CSE483: Computer Vision

Lab 1

Name: Alaa Mohamed Mohamed Hamdy Ahmed

ID: 19P6621

Group: 1 Section: 1

Task Required

Develop a code like the previous code [use other Kernels or some Open CV functions] to perform the following on any image from your choice:

1. Feature Extraction using SIFT
2. Feature Matching
3. Transformations (scaling, rotation, translation)

Table of Contents

Table of Contents

Test Images

Figure 2 Image for feature extraction

Figure 1 Image for feature matching

SIFT for Feature Extraction

Step 1 : Import needed libraries and packages

```
import cv2
import matplotlib.pyplot as plt
```

✓ 0.0s

Step 2: Load the 2 images we will operate on

```
# load the images
featureExtractionImage= cv2.imread("books.jpg")
featureMatchingImage= cv2.imread("zoomedbook.jpg")
```

Load th two images from disk and stores them in the featureExtractionImage and featureMatchingImage variables.

Step 3: Convert the images to grayscale

```
# Convert the images to grayscale
gray1 = cv2.cvtColor(featureExtractionImage, cv2.COLOR_BGR2GRAY) #featureExtractionImage
gray2 = cv2.cvtColor(featureMatchingImage, cv2.COLOR_BGR2GRAY)  #featureMatchingImage
```

Convert the color images to grayscale using the cv2.cvtColor() function. This is necessary because most feature detectors and descriptors work on grayscale images.

Step 4: Initialize the SIFT object detector

```
# Initialize SIFT detector
sift = cv2.xfeatures2d.SIFT_create()
```

This code creates an instance of the SIFT detector using the cv2.xfeatures2d.SIFT_create() function.

Step 5: Detect keypoints and compute descriptor in both images

```
# Detect keypoints and compute descriptors
keypoints_E, descriptors_E = sift.detectAndCompute(gray1, None) #featureExtractionImage
keypoints_M, descriptors_M = sift.detectAndCompute(gray2, None) #featureMatchingImage
```

Uses the detectAndCompute() method of the SIFT detector to extract keypoints and descriptors for both images. The keypoints and descriptors are stored in the keypoints_E, keypoints_M, descriptors_E, and descriptors_M variables.

Step 6: Draw the keypoints of each image on the image itself

```
# Draw keypoints on the image
featureExtractionImage_with_keypoints = cv2.drawKeypoints(featureExtractionImage, keypoints_E, None)  #featureExtractionImage
featureMatchingImage_with_keypoints = cv2.drawKeypoints(featureMatchingImage, keypoints_M, None) #featureMatchingImage
```

Draw the extracted keypoints of each image on the image itself, using drawKeypoints() function

Step 7: Plot the images

```
plt.imshow(featureExtractionImage_with_keypoints)
plt.xlabel('Extracted features from Image used for feature extraction')
plt.show()
plt.imshow(featureMatchingImage_with_keypoints)
plt.xlabel('Features of feature matching image')
plt.show()
```

Plot the 2 images

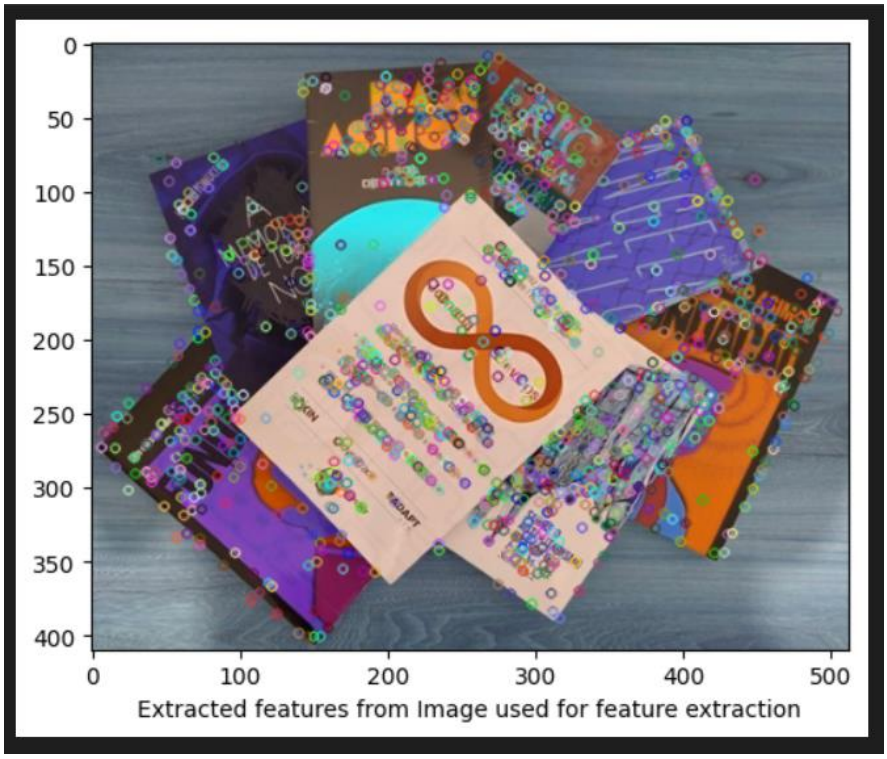

Figure 4 Extracted features from Image used for feature extraction

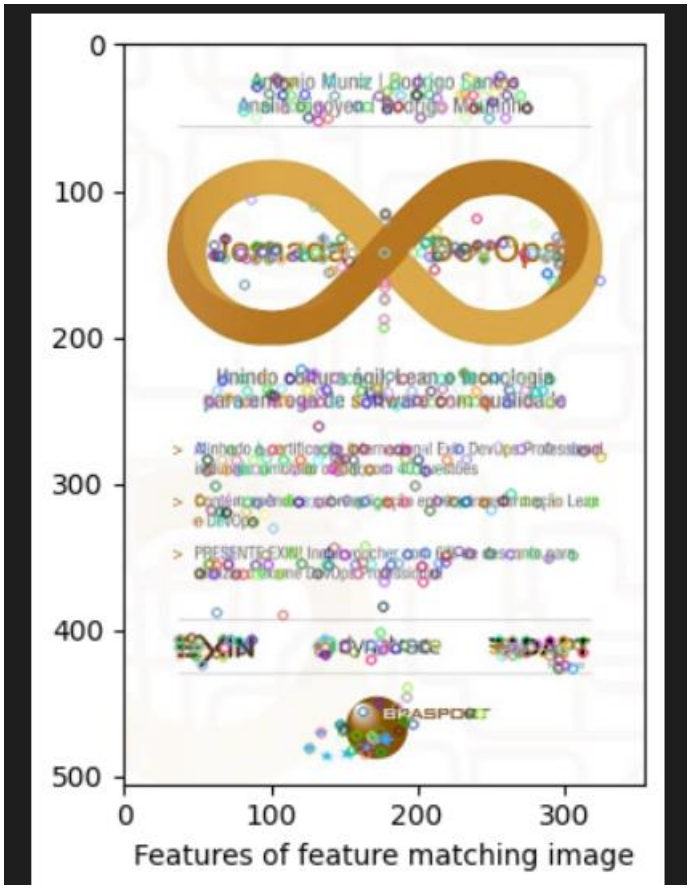

Figure 3 Extracted features from Image used for feature matching

Step 8: Initialize Brute-force matcher for feature matching

```
# Initialize Brute-Force matcher
bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
```

Create an instance of the Brute-Force matcher with L2 distance using the cv2.BFMatcher() function.

Step 9: Match the computed descriptors of both images using Brute-force matcher

```
# Match descriptors using feature matcher
matches = bf.match(descriptors_E, descriptors_M)
```

Match the descriptors from both images using the match() method of the matcher. The matches are stored in the matches variable.

Step 10: Sort matches by distance

```
# Sort matches by distance
matches = sorted(matches, key=lambda x: x.distance)
```

Sort the matches by distance using the sorted() function and a lambda function that extracts the distance value from each match.

Step 11: Draw matched features on a new image

```python
# Draw matches on a new image
matched_image = cv2.drawMatches(featureExtractionImage, keypoints_E, featureMatchingImage, keypoints_M, matches[:55], None, flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
```

Python

Use the drawMatches() function to draw the top 55 matches on a new image. The function takes the two input images, the keypoints and descriptors for both images, the matches to be drawn, and some optional parameters. The resulting image is stored in the matched_image variable.

Step 12: Plot Matched Image

```python
plt.imshow(matched_image)
plt.xlabel('Matched Image')
plt.show()
```

Plot the matched image

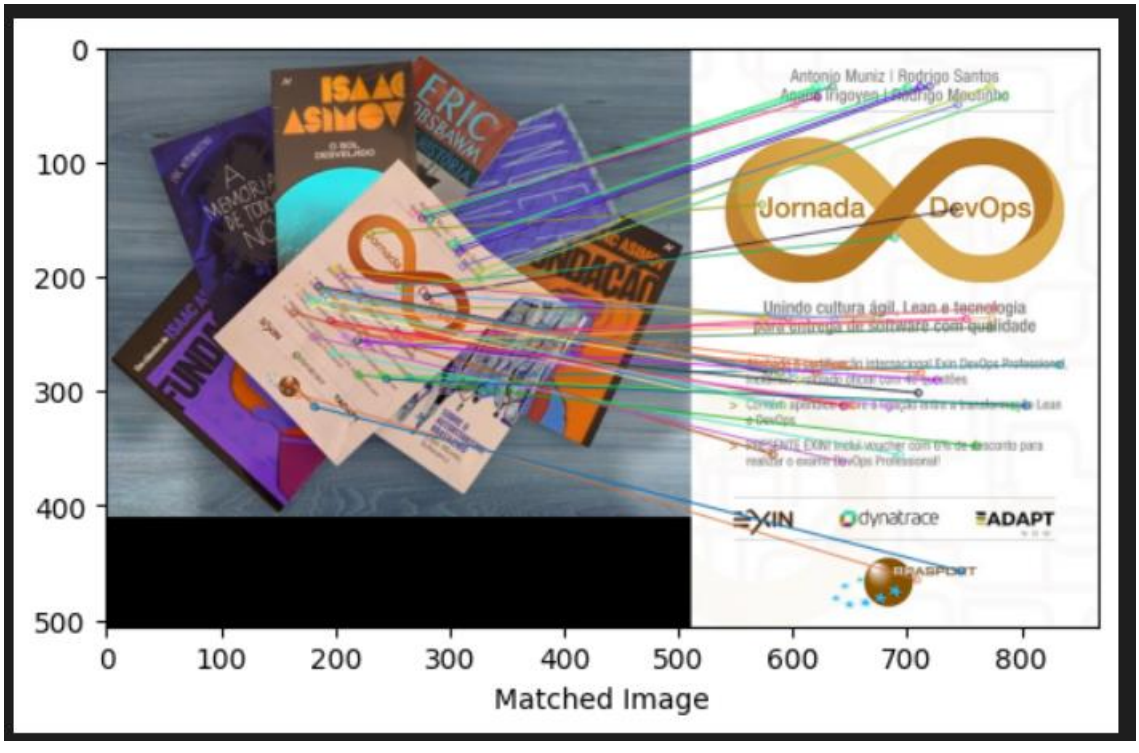

Figure 5 Matched Image

Code

```
import cv2
import matplotlib.pyplot as plt

# load the images
featureExtractionImage= cv2.imread("books.jpg")
featureMatchingImage= cv2.imread("zoomedbook.jpg")
# Convert the images to grayscale
gray1 = cv2.cvtColor(featureExtractionImage, cv2.COLOR_BGR2GRAY)
#featureExtractionImage
gray2 = cv2.cvtColor(featureMatchingImage,
cv2.COLOR_BGR2GRAY)  #featureMatchingImage
# Initialize SIFT detector
sift = cv2.xfeatures2d.SIFT_create()
# Detect keypoints and compute descriptors
#featureExtractionImage
keypoints_E, descriptors_E = sift.detectAndCompute(gray1, None)
#featureMatchingImage
keypoints_M, descriptors_M = sift.detectAndCompute(gray2, None)
# Draw keypoints on the image
featureExtractionImage_with_keypoints = cv2.drawKeypoints(featureExtractionImage,
keypoints_E, None)  #featureExtractionImage
featureMatchingImage_with_keypoints = cv2.drawKeypoints(featureMatchingImage,
keypoints_M, None) #featureMatchingImage
# Plot features extracted on both images
plt.imshow(featureExtractionImage_with_keypoints)
plt.xlabel('Extracted features from Image used for feature extraction')
plt.show()
plt.imshow(featureMatchingImage_with_keypoints)
plt.xlabel('Features of feature matching image')
plt.show()
# Initialize Brute-Force matcher
bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
# Match descriptors using feature matcher
matches = bf.match(descriptors_E, descriptors_M)
# Sort matches by distance
matches = sorted(matches, key=lambda x: x.distance)
# Draw matches on a new image
matched_image = cv2.drawMatches(featureExtractionImage, keypoints_E,
featureMatchingImage, keypoints_M, matches[:55], None,
flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
plt.imshow(matched_image)
plt.xlabel('Matched Image')
plt.show()
```

Applying Transformations

Translation

Translation

```
import cv2
import numpy as np

# Load the image
original_image = cv2.imread('zoomedbook.jpg')

# Define the translation matrix
tx = 50  # Translation in x direction
ty = 100  # Translation in y direction
translation_matrix = np.float32([[1, 0, tx], [0, 1, ty]])

# Apply the translation matrix to the image
translated_image = cv2.warpAffine(original_image, translation_matrix, (original_image.shape[1], original_image.shape[0]))

# Display the original and translated image

plt.imshow(original_image)
plt.xlabel('Original image')
plt.show()

plt.imshow(translated_image)
plt.xlabel('Translated image')
plt.show()
```

We first load the image using the cv2.imread() function. Then, we define the translation matrix using the np.float32() function. The translation matrix is a 2x3 matrix, where the first row represents the x direction translation (in pixels), the second row represents the y direction translation (in pixels), and the third row is a dummy row (not used in translation).

Next, we apply the translation matrix to the image using the cv2.warpAffine() function. The function takes the image, the transformation matrix, and the output image size as input, and returns the translated image.

Note that in this example, we have translated the image by 50 pixels in the x direction and 100 pixels in the y direction, but you can use any translation

values you want. Also, keep in mind that translating an image can result in a loss of image content if the translated area goes beyond the boundaries of the original image.

Code

```
import cv2
import numpy as np

# Load the image
original_image = cv2.imread('zoomedbook.jpg')

# Define the translation matrix
tx = 50  # Translation in x direction
ty = 100  # Translation in y direction
translation_matrix = np.float32([[1, 0, tx], [0, 1, ty]])

# Apply the translation matrix to the image
translated_image = cv2.warpAffine(original_image, translation_matrix,
(original_image.shape[1], original_image.shape[0]))

# Display the original and translated image

plt.imshow(original_image)
plt.xlabel('Original image')
plt.show()

plt.imshow(translated_image)
plt.xlabel('Translated image')
plt.show()
```

Output

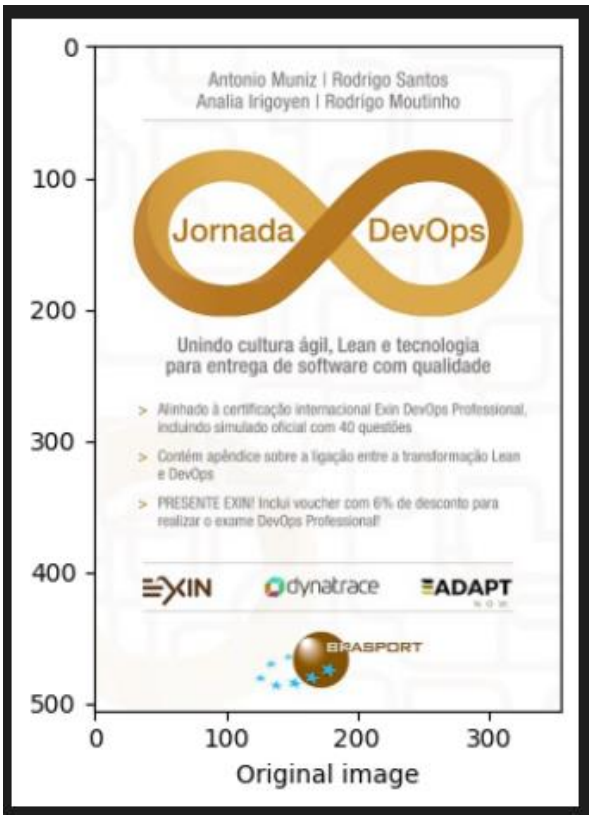

Figure 7 Non-Translated Image

Figure 6 Translated image

Rotation

Rotation

```
import cv2
import numpy as np

# Load the image
original_image = cv2.imread('books.jpg')

# Get the image dimensions
(height, width) = original_image.shape[:2]

# Define the rotation angle
angle = 45

# Define the rotation point (center of the image)
center = (width // 2, height // 2)

# Create the rotation matrix
M = cv2.getRotationMatrix2D(center, angle, 1.0)

# Apply the rotation transformation to the image
rotated_image = cv2.warpAffine(original_image, M, (width, height))

# Display the original and rotated image
plt.imshow(original_image)
plt.xlabel('Original image')
plt.show()

plt.imshow(rotated_image)
plt.xlabel('Rotated image')
plt.show()
```

We first load the image using the cv2.imread() function, and then we get the dimensions of the image using the shape attribute. We then define the rotation angle and the rotation point, which is the center of the image.

Next, we use the cv2.getRotationMatrix2D() function to create the rotation matrix. This function takes three arguments: the center of rotation, the rotation angle in degrees, and the scale factor (which is set to 1.0 in this example).

Finally, we apply the rotation transformation to the image using the cv2.warpAffine() function, which takes three arguments: the input image, the transformation matrix, and the output image size. We then plot the original and the rotated image

Code

```
import cv2
import numpy as np

# Load the image
original_image = cv2.imread('books.jpg')

# Get the image dimensions
(height, width) = original_image.shape[:2]

# Define the rotation angle
angle = 45

# Define the rotation point (center of the image)
center = (width // 2, height // 2)

# Create the rotation matrix
M = cv2.getRotationMatrix2D(center, angle, 1.0)

# Apply the rotation transformation to the image
rotated_image = cv2.warpAffine(original_image, M, (width, height))

# Display the original and rotated image
plt.imshow(original_image)
plt.xlabel('Original image')
plt.show()

plt.imshow(rotated_image)
plt.xlabel('Rotated image')
plt.show()
```

Output

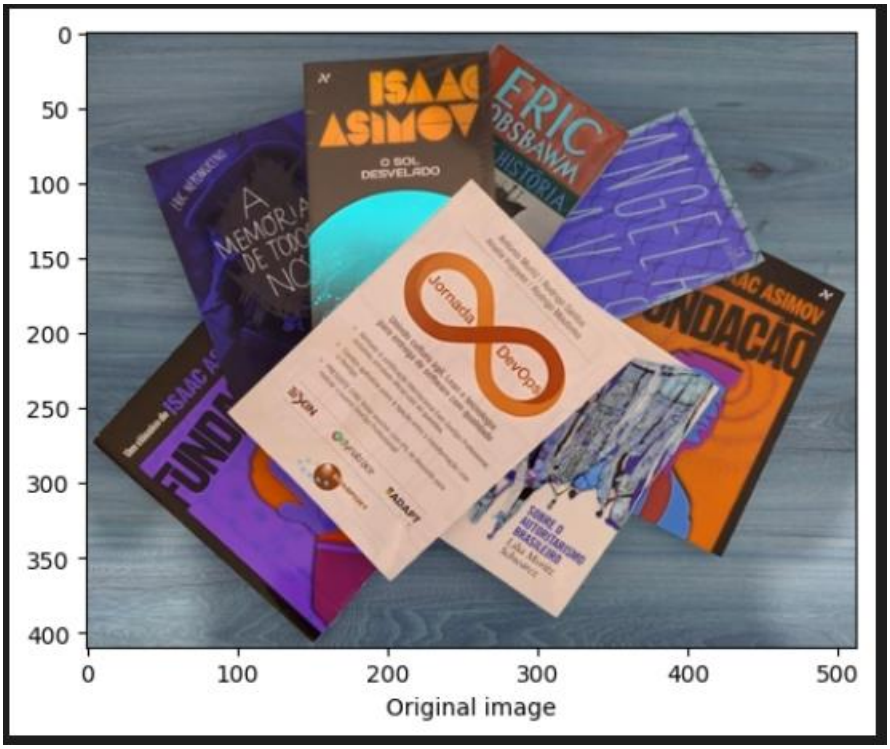

Figure 8 Un-rotated image

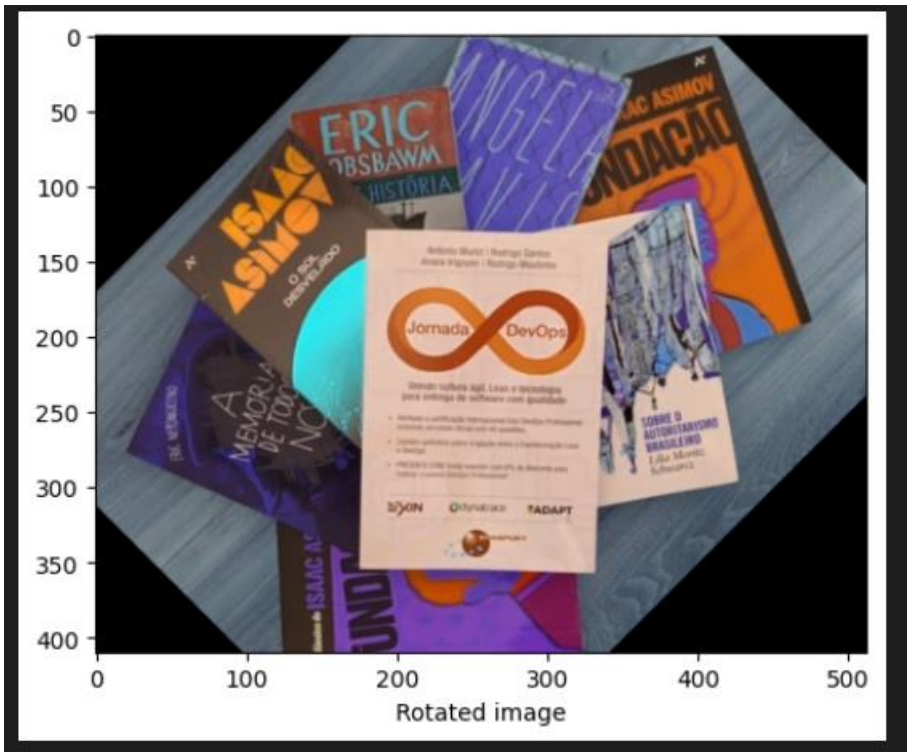

Figure 9 Rotated Image

Scaling

Scaling

```
import cv2

# Load the image
original_image = cv2.imread('books.jpg')

# Get the original image size
original_height, original_width = original_image.shape[:2]

# Define the scale factor
scale_factor = 0.5

# Calculate the new image dimensions
new_height = int(original_height * scale_factor)
new_width = int(original_width * scale_factor)

# Resize the image
resized_image = cv2.resize(original_image, (new_width, new_height))

# Display the original and resized image
plt.imshow(original_image)
plt.xlabel('Original image')
plt.show()

plt.imshow(resized_image)
plt.xlabel('Resized image')
plt.show()
```

We first load the image using the cv2.imread() function. Then, we get the original image size using the shape attribute of the image. Next, we define the scale factor (in this case, 0.5), which will be used to calculate the new image dimensions. We calculate the new dimensions by multiplying the original dimensions with the scale factor and converting them to integers using the int() function. Finally, we use the cv2.resize() function to resize the image to the new dimensions.

Note that in this example, we have scaled the image by half, but you can use any scale factor you want. Also, keep in mind that scaling an image can lead to distortion, so it's important to choose the right scale factor for your specific use case.

Code

```
import cv2

# Load the image
original_image = cv2.imread('books.jpg')

# Get the original image size
original_height, original_width = original_image.shape[:2]

# Define the scale factor
scale_factor = 0.5

# Calculate the new image dimensions
new_height = int(original_height * scale_factor)
new_width = int(original_width * scale_factor)

# Resize the image
resized_image = cv2.resize(original_image, (new_width, new_height))

# Display the original and resized image
plt.imshow(original_image)
plt.xlabel('Original image')
plt.show()

plt.imshow(resized_image)
plt.xlabel('Resized image')
plt.show()
```

Output

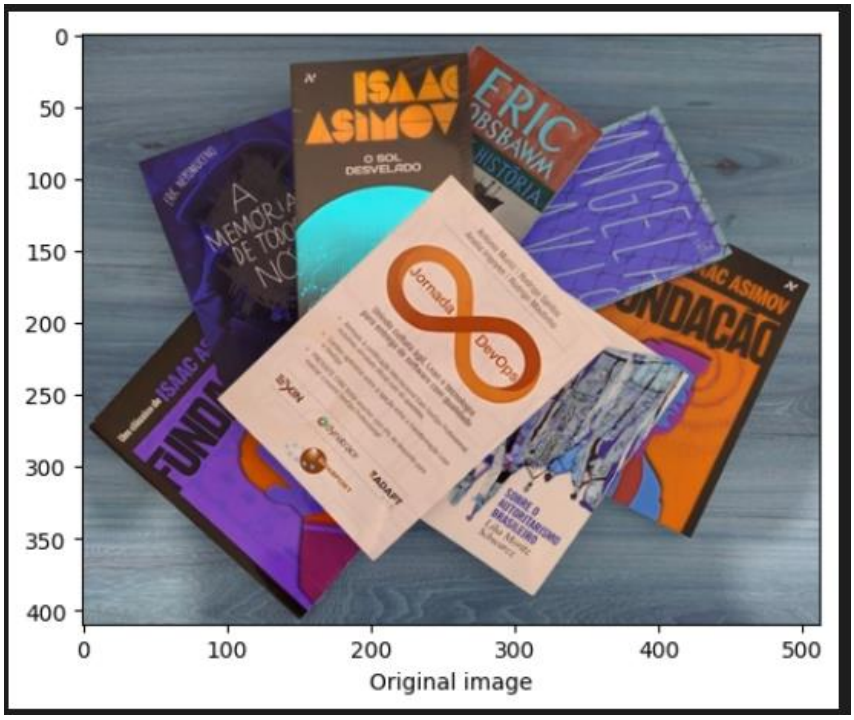

Figure 10 Original size image

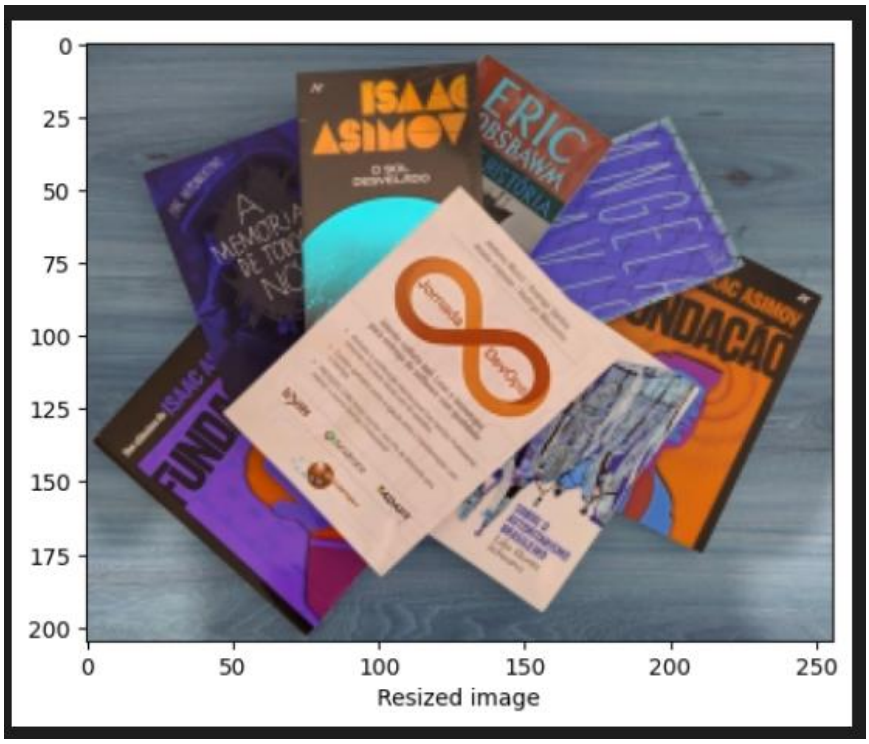

Figure 11 Resized image

Note that the image x-axis scale have been reduced to the half.

Github repository

https://github.com/Alaa-Hamdy/Lab1-CSE483-Computer-Vision.git