de déformer l'image qui devra se placer sur notre espace.

L'image de remplacement devra être associée à un fichier de type CSV tel l'exemple suivant :

```
landmark,x,y
49,23,22
50,47,21
51,74,21
52,104,20
53,129,20
54,152,22
55,190,22
56,183,45
57,163,65
58,121,79
59,69,73
60,37,51
```

Ce fichier contient les coordonnées des points similaires à ceux de notre première liste mais sur l'image de remplacement.

Attention : votre image doit avoir un fond transparent (channel alpha) pour de meilleur résultat

Renseignez votre image et son fichier de points correspondant :

```
mask_image = "masks/bouche.png"
mask_points = "masks/bouche.csv"
```

Ouvrez le fichier de point et parcourrez le de façon à obtenir votre tableau pour l'image :

```
with open(mask_points) as csv_file:
	csv_reader = csv.reader(csv_file, delimiter=",")
	src_pts = []
	for i, row in enumerate(csv_reader):
		# Si présence de ligne vide alors on passe à la suite
		try:
			src_pts.append(np.array([float(row[1]), float(row[2])]))
		except ValueError:
			continue
src_pts = np.array(src_pts, dtype="float32")
```

Vous avez maintenant deux tableaux :

- dst_pts : les coordonnées des points où se trouve l'objet à remplacer
- src_pts : les coordonnées internes à l'image de remplacement

Nous pouvons maintenant effectuer notre mise en correspondance et superposition :

```
# On applique une superposition si chaque coordonnées à une correspondance
if (forme > 0).all():
	print(">0")
	# Chargement de l'image
	mask_img = cv2.imread(mask_image, cv2.IMREAD_UNCHANGED)
	mask_img = mask_img.astype(np.float32)
	mask_img = mask_img / 255.0

	# On créer une matrice de transformation (perspective)
	M, _ = cv2.findHomography(src_pts, dst_pts)

	# On applique la transformation
	transformed_mask = cv2.warpPerspective(
		mask_img,
		M,
		(result.shape[1], result.shape[0]),
		None,
		cv2.INTER_LINEAR,
		cv2.BORDER_CONSTANT,
	)

	cv2.imshow("Image superposée simple", transformed_mask)


	alpha_mask = transformed_mask[:, :, 3]
	alpha_image = 1.0 - alpha_mask

	for c in range(0, 3):
		result[:, :, c] = (
			alpha_mask * transformed_mask[:, :, c]
			+ alpha_image * result[:, :, c]
		)
```

Exemple:

Image de base :

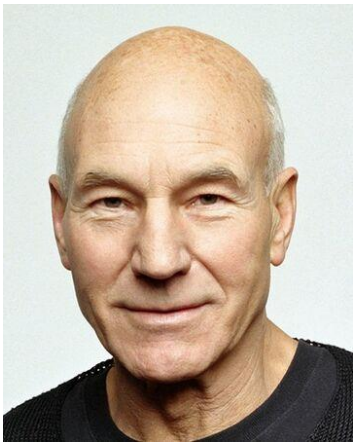
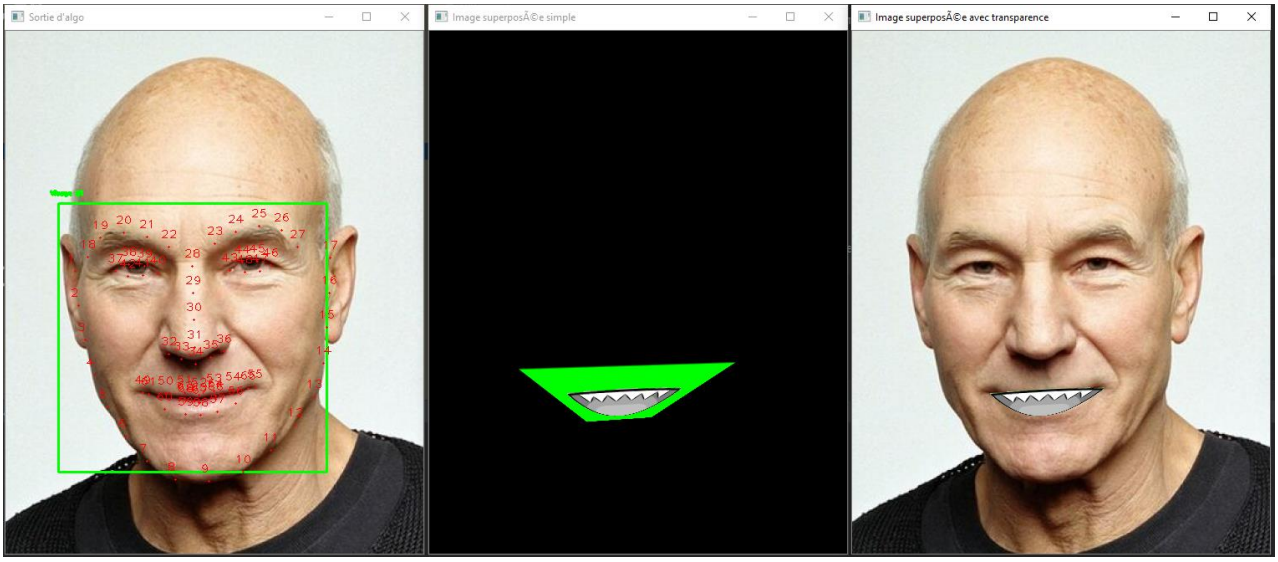


Image de remplacement :

Résultat :




Votre objectif est de créer un programme remplacant la bouche, le nez et les yeux en utilisant la même méthode.