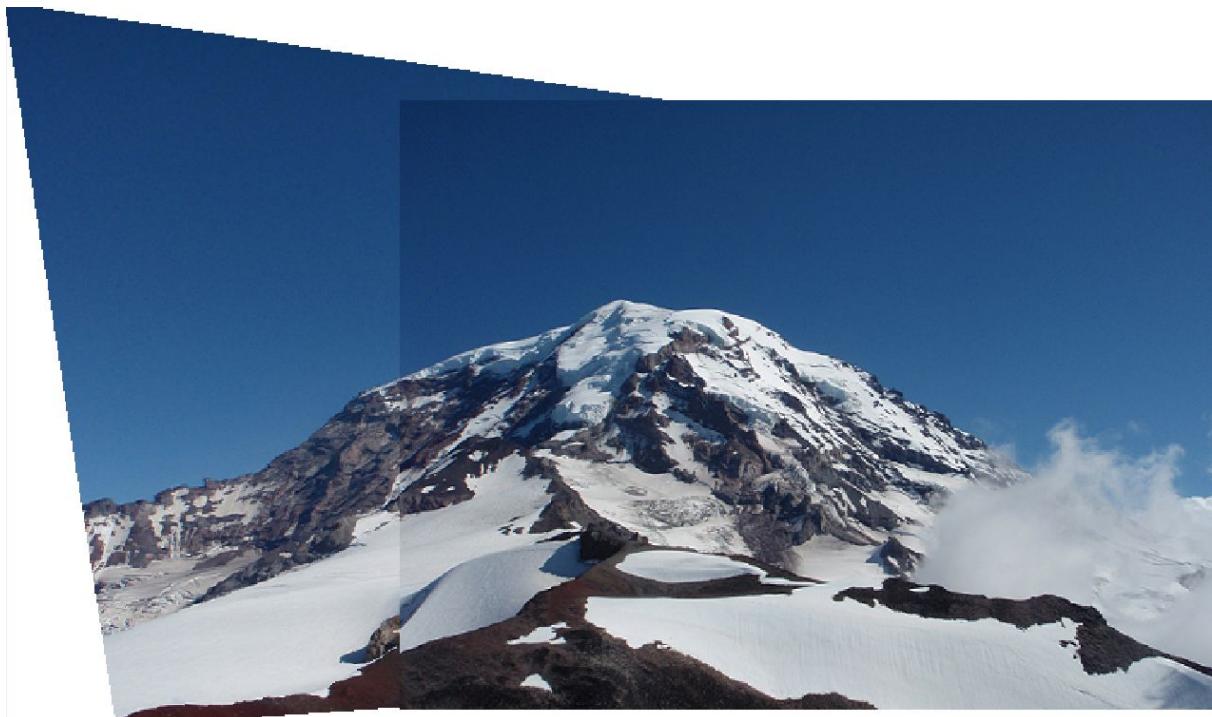


Gestion de Projet & Vision par Ordinateur : Reconstruction Panoramique



Durand Aurélien - M2-SIS

Ladjouzi Rachid - M2-SIS

Chadee Nopharut - M2-SIS

I. Introduction:

Lors de ce Projet nous nous sommes intéressés à la reconstruction panoramique et les algorithmes qui en découlent. Cette méthode est basée sur le principe de détection de points nommés points d'intérêt et de leurs mises en correspondance entre deux images. Nous nous sommes donc intéressés à différentes détections de points d'intérêt à savoir : le détecteur de Harris, le détecteur Sift et le détecteur Fast. A partir des détecteurs on a ensuite regardé le procédé algorithmique afin d'obtenir une reconstruction Homographique.

Au niveau du code et afin de lier les modules de Gestion de Projet et de Vision par Ordinateur nous avons réalisé les algorithmes suivants :

- Le détecteur Fast à partir de l'article d'Edward Rosten et al. Faster and better: a machine learning approach to corner detection[6]. Réalisé sous python 3.7.
- Le détecteur de Harris, réalisé à partir du TP n°3 du cours Vision par Ordinateur de Mr.Wang Chaoui. Pour le détecteur de Harris nous avons réalisé un notebook sous google colab. (Les images utilisées sont les images 1 et 2 du répertoire Harris à intégrer dans le répertoire de google colab)
- Le détecteur Sift, utilisation de la bibliothèque OpenCV.
- Le calcul d'Homographie et la reconstruction panoramique sous Python 3.7.

III. Détection des points d'intérêt dans une image :

Le but principal de la reconstruction de panorama est de détecter des similarités entre 2 ou plusieurs images d'une même scène. Ensuite on vient appliquer des opérations de translation/rotation/scale aux images pour construire l'image correspondant à la réunion de toutes les scènes. Il faut donc repérer des structures qui soient invariantes aux opérations de transformation dans une image. Ces structures sont alors définies comme des points d'intérêt. Un exemple de point d'intérêt sur une image:



Figure I.1: Exemple de points d'intérêt avec le détecteur FAST

Il existe plusieurs méthodes pour les identifier qui dépendent de la définition que l'on en fait. Par définition on va choisir un point d'intérêt comme un point représentant une forte variation d'intensité en deux directions différentes. Donc un point où le gradient est fort. Par exemple au niveau des yeux d'un des chats on a:

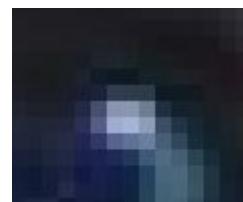


Figure I.2: Exemple de variation d'intensité dans une image.

Il faut donc estimer le gradient local aux niveaux des points d'intérêt dans une image. La grande majorité des détecteurs vont alors regarder le contenu d'un patch centré dans les coins d'un objet car c'est là que l'intensité du gradient va être la plus importante:

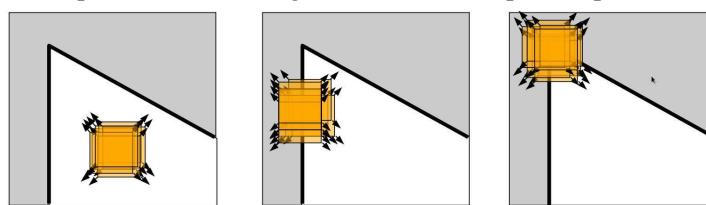


Figure I.3: Figure montrant les endroits où l'intensité du gradient est la plus forte

A partir de là il existe différents algorithmes qui vont permettre la détection de ces points. Les algorithmes permettant de gérer les rotations/translations ainsi que les changements d'échelle sont faits à partir d'un descripteur, ce dernier consiste à trouver une série de points dans l'image, définir une région autour de chaque point, extraire et normaliser la région, calculer le

descripteur (par exemple histogramme du patch) et enfin apparier les descripteurs entre deux images:

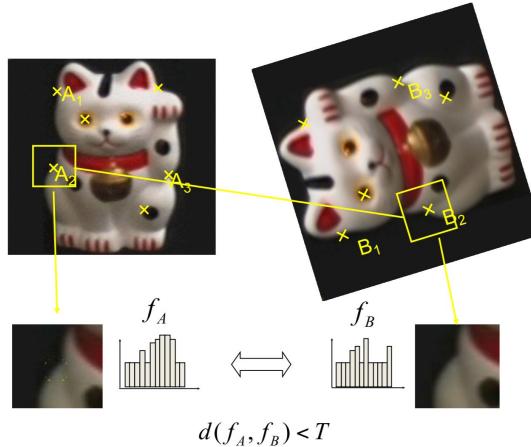


Figure I.4: Descripteur d'un patch dans une image à partir de son histogramme

Dans notre cas les descripteurs doivent nous rendre alors les points d'intérêt / coins de l'image. Pour une approche plus robuste l'algorithme SIFT va par exemple travailler sur plusieurs niveaux d'échelle afin de calculer son descripteur. Dans ce travail nous présentons trois détecteurs de points :

- Le détecteur de **Harris** dit “**Harris Corner Detector**” très répandu et utilisé dans un certain nombre d'applications.
- Le détecteur **SIFT** qui est une extension du détecteur de Harris corrigéant ces “défauts” et proposant une approche multi-échelle permettant de répondre aux problèmes de scale.
- Et enfin le détecteur **Fast** proposant une approche rapide de détection qui a la particularité d'être sans descripteur.

a. Algorithme de Harris

Les premières pistes concernant la détection de points d'intérêt remontent aux années 80 et au détecteur de Moravec dans un contexte de recherche de correspondance entre 2 (ou plusieurs) images. Cependant, ce détecteur trouve rapidement ses limites, notamment dans les réponses (approximatives) du détecteur. Ainsi, le détecteur de Harris, créé par Chris Harris et Mike Stephens, a été pensé pour corriger les limitations du détecteur de Moravec.

L'idée de ce détecteur est de repérer les points/pixels autour desquels l'intensité de l'image varie dans plusieurs directions. Pour l'algorithme de Harris on s'est basé sur un cours de [Frédéric Devernay \[1\]](#) ainsi que d'un rapport de projet de [Julien Deshayes et Pierre Bouge](#) de l'ISIMA [2].

Description de l'algorithme

- 1) Calcul (approximatif) des gradients (dérivées d'une image) à l'aide d'un filtre.
Filtre de Sobel :

Gradients selon x et y:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Filtre de Gauss :

$$g(x, y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Gradient selon x: $I_x = \frac{\partial}{\partial x} g(x, y)$

Gradient selon y: $I_y = \frac{\partial}{\partial y} g(x, y)$

2) Calcul d'un tenseur de structure (Matrice M) à l'aide d'un filtre Gaussien (avec un certain écart-type σ et une taille adaptée (5 par exemple)).

$$M = g(x, y) * \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}$$

3) Calcul des valeurs propres de la fonction de réponse (Matrice M) à partir du tenseur de structure (utilisation du déterminant et de la trace de la matrice). Les valeurs propres donnent en fait les directions principales des gradients locaux de l'image. On a alors pour la détection de coins deux valeurs propres car matrice 2x2 qui donne alors les propriétés suivantes en fonction des valeurs propres λ_1 et λ_2 :

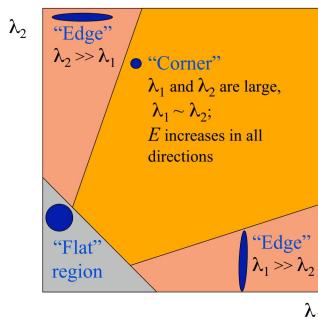


Figure I.5: Etiquetage des détections en fonction des valeurs propres du tenseur de structure. Les différentes possibilités étant un bord, une région homogène ou un coin.

Le calcul des valeurs propres λ_n étant trop lourd si on doit l'effectuer localement il est préférable en pratique de calculer une fonction de réponse R donnée par :

$$R = \det(M) - \alpha \cdot \text{trace}(M)^2$$

Les valeurs propres correspondent alors aux maximums locaux de R. Le paramètre α prenant pour valeur 0.04 où bien 0.06.

4) Seuillage et suppression non-maximale des valeurs de R (on ne garde que les N plus grandes valeurs, N étant défini arbitrairement même si 4 points d'intérêt suffisent pour faire une homographie).

Avantages/Inconvénients

- ✓ Invariance par rotation et translation.
- ✓ Invariance par changement d'intensité lumineuse.
- ✗ Peu robuste au changement d'échelle (l'algorithme SIFT permet de corriger cet inconvénient).

Une implémentation de l'algorithme de Harris donnera alors dans l'ordre les images suivantes pour la détection de coins :

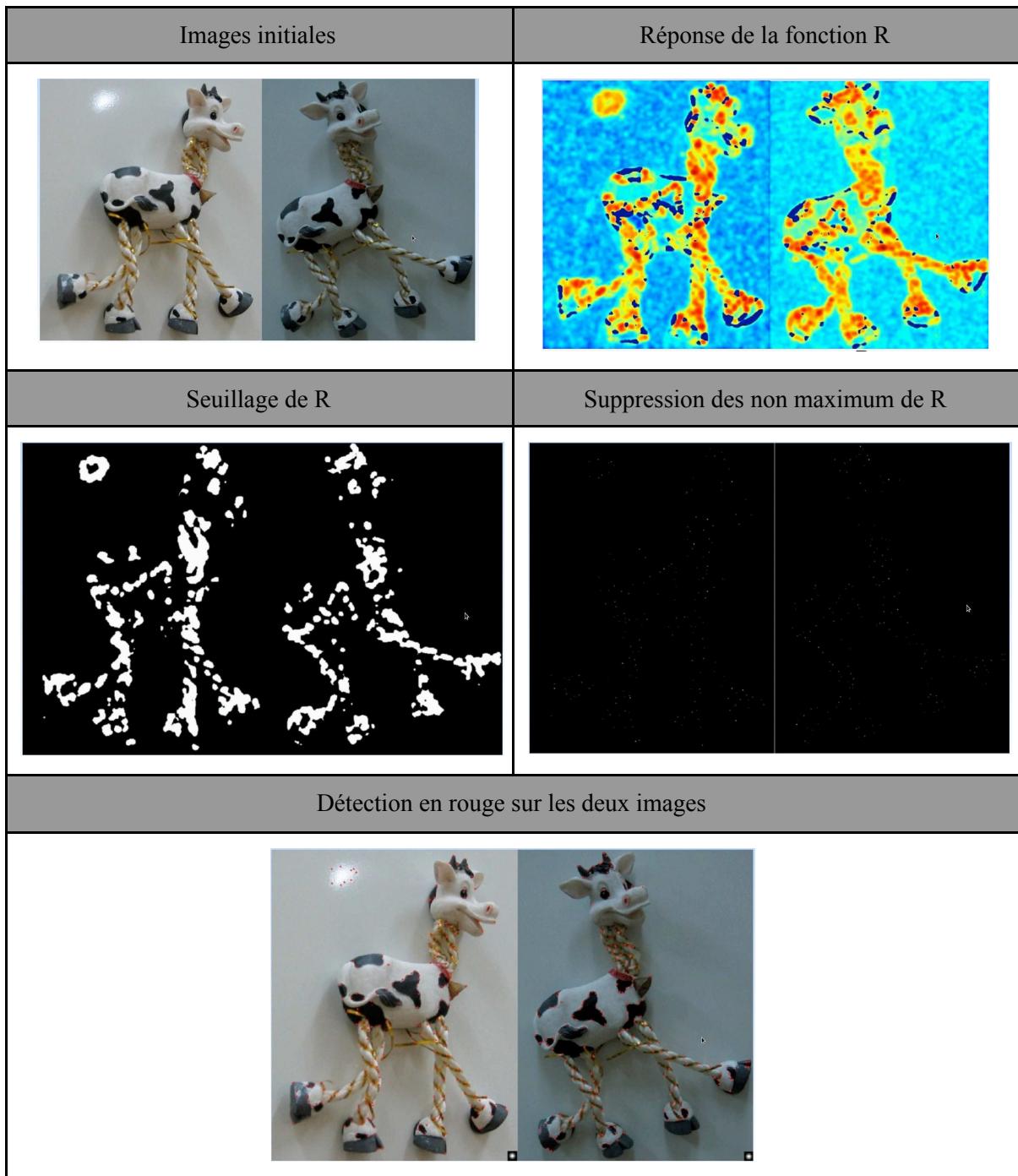


Figure I.6: Procédé détecteur de Harris

On a donc implémenté ce détecteur dans le notebook du dossier Harris à partir des images Images1 et Images2 du même dossier. Le principe est donc comme présenté de venir calculer la fonction R à partir du gradient de l'image ce qui donne:

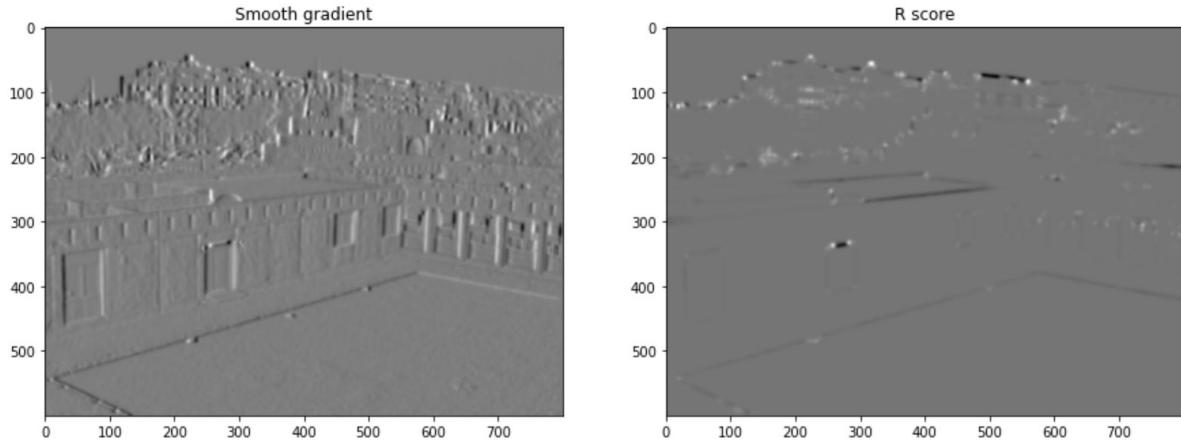


Figure I.7: Gradient et Fonction de réponse R

A partir de la fonction de réponse R on peut alors détecter des points d'intérêt en venant seuiller et en supprimant les non-maximum des valeurs de R. Cela permet alors d'obtenir les points d'intérêt suivants :

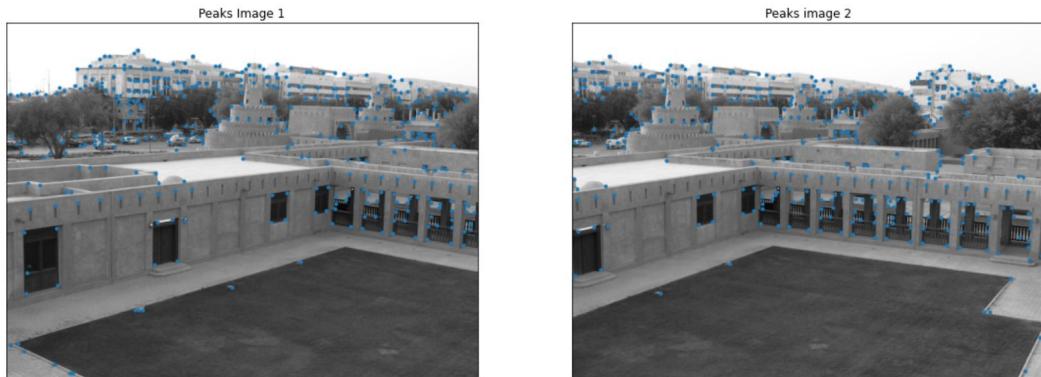


Figure I.8: Détection de points d'intérêt avec le détecteur de Harris

On obtient alors 392 détection dans la première image et 374 dans la seconde qu'il faudra alors appairier dans la partie Appariement.

b. Algorithme Sift

SIFT¹ est un algorithme proposé en 1999 par **David G. Lowe [3][4]** inspiré du détecteur de **Harris**, qui comble en grande partie ses limites. Pour cette partie on s'est basé sur les article de **David G. Lowe** ainsi que d'un site internet de **Philippe Poublang [5]**. Ce détecteur génère des descripteurs invariants à la rotation, au changement d'échelle et partiellement à la luminosité. L'algorithme comprend principalement quatre étapes :

- Détection des extrêmes de l'espace des échelles.
- Localisation des points d'intérêt.
- Affectation d'orientations aux points d'intérêt.
- Calcul des descripteurs.

En effet, certains détails des objets ne sont mis en évidence qu'à une échelle particulière. Pour permettre de décrire un objet sur différentes échelles, la première phase consiste d'abord à construire l'espace des échelles en convoluant l'image avec un noyau gaussien d'écart type σ (équation 1) à plusieurs facteurs d'échelle², et à plusieurs tailles d'image (voir **Figure I.9**) dont le but est d'estomper les artefacts qui ne sont pas pertinents à l'échelle. Quatre octaves³ et cinq noyaux gaussiens de σ différents sont jugés suffisants et recommandés par le créateur de l'algorithme (les images de même taille forment une octave ; entre chaque deux images adjacentes d'une même octave, le σ est multiplié par une constante k). Il s'agit ensuite de calculer l'ensemble des différences de gaussiennes (DoG) comme le montre l'équation 2, où L est l'image convoluée, I l'image d'origine, σ le noyau gaussien, et D la DoG. Il s'agit d'une approximation du Laplacien gaussien plus simple et rapide à calculer.

Équation 0

$$L(x, y, \sigma) = G(x, y, \sigma) \star I(x, y)$$

Équation 1

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Équation 2

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) \star I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma)$$

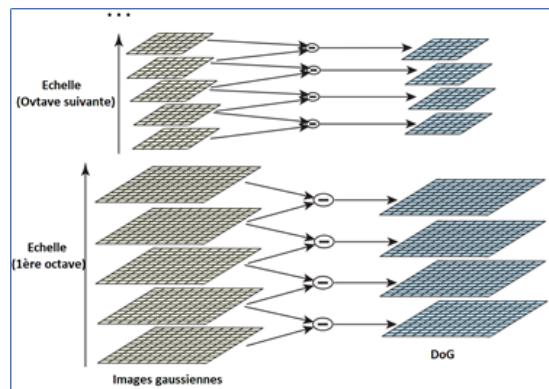


Figure I.9 : Construction de l'espace des échelles et DoG.

¹ Scale Invariant Feature Transform.

² Niveau de flou.

³ Une octave est formée par les images de même taille.

Sur une image on observe ainsi :

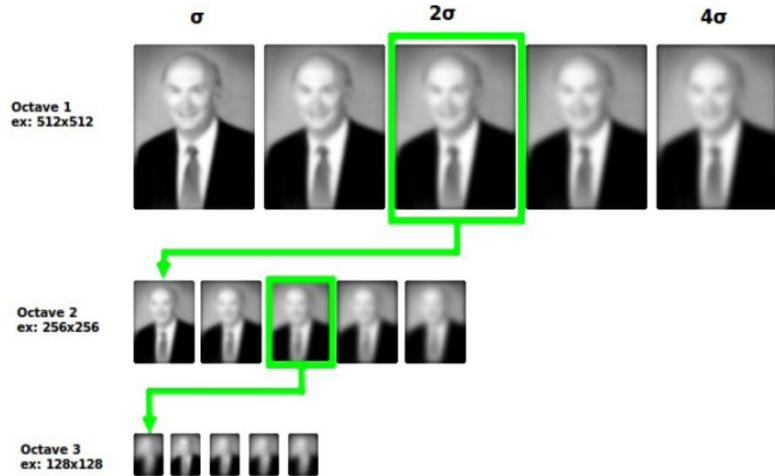


Figure I.10 : Construction de l'espace des échelles et DoG appliqués à une image.

Ainsi, l'image résultante ne contient que les détails dont l'échelle varie entre σ et $K\sigma$, K étant une constante. La présélection des points d'intérêt est faite en détectant les extrema locaux des DoG dans un petit voisinage de 26 pixels (voir **Figure I.11**). Certains points, qui se situent sur les bords ou dont l'intensité est faible, sont éliminés pour ne garder que les points d'intérêt pertinents.

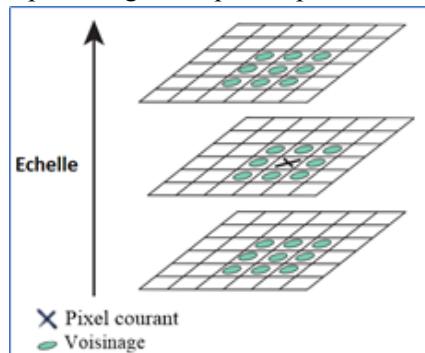


Figure I.11 : Voisinage de détection des extrema de DoG.

L'étape suivante consiste à affecter une orientation à chaque point d'intérêt. Pour ce faire, une région d'intérêt proportionnelle au facteur d'échelle est considérée autour de chaque point clé, l'amplitude et l'orientation du gradient [ab1] sont calculées pour l'ensemble des pixels de la région, puis un histogramme de 36 graduations d'orientations pondérées par la magnitude est construit, l'orientation majoritaire est donc affectée au point clé.

La dernière phase de calcul des descripteurs, consiste à prendre une fenêtre de 16×16 autour de chaque point clé, puis la diviser en seize sous régions de 4×4 . Pour chaque sous région, calculer un histogramme de huit graduations (les points cardinaux) d'orientation pondérées par la magnitude. La concaténation des histogrammes d'une région donne lieu à un vecteur descripteur de 128 éléments (16×16). Notons que les histogrammes sont normalisés pour plus de robustesse à la luminosité. La **Figure I.12** illustre cette étape sur une fenêtre de (8×8) .

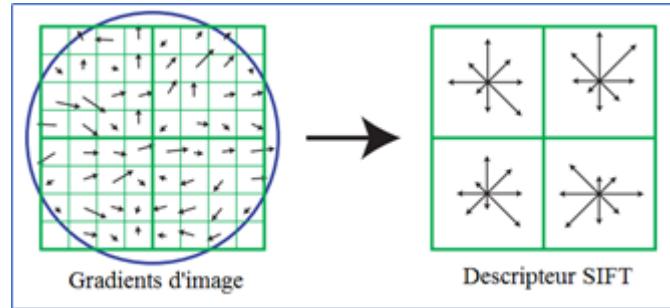


Figure I.12 : Exemple de descripteur SIFT.

L'algorithme nous rend la détection suivante:

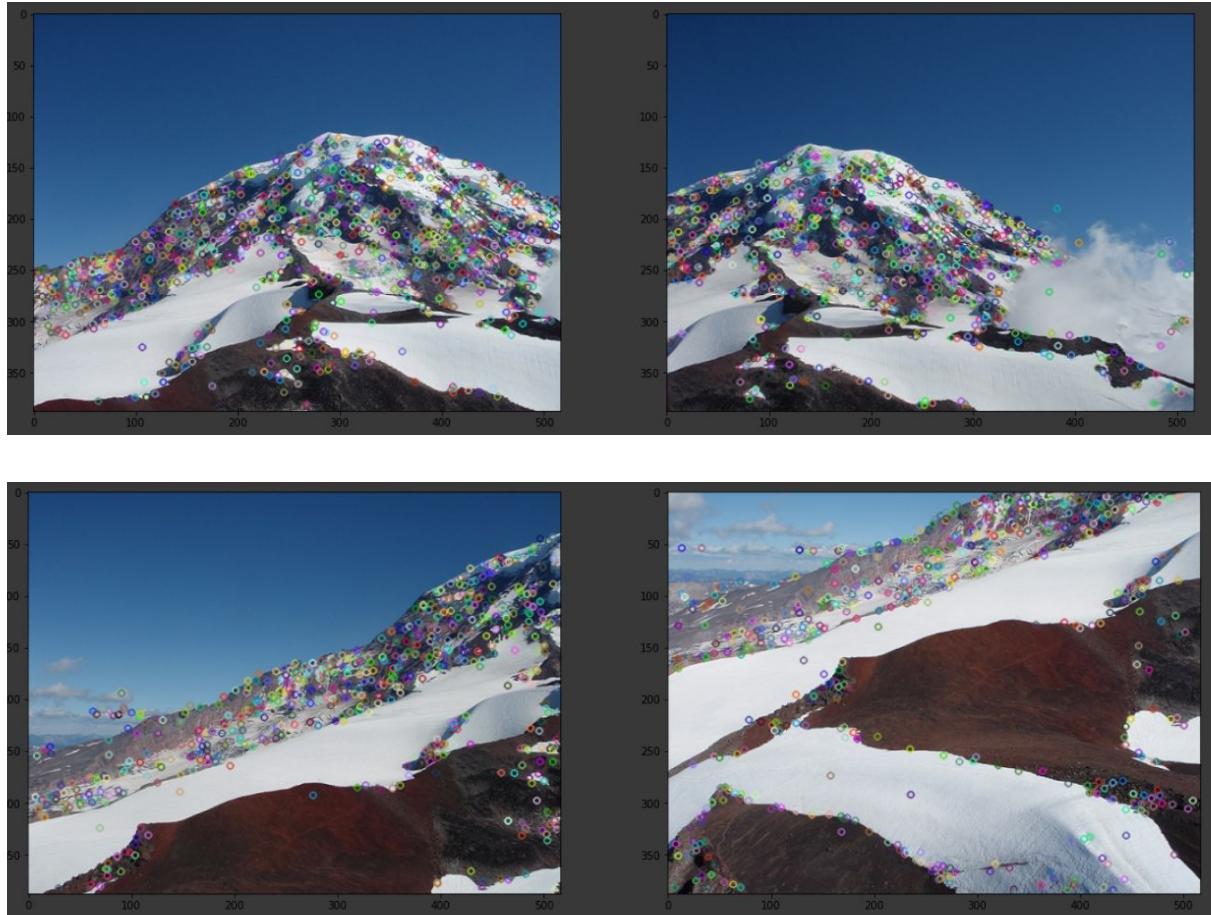


Figure I.13 : Détection de points d'intérêt avec le détecteur SIFT.

c. Algorithme Fast

L'algorithme **Features from Accelerated Segment Test** dit **FAST** comme son nom l'indique est un détecteur de coins qui va détecter des points dans une image de façon rapide. Pour ce faire il ne dispose pas de descripteur et dispose en fait d'une autre définition d'un coin. Ce détecteur est invariant aux changement par rotation/translation ainsi qu'aux changement d'intensité mais n'est pas robuste aux changement d'échelle. De plus au vu d'un objectif de rapidité, cette algorithme est alors peu robust aux bruit. La mise en place de cette algorithme a été publié par [Ed Rosten \[6\]](#) et nous nous sommes basés sur cet article pour décrire et implémenter l'algorithme de fast.

Pour le détecteur fast on commence en fait par définir un patch de 16 valeurs autour d'un pixel comme le montre la figure suivante:

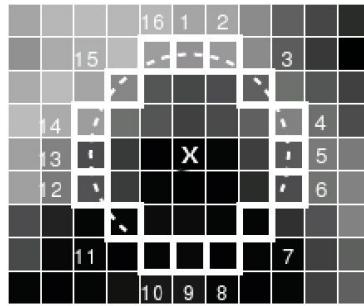


Figure I.14 : le Patch défini par l'algorithme FAST.

On vient alors parcourir le patch et associer à chaque pixel une valeur d'intensité par rapport au pixel central. On fixe un seuil t et on regarde si les pixels sont proches de l'intensité du pixel central à plus ou moins t .

$$S_{p \rightarrow x} = \begin{cases} d, & I_{p \rightarrow x} \leq I_p - t \\ s, & I_p - t < I_{p \rightarrow x} < I_p + t \\ b, & I_p + t \leq I_{p \rightarrow x} \end{cases}$$

On a alors les étiquettes: d pour un pixel plus sombre, s pour un pixel dont l'intensité est contenue dans le seuil, et b pour un pixel plus claire.

L'algorithme est alors le suivant:

- On annote chaque valeur du patch de 1 à 16
- On vient parcourir notre image, en évitant les effets de bord
- Pour chaque pixel si le point 1 et le point 9 sont compris dans un le seuil t alors le point considéré n'est pas compté en coins
- Même conditions mais pour les points 13 et 5.
- On regarde ensuite si parmi les points 1-5-9-13 si au moins 3 d'entre eux sont supérieurs ou inférieurs au seuil imposé. Si ce n'est pas le cas alors le point n'est pas un coin.
- Enfin on vérifie si dans le patch, contenant donc 16 points, au moins n points ont la même étiquette (soit d soit b). D'après l'article d'Edward Rosten, Reid Porter, et Tom Drummond le meilleur choix de n pour la répétabilité étant $n=9$ après étude.
- Si n points successifs valident le critère d'avoir la même étiquette (d ou b) alors le point central est considéré comme un coin.

L'algorithme nous rend alors la détection suivante:

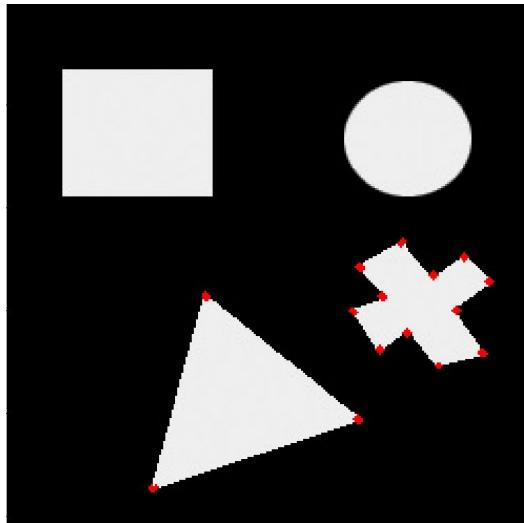


Figure I.15 : Exemple de points d'intérêts détectés par FAST et mis en place en Python.

On remarque que pour le triangle et la croix on détecte l'ensemble des coins. Qu'on ne détecte pas de coins pour le rond mais aussi pas de coins pour le carré. La non détection des coins du carré vient du fait des considérations faites sur le traitement des pixels 1-5-9-13 du patch. Ce n'est donc pas une erreur mais une conséquence du fait que l'on souhaite que l'algorithme tourne le plus vite possible.

Au niveau des détections on remarque qu'il y'a en fait plusieurs détections autour d'un coin, il faut alors mettre en place une élimination des points considérés comme non maximaux. Toujours sur la base de l'article d'*Edward Rosten, Reid Porter, et Tom Drummond*, on applique la partie de suppression.

Le principe est alors le suivant:

- On va parcourir l'ensemble des coins détectés dans l'image et regarder si il y a des voisins dans un masque 3x3.
- Si dans le patch 3x3 il y'a un voisin qui est étiqueté en tant que coin on va alors calculer une fonction de score, notée V . Ceci afin de pouvoir déterminer quel est le point qui est le plus significatif. Cette fonction V est donné par :

$$V = \max \begin{cases} \sum(\text{pixel values} - p) & \text{if } (\text{value} - p) > t \\ \sum(p - \text{pixel values}) & \text{if } (p - \text{value}) > t \end{cases}$$

- On vient regarder le maximum de la fonction V entre les deux points voisins et on retire celui qui a le score le plus faible. La valeur de V étant calculée sur le patch de 16 pixels.
- On met à jour le masque 3x3.
- On augmente la valeur du seuil t avec la valeur du plus petit score V calculé.
- On recommence l'algorithme tant qu'il y'a des voisins dans le masque 3x3. Le fait d'augmenter le seuil successivement permet de faire une élimination jusqu'à ce que le dernier coin le plus significatif soit détecté.

On obtient alors le résultat suivant :

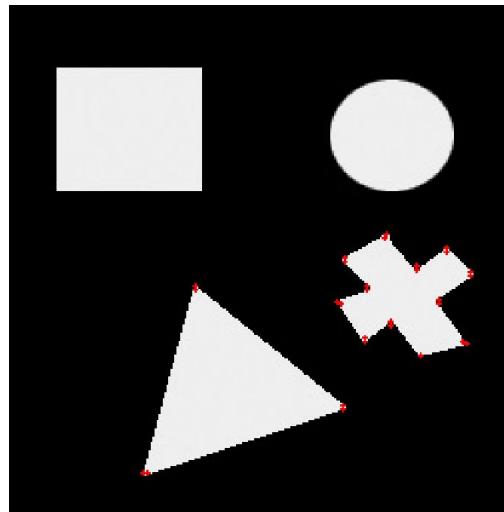


Figure I.16 : Exemple de points d'intérêt détectés par **FAST** avec élimination des non maximaux.

On passe alors de 72 détections à 29 détections. Il reste encore des doubles détections mais on a grandement réduit le nombre de fausses détections et cela va réduire le temps de calcul pour la partie d'appariement.

On a réalisé l'algorithme de détection de coins **FAST** avec l'élimination des non maximaux en Matlab et en Python. Pour une image on obtient alors le résultat suivant :

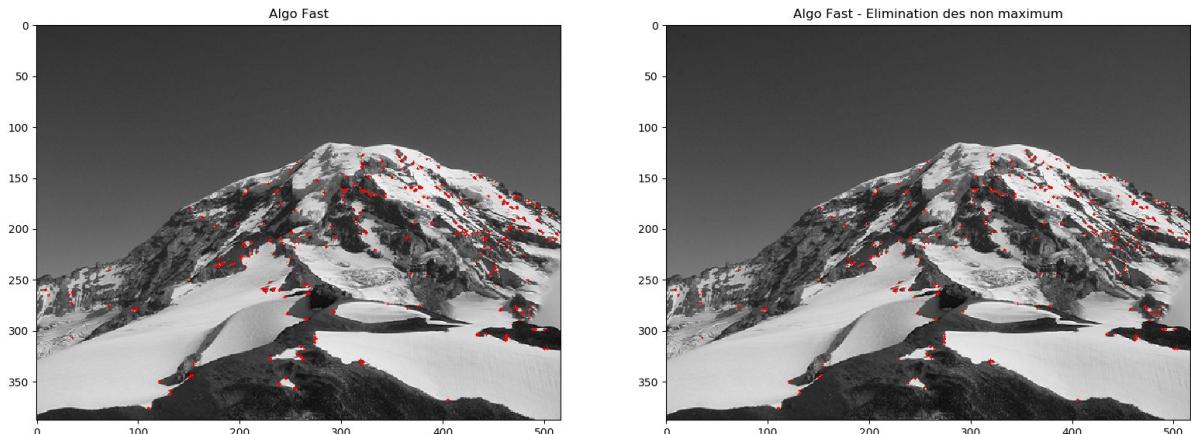


Figure I.17 : Exemple de points d'intérêts détectés par **FAST** avant et après élimination des non maximaux sur une image plus complexe.

On obtient alors 392 détections de coins et 249 après élimination des non maximaux.

Algorithme **FAST** et deep learning:

L'algorithme **FAST** peut être utilisé dans un cadre de deep learning où l'on va entraîner un classifieur sur un modèle et appliquer le classifieur à une image. Il est par exemple possible d'utiliser un système multicouche de Perceptron. Il faut alors entraîner le classifieur pour qu'il reconnaisse les coins détectés par l'algorithme. Le but va être d'entraîner le classifieur en même temps que l'on fait tourner le détecteur sur un ensemble d'images pour pouvoir construire l'arbre de décision du classifieur.

Pour chaque pixel évalué par le détecteur FAST, on peut regarder les 16 voisins du patch comme un vecteur comme le montre la figure suivante :

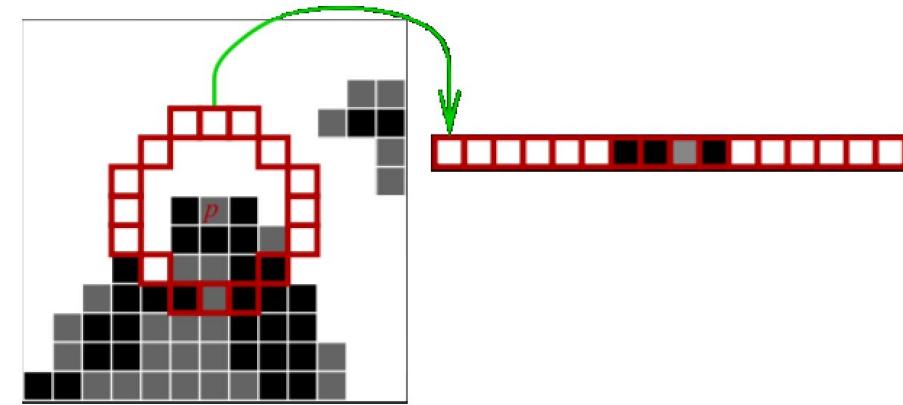


Figure I.18 : Mise en forme des vecteurs du patch cyclique

On regarde alors l'état du pixel p en fonction du vecteur, et il faut enregistrer les caractéristiques pour pouvoir entraîner le classifieur. Ces caractéristiques seront donc un état plus sombre, plus clair ou équivalent du pixel p. Ces données ainsi que le vecteur sont alors classifiés. Un exemple donné dans la présentation de l'algorithme FAST [6] donne alors l'arbre suivant:

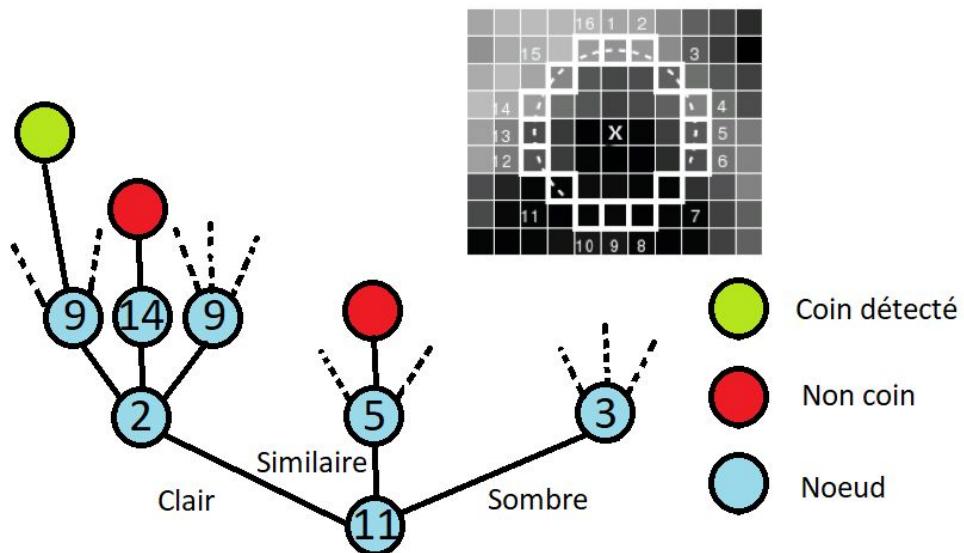


Figure I.19 : Arbre de décision de l'algorithme Fast et classification selon trois catégories. (s pour similaire, c pour un pixel plus clair et d pour un pixel plus sombre)

Chaque feuille de l'arbre reçoit alors une classe $K = 0$ si ce n'est pas un coin et 1 pour les coins. L'arbre est fait de façon récursive et permet d'explorer toutes les possibilités pour un pixel p donné. Donc un vecteur est donné en entrée et par exemple en commençant par le pixel 11 on va explorer récursivement toutes les possibilités qui amènent (ou non) à une détection de point. Pour chaque parcours et pour chaque pixel en appliquant toutes les conditions adéquates liées à l'algorithme FAST, on obtient alors un jeu de données par learning. Pour tester l'apprentissage il faudra ensuite tester sa répétabilité sur des data-set d'images après learning.

La répétabilité étant donnée par:

$$R = \frac{N_{repetition}}{N_{utile}}$$

N_{utile} = Nombre de points qui ont potentiellement un appariement vers la seconde image

$N_{repetition}$ = Nombre de fois où les paires de points apparaissent

Les images du dataset sont par exemple le jeu d'images suivant:



Figure I.20 : Jeu d'images d'une scène pris sous différents angles de caméra

Ici, la même scène est reprise sous différents angles de caméra afin d'évaluer la répétabilité et entraîner le détecteur.

La fonction de coût de l'arbre est:

$$k = \left(1 + \left(\frac{w_r}{r}\right)^2\right) \left(1 + \frac{1}{N} \sum_{i=1}^N \left(\frac{d_i}{w_n}\right)^2\right) \left(1 + \left(\frac{s}{w_s}\right)^2\right)$$

avec:

r = répétabilité

d_i = nombre de points détectés pour l'image i

N = nombre d'images

s = nombre de nœuds de l'arbre de décision

w_r, w_n, w_s = poids de régulation du coût dans l'article, ces valeurs sont prises respectivement à 1, 3.500 et 10.000.

Les poids permettent de réguler une bonne répétabilité, résiste à l'**overfitting**⁴ et la densité des coins et permet d'ajuster les performances des résultats du détecteur de coins. Pour obtenir les 3 valeurs de poids l'article propose en fait de lancer le détecteur avec 3 paramètres différents pour chacun des poids et de récupérer les "meilleurs" poids en fonction des combinaisons différentes pour un total de 27 combinaisons avec w_r (0.5, 1, 2), w_n (1750, 5300, 7000) et w_s (5000, 10000, 20000).

⁴ Overfitting : un modèle trop spécialisé sur les données du Training Set et qui se généralisera mal

L'algorithme FAST-Er est alors une extension de l'algorithme FAST pour obtenir une meilleure répétabilité et est plus consistant aux changements de variation d'un coin qui utilise alors un plus grand nombre de pixels dans le patch:

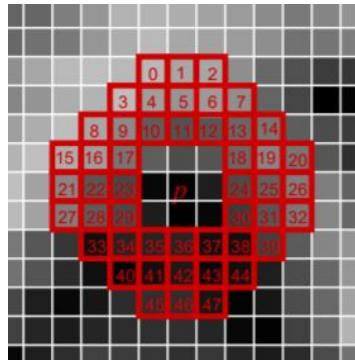


Figure I.21 : Patch étendu du détecteur FAST pour l'algorithme FAST-Er

Avantages et inconvénients du Fast-er par rapport à l'algorithme Fast:

- ✓ Permet d'avoir une meilleure répétabilité
- ✓ S'approche de 100% de répétabilité
- ✓ Chaque pixel est un chemin possible
- ✗ Plus lent en calcul et avec une fonction de coût plus importante que l'algorithme FAST car plus de pixels sont considérés dans cette approche.

La fonction de coût associée est alors

$$cost = (k_R + R^{-2})(k_N + N^2)(k_S + S^{-2})$$

R = Répétabilité.

N = Nombre d'éléments détectés.

S = Taille de l'arbre.

Avec cette algorithme, il faut alors appliquer une stratégie pour modifier l'arbre et obtenir la meilleure répétabilité possible. Dans la présentation, il propose alors de modifier l'arbre de façon aléatoire afin de tester le plus de possibilités totales pour entraîner le classifieur. Cela à pour conséquence d'augmenter le temps de calculs lié au classifieur mais permet d'obtenir à la fin un algorithme avec un meilleur taux de répétabilité.

L'arbre de décision est soumis à la contrainte qu'un bout d'arbre qui a pour étiquette "Similaire" va alors associer, à chacune de ces feuilles, la class K=0.

Les modifications aléatoires sont alors les suivantes:

Un nœud est choisi aléatoirement:

- Le nœud est un bout (coin ou non) alors on remplace avec une probabilité de 50%:
 - Soit le nœud est remplacé par un sous arbre de profondeur 1
 - Soit on copie une classification avec un autre nœud si le nœud n'est pas un point catégorisé en "Similaire"
- Le nœud est un nœud simple (pas en bout d'arbre) alors avec une probabilité égale on applique:
 - Soit on remplace le nœud (l'offset) par un autre entre 0 et 47.
 - Soit on remplace le nœud par un bout d'une classe aléatoire (sombreclair/similaire) (sousmis la contrainte)

- Enlever une branche sélectionnée aléatoirement du nœud et la remplacer par une autre branche du nœud (choisie aussi aléatoirement).

Les modifications sont acceptées par un critère d'acceptation de **Boltzman**:

$$P = e^{\frac{\hat{k}_{I-1} - k_I}{T}}$$

Où P est la probabilité d'accepter un changement à l'itération i .

\hat{k} est le nouveau coût après modification de l'arbre.

T est le paramètre de température définie par :

$$T = \beta e^{-\alpha \frac{I}{I_{max}}}$$

Les paramètres β , α sont choisis dans l'article respectivement à 100 et 30. I étant l'itération et I_{max} le nombre d'itérations totales.

Dans l'article il utilise alors un pentium 4 à 3Ghz pour 100 répétitions avec 100,000 itérations chacune ce qui leur demandait alors 200 heures de calcul pour évaluer l'arbre du Faster, temps de calcul qui pourrait donc être nettement réduit avec des processeurs plus récents.

Une fois l'arbre transcrit en code en une série de conditions if/then/else, le temps d'exécution en comparaison à d'autres méthodes de détection, l'algorithme Fast est plus intéressant ce qui est donné par la tableau ci-dessous:

Detector	Training set		Test set	
	Pixel rate (MPix/s)	%	MPix/s	%
FAST $n = 9$	188	4.90	179	5.15
FAST $n = 12$	158	5.88	154	5.98
Original FAST ($n = 12$)	79.0	11.7	82.2	11.2
FAST-ER	75.4	12.2	67.5	13.7
SUSAN	12.3	74.7	13.6	67.9
Harris	8.05	115	7.90	117
Shi-Tomasi	6.50	142	6.50	142
DoG	4.72	195	5.10	179

Figure I.22 : Temps d'exécution en Mpix/s pour le détecteur fast, le détecteur fast-er et un ensemble d'autres détecteurs couramment utilisés.

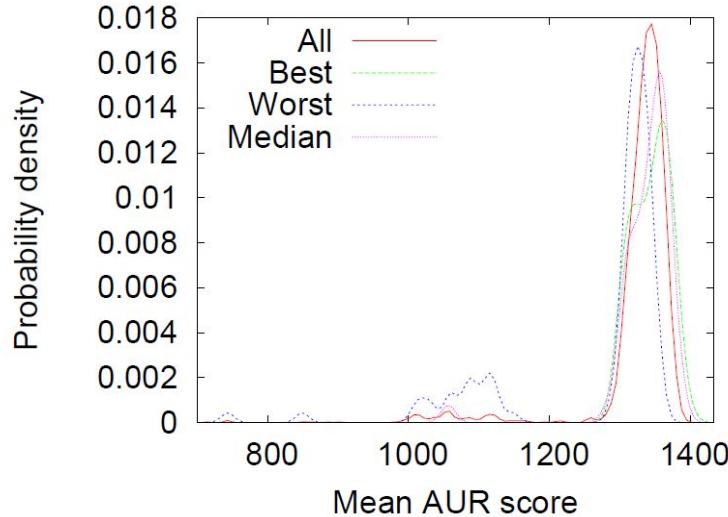
Les tests sont effectués sur le processeur utilisé pour calculer l'arbre de décision. Ils sont effectués sur deux datasets:

Le premier composé de 101 images monochromes haute définition venant d'une vidéo source de résolution 992x668 pixel.

Un deuxième set composé de 4968 images monochromes de résolution 352x288 (quater-PAL) vidéo. On peut voir que globalement le détecteur fast est plus rapide que les autres détecteurs sur les différents jeux de données fournis et qu'avec les considérations de rapidité de l'algorithme celui-ci performe très vite. L'algorithme Fast-er étant lui plus gourmand en calcul mais plus intéressant en

performance est lui aussi bien plus rapide que les autres algorithmes tester. De plus même si désigné pour travailler et rechercher des coins de façon très rapide, les performances en termes de détection et de répétabilité sont quand même très bonnes.

Quelque test de répétabilité de l'algorithme fast donnent alors le graphique suivant:



Les meilleurs paramètres trouvés dans l'étude étant $w_r = 2.0, w_n = 3500, w_s = 5000$ et la courbe avec le moins bon résultats pour $w_r = 0.5, w_n = 3500, w_s = 5000$. Pour obtenir une idée des performances voici enfin deux tableaux pour montrer la comparaison entre l'algo Fast et d'autres algorithmes de détection dont Ed Rosten et ses collègues ont fait les tests:

Tout d'abord les paramètres utilisés pour effectuer les tests sur les différents algorithmes :

DoG		SUSAN	
Scales per octave	3	Distance threshld	4.0
Initial blur σ	0.8		
Octaves	4	Harris-Laplace	
Harris, Shi-Tomasi		Initial blur σ	0.8
Blur σ	2.5	Harris blur	3
		Octaves	4
		Scales per octave	10
General parameters			
ε		5 pixels	

Figure I.24 : Paramètres de travail pour les algorithmes de détection de points autres que le Fast

Detector	<i>A</i>
FAST-ER	1313.6
FAST-9	1304.57
DoG	1275.59
Shi & Tomasi	1219.08
Harris	1195.2
Harris-Laplace	1153.13
FAST-12	1121.53
SUSAN	1116.79
Random	271.73

Figure I.25 : Aire de répétabilité pour un total de 2000 coins par images sur un ensemble de datasets.

On observe qu'en moyenne les algorithmes de fast et faster-er dans leur test obtiennent des résultats plus intéressants et une meilleure répétabilité. Là où l'algorithme Fast va présenter des faiblesses sera alors en présence de bruit car il est à la base conçu pour effectuer la recherche de coins la plus rapide possible.

II. Appariement :

Maintenant que l'on dispose de points d'intérêt, le but est d'apparier entre elle deux images d'un même paysage prises avec un angle de vue différent. Il existe plusieurs méthodes possibles telles que la corrélation, la relaxation proposées par Hummel et Zucker (1983) ou bien par multi-résolution proposée par, Chen et Hung (1993). Dans notre cas nous avons mis en place une solution par corrélation qui consiste à utiliser le calcul de la somme des différences des écarts à la moyenne au carré (ZMSSD : zero-mean sum of squared distances) qui est donnée par :

- On vient réaliser une boucle sur les coins, préalablement calculée avec le détecteur FAST /HARRIS ou SIFT, de l'image 1 puis celle de l'image 2.
- Dans cette double boucle on vient calculer des patchs de tailles N, 3x3 avec N=1, 5x5 avec N=2.
- Une fois les deux patchs calculés aux points (p_i , p_j) on applique la formule ZMSSD entre les deux patchs. Où μ_i sont les moyennes des patchs.
- On compare donc chaque patch de l'image 1 aux patchs de l'image 2 et on associe la valeur de ZMSSD minimum des patchs de l'image 1 fixés par rapport aux patchs de l'image 2. De plus on vient seuiller ce résultat pour ne pas obtenir trop de résultats parasites. L'ensemble des minimums trouvé sera alors l'appariement recherché.

Pour éviter trop d'appariements parasites on applique un double matching à savoir que l'on regarde tous les patchs de l'image 1 par rapport à l'image 2 puis ensuite on fait l'inverse, on regarde tous les patchs de l'image 2 par rapport à l'image 1. On définit alors l'appariement lorsque l'on retrouve un appariement dans chacun des matchings.

Avec le détecteur FAST on obtient alors les appariements suivants pour l'images des montagnes et un décalage de celles-ci:

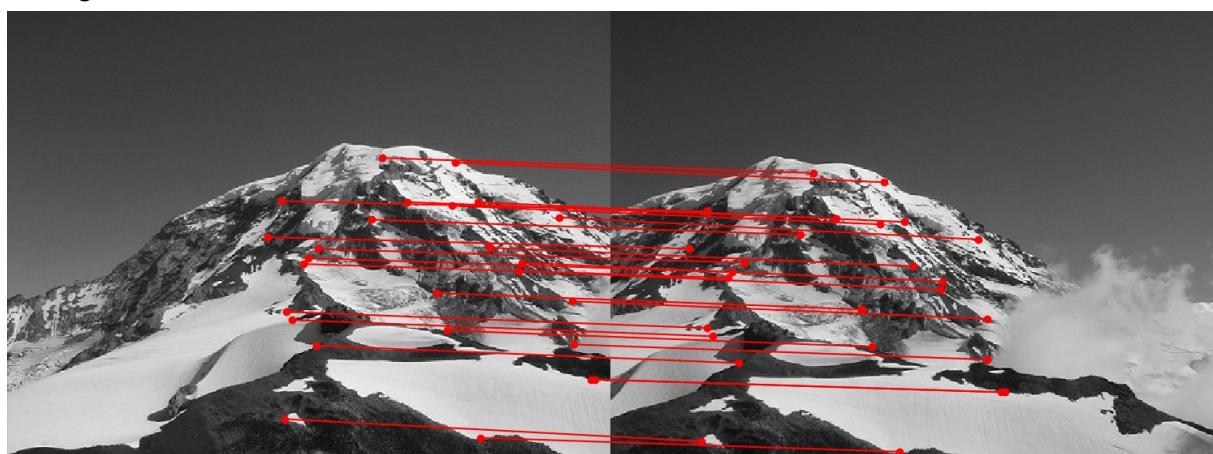


Figure II.1 : Appariement par ZMSSD d'une détection de coin par l'algorithme de fast implanté

On obtient alors 26 appariements entre les deux images. Localement on obtient par exemple les détections suivantes :

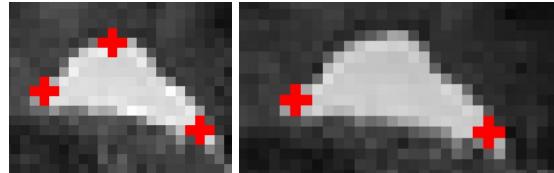


Figure II.2 : Détection du détecteur fast local entre 2 images

Au niveau de l'appariement comme on a choisi un seuil assez bas pour le calcul ZMSSD les points détectés ne sont donc pas tous appariés deux à deux. C'est par exemple ce que l'on peut observer au niveaux de ce pattern ou l'appariement donne alors:

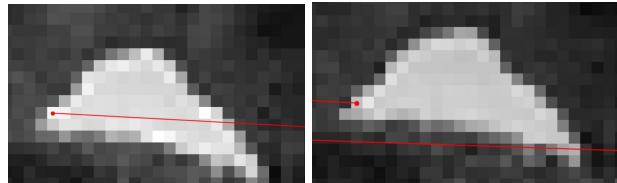


Figure II.3 : Appariement local des détections de coin, un seul appariement sur 2 possible pour un ensemble de 5 détections de points

Seulement une détection est appariée sur les deux possibilités ici mais sur l'image global il y a assez de détections (car 4 bonnes détections sont suffisantes) pour pouvoir réaliser l'homographie et reconstruire le panorama dans le cas d'une rotation/translation pure de caméra.

Avec L'algorithme de Harris du notebook et en adoptant la même stratégie on obtient alors le matching suivant :

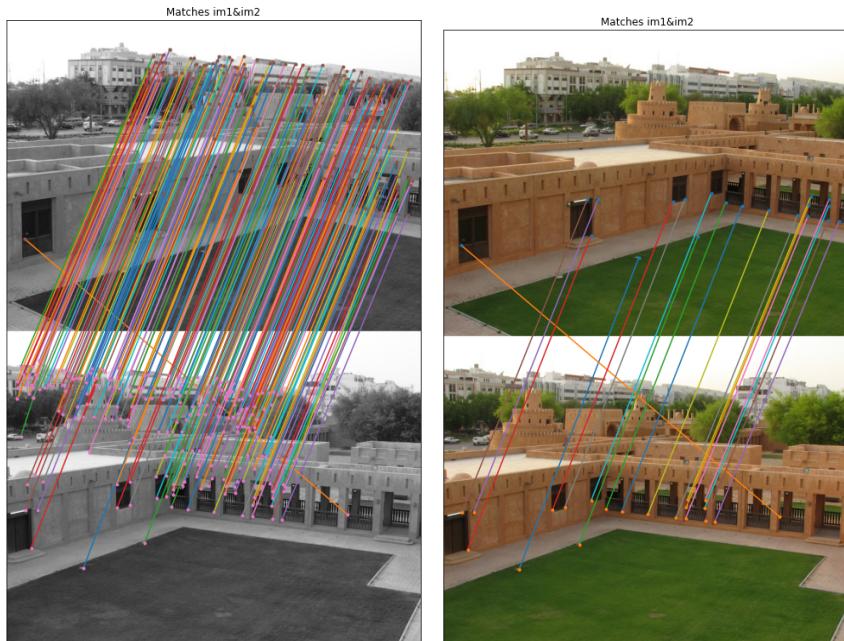


Figure II.4 : Détection et Appariement avec le détecteur de Harris (à droite toutes les détections, à gauche les 20 premières détections)

Ici on peut voir qu'il est possible dans un appariement qu'il y est encore des "outliers" (mauvaise correspondance entre 2 détections) qui nécessite alors d'adopter une stratégie de Ransac lorsque l'on établira l'homographie et ce peut importe le détecteur utilisé.

Avec L'algorithme de SIFT et l'utilisation de la bibliothèque OpenCV on obtient le résultat suivant:

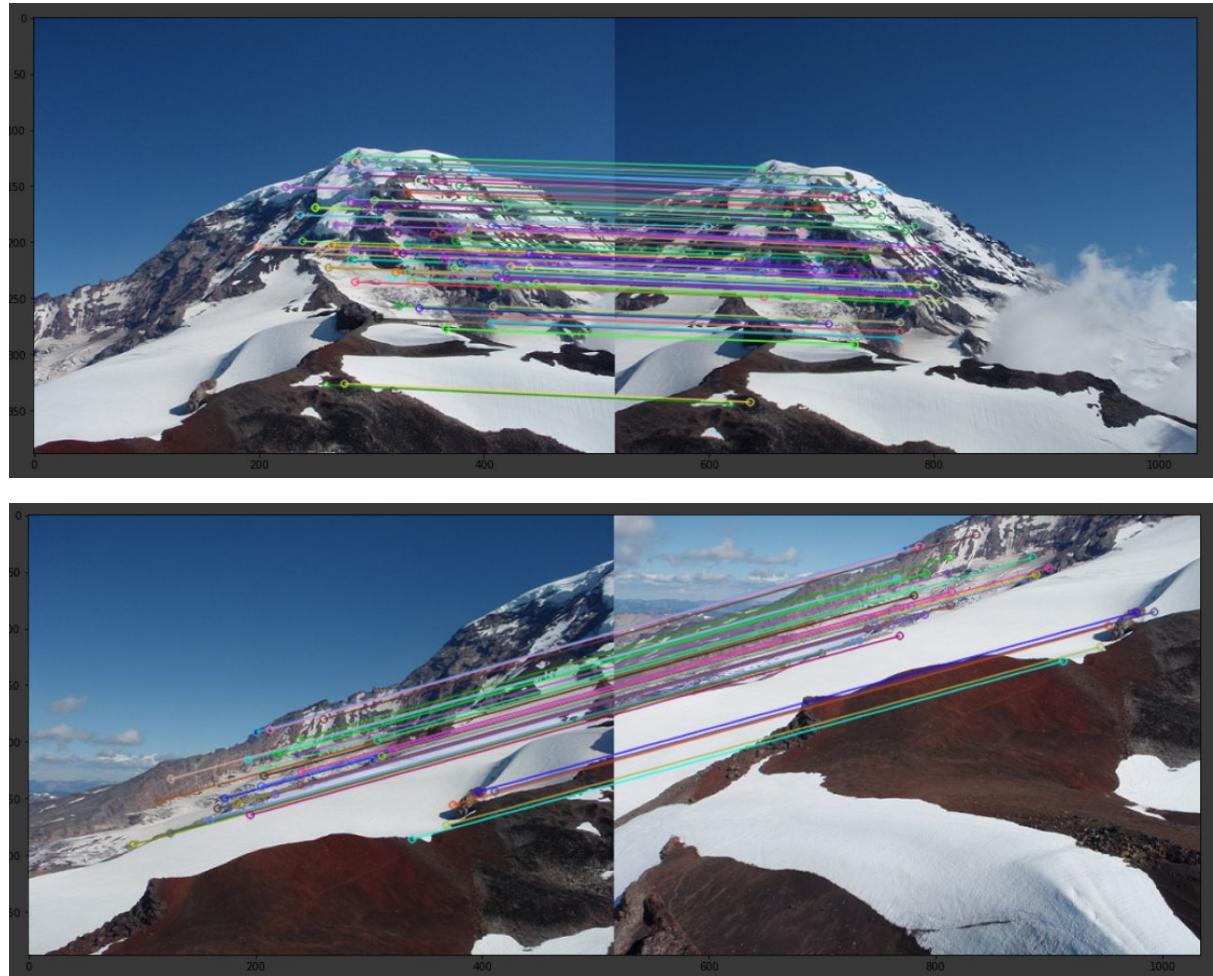


Figure II.5 : Détection du détecteur Sift d'OpenCV

Avec L'algorithme de SURF et l'utilisation de la bibliothèque OpenCV on obtient le résultat suivant:

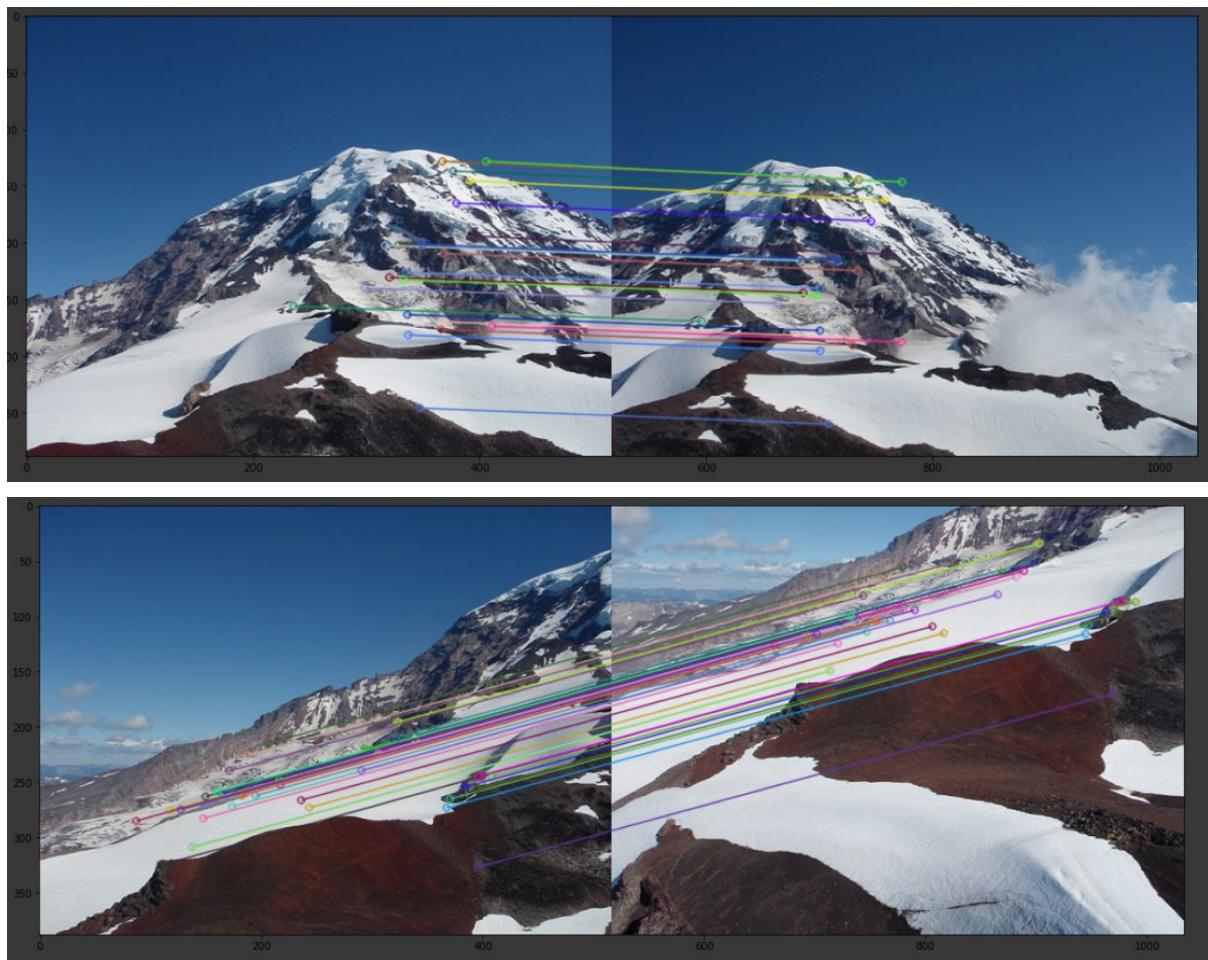


Figure II.6 : Détection du détecteur Surf d'OpenCV

IV. Homographie :

Maintenant que l'on dispose d'appariement entre deux images il va falloir mesurer les translations et rotations des différents pixels afin de pouvoir créer l'image panoramique. Dans les faits on va alors utiliser une matrice d'homographie et devoir résoudre une équation du type :

$$x = Hy$$

une homographie n'est rien d'autre qu'une matrice 3×3 comme indiqué ci-dessous, et représentée par la matrice H, x et y sont les coordonnées homogènes des deux pixels appariés.

Cette équation est donc une observation x et de y avec la matrice d'homographie H ici dans le cas d'une composition de transformation (rotation et translation pure de caméra (pas de mise à l'échelle)). On a donc une observation (x_1, y_1, x_2, y_2) où le coin détecté (x_1, y_1) doit être associé au coin (x_2, y_2) cela revient à devoir résoudre le système suivant :

$$\begin{bmatrix} wx_1 \\ wy_1 \\ w \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$

En développant $\frac{wx_1}{w}$ on obtient:

$$\begin{cases} x_1 = \frac{h_{00}x_2 + h_{01}y_2 + h_{02}}{h_{20}x_2 + h_{21}y_2 + h_{22}} \\ y_1 = \frac{h_{10}x_2 + h_{11}y_2 + h_{12}}{h_{20}x_2 + h_{21}y_2 + h_{22}} \end{cases}$$

Les inconnues du problèmes sont donc les h_{ij} qui sont les coefficient d'homographie:

$$\begin{cases} x_1(h_{20}x_2 + h_{21}y_2 + h_{22}) = h_{00}x_2 + h_{01}y_2 + h_{02} \\ y_1(h_{20}x_2 + h_{21}y_2 + h_{22}) = h_{10}x_2 + h_{11}y_2 + h_{12} \end{cases}$$

Le système se ramène sous forme matricielle à:

$$\begin{bmatrix} x_2 & y_2 & 1 & 0 & 0 & 0 & -x_1x_2 & -x_1y_2 & -x_1 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -y_1x_2 & -y_1x_2 & -y_1 \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Pour l'homographie le problème d'estimation est de l'ordre de K=4 comme on apparie deux points de coordonnées (x,y).

Afin d'obtenir 4 appariement correct pour calculer la matrice A il faut utiliser une stratégie de type Ransac sur la liste d'appariement obtenue aux étapes d'avant.

$$1 - p = (1 - w^n)^k$$

Il faut choisir n point de manière indépendante. On a p qui représente la probabilité que l'algorithme choisisse un bonne appariement, k le nombre d'itérations ,wⁿ représente la probabilité que l'ensemble des n points soit des points pertinents et 1 - wⁿ est donc la probabilité sur les point "Outlier".

k est obtenu avec :

$$k = \frac{\log(1 - p)}{\log(1 - w^n)}$$

Une fois les 4 points choisis par l'algorithme de Ransac (plus de détails sur l'algorithme de Ransac sont donnés en annexe) on doit donc résoudre $Ah = b$ qui s'écrit alors:

$$\begin{bmatrix} x_{2,1} & y_{2,1} & 1 & 0 & 0 & 0 & -x_{1,1}x_{2,1} & -x_{1,1}y_{2,1} \\ 0 & 0 & 0 & x_{2,1} & y_{2,1} & 1 & -y_{1,1}x_{2,1} & -y_{1,1}x_{2,1} \\ \vdots & \vdots \\ x_{2,n} & y_{2,n} & 1 & 0 & 0 & 0 & -x_{1,n}x_{2,n} & -x_{1,n}y_{2,n} \\ 0 & 0 & 0 & x_{2,n} & y_{2,n} & 1 & -y_{1,n}x_{2,n} & -y_{1,n}x_{2,n} \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \end{bmatrix} = \begin{bmatrix} x_{1,1} \\ y_{1,1} \\ \vdots \\ x_{1,n} \\ y_{1,n} \end{bmatrix}$$

On prend donc n=4 car 4 points sont suffisants pour reconstruire l'homographie. Le coefficient h_{22} est mis à 1 car le système est déterminé à un facteur multiplicatif, il y'a plus d'inconnue que d'équation. Pour résoudre ce système une méthode consiste à appliquer la solution par moindre carré sur $Ah = b$. Il faut alors:

- Calculer la décomposition en valeur singulière de la matrice A c'est à dire la diagonaliser telle que $A = UDV^T$, U la matrice des vecteurs à gauche, V la matrice des vecteurs à droite et D la Diagonale contenant les valeurs propres de A.
- On calcule ensuite $b' = U^T b$
- Calculer $y_i = b'_i / d_i$.
- La solution pour h est alors $h = Vy$

A partir des coefficient de la matrice H on peut alors établir l'image panoramique:

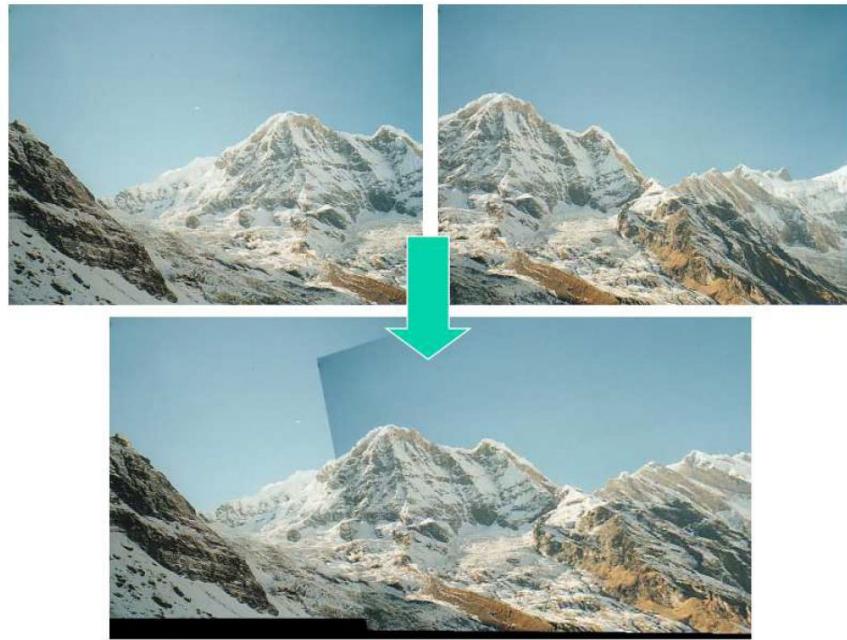


Figure III.1 : Exemple d'Homographie

Dans notre cas, avec l'algorithme de Fast et la mise en place de l'homographie, nous obtenons :

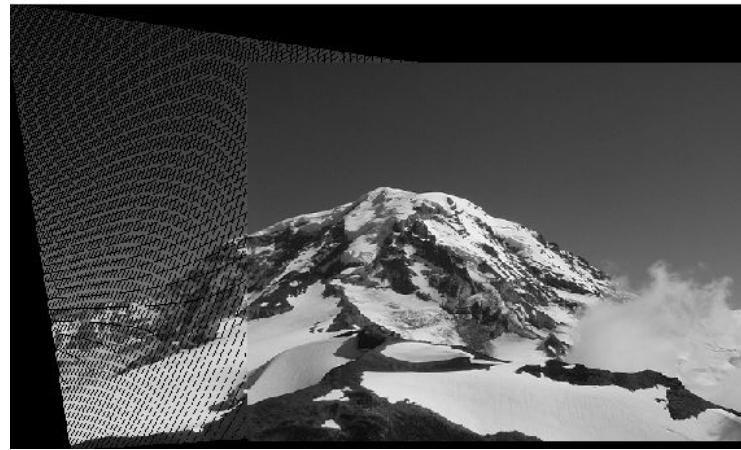


Figure III.2 : Une Homographie obtenue avec le détecteur Fast

Ici nous avons simplement pris 4 appariement obtenues avec le détecteur Fast pour obtenir cette Homographie sans stratégie de Ransac. Le résultat obtenu est plutôt satisfaisant cependant l'image n'est pas pleinement reconstituée. Il faut alors adopter une stratégie de transformation Backward afin de combler les espaces où l'homographie n'a pas projeté d'information.

La transformation Backward consiste à partir du résultat obtenu **Figure III.2** d'effectuer une transformation inverse en partant donc de l'homographie obtenue. Cela revient à calculer :

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}^{-1} \begin{bmatrix} wx_1 \\ wy_1 \\ w \end{bmatrix}$$

En effectuant cette opération cela permet de retrouver les plus proches voisins des zones où l'information est manquante et ainsi couvrir ces zones. On ajoute alors de l'information ce qui permet de combler les trous et ainsi obtenir une homographie complète entre 2 images.

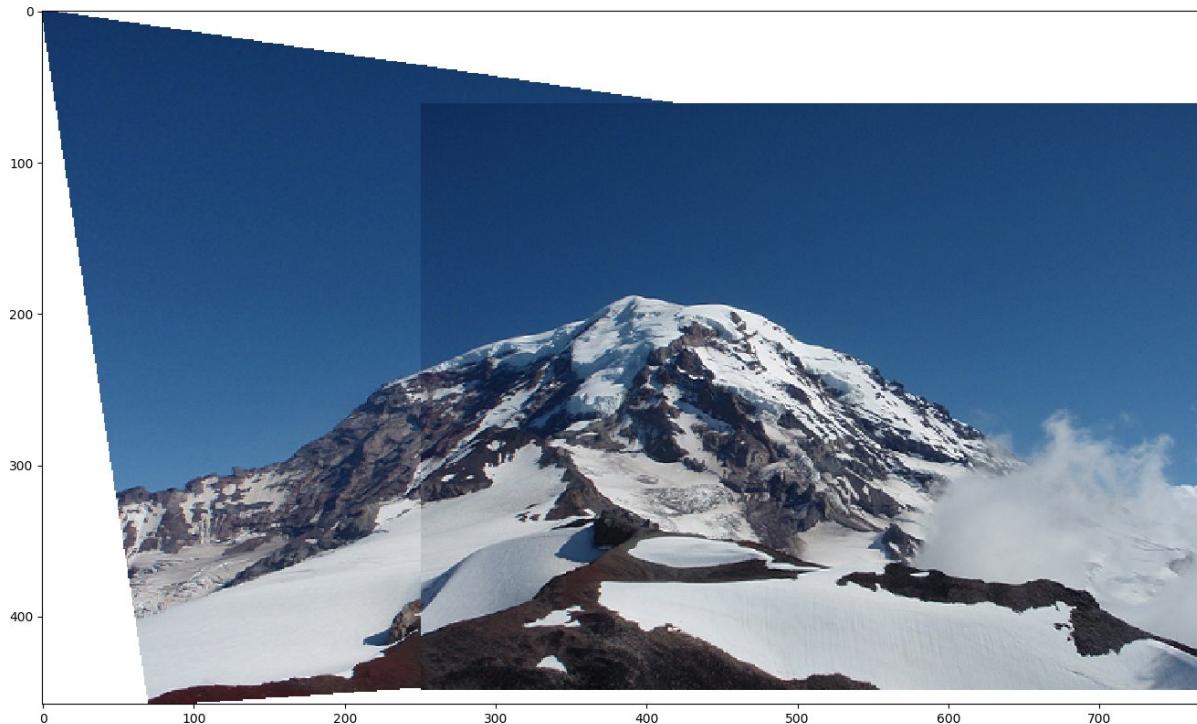


Figure III.3 : Homographie avec transformation Backward

Ainsi est complété le procédé de reconstruction homographique, il resterait une dernière étape consistant à venir égaliser l'intensité entre les deux images pour ne plus observer la séparation des deux images dans l'homographie.

V. Homographie et deep learning :

Il existe une méthode basée sur l'apprentissage machine qui permet de calculer une estimation d'une homographie entre 2 images, abordée dans l'article Deep Image Homography Estimation de Daniel DeTone, Tomasz Malisiewicz et Andrew Rabinovich [7]. Pour entraîner **efficacement** des réseaux de neurones convolutifs, il est nécessaire d'avoir une gigantesque quantité de données (aujourd'hui, on parle de quantités se mesurant en Téraoctets). En appliquant des homographies générées aléatoirement à des images issues d'un jeu de données volumineux, on peut générer une quantité colossale de données labellisées que l'on donnera "à manger" aux réseaux de neurones.

Étapes de l'algorithme (génération d'une donnée d'entraînement à partir d'une image)



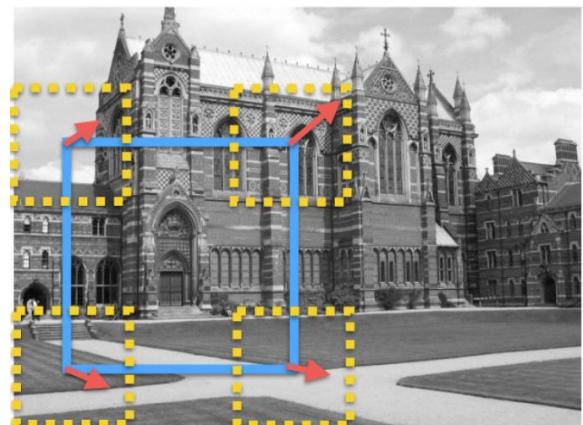
Étape 1

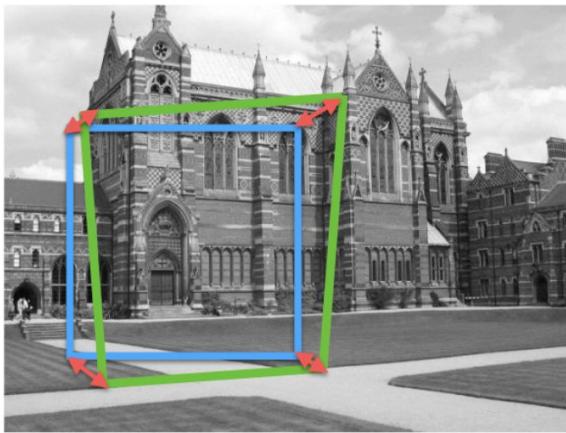
Créer un patch carré A en rognant l'image I à une position P (probablement un point) qui est déterminée aléatoirement.

On fait en sorte d'éviter les bordures de l'image pour éviter des effets indésirables (appelés *border artifacts*) lors de la génération de la donnée.

Étape 2

On "perturbe" les 4 coins du patch A en appliquant des translations aléatoires de valeurs situées dans un certain intervalle $[-\rho; \rho]$.





Étape 3

Les 4 correspondances ainsi perturbées permettent de définir l'homographie H^{AB} .

Étape 4

Appliquer l'inverse de cette homographie $H^{BA} = (H^{AB})^{-1}$ à l'image I .

À partir de l'image obtenue, on crée un patch B en rognant l'image à la position p .



Étape 5

Les 2 patches A et B issus des précédentes étapes sont fournis au réseau de neurones convolutifs. Il en résulte alors un paramétrage de l'homographie à 4 points.

Ce paramétrage est utilisé comme label d'entraînement pour les valeurs de vérité du terrain (*ground truths* - valeurs établies/fournies à partir d'observations directes).

Paramétrisation (à 4 points) d'une homographie

La manière la plus simple de paramétriser une homographie se fait à l'aide d'une matrice 3x3 et d'une échelle (de grandeur) fixe. L'homographie, définie à l'échelle donnée, permet d'appliquer une transformation sur un point, tel que pour tout point $[u, v]$:

$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Cependant, ce n'est pas une solution adaptée pour à un problème impliquant de l'optimisation puisqu'ici les termes de **rotation** et de **translation** sont mélangés :

$$\rightarrow \begin{bmatrix} H_{11} & H_{12} & H_{13} & H_{21} & H_{22} & H_{23} & H_{31} & H_{32} & H_{33} \end{bmatrix}$$

Ainsi, une solution plus moderne consiste à utiliser les coordonnées des 4 coins des 2 images (entrée et sortie) pour construire une paramétrisation à 4 points qui représente une homographie telle que :

$$H_{4-points} = \begin{bmatrix} \Delta_{u_1} & \Delta_{v_1} \\ \Delta_{u_2} & \Delta_{v_2} \\ \Delta_{u_3} & \Delta_{v_3} \\ \Delta_{u_4} & \Delta_{v_4} \end{bmatrix}$$

avec $\Delta_{x_i} = x'_i - x_i$, $x_i = u_i$ ou v_i , $i \in [1, 4]$

Cette paramétrisation à 8 valeurs, utilisée dans les méthodes traditionnelles d'estimation homographique devient ainsi utilisée dans des méthodes basées sur le machine learning. Une fois qu'on connaît les coordonnées d'un coin, on peut facilement convertir la paramétrisation en sa matrice associée (et vice-versa) avec, par exemple, l'algorithme de Transformation Linéaire Directe (DLT) ou la fonction d'OpenCV `getPerspectiveTransform()`.

On peut améliorer la robustesse de ce type de programme face à un type particulier de bruit d'image (comme les floutages ou les occlusions) en appliquant ce bruit à l'ensemble d'images d'entraînement. Cela passe par une gestion rigoureuse de la pipeline de génération des images d'entraînement.

Conclusion

Lors de ce projet nous avons pu comprendre le principe de l'homographie et d'une reconstruction panoramique. Au travers de la détection de points d'intérêt étant les coins d'une image, la ou le gradient est le plus élevé, on a pu voir qu'il existait différentes stratégies pour les détecter. On s'est donc intéressé à 3 détecteur de coins. Le premier, Harris est un des détecteur les plus utilisés et sûrement le plus répandu. Le second Sift est une amélioration se basant sur le détecteur de Harris et permet d'agrandir les possibilité de détections à plusieurs niveaux d'échelles. Enfin le troisième, Fast, est quant à lui une méthode qui contrairement aux deux premières n'utilise pas de descripteur et a pour but d'effectuer une recherche de coins la plus rapide possible. On a pu mettre en place le détecteur de Harris ainsi que le détecteur Fast. Pour le détecteur Sift on a simplement utilisé son implémentation dans OpenCV. Après avoir obtenu les points d'intérêts on a donc vu comment appairer les détections pour pouvoir établir la matrice d'homographie et ainsi calculer notre image panoramique.

Annexe

1- Les coordonnées homogènes:

les transformations de base s'écrivent comme suit:

$$\text{translation : } P' = T + P$$

$$T = \begin{bmatrix} T_0 \\ T_1 \end{bmatrix}$$

$$\text{scale: } P' = S * P$$

$$S = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$

$$\text{rotation } P' = R * P$$

$$R = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ \sin(-\theta) & \cos(\theta) \end{bmatrix}$$

dans ce cas là, si on veut combiner un changement d'échelle suivi d'une rotation, il suffit de faire:

$$P' = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ \sin(-\theta) & \cos(\theta) \end{bmatrix} P$$

et si on veut combiner une rotation suivie d'un changement d'échelle, il suffit de faire:

$$P' = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ \sin(-\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} P$$

Par contre, il est impossible d'intégrer la translation dans une composition de transformation sous forme de multiplication de matrice.

Afin de les utiliser de la même manière et pouvoir les combiner, on passe des coordonnées cartésiennes aux coordonnées homogènes. On augmente la dimension de l'espace afin d'écrire la translation en forme de multiplication de matrices.

Dans un système homogène, un point $P(x,y)$ est représenté par $P(X,Y,w)$. Pour chaque facteur possible $w \neq 0$, $X = x/w$, $Y = /w$, $Z = z/w$

le facteur w est fixé à 1, un point est donc représenté par $P(x,y,1)$, la translation peut être traitée comme une multiplication de matrices comme les deux autres transformations.

Remarque: On a ajouté une 3eme dimension.

$$\text{translation : } P' = T * P$$

$$T = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{scale: } P' = S * P$$

$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

rotation $P' = R * P$

$$R = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ \sin(-\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2- L'algorithme RANSAC

RANSAC, abréviation de « Random sample consensus », est une méthode itérative qui permet de déterminer, pour un ensemble de données contenant des « inliers » (données pertinentes) et des « outliers » (données aberrantes), les paramètres d'un modèle qui se rapproche au maximum de tous les « inliers ». Il s'agit d'un algorithme non déterministe dans le sens où il produit un résultat correct avec une certaine probabilité. [\[Ref1 : Abhijit Jana. 2012. Kinect for Windows SDK Programming Guide\]](#)

Publiée pour la première fois par Fischler et Bolles en 1981, cette méthode est très efficace lorsque les données contiennent beaucoup de bruit, et gagne de plus en plus de précision avec le nombre d'itérations appliquées. [\[Ref2: Microsoft. 2014. The Kinect for Windows v2 sensor and free SDK 2.0\]](#)

Les données d'entrée de l'algorithme sont l'ensemble de valeurs observées, un modèle paramétré (qui va être ajusté pour répondre aux observations), et pour finir un intervalle de confiance qui va dépendre de plusieurs variables :

- N : le nombre minimum de données nécessaires pour ajuster le modèle.
- K : le nombre maximal d'itérations de l'algorithme.
- T : une valeur seuil pour déterminer si une donnée correspond à un modèle ou pas.
- D : le nombre de données proches des valeurs nécessaires pour faire valoir que le modèle correspond bien aux données.

Les sorties de l'algorithme sont :

- Meilleur modèle : les paramètres du modèle qui se rapprochent au maximum des données (ou zéro si aucun bon modèle n'a été trouvé).
- Points du meilleur ensemble : les données utilisées à partir desquelles le meilleur modèle a été estimé.
- Erreur du meilleur modèle : l'erreur de ce modèle est choisie par rapport aux données. Algorithme :

Algorithm RANSAC

```
itérateur := 0
meilleur_modèle := aucun
meilleur_ensemble_points := aucun
meilleure_erreur := infini
while itérateur < k do
    points_aléatoires := n valeurs choisies au hasard à partir des données.
    modèle_possible := paramètres du modèle correspondant aux points_aléatoires.
    ensemble_points := points_aléatoires
    for chaque point des données pas dans points_aléatoires do
        if le point s'ajuste au modèle_possible avec une erreur inférieure à t then
            Ajouter un point à ensemble_points
        end if
    end for
    if le nombre d'éléments dans ensemble_points est > d then
        //ce qui implique que nous avons peut-être trouvé un bon modèle, on teste maintenant
        dans quelle mesure il est correct
        modèle_possible := paramètres du modèle réajusté à tous les points de ensemble_points

        erreur := une mesure de la manière dont ces points correspondent au modèle_possible
        if erreur < meilleure_erreur then
            //nous avons trouvé un modèle qui est mieux que tous les précédents, le garder
            jusqu'à ce qu'un meilleur soit trouvé
            meilleur_modèle := modèle_possible
            meilleur_ensemble_points := ensemble_points
            meilleure_erreur := erreur
        end if
    end if
    incrémentation de l'itérateur
    return meilleur_modèle, meilleur_ensemble_points, meilleure_erreur
end while
```

La figure Annexe 1 montre un exemple de l'application de l'algorithme sur un nuage de point en deux dimensions.

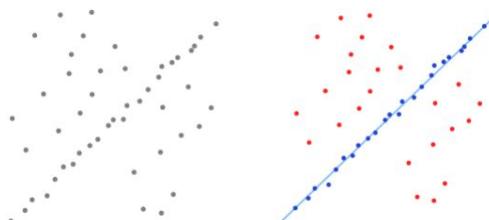


Figure Annexe 1: Utilisation de l'algorithme **RANSAC** pour trouver une droite dans un nuage de points 2D [Ref3: https://en.wikipedia.org/wiki/Random_sample_consensus]

Il existe quelques variations de l'algorithme, la plus connue étant celle qui arrête la boucle dès qu'un modèle avec une certaine marge d'erreur est trouvé, cela va minimiser le temps mais son application n'obtient pas forcément le meilleur modèle.

Avantages :

- Peut avoir une grande précision d'estimation d'un modèle, malgré la présence d'un nombre important de bruits.
- En calculant un plus grand nombre d'itérations, la probabilité de production d'un modèle raisonnable est augmentée.

Inconvénients :

- C'est une méthode non déterministe, autrement dit, rien ne garantit d'obtenir un résultat.
- Il n'y a pas de limite sur le temps qu'il faut pour calculer ces paramètres. [[Ref4: Martin A. Fischler, Robert C. Bolles. 1981. Random Sample Consensus : A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography](#)]

Référence:

- [1] **Frédéric Devernay**, “Détection de points d’intérêts”, cours de vision par ordinateur . INRIA.
Site référence auteur: <http://devernay.free.fr/>,
Site référence cours : <http://devernay.free.fr/cours/vision/pdf/c4.pdf>
- [2] **Julien Deshayes & Pierre Bouge**, “Création d’une image panoramique à partir de plusieurs images”, rapport de projet,. ISIMA, 2008.
Rapport de projet:
<https://www.isima.fr/f4/projets2007/rapport%20projet%20Bouges.pdf>
- [3] **David Low**, “Object Recognition from Local Scale-Invariant Features”, Conference Paper, 1999
Article: <https://www.cs.ubc.ca/~lowe/papers/iccv99.pdf>
- [4] **David G. Lowe**. “Distinctive Image Features from Scale-Invariant Keypoints”, Journal of Computer Vision, Computer Science Department University of British Columbia Vancouver, B.C., Canada, January 5, 2004.
Article: <https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf>
- [5] **Philippe Poublang**, site référence :<https://sites.google.com/site/poublangsift/>
Algorithme des sifts :
<https://sites.google.com/site/poublangsift/l-algorithme-des-sift>
- [6] **Edward Rosten, Reid Porter, and Tom Drummond**. 2008. Faster and better: a machine learning approach to corner detection. IEEE Trans. PAMI, 32 (2010), 105–119, 35p.
Article : <https://arxiv.org/pdf/0810.2434.pdf>
Présentation: https://www.edwardrosten.com/work/rosten_2008_faster_presentation.pdf
- [7] **Daniel DeTone, Tomasz Malisiewicz, Andrew Rabinovich**. 13 Juin 2016. Deep Image Homography Estimation. arXiv:1606.03798v1 [cs.CV] .
Article: <https://arxiv.org/pdf/1606.03798.pdf>