**Dokumentáció**

**Bevezetés**

A féléves beadandó témák közül a panoráma kép készítésére esett a választásom. Számomra érdekesnek tűnt hogy hogyan lehet megvalósítani két darab képből egy darabot készíteni. A téma előtt már találkoztam ezzel a programmal az okos telefonok segítségével tudtam készíteni panoráma képeket, de mélyebbre nem sikerült jutnom, hogy hogyan is lehet egy ilyen programot megírni. Mikor a félév során elkezdtem utána nézni, hogy hogyan is lehet egy ilyen programot megvalósítani és megérteni az algoritmusát nem tűnt annyira bonyolultnak.

Csoportosítani lehet képek helyzete szerint:

* Körpanoráma (360 fokos)
* Rácspanoráma (nagy objektumok)
* Sorpanoráma (tájképek, hosszú épületek)
* Oszloppanoráma (magas épületek)

Az én választásom amit bemutatok a féléves beadandóm alkalmával a sorpanoráma lesz. Két kép felhasználásával fogom bemutatni a panoráma kép eredményeimet. Panoráma képet bármennyi képből lehet készíteni ha a képek között van átfedés.

**Elméleti háttér**

**Sarokpont (kulcspont) detektálás:**

Olyan pontokat keresnünk, amelyek megbízhatóan jellemzik az adott képen látható objektumokat.

**Sarokpontok (kulcspontok) párosítása:**

A képekből kinyert pontokat párosítani kell egymással.

**Párosított pontok szűrése:**

Olyan pontokat kell kiválasztani, melyek valóban megegyeznek mindkét objektumon.

**Transzformáció:**

A helyes párosításból elő kell állítanunk a transzformációhoz szükséges adatokat, majd végrehajtani azt.

**Utómunkálatok:**

Két kép illesztése után felmerülhetnek problémák, mint például az illesztési vonalak feltűnősége, vagy kettőnél több kép illesztése.

**Megvalósítás terve, teszteléssel**

**Sarokpont detektálás**

A sarokpontok detektálására az opencv beépített függvényeinek segítségével tudjuk megvalósítani:

1. sift = cv2.xfeatures2d.SIFT\_create()

kp1, des1 = sift.detectAndCompute(I,None)



1. surf = cv2.xfeatures2d.SURF\_create()

kp1, des1 = surf.detectAndCompute(I,None)



1. brisk= cv2.BRISK\_create()

kp1, des1 = brisk.detectAndCompute(I,None)



1. orb = cv2.ORB\_create()

kp1, des1 = orb.detectAndCompute(I,None)

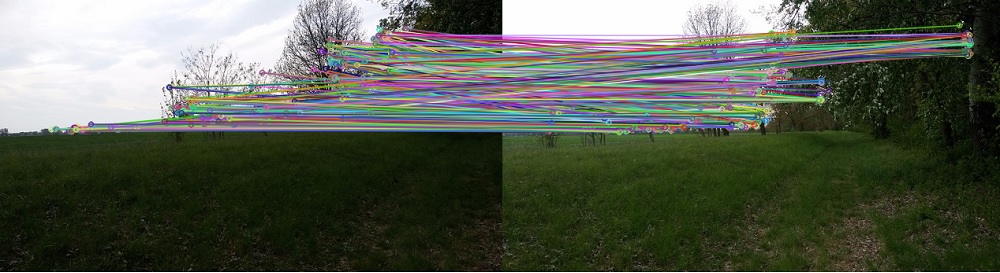


**Sarokpontok párosítása**

orb = cv2.ORB\_create() beépített függvény segítségével megkerestük a sarokpontokat mind a 2 képen, majd párosítottam mindkét képen a megegyező sarokpontokat

bf = cv2.BFMatcher()

matches = bf.knnMatch(des1,des2,k=2)



**Párosított pontok szűrése**

Egy ponthalmazból kiszűri azokat a pontokat, amelyek a legkevésbé illeszkednek közéjük.

****

**Transzformáció**

A RANSAC szűrés után megmaradó pontpárokból felírható egy [túlhatározott lineáris egyenletrendszer](https://en.wikipedia.org/wiki/Overdetermined_system). Ennek az egyenletrendszernek a megoldásából felírható a [homográfia](https://en.wikipedia.org/wiki/Homography), ami az egyik képet a másikba viszi át. A homográfia egy olyan 3x3-as mátrix, ami tartalmazza a szükséges transzformációkat, hogy a két kép illeszkedjen.

src = np.float32([ kp1[i.queryIdx].pt for i in matches[:,0] ]).reshape(-1,1,2)

dst = np.float32([ kp2[i.trainIdx].pt for i in matches[:,0] ]).reshape(-1,1,2)

H, maszk = cv2.findHomography(src, dst, cv2.RANSAC, 5.0)

Ihomografia = cv2.warpPerspective(I,H,(I.shape[1], I.shape[0]))



**Utómunkálatok**

Utómunkálatok nélkül



Utómunkálattal határon kívüli fekete szín levágása



**Felhasználói leírás**

A programot a Python programozási nyelvben írtam meg, azon belül a Thonny fejlesztő környezetet használtam. A Thonny fejlesztő környezetet ingyenesen le lehet tölteni a <https://thonny.org/> oldalról. A külső könyvtárak közül 2 darabot használtam a Numpy és az OpenCV csomagokat. Ezeket a Thonny futása után a Tools menüpontban az Open system shell lehetőséget választva, a megjelenő ablakban lehet telepíteni, ha beleírjuk a **pip install opencv-contrib-python** parancssort. A parancssor után telepíti az OpenCV és Numpy csomagokat. A program futásához még telepíteni kell a 3.4.2.16 verziót. Ezt szintén az Tools menüpontban az Open system shell lehetőséget választva, a megjelenő ablakban a pip install opencv-hozzájár-python == 3.4.2.16 parancssorral kell begépelni majd az enter gombot lenyomni, és ezután települ. A program 2 darab képet tud összefűzni, de csak akkor ha van elég közös kulcspontjuk(átfedés). Képet a **cv2.imread("kep neve.jpg")** parancssal tudjuk a képet betölteni, majd ugyanígy a bal oldali képet. Fontos hogy a képet fordított sorrendben adjuk meg.

Kulcspontok meghatározásához az cv2.xfeatures2d.SIFT\_create()beépített függvényt használtam, de ezen kívül másik három OpenCV beépített függvényt is használni lehet:

cv2.xfeatures2d.SURF\_create()

cv2.BRISK\_create()

cv2.ORB\_create()

A párosított pontok szűrésénél 0.65 választottam, a lényege hogy a szám minél közelebb van a 0 – hoz annál több párosított pontokat szűr meg. Az így megmaradt párosított pont adatokat elmentem egy tömbbe. Ezután egy feltétellel megnézem a tömb hosszát és ha több vagy egyenlő mint az általam megadott érték (a programban 5) akkor homográfi mátrixot készítünk a *cv2.findHomography*() beépített függvény segítségével, ha kevesebb mint a megadott érték (5) akkor egy hibaüzenetet ír ki, hogy (Nem található elég közöskulcspont!) és a program nem is fut tovább. Elegendő kulcspontnál tovább fut és a jobb oldali képből elkészíti a homográfia képet és közös kulcspontok alapján összefűzi a két képet.

**Felhasznált irodalom**

<https://docs.opencv.org/trunk/dc/dc3/tutorial_py_matcher.html>

<https://towardsdatascience.com/image-stitching-using-opencv-817779c86a83>

<https://medium.com/pylessons/image-stitching-with-opencv-and-python-1ebd9e0a6d78>

<https://hu.wikibooks.org/wiki/Automatikus_panor%C3%A1mak%C3%A9p-k%C3%A9sz%C3%ADt%C3%A9s>