



**POLYTECHNIQUE
MONTRÉAL**

UNIVERSITÉ
D'INGÉnierie

PHS2223 – INTRODUCTION À L’OPTIQUE MODERNE

Équipe : 04

Expérience 2

Objectif de caméra

Présenté à
Guillaume Sheehy
Esmat Zamani

Par :
Émile Guertin-Picard (2208363)
Laura-Li Gilbert (2204234)
Tom Dessauvages (2133573)

21 octobre 2024
Département de Génie Physique
Polytechnique Montréal

Table des matières

1 Résultats	1
1.1 Profondeur de champ	1
1.2 Analyse de résolution	3
1.3 Vignettage	5
1.4 Facteur de zoom	7
1.5 Tableau des caractéristiques	8
2 Discussion	8
2.1 Retour sur l'hypothèse	8
2.2 Sources d'erreurs	9
2.3 Réponses aux questions	9
3 Conclusion	10
4 Annexe	11
4.1 Codes contours zoom minimal	11
4.2 Codes contours zoom maximal	15

1 Résultats

Cette section présente et analyse les différentes images capturées lors des manipulations.

1.1 Profondeur de champ

Tout d'abord, les images qui suivent permettent d'analyser la profondeur de champ du système de lentilles dans deux configurations, soit celle offrant un zoom minimal, et celle offrant un zoom maximal. Pour obtenir ces images, à chaque extrême de zoom, le focus est mis sur une cible graduée, à une distance de 1m. Ensuite, cette cible est déplacée d'abord devant, puis derrière la distance focale, jusqu'à perte du focus. Cela est déterminé approximativement par la perte de définition entre les graduations présentes sur la cible. La distance de déplacement jusqu'à cette perte est notée afin de faire le calcul de profondeur de champ. Pour le zoom minimal :

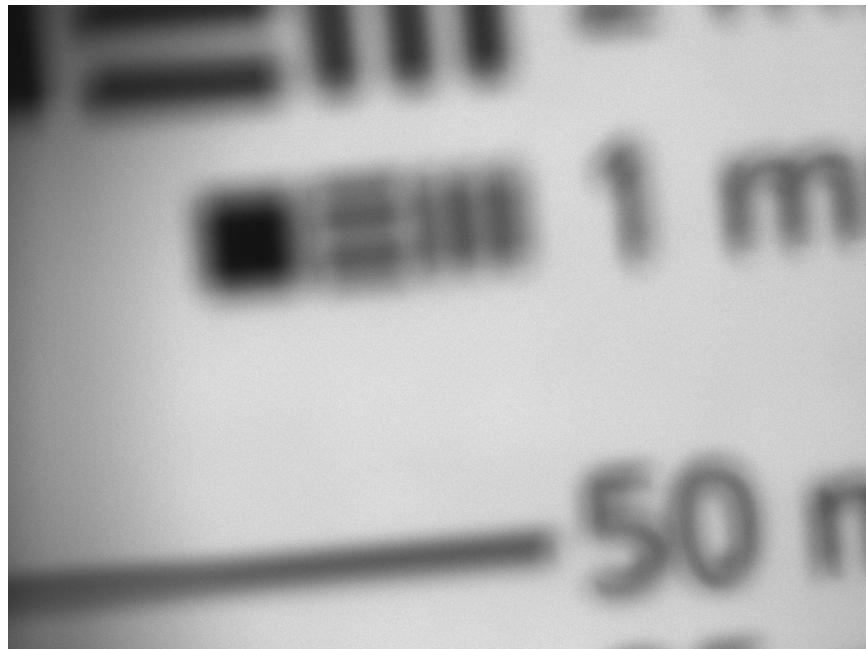


Figure 1 : Photo du perte de focus en amont du point focal de 1m, soit à 0.458m, pour le zoom minimal.

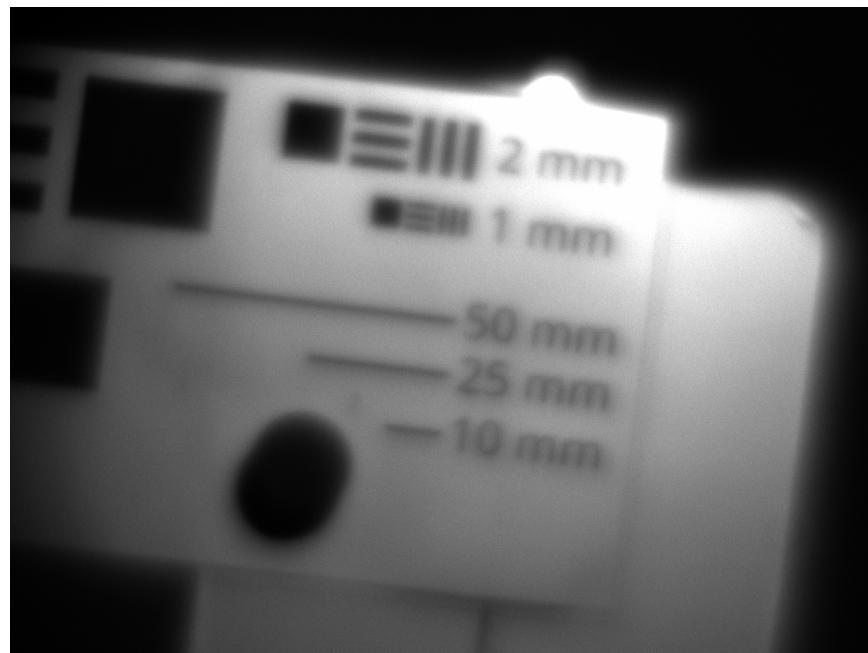


Figure 2 : Photo du perte de focus en aval du point focal de 1m, soit à 1.986m, pour le zoom minimal.

La différence entre les deux positions de perte de focus est donc la profondeur de champ. Il est donc possible de trouver $\delta_{z,min} = 1.528\text{m}$. Pour le zoom maximal :

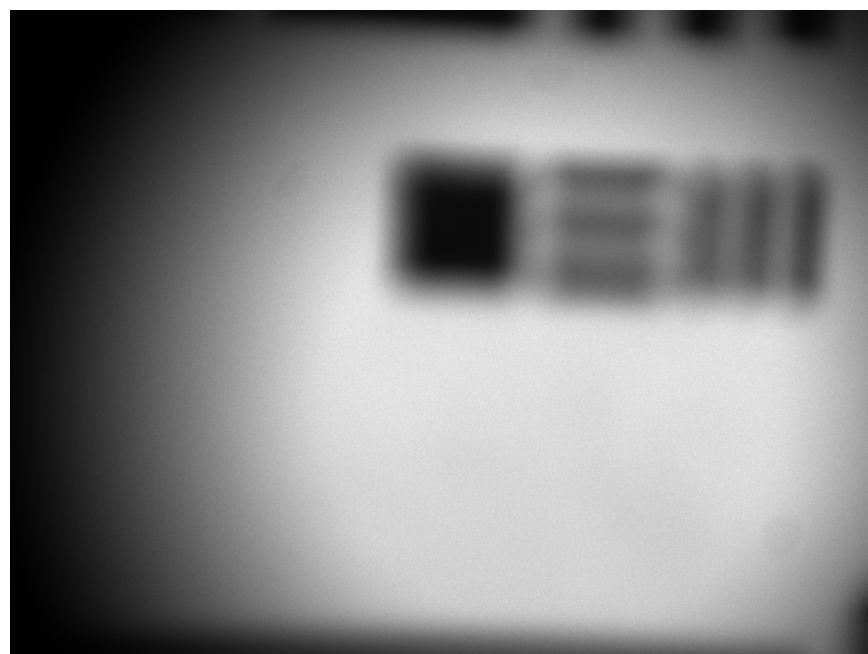


Figure 3 : Photo du perte de focus en amont du point focal de 1m, soit à 0.711m, pour le zoom maximal.

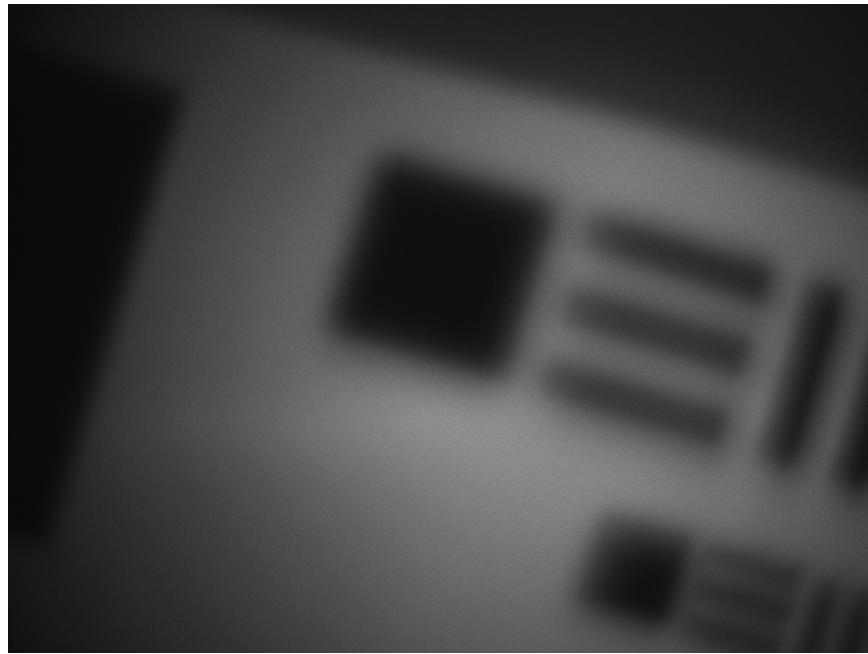


Figure 4 : Photo du perte de focus en aval du point focal de 1m, soit à 1.174m, pour le zoom maximal.

De la même manière, il est possible de trouver la profondeur de champ $\delta_{z,max} = 0.463\text{m}$.

1.2 Analyse de résolution

Afin d'analyser la résolution de la caméra, deux photos de la cible ont été prises au focus de la caméra à 1m, autant pour le zoom minimal que pour le zoom maximal :



Figure 5 : Photo de l'objet au focus de 1m pour le zoom minimal.

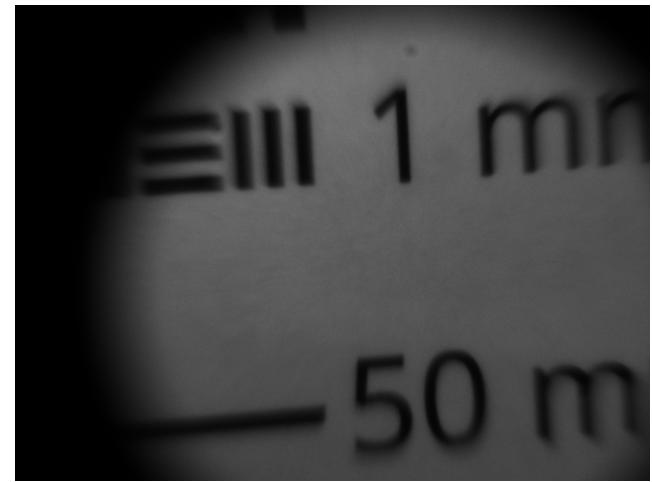


Figure 6 : Photo de l'objet au focus de 1m pour le zoom maximal.

La résolution de ces images peut être définie de façon générale comme étant :

$$r = \frac{\text{taille de référence}}{\text{nombre de pixels}} [L \cdot \text{pixels}^{-1}] \quad (1)$$

La taille de référence correspond à une dimension connue de l'espace réel. Sur les figures 5 et 6 elle est représentée par les barres de dimensions fixe : 1 mm, 2 mm, 10 mm, 25 mm et 50 mm. Pour se ramener à l'équation 1 il est donc nécessaire de comptabiliser le nombre de pixels occupé par chacune de ces tiges, ce qui est possible grâce à un programme python, disponible en annexe. Les figures 7 et 8 présente les contours détectés pour chacune des images présentées en figure 5 et 6

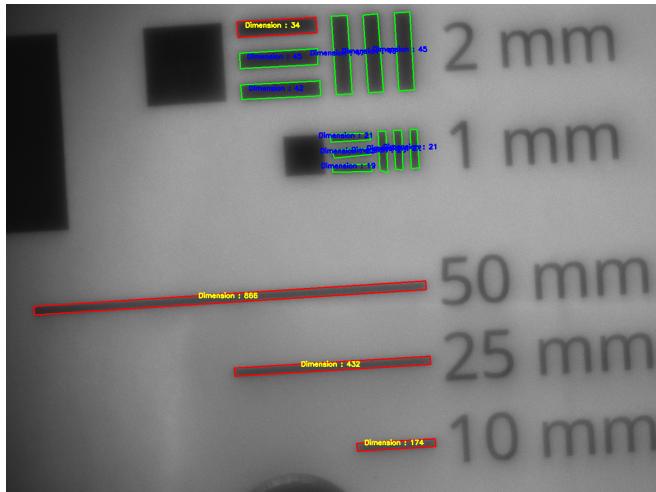


Figure 7 : Contours obtenus sur les tiges de références le zoom minimal.

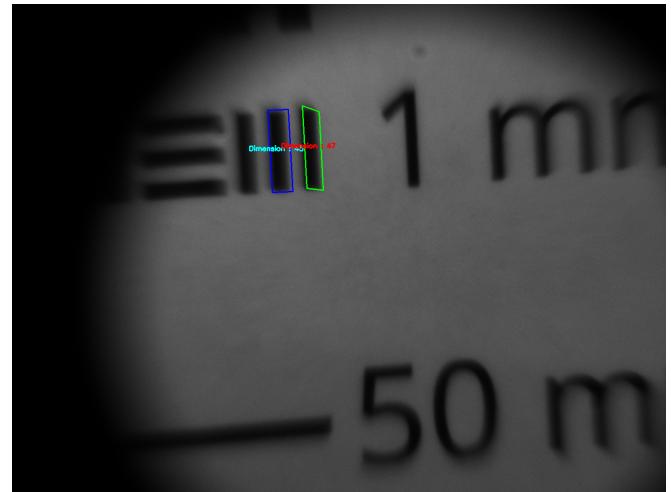


Figure 8 : Contours obtenus sur les tiges de références le zoom maximal.

Les données "dimensions" représentées en pixels ont été répertoriées dans des tableaux

Taille réelle (mm)	Nombre d'échantillons	Moyenne (pixels)	Écart-type (pixels)
1	6	21.500000	1.516575
2	6	43.222657	4.664085
10	1	174.214767	0.000000
25	1	432.901978	0.000000
50	1	866.068909	0.000000

Table 1 – Tableau du nombre de pixels en fonction des tiges de référence pour le zoom minimum.

Taille réelle (mm)	Nombre d'échantillons	Moyenne (pixels)	Écart-type (pixels)
1	2	46.048563	1.345535

Table 2 – Tableau du nombre de pixels en fonction des tiges de référence pour le zoom maximum.
Ces écarts peuvent être causés par de nombreux facteurs, notamment la qualité des images enregistrées qui impacte grandement la détection des contours via python. Ces approximations sont extrêmement explicites pour le zoom maximal présenté en figure 2 car les tiges de référence sont dès le départ floues.

Les codes python utilisés et présentés en annexe prennent en compte ce manque de contraste au maximum, mais ne permettent pas de l'annuler.

Dans ces tableaux les données ont été traitées de façons statistiques en fonction du nombre d'échantillons et de leur valeur. En les mettant ensuite en lien avec l'équation 1 il est possible d'en déduire des valeurs de résolution. Le tableau ?? présente les résolutions obtenues en fonction des positions de zoom.

Les statistiques présentées dans ce tableaux n'ont pas été mise directement en relation avec celle des

Zoom	Résolution (mm/pixel)	Écart-type (résolution)
Min	0.053133	0.006156
Max	0.021716	0.000000

Table 3 – Tableau de la résolution en fonction de la position du zoom.

tableaux 1 et 2, elles ne représentent que celles obtenues grâce au calcul des résolutions pour les différentes tiges de référence. Ces résultats sont très proches de ceux obtenus de façon théorique dans le rapport préliminaire malgré un facteur d'échelle non explicité sur les graphiques du rapport préliminaire. Le tableau 4 présente les écarts entre les valeurs théoriques et expérimentales.

Résolution théorique	Résolution expérimentale	Écart (absolu)
0.050	0.053133	0.003
0.023 (A VERIFIER)	0.021716	0.001

Table 4 – Tableau des écarts entre les valeurs théoriques et expérimentales.

1.3 Vignettage

L'analyse du vignettage se fait qualitativement en comparant sa présence aux deux niveaux de zoom, mais aussi à deux positions de l'objet, soit 1m et "l'infini". Un vignettage fort est défini par un contour net entre une région circulaire de l'image et la région "extérieure" au cercle comme noir. Moins ce contour est net, moins le vignettage est fort. Pour l'objet à 1m, en commençant par le zoom minimal :



Figure 9 : Image avec vignettage pour un objet à 1m vu avec un zoom minimal.

Ce vignettage peut être considéré comme modéré en regardant la région assombrie en bas à gauche. Ensuite, avec le zoom maximal :



Figure 10 : Image avec vignettage pour un objet à 1m vu avec un zoom maximal.

Le vignettage est beaucoup plus important avec ce niveau de zoom, tel que montré par la région totalement sombre à gauche. Pour l'objet à l'infini avec le zoom minimal :

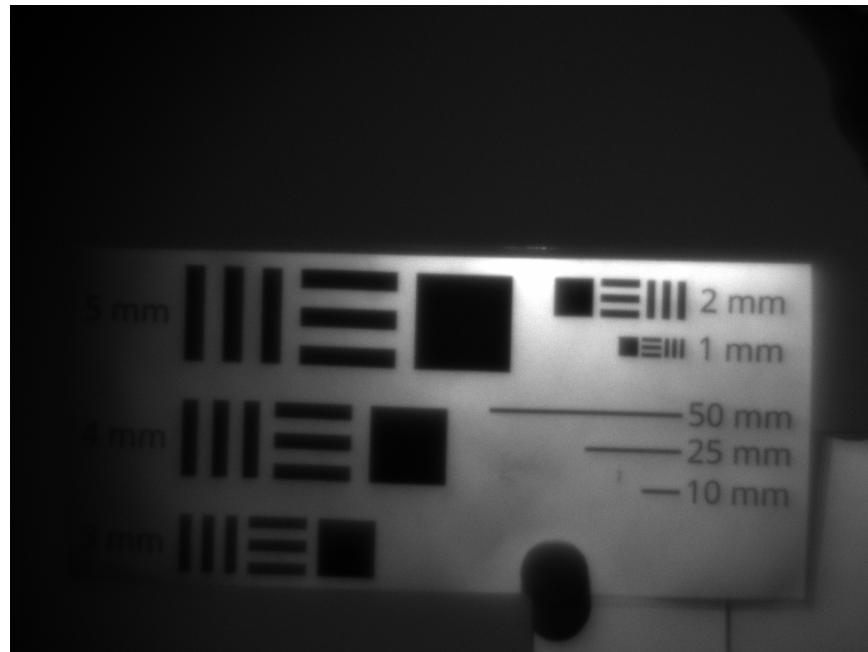


Figure 11 : Image avec vignettage pour un objet à l'infini vu avec un zoom minimal.

Ce vignettage est faible, mais perceptible à chaque coin inférieur de l'image. Avec le zoom maximal :

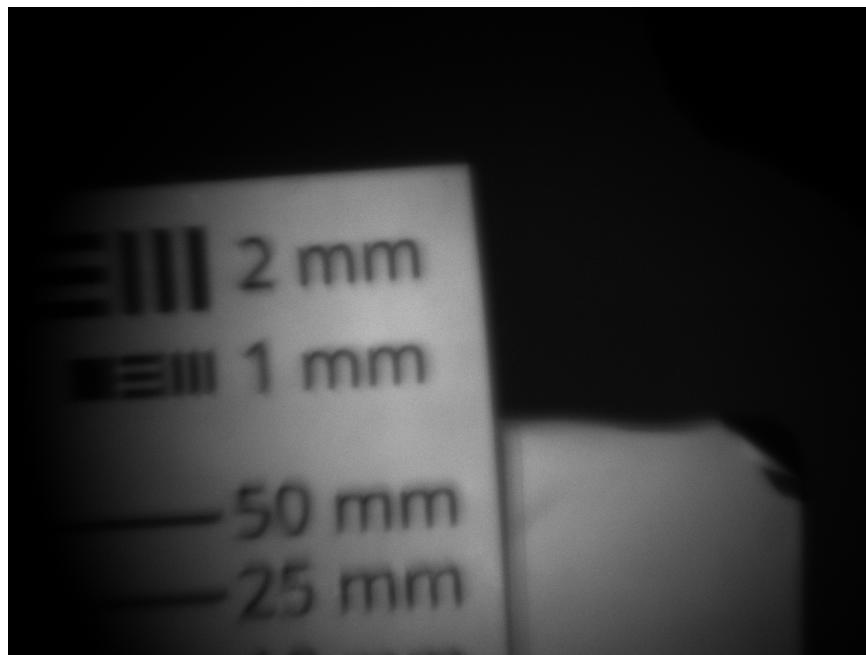


Figure 12 : Image avec vignettage pour un objet à l'infini vu avec un zoom maximal.

Enfin, ce vignettage est aussi modéré, voir fort. On voit une grande région assombrie à gauche, mais moins définie qu'à la figure 10.

1.4 Facteur de zoom

Le facteur de zoom est un rapport entre le grossissement maximal et minimal du système. Le grossissement pouvant être mis en lien avec la résolution comme :

$$g = \frac{\text{taille pixel}}{r}$$

Or la résolution étant elle même un rapport entre la taille d'un objet dans l'espace réel et pixel le facteur de grossissement est défini comme :

$$X = \frac{g_{max}}{g_{min}} = \left(\frac{\text{nombre de pixels max}}{\text{nombre de pixels min}} \right)_{ref}$$

Etant donné le jeu de données présenté dans les tableaux 1 et 2 le système a pour facteur de grossissement :

$$X \approx \frac{46.05}{21.50} \pm \Delta X \approx 2.14 \pm \Delta X$$

Où l'écart-type ΔG est exprimé comme :

$$\Delta X = 2.14 \cdot \sqrt{\left(\frac{1.345535}{46.048563} \right)^2 + \left(\frac{1.516575}{21.500000} \right)^2} \approx 0.16$$

Ce qui donne comme résultat final :

$$X \approx 2.14 \pm 0.16$$

1.5 Tableau des caractéristiques

Distance objet	Zoom	Profondeur champ	Résolution	Vignettage	Facteur zoom
1m	min	1.528m	[valeur]	modéré	g
1m	max	0.463m	[valeur]	très fort	
Infini	min	-	[valeur]	faible	
Infini	max	-	[valeur]	fort	

Table 5 – Tableau des principaux résultats acquis lors des manipulations.

2 Discussion

Cette section est dédiée à l'analyse, l'interprétation et la critique des résultats obtenus lors des manipulations.

2.1 Retour sur l'hypothèse

Les hypothèses émises précédemment comparaient certaines caractéristiques de performance de la caméra à la distance entre les lentilles L_2 et L_1 . Cela revient essentiellement à analyser les caractéristiques en fonction du zoom. En guise de rappel, les hypothèses suivantes ont été émises :

- La profondeur de champ reste constante à environ 6.6mm peu importe le zoom.
- Plus l'espace entre les lentilles est élevé, plus le zoom est important.
- Plus l'espace entre les lentilles est élevé, moins la résolution est importante.

D'abord, il est très facile de voir que la première hypothèse sur la profondeur de champ est fausse. Non seulement le comportement n'était pas constant avec le zoom, mais l'ordre de grandeur est en réalité plus près du mètre que du millimètre. Cette disparité s'explique facilement par l'omission de l'iris ajustable dans la simulation faite sur python. Cette dernière peut jouer le rôle d'ouverture d'arrêt, et donc avoir un impact direct sur la profondeur de champ [Source A](#). Sans l'avoir prise en considération, il est normal que l'hypothèse soit erronée.

Ensuite, il est possible de confirmer la deuxième hypothèse sur le zoom. La configuration avec le zoom maximal utilisé lors de la prise de photo était celle avec la plus grande séparation entre les lentilles. Le comportement précis de la courbe trouvée par simulation ne peut être confirmé toutefois, car seulement les deux extrêmes de zoom ont été testés. Cela implique donc qu'une augmentation de la distance $L_1 - L_2$ implique une augmentation du zoom.

Cette dernière conclusion aide à redéfinir la troisième hypothèse : plus le zoom est grand, moins la résolution est importante. [A FINIR : Emile.](#)

2.2 Sources d'erreurs

Les divergences des résultats par rapport aux hypothèses émises peuvent être expliquées par certaines causes d'erreurs. Une première cause possible correspond à l'éclairage inconstant de la pièce. Ces fluctuations, causées par l'ouverture et la fermeture de lumières, ont influencé les conditions d'illumination des objets photographiés, notamment en affectant l'exposition, le contraste et la netteté de l'image. Ensuite, lorsque la profondeur de champ a été observée, certaines des photos prises impliquaient de se tenir entre deux tables. Pour ce faire, une personne devait maintenir l'objet à la hauteur de la caméra, causant du mouvement dans la cible. Ce manque de stabilité dans ces prises a entraîné un décalage dans certaines des images. En effet, entre l'enregistrement de l'image et l'ajustement du paramètre observé, il est possible que le manque de stabilité ait provoqué des flous, altérant la qualité de l'image.

De plus, l'ouverture de l'iris, correspondant à l'ouverture d'arrêt dans ce système optique, a été changée au cours de l'expérimentation, provoquant une variation dans l'ouverture influence le résultat obtenu. Par exemple, dans le cas de la profondeur de champ et de la résolution, une grande ouverture réduit la profondeur de champ alors qu'elle améliore la résolution et inversement pour une petite ouverture. De ce fait, pour un même point, le profondeur de champ et la résolution varient en fonction de l'iris.

Finalement, une autre cause d'erreur possible est le délai dans l'acquisition des images. Puisque les photos ne s'enregistraient pas exactement au moment choisi, le délai entre l'enregistrement et la capture choisie peut avoir varié dû aux conditions externes, soient, par exemple, l'éclairage et le mouvement de la cible.

2.3 Réponses aux questions

Dans cette expérience, l'iris joue le rôle d'ouverture d'arrêt dans le système optique, c'est-à-dire de réguler la luminosité passant à travers les lentilles. De cette manière, cette régulation permet à l'ouverture d'effectuer trois fonctions principales : le contrôle de l'exposition, de la profondeur de champ et des aberrations. De cette manière, dans un objectif commercial, l'iris est souvent utilisé pour ajuster l'exposition et, par le fait même, d'améliorer la profondeur de champ de manière précise. En effet, en limitant ou augmentant la lumière entrante, il est possible d'ajuster l'exposition de l'image. Cette variation d'exposition permet de contrôler la zone de netteté de l'objet, soit la profondeur de champ, et les rayons lumineux hors focaux, de sorte que la qualité de l'image soit améliorée en fonction de la taille ([Source 1](#)). Bien que l'iris octroie plusieurs bénéfices, celui-ci n'est pas toujours nécessaire dans les systèmes optiques. Dans cette expérience, l'iris correspondait à l'ouverture d'arrêt, ce qui n'est pas toujours le cas dans tous les systèmes optiques. Ainsi, dans ces systèmes où l'ouverture d'arrêt correspond à un autre composant, la lumière est régulée avec ou sans l'utilisation de l'iris.

Les systèmes optiques, tel que les objectifs de caméra, peuvent entraîner des aberrations malgré l'utilisation d'une ouverture d'arrêt. Parmi les aberrations possibles, il n'existe pas directement d'aberrations plus fréquentes que d'autres. Cependant, deux aberrations communes correspondent aux aberrations sphériques et chromatiques transversales. Les aberrations sphériques surviennent lorsque les rayons se focalisent à des distances différentes dû à leur interaction avec les lentilles. En d'autres termes, en fonction du point de contact avec les lentilles ou les miroirs, les faisceaux ne se dirigent pas tout à fait au même point, diminuant la qualité de l'image ([Source 2](#)). Plusieurs méthodes peuvent être utilisées pour corriger cette aberration. Parmi celles-ci, l'utilisation de lentilles anti-sphériques est possible. Ces lentilles limitent les courbes sphériques, telles qu'illustrées dans la figure suivante :

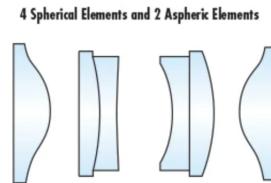


Figure 13 : Image de lentilles de forme anti-sphériques ([Source 3](#)).

Une autre méthode pouvant diminuer les aberrations sphériques correspond à utiliser un système optique symétrique. Cette méthode consiste à placer deux lentilles de sorte que les aberrations créées par la première lentille soient corrigées par la deuxième lentille, souvent placée afin que les deux lentilles se miroitent.

En ce qui concerne la seconde aberration mentionnée, soit celle chromatique, celle-ci surviennent lorsque de la réfraction des rayons sur les lentilles. En effet, les différentes longueurs d'onde de la lumière, dû à leur diverse indice de réfraction, ne se focalisent pas au même point ([Source 2](#)). Il existe deux types d'aberrations chromatiques, soit les longitudinales et les transversales. Dans le cas des caméras, les transversales sont les plus communes. Celles-ci résultent du décalage latéral des couleurs par rapport au point de focalisation ([Source 4](#)). Une méthode de correction de ce type d'aberration est d'utiliser des verres comportant des caractéristiques qui compensent la réfraction des différentes longueurs d'onde telles que des lentilles achromatiques ou des lentilles à gradient d'indice.

3 Conclusion

A FAIRE

4 Annexe

4.1 Codes contours zoom minimal

```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from statistics import mean, stdev
5 import pandas as pd
6 import math
7
8 taille_reelle_pixel = 3.6e-3
9
10 # Etape 1 : Charger l'image
11 image = cv2.imread('C:/Users/tomde/Desktop/Cours/Autonome 2024/Optique/Laboratoire 2/Res/
12   res_1m_min.png')
13
14 # Obtenir la hauteur de l'image
15 image_height = image.shape[0]
16 half_height = image_height // 2 # Moitie de l'image
17
18 # Convertir en niveaux de gris
19 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
20
21 # Appliquer un flou gaussien pour reduire le bruit
22 gray_blurred = cv2.GaussianBlur(gray, (5, 5), 0)
23
24 # Appliquer un seuillage adaptatif binaire inverse avec un blockSize petit
25 thresh = cv2.adaptiveThreshold(
26     gray_blurred, 255,
27     cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
28     cv2.THRESH_BINARY_INV,
29     blockSize=11, # BlockSize petit
30     C=2
31 )
32
33 # Operations morphologiques pour ameliorer les formes
34 kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
35 # Fermeture pour combler les petits trous
36 thresh = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel, iterations=3)
37 # Ouverture pour supprimer les petits objets
38 thresh = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations=1)
39
40 # Detection des contours
41 contours, hierarchy = cv2.findContours(
42     thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
43 )
44
45 # Creer une copie de l'image pour le dessin
46 image_contours = image.copy()
47
48 # Liste pour stocker les longueurs des rectangles
49 rectangle_lengths = []
50
51 # Compteur de rectangles
52 rectangle_counter = 1
53
54 # Liste des longueurs a supprimer
55 longueurs_a_supprimer = [37, 69, 78, 79, 87, 103, 104, 105, 106, 107, 111, 112, 173]
```

```
55
56 # Condition spécifique : longueur = 87 et largeur > 12
57 longueur_specific = 87
58 largeur_minimale = 12
59
60 for contour in contours:
61     # Calculer l'aire et le périmètre du contour
62     area = cv2.contourArea(contour)
63     perimeter = cv2.arcLength(contour, True)
64
65     # Filtrer les contours de petite taille
66     if area > 600: # Ajuster ce seuil en fonction de votre image
67         # Approximation du contour
68         epsilon = 0.02 * perimeter # Garder epsilon petit pour une approximation
69         precise
70         approx = cv2.approxPolyDP(contour, epsilon, True)
71
72         # Vérifier si c'est un quadrilatère
73         if len(approx) == 4:
74             # Vérifier si le contour est convexe
75             if cv2.isContourConvex(approx):
76                 # Utiliser le rectangle englobant pour calculer la longueur et la
77                 # largeur
78                 x, y, w, h = cv2.boundingRect(approx)
79                 length = max(w, h)
80                 width = min(w, h)
81
82                 # Vérifier si la position verticale est au-dessus de la moitié de l'
83                 # image
84                 if y + h // 2 < half_height:
85                     # Si le centre du rectangle est au-dessus de la moitié de l'image
86                     length_to_use = width # On utilise la largeur
87                 else:
88                     length_to_use = length # Sinon, on garde la longueur
89
90                 # Vérifier si la longueur fait partie des longueurs à supprimer
91                 if int(length_to_use) in longueurs_a_supprimer:
92                     continue # Ne pas dessiner ni afficher cette longueur
93
94                 if abs(width - length) < 20:
95                     continue
96
97                 rectangle_lengths.append(length_to_use)
98
99                 # Dessiner le contour
100                cv2.drawContours(image_contours, [approx], -1, (0, 255, 0), 2)
101
102                # Afficher le numéro du rectangle et sa longueur ou largeur sur l'image
103                cX = x + w // 2
104                cY = y + h // 2
105                text = f"Dimension : {int(length_to_use)}"
106                cv2.putText(
107                    image_contours,
108                    text,
109                    (cX - 70, cY),
110                    cv2.FONT_HERSHEY_SIMPLEX,
111                    0.5,
112                    (255, 0, 0),
113                    2
```

```
111         )
112         rectangle_counter += 1
113     else:
114         # Gestion des contours non quadrilatéraux
115         rect = cv2.minAreaRect(contour)
116         (center), (width, height), angle = rect
117         length = max(width, height)
118         width = min(width, height)
119
120         # Vérifier si la position verticale est au-dessus de la moitié de l'image
121         if center[1] < half_height:
122             # Si le centre du rectangle est au-dessus de la moitié de l'image
123             length_to_use = width # On utilise la largeur
124         else:
125             length_to_use = length # Sinon, on garde la longueur
126
127         # Vérifier si la longueur fait partie des longueurs à supprimer
128         if int(length_to_use) in longueurs_a_supprimer:
129             continue # Ne pas dessiner ni afficher cette longueur
130
131     rectangle_lengths.append(length_to_use)
132
133     # Obtenir les points du rectangle
134     box = cv2.boxPoints(rect)
135     box = np.intp(box)
136
137     # Dessiner le rectangle
138     cv2.drawContours(image_contours, [box], 0, (0, 0, 255), 2)
139
140     # Afficher le numéro du rectangle et sa longueur ou largeur sur l'image
141     cX, cY = np.intp(center)
142     text = f"Dimension : {int(length_to_use)}"
143     cv2.putText(
144         image_contours,
145         text,
146         (int(cX) - 70, int(cY)),
147         cv2.FONT_HERSHEY_SIMPLEX,
148         0.5,
149         (0, 255, 255),
150         2
151     )
152     rectangle_counter += 1
153
154 # Afficher l'image avec les rectangles détectés
155 image_rgb = cv2.cvtColor(image_contours, cv2.COLOR_BGR2RGB)
156 plt.imshow(image_rgb)
157 plt.axis('off')
158 plt.show()
159
160 # Calcul de la moyenne et de l'écart-type pour chaque groupe de rectangles similaires
161
162 # Trier les longueurs détectées
163 rectangle_lengths.sort()
164
165 # Grouper les rectangles similaires (différence maximale de 10)
166 grouped_lengths = []
167 current_group = [rectangle_lengths[0]]
168
169 for i in range(1, len(rectangle_lengths)):
```

```
170     if abs(rectangle_lengths[i] - rectangle_lengths[i - 1]) <= 10:
171         current_group.append(rectangle_lengths[i])
172     else:
173         grouped_lengths.append(current_group)
174         current_group = [rectangle_lengths[i]]
175
176 # Ajouter le dernier groupe
177 if current_group:
178     grouped_lengths.append(current_group)
179
180 # Associer chaque groupe à une taille en mm (1mm, 2mm, 10mm, 25mm, 50mm)
181 group_to_size = {1: '1', 2: '2', 3: '10', 4: '25', 5: '50'}
182
183 # Création des colonnes pour la taille du groupe, la moyenne et l'écart-type
184 data = []
185
186 # Calcul de la moyenne et de l'écart-type pour chaque groupe
187 for idx, group in enumerate(grouped_lengths, 1):
188     if len(group) > 1: # Calculer uniquement si le groupe contient plus d'un rectangle
189         group_std = stdev(group)
190     else:
191         group_std = 0
192     group_mean = mean(group)
193     size = group_to_size.get(idx, "Inconnu")
194     data.append([size, group_mean, group_std])
195
196 # Création du tableau avec pandas
197 df = pd.DataFrame(data, columns=['Taille (mm)', 'Moyenne (pixels)', 'Ecart-type (pixels)'])
198
199 print(df)
200
201 # Calcul de la résolution (taille en mm / moyenne en pixels)
202 df['Taille (mm)'] = df['Taille (mm)'].astype(float) # Convertir la taille en numérique
203 df['Résolution (mm/pixel)'] = df['Taille (mm)'] / df['Moyenne (pixels)']
204
205 # Calcul de la moyenne et de l'écart-type des résolutions
206 resolution_mean = df['Résolution (mm/pixel)'].mean()
207 resolution_std = df['Résolution (mm/pixel)'].std()
208
209 if math.isnan(resolution_std):
210     resolution_std = 0
211
212 # Afficher le tableau des statistiques de résolution
213 print(f"\nMoyenne de la résolution : {resolution_mean:.6f} mm/pixel")
214 print(f"Ecart-type de la résolution : {resolution_std:.6f} mm/pixel")
215
216 # Calcul du grossissement
217 df['Taille (mm)'] = df['Taille (mm)'].astype(float) # Convertir la taille en numérique
218 df['Grossissement (mm/pixel)'] = (df['Moyenne (pixels)'] * taille_reelle_pixel)/df['Taille (mm)']
219
220 # Calcul de la moyenne et de l'écart-type du grossissement
221 grossissement_mean = df['Grossissement (mm/pixel)'].mean()
222 grossissement_std = df['Grossissement (mm/pixel)'].std()
223
224 if math.isnan(grossissement_std):
225     grossissement_std = 0
```

```

227 # Afficher le tableau des statistiques du grossissement
228 print(f"\nMoyenne du grossissement : {grossissement_mean:.6f}")
229 print(f"Ecart-type du grossissement : {grossissement_std:.6f}")
230
231 # Sauvegarder l'image modifiée
232 output_image_path = 'C:/Users/tomde/Desktop/Cours/Autonme 2024/Optique/Laboratoire 2/Res/
    /rectangles_detectes_min.png'
233 cv2.imwrite(output_image_path, image_contours)
234
235 # Sauvegarder le tableau en LaTeX
236 latex_table = df.to_latex(index=False)
237
238 # Écrire le tableau LaTeX dans un fichier .tex
239 output_latex_path = 'C:/Users/tomde/Desktop/Cours/Autonme 2024/Optique/Laboratoire 2/Res/
    /tableau_longueurs_min.tex'
240 with open(output_latex_path, 'w') as f:
241     f.write(latex_table)
242
243 #print(f"Image sauvegardee : {output_image_path}")
244 #print(f"Tableau LaTeX sauvegarde : {output_latex_path}")

```

4.2 Codes contours zoom maximal

```

1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from statistics import mean, stdev
5 import pandas as pd
6 import math
7
8 taille_reelle_pixel = 3.6e-3
9
10 # Etape 1 : Charger l'image
11 image = cv2.imread('C:/Users/tomde/Desktop/Cours/Autonme 2024/Optique/Laboratoire 2/Res/
    /res_1m_max.png')
12
13 # Obtenir la hauteur de l'image
14 image_height = image.shape[0]
15 half_height = image_height // 2 # Moitié de l'image
16
17 # Délimiter la zone d'intérêt
18
19 x_start, y_start, x_end, y_end = [480, 200, 700, 450]
20
21 # Convertir en niveaux de gris
22 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
23
24 # Créer un masque de l'image de même taille et le remplir de blanc
25 mask = np.ones_like(gray)
26
27 # Définir la zone d'intérêt sur le masque et la remplir avec les pixels d'origine
28 mask[y_start:y_end, x_start:x_end] = gray[y_start:y_end, x_start:x_end]
29
30 masked_image_rgb = cv2.cvtColor(mask, cv2.COLOR_GRAY2RGB)
31
32 # Afficher l'image modifiée
33 plt.imshow(masked_image_rgb)
34 plt.axis('off')

```

```
35 plt.show()
36
37 # Appliquer un flou gaussien pour reduire le bruit
38 gray_blurred = cv2.medianBlur(mask, 5)
39
40 # Appliquer un seuillage adaptatif binaire inverse avec un blockSize petit
41 thresh = cv2.adaptiveThreshold(
42     gray_blurred, 255,
43     cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
44     cv2.THRESH_BINARY_INV,
45     blockSize=11, # BlockSize petit
46     C=2
47 )
48
49 # Operations morphologiques pour ameliorer les formes
50 kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (2, 3))
51 # Fermeture pour combler les petits trous
52 thresh = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel, iterations=4)
53 # Ouverture pour supprimer les petits objets
54 thresh = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations=2)
55
56 plt.imshow(thresh)
57 plt.axis('off')
58 plt.show()
59
60 # Detection des contours
61 contours, hierarchy = cv2.findContours(
62     thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
63 )
64
65 # Creer une copie de l'image pour le dessin
66 image_contours = image.copy()
67
68 # Liste pour stocker les longueurs des rectangles
69 rectangle_lengths = []
70
71 # Compteur de rectangles
72 rectangle_counter = 1
73
74 for contour in contours:
75     # Calculer l'aire et le perimetre du contour
76     area = cv2.contourArea(contour)
77     perimeter = cv2.arcLength(contour, True)
78
79     # Filtrer les contours de petite taille
80     if area > 800: # Ajuster ce seuil en fonction de votre image
81         # Approximation du contour
82         epsilon = 0.02 * perimeter # Garder epsilon petit pour une approximation
83         precise
84         approx = cv2.approxPolyDP(contour, epsilon, True)
85
86         # Vérifier si c'est un quadrilatère
87         if len(approx) == 4:
88             # Vérifier si le contour est convexe
89             if cv2.isContourConvex(approx):
90                 # Utiliser le rectangle englobant pour calculer la longueur et la
91                 largeur
92                 x, y, w, h = cv2.boundingRect(approx)
93                 length = max(w, h)
```

```
92         width = min(w, h)
93         length_to_use = width
94
95         if abs(width - length) < 20:
96             continue
97
98         rectangle_lengths.append(length_to_use)
99
100        # Dessiner le contour
101        cv2.drawContours(image_contours, [approx], -1, (0, 255, 0), 2)
102
103        # Afficher le numero du rectangle et sa longueur ou largeur sur l'image
104        cX = x + w // 2
105        cY = y + h // 2
106        text = f"Dimension : {int(length_to_use)}"
107        cv2.putText(
108            image_contours,
109            text,
110            (cX - 70, cY),
111            cv2.FONT_HERSHEY_SIMPLEX,
112            0.5,
113            (255, 0, 0),
114            2
115        )
116        rectangle_counter += 1
117    else:
118        # Gestion des contours non quadrilateraux
119        rect = cv2.minAreaRect(contour)
120        (center), (width, height), angle = rect
121        length = max(width, height)
122        width = min(width, height)
123
124        # Vérifier si la position verticale est au-dessus de la moitié de l'image
125        if center[1] < half_height:
126            # Si le centre du rectangle est au-dessus de la moitié de l'image
127            length_to_use = width # On utilise la largeur
128        else:
129            length_to_use = length # Sinon, on garde la longueur
130
131        rectangle_lengths.append(length_to_use)
132
133        # Obtenir les points du rectangle
134        box = cv2.boxPoints(rect)
135        box = np.intp(box)
136
137        # Dessiner le rectangle
138        cv2.drawContours(image_contours, [box], 0, (0, 0, 255), 2)
139
140        # Afficher le numero du rectangle et sa longueur ou largeur sur l'image
141        cX, cY = np.intp(center)
142        text = f"Dimension : {int(length_to_use)}"
143        cv2.putText(
144            image_contours,
145            text,
146            (int(cX) - 70, int(cY)),
147            cv2.FONT_HERSHEY_SIMPLEX,
148            0.5,
149            (0, 255, 255),
150            2
```

```
151         )
152         rectangle_counter += 1
153
154 # Afficher l'image avec les rectangles detectes
155 image_rgb = cv2.cvtColor(image_contours, cv2.COLOR_BGR2RGB)
156 plt.imshow(image_rgb)
157 plt.axis('off')
158 plt.show()
159
160 # Calcul de la moyenne et de l'écart-type pour chaque groupe de rectangles similaires
161
162 # Trier les longueurs detectees
163 rectangle_lengths.sort()
164
165 # Grouper les rectangles similaires (difference maximale de 10)
166 grouped_lengths = []
167 current_group = [rectangle_lengths[0]]
168
169 for i in range(1, len(rectangle_lengths)):
170     if abs(rectangle_lengths[i] - rectangle_lengths[i - 1]) <= 10:
171         current_group.append(rectangle_lengths[i])
172     else:
173         grouped_lengths.append(current_group)
174         current_group = [rectangle_lengths[i]]
175
176 # Ajouter le dernier groupe
177 if current_group:
178     grouped_lengths.append(current_group)
179
180 # Associer chaque groupe a une taille en mm (1mm, 2mm, 10mm, 25mm, 50mm)
181 group_to_size = {1: '1'}
182
183 # Creation des colonnes pour la taille du groupe, la moyenne et l'écart-type
184 data = []
185
186 # Calcul de la moyenne et de l'écart-type pour chaque groupe
187 for idx, group in enumerate(grouped_lengths, 1):
188     if len(group) > 1: # Calculer uniquement si le groupe contient plus d'un rectangle
189         group_std = stdev(group)
190     else:
191         group_std = 0
192     group_mean = mean(group)
193     size = group_to_size.get(idx, "Inconnu")
194     data.append([size, group_mean, group_std])
195
196 # Creation du tableau avec pandas
197 df = pd.DataFrame(data, columns=['Taille (mm)', 'Moyenne (pixels)', 'Ecart-type (pixels)'])
198
199 print(df)
200
201 # Calcul de la resolution (taille en mm / moyenne en pixels)
202 df['Taille (mm)'] = df['Taille (mm)'].astype(float) # Convertir la taille en numerique
203 df['Resolution (mm/pixel)'] = df['Taille (mm)'] / df['Moyenne (pixels)']
204
205 # Calcul de la moyenne et de l'écart-type des resolutions
206 resolution_mean = df['Resolution (mm/pixel)'].mean()
207 resolution_std = df['Resolution (mm/pixel)'].std()
```

```
209 if math.isnan(resolution_std):
210     resolution_std = 0
211
212 # Afficher le tableau des statistiques de resolution
213 print(f"\nMoyenne de la resolution : {resolution_mean:.6f} mm/pixel")
214 print(f"Ecart-type de la resolution : {resolution_std:.6f} mm/pixel")
215
216 # Calcul du grossissement
217 df['Taille (mm)'] = df['Taille (mm)'].astype(float) # Convertir la taille en numerique
218 df['Grossissement (mm/pixel)'] = (df['Moyenne (pixels)'] * taille_reelle_pixel)/df['
    Taille (mm)']
219
220 # Calcul de la moyenne et de l'ecart-type du grossissement
221 grossissement_mean = df['Grossissement (mm/pixel)'].mean()
222 grossissement_std = df['Grossissement (mm/pixel)'].std()
223
224 if math.isnan(grossissement_std):
225     grossissement_std = 0
226
227 # Afficher le tableau des statistiques du grossissement
228 print(f"\nMoyenne du grossissement : {grossissement_mean:.6f}")
229 print(f"Ecart-type du grossissement : {grossissement_std:.6f}")
230
231 # Sauvegarder l'image modifiee
232 output_image_path = 'C:/Users/tomde/Desktop/Cours/Autonme 2024/Optique/Laboratoire 2/Res
    /rectangles_detectes_min.png'
233 cv2.imwrite(output_image_path, image_contours)
234
235 # Sauvegarder le tableau en LaTeX
236 latex_table = df.to_latex(index=False)
237
238 # Ecrire le tableau LaTeX dans un fichier .tex
239 output_latex_path = 'C:/Users/tomde/Desktop/Cours/Autonme 2024/Optique/Laboratoire 2/Res
    /tableau_longueurs_min.tex'
240 with open(output_latex_path, 'w') as f:
241     f.write(latex_table)
242
243 #print(f"Image sauvegardee : {output_image_path}")
244 #print(f"Tableau LaTeX sauvegarde : {output_latex_path}")
```