الجلسة الخامسة

Feature Extraction

SIFT Features & SIFT feature matching

كشف السمات الثابتة واستخدامها في مطابقة الصور

1.1 خوارزمية SIFT Scaled Invariant Feature Transform

- هي خوارزمية في مجال الرؤية الحاسوبية طورها البروفيسور الكندي ديفيد لو في عام 1999. وتعتبر من أهم الخوارزميات المستخدمة لأغراض التعرف على الاجسام.
- تعتبر من أهم خوارزميات الرؤية الحاسوبية لكشف السمات بسبب كونها مستقلة عن التحجيم scale والدوران Rotation (orientation) والموقع Location.
- تعتبر مستقلة بشكل جزئي عن التغيرات الأخرى مثل الإضاءة والتباين.

1.2 التنفيذ العملي:

في هذا الكود سنقوم بقراءة صورتين ثم حساب سمات SIFT لكل منهما ثم عرضها حيث تمثل الصورة الأولى صورة لعدة قطع نقدية ورقية، في حين تمثل الصورة الثانية صورة قطعة نقدية واحدة فقط تمثل أحد أنواع القطع النقدية الورقية الموجودة في الصورة الأولى. سنقوم بعدها بإجراء عملية مطابقة للسمات بين الصورة الأصلية والصورة القالب بحيث يتم العثور على أفضل مكان للقالب ضمن الصورة الأصلية وعرضه إضافة لعرض السمات المتطابقة بين الصورتين.

تم تحصيل الصور من الموقع

https://medium.com/analytics-vidhya/opencv-feature-matching-sift-algorithm-scale-invariant-feature-transform-16672eafb253

```
import cv2

import numpy as np

import matplotlib.pyplot as plt

# Load the images

original = cv2.imread('original.jpg', 1)  # Original image

template = cv2.imread('template1.jpg', 1)  # Template image
```

قراءة الصورتين الأصلية والقالب بصيغتهما الملونة

```
# Create SIFT object
sift = cv2.SIFT_create()
# Find keypoints and descriptors with SIFT for both images
kp1, des1 = sift.detectAndCompute(original, None)
kp2, des2 = sift.detectAndCompute(template, None)
# Draw keypoints on the images
original_keypoints = cv2.drawKeypoints(original, kp1, None,
flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
template_keypoints = cv2.drawKeypoints(template, kp2, None,
flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
# Display the images with keypoints
cv2.imshow('Original Image with Keypoints', original_keypoints)
cv2.imshow('Template Image with Keypoints', template_keypoints)
cv2.waitKey(0)
cv2.destroyAllWindows()
# FLANN parameters for feature matching
FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=20)
search_params = dict(checks=5)
flann = cv2.FlannBasedMatcher(index_params, search_params)
#flann= cv2.BFMatcher()
matches = flann.knnMatch(des1, des2, k=2)
# Need to draw only good matches (ratio test as per Lowe's paper)
good_matches = []
for m, n in matches:
    if m.distance < 0.7* n.distance:
        good_matches.append(m)
```

تطبيق تحويل SIFT على الصورتين وفي حال أردنا استخدام قناع mask لتحديد جزء من الصورة نمرر القناع بدلاً من None

نحصل من التحويل على kp,des

Kp النقاط المهمة key points

Des السمات أو الواصفات أو المعلومات المهمة المتعلقة بكل نقطة مهمة ضمن جوار هذه النقطة

رسم النقاط المهمة والسمات أو الواصفات على كل من الصورة الأصلية والقالب

هنا نستخدم خوارزمية FLANN للبحث عن التطابقات بين سمات أو واصفات الصورة الأصلية والقالب Des1, Des2

تعيد هذه الخوارزمية مصفوفة Matches تمثل التطابقات بين سمات الصورتين باستخدام خوارزمية الجار الأقرب KNN حيث يتم استخدام K=2 كون أننا نبحث عن تطابق بين سمتين من الصورتين

تستخدم خوارزمية FLANN مبدأ البحث الشجري لذلك تحتاج لتحديد عدد الأشجار المستخدمة لتقسيم فضاء السمات إلى أقسام فرعية وهنا العدد 20 (العدد الأكبر أفضل للدقة)

أما Checks=5 فهي تمثل كم مرة ستقوم FLANN بالتحقق من المجاورات خلال عملية البحث عن أفضل تطابق (القيمة العالية لهذا المعامل تعطي دقة أفضل لكنها تبطئ التنفيذ)

فلترة مصفوفة Matches باستخدام عتبة من أجل الحصول على أفضل التطابقات وإهمال التطابقات التي قد تكون ضجيجية

m,n تمثلان تطابقين اثنين لكل واصفة من الصورة الأصلية مع أقرب واصفتين لها من الصورة القالب.

m.distance و n.distance تمثل المسافتان بين الواصفة في الصورة الأصلية وبين أقرب جارين لها من واصفات القالب، ففي حال كانت المسافتان متقاربتان يتم اعتبار التطابق صحيح والا يهمل

استخدام عتبة أخرى وهي عدد نقاط التطابق التي حصلنا عليها بعد عملية الفلترة فإن كانت هنا أكبر من 35 نقطة نأخذ إحداثيات التطابق ونرسم هذه النقاط ونحدد المستطيل الذي يمثل مكان تطابق القالب مع الصورة الأصلية

```
MIN_MATCH_COUNT = 35  # Adjust this threshold as needed
if len(good_matches) > MIN_MATCH_COUNT:
    src_pts = np.float32([kp1[m.queryIdx].pt for m in good_matches]).reshape(-1, 1, 2)
    dst_pts = np.float32([kp2[m.trainIdx].pt for m in good_matches]).reshape(-1, 1, 2)
    # Estimate the transformation matrix using RANSAC
    M, _ = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
    # Apply the transformation to the template corners
    h, w, ss = template.shape
    template_corners = np.float32([[0, 0], [w - 1, 0], [w - 1, h - 1], [0, h - 1]])
    transformed_corners = cv2.perspectiveTransform(template_corners.reshape(1, -1, 2), M)
    # Draw the bounding box around the detected template
    original_with_box = cv2.polylines(original, [np.int32(transformed_corners)], True, (255, 0, 0), 3)
    # Show the result
    cv2.imshow('Detected Template in Original Image', original_with_box)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    img3=cv2.drawMatchesKnn(original, kp1, template, kp2, [good_matches], None, flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
    plt.imshow(img3)
    plt.show()
else:
    print("Not enough matches are found - %d/%d" % (len(good_matches), MIN_MATCH_COUNT))
```

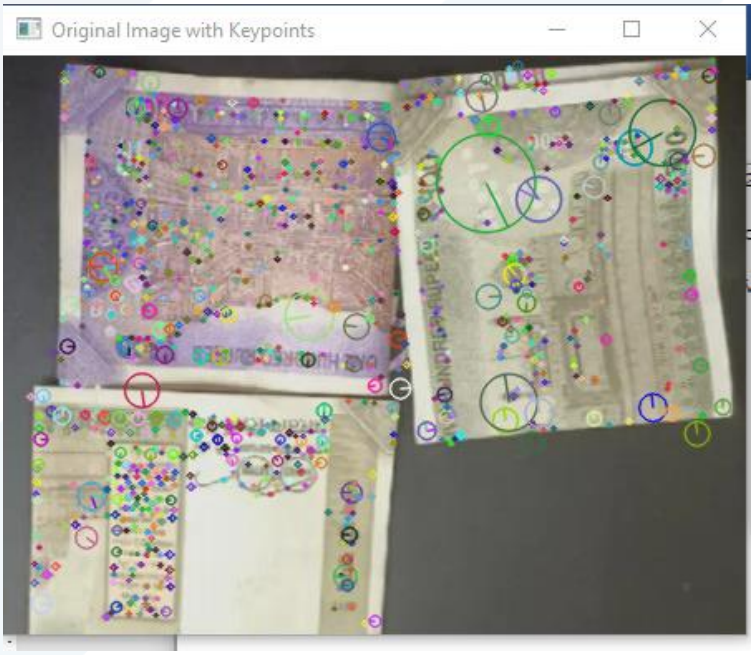
في حال عدم تحقق الشروط السابقة يتم طباعة عبارة أنه لا يوجد تطابق.

ملاحظة: تغيير قيم العتبات يغير نتيجة المطابقة.

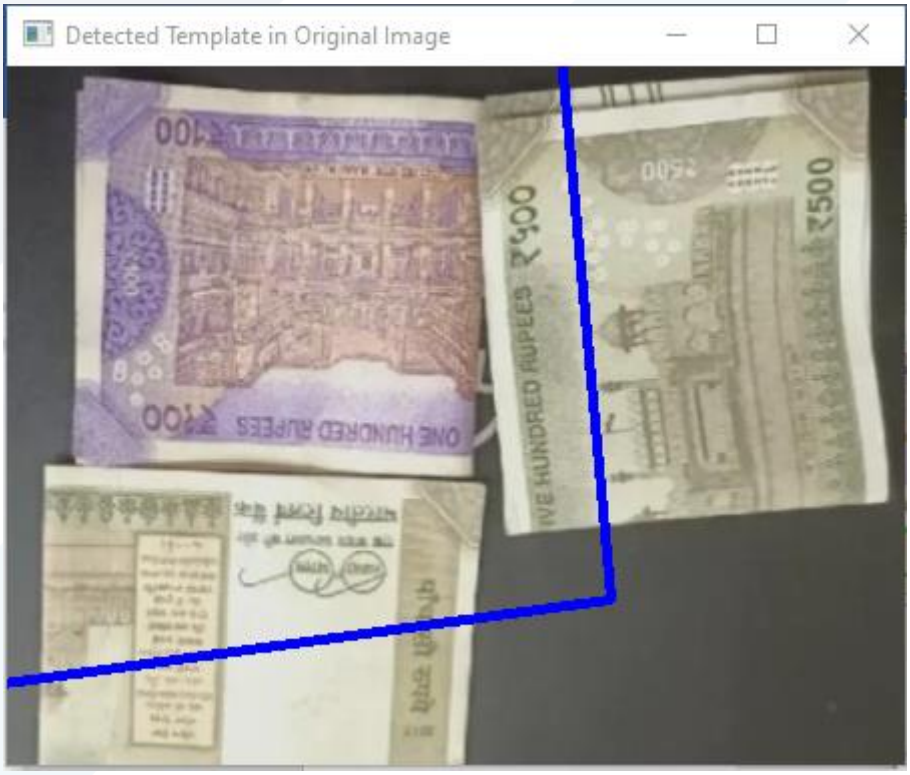
ناتج تنفيذ الكود باستخدام الصورة الأصلية والقالب التالي:

	
Template	Image

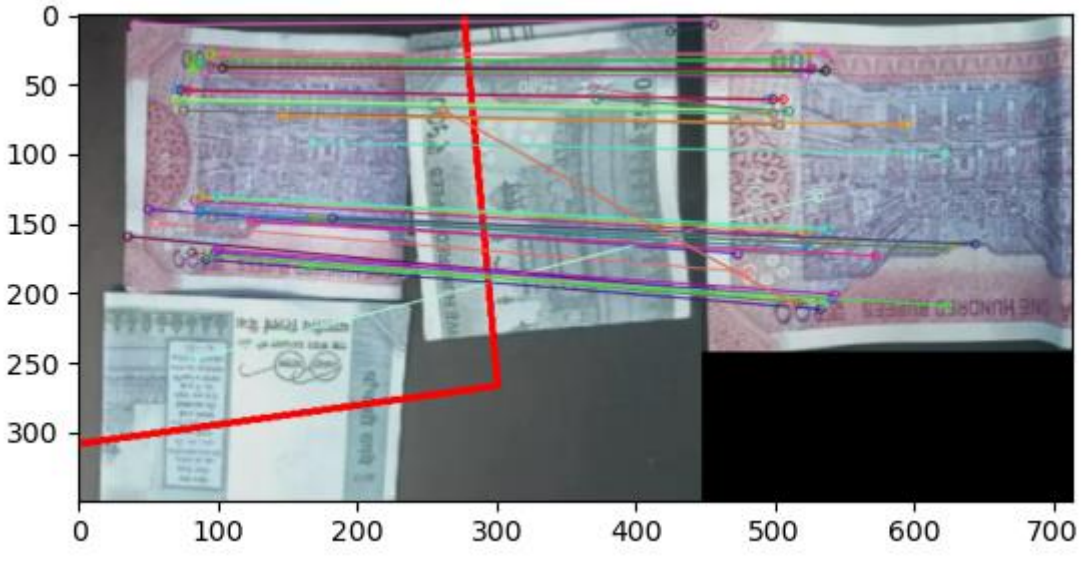
نتيجة عرض سمات SIFT للصورة والقالب:



المستطيل المحيط بالمنطقة التي تمت مطابقتها من الصورة مع القالب.



صورة توضح تطابق السمات بين الصورة الأصلية والقالب:


0
50
100
150
200
250
300
0
100
200
300
400
500
600
700