МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра систем штучного інтелекту

Лабораторна робота № 2

з дисципліни
«Обробка зображень методами штучного інтелекту»

Виконав:
студент групи КН-408
Мокрик Ярослав
Викладач:
Пелешко Д.Д.

Львів – 2022 р.

Тема: Суміщення зображень на основі використання дескрипторів

Мета: навчитись вирішувати задачу суміщення зображень засобом видобування особливих точок і використання їх в процедурах матчінгу

Варіант 11

Завдання

Вибрати з інтернету набори зображень з різною контрастністю і різним флуктуаціями освітленості. Для кожного зображення побудувати варіант спотвореного (видозміненого зображення). Для кожної отриманої пари побудувати дескриптор і проаналізувати можливість суміщення цих зображень і з визначення параметрів геметричних перетворень (кут повороту, зміщень в напрямку x і напрямку y).

11. ORB

Для перевірки збігів необхідно написати власну функцію матчінгу, а результати її роботи перевірити засобами OpenCV. Якщо повної реалізації дескриптора не має в OpenCV, то такий необхідно створити власну функцію побудови цих дискрипторів. У цьому випадку матчінг можна здійснювати стандартними засобами (якщо це можливо).

Код програми

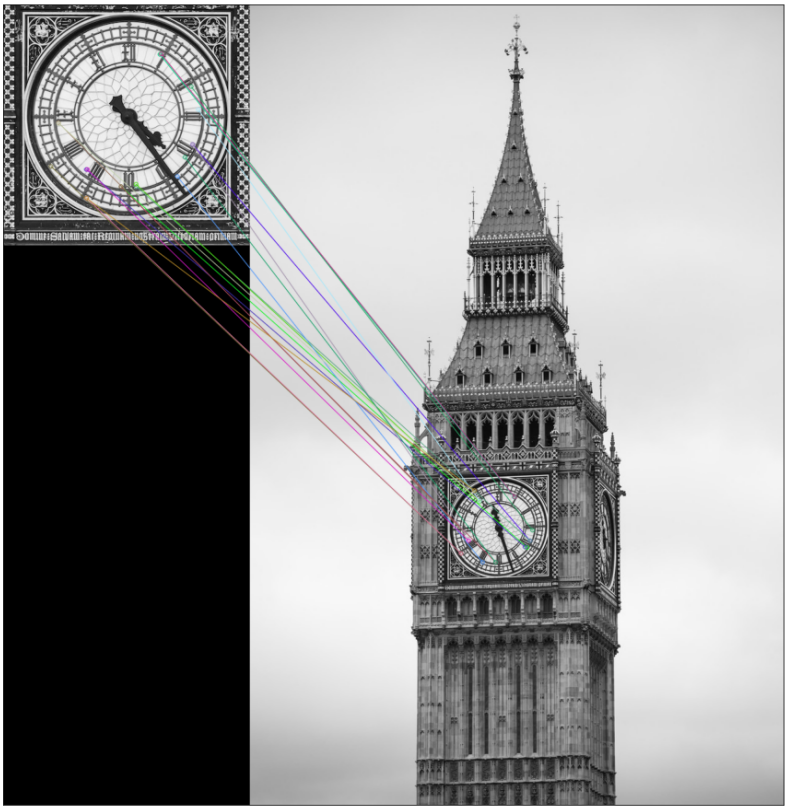
```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
from scipy.spatial.distance import hamming

def orb_detect(img1, img2):
    orb = cv.ORB_create()
    kp1, des1 = orb.detectAndCompute(img1, None)
    kp2, des2 = orb.detectAndCompute(img2, None)

          print("Keypoints:  {},  descriptors:  {}".format(len(kp1),
des1.shape))
          print("Keypoints:  {},  descriptors:  {}".format(len(kp2),
des2.shape))
    return kp1, kp2, des1, des2

def bf_match(img1, img2, kp1, kp2, des1, des2):
    bf = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True)
    matches = bf.match(des1, des2)
```

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

    fig = plt.figure(figsize=(16, 16))
    ax = fig.add_subplot()
    img3 = cv.drawMatches(img1, kp1, img2, kp2, matches[:20], None,

flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(img3)
    plt.show()

def custom_match(img1, img2, kp1, kp2, des1, des2):
    matches = []
    for i, k1 in enumerate(des1):
        for j, k2 in enumerate(des2):
              matches.append(cv.DMatch(_distance = hamming(k1, k2) *
len(k1),
                                       _imgIdx = 0, _queryIdx = i,
_trainIdx = j))
    matches = sorted(matches, key = lambda x: x.distance)

    fig = plt.figure(figsize=(16, 16))
    ax = fig.add_subplot()
    img3 = cv.drawMatches(img1, kp1, img2, kp2, matches[:20], None,

flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(img3)
    plt.show()
```

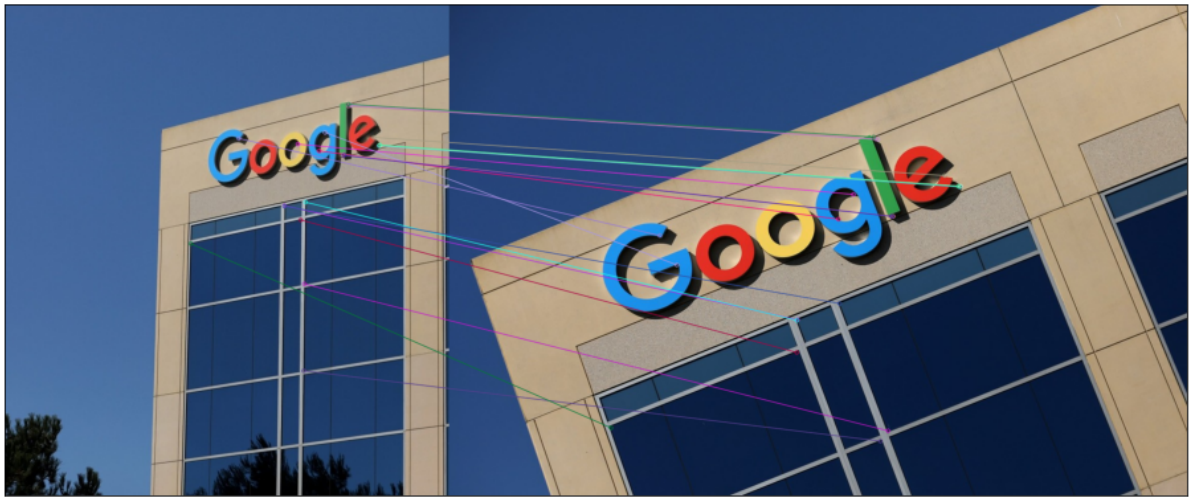
Результати

Проводимо feature matching першого зображення імплементованим в opencv brute force алгоритмом і власним алгоритмом з допомогою ORB:

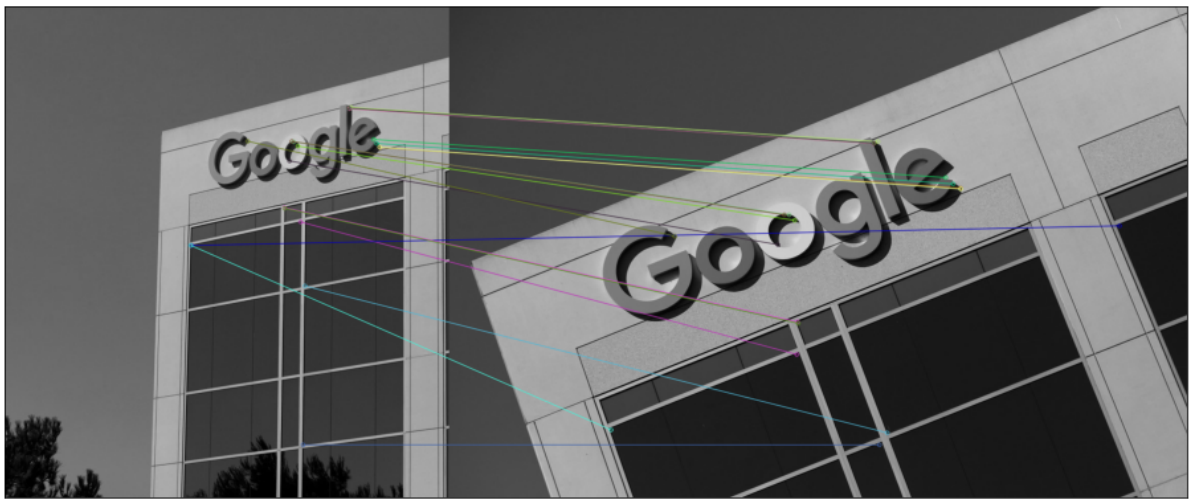
Проводимо feature matching першого зображення, переведеного в чорно-білий формат, імплементованим в opencv brute force алгоритмом і власним алгоритмом з допомогою ORB:

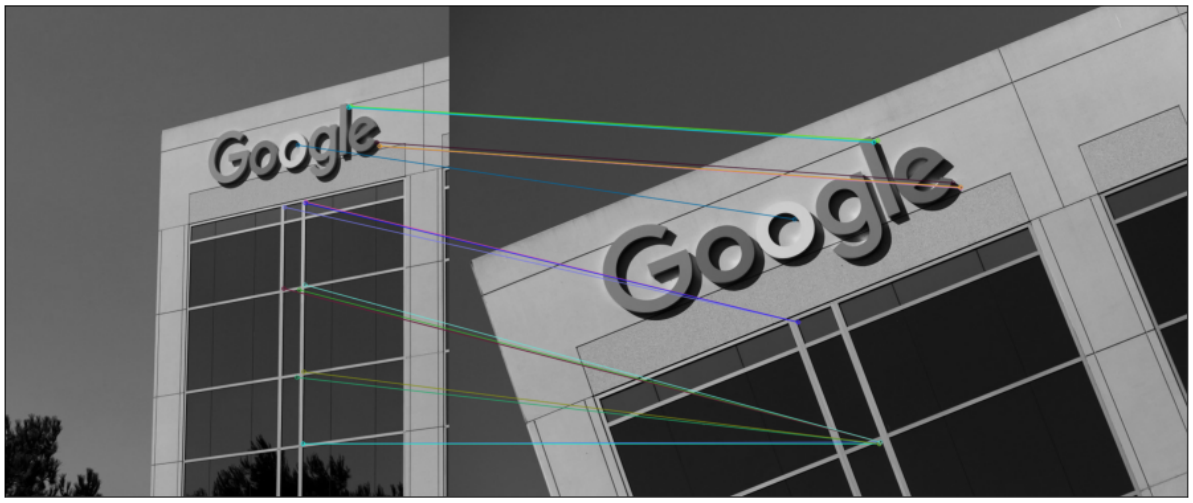


Проводимо feature matching другого зображення імплементованим в opencv brute force алгоритмом і власним алгоритмом з допомогою ORB:



Проводимо feature matching другого зображення, переведеного в чорно-білий формат, імплементованим в opencv brute force алгоритмом і власним алгоритмом з допомогою ORB:





Висновки

Під час виконання цієї лабораторної роботи я навчився вирішувати задачу суміщення зображень засобом видобування особливих точок і використати їх в процедурах матчінгу.

Як бачимо, алгоритм, розроблений мною, працює так само, як вбудована opencv імплементація brute force алгоритму. Вони знаходять такі самі співпадіння між особливими точками зображень. Варто зауважити, що розроблений мною алгоритм є дещо повільнішим порівняно з вбудованим brute force алгоритмом. Також, можна зазначити, що чорно-білі зображення дещо погіршують якість знаходження співпадінь між особливими точками зображень.