

AIV1 - TP04
Mise en correspondance d'images

Suliac Lavenant et Antoine Nollet

4 February 2022

Table des matières

1 Anatomie de la méthode SIFT	3
1.1 Points-clés SIFT	3
1.1.1 Obtention des points clés d'une image	3
1.1.2 Propriétés d'un objet Keypoint	4
1.1.3 Points-clé de forte réponse	5
1.2 Paramètres de SIFT sous OpenCV	6
1.2.1 Le code	9
1.3 Comparaison Outil en ligne	10
2 Mise en correspondance d'images et applications	12
2.1 Critère de correspondance	12
2.1.1 Critère de Lowe	12
2.1.2 Stratégie Brute	13
2.1.3 Stratégie Lowe	14
2.1.4 Le code	14
2.2 Applications	15
2.2.1 Alignement d'images	15
2.2.2 Détection d'objets	16

1 Anatomie de la méthode SIFT

1.1 Points-clés SIFT

1.1.1 Obtention des points clés d'une image

Dans le cours nous avons vu un premier code pour nous permettre de détecter et représenter les points-clés d'une image (le code écrit ci-dessous). Le code ouvre l'image initialise un SIFT et l'applique sur l'image pour récupérer les points clés. Ces points-clés sont visuellement ajoutés à l'image pour ensuite afficher le tout.

```
1 def extractKeyPoints(fileName):
2     img = cv2.imread(fileName, cv2.IMREAD_GRAYSCALE)
3     # Constructor default params: nOctaveLayers = 3,
4     # contrastThreshold = 0.04, edgeThreshold = 10, sigma = 1.6
5     sift = cv2.SIFT_create() # construct
6     kp = sift.detect(img) # detect keypoints
7     img_with_kp = cv2.drawKeypoints(img, kp, img, None, cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
8
9     plt.imshow(img_with_kp)
10    plt.show()
11    plt.imsave("img_with_kp.png", img_with_kp)
```

Nous appliquons ce code sur l'image box.png.



FIGURE 1 – Image de base box.png

Nous obtenons ainsi le résultat suivant :



FIGURE 2 – Image de base box.png avec visualisation de ses points clés

Sur cette image obtenu nous constatons plusieurs cercles. Chacun de ces cercles correspond à un point clé de l'image, c'est à dire à un "coin". Chacun des cercles à un ou deux rayons de tracés : cela correspond à l'attribution de la ou les orientation par le SIFT (selon les pics des histogrammes des points candidats, si le pic secondaire correspond à 80% du pic maximal, alors on verra deux traits dans le cercle. De plus, on constate que les cercles ont des échelles différentes. Cela s'explique par le fait que les points candidats (qui deviendront points clés s'ils sont toujours présents après l'affinement des points candidats par le SIFT) sont détectés à des octaves différents de l'espace d'échelle. Si le point a été détecté à partir d'une petite échelle de l'image, alors le cercle sera d'autant plus grand lorsque nous remettions l'image à son échelle de base. Ainsi de même, si le point a été détecté à partir d'une échelle proche de celle de base de l'image, le cercle sera perçu comme petit.

Prenons un exemple :



FIGURE 3 – un point clé de box.png

Le cercle du point clé est assez grand car il a été récupéré après plusieurs octaves. Si maintenant on s'intéresse au point clé il s'agit du coin d'une lettre qui a donc été détecté tardivement. Pour le trait présent dans le cercle, correspondant à la direction, il est de 170°(console) et on peut facilement avoir une estimation visuelle (l'angle 0 sur un repère normalisé au centre du cercle serait (1,0)).

1.1.2 Propriétés d'un objet Keypoint

Voici les propriétés du Keypoint vu précédemment :

```
Class id : -1
Angle : 170.80426025390625
Octave : 14615042
Coordinate : (101.02662658691406, 135.0168914794922)
Reponse : 0.02134360745549202
Size : 22.15510368347168
```

FIGURE 4 – Détails du précédent Keypoint

Les propriétés utilisés pour la représentation graphique sont donc : l'angle en degré représenté par un flottant (qui permettra de dessiner le trait du cercle), les coordonnées du point représentées par un tuple de flottants (coordonnées en x et en y) et la taille du cercle représenté par un flottant.

1.1.3 Points-clé de forte réponse



FIGURE 5 – box.png avec les 10 points clés ayant la plus forte réponse



FIGURE 6 – box.png avec les 100 points clés ayant la plus forte réponse



FIGURE 7 – box.png avec les 500 points clés ayant la plus forte réponse

La propriété de réponse correspond à la pertinence du point clé, plus la réponse est forte, et plus il correspond à un "coin" de l'image. Dans l'espace de recherche, les points clés les plus pertinents sont ceux qui sont détectés à l'échelle la plus proche de celle de l'image de base (ce qui justifie la petite taille des cercles tracés). La propriété octave correspond à 2 octets décrivant le niveau d'octave de keypoint, sa couche et son échelle. Il est possible de lire ces octets grâce à la fonction suivante (source : stack overflow) :

```

1 def unpackSIFTOctave(kpt):
2     """unpackSIFTOctave(kpt)->(octave, layer, scale)
3     @created by Silencer at 2018.01.23 11:12:30 CST
4     @brief Unpack Sift Keypoint by Silencer
5     @param kpt: cv2.KeyPoint (of SIFT)
6     """
7     _octave = kpt.octave
8     octave = _octave&0xFF
9     layer = (_octave>>8)&0xFF
10    if octave>=128:
11        octave |= -128
12    if octave>=0:
13        scale = float(1/(1<<octave))
14    else:
15        scale = float(1<<-octave)
16    return (octave, layer, scale)

```

Lorsque nous trions les points clés par rapport à l'importance de leurs réponses, et qu'on applique ce code sur le point à la réponse la plus forte, nous obtenons le résultat suivant : (-1, 1, 2.0). Ce qui reste dans la logique précédemment citée, à l'octave -1, le point a été détecté sans "zoom" sur l'image, ce qui explique la petite taille de son cercle tracé.

1.2 Paramètres de SIFT sous OpenCV

le premier paramètre est "nfeatures" il correspond au nombre de meilleurs points clés à garder selon le classement mesuré par un algorithme de SIFT. Il est souvent à 0 car on veut trier les points clés nous-même suite au SIFT.

Le second paramètre est "nOctaveLayers" comme son nom l'indique il correspond au nombre de couche que l'on veut pour chaque octave. Il impactera les points clés détectés car ceux-ci seront potentiellement détectés à une plus basse octave qu'avant car plus le nombre de couche est grand dans une octave, plus on a de chance de détecter un point clé dans celle-ci.

Le troisième paramètre est "contrastThreshold", c'est à dire le seuil de contraste. Il sert à filtrer les points clés faibles dans les régions avec peu de contraste. Plus le seuil est grand, moins il y a de points clés produit par le SIFT (voir image ci dessous).

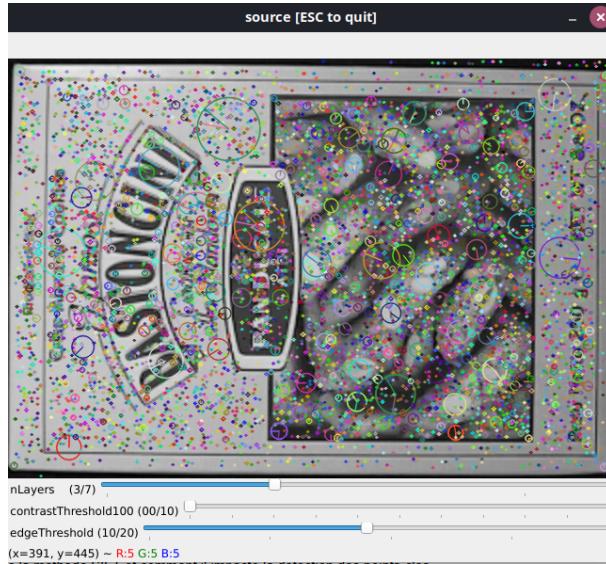


FIGURE 8 – Seuil de contraste à 0

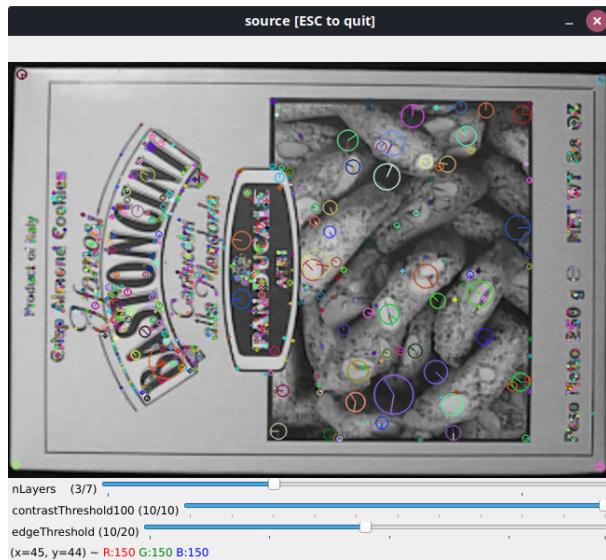


FIGURE 9 – Seuil de contraste à 0.4

Le quatrième paramètre est "edgeThreshold", c'est à dire le seuil de bord. Il sert à déterminer la limite de la valeur que peut prendre le ratio des vecteurs propres des courbures principales du développement en série de Taylor au point-clé. Contrairement au contrastThreshold, plus le edgeThreshold est grand, plus il y aura de points clés détectés (et plus il est petit, moins il y aura de points clés détectés). Par exemple, avec une valeur de 1 (voir ci-dessous), il n'y a plus de points clés, car chaque point clé a une valeur de ratio supérieure à 1, donc tout les points clés sont filtrés.

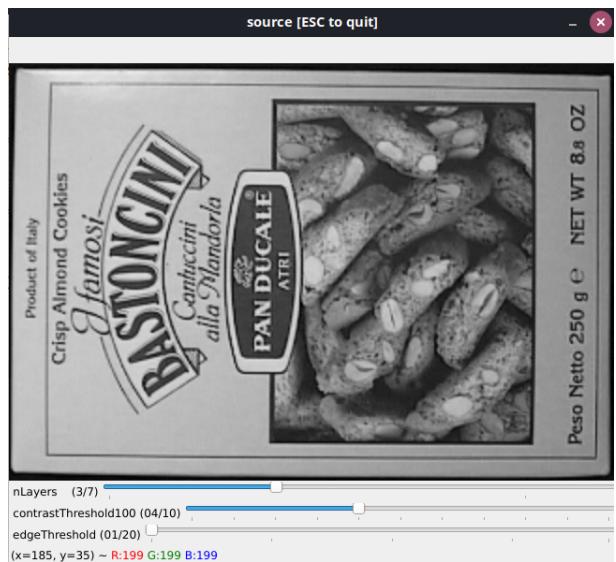


FIGURE 10 – Seuil de bord à 1

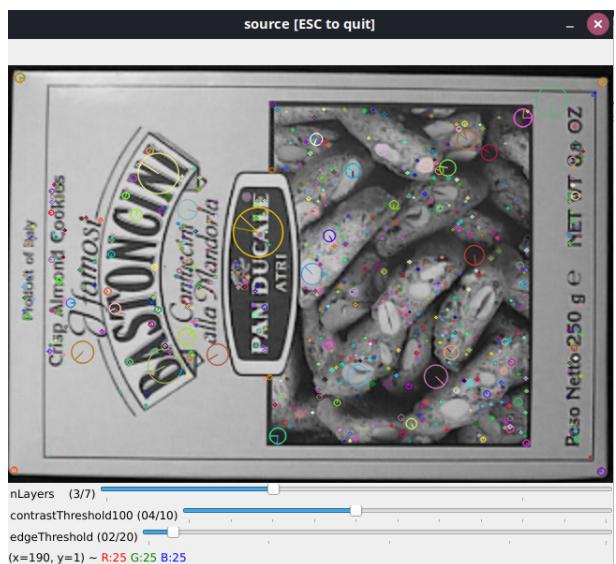


FIGURE 11 – Seuil de bord à 2

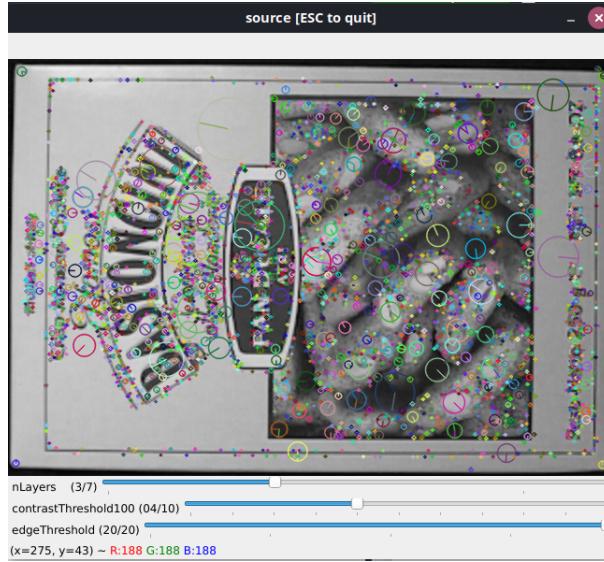


FIGURE 12 – Seuil de bord à 20

1.2.1 Le code

```

1 import cv2
2
3 def nothing(x):
4     pass
5
6 # Parameter default value
7 nLayers = 3
8 previousLayer = -nLayers
9 contrastThreshold100 = 4 #[0, 0.1]
10 previouscontrastThreshold100 = -contrastThreshold100
11 edgeThreshold = 10 #[1, 20].
12 previousEdgeThreshold = -edgeThreshold
13
14 # Create window and trackbar
15 win_name = 'source_[ESC_to_quit]'
16 cv2.namedWindow(win_name)
17 cv2.createTrackbar('nLayers', win_name, nLayers, 7, nothing)
18 cv2.setTrackbarMin('nLayers', win_name, 1)
19
20 cv2.createTrackbar('contrastThreshold100', win_name, contrastThreshold100, 10, nothing)
21 cv2.setTrackbarMin('contrastThreshold100', win_name, 0)
22
23 cv2.createTrackbar('edgeThreshold', win_name, edgeThreshold, 20, nothing)
24 cv2.setTrackbarMin('edgeThreshold', win_name, 1)
25
26 img = cv2.imread('box.png', cv2.IMREAD_GRAYSCALE)
27 img =cv2.resize(img,(img.shape[1]*2,img.shape[0]*2))
28
29 while True:
30     nLayers = cv2.getTrackbarPos('nLayers', win_name)
31     contrastThreshold100 = cv2.getTrackbarPos('contrastThreshold100', win_name)
32     edgeThreshold = cv2.getTrackbarPos('edgeThreshold', win_name)
33     if(previousLayer != nLayers or previouscontrastThreshold100 != contrastThreshold100 or
34     previousEdgeThreshold != edgeThreshold):
35         sift = cv2.SIFT_create(0, nLayers, contrastThreshold100/100, edgeThreshold)
36         keypoints = sift.detect(img)
37         img_with_kp = cv2.drawKeypoints(img, keypoints, None,
38             flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
39         cv2.imshow(win_name, img_with_kp)
40
41     #update previous

```

```

42     previousLayer = nLayers
43     previousContrastThreshold100 = contrastThreshold100
44     previousEdgeThreshold = edgeThreshold
45     if cv2.waitKey(1) == 27:
46         break
47     cv2.destroyAllWindows()

```

1.3 Comparaison Outil en ligne

Nous utilisons l'outil en ligne disponible depuis ce lien : <http://demo.ipol.im/demo/82/> sur ces deux images :



FIGURE 13 – Notre image de base box.png



FIGURE 14 – Insertion de notre image de base dans une scène

Et voici le résultat obtenu :

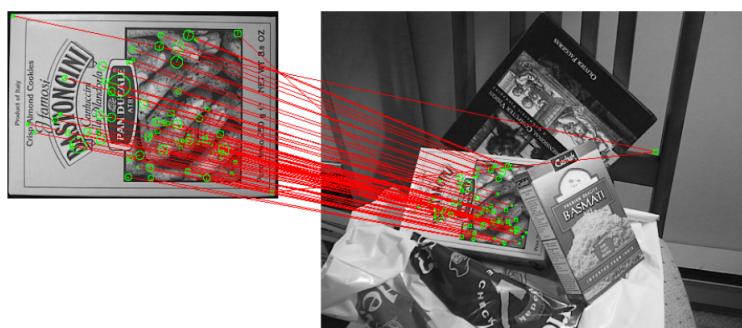


FIGURE 15 – Insertion de notre image de base dans une scène

Voici ce qu'il se passe lorsqu'on augmente la valeur du C_{DoG} (correspondant à notre précédent "contrastThreshold") :



FIGURE 16 – modification de C_{DoG} à 0.09

On constate que cela correspond à notre précédent observation, plus ce paramètre est élevé, moins il y a de points clés détectés. Et à l'inverse en la diminuant on obtient plus de points clés et de correspondances :

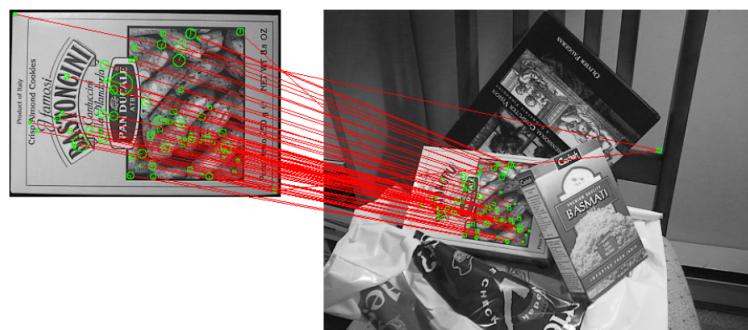


FIGURE 17 – modification de C_{DoG} à 0.01

Voici maintenant ce qu'il se passe lorsqu'on augmente la valeur du C_{edge} (correspondant à notre précédent "edgeThreshold") :

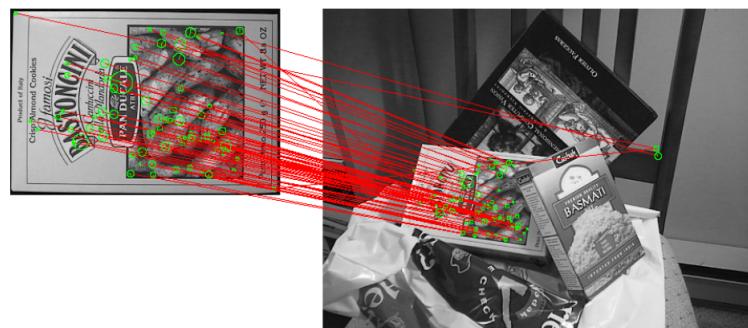


FIGURE 18 – modification de C_{edge} à 20

On constate que cela correspond à notre précédent observation, plus ce paramètre est élevé, plus il y a de points clés détectés. Et à l'inverse en la diminuant on obtient moins de points clés et de correspondances :

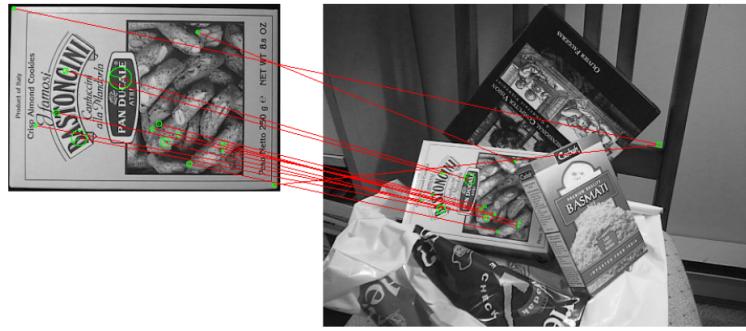


FIGURE 19 – modification de C_{edge} à 2

2 Mise en correspondance d’images et applications

2.1 Critère de correspondance

2.1.1 Critère de Lowe

Le principe du critère de Lowe appliqué à la correspondance de points-clés consistent à faire correspondre chaque descripteur sift de l’image de base à ses deux meilleures correspondances dans l’image de scène (où on retrouve apriori notre image de base). Ces deux correspondances doivent être assez éloignées et donc respecter la relation suivante :

$$\|d_1, d_{2a}\| < \|d_1, d_{2b}\| * C_{match}$$

Le paramètre C_{match} correspondra à celui là même de la démonstration en ligne précédemment citée, il est de base égal à 0.75.



FIGURE 20 – modification du critère de correspondance de Lowe à 0

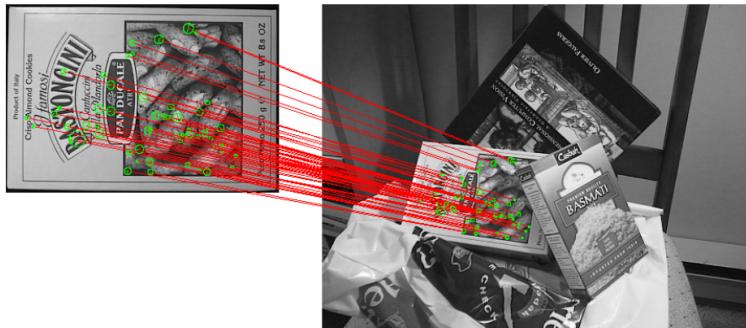


FIGURE 21 – modification du critère de correspondance de Lowe à 0.5

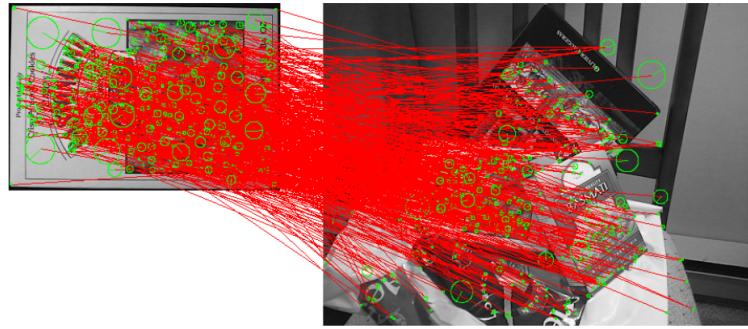


FIGURE 22 – modification du critère de correspondance de Lowe à 1

On constate que le paramètre C_{match} égal à 0.5 fournit les meilleurs résultats. Cela est logique, s'il est égal à 0, aucun descripteur n'aura de correspondance inférieur à 0 et donc nous aurons moins de correspondances que désiré. De plus, s'il est égal à 1, les correspondances ne seront pas "filtrées" par le critère de Lowe et nous aurons bien trop de correspondances que souhaité.

2.1.2 Stratégie Brute

Dans la fonction cv2.drawMatches les paramètres matchColor et singlePointColor ont comme valeur par défaut "Scalar : all(-1)" ce qui signifie que les couleurs des points et des matchs sont de couleurs aléatoires. Le paramètre matchesMask est vide par défaut ce qui signifie que tous les matchs sont dessinés. On peut faire le pari avec le critère de Lowe avec un C_{match} égal à 1, qui revient à ne pas filtrer les matchs, et donc nous obtenons la même image qu'ici avec seulement les couleurs qui changent.

Dans cette stratégie brute, tous les matchs sont apparents :

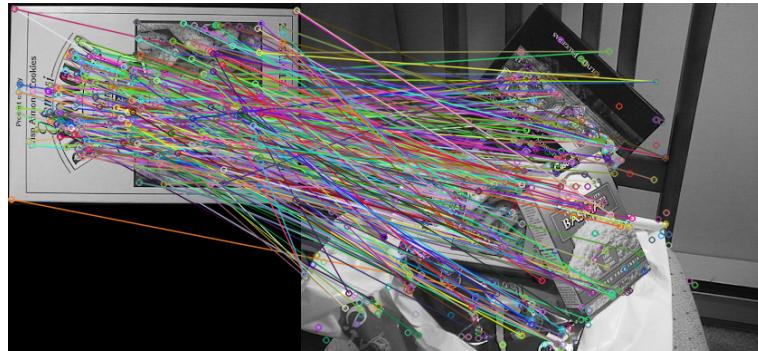


FIGURE 23 – Résultat de la stratégie brute

2.1.3 Stratégie Lowe

En utilisant un ratio de correspondance (le C_{match}) de 0.8, voici les 10 meilleures correspondances obtenues :

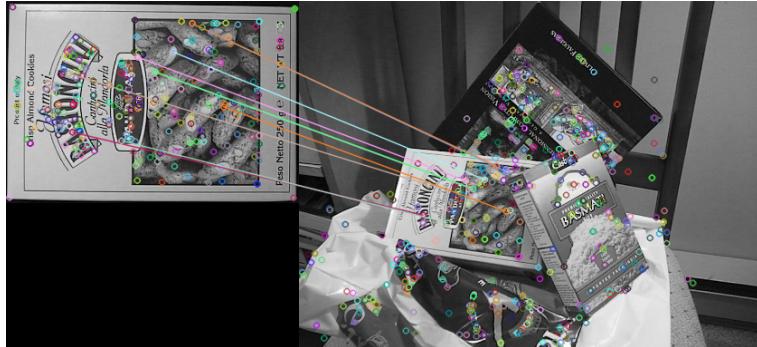


FIGURE 24 – Résultat de la stratégie brute

Les 10 meilleures correspondances sont ainsi correctes, on peut en conclure que nous pouvons utiliser sans crainte la valeur de 0.8 comme ratio de correspondance.

Les dix meilleurs ont été obtenue en triant la liste des bonnes correspondances par critère de distance :

```
1 | good = sorted(good, key=lambda x: x.distance)[:11]
```

2.1.4 Le code

Ci dessous le code de cette partie :

```

1 import cv2
2 import matplotlib.pyplot as plt
3
4 def matchImg1Img2(fileName1, fileName2, strategy="lowe", loweFilter=0.75):
5     img1 = cv2.imread(fileName1, cv2.IMREAD_GRAYSCALE)
6     img2 = cv2.imread(fileName2, cv2.IMREAD_GRAYSCALE)
7
8     sift = cv2.SIFT_create()
9
10    sift = cv2.SIFT_create(contrastThreshold=0.1) # construct
11    kp1, des1 = sift.detectAndCompute(img1, None) # detect keypoints
12    kp2, des2 = sift.detectAndCompute(img2, None) # detect keypoints
13
14    bf = cv2.BFMatcher()
15    matches1to2 = bf.knnMatch(des1, des2, k=2)
16
17    good = []
18    for m, n in matches1to2:
19        if strategy=="force_brute":
20            good.append(m)
21        elif m.distance < loweFilter*n.distance:
22            good.append(m)
23
24    img3 = cv2.drawMatches(img1, kp1, img2, kp2, good, None)
25
26    plt.imshow(img3), plt.show()
27    plt.imsave("imgMatch.png", img3)
28
29    return img3
30
31 matches1to2 = matchImg1Img2('box.png', 'box_in_scene.png', 'lowe', 0.8)

```

2.2 Applications

2.2.1 Alignement d'images

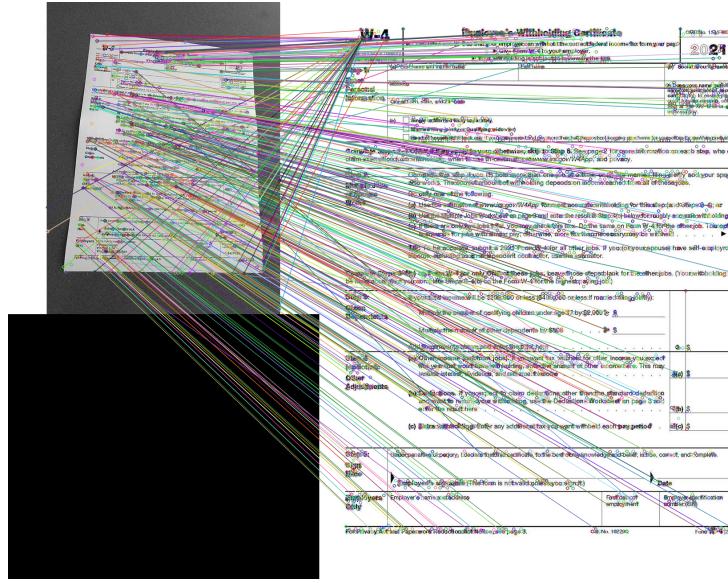


FIGURE 25 – matchs des deux images

Form W-4 Employee's Withholding Certificate <small>Complete Form W-4 so that your employer can withhold the correct federal income tax from your pay. Give Form W-4 to your employer.</small> 2021	<small>OMB No. 1545-0074</small> <small>Department of the Treasury Internal Revenue Service</small> Step 1: (a) First name and middle initial Last name (b) Social security number Enter Personal Information Address <small>City or town, state, and ZIP code</small> <input type="checkbox"/> Single or Married filing separately <input type="checkbox"/> Married filing jointly (Qualifying widow(er)) <input type="checkbox"/> Head of household (Check only if you're separated and pay more than half the costs of keeping up a home for yourself and a qualifying individual)	<small>► Complete Form W-4 so that your employer can withhold the correct federal income tax from your pay. ► Give Form W-4 to your employer. ► Your withholding is subject to review by the IRS.</small> 2020 Form W-4 Employee's Withholding Certificate <small>Complete Form W-4 so that your employer can withhold the correct federal income tax from your pay. Give Form W-4 to your employer.</small> 2020 <small>OMB No. 1545-0074</small> <small>Department of the Treasury Internal Revenue Service</small> Step 1: (a) First name and middle initial Last name (b) Social security number Enter Personal Information Address <small>PO Box 1798 F-17900 Cincinnati, OH 45244-1790</small> <small>► Complete Form W-4 so that your employer can withhold the correct federal income tax from your pay. ► Give Form W-4 to your employer. ► Your withholding is subject to review by the IRS.</small> Step 2: Complete this step if you (1) hold more than one job at a time, or (2) are married filing jointly and your spouse also holds more than one job. The correct amount of withholding depends on income earned from all of these jobs. Do only one of the following: (a) Use the estimator at www.irs.gov/W4App for most accurate withholding for this step (and Steps 3-4); or (b) Use the Multiple Jobs Worksheet on page 3 and enter the result in Step 4(c) below for roughly accurate withholding; or (c) If there are only two jobs total, you may check this box. Do the same on Form W-4 for the other job. This option is accurate for jobs with similar pay; otherwise, more tax than necessary may be withheld. □ <small>TIP: To be accurate, submit a 2021 Form W-4 for all other jobs. If you (or your spouse) have self-employment income, including as an independent contractor, use the estimator.</small> Step 3: Complete Steps 3-4(b) on Form W-4 for ONLY ONE of these jobs. Leave those steps blank for the other jobs. (Your withholding will be most accurate if you complete Steps 3-4(b) on the Form W-4 for the highest paying job.) Step 4: Complete this step if you (1) hold more than one job at a time, or (2) are married filing jointly and your spouse also holds more than one job. The correct amount of withholding depends on income earned from all of these jobs. Do only one of the following: (a) Use the estimator at www.irs.gov/W4App for most accurate withholding for this step (and Steps 3-4); or (b) Use the Multiple Jobs Worksheet on page 3 and enter the result in Step 4(c) below for roughly accurate withholding; or (c) If there are only two jobs total, you may check this box. Do the same on Form W-4 for the other job. This option is accurate for jobs with similar pay; otherwise, more tax than necessary may be withheld. □ <small>TIP: To be accurate, submit a 2020 Form W-4 for all other jobs. If you (or your spouse) have self-employment income, including as an independent contractor, use the estimator.</small> Step 5: Complete Steps 3-4(b) on Form W-4 for ONLY ONE of these jobs. Leave those steps blank for the other jobs. (Your withholding will be most accurate if you complete Steps 3-4(b) on the Form W-4 for the highest paying job.) Step 6: Complete this step if you (1) hold more than one job at a time, or (2) are married filing jointly and your spouse also holds more than one job. The correct amount of withholding depends on income earned from all of these jobs. Do only one of the following: (a) Use the estimator at www.irs.gov/W4App for most accurate withholding for this step (and Steps 3-4); or (b) Use the Multiple Jobs Worksheet on page 3 and enter the result in Step 4(c) below for roughly accurate withholding; or (c) If there are only two jobs total, you may check this box. Do the same on Form W-4 for the other job. This option is accurate for jobs with similar pay; otherwise, more tax than necessary may be withheld. □ <small>TIP: To be accurate, submit a 2020 Form W-4 for all other jobs. If you (or your spouse) have self-employment income, including as an independent contractor, use the estimator.</small> Step 7: Complete this step if you (1) hold more than one job at a time, or (2) are married filing jointly and your spouse also holds more than one job. The correct amount of withholding depends on income earned from all of these jobs. Do only one of the following: (a) Use the estimator at www.irs.gov/W4App for most accurate withholding for this step (and Steps 3-4); or (b) Use the Multiple Jobs Worksheet on page 3 and enter the result in Step 4(c) below for roughly accurate withholding; or (c) If there are only two jobs total, you may check this box. Do the same on Form W-4 for the other job. This option is accurate for jobs with similar pay; otherwise, more tax than necessary may be withheld. □ <small>TIP: To be accurate, submit a 2020 Form W-4 for all other jobs. If you (or your spouse) have self-employment income, including as an independent contractor, use the estimator.</small> Step 8: Sign Here <small>Under penalties of perjury, I declare that this certificate, to the best of my knowledge and belief, is true, correct, and complete.</small> Employers Only Employer's name and address First date of employment Employer identification number (EIN) <small>For Privacy Act and Paperwork Reduction Act Notice, see page 3.</small> <small>Cat. No. 10229G Form W-4 (2021)</small> <small>For Privacy Act and Paperwork Reduction Act Notice, see page 3.</small> <small>Cat. No. 10229G Form W-4 (2020)</small>
---	---	--

FIGURE 26

En comparant ces deux images, on remarque que celle extraite par homographie est légèrement déformée et qu'elle contient des bordures non désirées. Cependant, l'image extraite par homographie reste lisible et fidèle à l'originale.

Le code suivant a été rajouté à la fonction pour obtenir ce résultatat ci-dessus (il est inspiré de l'exemple du tuto disponible à cette url).

```

1 src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
2 dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
3
4 H, mask = cv2.findHomography(dst_pts, src_pts, method=cv2.RANSAC)
5 h, w = img1.shape[:2]
6
7 aligne = cv2.warpPerspective(img2, H, (w, h))
8
9 plt.imshow(aligne, cmap='gray'), plt.show()
10 plt.imsave("aligne.png", aligne, cmap='gray')

```

Il a été question, dans un premier temps, d'estimer l'homographie, grâce à la méthode de RANSAC, de l'image de scène à l'image de base (par leurs points-clés). Dans un second temps, l'homographie a été appliquée à l'image de scène afin de retrouver un alignement sur la correspondance avec l'image de base.

2.2.2 Détection d'objets

```

1 def matchImg1Img2(fileName1, fileName2, strategy="lowe", loweFilter=0.75):
2     img1 = cv2.imread(fileName1, cv2.IMREAD_COLOR)
3     img2 = cv2.imread(fileName2, cv2.IMREAD_COLOR)
4
5     sift = cv2.SIFT_create()
6
7     sift = cv2.SIFT_create(contrastThreshold=0.1) # construct
8     kp1, des1 = sift.detectAndCompute(img1, None) # detect keypoints
9     kp2, des2 = sift.detectAndCompute(img2, None) # detect keypoints
10
11    bf = cv2.BFMatcher()
12    matches1to2 = bf.knnMatch(des1, des2, k=2)
13
14    good = []
15    for m, n in matches1to2:
16        if strategy=="force_brute":
17            good.append(m)
18        elif m.distance < loweFilter*n.distance:
19            good.append(m)
20
21    img3 = cv2.drawMatches(img1, kp1, img2, kp2, good, None)
22
23    cv2.imshow("window_name", img3)
24    cv2.waitKey(0)
25    cv2.imwrite("imgMatch.png", img3)
26
27    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
28    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
29
30    #H, mask = cv2.findHomography(dst_pts, src_pts, cv2.RANSAC)
31    H, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC)
32    h, w = img1.shape[:2]
33
34    matchesMask = mask.ravel().tolist()
35
36    pts = np.float32([[0,0],[0,h-1],[w-1,h-1],[w-1,0]]).reshape(-1,1,2)
37    dst = cv2.perspectiveTransform(pts,H)
38
39    img2 = cv2.polylines(img2,[np.int32(dst)],True,(255,0,0),3, cv2.LINE_AA)
40    cv2.imshow("window_name", img2)
41    cv2.waitKey(0)
42    cv2.imwrite("encadrer.png", img2)
43
44    return img3

```



FIGURE 27 – matchs des deux images

On peut visuellement conclure que, malgré un encadrement qui est légèrement en biais, l'estimation par homographie est en effet robuste.

Essayons cette même méthode sur diverses images et constatons les détections :



FIGURE 28 – exemple 1 de détection - changement de perspective

(0.85)



FIGURE 29 – exemple 2 de détection - changement d'échelle



FIGURE 30 – exemple 3 de détection - changement d'illumination



FIGURE 31 – exemple 4 de détection - changement de météo (luminance bis)

On constate à travers les exemples vues ci-dessus que :

- Le changement de perspective n'est pas significativement impactant
- La détection, lorsque l'échelle devient très petite, devient inefficace (malgré à encadrement correct en axe y)
- Le changement de luminance, lorsqu'il est moindre, n'affecte que légèrement la détection. (et affecte beaucoup la détection quand la luminance change beaucoup)
- En général, les qualités des images tests ont beaucoup impactées les détections