Système de Reconnaissance faciale

Traitement du signal et des images

Fadil boodoo & Adrien Leroy-Lechat - 12 April 2019

Table de matière

Introduction

Nous cherchons à implementer un système capable de reconnaitre le visage d'une personne à partir d'une base de donnée contenant déjà les images des personnes. Ce sytème est pensé pour être couplé à un régulateur de flux de personne (porte d'entrée, portique, checkpoint,..) afin de contrôler de manière automatique par reconnaissance faciale les personnes autorisées à passer.

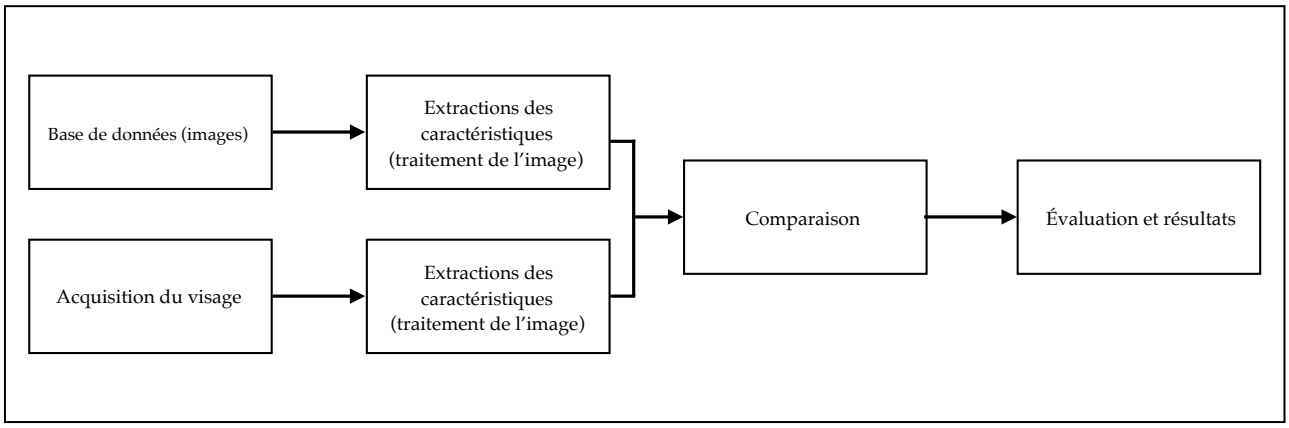

Figure 1 : Schéma fonctionnel du système

Différentes méthode de reconnaissance faciale

Pour répondre à ce besoin, il existe plusieurs méthodes divisibles en deux catégories: la reconnaissance 3D, et la reconnaissance 2D.

La reconnaissance 2D est elle-même divisible en deux sous-catégories : les algorithmes qui créer une image géométrique de l'utilisateur en fonction de différents paramètres comme la taille d'éléments du visage, forme et distance entre eux. Et les algorithmes qui encode numériquement l'image, en utilisant par exemple les algorithmes de fourrier, les Eigenfaces, les Fisherfaces, le Scale Invariant Features Transform (SIFT), le Speed Up Robust Features (SURF), ou encore le Local Binary Patternes Histograms (LBPH).

Nous avons choisi de répondre à ce besoin en utilisant la méthode du Local Binary Patternes Histograms pour sa facilité d'implémentation, tout en conservant une bonne efficacité dans la reconnaissance faciale.

La méthode LBPH

La méthode LBPH se repose principalement sur le Local Binary Pattern (LBP). C'est une méthode de traitement d'image qui permet de prendre en compte la texture de l'image en considérant pour un pixel donnée un seuillage binaire entre le pixel et ses voisins.

Il a était décrit pour la première fois en 1994. Il est depuis considéré comme un puissant outils pour la reconnaissance de texture en traitement d'image. Il a était par la suite, découvert que associé à un histogramme, la méthode LPB devenait bien plus performant dans la reconnaissance d'image. Avec ce dernier, nous pouvons representer les images comme de simple vecteur.

Données d'entrée

Comme donnée d'entrée, nous allons prendre des images des personnes autorisées à entrées. Ces images sont stockées dans un dossier. Nous avons aussi besoin de l'image de la personne qui essaie de pénétrer.

Calcul du LBP de l'image

Une fois l'image acquise, nous allons créer une nouvelle image qui décrit mieux l'image originale, en faisant apparaitre les caractéristiques faciales. Pour faire ceci, l'algorithme applique sont calcul par fenêtre (matrice), en se basant sur les paramètres de **rayon**, et de **voisinage**.

Nous supposons l'image d'origine est en niveaux de gris.
La fenêtre est une matrice dont la taille depends du rayon, et du voisinage.

Le **rayon** représente le rayon autour du pixel central. Ce rayon contient les pixels pris en compte le calcul du LBP. Plus le rayon est élevé, plus les pixels pris en compte pour le calcul seront éloignés. Il est généralement mis à 1.

Le **voisinage** représente le nombre de pixel pris en compte pour le calcul du LBP. Plus ce nombre est élevé, plus le coût de calcul devient élevé. Il est généralement fixé à 8.
A noter que si la valeur du nombre de voisinage ne permet pas de couvrir tous les pixels se situant dans le disque couvert par le rayon, alors ce sont les pixels les plus éloignés qui sont pris en compte en premier.

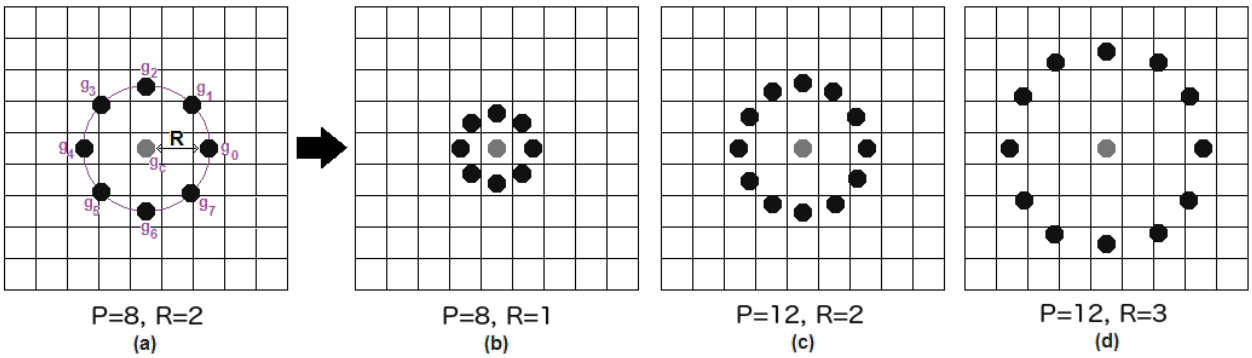

Figure 2 : Influence des paramètres rayon et voisinage dans la définition de la taille de la fenêtre

Une fois la fenêtre (taille de la matrice) fixé, on applique cette fenêtre pour un pixel de l'image. La valeur centrale de la fenêtre sera la valeur du pixel et servira de seuil. Si la valeur du pixel voisin est inférieure à la valeur du seuil, on mets 0 dans la fenêtre pour ce pixel voisin. Sinon on mets 1. Et on fais ceci pour tous les voisins du pixel considérés.
On obtient alors une matrice avec au centre le seuil, et autour des valeurs binaires.
On "lit" la valeur binaire obtenue (nous pouvons la lire de haut en bas, ou dans le sens horaire, mais la manière de le lire importe peu), et on transforme ce nombre binaire an nombre decimal.
On stocke ce nombre decimal dans une nouvelle matrice. Ce nombre decimal correspond à la transformation LBP du pixel considéré.

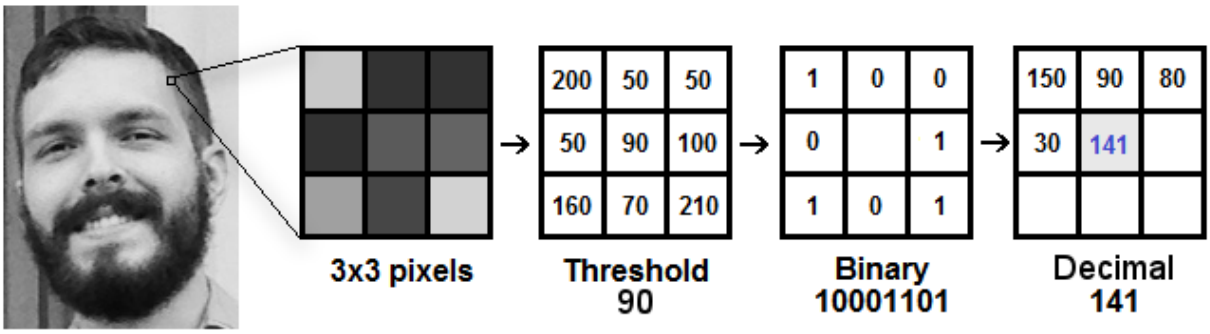

Figure 3 : Processus du calcul LBP d'un pixel de l'image. Sur cette exemple, la valeur du pixel et par ailleurs du seuil est à 90, ce qui donne pour nombre binaire $(10001101)_2$. Ce qui fait 141 en décimal, cette dernière est la valeur LPB du pixel.

On répète ce processus pour chaque pixel de l'image. On obtient ainsi la matrice LPB de l'image. Cette nouvelle matrice LPB est bien plus représentatif des caractéristiques de l'image que l'image originale.
A noter que les valeurs de la matrice LPB changent en fonction des paramètres (rayon, voisinage) de la fenêtre.

Extraction des l'histogrammes

Une fois la matrice LBP de l'image obtenue. Nous allons calculer son histogrammes. Mais cela ne se fait pas en une fois. En effet, il est important de considérer l'image par region, pour prendre les caractéristiques macroscopique de l'image (bouche, oeil, nez, ..). Pour ce faire, nous allons faire un calcul d'histogramme par région.
Il faut tout d'abord créer une matrice qui servira de grillage pour delimiter la matrice LBP de l'image en region. Cette matrice grillage depends de sa taille. Elle a donc comme paramètre sa **longueur x**, et sa **largeur y**.
La **longueur x** correspond donc au nombre de pixel en vertical pris en compte pour le calcul de l'histogramme régional. Il est généralement mis à 8.
La **longueur y** correspond au nombre de pixel en horizontale pris en compte pour le calcul de l'histogramme régional. Il est généralement mis à 8.
On applique le grillage sur la matrice LBP de l'image, et on calcul l'histogramme du grillage.

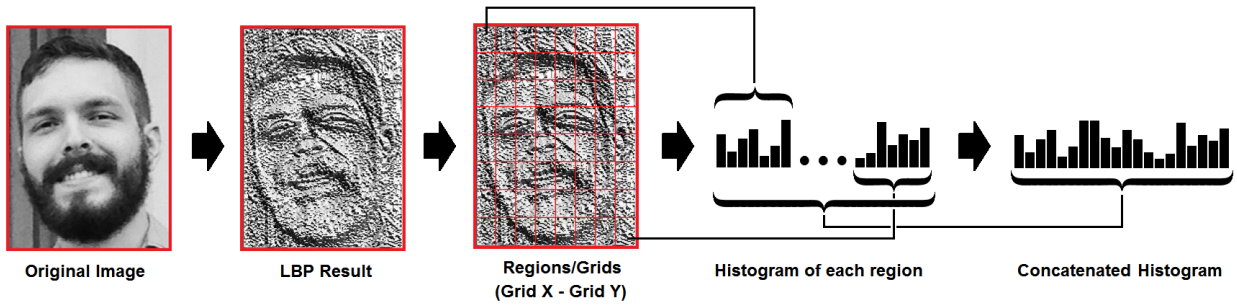

Figure 4 : Processus du calcul de l'histogramme. On divise la matrice LBP en sous-region. On calcul l'histogramme de chaque region indépendamment, pour ensuite les concaténer en un seul histogramme.

On répète ce processus da manière à couvrir toute la matrice LBP de l'image. Ensuite, on fait une concaténation de tous ces histogrammes régionaux, en un seul histogramme.
Cet histogramme est l'histogramme LBP de l'image.

Reconnaissance de visage

On considère à cette étape que l'algorithme a déjà calculé l'histogramme LBP de toutes les images contenues dans la base de donnée. On veut à present lui presenter une nouvelle image et lui demander s'il reconnait ce visage.
Pour ce faire il faudra tout d'abord qu'il calcule l'histogramme LBP de cette nouvelle image. Ensuite, il faudra qu'il calcule la distance euclidienne entre l'histogramme LBP de cette nouvelle image, et l'histogramme LBP d'une images de la base de donnée.
La distance euclidienne se calcul à travers cette formule :

$$distance\ eucledienne = \sqrt{\sum_{i=1}^{n} (histogrammeLBP1_i - histogrammeLBP2_i)^2}$$

On répète ce processus pour chacune des images de la base des données.
Ensuite on compare les distances euclidiennes. Plus la distance est petite, plus l'algorithme le considère les deux images comme étant proche.
On peut donc définir un seuil de confiance en dessous du quel on estime que les deux images correspondent au même visage.

Données en sortie

Si la plus petite distance euclidienne est en dessous du seuil de confiance, alors l'algorithme considérera que c'est une personne qu'il connait, et retournera une message positif. Dans le cas contraire, c'est un message négatif qui sera retourné.

Implémentation du code

Le code est implémenté en faisant appel à plusieurs fonctions, dont nous les détaillerons un à un dans cette partie :

Calcul du LBP de l'image

```
function image_lbp = lbp (image)
    size_image=size(image);
    matrix_arrows = size_image(1);
    matrix_columns = size_image(2);
    image_lbp = [];

    for i=2:matrix_arrows-1
        for j=2:matrix_columns-1
            m=1;
            for k=-1:1
                for l=-1:1
                    if k==0 && l==0
                    else
                        if image(i+k,j+l) < image(i,j)
                            windows(m) = 0;
                            m = m+1;
                        else
                            windows(m) = 1;
                            m = m+1;
                        end
                    end
                end
            end
            image_lbp(i,j)=bintodec(windows)
        end
    end
endfunction
```

Figure 5 : Detail de la fonction *lbp*, qui permet de renvoyer une matrice lpb d'une image à partir d'une image qui lui a été entrée.

Le calcul du LBP de l'image se fait à travers la fonction *lbp*.
Pour les bords, nous avons décider de les mettre à 0.
La fonction *lbp* permet de renvoyer une image lbp à partir de la matrice image qui lui a été entrée. Il faut en effet appliqué la fonction Imread sur l'image avant de l'envoyer dans cette fonction.
A noter que cette fonction fait appelle à une autre fonction *bintodec* pour transformer la valuer binaire obtenu en valeur decimal.

Voici le detail de cette fonction :

```
function decimal=bintodec(bin)
    size_vector = size(bin)(1);
    decimal = 0;
    for i=1:size_vector
        decimal = decimal + bin(i)*(2^(size_vector-i));
    end
endfunction
```

Figure 6 : Detail de la fonction *binotec* qui permet de renvoyer un nombre en decimal à a partir d'un nombre en binaire

Extraction des l'histogrammes

L'extraction des histogrammes se fait à travers la fonction *hist_lbp*.
La fonction calcul chaque histogramme régional, et l'introduit dans une matrice hist_lbp_image.

```
function hist_lbp_image = hist_lbp(image_lbp)
    size_image = size(image_lbp);
    image_length = size_image(1);
    image_width = size_image(2);
    hist_lbp_image = [];

    grid_size_x = 8;
    grid_size_y = grid_size_x;
    x=[-0.5:255.5];
    indice=1;
    k_max = int(image_length/grid_size_x);
    l_max = int(image_width/grid_size_y);

    for k=0:k_max-1
        for l=0:l_max-1
            for i=8*k+1 : 8*k+grid_size_x
                for j=8*l+1 : 8*l+grid_size_y
                    grid(i-8*k,j-8*l)=image_lbp(i,j);
                end
            end
            hist_lbp_image(indice,:) = histc(x,grid,normalization=%f);
            indice = indice +1;
        end
    end
endfunction
```

Figure 7 : Detail de la fonction *hist_lbp* qui permet de l'histogramme lbp d'une image à partir d'une matrice lbp d'une image.

Initialisation du système

Les images contenants les personnes autorisés à y entrer sont stocké dans le dossier base_de_donnee situé à la racine du code.
À l'initialisation du système, le code va charger toutes le images contenu dans le dossier base_de_donnee. Il va stoker les image dans une list database_image, et va calculer leur histogramme et les stocker dans la list database_lbp.
Une fois l'initialisation achevé, il enverra un massage indiquant que l'initialisation est terminé.
Tout ceci se fait grâce à la fonction *load_database_image*, et *database_imagetolbp* :

```
1  function database_image = load_database_image()
2      f= findfiles('../base_de_donnee','*.png');
3      f_max = size(f)(1);
4      database_image = list();
5
6      for i=1:f_max
7          image=string(f(i));
8          database_image(i)=imread(image)(:,:,1);
9      end
10 endfunction
```

Figure 8 : Detail de la fonction *load_database_image* qui permet de charger toutes les images contenue dans le dossier base_de_donnee, et de les stocker dans database_image

```
1  function database_lbp = database_imagetolbp(database_image)
2          i_max = size(database_image);
3          database_lbp = list();
4
5          for i=1:i_max
6              image_lbp = lbp(database_image(i));
7              database_lbp(i)= hist_lbp(image_lbp);
8          end
9  endfunction
```

Figure 9 : Detail de la fonction *database_imagetolbp* qui permet de calculer l'histogramme lbp de toutes les images lbp contenue dans database_image et les stocker dans database_lbp.

Tout ceci se lance grâce à ces trois lignes de codes

```
108 database_image = load_database_image();
109 database_lbp = database_imagetolbp(database_image);
110 disp("base de donnée chargé");
```

Figure 10 : Code permettant l'initialisation du système

Lorsqu'une image est donné à l'algorithme pour être reconnu grâce à l'interface graphique, l'image est envoyé à la fonction *compare_lbp* qui va prendre l'image, calculer son histogramme lbp, et la comparer avec ceux existant dans la base de donnée. Il va aussi retourner toutes la distances euclidienne pour chaque image de la base de donnée qu'il a comparé.

```
function distances = compare_lbp(image,database_lbp)
    image = image(:,:,1);
    image_lbp = lbp (image);
    hist_lbp_img = hist_lbp(image_lbp);
    i_max= size(database_lbp);
    distances = [];

    for i=1:i_max
        distances(i)=distance_eucledienne(hist_lbp_img, database_lbp(i))
    end
    disp("compare_lbp terminé");
endfunction
```

Figure 11 : Fonction *compare_lbp permettant de comparer une image avec les images de la base de donnée*

A noter que cette fonction utilise la fonction suivante pour le calcul de la distance euclidienne :

```
function distance = distance_eucledienne(matrice1, matrice2)
    distance = sqrt(sum((matrice1-matrice2).^2));
endfunction
```

Figure 12 : Fonction *distance_eucledienne* permettant de calculer la distance euclidienne de 2 matrices.

Résultas

Constitution de la base de donnée

Il existe plusieurs base de données en libre de droit contenant des images de visages permettant de tester les algorithme de reconnaissance faciale.
Il existe par exemple celui de NIST contenant les images faciales de 1573 personnes, ou bien la PIE database contenant 41369 images de 68 personnes.

Pour notre test, nous avons utilisé la base de donnée de l'université de Yales. C'est une base de donnée contenant 11 images de 15 personnes dans différentes expression : neutre, content, triste, endormie, etc. Les images sont en niveaux de gris. Les photos ont était prises avec la même luminosité, et sur le même fond blanc. Les images sont toutes de résolution identiques (320×243) en format GIF.
Cette base de donnée est libre de droit, et téléchargeable à cette adresse : http://www.face-rec.org/databases/

Test

Pour le test, nous avons utilisé entrée dans la base de donnée les 15 image des personnes avec une expression neutre.
Ensuite, nous avons demandé de reconnaître une image d'une personne avec l'expression content, et ceci pour chaque personne. Nous avons relevé la distance euclidienne minimum, et si cela correspondait bien à la personne en question.

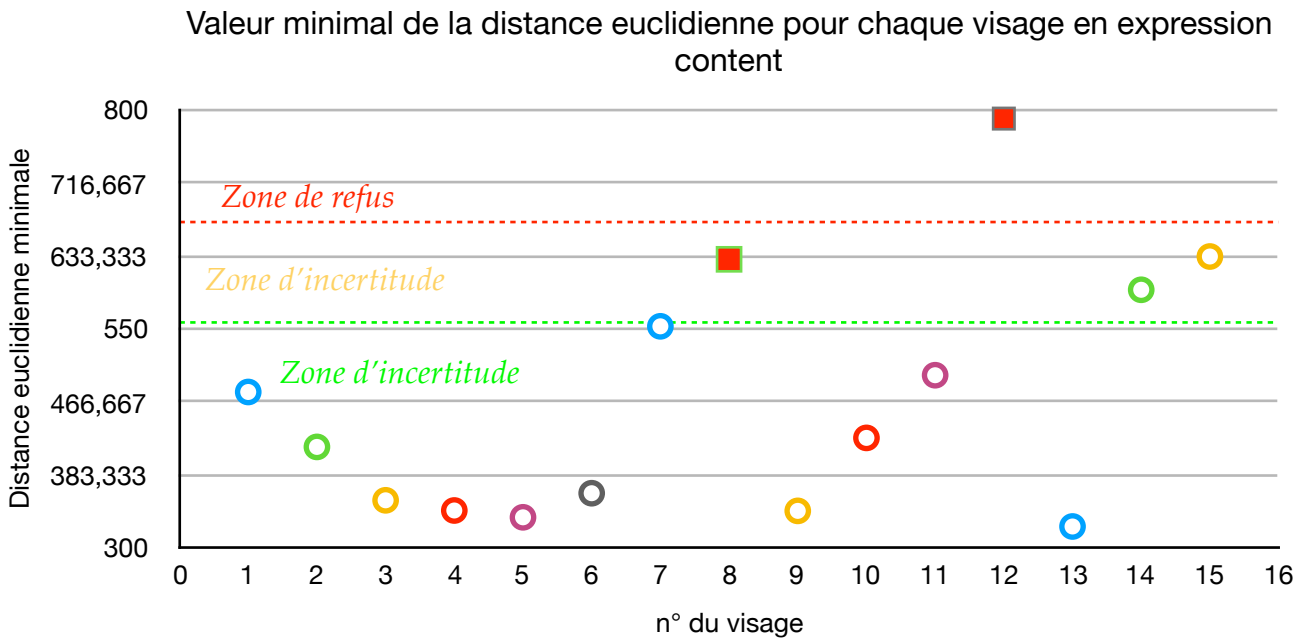

Figure 13 : graphique montrant la distance minimal euclidienne pour chaque visage en expression content, en fonction de chaque image en expression neutre. Les carrées rouges indiquent que l'algorithme s'est trompé.

On remarque avec la figure 13 que les l'algorithme s'est trompé sur 2 visages pour 15 visages présentés. Ce qui fait un succès de de 86,7%. L'algorithme s'est trompé pour le visage n°8 et le visage n°12 avec pour valeur respective de la distance euclidienne 629,35 et 790,09.
Par contre il a trouvé correctement pour le visage n°15, malgré une valeur euclidienne à 632,56.
Ce qui nous amène à créer trois zones : une zone de confiance, où nous pouvons conclure que c'est bien le même visage, une zone d'incertitude ou nous ne pouvons lever l'indétermination , et zone de refus et nous pouvons refuser la personne.

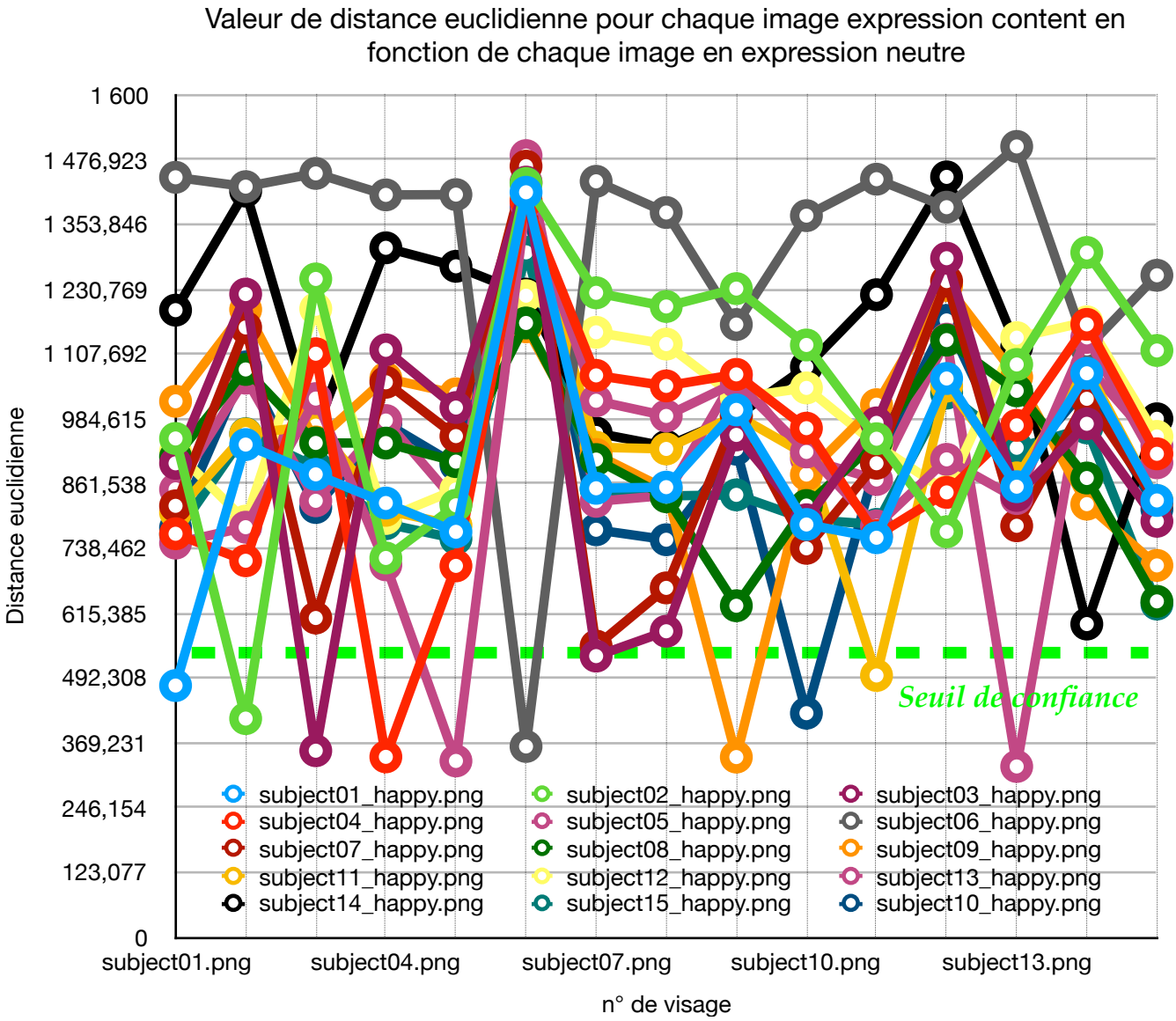

Figure 14 : graphique montrant la distance minimal euclidienne pour chaque visage en expression content, en fonction de chaque image en expression neutre.

Dans la figure ci-dessus, on remarque que tous les points situé en dessous de 550 indique la bonne personnes. Nous pouvons donc prendre la valeur de **550 comme seuil de confiance**.

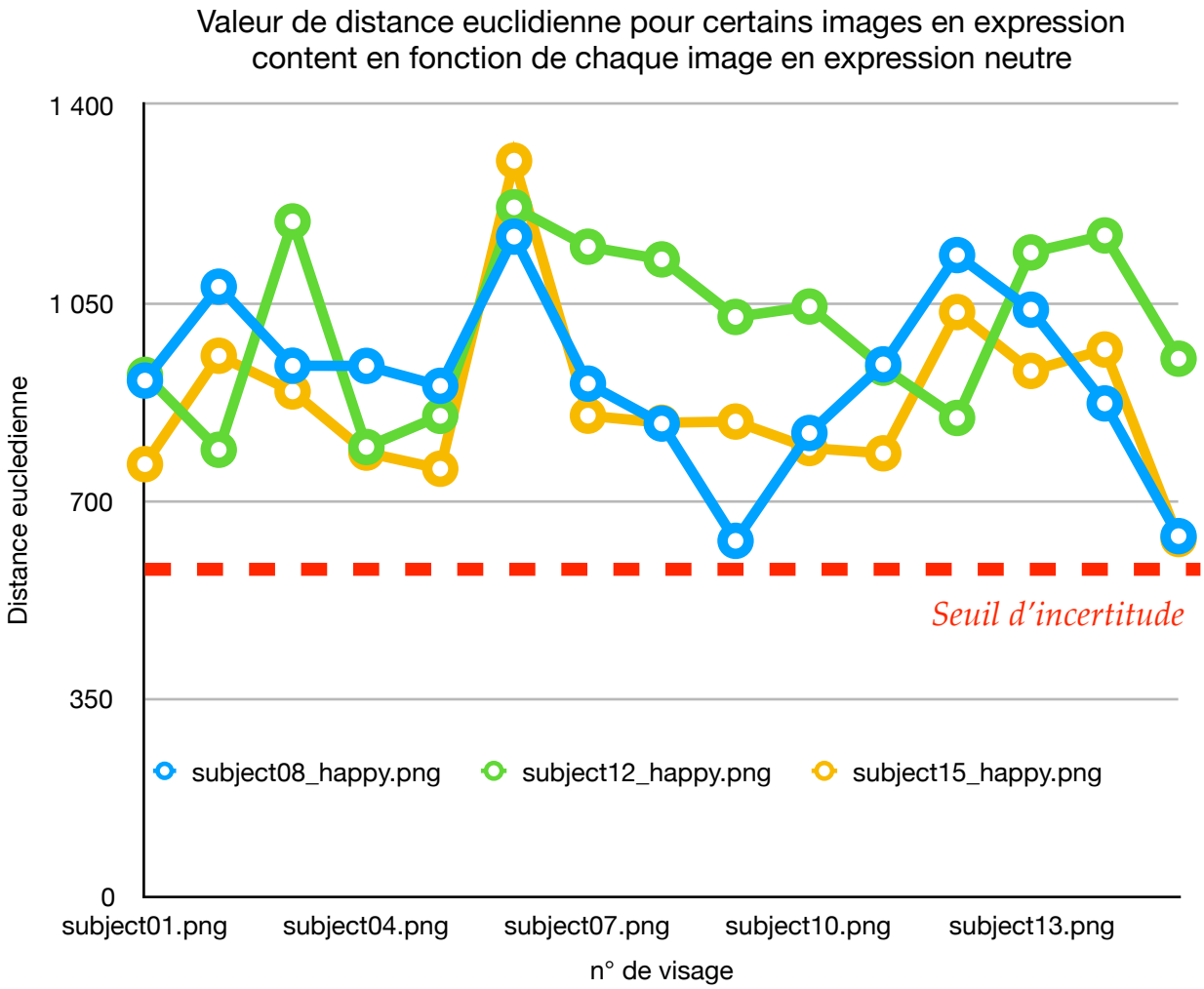

Figure 15 : graphique montrant la distance minimal euclidienne pour les visages 8, 12, et 15 en expression content, en fonction de chaque image en expression neutre.

Dans le figure 15, nous avons tracer plus les valeurs de la distance euclidienne pour les image qui ne donne pas un résultats correcte, à savoir le visage n° 8 et 12. Nous avons tracer celui de n°15 qui malgré qu'il donne un résultat correcte, reste très haut en valeur. On remarque que les courbes reste très hautes et ne descende pas en dessous de 629. Nous pouvons donc placer notre **seuil d'incertitude à 600**.

Conclusion

L'efficacité de l'algorithme LBPH est assez impressionnant compte tenu de la simplicité du calcul. Il est aussi pas très gourmand en ressource informatique, et assez robuste.

Néanmoins, plusieurs axes d'améliorations sont envisageables, notamment le fait que pour avoir des résultats significatives, il faut que les photos soit prises dans le même environnement. La définition plus fine des paramètres de rayon, et voisinage lors du calcul du paramétrage de la fenêtre pour le calcul de la LPB de l'image, ou bien de la longueur x, et la largeur y pour le paramétrage du grillage pour la calcul de l'histogramme regional, pourrait améliorer considérablement les résultats, même pour des photos prises dans des environnements différents.

Enfin, on remarque que l'algorithme se comporte de manière singulière pour les images qui ne donnent pas un résultats correcte, et ceci quelque soit l'image avec laquelle elle comparée. Ce qui nous laisse pensé que la définition des zones de confiance, d'incertitude, et de refus, serait probablement amélioré, en couplant l'analyse des valeurs euclidiennes avec une machine learning. Et non par la définition d'un seuil.

La conception de ce système nous a permis de comprendre les différents algorithmes existantes, ainsi que les différents possibilités qu'ils offraient. Il nous a aussi permis de comprendre l'importance du traitement de l'image dans la reconnaissance faciale, et les axes d'améliorations possible dans ce domaine de recherche.

Bibliographie

- Page Wikipedia : https://en.wikipedia.org/wiki/Local_binary_patterns

- Ahonen, Timo, Abdenour Hadid, and Matti Pietikainen. “Face description with local binary patterns: Application to face recognition.” IEEE transactions on pattern analysis and machine intelligence 28.12 (2006): 2037–2041.

- Ojala, Timo, Matti Pietikainen, and Topi Maenpaa. “Multiresolution gray-scale and rotation invariant texture classification with local binary patterns.” IEEE Transactions on pattern analysis and machine intelligence 24.7 (2002): 971–987.

- Ahonen, Timo, Abdenour Hadid, and Matti Pietikäinen. “Face recognition with local binary patterns.” Computer vision-eccv 2004 (2004): 469–481.

- LBPH OpenCV: https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html#local-binary-patterns-histograms

- Local Binary Patterns: http://www.scholarpedia.org/article/Local_Binary_Patterns

Annexe

Tableau des valeurs

Valeur des distances minimales en fonction de chaque image en expression content en fonction visage en expression neutre

	subject01_happy.png	subject02_happy.png	subject03_happy.png	subject04_happy.png	subject05_happy.png	subject06_happy.png	subject07_happy.png	subject08_happy.png	subject09_happy.png	subject10_happy.png	subject11_happy.png	subject12_happy.png	subject13_happy.png	subject14_happy.png	subject15_happy.png
subject01.png	477,80749	946,94984	899,77886	766,0248	748,29272	1442,8548	819,06532	912,21818	1017,5461	777,47669	806,41553	922,28412	850,34464	1190,5612	764,65286
subject02.png	935,37693	415,06867	1221,1093	714,74471	778,48699	1425,0993	1157,7331	1077,9824	1191,8087	1082,8426	955,30309	790,09367	1061,417	1416,5458	956,01151
subject03.png	878,81397	1249,848	354,07626	1107,8574	1022,6896	1450,2427	606,07755	938,48815	940,28187	815,89338	971,35575	1193,1974	828,58313	961,33241	893,03863
subject04.png	824,39554	720,04305	1115,2569	342,44416	706,99505	1409,4651	1053,6479	938,15884	1061,2822	975,93545	810,53686	794,97925	980,97706	1308,9836	785,02229
subject05.png	769,65447	820,04634	1005,6371	705,06312	334,66102	1409,9851	951,45152	902,97508	1031,6841	896,81213	781,96931	849,95529	826,32923	1273,8477	756,60426
subject06.png	1414,4108	1431,1876	1433,7475	1397,6573	1392,9494	362,16847	1463,9891	1166,2821	1158,9236	1388,1153	1398,9496	1218,069	1484,0263	1220,9693	1300,3569
subject07.png	852,35908	1223,5612	533,7209	1066,7165	1018,5657	1435,2373	552,81462	906,90573	916,5075	773,6136	930,29243	1148,5156	827,53489	957,39229	850,26702
subject08.png	854,51975	1197,5734	580,90619	1046,7922	989,73431	1375,9891	663,26616	836,33606	850,39873	754,66549	927,08468	1126,1501	838,80868	931,89055	837,83531
subject09.png	1001,2462	1231,5064	953,68758	1068,6243	1053,7533	1163,2695	990,44031	629,35364	342,00292	926,58729	991,35765	1024,5536	1067,5542	988,79927	839,89285
subject10.png	783,62491	1124,4857	792,62728	966,51746	921,40328	1370,0562	738,39014	819,66213	878,38944	425,36337	915,67243	1042,9324	908,34135	1083,0771	793,14438
subject11.png	758,46819	946,47028	976,55517	782,30834	780,93278	1439,2352	899,89999	940,85068	1011,1647	901,32236	496,95271	934,8766	869,01438	1219,9139	783,80355
subject12.png	1060,8374	769,87661	1289,1982	844,53301	907,28937	1386,0837	1245,8098	1133,7389	1237,4926	1171,0593	1046,3651	846,24346	1135,5175	1443,6475	1033,1012
subject13.png	855,39348	1087,4686	838,82418	971,9074	832,84572	1501,4493	781,19268	1037,389	1063,7989	889,01294	872,14104	1138,3901	324,17896	1120,9612	929,27714
subject14.png	1070,7465	1300,1285	974,84255	1163,5961	1138,8503	1118,	1022,2945	871,87728	822,23476	1044,5181	1080,4499	1168,4314	1085,0963	594,5452	967,06877
subject15.png	829,26956	1114,0189	789,73033	917,82133	900,79853	1256,7983	826,22878	637,65351	705,91926	812,57615	828,82447	950,86908	933,27488	984,35867	632,55672