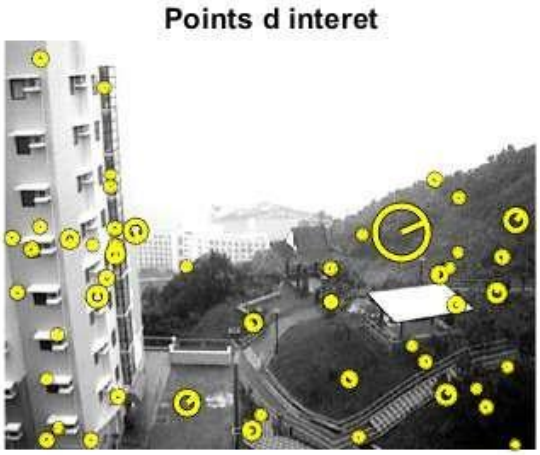
Dmytro Savchuk 15.10.21

TP : mosaique (mosaicing) avec les descripteurs SIFT.

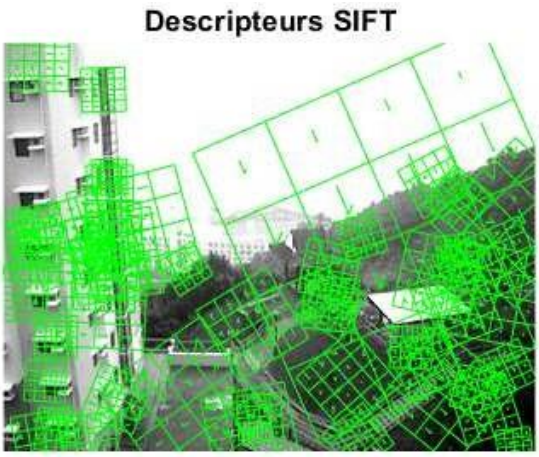
```
clear all; clc, close all

% On ouvre deux images.
im1 = imread('1.jpeg');
im2 = imread('2.jpeg');

% Transformation en format single (virgule flotante sur 2 octets).
im1 = im2single(im1) ;
im2 = im2single(im2) ;

% Transformation en images en niveau de gris si besoin.
% Si la troisième dimension de l'image est > 1 (pluseurs matrices ex:RGB) alors
% l'image n'est pas binaire ni niveau de gris. -> On la convertit en niveaux de gris
if size(im1,3) > 1
    im1g = rgb2gray(im1);
else
    im1g = im1 ;
end
if size(im2,3) > 1
    im2g = rgb2gray(im2) ;
else
    im2g = im2 ;
end
```

Calcul des SIFT et affichage.

```
% On cherche les points clés des deux images
% Calcule des frames et des descripteurs de SIFT de chaque image
% Chaque colonne de f1 ou f2 est la frame caractéristique
% Chaque colonne de d1 ou d2 est le descripteur local du cadre associé
% (Un descripteur est un vector de dimension 128 de class UINT8)
[f1,d1] = vl_sift(im1g) ;
[f2,d2] = vl_sift(im2g) ;

% Affichage des points clés de la première image
figure(1)
imshow(im1g)
hold on
% Randperm retourne un vecteur contenant une permutation random des entiers
% de 1 à la taille de la seconde dimension (colonne) de f1
perm = randperm(size(f1,2)) ;
% sel correspond aux 50 premiers entiers du vecteur
sel = perm(1:50) ;
% vl_plotframe affiche les formes géométriques données en paramètre
h1 = vl_plotframe(f1(:,sel)) ; % Affiche les frames caractéristiques
h2 = vl_plotframe(f1(:,sel)) ; % Affiche les frames caractéristiques
set(h1,'color','k','linewidth',3) ;
set(h2,'color','y','linewidth',2) ;
title('Points d interet');
```




```
drawnow ;

% Affichage de la seconde image
figure(2)
imshow(im1g)
hold on
h3 = vl_plotsiftdescriptor(d1(:,sel),f1(:,sel)) ;
set(h3,'color','g') ;
title('Descripteurs SIFT');
```




```
drawnow ;
```

Matching des descripteurs.

```
% Sur cette ligne, on pourra faire varier le parametre de threshold
% (seuillage) de la fonction pour voir son influence.
% Va chercher les matchs des deux sets de vecteurs des SIFTs
[matches, scores] = vl_ubcmatch(d1,d2) ; % Permet de faire correspondre les
% Descripteurs sift des deux images
% Retourne les correspondances ainsi que les distances euclidienne au carrées

numMatches = size(matches,2) ; % Récupère le nombres de correspondances

% On passe dans l'espace projectif.
% X1 correspond aux lignes 1 et 2 de f1 avec les colonnes correspondant aux matchs.
% La première ligne de matches correspond aux coordonnées des matches sur l'image1
% et la seconde aux coordonnées des matchs sur l'image 2
```

```
% On finit de compléter X1 et X2 par une ligne de 1
X1 = f1(1:2,matches(1,:)) ; X1(3,:) = 1 ;
X2 = f2(1:2,matches(2,:)) ; X2(3,:) = 1 ;
```

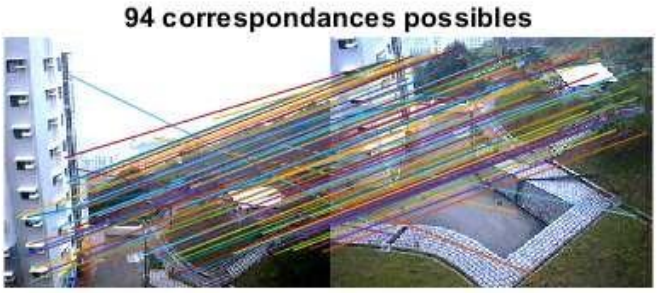
RANSAC

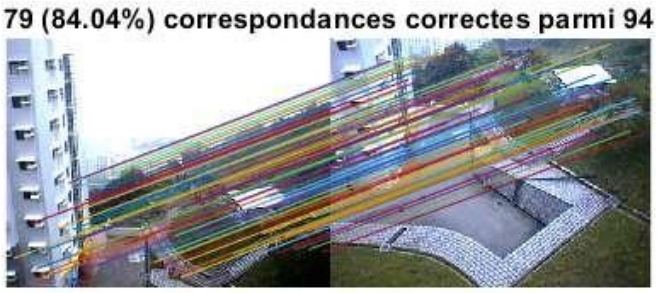
```
for t = 1:100
% Estimation de l'homographie en choisissant aleatoirement 4 mises en
% correspondance parmis celles trouvees.
% Retourne quatre colonne de numMatches -> du coup 4 valeurs de numMatches
  subset = vl_colsubset(1:numMatches, 4) ;
  A = [] ;
  for i = subset
% Pour chaque colonne du subset on concatène ces matrices
% cat concatène les trois matrics suivant la dimension 1 (ligne)
% kron est le produit de tenseur de kronecker
      A = cat(1, A, kron(X1(:,i)', vl_hat(X2(:,i)))) ;
% vl_hat renvoie la matrice symétrique asymétrique en prenant le vecteur 3D X2.
  end
% svd produit une matrice diagonale S de la dimension de A avec des
% éléments diagonaux non-négatifs en ordre décroissant. Et des matrices
% unitaires U et V telles que A = U*S*V
  [U,S,V] = svd(A) ; % Décomposition en valeur simple
  H{t} = reshape(V(:,9), 3, 3) ;

% Score de l'homographie (la variable 'ok' permet de controler la décomposition et les correspondance).
  X2_ = H{t} * X1 ;
  du = X2_(1,:)./X2_(3,:) - X2(1,:)./X2(3,:) ;
  dv = X2_(2,:)./X2_(3,:) - X2(2,:)./X2(3,:) ;
  ok{t} = (du.*du + dv.*dv) < 6*6 ;
% On peut prendre un autre nombre que 6*6, ce qui augmenterai ou baisserai le niveau de concordance
  score(t) = sum(ok{t}) ;
end

% Choix du meilleur score.
[score, best] = max(score) ;
H = H{best} ;
ok = ok{best} ;
```

Affichage de la mise en correspondance.

```
% Ici, on va afficher l'une à cote de l'autre deux images de tailles differentes.
% dh1 correspond max(nb lignes img2 - nb lignes img1) -> différence
% de auteur de la première image par rapport à la seconde
dh1 = max(size(im2,1)-size(im1,1),0) ;
dh2 = max(size(im1,1)-size(im2,1),0) ;

figure(3) ; clf ;
subplot(2,1,1) ;
% padarray:
imagesc([padarray(im1,dh1,'post') padarray(im2,dh2,'post')]) ;
o = size(im1,2) ;
line([f1(1,matches(1,:));f2(1,matches(2,:))+o], ...
     [f1(2,matches(1,:));f2(2,matches(2,:))]) ;
title(sprintf('%d correspondances possibles', numMatches)) ;
axis image off ;

% Ici, on affiche les deux images l'une à côté de l'autre
% Avec des lignes reliant les correspondances entre les deux images
% Mais cette fois-ci avec les correspondances que l'on a estimé non
% aberrantes avec le RANSAC
subplot(2,1,2) ;
imagesc([padarray(im1,dh1,'post') padarray(im2,dh2,'post')]) ;
o = size(im1,2) ;
line([f1(1,matches(1,ok));f2(1,matches(2,ok))+o], ...
     [f1(2,matches(1,ok));f2(2,matches(2,ok))]) ;
title(sprintf('%d (%.2f%%) correspondances correctes parmi %d', ...
              sum(ok), ...
```

```
            100*sum(ok)/numMatches, ...
            numMatches)) ;
axis image off ;
```






```
drawnow ;
```

Resultat final

```
% Calcul de limites d'affichage pour la mosaique.
box2 = [1  size(im2,2) size(im2,2)  1 ;
        1  1           size(im2,1)  size(im2,1) ;
        1  1           1            1 ] ;
box2_ = inv(H) * box2 ;
box2_(1,:) = box2_(1,:) ./ box2_(3,:) ;
box2_(2,:) = box2_(2,:) ./ box2_(3,:) ;
ur = min([1 box2_(1,:)]):max([size(im1,2) box2_(1,:)]) ;
vr = min([1 box2_(2,:)]):max([size(im1,1) box2_(2,:)]) ;

% On cree une grille de definition commune pour les deux images.
[u,v] = meshgrid(ur,vr) ;

% Ici, on calle l'image im1 sur cette nouvelle grille.
im1_ = vl_imwbackward(im2double(im1),u,v) ;

% On transforme im2 par H.
z_ = H(3,1) * u + H(3,2) * v + H(3,3) ;
u_ = (H(1,1) * u + H(1,2) * v + H(1,3)) ./ z_ ;
v_ = (H(2,1) * u + H(2,2) * v + H(2,3)) ./ z_ ;

% On utilise une interpolation bilinéaire car elle permet d'obtenir de meilleurs
% résultats que l'interpolation par plus proche voisin tout en restant de
% complexité raisonnable
im2_ = vl_imwbackward(im2double(im2),u_,v_) ;

% Comment moyenne-t-on les images au niveau du recollement ?
% Les images au niveau du recollement sont moyennées grâce à la fonction
% isnan soit une fonction qui transforme l'image en tableau de 1 et 0
% (suivant si la case est un chiffre ou non)
% Ainsi les 2 images de isnan des images sont additionnées et divisées à droite par
% l'inverse de isnan
mass = ~isnan(im1_) + ~isnan(im2_) ;
im1_(isnan(im1_)) = 0 ;
```

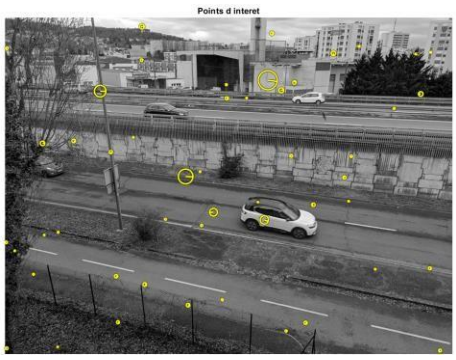
```
im2_(isnan(im2_)) = 0 ;
mosaic = (im1_ + im2_) ./ mass ;

figure(4) ; clf ;
imagesc(mosaic) ; axis image off ;
title('Mosaique') ;
```

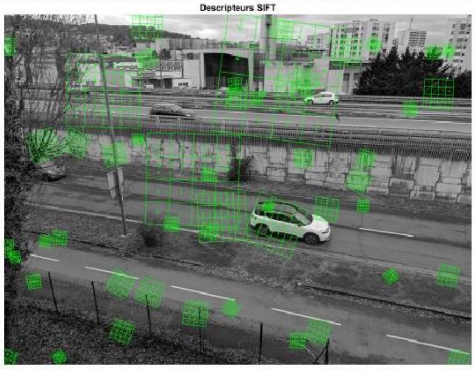
Tests des limites de l'algorithme dans différentes situations

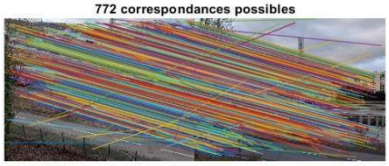
A) Test de l'algorithme avec le même type de photo que la démo (décalage verticalement)

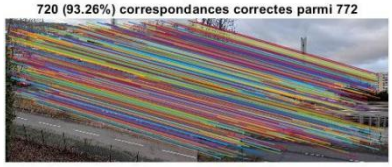
Photo 1

Photo 2







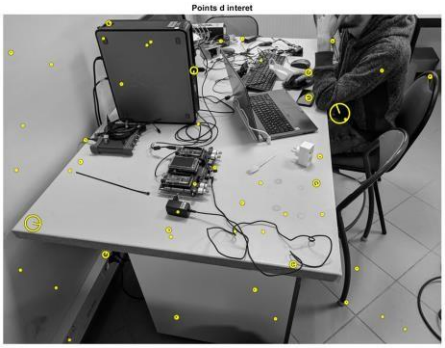


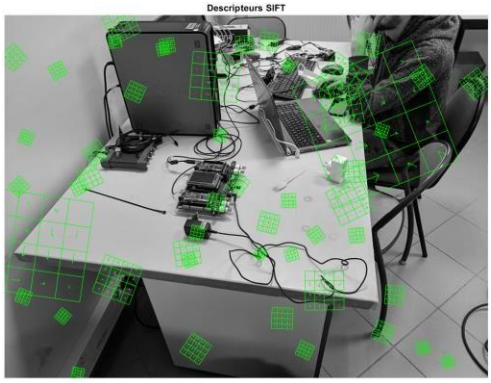
L'algorithme arrive parfaitement a récolé les images alors même que les voitures en mouvements présentent sur celle-ci créer des différences Ainsi l'algorithme arrive à prendre comme point d'intérêt des points fixes de l'image.

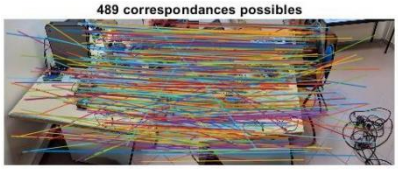
B) Test de l'algorithme avec deux photos décalées horizontalement

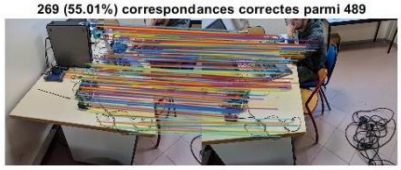
Photo 1

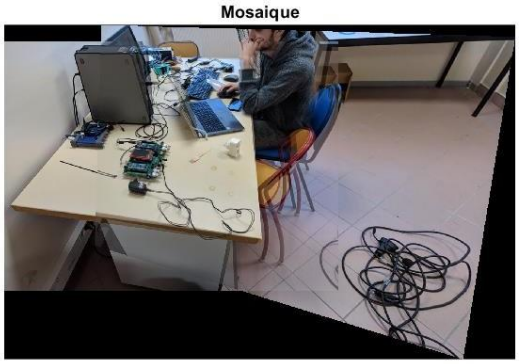
Photo 2











L'algorithme parvient à reconstituer l'image lorsqu'il y a un décalage sur le côté. Néanmoins, le collage est de bien moins bonne qualité et les concordances sont plus complexe du au changement d'angle trop fort.

C) Test de l'algorithme avec deux photos avec un fort décalage en angle

Photo 1

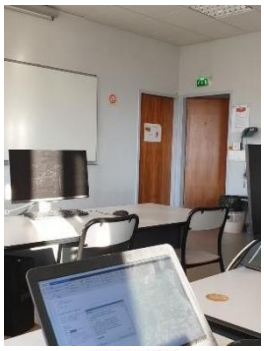
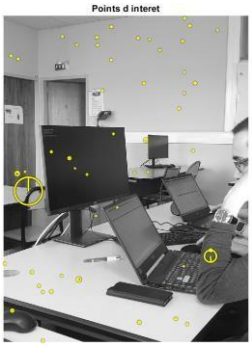
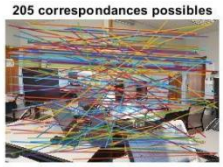
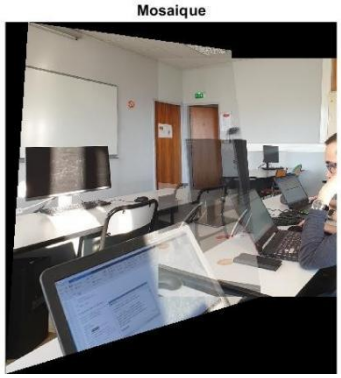


Photo 2





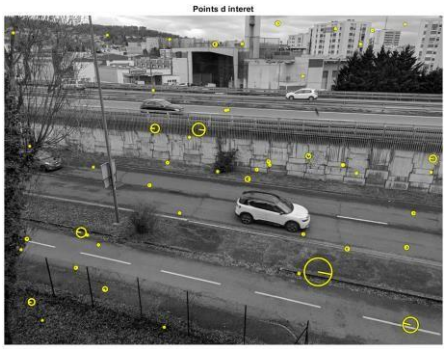


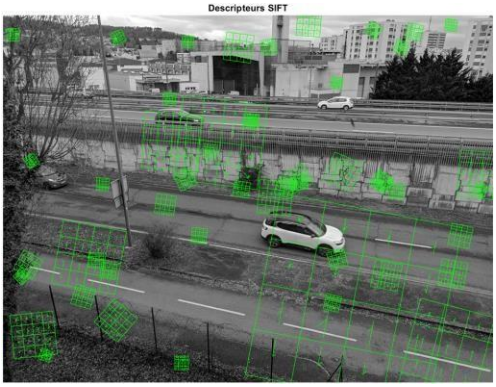
Lors d'un décalage trop prononcé entre les différentes images, le logiciel n'arrive pas à rassembler les photos efficacement. Ainsi il faut qu'il y ait au moins la moitié de l'image pour réaliser une mosaïque efficace.

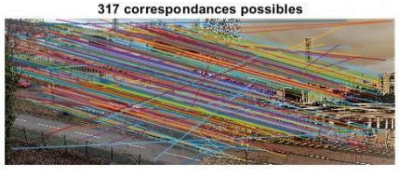
D) Test de l'algorithme avec une imagine en couleur et une image détériorer par un filtre (couleur)

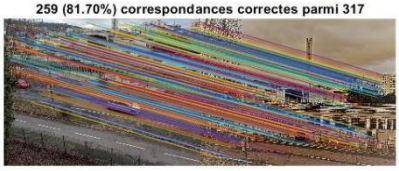
Photo 1

Photo 2









L'algorithme arrive avec un peu plus de difficulté à recréer la mosaïque par rapport au test A. En effet, la détérioration de l'image compromet les points de correspondance.

Pour conclure, cet algorithme est puissant pour permettre la superposition d'images et recréer la perspective entre deux images. Même avec des images de faible qualité et même en noir et blanc, il arrive à trouver des points d'intérêts sur les images et les associés. Il reste néanmoins efficace seulement lors de faible changement d'angle entre les deux images, une piste d'amélioration serait dans la capacité à comprendre les plus grand changement (pourquoi pas avec des représentations en 3D). De plus, nous pourrions améliorer cet algorithme en lui permettant de traiter plusieurs images en même temps.