Extraction d’avant-plan en temps réel sur une caméra en mouvement.

Étude de l’article "Moving Object Detection".

Test de différentes implémentations d’algorithmes de flux optique.

Table des matières

1 État de l’art. 2
1.1 Étude du papier Moving Object Detection. 2
1.2 Imiter le fonctionnement de la rétine : La solution de Prophesee: 3
1.3 Background Modelling from a Moving Camera 3
2 Équation du flux optique. 3
3 Détermination des vecteurs u, v. 4
4 Recherche de la matrice H 4
4.1 Avec les vecteurs de déplacement fourni. 4
4.2 Signification des coefficients de la matrice H. 6
4.3 Détermination expérimentale de la matrice d’homographie avec SIFT. 6
4.3.1 Mais qu’est-ce qu’un point ? 6
4.3.2 Images à analyser. 7
4.3.3 Recherche des points d’intérêts. 7
4.3.4 Recherche et filtrage des correspondances. 7
4.3.5 Recherche de la matrice H. 8
4.3.6 Correction de l’image par H et extraction de l’arrière-plan. 8
4.4 Étude avec ORB. 9
4.4.1 Comparaison de l’extraction de fond avec SIFT et ORB : 10

4.5 Extraction de l'arrière-plan – retour à l'étude du papier. 11
4.5.1 Déplacement de l'avant plan sans effet de zoom. 11
4.5.2 Déplacement de l'avant-plan avec effet de zoom. 12
4.5.3 Comment déterminer la nature du mouvement (effet de zoom ou non). 12
5 Algorithme proposé. 14
6 Note à propos des mouvements de rotation. 14
7 Conclusion de l'étude réalisée sur le papier. 14
7.1.1 Utilisation de FlowNet2.0 pour calculer les vecteurs du flux optique. 14
7.1.2 Le calcul de la matrice d'homographie. 14
7.1.3 Choix de l'algorithme (calcul de idx). 14
7.1.4 Déterminer les points d'avant-plan. 15
7.1.5 En résumé. 15
8 Annexes. 15
8.1 Code des différents programmes développé pour l'étude: 15
8.1.1 mouvement_objet_camera.ipynb : 15
8.1.2 extract_avant-plan_sift.py 15
8.1.3 extract_avant-plan_orb.py 15
8.2 Papiers scientifiques: 15
8.3 Étude de la documentation de la caméra STV 0991. 15

1 État de l'art.

L'extraction de l'avant-plan est un problème qui se pose très régulièrement en vision par ordinateur. Dans le papier qui sera étudié, l'utilisation du flux optique est à la base de la méthode. D'autres méthodes s'appuyant sur d'autres techniques permettent d'obtenir ce résultat.

L'une de ces méthodes est une solution de détection qui s'appuie sur un capteur "intelligent". La solution est donc "hardware" et fonctionne à des vitesses totalement inaccessibles aux solutions "softwares". C'est la solution proposée par la société Prophesee.

La plupart des méthodes logicielles n'utilise pas l'équation du flux optique mais sont basées sur la reconnaissance de points particuliers. On recherche alors ces points sur les images et on en déduit le déplacement d'un point par un simple calcul de différence sur les coordonnées.

Enfin nous disposons de caméra STV0991 de la société STMicroelectronics qui sont capables, par une méthode de reconnaissance de points, de sortir des vecteurs de flux optique s'appliquant sur ces points particuliers.

1.1 Étude du papier Moving Object Detection.

Le principe de cette étude est, à partir du flux optique (extrait grâce au réseau de neurone FlowNet 2.0) de calculer la matrice d'homographie par la relation : $H_{t\to t-k} * P_t = P_t + F_{t,t-k.}$

- $H_{t\to t-k}$ est la matrice d'homographie permettant de passer de l'instant t à t-k,

- P_t l'ensemble de point représentant l'image à l'instant t,
- $F_{t, t-k}$ l'ensemble des vecteurs (u, v, 0) qui à chaque points (x, y, 1) de l'ensemble P_t fait correspondre un point dans l'ensemble P_{t-k} de coordonnée (x+u, y+v, 1).

Par la suite, et en fonction du type de mouvement, le papier propose une modélisation de l'arrière-plan. Le fait d'utiliser une matrice d'homographie permettra d'extraire un arrière-plan même lorsque la caméra est en mouvement.

1.2 Imiter le fonctionnement de la rétine : La solution de Prophesee:

La technologie développée par Prophesee (PROPHESEE | Metavision for Machines) s'inspire de la rétine biologique. Plutôt que d'acquérir un flux vidéo image par image comme sur les caméras classiques, les pixels fonctionnent comme les photorécepteurs de la rétine. Ils sont donc indépendants et fonctionnent de façon asynchrone en délivrant un signal seulement si un changement s'est produit. Puisque un pixel produit un signal qu'en cas de changement, l'arrière-plan est acquis une et une seule fois.

Ce principe permet de réduire considérablement la taille des données à acquérir. C'est un processus analogique qui se charge du travail avec une précision temporel de l'ordre de quelques microsecondes. Avec cette vitesse d'acquisition, tout se passe comme si la caméra pouvait filmer à 10.000 voire 100.000 images par seconde. Cette vitesse d'acquisition est dépendante de la quantité de pixel ayant changé d'une image à l'autre.

Une telle vitesse d'acquisition peut avoir de nombreuse application dans le monde:

- Repérer visuellement des vibrations hautes fréquences sur une pièce mécanique (source de panne future voir de casse machine). Mais le même système repérera aussi efficacement les vibrations basses fréquence (qui peuvent aussi être source de problèmes).
- Comptage à haute vitesse d'éléments.
- Dans le domaine de l'aide à la conduite, détection très précoce d'objets ou personnes et évaluation rapide de leurs trajectoires ; ce qui permet de prévoir le plus tôt possible les risques de collision.
- Dans le domaine médical, et comme le principe même du fonctionnement de cette technologie s'inspire du fonctionnement de la rétine, l'utilisation de cette technologie pourrait permettre à des non-voyants de recouvrer la vue (partenariat avec la société Pixium vision - fabricant d'implant rétinien).

1.3 Background Modelling from a Moving Camera

Toujours dans une perspective d'utilisation de l'optical flow en temps réel, on peut citer le papier proposé lors du "Second International Symposium on Computer Vision and the Internet (VisionNet'15) " proposé par Amitha Viswanatha, Reena Kumari Beherab, Vinuchackravarthy Senthamilarasub et Krishnan Kuttyb.

Ici les auteurs proposent (comme pour le papier de cette étude) d'utiliser la matrice d'homographie pour corriger l'image et extraire ainsi l'avant-plan

2 Équation du flux optique.

Voir OpenCV : Flux optique .

Le flux optique fonctionne sur 2 hypothèses :

- Les intensités en pixels d'un objet ne changent pas entre les images consécutives.

- Les pixels voisins ont un mouvement similaire.

On considère la fonction f(x,y,t) représentant un pixel de la première image. Ce pixel se déplace de dx, dy dt. Puisque l'intensité d'un pixel ne change pas (d'après la 1ère hypothèse du flux optique) on peut écrire :

f(x,y,t)= f(x+dx,y+dy,t+dt)

Si on effectue un développement limité d'ordre 1 (formule de Taylor) au 2ème membre de cette équation on obtient :

$$f(x,y,t) = f(x,y,t) + \frac{\partial f(x,y,t)}{\partial x}dx + \frac{\partial f(x,y,t)}{\partial y}dy + \frac{\partial f(x,y,t)}{\partial t}dt + \in(0) \quad \text{avec} \lim_{0} \in(0) = 0.$$

En divisant les 2 membres par d_t et en posant $u = \frac{dx}{dt}$, $v = \frac{dy}{dt}$, $f_x = \frac{\partial f}{\partial x}$, $f_y = \frac{\partial f}{\partial y}$ et $f_t = \frac{\partial f}{\partial t}$ on obtient l'équation du flux optique :

$$f_x u + f_y v + f_t = 0$$

Comme nous avons 1 seule équation et 2 inconnus (u et v) celle-ci ne peut être résolu sans prendre en compte la 2ème hypothèse. La méthode de Lucas-Kanade propose de prendre une matrice de 3 par 3. Ainsi nous avons un système de 9 équations pour 2 inconnues. Ce système est surdéterminé et ne possède donc pas nécessairement de solution exacte. La meilleure des solutions est obtenu par la méthode des moindres carrés.

3 Détermination des vecteurs u, v.

Comme conseillé dans le papier scientifique nous utiliseront un réseau de neurone pour déterminée ces vecteurs. Il s'agit de Flownet2, dont le code provient du github de Clément Pinard :

ClementPinard/FlowNetPytorch: Pytorch implementation of FlowNet by Dosovitskiy et al. (github.com)

4 Recherche de la matrice H

4.1 Avec les vecteurs de déplacement fourni.

Si nous avons les vecteurs (u, v, 0) fourni par un algorithme (par exemple flownet2.0) nous pouvons calculer la matrice H.

La matrice H comprend 9 paramètres que l'on peut réduire à 8 en divisant l'ensemble des termes par le dernier coefficient de la matrice. La matrice s'écrit alors :

$$\begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{pmatrix}$$

L'équations à résoudre pour trouver la matrice H est :

$$H_{t,t-k} \times P_t = P_t + F_{t,t-k}$$

Nous aurons besoin de 4 points (donc 12 coordonnées) appartement à l'image de fond. En effet, le premier plan ayant un mouvement propre, ses points ne respectent pas l'équation). Soit p_1 (x_1,y_1,1), p_2 (x_2,y_2,1), p_3 (x_3,y_3,1) et p_4 (x_4,y_4,1) 4 points de l'arrière-plan; calculons le premier terme de l'équation $H_{t,t-k} * P_t$ avec ces points:

$$\begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{pmatrix} \times \begin{pmatrix} x_1 x_2 x_3 x_4 \\ y_1 y_2 y_3 y_4 \\ 1\ 1\ 1\ 1 \end{pmatrix} =$$

$$\left(\begin{array}{c|c|c|c} x_1.h_{11}+y_1.h_{12}+h_{13} & x_2.h_{11}+y_2.h_{12}+h_{13} & x_3.h_{11}+y_3.h_{12}+h_{13} & x_4.h_{11}+y_4.h_{12}+h_{13} \\ x_1.h_{21}+y_1.h_{22}+h_{23} & x_2.h_{21}+y_2.h_{22}+h_{23} & x_3.h_{21}+y_3.h_{22}+h_{23} & x_4.h_{21}+y_4.h_{22}+h_{23} \\ x_1.h_{31}+y_1.h_{32}+1 & x_2.h_{31}+y_2.h_{32}+1 & x_3.h_{31}+y_3.h_{32}+1 & x_4.h_{31}+y_4.h_{32}+1 \end{array}\right)$$

Le second terme est :

$$\left(\begin{array}{c|c|c|c} x_1+u_1 & x_2+u_2 & x_3+u_3 & x_4+u_4 \\ y_1+v_1 & y_2+v_2 & y_3+v_3 & y_4+v_4 \\ 1 & 1 & 1 & 1 \end{array}\right)$$

Où (u_n, v_n, 0) sont les coordonnées du vecteur de déplacement des points P_n (x_n, y_n, 1). Ceci nous permet d'arriver au système d'équation ci-dessous. Le système sera ordonné par colonne pour faire apparaître un système matriciel de type A.X= B ou X représente les inconnus (dans notre cas les coefficients h_{xx} de la matrice) :

$$\begin{cases} h_{11}.x_1 + h_{12}.y_1 + h_{13} + 0 + 0 + 0 + 0 + 0 = x_1 + u_1 \\ 0 + 0 + 0 + h_{21}.x_1 + h_{22}.y_1 + h_{23} + 0 + 0 = y_1 + v_1 \\ 0 + 0 + 0 + 0 + 0 + 0 + h_{31}.x_1 + h_{32}.y_1 = 0 \\ h_{11}.x_2 + h_{12}.y_2 + h_{13} + 0 + 0 + 0 + 0 + 0 = x_2 + u_2 \\ 0 + 0 + 0 + h_{21}.x_2 + h_{22}.y_2 + h_{23} + 0 + 0 = y_2 + v_2 \\ 0 + 0 + 0 + 0 + 0 + 0 + h_{31}.x_2 + h_{32}.y_2 = 0 \\ h_{11}.x_3 + h_{12}.y_3 + h_{13} + 0 + 0 + 0 + 0 + 0 = x_3 + u_3 \\ 0 + 0 + 0 + h_{21}.x_3 + h_{22}.y_3 + h_{23} + 0 + 0 = y_3 + v_3 \\ 0 + 0 + 0 + 0 + 0 + 0 + h_{31}.x_3 + h_{32}.y_3 = 0 \\ h_{11}.x_4 + h_{12}.y_4 + h_{13} + 0 + 0 + 0 + 0 + 0 = x_4 + u_4 \\ 0 + 0 + 0 + h_{21}.x_4 + h_{22}.y_4 + h_{23} + 0 + 0 = y_4 + v_4 \\ 0 + 0 + 0 + 0 + 0 + 0 + h_{31}.x_4 + h_{32}.y_4 = 0 \end{cases}$$

On peut alors écrire le système sous forme matriciel et le résoudre.

A= $\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x_1 & y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x_2 & y_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x_3 & y_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x_4 & y_4 \end{pmatrix}$, X= $\begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{pmatrix}$ et B= $\begin{pmatrix} x_1 + u_1 \\ y_1 + v_1 \\ 0 \\ x_2 + u_2 \\ y_2 + v_2 \\ 0 \\ x_3 + u_3 \\ y_3 + v_3 \\ 0 \\ x_4 + u_4 \\ y_4 + v_4 \\ 0 \end{pmatrix}$

Ce système étant surdéterminé avec des équations linéairement indépendantes les unes des autres, il est peu probable qu'il admette des solutions exactes. On utilisera alors les "meilleures" solutions en utilisant la méthode des moindres carrés sur les différentes équations. On rappelle qu'un choix de 8 parmi 12 génère $\frac{12!}{(12-8)!\,8!} = 495$ possibilités.

4.2 Signification des coefficients de la matrice H.

Dans un mouvement de translation de la caméra, la matrice est de la forme : $\begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{pmatrix}$ ou d_x et d_y sont les déplacements suivant les axes x et y.

Dans un mouvement de rotation de la caméra, la matrice est de la forme : $\begin{pmatrix} cos\,\theta & -sin\,\theta & 0 \\ sin\,\theta & cos\,\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$ ou θ est l'angle de rotation de la caméra.

Les coefficients de la matrice ne peuvent donc pas prendre n'importe quelle valeur contrairement au vecteur (u,v) issue du réseau de neurones.

Note à propos des vecteur u,v : Une des hypothèses est que l'image soit "plane", c'est-à-dire sans profondeur. Quel que soit le point de l'arrière-plan considéré, si la caméra effectue un mouvement de translation de (u,v) il bougera lui aussi de (u,v). La matrice sera donc : $\begin{pmatrix} 1 & 0 & u \\ 0 & 1 & v \\ 0 & 0 & 1 \end{pmatrix}$. Mais dans un cas réel, le mouvement apparent des objets dépend de la distance à la caméra. Cette disparité peut très bien se traduire lorsque l'on exprime les vecteurs de déplacement de façon individuel ce que ne permet pas la matrice H. D'autre part, la caméra induit des déformations de l'images ce qui ne peut être pris en compte par la matrice.

4.3 Détermination expérimentale de la matrice d'homographie avec SIFT.

Il existe plusieurs algorithmes pour rechercher des points identiques sur 2 images, nous choisissons d'utiliser "SIFT".

Une fois que les points sont déterminés, le principe pour déterminer la matrice est le suivant :

1. Déterminé des points "intéressant" dans la première image.
2. Déterminé des points "intéressant" dans la seconde image.
3. Déterminer une série de points commun entre les 2 images.
4. Choisir 4 de ces points correctement répartis dans l'image.
5. Calculer la matrice H.
6. Recommencer au point 4 jusqu'à une converge suffisante des résultats.

Toute la question est de savoir ce qu'es un point "intéressant" et wikipédia y répond merveilleusement bien (Scale-invariant feature transform — Wikipédia (wikipedia.org)).

En résumé, il s'agit de zone de l'image choisi pour qu'elle soit autant que possible invariant quel que soit l'échelle à laquelle on la regarde. Et c'est bien parce que l'on est capable de repérer les mêmes points dans les 2 images que l'on peut ensuite calculer la matrice H.

4.3.1 Mais qu'est-ce qu'un point ?

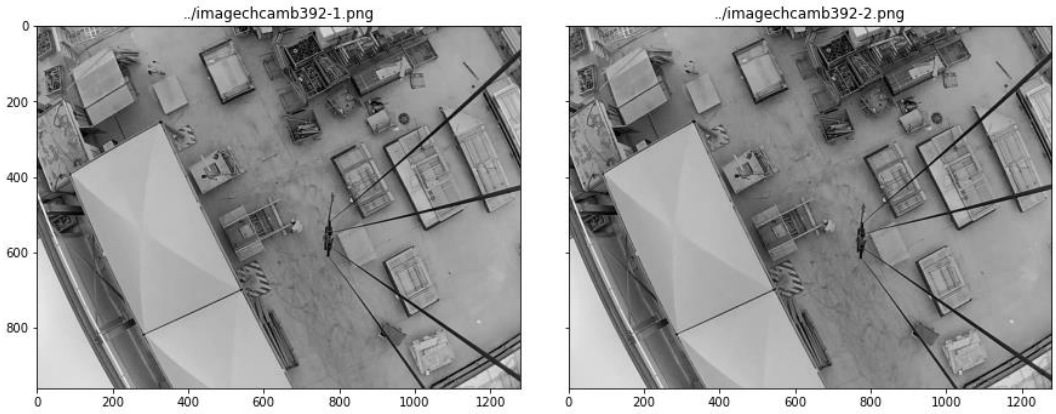
On pense souvent à un pixel. L'algorithme SIFT (comme ses dérivés) travaille sur des images en 256 niveaux de gris. Autant dire que sur une image au format classique de 720p il existe un certain nombre de pixels ayant la même intensité mais qui n'appartiennent pas au même "objet"…. La signification de point n'est donc pas synonyme de pixel. Un point est donc un ensemble de pixels ayant des caractéristiques bien particulière. Ces caractéristiques sont fonction de l'algorithme choisie : SIFT, SURF, ORB, … . C'est une différence fondamentale entre le résultat produit par FlowNet, qui associe à tous groupes de points composés de 4 pixels un vecteur, et les méthodes

basées sur la reconnaissance de points qui vont rechercher des caractéristiques particulières pour caractériser certains points composés de plusieurs pixels (quelques dizaines).

4.3.2 Images à analyser.

Nous avons ici 2 images séparées de 392 ms provenant d'un chantier. Le chariot est en mouvement dont la direction est du bas de l'image vers le haut.

```
fichier_image_1, fichier_image_2, _= nom_fichier(npy= "imagechcamb392-flow.npy")

img1= cv2.imread(fichier_image_1, cv2.IMREAD_GRAYSCALE) # , cv2.IMREAD_GRAYSCALE
img2= cv2.imread(fichier_image_2, cv2.IMREAD_GRAYSCALE) #cv2.IMREAD_COLOR

lst_images= [(img1, fichier_image_1), (img2, fichier_image_2)]
affichage_images(liste_images= lst_images, ligne_colonne= (1, 2), partage_xy= (False, True), haut_larg= (15,10),
                 ecart= (0.1, 0.2), carte_couleur= "gray")
```



4.3.3 Recherche des points d'intérêts.

La recherche des points d'intérêt avec SIFT. Pour plus d'informations sur SIFT voir https://www.geeksforgeeks.org/sift-interest-point-detector-using-python-opencv/ .

```
# création du descripteur SIFT.
sift = cv2.SIFT_create()
# Recherche des point clé dans les 2 images
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)
```

4.3.4 Recherche et filtrage des correspondances.

Une fois les points détectés dans chacune des images on va rechercher les correspondances en utilisant FLANN (Fast Library for Approximate Nearest Neighbours).

FLANN est une bibliothèque permettant d'effectuer des recherches rapides et approximatives des plus proches voisins dans des espaces de grande dimension. Elle contient une collection d'algorithmes efficaces pour la recherche du plus proche voisin et un système pour choisir automatiquement le meilleur algorithme et les paramètres optimaux en fonction de l'ensemble de données.

```
FLANN_INDEX_KDTREE= 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 10)
flann = cv2.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1, des2, k=2)

# Chaque point clé de la première image est mis en correspondance avec un certain nombre de points clés de la
#  seconde image. Nous gardons les 2 meilleures correspondances pour chaque point clé; (meilleures correspondances =
#  celles avec la plus petite mesure de distance).
#  Le test de Lowe vérifie que les deux distances sont suffisamment différentes. Si ce n'est pas le cas, le point clé
#  est éliminé et ne sera pas utilisé pour d'autres calculs car il est considéré comme non fiable.
good = []
coef= 0.7
for m,n in matches:
    if m.distance < coef * n.distance:
        good.append(m)
```

4.3.5 Recherche de la matrice H.

Il ne reste plus qu'à regrouper les différents points puis à utiliser la méthode findHomography pour trouver H.

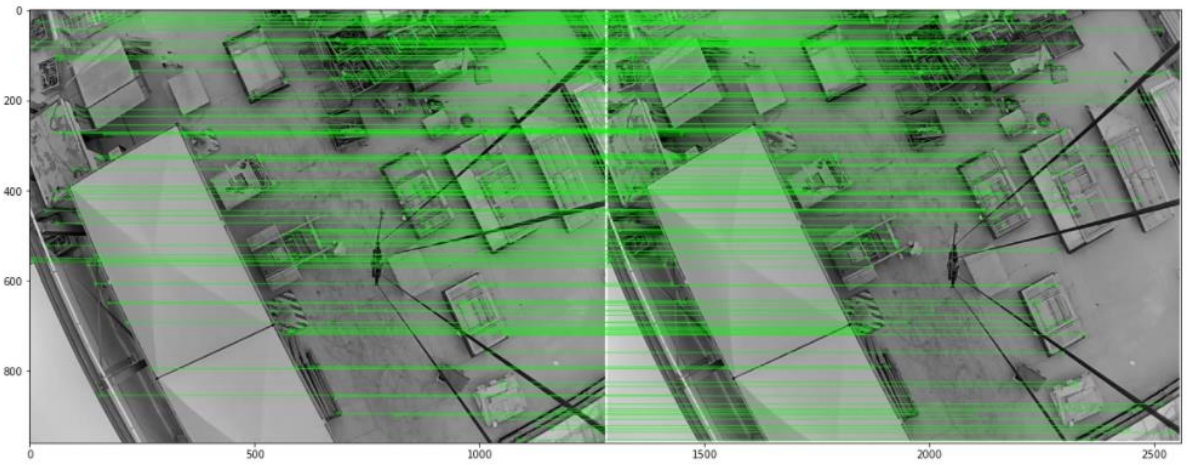
```
MIN_MATCH_COUNT= 10
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
    H, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC,5.0)
    matchesMask = mask.ravel().tolist()
else:
    print("Pas assez de correspondance !")
    matchesMask = None

print(np.round(H,2))
```

```
[[ 1.   -0.    2.39]
 [-0.    1.   -1.96]
 [ 0.    0.    1.  ]]
```

Comme on peut le voir la matrice indique qu'il y a eu un déplacement de 2.39 unités suivant l'axe des x et de -1.96 suivant l'axe y (pour rappel : le point de coordonnée (0,0) se trouve en haut à gauche de l'image). La valeur négative indique que la caméra s'est déplacée vers l'avant et donc, de son point de vue, c'est l'arrière-plan qui avance vers la caméra (d'où la valeur négative)!

Voici les 2 images et les points clé (200 choisi au hasard parmi plus de 12000 trouvé) sur les 2 images ayant permis le calcul de H.

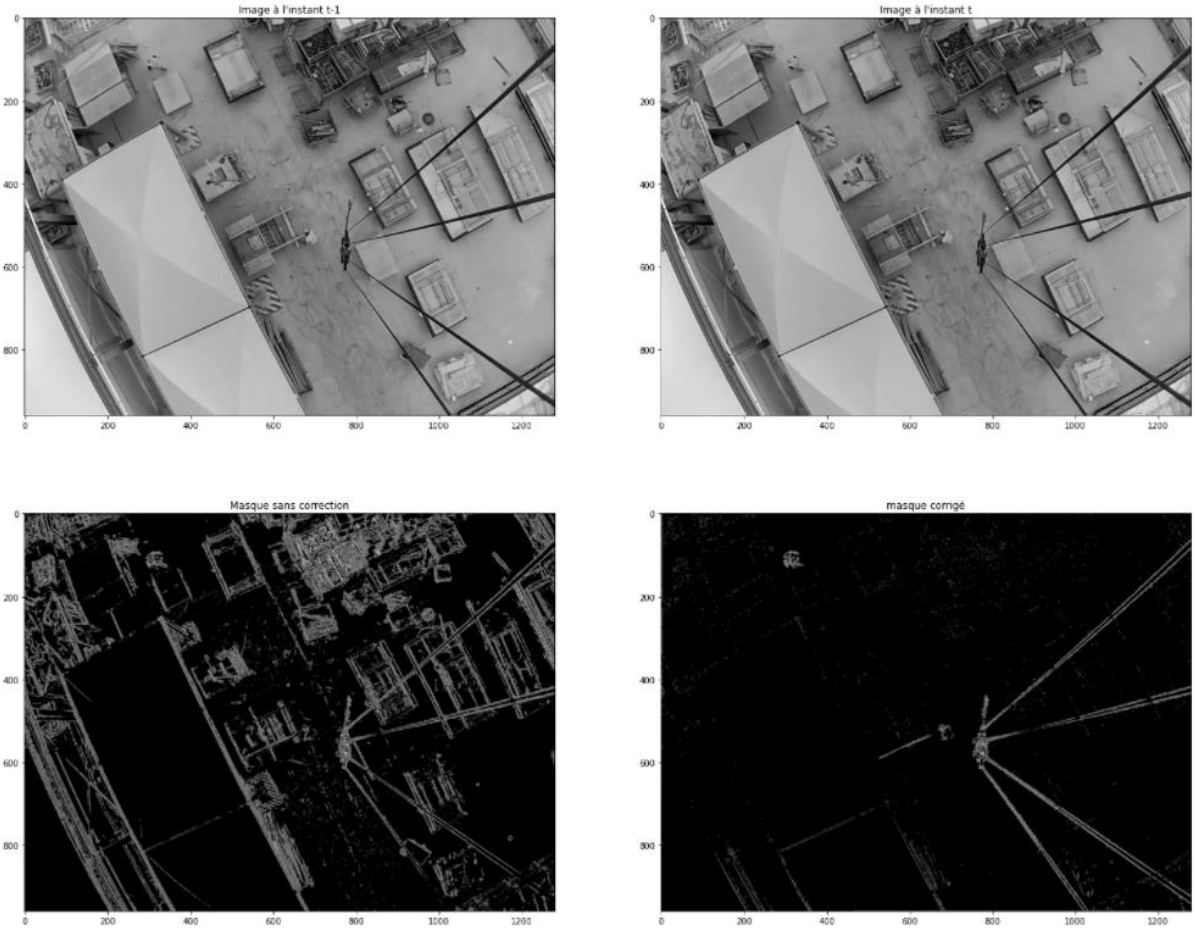
```
affiche_pointscle_img1_img2(nombre_point= 200, graine= 51165, liste_points= good, masque= matchesMask, image1= img1,
                            kp1= kp1, image2= img2_entoure, kp2= kp2, taillefig= (20,15))
```



4.3.6 Correction de l'image par H et extraction de l'arrière-plan.

Après avoir calculer la matrice H, il est possible de corriger la première image pour éliminer le mouvement de caméra et d'avoir ainsi une bonne représentation de l'avant-plan.

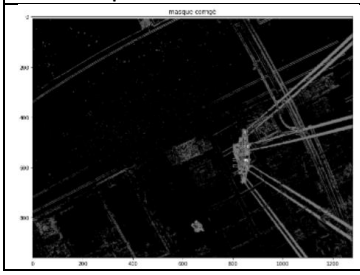
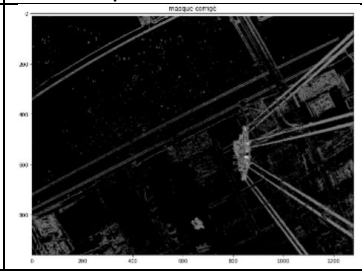
```python
# Correction de l'image
img1_corr= cv2.warpPerspective(img1, H, (img1.shape[1], img1.shape[0]))

# création des instances qui seront appliquées à l'extracteur
fgbg = cv2.createBackgroundSubtractorMOG2()
fgbg_c = cv2.createBackgroundSubtractorMOG2()

# Extraction sans correction
fgmask = fgbg.apply(img1)
fgmask = fgbg.apply(img2)

# Extraction avec correction
fgmask_c = fgbg_c.apply(img1_corr)
fgmask_c = fgbg_c.apply(img2)

affichage_images(liste_images= [(img1_corr, "Image à l'instant t-1"), (img2, "Image à l'instant t"),
                                (fgmask, "Masque sans correction"), (fgmask_c, "masque corrigé") ],
                 ligne_colonne= (2, 2), haut_larg= (25, 20), carte_couleur= "gray", ecart=(0.2,0.2))

```



4.4 Étude avec ORB.

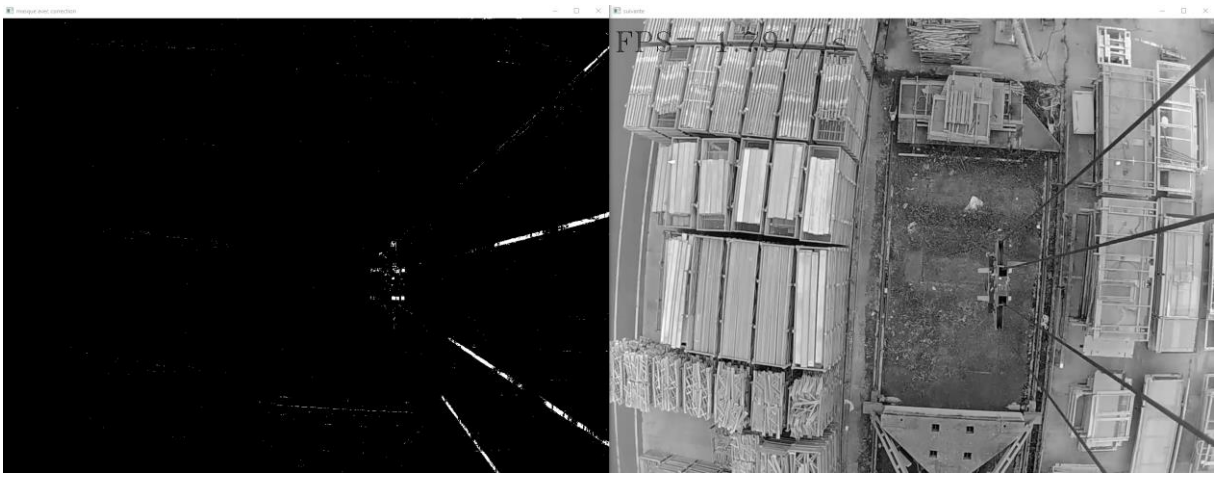
ORB est basé sur les algorithmes FAST et BRIEF qu'utilise la caméra. Les tests ont montré qu'ORB est moins performant que SIFT mais est réputé plus rapide. En effet SIFT est plus complexe dans la détermination des points puisqu'il va en choisir pour qu'il soit invariant au changement d'échelle ; on rappelle que SIFT est l'acronyme de Scale Invariant feature transform. Les caractéristiques de ces points étant plus complexe, les calculs générés sont plus lourds.

4.4.1 Comparaison de l'extraction de fond avec SIFT et ORB :

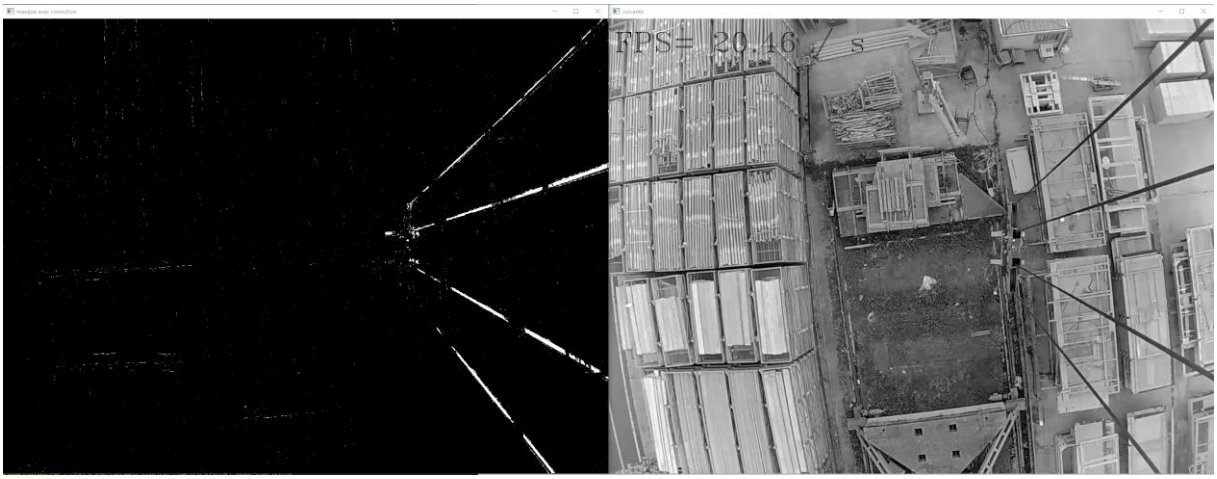
Arrière-plan correction SIFT:	Arrière-plan sans correction:	Arrière-plan correction ORB:
		

Un autre algorithme qui combine les avantages de SIFT et de ORB n'est malheureusement pas dans le domaine public ; il s'agit de SURF.

Mais si SIFT est meilleur qu'ORB c'est au prix d'un calcul beaucoup plus lourd. En effet comme le montre les images ci-dessous, avec un core I7 sur une vidéo en 1280×960 le programme basé sur SIFT peine à atteindre les 2 images par seconde.



Avec le programme basé sur ORB on dépasse sans problème les 20 images par seconde avec des pointe à 24/25.



4.5 Extraction de l'arrière-plan – retour à l'étude du papier.

Dans le papier étudié, 2 cas sont étudiés :

- Image sans effet de zoom : c'est-à-dire que l'avant plan évolue de façon perpendiculaire à l'axe de la caméra.
- Image avec effet de zoom : L'avant-plan se déplace essentiellement dans la profondeur, c'est-à-dire dans l'axe la caméra.

4.5.1 Déplacement de l'avant plan sans effet de zoom.

Puisque nous avons la matrice H qui s'applique à tout l'arrière-plan on peut écrire pour tous les points de l'arrière-plan:

$\widetilde{f}_{t\to t-k} = H_{t\to t-k} \times p_t - p_t$ où $\widetilde{f}_{t\to t-k} = [\tilde{u} \quad \tilde{v} \quad 0]^T$ représente le vecteur du flux optique de l'arrière-plan.

Cette équation amène plusieurs remarques :

- Étant donné sa nature La multiplication des coordonnées d'un point par la matrice H revient (dans le cas de mouvement de translation) à ajouter d_x à x et d_y à y. En effet dans un mouvement de translation, la matrice est de la forme $\begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{pmatrix}$ Donc tous les vecteurs sont identiques et égaux respectivement à d_x pour u et d_y pour v. Mais ceci suppose une image plane (sans profondeur) ce qui ne correspond pas à la réalité. Il faudra garder à l'esprit qu'il s'agit d'une approximation, et que si l'image à beaucoup de profondeur, les résultats risquent d'être mauvais.
- Pour un mouvement de rotation de la caméra, ou dans un mouvement combiné de rotation translation, chaque vecteur peut s'écrire : $\begin{matrix} \tilde{u} = x\,.\,cos(\theta) - y\,.\,sin(\theta) + d_x \\ \tilde{v} = x\,.\,sin(\theta) + y\,.\,cos(\theta) + d_y \end{matrix}$

La formule proposée pour la création d'un masque de l'avant plan est :

$M_t = \{Points \mid d_v > T_a\},\ d_v = \left\| f_{t,t-k} - \tilde{f}_{t,t-k} \right\|_2$ c'est-à-dire l'ensemble des points vérifiant l'inégalité $d_v > T_a$; d_v est la norme euclidienne de la différence entre le vecteur (u,v,0) du point considéré et le vecteur du flux optique de l'arrière-plan $(\tilde{u}, \tilde{v}, 0)$: Lorsque nous avons un mouvement de translation sans rotation, cette équation peut s'écrire :

$$d_v = \sqrt[2]{(u - d_x)^2 + (v - d_y)^2}\,.$$

T_a est un seuil défini par :

$$T_a = a_1 + a_2 \times \sqrt[2]{(H_{t\to t-k}\,(1,3))^2 + (H_{t\to t-k}\,(2,3))^2}$$

où $H_{t\to t-k}\,(1,3)$ et $H_{t\to t-k}\,(2,3)$ représente respectivement le coefficient de la 1ère ligne, 3ème colonne et celui de la 2ème ligne 3ème colonne de la matrice H c'est-à-dire d_x et d_y. On écrira plus simplement $T_a = a_1 + a_2 \times \sqrt[2]{{d_x}^2 + {d_y}^2}$. On note que cette formule ne peut rendre compte d'un mouvement de rotation.

Les coefficients a_1 et a_2 définis ainsi dans le papier :

- a_1 est la composante statique correspondant à la déstabilisation causée par la résolution du capteur ou la précision du flux optique.
- a_2 est utilisé pour introduire la partie dynamique de la composante, et nous utilisons un seuil élevé lorsque le capteur se déplace rapidement.

Autant dire que la détermination de ces coefficients se fera par la méthode scientifiquement éprouvée du … doigt mouillé !

Pour ce qui est de a_2, nous devrons utiliser une valeur relativement faible, les déplacements étant relativement lents.

La vitesse de déplacement, dans le cas d'une translation suivant l'axe des x et l'axe des y, peut être calculer.

- Sur l'axe x : $v_x = \frac{d_x}{d_t}$ ou d_t est l'intervalle de temps séparant les 2 images,
- Sur l'axe y: $v_y = \frac{d_y}{d_t}$

Malgré plusieurs essais, il n'a pas été possible de déterminer les coefficients a_1 et a_2 expérimentalement et obtenir une extraction d'avant-plan correct.

4.5.2 Déplacement de l'avant-plan avec effet de zoom.

Selon le papier, la différence de magnitude ($d_v = \left\| f_{t,t-k} - \tilde{f}_{t,t-k} \right\|_2$) calculé entre l'avant-plan et l'arrière-plan n'est plus suffisante pour pouvoir les discriminer de façon claire.

Le papier propose alors de comparer la direction des vecteurs d'avant-plan et la direction des vecteurs d'arrière-plan. Le masque de l'avant plan sera calculé ainsi : C'est l'ensemble des points vérifiant l'inéquation $d_c < T_c$. Ce qui peut s'écrire : $M_t = \{Points \mid d_c < T_c\}, d_c = cos(f_{t,t-k}, \tilde{f}_{t,t-k})$.

Pour ceux qui ont gardé le doigt mouillé, ils pourront l'utiliser pour déterminer T_c !

Note sur le cosinus de 2 vecteurs (Cosine similarity - Wikipedia)

$$cos(\vec{u}, \vec{v}) = \frac{\sum_{i=0}^{n} u_i v_i}{\sqrt[2]{\sum_{i=0}^{n} {u_i}^2}\ \sqrt[2]{\sum_{i=0}^{n} {v_i}^2}}$$

Dans notre cas le vecteur d'arrière-plan à pour coordonnées (dx,dy,0), la formule s'écrit alors :

$$\cos\left(\vec{f}, \vec{\tilde{f}}\right) = \frac{u.d_x + v.d_y}{\sqrt[2]{u^2 + {d_x}^2}\ \sqrt[2]{v^2 + {d_y}^2}} \text{ avec } \vec{f} = \begin{pmatrix} u \\ v \\ 0 \end{pmatrix} et\ \vec{\tilde{f}} = \begin{pmatrix} d_x \\ d_y \\ 0 \end{pmatrix}$$

4.5.3 Comment déterminer la nature du mouvement (effet de zoom ou non).

Lorsque nous avons un mouvement de zoom, la direction des vecteurs de l'arrière-plan converge vers le point de fuite de la perspective. L'idée est donc de calculer les coordonnées de ce point de fuite. Si celui-ci appartient à l'image, alors on considère que le mouvement est majoritairement un mouvement de zoom. Si ce n'est pas le cas, la composée principale du mouvement est de nature différente, et la 1ère méthode sera privilégié.

Comme pour la détermination de la matrice H, la détermination des coordonnées du point de fuite se fera sur plusieurs échantillons de 4 points. La méthode des moindres carrés permettra d'avoir la meilleure approximation possible des coordonnées de ce point.

$p_0 = (A^T * A)^{-1} * A^T * B$

A= [V -U] avec U= $[u_1\ u_2 \dots u_n]^T$ et V= $[v_1\ v_2 \dots v_n]^T$

B= [V ◦ X - U ◦ Y] avec X= $[x_1\ x_2 \dots x_n]^T$ et Y= $[y_1\ y_2 \dots y_n]^T$ ◦ est l'opération de multiplication matricielle des éléments.

$$A = \begin{pmatrix} v_1 & -u_1 \\ v_2 & -u_2 \\ \vdots & \vdots \\ v_n & -u_n \end{pmatrix} et\ A^T = \begin{pmatrix} v_1 & v_2 & \cdots & v_n \\ -u_1 & -u_2 & \cdots & -u_n \end{pmatrix} \text{ d'où } A^T * A = \begin{pmatrix} \sum_{i=1}^{n} {v_i}^2 & \sum_{i=1}^{n} -u_i\, v_i \\ \sum_{i=1}^{n} -u_i\, v_i & \sum_{i=1}^{n} {u_i}^2 \end{pmatrix}$$

Il faudra ensuite recherche la matrice inverse. Ce calcul ne sera pas détaillé car la méthode "linalg.inv" de la librairie numpy le fait bien ! Il faudra multiplier le résultat par la transposer de A qui a 2 lignes et n colonnes. À ce stade le résultat sera donc une matrice de 2 lignes et n colonnes.

$$V \circ \mathrm{X} = \begin{pmatrix} v_1 x_1 \\ v_2 x_2 \\ \vdots \\ v_n x_n \end{pmatrix} \text{ et } U \circ \mathrm{Y} = \begin{pmatrix} u_1 y_1 \\ u_2 y_2 \\ \vdots \\ u_n y_n \end{pmatrix}$$

$$\text{d'où } B = \begin{pmatrix} v_1 x_1 - u_1 y_1 \\ v_2 x_2 - u_2 y_2 \\ \vdots \\ v_n x_n - u_n y_n \end{pmatrix}$$

La matrice de 2 lignes et n colonnes (4 dans notre cas) précédemment trouvée multiplier par le vecteur B donnera bien un résultat qui sera sous la forme d'un vecteur de 2 lignes et 1 colonne.

$$p_0 = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = (A^T * A)^{-1} * A^T * B$$

Ce point appartient-il à l'image ? Une simple comparaison des coordonnées du point de fuite avec les dimensions de l'image répondra à la question.

Dernier "petit détail" à régler…Une deuxième condition posée, concerne la norme du gradient de f. Cette condition permet de filtrer le bruit et de s'assurer qu'il y a bien une différence significative entre les images.

Une variable booléenne idx est ainsi défini (x_{p0} et y_{p0} sont les coordonnées du point p_0, l et h sont la largeur et la hauteur de l'image):

$$idx = (x_{p0} \in [0, l]\ et\ y_{p0} \in [0, h])\ et\ (\left\| \nabla \left\| f_{t,t-k} \right\|_2 \right\|_2 > T_g)$$

En ce qui concerne T_g, il pourra être déterminé par la même méthode ayant servi pour déterminer T_c.

Pour déterminer $\left\| \nabla \left\| f_{t,t-k} \right\|_2 \right\|_2$ On va commencer par calculer la norme euclidienne de $f_{t,t-k}$ puis calculer son gradient avec $f_{t-k,t-k-1}$ (image précédente) :

$$f_{t,t-k} = \begin{pmatrix} u_{1,t} & v_{1,t} \\ u_{2,t} & v_{2,t} \\ & \vdots \\ u_{n,t} & v_{n,t} \end{pmatrix}$$

$$\left\| f_{t,t-k} \right\|_2 = \left(\sqrt{\sum_{i=1}^{n} {u_{i,t}}^2} \middle| \sqrt{\sum_{i=1}^{n} {v_{i,t}}^2} \right)$$

$$\nabla \left\| f_{t,t-k} \right\|_2 = \left(\sqrt{\sum_{i=1}^{n} {u_{i,t}}^2} - \sqrt{\sum_{i=1}^{n} {u_{i,t-k}}^2} \middle| \sqrt{\sum_{i=1}^{n} {v_{i,t}}^2} - \sqrt{\sum_{i=1}^{n} {v_{i,t-k}}^2} \right)$$

$$\left\|\nabla\left\|f_{t,t-k}\right\|_2\right\|_2 = \sqrt{\left(\sqrt{\sum_{i=1}^{n} {u_{i,t}}^2} - \sqrt{\sum_{i=1}^{n} {u_{i,t-k}}^2}\right)^2 + \left(\sqrt{\sum_{i=1}^{n} {v_{i,t}}^2} - \sqrt{\sum_{i=1}^{n} {v_{i,t-k}}^2}\right)^2}$$

5 Algorithme proposé.

L'algorithme proposé par le papier est le suivant :

1. Récupérer les images à l'instant t-1 et t (I_{t-1} et I_t)
2. Estimer le flux optique $f_{t,t-1}$ pour I_t et I_{t-1}
3. Grace à l'équation $H_{t\to t-1} * P_t = P_t + F_{t,t-1}$ obtenir $H_{t\to t-1}$
4. Obtenir le modèle d'arrière-plan grâce à l'équation : $\widetilde{f}_{t\to t-k} = H_{t\to t-k} \times p_t - p_t$
5. Recherche le point de fuite et voir si ses coordonnées vérifient la condition

$$\text{idx} = \left(x_{p0} \in [0, l] \text{ et } y_{p0} \in [0, h]\right) \text{ et } \left(\left\|\nabla\left\|f_{t,t-k}\right\|_2\right\|_2 > T_g\right)$$

6. Si idx= 0 alors
   - Utiliser $M_t = \{Points \mid d_v > T_a\}$, $d_v = \left\|f_{t,t-k} - \tilde{f}_{t,t-k}\right\|_2$
7. Sinon
   - Utiliser $M_t = \{Points \mid d_c < T_c\}$, $d_c = cos(f_{t,t-k}, \tilde{f}_{t,t-k})$.
8. Sortir le masque d'avant plan

6 Note à propos des mouvements de rotation.

Il faudra refaire l'étude sur un mouvement de rotation de la caméra qui n'est pas pris en compte dans cette étude. Les mouvements de rotation sont fréquents sur les grues.

7 Conclusion de l'étude réalisée sur le papier.

Le papier étudié propose d'extraire en temps réel l'arrière-plan avec une caméra en mouvement avec l'algorithme décrit à la section 5. Plusieurs remarques concernant cet algorithme:

7.1.1 Utilisation de FlowNet2.0 pour calculer les vecteurs du flux optique.

Sur une paire d'images au format 1280 × 960, le temps de calcul sur un core I7 est de 26 secondes. La contrainte temps réel n'est pas franchement respecté ! Cette méthode de calcul des vecteurs du flux optique ne peut être retenu ce qui n'est pas un problème puisque la caméra STV0991 permet d'extraire ces mêmes vecteurs mais en utilisant une méthode différente.

Cependant la répartition des vecteurs est très différente puisque l'algorithme de calcul de la STV0991 ne fournit les vecteurs que sur des points remarquables (en l'occurrence des coins sélectionnées par la chaîne d'algorithme détection de coin → suppression des non-maximum → filtre répartition spatial → caractérisation des points).

7.1.2 Le calcul de la matrice d'homographie.

La méthode de détermination de la matrice H proposé par le papier a été tester (sans pour autant utiliser estimateur RAMSAC) et est quasiment identique à la méthode "findHomography" proposée par open CV et qui implémente l'estimateur RAMSAC.

7.1.3 Choix de l'algorithme (calcul de idx).

Tout d'abord, il faut déterminer les coordonnées du point de fuite de l'avant-plan. Le principe est d'utiliser aussi l'estimateur RAMSAC compatible temps réel. 4 points sont choisis de façon aléatoire et correctement répartie sur l'image. Le calcul est un calcul matriciel, qui devrait s'exécuter rapidement.

Ensuite, il faut vérifier que les vecteurs présentent un gradient significatif. Ce calcul risque d'être plus lourd car il devra être réalisée pour tous les vecteurs.

7.1.4 Déterminer les points d'avant-plan.

En fonction de la variable booléenne idx on utilisera soit la méthode évaluant la différence entre les normes des vecteurs d'avant-plan et d'arrière-plan soit la méthode évaluant l'angle entre ces deux groupes de vecteurs. La méthode des cosinus risque d'être couteuse en termes de ressource machine.

7.1.5 En résumé.

- Le principal problème d'une méthode proposant plusieurs algorithmes en fonction du type de mouvement est qu'elle est difficile à utiliser si plusieurs objets se déplacent sans suivre le même schéma.
- Si le temps d'exécution de ces algorithmes est très différent cela peut ne pas respecter les contraintes liées au temps réel.
- Pour autant, la détection des mouvements avec effet de zoom est problématique, et il semble difficile à traiter. Trouver une méthode unique risque d'être particulièrement difficile.
- Les essais d'extraction de l'avant-plan ont montré que le choix de FAST/BRIEF est pertinent.
- Toutes les équations du papier n'ont pu être testé. La tentative d'implémentation de la formule $T_a = a_1 + a_2 \times \sqrt[2]{(H_{t\to t-k}(1,3))^2 + (H_{t\to t-k}(2,3))^2}$ a été un échec, les coefficients a_1 et a_2 n'ayant pu être déterminé.

8 Annexes.

8.1 Code des différents programmes développé pour l'étude:

8.1.1 mouvement_objet_camera.ipynb :

Ce jupyter notebook reprend les études réalisées en python pour comprendre, analyser et donner des solutions à la problématique de la détection d'avant-plan avec une caméra en mouvement. C'est un document évolutif et possède une zone "brouillon" dans laquelle un certain nombre de pistes non abouties sont en cours de recherche.

8.1.2 extract_avant-plan_sift.py

Programme d'extraction d'avant-plan utilisant SIFT pour la détection de points. Ces points dont les coordonnées sont calculées permettent de déduire le mouvement de la caméra. La matrice H est calculée et permet de corriger l'image du mouvement de la caméra. L'arrière-plan est ainsi déduit et soustrait avec l'algorithme "createBackgroundSubtractorMOG2".

8.1.3 extract_avant-plan_orb.py

Programme identique mais utilisant l'algorithme ORB pour se rapprocher au plus près des résultats que l'on pourrait obtenir avec la caméra STV0991.

8.2 Papiers scientifiques:

- Moving Object Detection.pdf
- Background Modelling from a Moving Camera.pdf

8.3 Étude de la documentation de la caméra STV 0991.

Optical flow STV0991 et simulation en python (d'après la documentation officielle de STMicroelectronics).