Szegedi Tudományegyetem

Informatikai Intézet

SZAKDOLGOZAT

Kis Krisztián

2020

Szegedi Tudományegyetem
Informatikai Intézet

Régiók többszörözésének észlelése a jellemzőpontok megfeleltetésének segítségével

Szakdolgozat

Készítette:
Kis Krisztián
Mérnökinformatikus BSc
szakos hallgató

Témavezető:
Dr. Németh Gábor
egyetemi adjunktus

Szeged
2020

FELADATKIÍRÁS

A dolgozatot meghirdető tanszék: Képfeldolgozás és Számítógépes Grafika Tanszék

A dolgozatot meghirdető oktató neve: Németh Gábor

A dolgozat címe: Régiók többszörözésének észlelése a jellemzőpontok megfeleltetésének segítségével

A dolgozat angol címe: Region Duplication Detection Using Image Feature Matching

Típusa: Szakdolgozat

Milyen szakos hallgatók számára: 1 fő, Programtervező informatikus Bsc, Mérnökinformatikus BSc, Gazdaságiinformatikus BSc, Info-Bionika, Informatika szakos hallgató számára

A feladat rövid leírása:

A digitális képhamisításban gyakori, hogy a kép egy adott részletét többszörösen, akár különböző méretben duplikálják a képre, amelynek célja, hogy bizonyos részleteket eltúlozzanak, például több ember vett részt egy tömegeseményen, vagy több kitűntetése van valakinek, esetleg több rakétát indítottak stb. Az ilyen képeken az ismétlődő részletek kiszűrhetők, ha jellemzőpontokat detektálunk a képen és leíróik segítségével megnézzük, hogy szerepel-e még a képen hasonló régió. A program Matlab kivételével tetszőleges programozási nyelven megvalósítható. Javasolt az OpenCV függvénykönyvtár használata.

Előismereti feltétel: nincs

Szakirodalom: X. Pan, S. Lyu: Region Duplication Detection Using Image Feature Matching, In IEEE Trans. on Information Forensics and Security, vol 5, No. 4, pp. 857-867, 2010, IEEE

TARTALMI ÖSSZEFOGLALÓ

- ***A téma megnevezése:***

  Régiók többszörözésének észlelése a jellemzőpontok megfeleltetésének segítségével

- ***A megadott feladat megfogalmazása:***

A szakdolgozatom célja egy olyan program létrehozása, ami képes detektálni a duplikált régiókat a képen, abban az esetben is, ha azt a duplikált régió el van forgatva, át van méretezve, el van torzítva vagy megváltozott a pixel intenzitása.

- ***A megoldási mód:***

A duplikált régiókat pixel korrelációval kapom, meg amihez szükség van az affin transzformációs mátrixra, amit a jellemző kulcspontok párosításának segítségével kapok meg.

- ***Alkalmazott eszközök, módszerek:***

A program Python nyelven készült az OpenCV függvénykönyvtár segítségével, emellett használtam a Numpy és a Matplotlib függvénykönyvtárakat is.

- ***Elért eredmények:***

A program sikeresen megtalálja a duplikált régiót módosítástól függetlenül és kiszínezi az eredeti és a másolt régió területét.

- ***Kulcsszavak:***

  Képhamisítás, Python, OpenCV, Numpy, Matplotlib

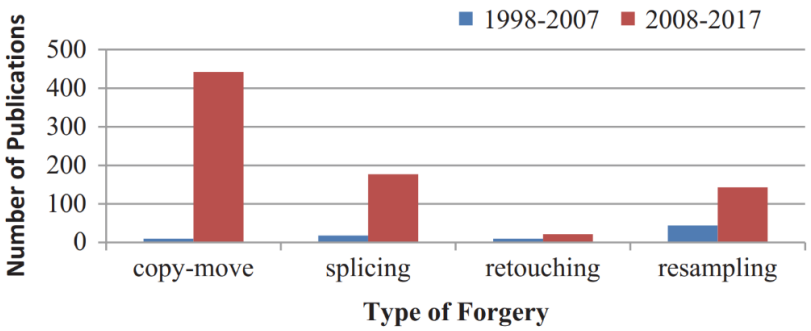
TARTALOMJEGYZÉK

BEVEZETÉS

A képhamisítás magába foglalja a kép átalakítását vagy annak módosítását különböző módszerek és technikák segítségével a kívánt eredmény elérése érdekében. Sokféle területen alkalmazzák a képmanipulálást különböző hatékonysággal. Minden nap találkozhatunk ezzel a jelenséggel, ha figyelmesek vagyunk, de sokszor lehet még így se vesszük észre a csalást annyira hitelesnek tűnik. Láthatunk, olyan példákat ahol művészi célból hoznak létre ilyen módon műalkotásokat, míg bizonyos esetekben etikátlan módon az emberek megtévesztése a cél.

Az alkalmazási terület és szándék függvényében fontos ezeknek a képeknek a felderítése és szűrése, hiszen az információ világában élünk ahol gyorsan terjednek a valós és az álhírek is. Sokszor használják a képhamisítást a politikában propaganda céljából ahol egy-egy ilyen képet is felhasználhatnak emberek lejáratására, megfélemlítésére, befolyásolásra. De ennél hétköznapibb területen is találkozhatunk hamisítással akár a sajtón keresztül vagy közösségi médián át, mivel számtalan szoftver elérhető az átlagember számára a digitális kép manipulálásra a belépő szintől a professzionális felhasználásig.

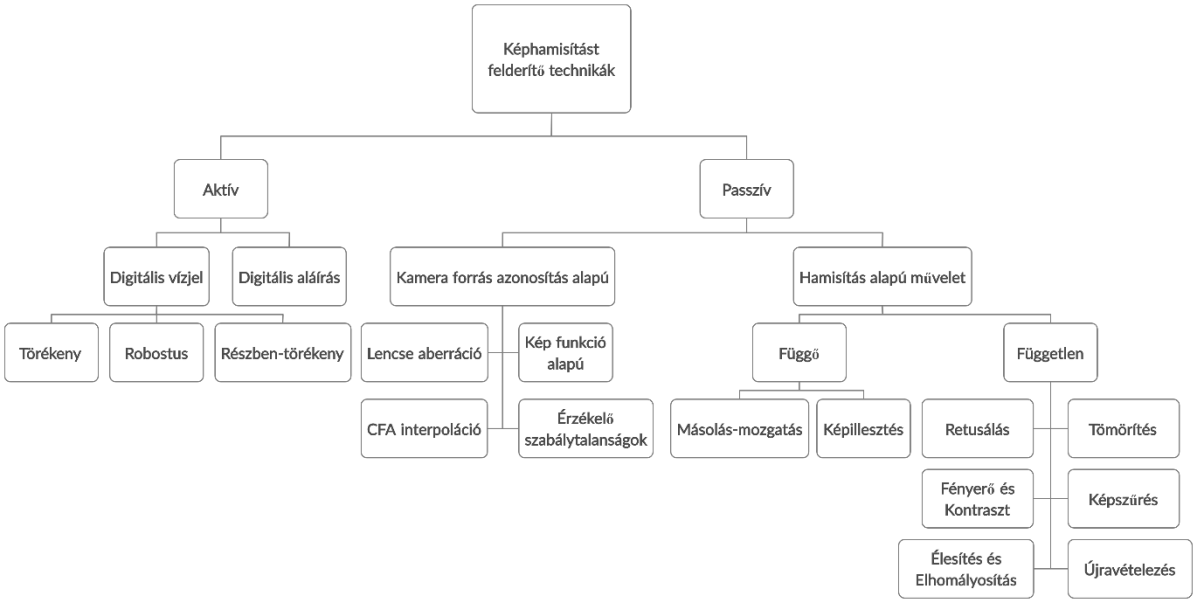
Épp ezért fontos, hogy legyenek, minél pontosabb módszereink melyeket program formájában megvalósíthatunk ezeknek a képeknek a detektálása érdekében. Számos módszert kidolgoztak ezeknek a képeknek az észlelésére. Jelentős mennyiségű kutatást [16] végeztek az elmúlt évtizedben a hamisítás észlelésének területén. Az alábbi oszlopdiagrammon (1.1 ábra) láthatunk négyféle képhamisítási észlelési technikát (másolás-mozgatás, képillesztés, retusálás, újravételezés) és a hozzájuk tartozó tanulmányok számát 1998 és 2017 között. Láthatjuk, hogy jelentős növekedést figyelhetünk meg az elmúlt évtizedben a másolás-mozgatás detektálásával foglalkozó tanulmányoknál és jelentős figyelmet kapott még a képillesztésdetektálása is. Azonban kevesebb figyelem jutott a retusálás detektálására ennek oka, pedig hogy ez a módszer a legkevesebbet használt az illegális tevékenységekre.



1.1 ábra: Képhamisítást felderítő technikákról szóló tanulmányok (2007-2017)

Forrás: [16]

Attól függően, hogy milyen módosítást szeretnénk felderíteni különböző módszereket kell használnunk. Hamisítás detektálási technikákat nagyjából két kategóriába sorolhatjuk; aktív és passzív (2. ábra).



1.2 ábra: Képhamisítást felderítő technikák

Aktív hamisításfelismerés technikáknál szükség van néhány előzetes információra a képről, amely esetleg beágyazódott a képbe a kép rögzítésekor vagy a kép megszerkesztése vagy annak későbbi szakaszai során. A digitális vízjel és digitális aláírás példái az aktív képhamisítás detektálási technikákra és ezek a megközelítések felhasználhatók a kép hitelességének tesztelésére a beágyazott információk alapján. Az alkalmazás alapján a digitális vízjel tovább kategorizálható törékeny, részben-törékeny és robosztus vízjelként. A gyakorlatban nagyon ritka, hogy a törvényszéki nyomozás céljából készített képek, mint az ujjlenyomatképek, a bűnügyi helyszíni képek, a bűnözők fényképei stb. tartalmazzák a vízjelet vagy az aláírást, ezért megállapítható, hogy az aktív hamisítás a detektálási technikák nem használhatók a digitális képek törvényszéki vizsgálatához.

Ezzel ellentétben a passzív észlelési technikáknál nincs szükség előzetes információra a képpel kapcsolatban. Ezek a technikák inkább a kép jellemzőit kivonva azonosítják a manipulációkat a módosítás típusa vagy a képet készítő eszköz azonosítása alapján. Passzív hamisításfelismerés tovább kategorizálható függő és független kategóriákra. Függő hamisítás esetén bármelyik manipuláció elvégezhető ugyanazon a képen belül a kép bizonyos területének másolásával és beillesztésével (másolás-mozgatás) vagy egynél több kép kombinálható (képillesztés), használatával hogy meggyőző legyen. Másrészt a független hamisítás az a

hamisítás, amelyben egyesek ugyanazon kép tulajdonságait manipulálják. Példa a független hamisításra magában foglalja az újramintavételt, retusálást, képforgatást, méretezést, átméretezést, zaj hozzáadását elmosódás, képtömörítés stb. A képpel kapcsolatos előzetes ismeretek bevonása nélkül a passzív hamisítás gyakoribb a hétköznapokban.

1. IRODALMI ÁTTEKINTÉS

1.1. Kamera alapú és más hamisítási észlelési technikák

Az elmúlt évtizedben számos kutatást végeztek a képhamisítás észlelésével kapcsolatban. Tran Van Lanh et al. [2] különféle technikákat tárgyalt a képhamisítás észlelésére a kamera alapján. Meggyőző észrevételt tettek, hogy a kamera-alapú technikák megbízhatóságukat tekintve jobbak, mint más hamisításfelismerési technikák.

Hany Farid [3] a képhamisító eszközöket öt csoportba sorolta, pixel alapú technikák, formátum alapú technikák, kamera-alapú technikák, fizikai alapú technikák és geometriai alapúak. Minden módszert részletesen kidolgozva taglal a tanulmányában.

Nemrégiben Nor Bakiah Abd Warif [4] áttekintette a másolás-mozgatás hamisításának észlelési technikáit. Főként két osztályba sorolták a másolás-mozgatás hamisítás észlelését: blokk alapú és kulcspont alapú megközelítés.

1.2. Hamisítási észlelési technikák

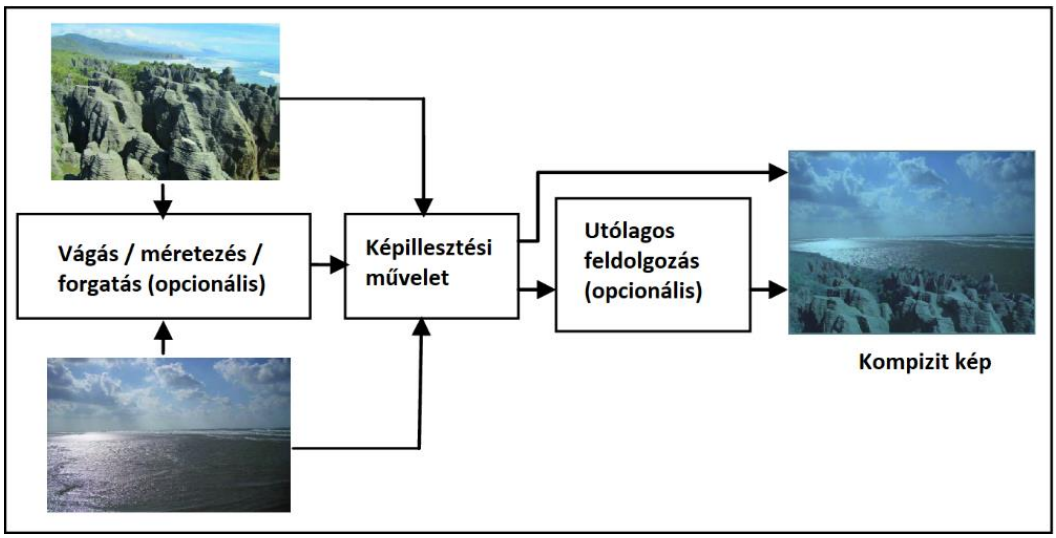
A digitális képek tekintetében, hamisítás alatt értünk minden manipulációt vagy módosítást a képen mely megváltoztatja szemantikai jelentését illegális célokból vagy annak megváltoztatását jogtalan célokra. Hatalmas mennyiségű kép készül a digitális képi törvényszéki vizsgálat előtt függetlenül attól, hogy a kép hiteles-e (a kép szemantikai értelmében nincs változás), vagy megváltoztatták-e. Mivel a képszerkesztő szoftverek szinte mindenki számára elérhetőek valamilyen formában, bárki megteheti, hogy manipulál egy képet, és rosszindulatú szándékkal felhasználja azt. A következőkben négy digitális képhamisítást detektáló technikát mutatok be, még pedig a képillesztést, újramintavételezést, retusálást és a dolgozatom fő témáját másolás-mozgatást.

1.2.1. Képillesztés detektálása

A digitális kép manipuláció általános formája két vagy több kép digitális összekapcsolása, összeillesztése egyetlen kompozícióba úgy, hogy az egyik kép részletét kivágjuk és bele illesztjük azt a másik képbe. Az összeillesztett kép valósághűbbé tétele érdekében utómunkálatok (méretezés, kivágás, retusálás, forgatás stb.) alkalmazhatók az egyes összetevőkön, továbbá az illesztési művelet elvégzése után ismét alkalmazható az utólagos feldolgozás az észrevehető hatások elrejtése érdekében. Bár bizonyos esetekben szakértők már ránézésre megtudják állapítani a képről, hogy ezzel a módszerrel hamisították. A tapasztalt

hamisító azonban olyan elegánssá teheti az összetett képet, hogy szinte képtelenség bármit is mondani a kép valódiságáról, pusztán a kép megnézésével.

Azonban Hany Farid [5] tanulmányában megmutatja, hogy az illesztés megzavarja a magasabb rendű Fourier statisztikákat, amelyek később felhasználhatók az illesztés kimutatásánál. Bár szabad szemmel ezek nem észrevehetők, de mindegy mennyire, elegánsan és tökéletesen van összemosva, korrigálva a háttérben ezekkel az adatokkal kimutatható a hamisítás. Xuanjing Shen [6] tanulmányában ennél újabb metódussal képesek voltak 98%-os pontossággal azonosítani a képillesztést.



2.2.1 ábra: Képillesztés

Forrás: [16]

1.2.2. Kép újramintavételezés detektálása

Az újravételezés matematikai technika a képfelbontás megváltoztatásához, főleg abból a célból, hogy növelje a kép méretét például transzparensek nyomtatásához stb., vagy, hogy csökkentse a kép méretét az e-mail és a weboldal használatához. Általánosságban elmondható, hogy szinte mindenfajta digitális képhamisítás tartalmaz méretezési, elforgatási vagy ferdeségi műveleteket a kép manipulálására. Ezekben a műveletekben elkerülhetetlen az újramintavétel és az interpolációs folyamatok használata. Ennélfogva nyomon követhető a képhamisítás tüneteinek észlelése a képen történő újramintavétlezésénél. Ezáltal a kép hamisításának észlelése a kép újramintavételének tüneteinek felkutatásával lehetséges. Számos tanulmány született ezzel a detektálással kapcsolatban is, közülük megemlítenék kettőt.

Popescu és Farid [7] egy módszert javasolt a digitális hamisítások feltárására az újramintavétel nyomainak kimutatásával. A hamisítás vak észlelésében nem áll rendelkezésre előzetes információ a képről, például arról, hogy melyik utólagos feldolgozási technikát

alkalmazták, melyik interpolációt alkalmazták a képen vagy a kép egy részének újramintavételezésére. Az újramintavételezés nyomainak azonosításához azonban az interpolációs részletek az újramintavételezés detektálásának alapjelzői lehetnek. Ezért a szerzők kihasználják az elvárás / maximalizálás algoritmusát annak megállapítására, hogy a jelet újra mintázták-e. Két modellt fejlesztettek ki, az egyik azokra a mintákra vonatkozik, amelyeknél korreláció van a szomszédjaikkal, a második modell pedig azoknak a mintáknak felel meg, amelyek nincsenek korreláció. Módszerük hatékony a lineáris vagy köbös interpoláció előjelének feltárására. Más bonyolultabb, nemlineáris interpolációs technikákat azonban nem sikerül kimutatni. Ezzel a módszerrel 80%-os pontossággal tudták detektálni a hamisítást és működik GIF formátummal is.

A másik tanulmány, amit kiemelnék Mahdian és Saic [8] munkája, ahol az általuk javasolt algoritmussal az interpoláció és az újramintavétel detektálására 100%-os pontossággal sikerült észlelniük. A módszer származtatott operátor és radon transzformáción alapult. Módszerük hatékony volt a méretezés, forgatás, ferde transzformációk nyomainak detektálására.

1.2.3. Kép retusálás detektálása

A retusálás meghatározható „kép csiszolása, finomításaként”. A retusálás általában utal a kép felületének utólagos javítására. A kontraszt javítása széles körben elterjedt, használt technika a nyilvánvaló vizuális nyomok eltávolítására a hamisított képről utómunkálatként. A retusálás nagyobb mértékű bevonása azonban látható a szórakoztató médiában, a magazinok borítóin stb., ahol a retusálást nem rosszindulatúan használják. A kontrasztjavító műveletek egyenértékűek a pixelérték-hozzárendelésekkel, amelyek bevezetnek néhány statisztikai nyomot. Ezért a retusálás kihasználható eszköz a képhamisítás észleléséhez.

Matthew Stamm és K.J. Ray Liu [9] módszerre a kép kontrasztjavításának kimutatására a szürke érték hisztogramja alapján. Kidolgozták a változatlan kép hisztogramjának modelljét, majd ezt a modellt kihasználva detektálták a manipulált artefaktumokat a képen. Állításuk szerint az algoritmus detektálási pontossága körülbelül 99%.

Gang Cao, Yao Zhao és Rongrong Ni [10] kifejlesztett egy technikát a digitális képek éles változásainak észlelésére. A szerzők megmérték a szürke hisztogram gradiens aberrációját, amely a kép telítetlen fénysűrűségű régióiból származik, és kihasználják az élesítés manipulációjának nyomait. Gang Cao [11] egy új módszert javasolt az éles maszkírozás élesítésének észlelésére, amely az oldal-sík élek körül előforduló túllövés tárgyát képezte. Kísérleti tanulmány szerint a szerzők arról számolnak be, hogy módszerük pontos a kisméretű

képek élesítésének kimutatására, még akkor is, ha JPEG utáni tömörítést és zajzavarokat alkalmaznak. Ugyan ezek a szerzők tovább magyaráztak egy módszert a digitális képek kontrasztjavításának kimutatására. Ezúttal hasznosították a hisztogram csúcs / rés artifaktum leírókat a korábban tömörített JPEG-formátumú képeken alkalmazott globális kontrasztnövelés detektálására. Javasolt módszerük hatékonyan azonosítja a hamisítást, amikor a kontrasztnövelést alkalmazzák a manipuláció utolsó lépéseként. A módszer azonban nem képes felismerni a hamisítást, ha a kép erősen tömörített.

1.2.4. Képrészlet Másolás-mozgatás detektálása

A másolás-mozgatás a legnépszerűbb és legelterjedtebb képmeghamisítási technika és könnyedén kivitelezhető. A technika magában foglalja a képből egy régió másolását és ugyanezt áthelyezi egy másik régióijába a képen belül. Mivel a másolt rész ugyan abból a képből származik ezért az intenzitás-tartomány és a szín kompatibilitás megmarad. Egyszerűségéből fakadóan ez az egyik leggyakrabban használt hamisítási mód. Éppen ezért erről a módszerről készült a legtöbb tanulmány is. Az alábbi példán láthatjuk a módszer szemléltetését.

Eredeti kép Hamisított kép

2.2.2 ábra: Másolás-mozgatás

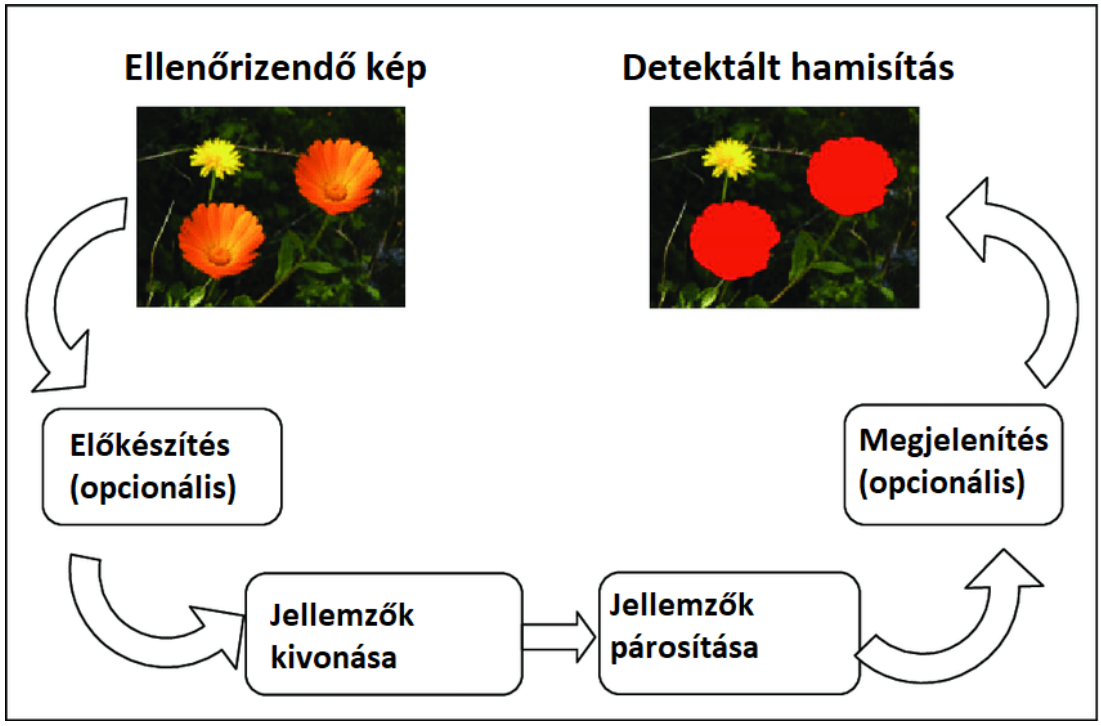
A másolt területeken végezhetnek különböző módosításokat, is mint például forgatás, nagyítás-kicsinyítés, szabad forma, illumináció módosítás vagy ezeknek együttes kombinációja is lehetséges akár. A másolás-mozgatásnál mellé is gyakran használnak egyéb utalógos feldolgozást, mint például homályosítás, hogy jobban beleolvadjon a másolt terület a környezetébe.

Ewerton Silva és társai [12] kifejlesztettek egy módszert a másolás-mozgatás észlelésére, ami a multiskála elemzést és szavazási folyamatot veszi alapul digitális képek

esetében. Ez a metódus kulcspontokat detektál a Speed-Up Robust Features (SURF) technika segítségével majd a Nearest Neighbor Distance Ratiot (NNDR) –t használja a kulcspontok párosításához. Az illusztrált módszer működik forgatás, átméretezéssel vagy mindkettő kombinációjával. Sajnos bizonyos esetekben nem talál elegendő mennyiségű kulcspontot kicsi vagy homogén területeken.

A Gabor-szűrés alapú megközelítés másolás-mozgatás hamisításának felderítésére Lee [13] mutatta be, amely magába foglalja a lexikográfiai rendezést, mint jellemző pontok párosítási technikát. A módszer időbeli összetettsége $(O(PNlogN) + O(2JPN)$ volt. Ardizzone [17] tanulmányában kifejt egy másolás-mozgatás hamisítás észlelési megközelítést, amely háromszögek egyezésén alapult, átlagos csúcsleírók alkalmazásával. Ez a megközelítés jobb teljesítményt mutat komplex jelenetek esetén; azonban sok hamis egyezés fordul elő rendszeres háttérrel.

Méretezett ORB alapján másolás-mozgatás hamisítás észlelési technikát javasolt Zhu [14]. Technikájuk először létrehoz egy Gauss-skála teret, majd kivonja a FAST kulcspontokat és az ORB jellemzőket minden skála térben. Továbbá a technika RANSAC algoritmust alkalmaz a hamisan egyező kulcspontok eltávolítására. A kísérleti eredmények azt mutatják, hogy a technika hatékony a geometriai transzformációhoz. Nagy felbontású képek esetén azonban a megközelítés időigényes.



2.2.3 ábra: Másolás-mozgatás detektálás folyamata

Forrás: [16]

2. HASZNÁLT MÓDSZER ÉS MEGVALÓSÍTÁS

Szakdolgozatomban Xunyu és Siwei [1] tanulmányát vettem alapul a megvalósításhoz. Tanulmányukban olyan technikát mutattak be a régió duplikációk detektálására, a SIFT jellemző kulcspont párok egyezése közötti transzformáció megbecsülése segítségével, ami invariáns a képfunkció-illesztés miatt bekövetkező torzulásokra. Az algoritmus átlagosan 99,08%-os detektálási pontosságot eredményez.

2.1. Jellemzők és kulcspontok megtalálása a képen

Az első lépése a metódusnak, hogy beimportáljuk a képet, amit vizsgálni szeretnénk, a kép lehet PNG vagy akár JPEG kiterjesztésű. Az RGB képet, ami 3 csatornát tartalmaz, átalakítjuk szürkeárnyalatos képpé, ami csak 1 csatornát tartalmaz egyszerű színtérkonverzióval, hogy könnyebben tudjuk vizsgálni a pixel intenzitási tartományt később.

RGB kép Szürkeárnyalatos kép

3.1.1 ábra

Python forráskód:

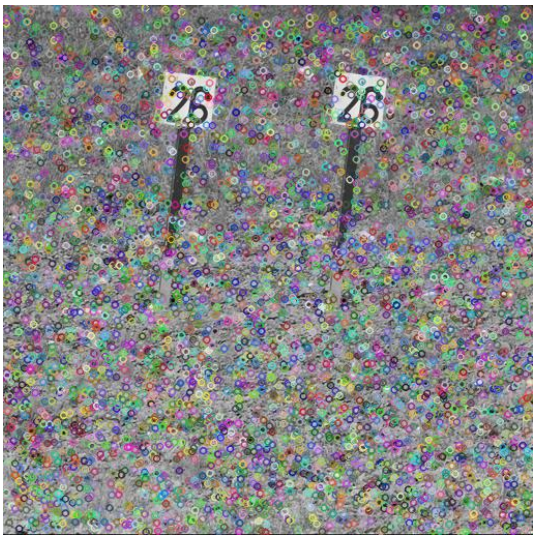
```
# 1) Read in the image
img = cv2.imread("exemple.png")

# 2) RGB image convert to grayscale
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

A beolvasás az imread() paranccsal történik, akár itt egyből beolvashatnánk már szürkeárnyalatos képként is, de ha esetleg szükség lenne az eredeti kép értékeire, tanácsosabb konvertálással. A konvertálás a szintérben a ctvColor() paranccsal történik, ahol szükség van legalább a beolvasott és átkonvertálni kívánt kép nevére és az átalakítás módjára.

Második lépésben szükségünk van a kulcspontok megtalálására. A kulcspontok különböző információkat hordozó helyek a kép tartalmáról. Mindegyik kulcspontot egy jellemző leíró vektor jellemzi, amely a megfelelő kulcspont helyi szomszédságában összegyűjtött képstatisztikákból áll. A jó kulcspontoknak és jellemzőknek a kép különálló helyeit kell képviselniük, hatékonyaknak kell lenniük a számításhoz, és robusztusak a helyi geometriai torzításokkal, a megvilágítás variációival, a zajjal és más romlásokkal szemben. A kulcspontok megtalálásra többféle módszer van, ezek természetesen különbözőképpen működnek és némelyikkel, több némelyikkel kevesebb kulcspontot találhatunk egy képen. A detektorok közül néhány ismert: Harris sarokdetektor, Shi-Thomas sarokdetektor, ORB detektor, SURF detektor, AKAZE detektor, BRIEF detektor és végül SIFT jellemzőpontdetektor.

Ebben a metódusban a SIFT-et fogom használni a jellemző kulcspontok megtalálására. A SIFT kulcspontokat olyan helyek keresésével találjuk meg, amelyek stabil helyi extrémák a skála térben. Minden kulcspontnál egy 128 dimenziós jellemző vektor jön létre a szomszédságában található helyi gradiensek hisztogramjaiból. Annak érdekében, hogy a jellemző vektor változatlan maradjon a forgatásra és méretezésre, a környezete méretét a kulcspont domináns skálája határozza meg, és az összes benne lévő gradiens igazodik a kulcspont domináns orientációjához. A kapott hisztogramokat egységnyi hosszúságra normalizáljuk, ami a jellemzővektort invariánssá teszi a helyi megvilágítási változásokra.

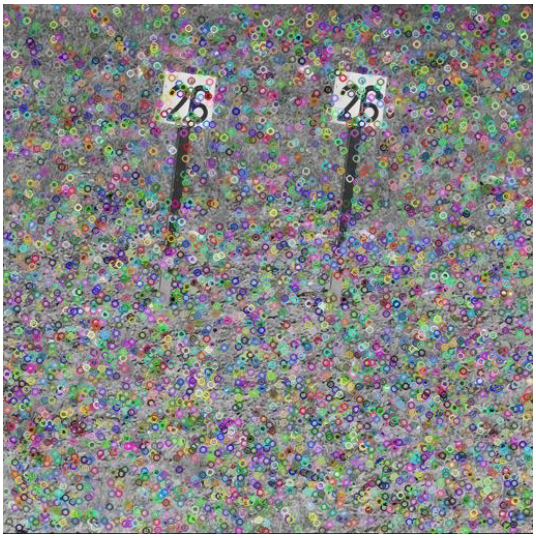
Szürkeárnyalatos kép



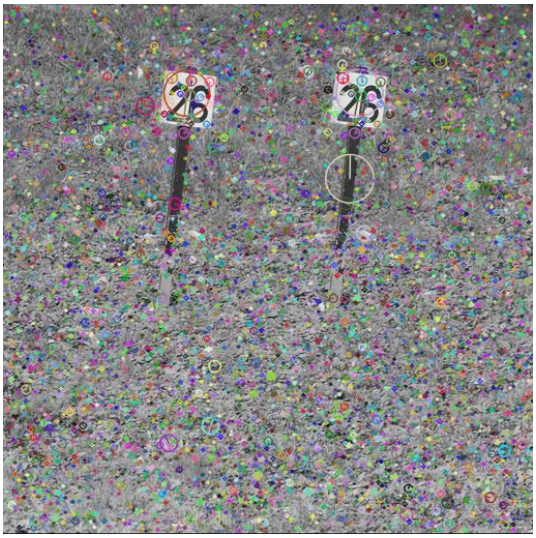
Jellemző kulcspontok a képen

3.1.2 ábra

Mint a fenti példán keresztül láthatjuk a kulcspontokat ki is tudjuk rajzolni, minden színes kis kör egy kulcspontot jelent. Viszont vizuálisan itt nem látjuk a jellemző kulcspontokhoz tartozó jellemző vektorokat. Azonban ezt is tudjuk ábrázolni a képen, méghozzá a jellemző vektorok nagyságát és irányát is megjeleníttethetjük vizuálisan.



Jellemző kulcspontok a képen
(vektorok iránya és nagysága nékül)



Jellemző kulcspontok a képen
(vektorok irányával és nagyságával)

3.1.3 ábra

Láthatjuk, hogy a második képen a kulcspontok másképp néznek ki bizonyos körök, nagyobbak, mint a többi és látható benne egy vonal is, ez a jellemző vektor nagysága és iránya.

Ezen a képen meglehetősen sok kulcspontot sikerült találni azonban ez nem minden képnél alakul így, ha nagyon homogén a kép egy része, mint például az ég, amin nincs sok felhő vagy egy nyugodt víz felszín akár.

3.1.4 ábra

Mint azt megfigyelhetjük ezen a képen is, az égen és a víz nagyrészén szinte alig találtunk jellemző kulcspontokat, ellenben a hídon, a fákon és a vízszakaszon ahol megtörik a hullám megtalálhatóak a kulcspontjaik kb. 90%-a.

Python forráskód:

```
# 3) Initiate SIFT detector (nfeatures = "keypoints number")
sift = cv2.SIFT_create(nfeatures=100000, nOctaveLayers=3, sigma=1.6)

# 4) Find the keypoints and descriptors with SIFT
keypoints, descriptors = sift.detectAndCompute(img, None)
```

A kódban először szükségünk van a SIFT jellemzőpontdetektor inicializálására, itt adhatjuk meg a detektor különböző paramétereit, mint például az nfeatures–t, ami a legjobb jellemző pontok száma. A SIFT rangsorolja a jellemző pontokat egy érték alapján, amit a lokális kontraszt alapján számol. Második paraméterünk nOctaveLayers ami a rétegek száma az egyes oktávokban. A Gauss sigma a bemeneti képre #0 oktávban alkalmazott bemeneti képre vonatkozik. Ezen kívül adhatunk neki más paramétereket is, minden paraméternek van egy adott alap értéke, ha nem változtatjuk meg automatikus az alapértékek lesznek beállítva.

Ezután a SIFT detektort alkalmazzuk a szürkeárnyalatos képünkre a detectAndCompute() függvénnyel, ami detektálja a kulcspontokat és a kulcspontokhoz tartozó jellemző vektorokat. Itt megtudnánk adni neki egy maszkot is paraméterként. A keypoints egy lista melyben a kulcspontokat találjuk és a descriptors pedig egy numpy tömb, ami a kulcspontokhoz tartozó 128 dimenziós tömböket tárolja.

A fenti kódnak nincs semmi vizuális kimenete, mint ahogy a példáimon látszik, erre a későbbiekben nem is lesz szükségünk, de nagyon könnyen megtudjuk jeleníteni a kulcspontokat az alábbi kód segítségével ahol a drawKeypoins() függvényt használjuk, egyik flaget használva rajzolja ki a vektor nagyságát és irányát is.

Python forráskód:

```
# 4.1) Draw out keypoints, FLAG is for vector size and direction
(optional)
img = cv2.drawKeypoints(img_gray, keypoints, img,
flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

# 4.2) Show image or save out in a file (optional)
cv2.imshow("result", img)
cv2.imwrite('keypoints.png', img)
```

Az imshow() függvény megjeleníti a képet egy új ablakban, az imwrite() függvény kimenti a képet egy file-ba.

2.2. Jellemzőkulcspontok párosítása

Következő lépésként a kapott jellemző kulcspontokat össze kell párosítanunk egymással hogy, egyezést találjunk. A kulcspontokat a vektorjaik alapján hasonlítjuk össze. A tanulmányban a best-bin-first algoritmust használják, ami egy keresési algoritmus, amit arra terveztek, hogy hatékonyan találjon megoldást a legközelebbi szomszéd (NN) keresési problémára nagyon nagy dimenziós terekben. Az algoritmus egy kd-fa keresési algoritmus azon változatán alapszik, amely lehetővé teszi a nagyobb dimenziós terek indexelését. A best-bin-first egy hozzávetőleges algoritmus, amely a legközelebbi szomszédot adja meg a lekérdezések nagy töredékében, máskülönben pedig egy nagyon közeli szomszédot.

A kulcspontok párosítását a programomban Brute Force matcher-rel oldottam meg, majd létrehoztam egy listát, ami a K legközelebbi szomszédokat tárolja (KNN).

Python forráskód:

```
# 5) Create Brute-Force Matcher,
bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=False)

# 5.1) Brute-Force with KNN, K the number of best matches
matches = bf.knnMatch(descriptors, descriptors, k=3)
```

A tanulmányban az euklideszi távolságot figyelik, két vektor között ezért használom a NORM_L2 típust a Brute Force matcher-nél, a crossCheck egy alternatívája lehet az arány tesztelés (ratio test) helyett, de ha ezt bekapcsoljuk párokat vesztünk, ami ebben az esetben nem megfelelő számunkra mivel saját arány tesztet használok a tanulmány alapján. A kódban a $K = 3$, tehát a három legjobb párt tárolja el, azért tárolok el három párt, mert a kapott kulcspontokat és a jellemző vektoriakat önmagukkal párosítja össze a párosító és mivel a K legjobb legközelebbi szomszédot nézi a kulcspontnak önmaga is a párja lesz. Ezeket a rossz párokat ki kell szűrni. Könnyen megtudjuk tenni ezt a lépést, ha egy egyszerű for ciklussal végig megyünk az összes talált páron a listában, ezek a párok hármasával szerepelnek, párokat összehasonlítjuk egymással egy if feltételvizsgálattal és csak azokat a párokat tartjuk, meg amik nem önmagukkal vannak összepárosítva. Így marad, minden ponthoz két darab pár ezeket hozzáadjuk egy új listához, aminek a neve better_matches. Minden pár két kulcsponthoz tartozó jellemző vektort tartalmaz train-t illetve a query-t, ami a kiinduló és a végpontot jelöli.

Python forráskód:

```
# 5.2) Removing self matching points
better_matches = []
for a, b, c in matches:
    if a.trainIdx == a.queryIdx:
        better_matches.append([b, c])
    elif b.trainIdx == b.queryIdx:
        better_matches.append([a, c])
    elif c.trainIdx == c.queryIdx:
        better_matches.append([a, b])
```

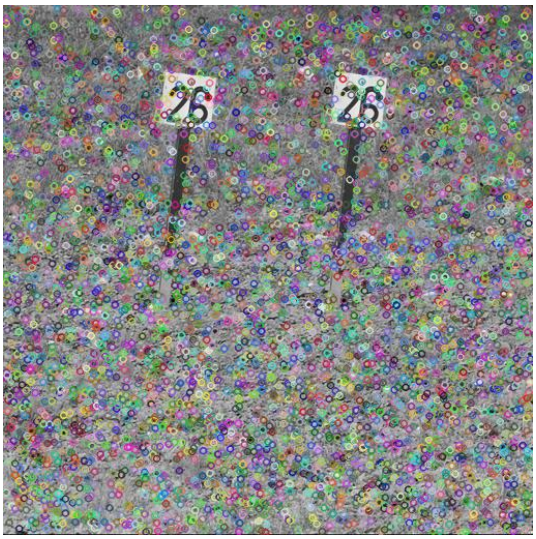
Mint feljebb említettem saját arány tesztet (ratio test) használok a párok további szűrésére, az arány tesztet láthatjuk, D. Lowe 2004-es tanulmányában ahol bemutatja a SIFT algoritmust. A tanulmány alapján a küszöb értékünk 0,5 lesz (ez az alapértelmezett érték D. Lowe tanulmányában), ez jó kompromisszum az illesztési pontosság és a kiugró arányok között.

A tesztnél megnézzük, hogy egy kulcsponthoz tartozó két pár közül melyik a jobb találat a távolságuk alapján. Előbbihez hasonlóan szükségünk van egy for ciklusra, ami végig megy az összes páron a listában, és a két megmaradt párt egy feltételben összehasonlítja a küszöb érték segítségével. A figyelembe vett kulcspont két legközelebbi pár közötti távolság arány kiszámításra kerül, és akkor jó, ha ez az érték küszöbérték alatt van. Ez az arány valóban lehetővé teszi a kétértelmű párok (a két legközelebbi szomszéd közötti távolság aránya közel egy) és a jól diszkriminált párok megkülönböztetését. A legjobb párt hozzáadjuk egy új listához, aminek a neve good_matches.

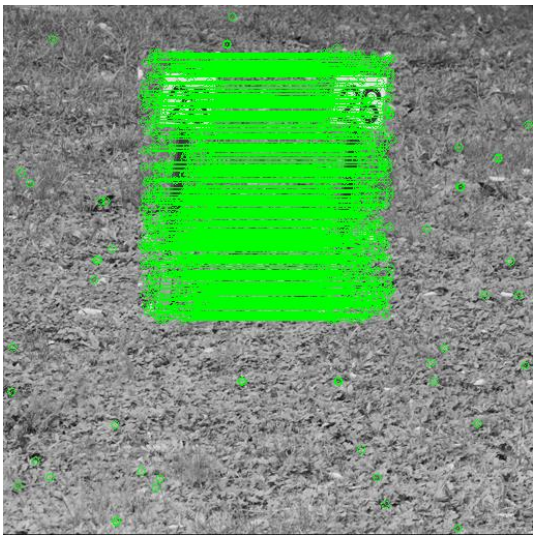
Python forráskód:

```
# 5.3) Apply ratio test
ratio_thresh = 0.5
good_matches = []
for m, n in better_matches:
    if m.distance < ratio_thresh * n.distance:
        good_matches.append(m)
```

Mint láthatjuk a következő képeken jelentősen lecsökkentettük a kulcspontok számát azokra, amelyek párban állnak egymással, azonban mint látszik, vannak olyan párjaink is, amik nem a másolt régióhoz tartoznak. A későbbiekben ezeket a párokat tovább fogjuk szűrni a RANSAC algoritmussal, hogy megkapjuk a végleges párok listáját, amik csak a másolt régióhoz tartoznak.



Jellemző kulcspontok a képen
(vektorok iránya és nagysága nékül)



Jellemző kulcspontpárok a képen
(hamis párok is szerepelnek)

3.2.1 ábra

Python forráskód:

```
# 5.3.1) Draw Lines and circles on the image (optional)
# Grayscale image convert to RGB image
img_RGB = cv2.cvtColor(img_gray, cv2.COLOR_GRAY2RGB)

list_point1 = []
list_point2 = []
for j in good_matches:

    # Get the matching keypoints for each of the images
    point1 = j.trainIdx
    point2 = j.queryIdx

    # Get the coordinates, x - columns, y - rows
    (x1, y1) = keypoints[point1].pt
    (x2, y2) = keypoints[point2].pt

    # Append to each list
    list_point1.append((int(x1), int(y1)))
    list_point2.append((int(x2), int(y2)))

    # Draw a small circle at both co-ordinates: radius 4, colour green,
thickness = 1
    # copy keypoints circles
    cv2.circle(img_RGB, (int(x1), int(y1)), 4, (0, 255, 0), 1)
    # original keypoints circles
    cv2.circle(img_RGB, (int(x2), int(y2)), 4, (0, 255, 0), 1)

    # Draw a line in between the two points, thickness = 1, colour green
    img3 = cv2.line(img_RGB, (int(x1), int(y1)), (int(x2), int(y2)), (0,
255, 0), 1)
```

A fenti kód nem része a végleges program kódnak csak a könnyebb szemléltetést szolgálja teljesen opcionális. A párok kirajzolásához szükségünk van a párok kezdő (train) és végpontjára (query) ezek ugye a kulcspontjaink, kulcspontoknak van egy (x, y) koordinátája erre szükségünk van a kirajzoláshoz. Mivel Az OpenCV-ben csak olyan kirajzolási módszerek vannak ahol mindenképpen kötelező két kép megadása, saját módszert kell rá írni, ha egy képen belül akarjuk ezt megtenni. Először is a képünket vissza kell konvertálni szürkeárnyalatos képből RGB-be színtér konverzióval, erre azért van szükség, hogy a körök és vonalak, amiket kirajzolunk, látványosak legyenek, és jól látszanak. A kulcspontokból kinyert (x, y) koordinátákat ezután hozzáadjuk két listához egyik listába a kezdő (train), másikba a végpontok (query) koordináták vannak. Ezt követően egyszerű dolgunk van a circle metódussal bekarikázzuk, a kulcspontjainkat majd a line metódussal összekötjük azokat. Mind a két metódusnak ugyan azokra a paraméterekre van szükségük, adhatunk nekik vastagságot, színt és a circle esetében sugarat.

2.3. Az Affin transzformáció megbízható becslése egyező kulcspontok között

Ezután a feltételezett kulcspont-egyezés alapján megbecsüljük a duplikált régiók lehetséges geometriai torzulásait. Az olyan transzformációk általánosításához, mint a forgatás, méretezés és nyírás, amelyeket a legtöbb fotószerkesztő szoftver támogat, a torzítást a pixelkoordináták affin transzformációjaként modellezzük. Adott két egymásnak megfeleltethető pixel pozíció, egy a régióból és a duplikátuma, mint $x = (x, y)^T$ és $\tilde{x} = (\tilde{x}, \tilde{y})^T$, köztük lévő kapcsolatot egy 2×2 –es mátrix T és eltolás vektor x_0, mint $\tilde{x} = Tx + x_0$.

$$\begin{pmatrix} \tilde{x} \\ \tilde{y} \end{pmatrix} = \begin{pmatrix} t_{11}\ t_{12} \\ t_{21}\ t_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$$

A transzformációs paraméterek $(t_{11}, t_{12}, t_{21}, t_{22}, x_0, y_0)$ egyedi megoldásának megszerzéséhez szükségünk van legalább három pár kulcspontra, amelyek nem kollineárisak. Feltudjuk használni a vélt SIFT kulcspont egyezéseket az affin transzformáció paramétereinek számításához, de pontatlan eredményt kapnánk tekintettel a nagyszámú rossz kulcspontpárok miatt. Ahhoz hogy kitudjuk szűrni a rossz kulcspont párokat és megkapjuk a pontos affin transzformációs paramétereket alkalmazzuk a széleskörben elterjedt robusztus becslési módszert, amelyet Random Sample Consensus (RANSAC) algoritmusnak nevezünk. A RANSAC algoritmus nagy pontossággal képes megbecsülni a modell paramétereit akkor is, ha jelentős számú nem egyező pár van jelen.

A RANSAC algoritmus olyan tanulási technika, amely a modell paramétereinek véletlenszerű mintavételezéssel történő megbecsülésére szolgál. Adott adatkészlet, amelynek adatelemei mind beleillő érték (inliers), mind kiugró értékek (outliers) tartalmaznak, a RANSAC a szavazási sémát használja az optimális illesztési eredmény megtalálásához. Az adatkészlet adatelemeit egy vagy több modell szavazására használják. Ennek a szavazási sémának a megvalósítása két feltételezésen alapul: a zajos tulajdonságok nem fognak következetesen egyetlen modellre sem szavazni (kevés kiugró érték), és elegendő tulajdonság van ahhoz, hogy egy jó modellben megállapodjanak (kevés hiányzó adat).

A RANSAC két lépést ismétel N iteráción keresztül, első lépés véletlenszerűen kiválaszt három vagy több párt, amik nem kollineárisak a SIFT kulcspontpárok közül majd megbecsüljük az affin transzformációs mátrixot és az eltolási vektort rájuk, minimalizálva a legkisebb négyzetek objektív függvényt. Második lépésben felhasználva a becsült transzformációs mátrixot és az eltolás vektort, osztályozzuk a SIFT kulcspont párokat beleillő érték (inliers) és kiugró értékek (outliers) közé. A RANSAC algoritmus a becsült transzformációs paraméterekkel tér vissza, amelyek a legtöbb beleillő értékeket (inliers) eredményezik.

A megvalósítása a következőképpen néz ki, szükségünk van egy feltételre, hogy megvizsgáljuk, legyen legalább három SIFT kulcspont párunk, mint ahogy azt már említettem korábban. Ha ez a feltétel teljesül szükségünk van két listára, amiben letaroljuk a kulcspont párjaink kezdő és végpontjainak koordinátait, ezek lesznek a src_pts és a dst_pts úgy, mint source points és destination points. Következő lépésben használjuk az estimateAffine2D() függvényt, melynek szüksége van a kezdő és végpontok koordinátáira, a módszer megadására, ami esetünkben a RANSAC lesz, $N = 100$ iterációt fogunk venni és $\beta = 3$, ami pedig a beleillő értékek (inliers) maximális távolság a RANSAC-ba. És végül megbízhatósági szint, 0 és 1 között a becsült transzformációhoz. Bármi, ami 0,95 és 0,99 között van, általában elég jó. Az 1-hez közeli értékek jelentősen lelassíthatják a becslést. A 0,8-0,9 alatti értékek hibásan becsült transzformáció eredményezhetnek.

Az beleillő értékeket (inliers) hozzá kell ezután adnunk egy listához, hogy tovább tudjuk szűrni a SIFT kulcspontjainkat, természetesen a kapott affin transzformáció mátrix már megbízhatóan van megbecsülve, ez a lépés szintén a szemléltetéshez lesz szükséges.

Python forráskód:

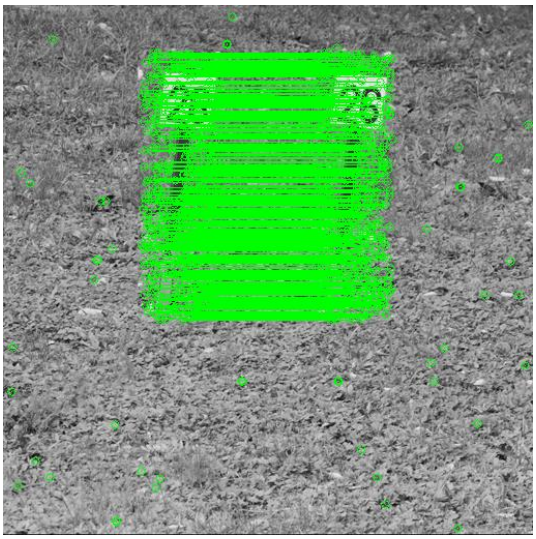
```
# 5.4) Affine Transform with RANSAC
MIN_MATCH_COUNT = 3
if len(good_matches) > MIN_MATCH_COUNT:
    src_pts = np.float32([keypoints[m.trainIdx].pt for m in
good_matches])
    dst_pts = np.float32([keypoints[m.queryIdx].pt for m in
good_matches])

    retval, inliers = cv2.estimateAffine2D(src_pts, dst_pts,
method=cv2.RANSAC, ransacReprojThreshold=3, maxIters=100,
                                           confidence=0.99)

    matchesMask = inliers.ravel().tolist()
```

A végleges párokat kiszeretnénk gyűjteni egy új listába a beleillő értékek (inliers) segítéségével. Alábbi módon tehetjük ezt meg, végig megyünk egy for ciklus segítségével a good_matches listán majd azzal, feltétellel vizsgáljuk, hogy a soron következő matchesMask listában, amit az inliers-ből hoztunk létre 1 vagy 0 szerepel, az 1 jelöli a beleillő értéket a 0 a kiugró értéket. Ahol 1-es szerepel, azt a párt hozzáadjuk a final_matches listához ezzel megkapva a végleges párokat.

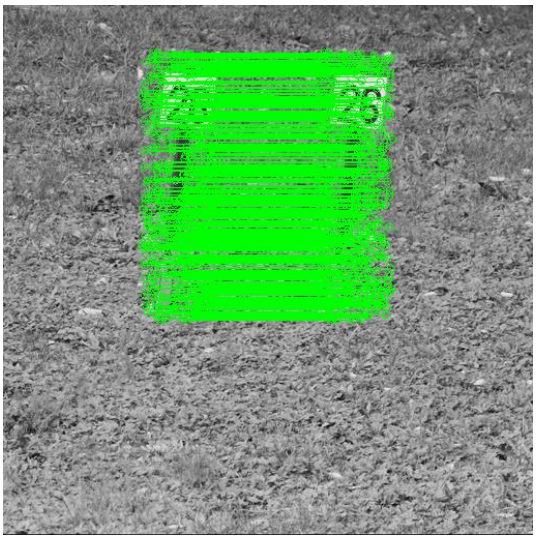
Python forráskód:

```
# 5.5) Filter with RANSAC
final_matches = []
for i in range(len(good_matches)):
    if matchesMask[i] == 1:
        final_matches.append(good_matches[i])
```

Ezt követően a megjelenítéshez ismét használjuk a már említett kódot csak ebben az esetben már nem a good_mathes elemin megyünk végig, hanem a final_matches elemein.



Jellemző kulcspontpárok a képen
(hamis párok is szerepelnek)



Jellemző kulcspontpárok a képen
(A végleges párokkal, RANSAC után)

3.3.1 ábra

Python forráskód:

```
# 5.3.1) Draw Lines and circles on the image (optional)
# Grayscale image convert to RGB image
img_RGB = cv2.cvtColor(img_gray, cv2.COLOR_GRAY2RGB)

list_point1 = []
list_point2 = []
for j in final_matches:

    # Get the matching keypoints for each of the images
    point1 = j.trainIdx
    point2 = j.queryIdx

    # Get the coordinates, x - columns, y - rows
    (x1, y1) = keypoints[point1].pt
    (x2, y2) = keypoints[point2].pt

    # Append to each list
    list_point1.append((int(x1), int(y1)))
    list_point2.append((int(x2), int(y2)))

    # Draw a small circle at both co-ordinates: radius 4, colour green,
thickness = 1
    # copy keypoints circles
    cv2.circle(img_RGB, (int(x1), int(y1)), 4, (0, 255, 0), 1)
    # original keypoints circles
    cv2.circle(img_RGB, (int(x2), int(y2)), 4, (0, 255, 0), 1)

    # Draw a line in between the two points, thickness = 1, colour green
    img3 = cv2.line(img_RGB, (int(x1), int(y1)), (int(x2), int(y2)), (0,
255, 0), 1)
```

2.4. Régió korrelációs térkép számítása

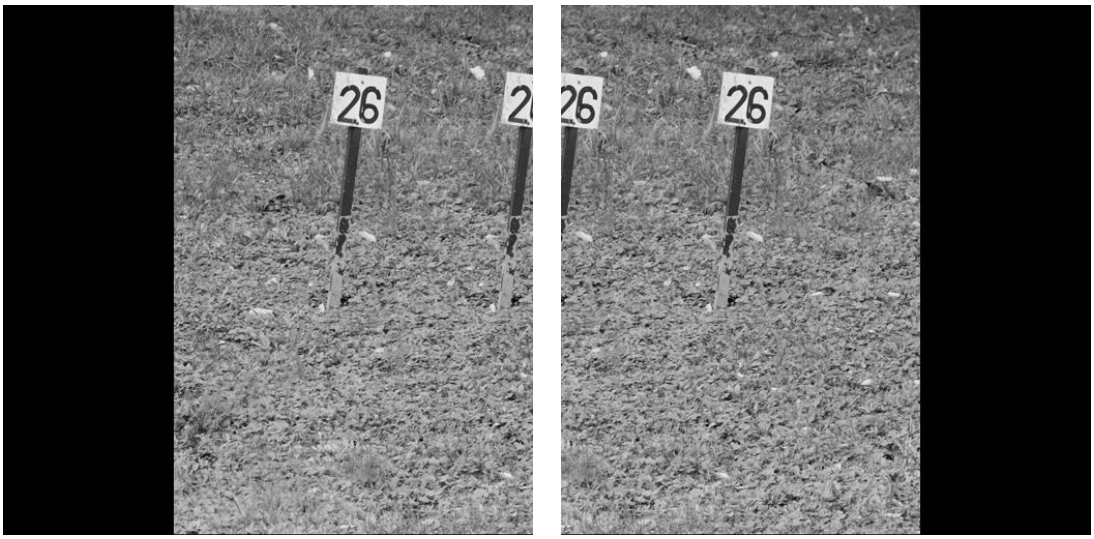
A duplikált régiók lokalizálása érdekében először a becsült affin transzformációs mátrix és az inverz transzformáció segítségével kiszámítjuk az eredeti kép és a transzformált kép közötti korrelációs térképet.

Első lépésként szükségünk van a transzformált képek megvalósítására az affin transzformációs mátrixok használatával. Mint már korábban említettem ezt az estimateAffine2D() függvénnyel kaphatjuk meg. Mivel két transzformált képre van szükségünk a korrelációs térképhez nem elég csak egy affin transzformációs mátrix annak inverzére is szükségünk van. Ezt legkönnyebben úgy kaphatjuk meg, ha ismét használjuk az estimateAffine2D() függvényt csak a kezdő és vég kulcspontok sorrendéjt megfordítjuk, így pontos eredményt kapunk mivel más mátrix invertálási módszer pontatlanságot okozhat a végeredményben az affin esetében.

Python forráskód:

```
# Affine Transform matrix for forward transform matrix
    retval_forward, inliers_forward = cv2.estimateAffine2D(src_pts,
dst_pts, method=cv2.RANSAC, ransacReprojThreshold=3, maxIters=100,

confidence=0.99)
# Affine Transform matrix for backward transform matrix
    retval_backward, inliers_backward = cv2.estimateAffine2D(dst_pts,
src_pts, method=cv2.RANSAC, ransacReprojThreshold=3, maxIters=100,

confidence=0.99)
```

Ha megvan a két affin transzformációs mátrix, akkor a warpAffine() függvény alkalmazható az eredeti szürkeárnyalatos képre. A paramétereknél szükség van a kép megadására, amire alkalmazni szeretnénk, második paraméter a transzformáció mátrix, harmadik pedig a végeredmény kép szélessége és magassága, ami megegyezik az eredeti kép szélességével és magasságával.



Affin traszformációs mátrix előre (Forward)

Affin traszformációs mátrix hátrafelé (Backward)

3.4.1 ábra

Python forráskód:

```
# 6) Computing region correlation maps
# 6.1) WrapAffine for forward map
wrapAffine_img_forward = cv2.warpAffine(img_gray, retval_forward, (w, h))

# 6.1) WrapAffine for backward map
wrapAffine_img_backward = cv2.warpAffine(img_gray, retval_backward, (w,
h))
```

A korrelációs képlet megvalósítása elött szükségünk van két üres kép létrehozására melyeknek minden pixele nullával van feltöltve és megegyező a mérete az eredeti képünk méretével, float32 típussal a pontos számolás végett. Később ezt fogjuk feltölteni pixelenként a korrelációs képlet alapján.

Python forráskód:

```
# 6.2) Black images filled up with zeros
forward_correlation_map = np.zeros((h, w, 1), np.float32)
backward_correlation_map = np.zeros((h, w, 1), np.float32)
```

Az imént említett képlet ami, a tanulmányba szerepel nem hozta a várt eredményt, ezért más tanulmányokban kezdtem kutatni és végül Guonian Jin és Xiaoxia Wan [27]

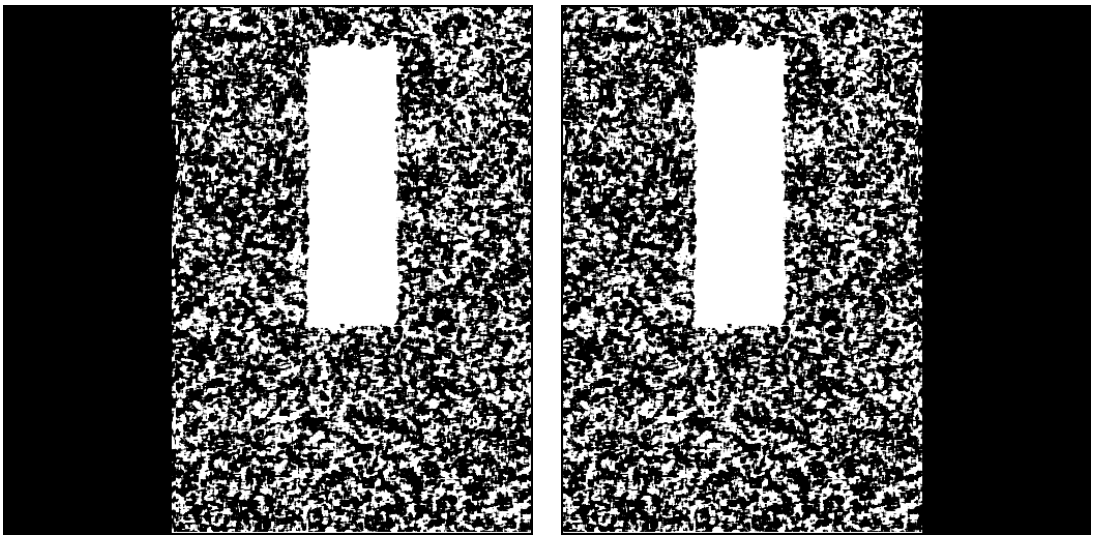
tanulmányából használtam fel a korrelációs képletet, amely annyiban tér el Xunyu Pan és Siwei Lyu [1] tanulmányában szereplő képlettől, hogy a nevezőben elvégeznek egy gyökvonást. A korrelációs térképet az eredeti kép I és a transzformált kép W pixel intenzitásának összehasonlításával kapjuk meg. A képlet a következőképpen néz ki ahol $C_f(x)$ a korrelációs térkép.

$$C_f(x) = \frac{\sum_{v\in\Omega(x)}(I(v) - \bar{I})(W(v) - \bar{W})}{\sqrt{\sum_{v\in\Omega(x)}(I(v) - \bar{I})^2(W(v) - \bar{W})^2}}$$

A képletben $\Omega(x)$ az 5×5 pixel szomszédos terület, az ablak középső pixele az x, $I(v)$ és $W(v)$ a pixel intenzitást jelöli v helyen, $\bar{I}$ és $\bar{W}$ az átlag pixel intenzitás I és W szomszédos területnek $\Omega(x)$.

Tehát az eredeti szürkeárnyalatos képen és az affin transzformációs mátrix segítségével létrehozott képen végig megyünk párhuzamosan minden egyes pixelen és annak 5×5 környezetén, kiszámoljuk a környezet átlagát, amit ezután kivonunk minden egyes pixelből az 5×5 környezetben. Miután ezt elvégeztük mind a két kép esetében a két pixel 5×5 környezetét összeszorozzuk, egymással pixelenként majd ezek összegezve kapjuk meg a számlálót. A nevezőt hasonlóképpen kapjuk meg csak az összeszorzás elött négyzetre kell emelnünk minden elemet, és végül az egész nevezőből gyököt vonunk.

A korrelációs térkép megvalósításához nincs beépített függvény az OpenCV-be ezért saját módszert kellett rá írni, ez jelentősen rontja a program futási idejét. Mivel nincs beépített függvény rá ezért az ablakozásnál felmerül az a probléma, hogy a kép szélén lévő pixeleknél túl nyúlik az ablak a kép szélén. Ezt azzal orvosoltam, hogy a szélső négy pixelt a kép jobb és alsó szélén illetve két pixelt a bal és felső szélén nem veszek bele korrelációs térkép számításába. A végső lépés, hogy a két üres képet, amit előre létrehoztam feltöltöm pixelenként a kapott eredménnyel. Két különböző feltételt nézünk az üres képek feltöltésénél, ha a nevező nem egyenlő nullával ekkor a pixel intenzitás a képlet alapján kapjuk meg, és ha egyenlő nullával ebben az esetben a pixel intenzitás nulla lesz. Mivel az ablak létrehozásánál paraméterként a bal felső sarkot kell megadni a kezdő (x, y) és a jobb alsó végpontot $(x + 5, y + 5)$, ahhoz hogy megkapjam az ablak középső elemét az $(x + 2, y + 2)$ helyre kell beraknom a végeredményt. A másik transzformált kép esetében is ugyan ez az eljárás.



Előre traszformált korrelációs térkép (Forward)

Hátrafelé traszformált korrelációs térkép (Backward)

3.4.2 ábra

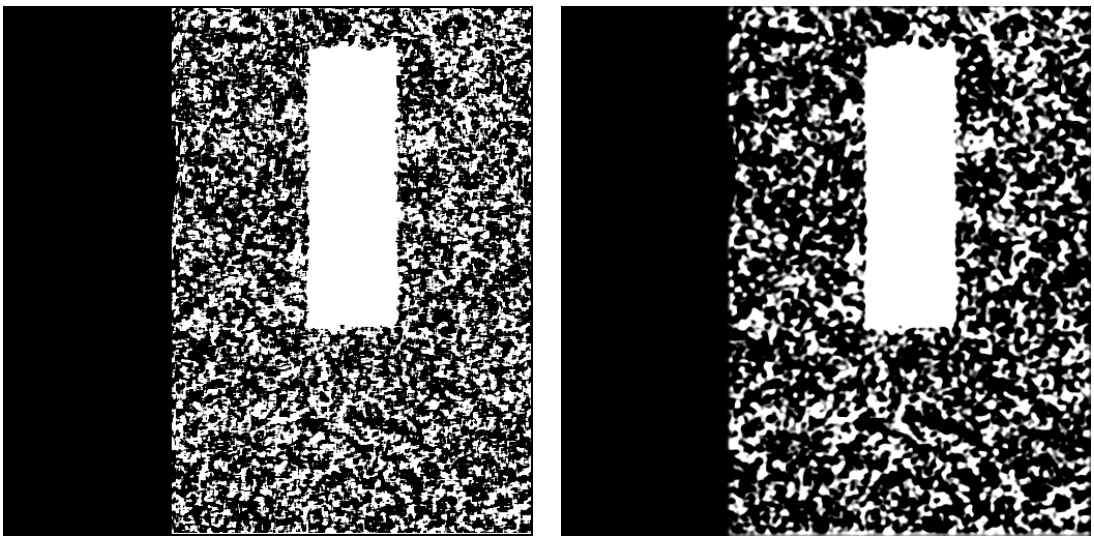
Mint megfigyelhetjük ahol a másolt régió található ott nagyon sűrű a korreláció viszont a kép többi részén is vannak fehér kis foltok. Ezért szükség van ezek eltüntetésére, hogy egy tiszta maszkot kapjunk a végén.

Python forráskód:

```
# 6.3.1) Computing region correlation map based on Forward affine
transform matrix
for y in range(0, h - 4):
    for x in range(0, w - 4):
        window1 = img_gray[y: y + 5, x: x + 5]
        window2 = wrapAffine_img_forward[y: y + 5, x: x + 5]

        I = window1 - window1.mean()
        W = window2 - window2.mean()

        top = np.sum(np.multiply(I, W, dtype=np.float32),
dtype=np.float32)
        bottom = math.sqrt(np.sum(np.multiply(np.multiply(I, I,
dtype=np.float32), np.multiply(W, W, dtype=np.float32),
                                                dtype=np.float32),
dtype=np.float32))

        if bottom != 0.0:
            intensity = (top / bottom)
            forward_correlation_map[y + 2, x + 2] = intensity
        elif bottom == 0.0:
            intensity = 0
            forward_correlation_map[y + 2, x + 2] = intensity
```

2.5. Kép utólagos feldolgozása

Módszerünk utolsó lépése a régiókorrelációs térképek feldolgozása a duplikált régiók megszerzéséhez. A korrelációs térkép utólagos feldolgozásához több lépésre van szükségünk, hogy eltüntessük a nem kívánatos hamis korrelációkat. Első lépésként 7×7 pixeles Gauss-szűrést alkalmazunk a két korrelációs térképünkre, amit a GaussianBlur() függvénnyel kapunk meg.



Előre traszformált korrelációs térkép (Forward)

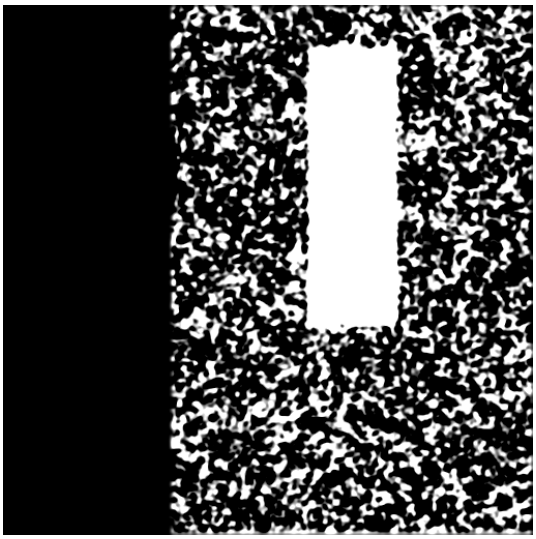
Előre traszformált korrelációs térkép a Gauss-szűrés után

3.5.1 ábra

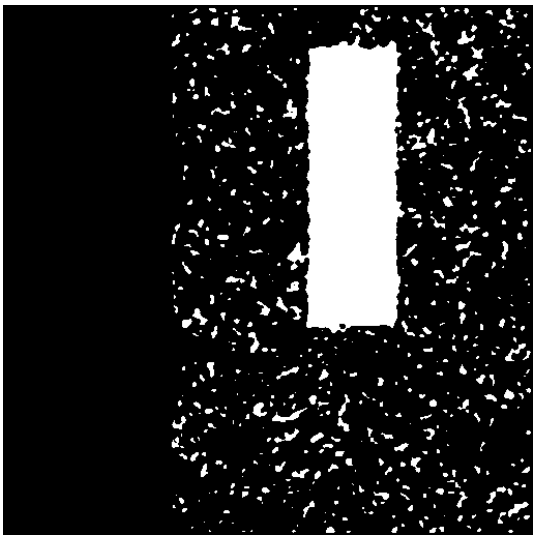
Python forráskód:

```
# 7) Post processing
# 7.1) Apply gaussian filter, kernel(7, 7)
gaussian_forward = cv2.GaussianBlur(forward_correlation_map, (7, 7), 0)
gaussian_backward = cv2.GaussianBlur(backward_correlation_map, (7, 7), 0)
```

Ezután globális küszöbölést hajtunk végre ahol a küszöbérték a maximális pixel intenzitás érték 30%-a. A küszöbölés után bináris képet kapunk eredményül. Ahhoz hogy megkapjuk a maximális pixel intenzitás értékét a minMaxLoc() függvényt használjuk.



Korrelációs térkép a Gauss-szűrés után



Küszöbölés után bináris kép

3.5.2 ábra

Python forráskód:

```
# 7.2) Apply binary threshold
# 7.2.1) correlation map max intensity value
minVal_f, maxVal_f, minLoc_f, maxLoc_f =
cv2.minMaxLoc(forward_correlation_map)
print(maxVal_f)
minVal_b, maxVal_b, minLoc_b, maxLoc_b =
cv2.minMaxLoc(backward_correlation_map)
print(maxVal_b)

# 7.2.2) Threshold
ret1, threshold_forward = cv2.threshold(gaussian_forward, (maxVal_f *
0.3), 255, cv2.THRESH_BINARY)
ret2, threshold_backward = cv2.threshold(gaussian_backward, (maxVal_b *
0.3), 255, cv2.THRESH_BINARY)
```

A küszöbölés után kapott bináris képen még mindig sok olyan terület van, ami nem megfelelő ennek további szűrésére morfológiai műveletekkel letisztítjuk a bináris képet. Ezekhez először létre kell hozni egy strukturáló elemet, ami esetünkben egy 12×12 pixeles négyzet alakú ablak lesz, a méret próbák alapján választottam meg, kép mérettől függetlenül ezzel a mérettel kaptam a legtisztább eredményeket, a tanulmányba szereplő méret sajnos nem hozott ideális eredményt. A morfológiai szűrést a dilatáció (dilate) és erózió (erode) egymás utáni végrehajtásával valósítjuk meg. Az erózió csökkent a képi komponenseken viszont azt a területet is erodálja (csökkenti) ami a valódi korrelációs terület, a dilatáció hizlal a képi komponenseken. Ezt a két egymás után következő függvényt helyettesíteni lehet egy másik

függvénnyel. A sorrend használatától függően használhatjuk a morfológiai nyitást (open) és zárást (close), a nyitás először eróziót végez aztán dilatációt, a zárás pedig először a dilatációt aztán eróziót. Elsőként a nyitást végzem el, hogy az erózió eltüntesse a kisebb területeket így viszont a valódi korrelációs területből is vesztünk ezért az ezt követő dilatáció visszanöveli a területét, de csak a valódi korrelációs területet növeli mivel az erózió eltüntette a kisebb nem kívánatos területeket. Ezt követően ennek a megismétlése fordított sorrendben a zárás operátorral.

Nyitás operátor után Zárás operátor után

3.5.3 ábra

Python forráskód:

```
# 7.3) Remove small isolated regions and close holes
# 7.3.1) Creating struck for morphological operations
struct = cv2.getStructuringElement(cv2.MORPH_RECT, (12, 12))

# 7.3.2) Discard all small isolated regions
# Apply the morphological open operation (erode->dilate)
open_forward = cv2.morphologyEx(threshold_forward, cv2.MORPH_OPEN,
struct, iterations=1)
open_backward = cv2.morphologyEx(threshold_backward, cv2.MORPH_OPEN,
struct, iterations=1)

# 7.3.3) Apply the morphological closing operation (dilate->erode)
close_forward = cv2.morphologyEx(open_forward, cv2.MORPH_CLOSE, struct,
iterations=1)
close_backward = cv2.morphologyEx(open_backward, cv2.MORPH_CLOSE, struct,
iterations=1)
```

Az erózió és a dilatáció miatt megváltozott a területünk alakja, így egy bitenként műveletet hajtok végre rajta a bitwise_and() függvénnyel. Így eltűnik az a rész ahol túl lövés van a küszöbölt képhez képest. De előfordulhat, hogy a területen belül ilyenkor újra lesz egy üres rész, ha csak egy pixel is, ezért a zárást kell ismét alkalmazni.

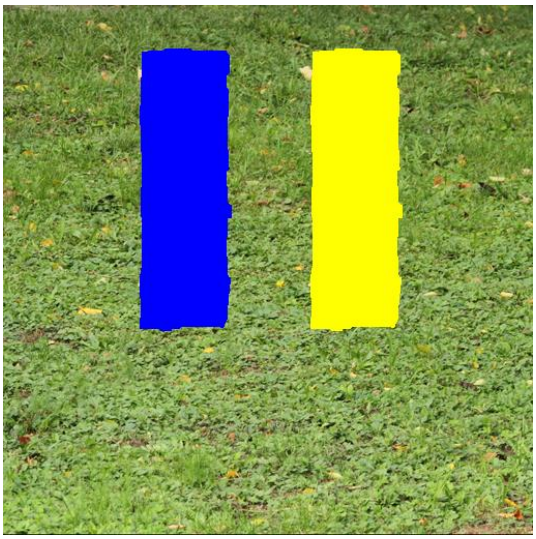
Nyitás operátor után Zárás operátor után

3.5.4 ábra

Python forráskód:

```
# Bitwise operation to get closer to the original shape
bitwise_forward = cv2.bitwise_and(close_forward, threshold_forward)
bitwise_backward = cv2.bitwise_and(close_backward, threshold_backward)
# Close holes after Bitwise
close_forward2 = cv2.morphologyEx(bitwise_forward, cv2.MORPH_CLOSE,
struct, iterations=1)
close_backward2 = cv2.morphologyEx(bitwise_backward, cv2.MORPH_CLOSE,
struct, iterations=1)
```

Végső lépésként a kapott eredmény területét valamilyen színnel megjelenítjük az eredeti RGB képen. Ahol a bináris képen egyes szerepel, csak ott kap másik színt. Ez a metódus utolsó lépése. Végül szeretném megjeleníteni négy képet a folyamatról (eredeti kép, kulcspontpárok, korrelációs térkép, detektált duplikált régiók az eredeti képen) ezt a matplotlib könyvtár használatával tudom megtenni a következő képpen (3.1 ábra):

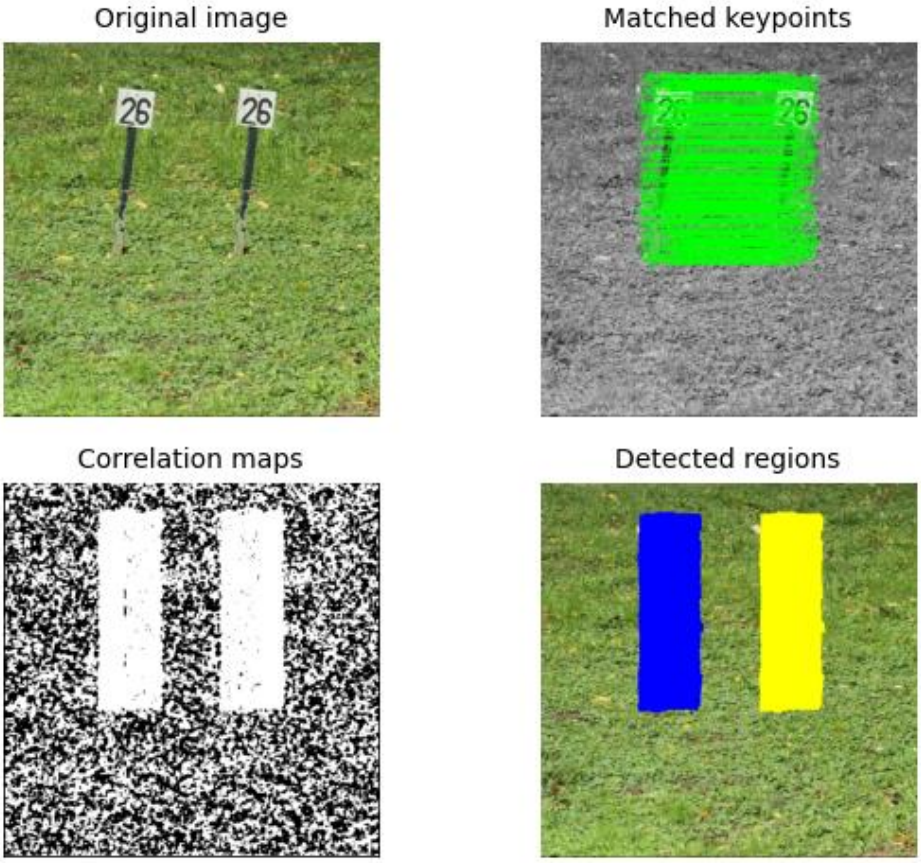


Eredeti kép A két duplikált régió

3.5.5 ábra

Python forráskód:

```
# 7.4) Color the original image where correlate
img_forward = img.copy()
img_forward[threshold_forward > 0] = 0, 255, 255

img_backward = img_forward.copy()
img_backward[threshold_backward > 0] = 255, 0, 0

# 8) Output
# 8.1) Need to convert BGR to RGB for matplotlib
img_BRG = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img_detected = cv2.cvtColor(img_backward, cv2.COLOR_BGR2RGB)
# 8.2) Both correlation map in one image
forward_correlation_map_show = cv2.cvtColor(forward_correlation_map,
cv2.COLOR_GRAY2RGB)
backward_correlation_map_show = cv2.cvtColor(backward_correlation_map,
cv2.COLOR_GRAY2RGB)
correlation_map = np.add(forward_correlation_map_show,
backward_correlation_map_show, dtype=np.float32)*255
# 8.3) Final output
plt.subplot(2, 2, 1), plt.title("Original image", fontsize=10),
plt.axis('off'), plt.imshow(img_BRG)
plt.subplot(2, 2, 2), plt.title("Matched keypoints", fontsize=10),
plt.axis('off'), plt.imshow(img_Fmatches)
plt.subplot(2, 2, 3), plt.title("Correlation maps", fontsize=10),
plt.axis('off'), plt.imshow(correlation_map)
plt.subplot(2, 2, 4), plt.title("Detected regions", fontsize=10),
plt.axis('off'), plt.imshow(img_detected)
plt.tight_layout()
plt.savefig('final_image.png')
plt.show()
cv2.waitKey(0)
cv2.destroyAllWindows()
```



3.5.6 ábra: A végső eredmény, négy fő lépés összesítve

3. PROGRAM TESZTELÉSE ÉS ELÉRT EREDMÉYNEK

A program teszteléséhez készítettem összesen 100 db képet, JPEG és PNG formátumban típusonként 50-50 db. Tíz különböző kép módosításával készítettem, mindegyik képen 5 módosítást hajtottam végre:

1. Másolás-mozgatás: A duplikált régiót torzítás nélkül átmásoltam a célhelyre.
2. Forgatás: A duplikált régiót véletlenszerűen forgattam el egy szögbe (0°-360°).
3. Nagyítás: A duplikált régiót véletlenszerűen átméreteztem kicsinyítéssel (max 80%-a az eredeti méretnek) vagy nagyítással (max 200%-a az eredeti méretnek).
4. Szabadalakítás: A duplikált régiót eltorzítottam véletlenszerűen.
5. Pixel intenzitás: A duplikált régió itt megegyezik a másolás-mozgatásnál kapottnál, de az eredeti pixel intenzitás értékét megváltoztattam annak 80%-ra.

A képeket Photoshop segítségével készítettem el. Mindegyik kép mérete 512×512 pixel. A képek forrása CoMoFoD - Image Database for Copy-Move Forgery Detection [35], csak az eredeti módosítás nélküli képeket használtam fel, majd módosítottam azokat.
A program minden esetben sikeresen megtalálta a duplikált régiót. Azonban a program gyenge pontja, ha a másolt régió túlságosan homogén vagy kicsi területű és nem található rajta megfelelő számú kulcspont, ezentúl a megtalált régiók területét nem fedi le 100%-os pontossággal a végső színezett terület mivel az utólagos feldolgozás miatt néhány helyen túl lövés van és azokat a területeket is beszínezi, ami nem tartozik a duplikált részhez, bizonyos részeken pedig alul lövés vehető észre.

Másolás-mozgatás | Forgatás | Átméretezés | Szabad forma | Pixel intenzitás

4. PROGRAM ELŐKÉSZÍTÉSE ÉS HASZNÁLATI ÚTMUTATÓ

A programot az OpenCV platformon alapuló Python nyelv használatával valósítottam meg. A Python egy interpretált, általános célú programozási szkriptnyelv A program futtatásához Python 3-as verziójára van szükség. Emellé szükség van az OpenCV függvénykönyvtárra. Fontos hogy OpenCV 4.4.0 vagy újabb verzió legyen telepítve mivel SIFT nincs benne a korábbi alap függvénykönyvtárban egészen 2.4.x verzióig visszamenőleg. Ha a köztes verziókkal szeretnénk használni szükség, van az opencv_contrib telepítésére is, de ez önmagában nem elég, CMmake-el saját könyvtárat kell készíteni hozzá hogy, bekapcsolhassuk az OPENCV_ENABLE_NONFREE modult is, A SURF használatához jelenleg is ez a módszer szükséges.

Python telepítése: A megvalósításhoz én Anacondát telepítettem fel, ami egy Python disztribúció, azért esett erre a választás, mert fontos program csomagok előre telepítve vannak benne, mint például a numpy és a matplotlib. Sima Python interpretert letölthetjük a Python hivatalos oldaláról a python.org-ról.

OpenCV telepítése: A legújabb OpenCV verzió a legegyszerűbben a Pip csomag kezelővel telepíthető. A csomagkezelő megtalálható a Python 3.4-es verziótól kezdve külön telepítés nélkül, azonban érdemes frissíteni azt, ezt egy parancssor megnyitásával tehetjük meg ezután a következő paranccsal frissíthetjük: pip install --upgrade pip. Ha sikeresen lefrissítettünk a Pip csomagkezelőt, akkor ezután még egy parancs szükséges az OpenCV függvénykönyvtár telepítéshez: pip install opencv-python, ez automatikusan letölti a legfrissebb elérhető OpenCV-t.

Python IDE: A Python programok fejlesztéséhez integrált fejlesztői környezetet (IDE) is használhatunk. Én a JetBrains által készített PyCharm-ot használtam. A program futtatásához ez nem szükséges parancssorból is lehet futtatni, de kényelmesebb megoldás az IDE. A beállításokban fontos megmutatni neki a telepített Python interpretert.

Egyéb függvénykönyvtárak telepítése: Mint említettem az Anaconda esetében ez nem szükséges azonban sima Python interpreter esetében szükség van a Numpy, Matplotlib telepítésére is. A következő két parancs szükséges hozzá szintén a Pip csomagtelepítő segítsévével: pip install numpy, pip install matplotlib.

IRODALMI JEGYZÉK

[1] X. Pan, S. Lyu: Region Duplication Detection Using Image Feature Matching, In IEEE Trans. on Information Forensics and Security, vol 5, No. 4, pp. 857-867, 2010, IEEE

[2] Lanh, T.V.L.T., Van Chong, K.-S., Chong, K.-S., Emmanuel, S., Kankanhalli, M.S.: A survey on digital camera image forensic methods. In: 2007 IEEE International Conference on Multimedia and Expo, pp. 16–19 (2007)

[3] Farid, H.: A survey of image forgery detection techniques. IEEE Signal Process. Mag. 26, 16–25 (2009)

[4] Nor Bakiah Abd Warifa, Ainuddin Wahid Abdul Wahaba, Mohd Yamani Idna Idris, Roziana Ramli, Rosli Salleha, Shahaboddin Shamshirbanda, Kim-Kwang Raymond Choobc: Copy-move forgery detection: Survey, challenges and future directions, 2016

[5] Farid, H.: Detecting digital forgeries using bispectral analysis. Mit Ai Memo Aim-1657 Mit (1999)

[6] Shen, X., Shi, Z., Chen, H.: Splicing image forgery detection using textural features based on the grey level co-occurrence matrices. IET Image Process. 11, 44–53 (2017)

[7] Popescu, A.C., Farid, H.: Exposing Digital Forgeries by Detecting Traces of Resampling Resampling Detecting Resampling Experiment Results (2005)

[8] Mahdian, B., Saic, S.: Blind authentication using periodic properties of interpolation. IEEE Trans. Inf. Forensics Secur. 3, 529–538 (2008)

[9] Stamm, M., Ray, K.J.: Blind forensics of contrast enhancement in digital images. In: Proceedings of International Conference on Image Process ICIP, pp. 3112–3115 (2008)

[10] Cao, G., Zhao, Y., Ni, R., Kot, A.C.: Unsharp masking sharpening detection via overshoot artifacts analysis. IEEE Signal Process. Lett. 18, 603–606 (2011)

[11] Cao, G., Zhao, Y., Ni, R., Li, X.: Contrast enhancement-based forensics in digital images. IEEE Trans. Inf. Forensics Secur. 9, 515–525 (2014)

[12] Silva, E., Carvalho, T., Ferreira, A., Rocha, A.: Going deeper into copy-move forgery detection: Exploring image telltales via multi-scale analysis and voting processes. J. Vis. Commun. Image Represent. 29, 16–32 (2015)

[13] Lee, J.C., Chang, C.P., Chen, W.K.: Detection of copy-move image forgery using histogram of orientated gradients. Inf. Sci. (Ny) 321, 250–262 (2015)

[14] Zhu, Y., Shen, X., Chen, H.: Copy-move forgery detection based on scaled ORB. Multimed. Tools Appl. 75, 3221–3233 (2016)

[15] Lowe, D.G. Distinctive Image Features from Scale-Invariant Keypoints.International Journal of Computer Vision 60, 91–110 (2004).

[16] Kunj Bihari Meena, Vipin Tyagi: Image Forgery Detection: Survey and Future Directions, 2019

[17] Ardizzone, E., Bruno, A., Mazzola, G.: Copy-move forgery detection by matching triangles of keypoints. IEEE Trans. Inf. Forensics Secur. 10, 2084–2094 (2015)

[18] Saba Mushtaq, Ajaz Hussain Mir: Digital Image Forgeries and Passive Image Authentication Techniques: A Survey, 2014

[19] Ewerton Silva, Tiago Carvalho, Anselmo Ferreira, Anderson Rocha: Going deeper into copy-move forgery detection: Exploring image telltales via multi-scale analysis and voting processes, 2015

[20] Matthew Stamm, K.J. Ray Liu: Blind forensics of contrast enhancement in digital images. In 2008 15th IEEE International Conference on Image Processing, 12-15 Oct. 2008

[21] David Doermann: Handbook of Document Image Processing and Recognition, Springer, 2014

[22] Rafael C. Gonzalez, Richard E. Woods: Digital Image Processing, 3rd Edition, Prentice Hall, 2008 Warif, N.B.A., Wahab, A.W.A., Idris, M.Y.I.: Copy-move forgery detection: survey, challenges and future directions. J. Netw. Comput. Appl. (2016)

[23] Lanh, T.V.L.T., Van Chong, K.-S., Chong, K.-S., Emmanuel, S., Kankanhalli, M.S.: A survey on digital camera image forensic methods. In: 2007 IEEE International Conference on Multimedia and Expo, pp. 16–19 (2007)

[24] Gajanan K.Birajdara, Vijay H.Mankarb: Digital image forgery detection using passive techniques: A survey

[25] Mostafa Mokhtari Ardakan,Masoud Yerokh, Mostafa Akhavan Saffar: A New Method to Copy-Move Forgery Detection in Digital Images Using Gabor Filter, 26 July 2018

[26] Mahmood, T., Nawaz, T., Irtaza, A., Ashraf, R., Shah, M., & Mahmood, M.T. (2016). Copy-Move Forgery Detection Technique for Forensic Analysis in Digital Images. Mathematical Problems in Engineering, 2016, 1-13.

[27] G. Jin, X. Wan, An improved method for SIFT-based copy-move forgery detection using non-maximum value suppression and optimized J-Linkage, Signal Processing: Image Communication (2017)

[28] Hany Farid: Image Forgery Detection, 2009

[29] Hany Farid: Photo Forensics, MIT Press, 2016

[30] Hany Farid: Fake Photos, MIT Press, 2019

[31] Hany Farid: Digital Image Forensics, PDF

[32] Hivatalos OpenCV dokumentáció: https://docs.opencv.org/master/

[33] Németh Gábor weboldala: https://www.inf.u-szeged.hu/~gnemeth/oktatas.html

[34] Tanács Attila weboldala: http://www.inf.u-szeged.hu/~tanacs/pyocv/

[35] Felhasznált képek forrása: CoMoFoD - Image Database for Copy-Move Forgery Detection, https://www.vcl.fer.hr/comofod/index.html

KÖSZÖNETNYÍLVÁNÍTÁS

Ezúton szeretném köszönetet mondani témavezetőmnek, Dr. Németh Gábor egyetemi adjunktusnak, aki a szakdolgozatom írása során időt nem sajnálva mindvégig segítette a munkám, és tanácsaival, tapasztalatival, információkkal elősegítette a szakdolgozatom sikerességét. Tisztaszívvel köszönöm szüleimnek és családomnak, hogy idáig eljuthattam és a sok gondoskodást, ami elkísért a tanulmányaim során, továbbá barátaimnak, akik szeretetükkel támogattak egyetemi tanulmányaim közben.

NYILATKOZAT

Alulírott Kis Krisztián mérnökinformatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet képfeldolgozás és számítógépes grafika Tanszékén készítettem, mérnökinformatikus BSC diploma megszerzése érdekében. Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel. Tudomásul veszem, hogy szakdolgozatomat / diplomamunkámat a Szegedi Tudományegyetem Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

Szeged, 2020.december 12. .....

Aláírás

NYILATKOZAT

Alulírott Kis Krisztián mérnökinformatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet képfeldolgozás és számítógépes grafika Tanszékén készítettem, mérnökinformatikus BSC diploma megszerzése érdekében. Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel. Tudomásul veszem, hogy szakdolgozatomat / diplomamunkámat a Szegedi Tudományegyetem Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

Szeged, 2020.december 12.

Aláírás