



ENSTA

IP PARIS

ENSTA PARIS  
INSTITUT POLYTECHNIQUE DE PARIS

---

## CSC\_5RO17\_TA, Vision 3D

Travail Pratique 1

---

by

Guilherme NUNES TROFINO and Luiz Henrique MARQUES GONÇALVES

supervised by

Antoine MANZANERA

François GOULETTE

Marwane HARIAT

**Confidentiality Notice**  
Non-confidential and publishable report

ROBOTIQUE  
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET COMMUNICATION

---

Paris, France  
5 novembre 2024

# Table des matières

---

<b>1 Estimation et Correction d'une Homographie</b>	<b>2</b>
1.1 Question 1 . . . . .	2
1.2 Question 2 . . . . .	3
1.3 Question 3 . . . . .	4
1.3.1 Direct Linear Transformation . . . . .	4
1.3.2 Normalisation . . . . .	4
1.3.3 Résultats . . . . .	4
1.3.4 Erreur . . . . .	5
1.3.5 Analyse . . . . .	5
<b>2 Création de Panorama</b>	<b>6</b>
2.1 Question 4 . . . . .	6
2.1.1 Homographie . . . . .	6
2.1.2 Correction de couleurs . . . . .	7
2.1.3 Filtre de fusion progressive . . . . .	7
2.2 Questions . . . . .	8

## 1. Estimation et Correction d'une Homographie

### 1.1. Question 1

**Définition 1.1.** Une **Homographie**  $\mathbf{H}$  est une transformation projective entre deux plans  $\pi_1$  et  $\pi_2$  qui conserve les lignes droites.

Une **Homographie** peut-être représentée par l'équation suivant :

$$\underbrace{\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}}_{\mathbf{x}_2} = \underbrace{\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ v_1 & v_2 & 1 \end{bmatrix}}_{\text{Homographie, } \mathbf{H}} \times \underbrace{\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}}_{\mathbf{x}_1} \quad \text{où} \quad \begin{cases} \mathbf{a} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \\ \mathbf{t} = (t_x \ t_y)^\top \\ \mathbf{v} = (v_1 \ v_2) \end{cases} \quad (1.1)$$

Où :

1. **x<sub>2</sub>, Vecteur Destination** : coordonnées après l'homographie ;
2. **x<sub>1</sub>, Vecteur Source** : coordonnées avant l'homographie ;
3. **a, Matrix d'Affinité** ;
4. **t, Vecteur de Translation** ;
5. **v, Vecteur de Relation** ;

**Remarque.** Dans ce travail pratique, une vecteur  $\mathbf{x}_i = [x_i \ y_i \ 1]^\top$  sera construit à partir d'un point  $\mathbf{p}_i = (x_i \ y_i)$ .

En termes simples, cela signifie que si deux images d'une scène sont prises sous certaines conditions géométriques, la relation entre ces images peut être modélisée par une **Homographie**.

Cas où la transformation est une homographie :

1. images d'un même plan vu sous deux poses 3D différentes ;
2. images prises par une caméra tournant autour de son centre optique ;

**Remarque.** Les images destination et source devront avoir des données en commun pour que la transformation soit considérée comme une homographie.

## 1.2. Question 2

Étant donné qu'une **homographie** contient 8 degrés de liberté, représentés par 8 inconnues, au moins 4 points  $\mathbf{p}_i = (x \ y)$  avec des coordonnées  $x$  et  $y$  sont nécessaires. Plus de points peuvent être utilisés, mais dans ce cas, la solution ne sera pas unique.

Le choix des points est nécessaire et essentiel pour l'application de l'homographie. Normalement, ces points sont définis à l'aide d'algorithmes. Dans ce travail pratique, cependant, les points seront choisis manuellement et sont notés ci-dessus :

-	$x$	$y$
$\mathbf{p}_{s_1} = \begin{pmatrix} x_{s_1} & y_{s_1} \end{pmatrix}$	120	023
$\mathbf{p}_{s_2} = \begin{pmatrix} x_{s_2} & y_{s_2} \end{pmatrix}$	073	287
$\mathbf{p}_{s_3} = \begin{pmatrix} x_{s_3} & y_{s_3} \end{pmatrix}$	409	299
$\mathbf{p}_{s_4} = \begin{pmatrix} x_{s_4} & y_{s_4} \end{pmatrix}$	386	022

(a) Points Source

-	$x$	$y$
$\mathbf{p}_{d_1} = \begin{pmatrix} x_{d_1} & y_{d_1} \end{pmatrix}$	083	000
$\mathbf{p}_{d_2} = \begin{pmatrix} x_{d_2} & y_{d_2} \end{pmatrix}$	083	333
$\mathbf{p}_{d_3} = \begin{pmatrix} x_{d_3} & y_{d_3} \end{pmatrix}$	416	333
$\mathbf{p}_{d_4} = \begin{pmatrix} x_{d_4} & y_{d_4} \end{pmatrix}$	416	000

(b) Points Destination

TABLE 1.1 : Points Homographie

**Remarque.** Toutes les coordonnées sont données en pixels.

Pour maximiser l'effet de recadrage, les points de destination ont été choisis de façon à ce que le carré ait pour dimension le minimum entre l'hauteur, 333 pixels, et la largeur, 500 pixels, de l'image. Dans ce cas, cependant, le carré aura 333 pixels de largeur et sera centré sur le centre de l'image.

Ci-dessous, les points sont présentés sur l'image source et sur l'image destination :



(a) Image Source



(b) Image Destination

FIGURE 1.1 : Points choisis pour l'Homographie

Dans cette approche, il faut au moins 4 correspondances de points entre l'image source  $\mathbf{p}_{s_i}$  et l'image destination  $\mathbf{p}_{d_i}$  afin d'établir un système d'équations à résoudre par une méthode telle que la Transformation Linéaire Directe.

Les points choisis sont les points d'intérêt de la transformation requise, les quatres coins du carré à transformer. Ils ont été choisis avec l'aide de l'interface graphique déjà disponible, puis maintenus pour garantir la reproductibilité des résultats.

### 1.3. Question 3

#### 1.3.1. Direct Linear Transformation

À partir de la définition de l'Homographie, l'application de la Direct Linear Transformation (**DLT**) donne l'équation suivante :

$$\underbrace{\begin{bmatrix} -x_{11} & -y_{11} & -1 & 0 & 0 & 0 & x_{21}x_{11} & x_{21}y_{11} & x_{21} \\ 0 & 0 & 0 & -x_{11} & -y_{11} & -1 & y_{21}x_{11} & y_{21}y_{11} & y_{21} \\ & & & & \vdots & & & & \\ -x_{1i} & -y_{1i} & -1 & 0 & 0 & 0 & x_{2i}x_{1i} & x_{2i}y_{1i} & x_{2i} \\ 0 & 0 & 0 & -x_{1i} & -y_{1i} & -1 & y_{2i}x_{1i} & y_{2i}y_{1i} & y_{2i} \\ & & & & \vdots & & & & \\ -x_{1N} & -y_{1N} & -1 & 0 & 0 & 0 & x_{2N}x_{1N} & x_{2N}y_{1N} & x_{2N} \\ 0 & 0 & 0 & -x_{1N} & -y_{1N} & -1 & y_{2N}x_{1N} & y_{2N}y_{1N} & y_{2N} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} a_{11} \\ a_{12} \\ t_x \\ a_{21} \\ a_{22} \\ t_y \\ v_1 \\ v_2 \\ 1 \end{bmatrix}}_{\mathbf{h}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (1.2)$$

Cet équation peut être résolu à l'aide du méthode de Singular Value Decomposition (**SVD**) :

$$\underbrace{\mathbf{A}}_{2N \times 9} = \underbrace{\mathbf{U}}_{2N \times 9} \underbrace{\mathbf{S}}_{9 \times 9} \underbrace{\mathbf{V}}_{9 \times 9} \quad \text{avec} \quad \begin{cases} \mathbf{U}^\top \mathbf{U} = \mathbf{I}_9 \\ \mathbf{V}^\top \mathbf{V} = \mathbf{I}_9 \end{cases} \quad (1.3)$$

Où  $\mathbf{h}$  est donné par la dernière ligne de la matrice  $\mathbf{V}$  avec les plus petits valeurs propres.

#### 1.3.2. Normalisation

L'approche **DLT** permet algébriquement de trouver l'homographie nécessaire pour récadre une image avec une erreur proportionnelle à la taille de l'image. À fin de miniser l'influence de l'erreur sur les calculs une normalisation de l'image pourrait être appliquée.

La normalisation peut être fait avec la transformation suivant :

$$\mathbf{T} = \begin{bmatrix} \frac{2}{w} & 0 & -1 \\ 0 & \frac{2}{h} & -1 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{où} \quad \begin{cases} w & \text{width de l'image} \\ h & \text{height de l'image} \end{cases} \quad (1.4)$$

Cette transformation resize l'image pour que elle ait taille 2 par 2 et depuis change l'origine de l'image pour le centre du carré.

L'utilisation de cette méthode se fait comme explique ci-dessus :

1. application de la normalisation sur les coordonnées :

$$\tilde{\mathbf{x}}_{1i} = \mathbf{T}\mathbf{x}_{1i} \quad \tilde{\mathbf{x}}_{2i} = \mathbf{T}\mathbf{x}_{2i} \quad (1.5)$$

2. application du méthode **DLT** avec les coordonnées normalisées

3. application de la de-normalisation sur les coordonnes :

$$\mathbf{H} = \mathbf{T}^{-1} \tilde{\mathbf{H}} \mathbf{T} \quad (1.6)$$

**Remarque.** Aucune changement est nécessaire pour l'algorithme **DLT**.

#### 1.3.3. Résultats

Après cette calcule théorique, ses expressions ont été implementées en Python et l'image suivante a été obtenu :

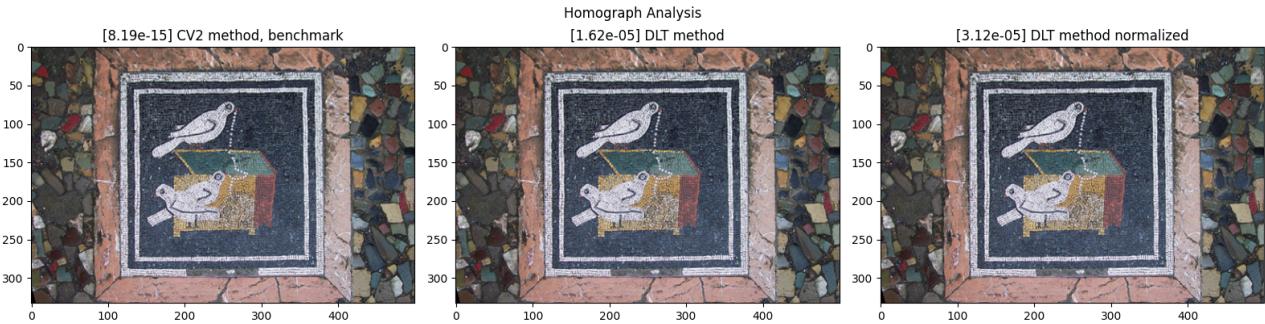


FIGURE 1.2 : Application de l'Homographie

L'image plus à gauche présent le résultat obtenu avec la fonction `cv2.homography()`, l'image au milieu présent le résultat obtenu avec le méthode **DLT** sans normalisation et l'image plus à la droite présent le résultat obtenu avec le méthode **DLT** avec normalisation.

### 1.3.4. Erreur

Sur le titre de chaque image, l'erreur est présente pour que les différentes méthodes puissent être comparées. Dans ce cas, l'indice de confiance choisi a été **l'Erreure de Reprojection**, qui permet de mesurer la proximité des points projetés par l'homographie par rapport aux points de l'image originale. Ainsi, plus l'erreur de reprojection est petite, meilleure est la correspondance entre les points de la source et les points corrigés.

L'Erreure de Reprojection est calculée avec l'équation ci-dessous :

$$\text{ER} = \sqrt{\sum_{i=0}^n \|\mathbf{p}_{d_i} - \mathbf{p}_{s_i}\|} \quad (1.7)$$

Avec l'implémentation suivante en Python :

```

1 def get_reprojection_error(homograph: np.array, src_pts: np.array, dst_pts: np.array) -> float:
2     """
3         Return reprojection error of homograph.
4
5         Note: how closely the projected points via the homograph match the actual points in the image.
6
7         Args:
8             homograph (np.array) : homograph matrix.
9             src_pts (np.array) : source image points.
10            dst_pts (np.array) : destination image points.
11
12    """
13    projected_pts = cv2.perspectiveTransform(src_pts.reshape(-1, 1, 2), homograph).reshape(-1, 2)
14    error = np.sqrt(np.sum((projected_pts - dst_pts) ** 2, axis=1))
15
16    return np.mean(error)

```

### 1.3.5. Analyse

Il est notable qu'avec cette erreur, la méthode normalisée entraîne une erreur plus importante, environ le double, par rapport à la méthode non normalisée. Cela pourrait être causé par la taille de l'image, qui n'était pas très élevée, et le fait que la dénormalisation nécessite une inversion de matrice, ce qui peut accumuler des erreurs numériques significatives.

Le résultat obtenu avec OpenCV est ajouté à l'image ci-dessus comme référence d'un résultat idéal, et comme attendu, c'est la méthode qui présente la plus petite erreur parmi les trois méthodes implémentées, montrant l'optimisation des fonctions utilisées par OpenCV.

Il est curieux que la normalisation, dans ce cas, présente un résultat marginalement moins efficace que la version non normalisée. Cela dit, cet effet peut être dû au fait que la taille de l'image originale n'est pas très grande et que l'erreur de l'homographie est moins significative que l'erreur numérique introduite par l'inversion de matrices nécessaire à la normalisation.

## 2. Création de Panorama

### 2.1. Question 4

Le fichier *q1.py*, associé à la bibliothèque de fonctions *q1.lib.py*, contient le code avec les modifications nécessaires pour réaliser le panorama. Différentes techniques, en plus de la simple homographie avec des points communs arbitraires, ont été utilisées pour obtenir des résultats plus intéressants, comme le choix stratégique des points, la correction de couleur et l'application d'un filtre gaussien dans les régions superposées. Ces développements sont expliqués ci-dessous.

#### 2.1.1. Homographie

La première idée pour la construction du panorama était de réaliser une homographie sur les images latérales en utilisant des correspondances arbitraires de points. Nous avons pris soin de translater les images de manière à ce que ces points se superposent. Bien qu'il ait été possible d'obtenir un résultat visuellement continu, certaines combinaisons de points sélectionnés créaient des trous près des coins des images, entraînant une visualisation peu agréable pour un panorama, comme le montre l'exemple ci-dessous.

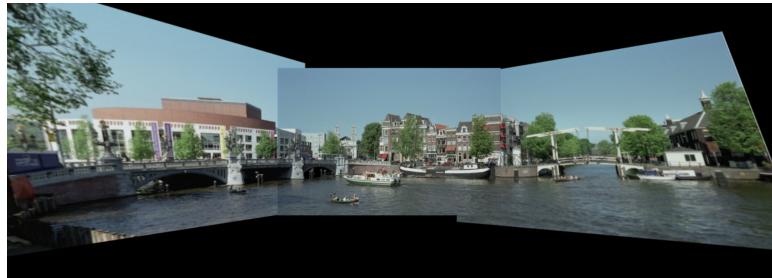


FIGURE 2.1 : Résultat sans traitement.

Pour résoudre ce problème, la première solution envisagée consistait à mieux sélectionner les points correspondants afin de mieux aligner les coins des images et d'éviter ces zones noires. Pour cela, nous avons ajouté des points supplémentaires sur les bords supérieur et inférieur (à la même position en y que le point le plus proche du bord) aux points utilisés pour le calcul de l'homographie. Voici le code correspondant :

```

1 # Add boundary points for alignment
2 X_init1.append([max([point[0] for point in X_init1]), new_height - 1])
3 X_init1.append([max([point[0] for point in X_init1]), 0])
4 X_final2.append([min([point[0] for point in X_final2]), new_height - 1])
5 X_final2.append([min([point[0] for point in X_final2]), 0])
6 X_final1.append([max([point[0] for point in X_final1]), new_height - 1])
7 X_final1.append([max([point[0] for point in X_final1]), 0])
8 X_init2.append([min([point[0] for point in X_init2]), new_height - 1])
9 X_init2.append([min([point[0] for point in X_init2]), 0])

```

En général, cette approche a conduit à des résultats plus satisfaisants, comme le montre la figure 2.2 suivante.

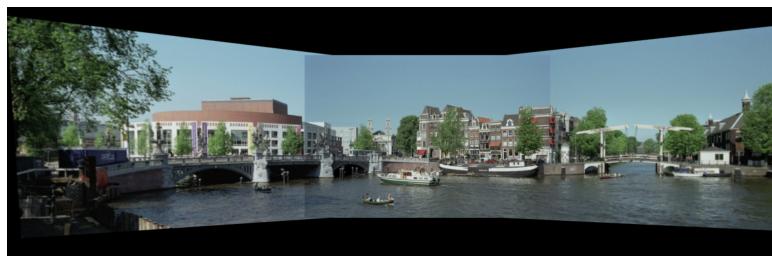


FIGURE 2.2 : Résultat avec une choix stratégique des points.

Cependant, une discontinuité importante persistait dans les transitions entre les images, nécessitant d'autres ajustements.

### 2.1.2. Correction de couleurs

Une première idée pour résoudre cette différence observée a été de réaliser un traitement de couleur afin d'égaliser les tonalités entre les deux images. Pour cela, nous avons choisi un point dans chaque image (le premier à être sélectionné pour l'homographie) qui devrait avoir la même couleur. Idéalement, ce point se trouve sur une surface uniforme, comme le ciel dans cette image d'Amsterdam, afin d'obtenir de bons résultats. Ensuite, nous avons appliqué à toute l'image latérale la différence de couleur entre ces points pour les égaliser. Voici le code correspondant :

```

1 def fix_color(img_source, img_reference, x_source, y_source, x_ref, y_ref):
2
3     # Get pixel colors for color adjustment
4     color_source = img_source[y_source, x_source].astype(np.float32)
5     color_reference = img_reference[y_ref, x_ref].astype(np.float32)
6
7     # Calculate the necessary color adjustments and applies
8     diff = color_reference - color_source
9     img_source = cv2.add(img_source.astype(np.float32), diff)
10
11    # Clip pixel values to valid range
12    img_source = np.clip(img_source, 0, 255).astype(np.uint8)
13
14    return img_source
15
16 # Apply color adjustments to images
17 img1 = fix_color(img1, img2, int(X_init1[0, 0]), int(X_init1[0, 1]), int(X_final1[0, 0]), int(X_final1[0, 1]))
18 img3 = fix_color(img3, img2, int(X_init2[0, 0]), int(X_init2[0, 1]), int(X_final2[0, 0]), int(X_final2[0, 1]))

```

Cette approche a considérablement amélioré les transitions entre les images, comme on peut le voir dans la Figure 2.3. La petite disparité restante a été résolue en appliquant un filtre pour adoucir le gradient.



FIGURE 2.3 : Résultat l'égalisation de couleurs.

### 2.1.3. Filtre de fusion progressive

Enfin, pour résoudre les artefacts de division restants, un filtre en dégradé a été ajouté pour assurer une transition plus fluide dans les régions superposées, comme illustré dans l'extrait de code ci-dessous :

```

1 ...
2
3     # Blend img1 and img2 in the left overlap region
4     elif left_blend_start <= col < left_blend_start + blend_region_width:
5         blend_position = col - left_blend_start
6         alpha = blend_position / blend_region_width
7         img2[row, col] = (1 - alpha) * img1_warp[row, col] + alpha * img2[row, col]
8
9     # Blend img2 and img3 in the right overlap region
10    elif right_blend_end - blend_region_width <= col < right_blend_end:
11        blend_position = col - (right_blend_end - blend_region_width)
12        alpha = blend_position / blend_region_width
13        img2[row, col] = (1 - alpha) * img2[row, col] + alpha * img3_warp[row, col]

```

Ce filtre de fusion progressive permet de rendre la transition entre les images plus homogène dans les zones de chevauchement, en utilisant un coefficient  $\alpha$  qui varie linéairement en fonction de la position dans la région

de superposition. Cela permet de réduire les discontinuités visibles et de créer un rendu plus cohérent dans la composition finale.



FIGURE 2.4 : Résultat final.

Les mêmes résultats présentés dans ce rapport pour les autres images exemples peuvent être obtenus en exécutant le code avec la flag *USE\_PRE\_SELECTED\_POINTS* égale à *True*.

## 2.2. Questions

### 1. Quelle condition doit respecter la prise de vue entre les deux images ?

Pour réaliser l'homographie entre deux images, il est essentiel que ces dernières possèdent un recouvrement suffisant. Cela permet de détecter des points d'intérêt communs. De plus, la différence de point de vue entre les deux images doit être essentiellement une simple rotation. En effet, c'est uniquement dans ce cas que la transformation de l'image projetée peut être décrite par une homographie, et elles représentent alors une bonne approximation du point de vue central élargi.

### 2. Comment retrouver les paramètres de cette transformation à partir de ceux de l'homographie ?

Comme indiqué dans le matériel de cours, pour une rotation simple, il est possible de réécrire la transformation sous la forme :

$$\tilde{m}' = K R K^{-1} \tilde{m}$$

Dans ce contexte, l'homographie, qui est connue, est égale à  $K R K^{-1}$ , où  $R$  contient les informations sur la rotation réalisée, et  $K$  représente les paramètres intrinsèques de la caméra. Par conséquent, si nous connaissons les caractéristiques de la caméra utilisée, nous pouvons retrouver  $R$  et ainsi obtenir les paramètres de la rotation observée.