

# RANSAC générique et Panorama

**Diego de Rochebouët  
Laurin Brandner**

Analyse d'Images et Vision par Ordinateur

INF552  
École Polytechnique

## 1 Introduction

L'objectif de ce projet est d'implémenter l'algorithme RANSAC (RANdom SAmple Consensus) de manière générique, en utilisant tous les outils de C++, pour que celui là puisse être utilisé en nombreux problem complètement différents.

Pour tester RANSAC, on l'utilisera pour calculer une droite moyenne dans une nuage de points et puis on implémentera RANSAC pour calculer une homographie donné deux images.

Une fois complété et testé le cas de homographie, on utilisera ces homographies pour calculer des panorama données des listes d'images.

## 2 RANSAC

### 2.1 Theorie

RANSAC est un algorithme non-déterministe qui prend un ensemble de donnée et puis calcule un modèle qui représente ces données. Pour faire cela, RANSAC est un algorithme itératif qui dans chaque étape choisit un subset de mesures au hasard et estime le modèle représentant ce subset, retournant à la fin le meilleur modèle. Pour estimer quel modèle est le meilleur, il calcule l'erreur que chaque donnée a avec ce nouveau modèle et si cet erreur est moins qu'un seuil ce point est considéré comme un "inlier" et sinon c'est un "outlier". RANSAC retourne donc le modèle qui a plus de inliers. Du coup le modèle final n'est pas nécessairement le meilleur modèle et celui-là devient plus précis si on utilise plus itérations.

### 2.2 Implémentation

Pour implémenter RANSAC, comme l'implémentation était générique on a dû en premier trouver toute les variables qui étaient nécessaires pour l'algorithme.

RANSAC a besoin des arguments suivants pour estimer un modèle correctement:

<b>data:</b>	l'ensemble des points d'où on va obtenir le modèle
<b>minNumberOfDataPoints:</b>	le nombre minimum de données nécessaires pour obtenir le modèle
<b>maxNumberOfIterations:</b>	le nombre maximal d'itérations
<b>calculateParameters:</b>	fonction qui calcule le modèle à partir d'un ensemble de donnée
<b>errorThreshold:</b>	l'erreur maximum pour qu'une donnée soit considérée comme inlier
<b>calculateError:</b>	fonction qui calcule l'erreur d'une donnée avec un modèle
<b>bestFittingParameters:</b>	variable où va être stockée le résultat de l'algorithme.

RANSAC marche avec n'importe quel type de données si on lui donne ces variables. Mais comme on ne c'est pas quel type de donnée on va recevoir, c'est pourquoi tout doit être générique. Ici on montre comment c'est la déclaration de notre fonction RANSAC:

```
template <class Parameter_T , class Data_T ,
           class ChooseSubset_F , class CalculateParameter_F ,
           class CalculateError_F>
bool ransac(int minNumberOfDataPoints ,
              vector<Data_T> data ,
              CalculateParameter_F calculateParameters ,
              ChooseSubset_F chooseSubset ,
              double errorThreshold ,
              CalculateError_F calculateError ,
              int maxNumberOfIterations ,
              Parameter_T& bestFittingParameters ,
              vector<bool>* mask = NULL);
```

Comme on peut voir, pour pouvoir faire marcher la fonction générique on a dû attendre les type de donnée et les fonctions a utiliser comme des templates. En plus des paramètres nécessaires on ajoute les paramètres suivants:

**chooseSubset** Dans le cas de trouver l'homographie, c'est plus performant si on fait attention que le subset ne contient pas des points sur la même droite. Cela a du sens parce que pour chaque modèle estimé on doit le comparer à tout les données. Du coup c'est plus efficace si on tire le subset afin que la probabilité d'un bon résultat est déjà élevé. Par conséquent on a besoin de moins d'itérations.

**mask** RANSAC retourne de tous les modèles qu'il a trouvé celui qui a plus d'inliers. Alors, c'est utile à savoir quelles données étaient classifiées comme inliers ou outliers. Dans le cas de trouver l'homographie, en utilisant ce mask on peut afficher les matches qu'il a choisis comme bons.

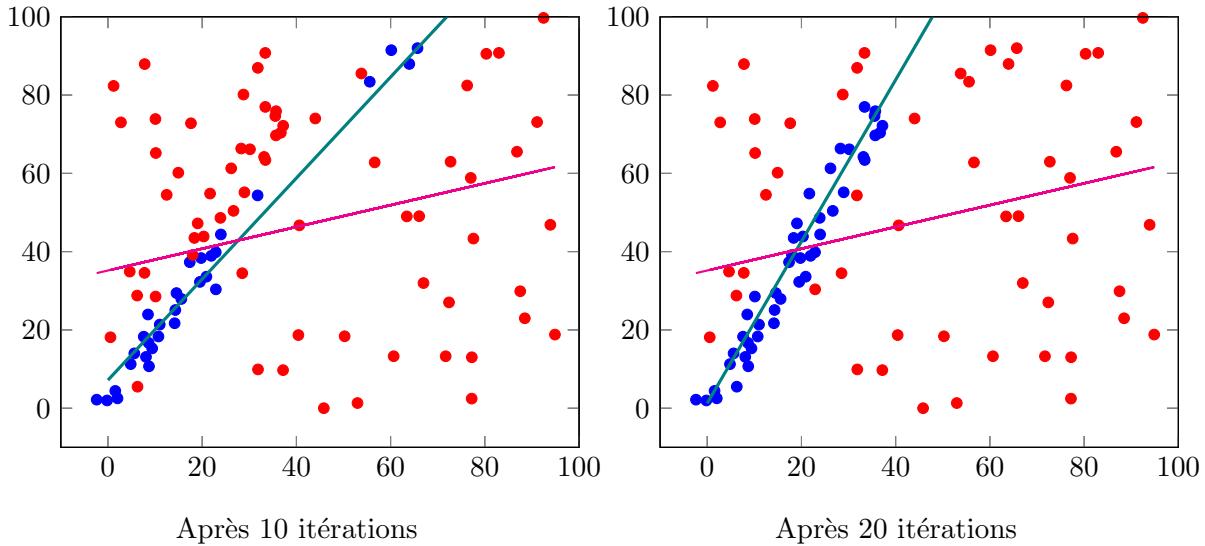
### 2.3 Droit Moyenne

Une foie implémenté RANSAC, cette fonction a été testé. Pour cela on a commencé par implémenter l'algorithme pour calculer la droite moyenne.

Dans ce cas, les donnée seraient une nuage de points avec le nombre minimum de donnée étant 2. Pour la fonction qui calcule les paramètres on a utilisé un fonction simple qui calcules les paramètres d'une droite à partir de deux points et la fonction de l'erreur était la distance carré d'un point a la droite.

Pour le tester on a appliqué l'algorithme à deux ensembles de données avec 100 mesures chacun. Une quantité des mesures sont situés au hasard et le rest est concentré environ sur la droite  $y = 2x$ . Pour le premier ensemble on utilise 40% des mesures sur la droite  $2x$  and pour le deuxième 20%, et le reste des points au hasard. Nous avons appliqué l'algorithme plusieurs fois à chaque ensemble en modifiant le nombre d'itération

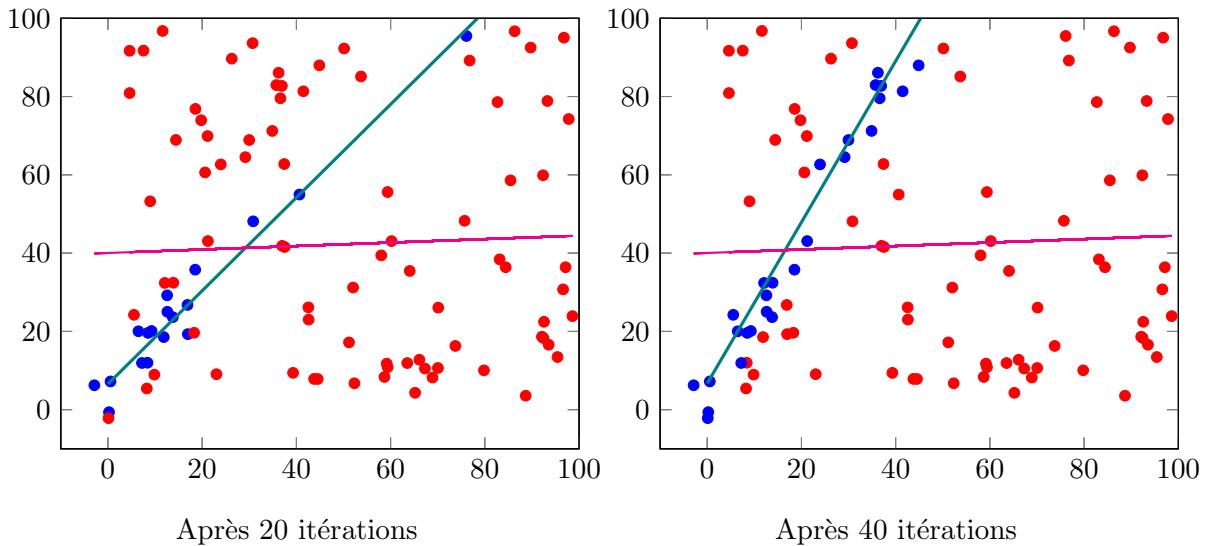
### 2.3.1 Ensemble 1



La ligne verte represent le résultat, la ligne pourpre represent la vrai moyenne d'ensemble

Pour la première ensemble, à gauche on a le résultat de l'algorithme avec 10 itérations et à droite avec 20. Il est facile de voir que le résultat est mieux si on le laisse exécuter avec plus d'itérations. Plus précisément, avec 10 itérations l'algorithme pouvais seulement trouver une ligne avec 31 inliers, après 20 itérations il trouvait 46.

### 2.3.2 Ensemble 2



La ligne verte represent le résultat, la ligne pourpre represent la vrai moyenne d'ensemble

On peut observer le même phénomène dans cet ensemble. Après 20 itérations l'algorithme a trouvé une ligne avec 18 inliers, après 40 itérations il pouvait identifier 26.

### 2.3.3 Conclusion

On voit donc que notre algorithme marche très bien dans le cas de la droite moyenne et on confirme comme on s'attendait que le nombre d'itération est très important. On a donc besoin d'être sûr qu'un bon nombre d'itération minimum est fait pour avoir un bon modèle.

## 3 Homographie

Réalisée la droite moyenne, on a passé à des test plus compliqué pour analyser vraiment si le RANSAC marchait bien. On l'a donc testé dans le cas des homographies.

### 3.1 Theorie

Une homographie est une matrice  $3 \times 3$  qui relie deux images qui ont la camera dans le même endroit sauf que la caméra pour la deuxième image à surfer une rotation  $R$ . Pour obtenir une homographie on a donc besoin de connaître les 9 variables de la matrice, mais comment une des variables sert à mettre à l'échelle la matrice, pour le différencier des matrices homogènes, on peut mettre le valeur dans la position  $(3,3)$  à 1 et calculer les autres 8 valeur. On a donc besoin de quatre pair de point entre les images qui correspondent au même point pour faire 8 équation et retrouver les valeur de homographie.

On a donc comme variables pour notre RANSAC:

<b>Données:</b>	C'est un ensemble de pair de points, des "matches", qui correspond au même point entre les deux images.
<b>Nombre minimum de donnée:</b>	4, puisqu'on a besoin de 4 pair de point pour calculer l'homographie
<b>Error threshold:</b>	Ce paramètre dépend du problème et dans ce cas on a utilisé 3.0 car les résultat était correct.
<b>CalculateParametres:</b>	Fonction qui calcule homographie
<b>CalculateError:</b>	Fonction qui avec un match et une homographie calcule un erreur.
<b>bestFittingParameters:</b>	On attend une homographie (représentée par une matrice $3 \times 3$ ).

### 3.2 Calcul Homographie

Si on a deux point  $m_1$  et  $m_2$  avec  $m_2 = H(m_1)$ .

$$\text{Pour obtenir } H = \begin{pmatrix} a, b, c \\ d, e, f \\ g, h, i \end{pmatrix}$$

$$\begin{aligned} \text{On a } u_2 &= \frac{a \cdot u_1 + b \cdot v_1 + c}{g \cdot u_1 + h \cdot v_1 + i} \\ v_2 &= \frac{d \cdot u_1 + e \cdot v_1 + f}{g \cdot u_1 + h \cdot v_1 + i} \\ u_2 &= \frac{a \cdot u_1 + b \cdot v_1 + c}{g \cdot u_1 + h \cdot v_1 + 1} \end{aligned}$$

Comme c'est à 1 coefficient près, on fix  $i = 1$ :

$$\begin{aligned} \text{Puis: } g \cdot u_2 \cdot u_1 + h \cdot u_2 \cdot v_1 + u_2 &= (a \cdot u_1 + b \cdot v_1 + c) \\ u_2 &= a \cdot u_1 + b \cdot v_1 + c - g \cdot u_2 \cdot u_1 - h \cdot u_2 \cdot v_1 \\ v_2 &= d \cdot u_1 + e \cdot v_1 + f - g \cdot u_1 \cdot v_2 - h \cdot v_1 \cdot v_2 \end{aligned}$$

On écrit alors une matrice  $A$  avec les coefficient de l'équation et un vecteur  $b$  avec les valeur  $(u, v)$  pour les quatre pairs de points tel que:

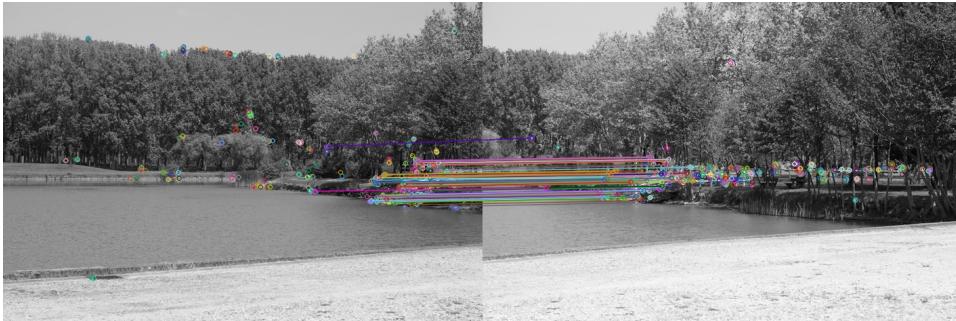
$$AH = b \Rightarrow H = A^{-1}b$$

### 3.3 Calcul erreur

Pour calculer l'erreur d'une donnée avec une homographie on projète le point de la deuxième image dans la première image et puis on utilise la distance euclidienne de ces points.

### 3.4 Test Homographie entre deux Images

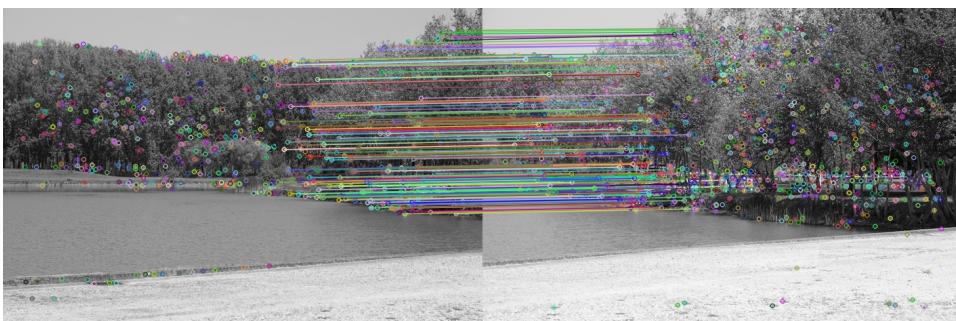
On a donc testé notre homographie avec des images d'un panorama 360° de l'X. Et on a vu que même si en la plupart des cas on avaient des bons résultats, en beaucoup d'autres les pair d'images n'avaient pas une bonne homographie. On analysant pourquoi on a vu que les matches n'étaient pas idéales et même en utilisant `findHomography()` de openCV on avait des mauvais résultats. On a donc testé avec des différentes méthodes pour trouver les matches et on a vu qu'avec AKAZE on avaient des result presque parfait par rapport à ORB, celui qu'on était entrain d'utiliser. Sauf pour quelques paires d'images où les résultats restaient mauvais.



Matching en utilisant ORB



Le pano résultant en utilisant ORB



Matching en utilisant AKAZE



Le pano résultant en utilisant AKAZE

Une autre différence qu'on a vu c'est que notre algorithme étaient beaucoup plus lent que `findHomography()`. En analysant plus précisément on a réalisé que c'était dû au nombre d'itération.

FindHomography() pouvait trouver un bonne homographie en 100 iteration pendant que nous on trouvait pas et devait faire plutôt 2000-3000 iteration. Pour améliorer cela on donc essayé en améliorant notre RANSAC. Au lieu de retourner l'homographie avec plus de inliers, retourner une homographie avec un minimum de inliers et avec le moins possible d'erreur moyen. Mais cela n'a pas marché, le nombre de inlier étant un bonne mesure de la qualité du modèle. On a donc ajouté pour seulement essayé de faire l'homographie quand les points n'était pas linéaires et c'est là que le nombre iteration nécessaires pour trouver un bon résultat c'est réduit beaucoup.

## 4 Panorama

Une fois décidé que l'homographie marchait correctement avec notre RANSAC, on a passé à l'étape suivante; utiliser notre RANSAC et les homographies pour faire un panorama avec les images de l'X. L'algorithme devrait trouver une manière de relier toute les images en une seule grande image horizontale.

Originalement on allait faire un calcul avec les images dans n'importe quel ordre, on prenant à chaque itération la pair d'image qui a le plus de matches. Mais vu que l'utilisateur du programme qui calcule des homographie a rarement besoin de faire un panorama avec des images sans ordre, vu que normalement on prend les images dans l'ordre et sinon on peut leur donner un ordre manuellement, donc on a décidé de faire un algorithme qui attend les images dans l'ordre de gauche à droite.

### 4.1 Premier Essai

Notre première idée était de prendre la première image et de calculer l'homographie qui correspond de chaque image à la première. Pour faire cela on calcule tous les pair d'homographies  $H_i$  que relient un image avec l'image suivante. Une fois calculé cela on peut calculer  $G_i$ , les homographie entre la première images et les autres images. On calcule donc itérativement  $G_i = G_{i-1} \cdot H_i$ .

On project les images une à une en utilisant ces homographies pour obtenir le panorama. On faisant cela et on obtient:



Le pano composé de 5 images



Le pano composé de 6 images

Comme on peut voir, les images s'étirent de plus en plus. Résultant en un panorama de plus en plus mauvais.

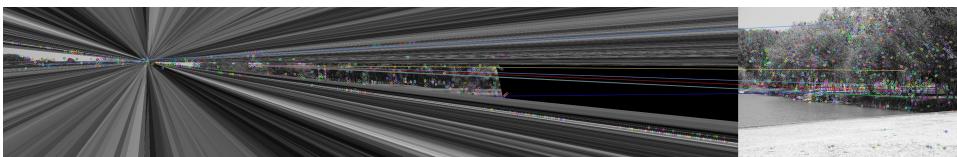
Ce qui a passé c'est comme c'est un panorama 360, à chaque étape il y a une distorsion de l'image pour projeter l'image sur l'autre, et cette distorsion est plus grande quand plus grand c'est l'angle et cet erreur s'accumule avec les successives opérations. Le résultat était donc pas bon et on pouvait faire un maximum de 4-5 images pour le panorama avant d'avoir un résultat mauvais.

## 4.2 Deuxième Essai

Pour un deuxième essai, comme on voulais réduire le problème de la distorsion à cause des grands angles de rotation, au lieu de calculer les homographies de la première image aux autres on a décidé de faire un panorama auxiliaire composé des deux premières images et puis itérativement calculer l'homographie entre le panorama auxiliaire et l'image suivante.



Le matching en utilisant 3 images



Le matching en utilisant 4 images

Comme on peut voir, cet algorithme a des problèmes car à mesure que le panorama auxiliaire devient plus grand, c'est de plus en plus difficile de trouver les matches, ce qui provoque qu'en quelques itérations le résultat soit complètement mauvais.

Mais vu que les matches peuvent être seulement entre la dernière image ajoutée au panorama et la nouvelle image, et donc de la dernière partie du panorama à la nouvelle image, on a changé l'algorithme pour seulement chercher des matches entre la nouvelle image et une section du panorama auxiliaire qui avait une distance du bord plus petite que la longueur d'une image. De cette manière on chercherait seulement les matches dans la section correspondante à la dernière image, ce qui devrait réduire le nombre de mauvais matches.

Avec cette nouvelle condition on a obtenu ces résultats:



Le matching en utilisant 4 images



Le matching en utilisant 5 images

Cela a amélioré beaucoup le résultat mais également le résultat n'est pas idéal vu que à chaque itération les nouvelles images sont de plus en plus déformées du à leur projection et donc les matches sont de plus en plus mauvais jusqu'à ce que l'homographie devienne fausse.

### 4.3 Troisième Essai

Vu que l'erreur était de plus en plus grand à chaque itération, on a décidé qu'au lieu d'avoir un grand panorama auquel on ajoutait des image une à une, on regrouperait les images stile arbre binaire. Cela veut dire, regroupant les images en paires pour créer des nouvelles images et itérant de cette manière jusqu'à qu'on ait une seule image. De cette manière, l'erreur dans les bord de l'image n'est pas si grand car les images sont dans une profondeur  $\log(n)$  dans l'arbre, au lieu d'être à une profondeur  $n$  (maximum). Avec ce troisième essaie on a obtenu ces résultats:



L'algorithme binaire appliqué à 8 images

Comme on peut voir, les résultat continuent à ne pas être parfait, mais maintenant c'est pas du à un erreur accumulé mais dû au fait que quelques paires d'images ont des homographies très mauvaises, que c'est le cas de quelque images du panorama de la cour de Polytechnique où la section où quelques images se superposent est très petite.

## 5 Conclusion

L'algorithme de RANSAC est un algorithme avec une logique très simple mais très puissant, vu que celui là peut être utilisé dans n'importe quel problème où on doit trouver un modèle à partir de ensemble de donnée. Seulement on doit connaitre comment calculer le modèle à partir des donnée (et donc aussi le nombre minimum des donnée nécessaires) et comment calculer l'erreur du modèle. Et vu que ces fonctions sont normalement naturelle au problèmes, elle ne sont pas difficiles de trouver, ce qui permet avoir une rapide et facile solution au problèmes de ce type. D'une autre part, on peut voir que nos panoramas fonctionnent plutôt bien, mais si et seulement si les homographies originels fonctionnait bien, ça serait intéressant d'essayer de trouver d'autre manières pour trouver les homographies et les tester pour voir si il y a une méthode qui, même si elle est plus lente, fonctionne mieux dans ces cas.

Et dernièrement, notre panorama ne marche que pour un panorama horizontal. Ça serait aussi très intéressant d'élargir cette algorithme pour qu'il puisse créer des panorama aussi verticalement, et puis faire des panorama sphériques.

## Sources

**RANSAC:** Wikipedia, [https://en.wikipedia.org/wiki/Random\\_sample\\_consensus](https://en.wikipedia.org/wiki/Random_sample_consensus)  
**Calculating Homography:** Per Rosengren, <http://www.csc.kth.se/~perrose/files/pose-init-model/node17.html>