Spajanje fotografija snimljenih iz zraka koristeći Python

Repac, Dino

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

SPAJANJE FOTOGRAFIJA SNIMLJENIH IZ ZRAKA KORISTEĆI PYTHON

Dino Repac

Diplomski rad

Osijek, 2017

1. UVOD

Fotografiranje Zemlje iz zraka postaje sve popularnije razvojem bespilotnih letjelica (engl. *drone*). Takve letjelice su jednostavne, male, agilne i poprilično pristupačne za obične ljude. Također, razvojem tehnologije, dimenzije i težina kamera se smanjila, a njihova kvaliteta snimanja povećala. Bespilotne letjelice vrlo lako mogu nositi minijaturnu kameru koja ima sposobnost snimanja fotografija velike rezolucije.

Iako su vrlo sofisticirane, takve letjelice se u velikom postotku koriste za snimanje krajolika, te njihov potencijal nije u potpunosti iskorišten. Od nedavno, bespilotne letjelice s kamerama se koriste i u poljoprivredi, za nadzor polja i usjeva. Također u elektroprivredi za pregledavanje dalekovoda na nedostupnim lokacijama kao što su proplanci, vrhovi planina, guste šume, itd.

Pregled niza fotografija često predstavlja problem, jer je potrebno definirati gdje se objekti na fotografiji nalaze u odnosu na prethodne fotografije, stoga cilj ovog diplomskog rada je izraditi pseudokod koji će od niza fotografija izraditi jednu, velike rezolucije.

Set fotografija koje će se koristiti za testiranje algoritma zabilježene su na visini od 100 metara te s minimalnim preklapanjem od 70%. Fotografije su zabilježene kamerom GoPro Hero 4 Black. Izobličenja uzrokovana lećom kamere uklonit će se prije pokretanja algoritma za spajanje.

U uvodnom poglavlju dodatno će biti objašnjen način prikupljanja fotografija te obrada istih. U drugom poglavlju dat će se teorijska osnova na kojoj je zasnovan pseudokod. Objasnit će algoritmi korišteni za prepoznavanje značajki i algoritmi za spajanje fotografija. U trećem poglavlju biti će prikazan razvoj algoritma uz popratne fotografije koje će vizualizirati rad pseudokoda. Te će u četvrtom poglavlju biti prikazani rezultati programskog rješenja.

1.1. Diplomski zadatak

Slike su snimane dronom, s unaprijed definirane visine, okomito na površinu Zemlje. Slike su snimljene kamerom visoke rezolucije te se kvaliteta slike treba zadržati prilikom spajanja. Potrebno je implementirati pseudokod spajanja niza slika koristeći Python programski jezik. Zabilježene slike su poboljšane prije spajanja kako bi se uklonila izobličenja uzrokovana kamerom. Sumentor: Denis Sušac

2. ZNAČAJKE, HOMOGRAFIJA I ALGORITMI

U ovom će poglavlju biti dana teorijska podloga za metode korištene u algoritmu. Prvo će se objasniti SIFT(engl. *Scale-Invariant Feature Transform*) za prepoznavanje značajki fotografija, zatim procjena homografije, točnije matrice homografije, i filtriranje vanpopulacijskih značajki koristeći RANSAC pseudokod.

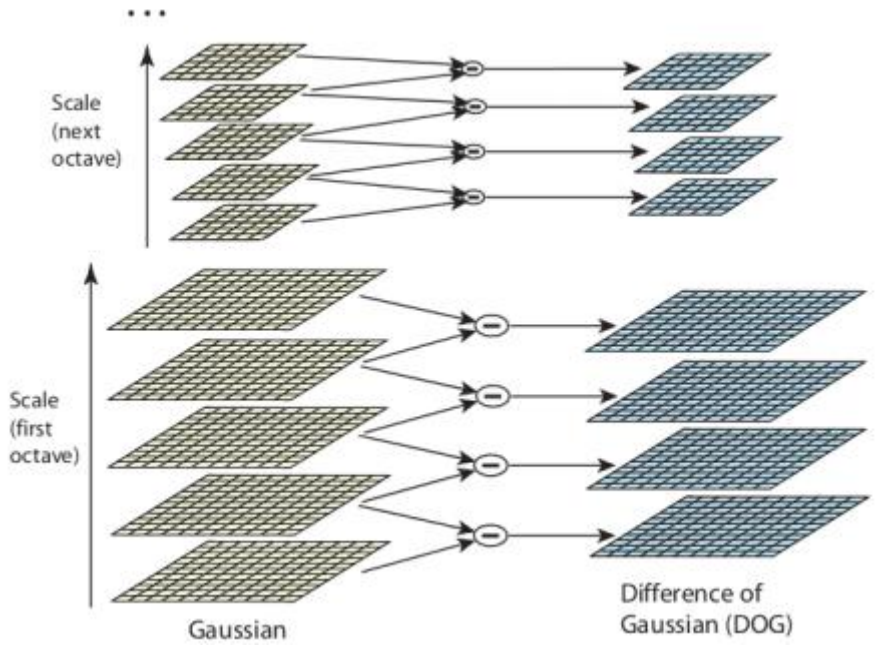
2.1.SIFT

SIFT ili *Scale-invariant Feature Transform* je pseudokod za prepoznavanje i opis značajki neke fotografije. Prema [1] SIFT spada u lokalne deskriptore koji se temelji na histogramu smjerova gradijenta svjetline. Način na koji SIFT funkcionira detaljno je opisan u [2]. U daljnjem će tekstu biti ukratko opisana funkcionalnost SIFT-a. Sam SIFT se dijeli na nekoliko koraka.

Detekcija ekstrema

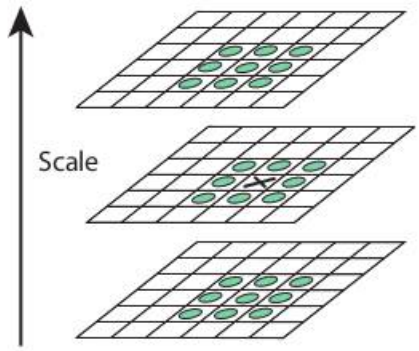
Za prepoznavanje kutova potrebni su određeni okviri. Ukoliko se želi prepoznati veći kut, potreban je veći okvir. Zbog toga, koristi se filtriranje po skala-prostoru, a zatim se pronalazi Laplace Gaussiana(u daljnjem tekstu - LoG) s različitim σ vrijednostima. LoG se ponaša kao detektor anomalija različitih veličina zbog promjene σ. Ukratko, σ se ponaša kao parametar skaliranja. Na primjer, Gaussova jezgra s malim σ daje velike vrijednosti za mali kut, dok Gaussova jezgra s velikim σ daje velike vrijednosti za veliki kut. Rezultat toga je lista (x,y, σ) vrijednosti koje nam daju potencijalne kandidate za kutove na lokaciji (x,y) i skali σ.

LoG je nepraktičan iz razloga što je neefikasan, i zbog toga SIFT koristi Razliku Gaussiana(Difference of Gaussians, dalje u tekstu DoG). DoG se dobije na način da se pronađe razlika mutnih slika s različitim σ vrijednostima(σ i kσ). Ovaj proces se odvija na različitim razinama slike u Gausovoj piramidi. Vidi sliku 2.1.



Slika 2.1 - Razlika Gaussian-a kroz različite vrijednosti σ [2]

Kad se DoG pronađu, na slici se traže lokalni ekstremi kroz skala-prostor. Dakle, svaki piksel slike je uspoređen sa svojih osam susjeda, također devet piksela u idućoj skali i devet piksela u prethodnoj skali. Ako je lokalni ekstrem, postoji šansa i da je ključna točka. Vidi sliku 2.2.



Slika 2.2 - usporedba piksela s promjenjivom skalom [2]

Također, Lowe je dao optimalne vrijednosti algoritma: broj oktava = 4, broj nivoa skaliranja = 5, početni $\sigma = 1.6$, $k = \sqrt{2}$

Lokalizacija ključnih točaka

Nakon što se potencijalne ključne točke pronađu, moraju se provjeriti kako bi se dobili što točniji rezultati. U ovom koraku se koristi proširivanje skala-prostora Taylorovim redom kako bi se dobili točniji rezultati lokalnih ekstrema i ako je intenzitet ekstrema manji od zadanog praga(0.03 kako je definirao Lowe), lokalni ekstrem je odbačen. Prag od 0.03 zove se *contrastThreshold* u OpenCV biblioteci.

DoG ima mogućnost prepoznavanja i rubova, stoga se i rubovi trebaju ukloniti. Za to, koristi se koncept sličan Harrisovom detektoru rubova. Koristi se 2x2 Hessian matrica(H) kako bi se izračunala glavna zakrivljenost. Ako je omjer svojstvenih vrijednosti veći od praga(*edgeThreshold* u OpenCV biblioteci), ključna točka je odbačena. Preporučena vrijednost praga je 10. Ovime su uklonjene ključne točke slabog kontrasta i rubovi, ostaju samo točke velikog interesa.

Definiranje orijentacije

Na rezultate iz prethodnog koraka dodjeljuje se orijentacija kako bi rezultati bili invarijantni na orijentaciju. Ovisno o skali, uzima se jedan susjed oko ključne točke te se računa gradijent magnitude i smjer unutar regije interesa. Zatim se stvara histogram s 36 košara koji pokriva 360 stupnjeva. Iz histograma se uzima najviši vrh i svaki vrh veći od 80% se koristi za računanje orijentacije. Kreira se ključna točka s istom lokacijom i skalom, ali različitom orijentacijom. To doprinosi stabilnosti kod pronalaska ključne točke.

Deskriptor ključnih točaka

Oko ključne točke uzima se 16x16 polje što definira deskriptor. Deskriptor je zatim podijeljen u 16 blokova veličine 4x4. Za svaki blok stvara se histogram s 8 košara, a ukupno 128 košara. Predstavljen je kao vektor koji formira deskriptor ključne točke.

2.2. Homografija i RANSAC

Drugi dio algoritma se sastoji od računanja matrice homografije, tj. ravninske transformacije između dvije susjedne fotografije. Za izračun matrice homografije potrebne su koordinate značajki obje fotografije. Matricu homografije se primjenjuje na iduću sliku u nizu kako bismo tu fotografiju prebacili u ravninu referentne, odnosno inicijalne, fotografije.

Homografija se može opisati jednadžbom:

$$\lambda * x' = H * x \tag{2-1}$$

gdje je x' točka u koordinatnom sustavu transformirane fotografije, x točka u koordinatnom sustavu inicijalne fotografije, a H matrica homografije, te λ koeficijent. Ako uvedemo koordinate, jednadžba izgleda ovako:

$$\lambda * \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2-2}$$

Prije računanja homografije potrebno je odbaciti sve vanpopulacijske značajke(engl. *Outliers*). To će odraditi pseudokod RANSAC koji će se detaljnije opisati u nastavku.

Vanpopulacijskim značajkama se smatraju oni podaci koji ne odgovaraju "pravom" modelu iz seta "pravih" podataka uz prag pogreške koji definira maksimalnu devijaciju koja se može dodijeliti postojećem šumu.[4]

RANSAC

Prema [3,4] RANSAC ili *Random Sample Consensus* je iterativna metoda za procjenu parametara matematičkog modela iz skupa podataka. RANSAC radi na setu podataka koji je zasićen vanpopulacijskim podacima. Udio vanpopulacijskih podataka može biti veći od 50% unutar zadanog seta podataka za normalan rad RANSAC-a. Taj postotak, poznatiji još kao točka pucanja, je praktično ograničenje za mnoge druge algoritme(kao npr. LMeD – Least-Median of Squares).

RANSAC se u praksi sastoji od dva koraka koji se ponavljaju, a to su hipoteza i test.

- Hipoteza: prvi minimalni set podataka je nasumično odabran iz ulaznog seta podataka model parametara je sastavljen koristeći samo minimalni set podataka. Kardinalni broj minimalnog seta podataka je najmanja moguća da se odredi model parametara[1].
- Test: drugi korak provjerava koji elementi cijelog seta podataka su konzistentni s modelom instanciranim s parametrima procijenjenim u prvom koraku hipoteze. Set takvih elemenata se zove konsenzus set(engl. *Consensus set*) podataka.

Pseudokod se prekida kada vjerojatnost pronalaska bolje rangiranog seta konsenzusa padne ispod određenog praga.

[1] Na primjer, želimo procijeniti dužinu. Kardinalni broj minimalnog seta podataka je 2, jer s najmanje dvije točke možemo definirati dužinu.

3. RAZVOJ ALGORITMA

Pseudokod za spajanje fotografija koristi metode iz biblioteke OpenCV. OpenCV implementira… Prije svega potrebno je definirati osnovne zahtjeve algoritma.

Zahtjevi:

- Slijedno učitavanje fotografija iz datoteke
- Fotografijama se ne smije mijenjati rezolucija, smanjivati ih ili povećavati
- Učitavanje bazne(inicijalne) fotografije
- Detekcija kutova koristeći SIFT
- Učitavanje iduće fotografije u nizu
- Detekcija kutova nove fotografije
- Računanje matrice homografije koristeći RANSAC
- Spajanje fotografija
- Spremanje fotografije u datoteku izlaza

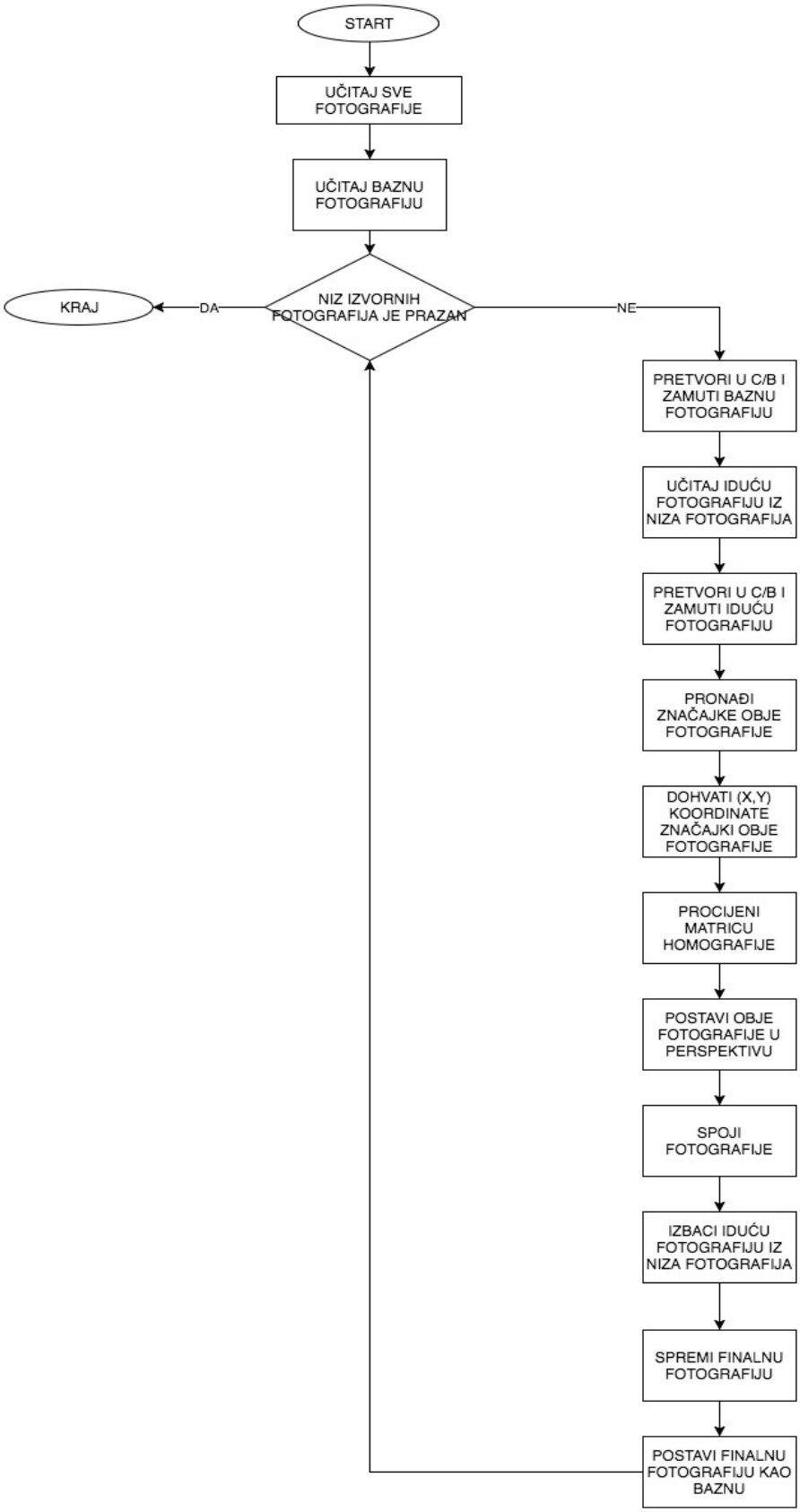
Način na koji pseudokod radi prikazan je dijagramom toka u koji se nalazi na idućoj stranici. Iz dijagrama toka se može vidjeti kako pseudokod radi rekurzivno. Iz niza učitanih fotografija, nakon učitavanja bazne fotografije učitava se prva iz niza izvornih fotografija. Kada se spoje bazna i iduća fotografija iz niza, pseudokod poziva sam sebe i pritom prosljeđuje fotografiju spojenu prethodnim izvođenjem kao baznu. Iz niza izvornih fotografija ponovno se učitava prva te se pseudokod spajanja ponovno izvodi. Pseudokod se izvodi dok se ne spoje sve fotografije iz niza izvornih fotografija. Detaljnije funkcioniranje algoritma i programski kod biti će objašnjeni u daljnjem tekstu.

Pseudokod je razvijen u programskom jeziku Python i detaljno će biti opisan u daljnjem tekstu uz pripadajući kod.

```
1.  import os
2.  import sys
3.  import cv2
4.  import numpy as np
5.  import utils
6.
7.  from numpy import linalg
```

Programski kod 1 - potrebne biblioteke

Na početku je potrebno dodati biblioteke koje će se koristiti u daljnjem kodu. Biblioteka *os* je biblioteka koja nam daje funkcionalnosti operativnog sustava, kao što su čitanje iz datoteka i upisivanje u datoteku. Biblioteka *sys* daje pristup prevoditelju i njegovim varijablama. Zatim slijede biblioteka *cv2* što je skraćeno ime za OpenCV, *numpy* koji je temeljna biblioteka za bilo kakva računanja te pruža algoritme za linearnu algebru(*linalg*), Fourierovu transformaciju i nasumične brojeve. Biblioteka *utils* je pomoćna biblioteka koja u sebi sadrži metode za ispisivanje slike na zaslon, također cijela biblioteka se nalazi u prilogu.



Dijagram toka 1 - tok rada pseudokoda

```
1.  class Stitch(object):
2.      def __init__(self, image_dir, key_frame, output_dir):
3.          # do something on init
4.
5.
6.  if __name__ == '__main__':
7.      if(len(sys.argv) < 4):
8.          print >> sys.stderr, ("Usage: %s <img_dir> <key_frame> <output>" % sys.argv[0])
9.          sys.exit(-1)
10.
11.     Stitch(sys.argv[1], sys.argv[2], sys.argv[3])
```

Programski kod 3 - klasa Stitch i main metoda

Nakon što su uključene potrebne biblioteke definirana je klasa *Stitch* koja na inicijalizaciji kroz konstruktor prima datoteku s izvornim fotografijama, fotografiju koja će biti prva ili bazna, te putanju do datoteke u koju će se spremiti izlaz. Od linije 6 u programskom kodu 2 do linije 11 definiran je način na koji se potrebni podaci učitavaju. Učitavaju se kroz komandnu liniju na način da se pozove iduća naredba:
python main.py putanja-do-izvora putanja-do-prve-fotografije/ime.ekstenzija putanja-do-direktorija-za-rezultat

U idućem potpoglavlju biti će objašnjena implementacija algoritma.

3.1. Implementacija algoritma

U programskom kodu 3 prikazana je implementacija konstruktora *Stitch* klase.

```
1.  self.key_frame_file = os.path.split(key_frame)[-1]
2.  self.output_dir = output_dir
3.
4.  self.dir_list = []
5.  try:
6.      self.dir_list = os.listdir(image_dir)
7.      try:
8.          self.dir_list.remove('Thumbs.db') #removes Thumbs.db if existent(Win only)
9.      except ValueError:
10.         pass # this will happen if there is no Thumbs.db file, so let it pass
11. except:
12.     print >> sys.stderr, ("Unable to open dir: %s" % image_dir)
13.     sys.exit(-1)
14.
15. self.dir_list = map(lambda x: os.path.join(image_dir, x), self.dir_list)
16. self.dir_list = filter(lambda x: x != key_frame, self.dir_list)
17.
18. base_img_rgb = cv2.imread(key_frame)
19. if base_img_rgb == None:
20.     raise IOError("%s doesn't exist" % key_frame)
21.
22. #TODO: maybe display base image?
23.
24. final_image = self.stitch(base_img_rgb, 0)
```

Programski kod 4 - __init__ metoda

U __*init*__ metodi se popunjava *dir_list* niz koji sadrži putanju do svih fotografija iz datoteke s izvornim fotografijama koje će se koristiti prilikom spajanja. Zatim se iz tog niza izbacuje inicijalna ili bazna fotografija te se ona učitava koristeći se metodom iz biblioteke OpenCV. Nakon što se učita bazna fotografija, ona se prosljeđuje metodi *stitch* iz klase *Stitch*.

Metoda *stitch* je glavna metoda koja spaja fotografije. U daljnjem tekstu biti će opisan proces spajanja fotografija. Kao ulazni parametar, metoda prima baznu fotografiju u boji. Zbog toga što SIFT pseudokod zahtjeva crno bijelu i zamućenu fotografiju, potrebno je to i učiniti na način prikazan u programskom kodu 4.

```
1.  base_img = cv2.GaussianBlur(cv2.cvtColor(base_img_rgb,cv2.COLOR_BGR2GRAY), (5,5), 0)
```

Programski kod 5 - pretvaranje u crno-bijelo i zamućivanje

Na slici 3.1 je prikazana originalna fotografija, a slika 3.2 prikazuje rezultat metode iz programskog koda 4. Fotografija sa slike 3.2 je pretvorena u crno bijelo i zamućena te se na nju može primijeniti SIFT detektor što će biti prikazano i objašnjeno u daljnjem tekstu.

Sl. 3.1 - originalna fotografija prije procesiranja algoritmom

Sl. 3.2 - fotografija poslije procesiranja algoritmom

Zatim se kreira SIFT detektor te se prosljeđuje fotografija.
Kao rezultat dobijemo značajke i deskriptore fotografije koji će se koristiti kasnije kod spajanja fotografija. Za spajanje fotografija koristit će se FLANN(*Fast Library for Approximate Nearest Neighbors*) pseudokod o kojem se detaljnije može pročitati u poglavlju 2.

```
1.  detector = cv2.xfeatures2d.SIFT_create()
2.  base_features, base_descs = detector.detectAndCompute(base_img, None)
```

Programski kod 6 - detekcija značajki

Prva fotografija je spremna za uspoređivanje, potrebno je ponoviti isti proces za iduću fotografiju. Zbog toga što su izvorne fotografije zabilježene slijedno, ne mora se vršiti analiza svake fotografije s baznom, potrebno je samo uspoređivati s idućom iz niza izvornih fotografija.

Stoga, po istom principu se učitava iduća fotografija u nizu, pretvara se u crno bijelu i na nju se dodaje zamućivanje kao što je prikazano u programskom kodu 4. Ponavlja se linija 2 iz programskog koda 6 za novu fotografiju te se dobiva druga fotografija koja je spremna za spajanje.

Prethodno definiranom algoritmu spajanja je potrebno proslijediti deskriptore obje fotografije, te koeficijent *k* koji definira koliko će rezultata za svako podudaranje pseudokod vratiti. Na slici 3.3 se može vidjeti kako izgledaju fotografije jedna pored druge netom prije spajanja.

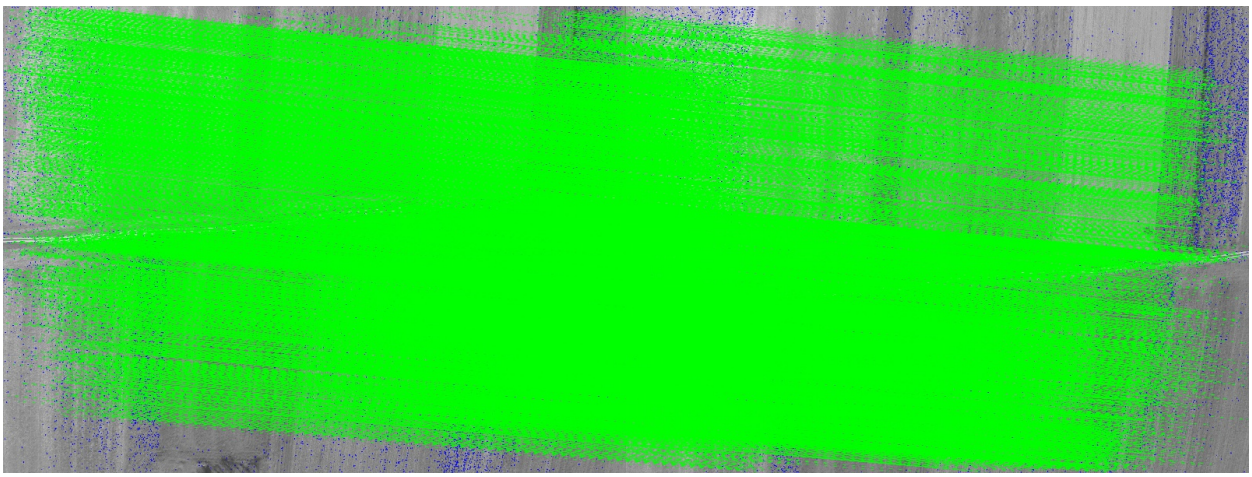
Sl. 3.3 - usporedba fotografija prije procesa pronalazaka podudarnosti

```
1.  matches = matcher.knnMatch(next_descs, trainDescriptors=base_descs, k=2)
```

Programski kod 7 - pronalazak podudarnosti

Programski kod 7 prikazuje način na koji se dolazi do pronalazaka podudarnosti. Ova metoda slijedno uzme jedan deskriptor iz fotografije, i usporedi ga sa svima iz druge fotografije. Rezultat je *k* deskriptora najbližih inicijalnom.

Proces pronalazaka podudaranja prikazan je na slici 3.4. Na slici 3.4 plave točkice predstavljaju sve značajke pronađene između dvije fotografije. Zelene linije prikazuju rezultat spajanja korespondentnih značajki. Može se primijetiti da neke značajke nisu spojene, to je posljedica toga što na drugoj fotografiji nisu pronađene značajke koje odgovaraju.



Sl. 3.4 - podudaranja dviju fotografija

Fotografije su izvorne veličine, dimenzija 3840x2160 piksela, stoga je normalno dobiti veliki broj podudaranja, pogotovo jer je preklapanje fotografija 80%. Dobiveni rezultat je potrebno filtrirati kako bi se uklonio višak podataka dobiven programskim kodom 6.

Iz filtriranog rezultata se izvlače ključne točke, točnije značajke, koje će se koristiti kod dobivanja matrice homografije. U programskom kodu 8 prikazan je način dobivanja ključnih točaka iz filtriranih podataka.

```
for match in matches_subset:
      kp1.append(base_features[match.trainIdx])
      kp2.append(next_features[match.queryIdx])

  p1 = np.array([k.pt for k in kp1])
  p2 = np.array([k.pt for k in kp2])
```

Programski kod 8 - način izvlačenja ključnih točaka

Matrica homografije dobivena je programskim kodom 9.

```
H, status = cv2.findHomography(p1, p2, cv2.RANSAC, 5.0)
```

Programski kod 9 - računanje homografije

Kao rezultat dobijemo transformacijsku matricu, tj. Matricu homografije i masku. Iz maske se mogu dobiti podaci koji nam govore koliko ima populacijskih podataka te koliko ima podudaranja.

Iz toga se može izračunati udio populacijskih podataka u podudaranjima. Iz izvornih fotografija, udio populacijskih podataka je bio veći od 50%.

Kada je matrica homografije izračunata, potrebno je pripremiti iduću fotografiju za spajanje s baznom. Fotografije se poravnavaju koristeći se matricom homografije tako da se podudaraju jedna s drugom. Kako bi se obje fotografije u cijelosti vidjele, potrebno je kreirati pozadinsku sliku na koju će se zalijepiti bazna fotografija i iduća fotografija.

```
enlarged_base_img = np.zeros((img_h, img_w, 3), np.uint8)
```

Programski kod 10 - kreiranje pozadinske slike

Gdje *img_h* i *img_w* predstavljaju maksimalnu visinu i širinu slike na koju je primijenjena matrica homografije. Na pozadinsku sliku se dodaje prvo bazna fotografija, zatim iduća u nizu.

```
enlarged_base_img = cv2.add(enlarged_base_img, base_img_warp,
    mask=np.bitwise_not(data_map),
    dtype=cv2.CV_8U)

final_img = cv2.add(enlarged_base_img, next_img_warp,
    dtype=cv2.CV_8U)
```

Programski kod 11 - dodavanje bazne i iduće fotografije u pozadinsku

base_img_warp i *next_img_warp* predstavljaju baznu i iduću fotografiju nakon primjene matrice homografije. Dodatno se još briše višak pozadinske slike, te se finalna fotografija reže na njene stvarne dimenzije. Proces zaslužan za to se može pronaći u izvornom kodu u prilogu.

Pseudokod zatim sprema finalnu fotografiju, poziva sam sebe i proces počine ponovno. No ovoga puta bazna slika je slika dobivena prethodnim procesom.

4. REZULTATI

U potpoglavlju 1.1 opisan je način prikupljanja fotografija za spajanje i njihovo pretprocesiranje kako bi se dobile fotografije pogodne za spajanje. U ovom poglavlju biti će prikazani rezultati za sve oblike pretprocesiranja fotografije. Također, biti će prikazani dobri i loši rezultati te će se dat usporedba i moguća rješenja za loše rezultate.

4.1. Proces prikupljanja podataka

Za snimanje fotografija korištena je kamera GoPro Hero 4 Black. Kamera je stabilizirana pomoću 3-osovinskog nosača sa žiroskopima za stabilizaciju u mjestu. Cijeli nosač je montiran na letjelicu Solo 3DR s maksimalnom duljinom leta od 20 minuta.

Za razvoj i pokretanje algoritma korišten je Apple MacBook Pro (early 2015) te su na njemu testirane performanse algoritma.

Proces prikupljanja podataka kreće od spajanja letjelice na pametni uređaj te planiranja misije. Letjelica zatim sama polijeće, pozicionira se i kreće sa snimanjem fotografija. Bitna stavka kod postavljanja misije je da se fotografije preklapaju s minimalno 70%, no zbog mogućih pogrešaka postotak preklapanja je postavljen na 80%.

Zbog toga što je kvaliteta fotografija izrazito bitna, fotografije su morale proći određeno pretprocesiranje. Na slici 4.1 prikazana je originalna fotografija zabilježena tokom prikupljanja podataka. Takva fotografija nije pogodna za spajanje zbog svoje velike distorzije uzrokovane lećom kamere, te se takva fotografija mora procesirati kako bi se uklonila sva izobličenja.

Sl. 4.1 - originalna zabilježena fotografija

Pokušana su tri procesa uklanjanja izobličenja. Prvi je bio programski na način da se preko šahovske ploče dobije matrica izobličenja leće te se pomoću te matrice ukloni izobličenje sa slike, taj proces nije davao dobre rezultate. Na slici 4.2 prikazana je fotografija na kojoj je izobličenje uklonjeno korištenjem podataka iz testiranja kamere šahovskom pločom. Crvene linije naknadno su dodane kako bi se mogla procijeniti kvaliteta uklanjanja izobličenja. Iz slike se može vidjeti da linije polja sijeku crvenu crtu što nije dobro.

Sl. 4.2 - uklanjanje izobličenja principom šahovske ploče

Drugi proces koji je korišten je uzorkovanje videa. Uzorkovanje videa bio je obećavajući proces jer se za uklanjanje izobličenja koristio GoPro studio. GoPro studio ima mogućnost uklanjanja izobličenja leće samo kroz video, stoga je bilo potrebno snimiti video polja(umjesto fotografija) te nakon uklanjanja izobličenja video uzorkovati kako bi se dobile fotografije s minimalno 70% preklapanja. Primjer uzorkovane fotografije može se vidjeti na slici 4.3. Može se vidjeti da je izobličenje u potpunosti uklonjeno. Programski kod za uzorkovanje videa nalazi se u prilogu.

Sl. 4.3 - uzorak fotografije iz videa

Sl. 4.4 - uklanjanje izobličenja u programu Lightroom

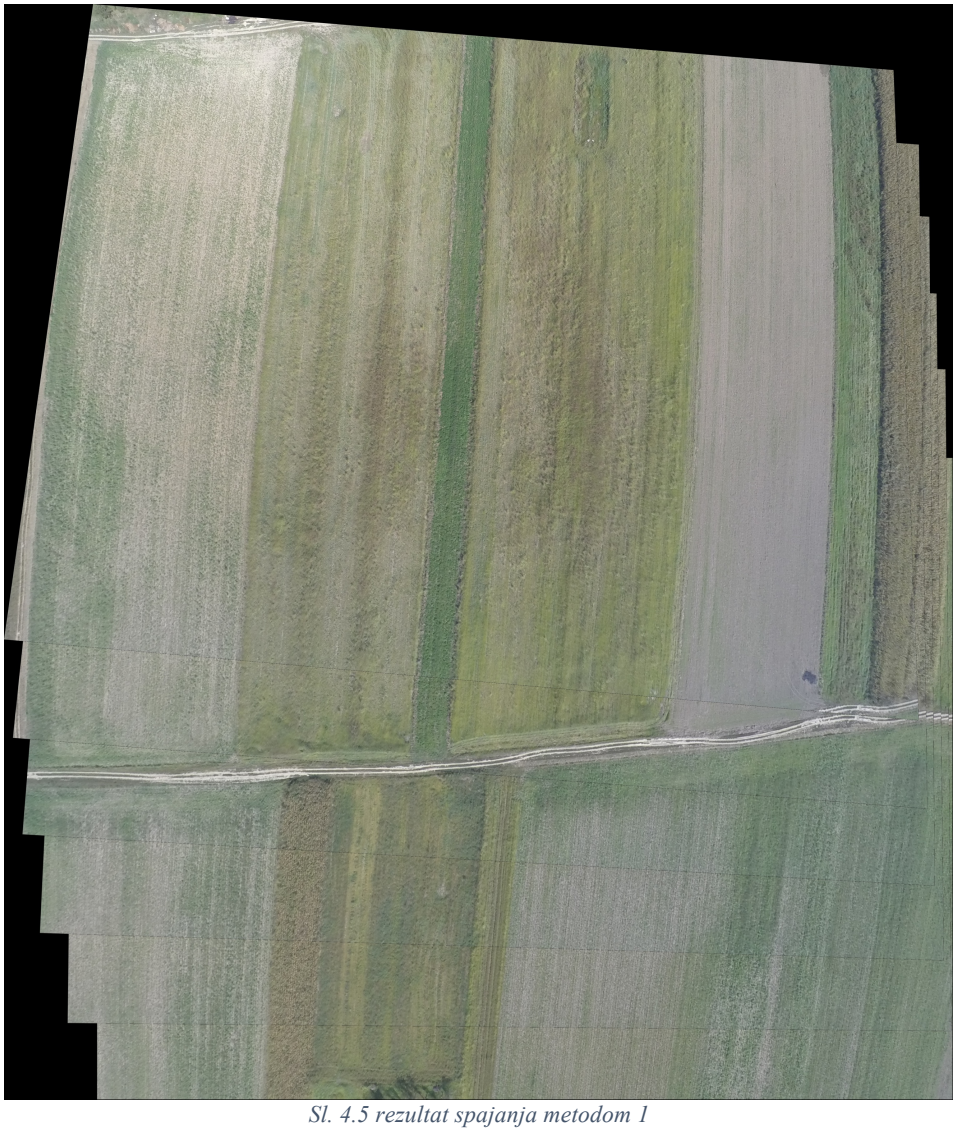
No, iako je ovaj proces dao fotografije bez izobličenja, fotografije su smanjene na rezoluciju 1280x720 piksela. Smanjenje rezolucije ne odgovara zahtjevima zadatka, stoga je potrebno pronaći još jedan način na koji se može ukloniti izobličenje, bez smanjenja rezolucije.

Treći proces koji je korišten za uklanjanje izobličenja je program Lightroom od Adobe-a. Lightroom u sebi ima profile većine GoPro kamera koji se mogu koristiti za uklanjanje izobličenja. Ono što je dobra strana Lightroom-a je da se rezolucija fotografija ne mora smanjiti kako bi se uklonila izobličenja. Rezultat uklanjanja izobličenja je prikazan slikom 4.4. Iako uklanjanje izobličenja nije savršeno kao kod prethodne metode, kvaliteta je i dalje na vrlo visokom nivou, a uz to rezolucija se nije morala smanjiti.

Pseudokod je pokretan na svim prikupljenim fotografijama i rezultati su prikazani u idućim potpoglavljima.

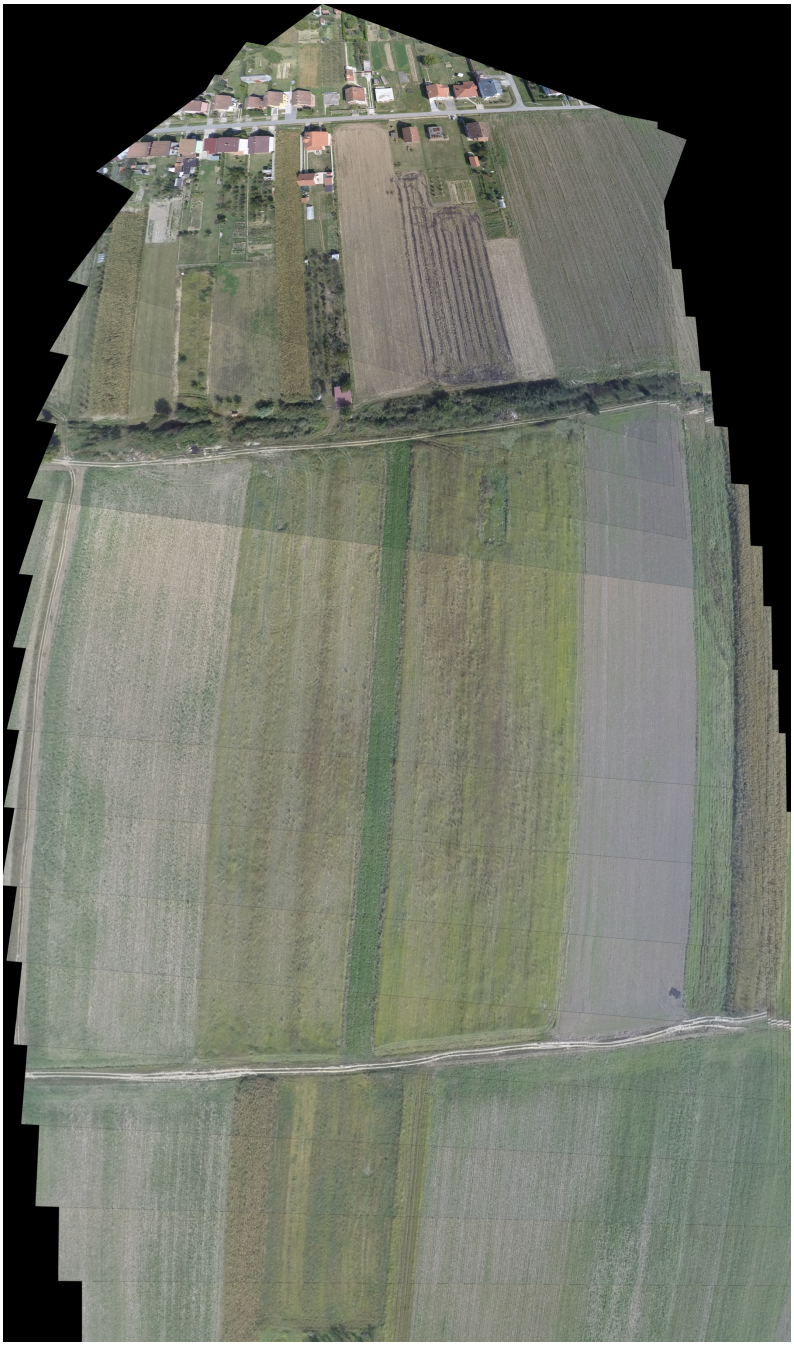
4.2. Rezultati pretprocesiranja kalibracijom kamere šahovskom pločom

Prva metoda koja je korištena za uklanjanje izobličenja bila je metoda računanja matrice izobličenja koristeći šahovsku ploču. Pomoću dobivenih rezultata iz fotografija je uklonjeno izobličenje. Rezultat uklanjanja izobličenja nije bio zavidan što je vidljivo na slici 4.2. Rezultat spajanja takvih fotografija prikazan je slikom 4.5.



Sl. 4.5 rezultat spajanja metodom 1

Na slici je vidljivo da je spajanje više-manje prošlo uredu, no s desne strane se može vidjeti kako spajanje i nije prošlo dobro. Razlog tomu je taj što na fotografijama nije u potpunosti uklonjeno izobličenje.

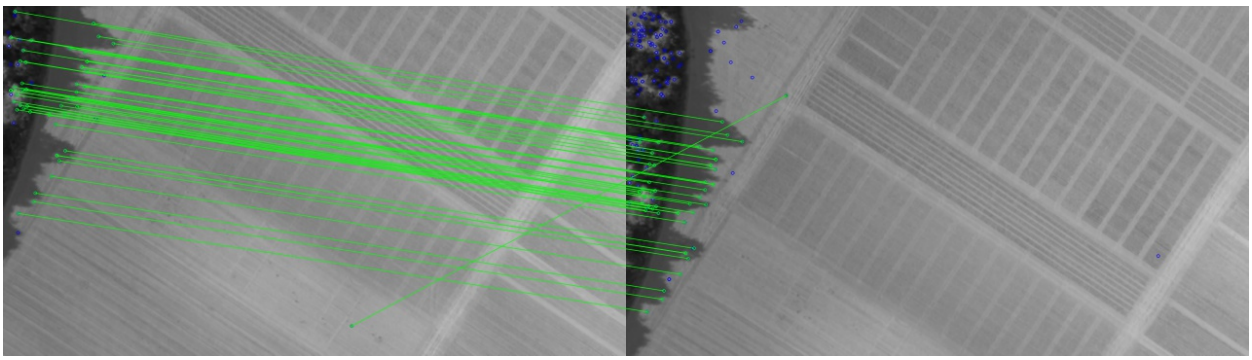


Sl. 4.6 rezultat spajanja metodom 1 s više fotografija

Na slici 4.6 se vidi rezultat korištenja proširenog seta fotografija, također može se vidjeti kako se zbog prisutnog izobličenja rezultantna fotografija smanjuje prema vrhu, te da rubovi fotografija odstupaju od prethodne fotografije.

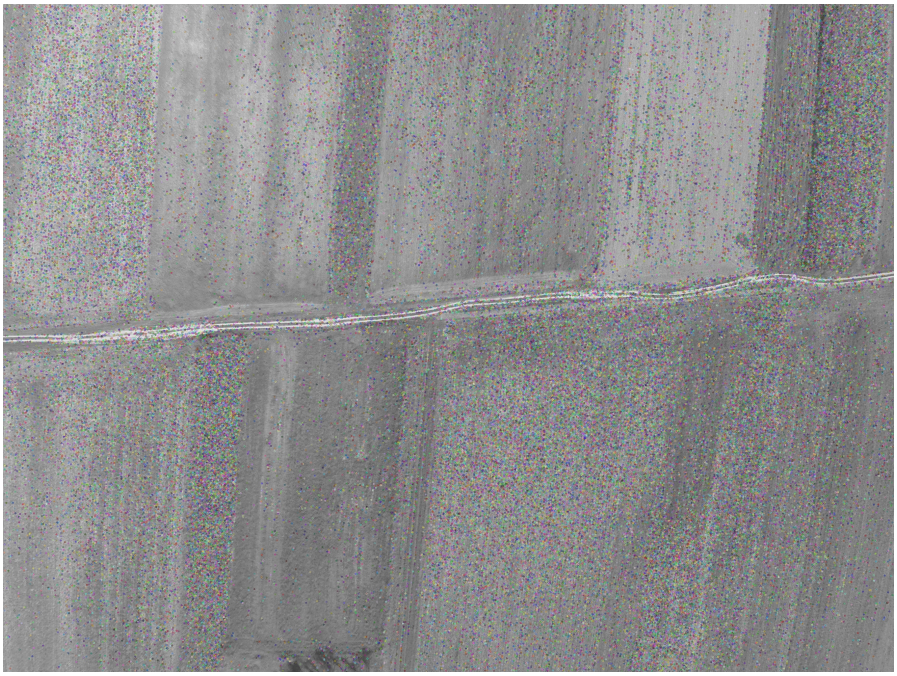
4.3. Rezultati spajanja fotografija metodom uzorkovanja videa

Kako su sve fotografije zabilježene GoPro Hero 4 Black kamerom, sva izobličenja su rezultat leće kamere. GoPro ima odličnu softversku podršku za obradu videozapisa, između ostalog i mogućnost uklanjanja izobličenja uzrokovanog lećom kamere. Stoga je druga metoda bila snimanje videa te zatim uzorkovanje tog videa kako bi se dobile fotografije koje se preklapaju minimalno 70%. Metodom pokušaj-pogreška došlo se do zaključka da je 2 sekunde optimalni interval uzorkovanja videa kako bi se dobilo potrebno preklapanje. Problem kod ovog pristupa je bio taj što je GoPro studio, koji je vršio uklanjanje izobličenja, također smanjivao rezoluciju videa.



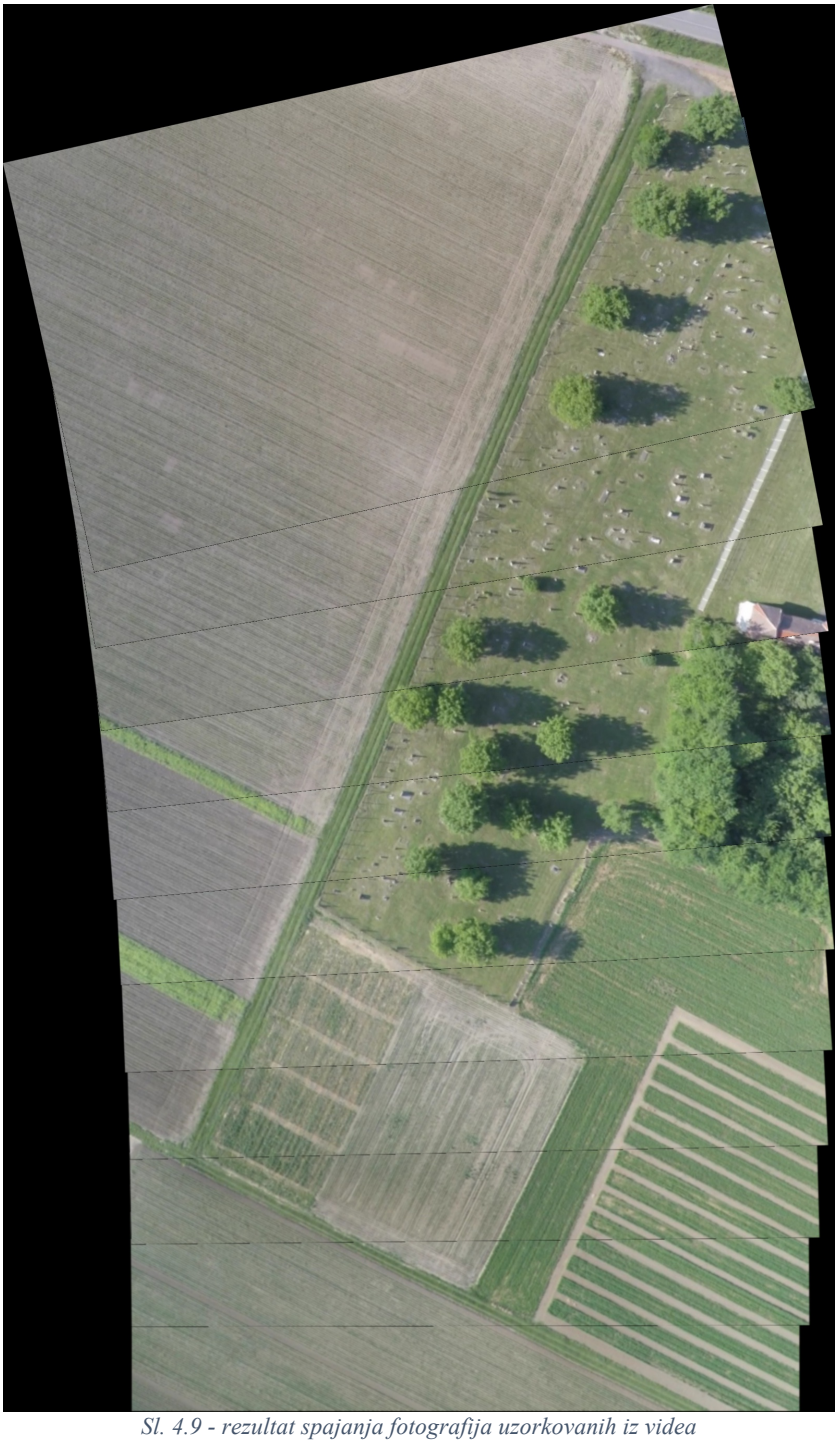
Sl. 4.7 detekcija značajki korištenjem metode 2

Zbog monotonosti fotografija (točnije polja), dolazi do situacije prikazane na slici 4.7. Značajke koje bi se nalazile na polju uopće nisu zabilježene i stoga spajanje fotografija je gotovo nemoguće kvalitetno odraditi. Također može se vidjeti da postoje i određene greške kod prepoznavanja odgovarajućih značajki. Na slici 4.8 može se vidjeti fotografija pune rezolucije te količina značajki koju je SIFT pronašao. Također, u ovoj metodi pokušano je u potpunosti izbjegavanje zamućivanja. Točnije, fotografije su uopće nisu zamućivale s ciljem da SIFT pronađe što više značajki.



Sl. 4.8 SIFT detektirane značajke

No, ukoliko na fotografijama postoje određene strukture i razlike koje SIFT može detektirati, rezultat će biti zadovoljavajući. Na slici 4.9 može se vidjeti primjer spajanja fotografija koje su uzorkovane iz videa.

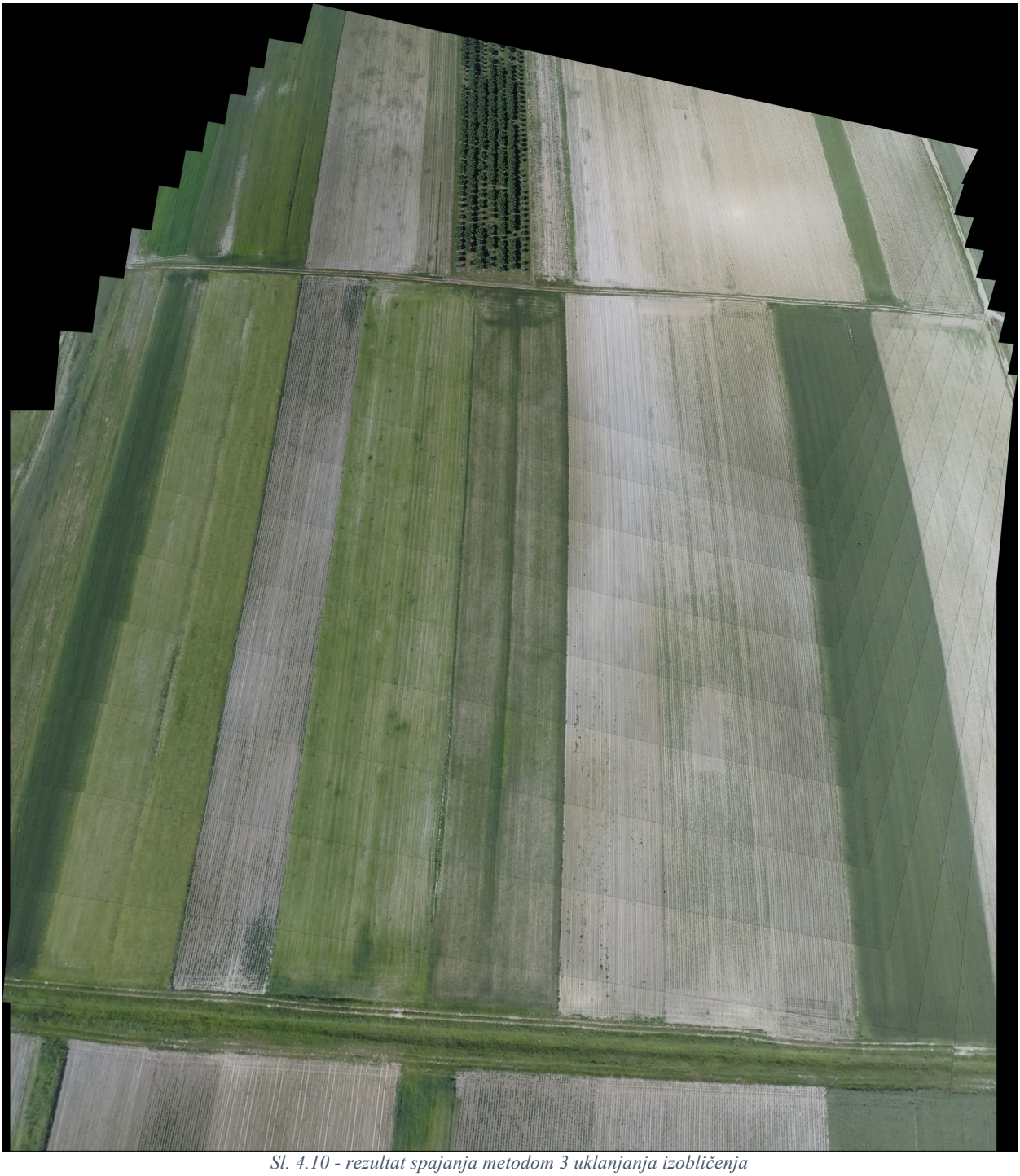


Sl. 4.9 - rezultat spajanja fotografija uzorkovanih iz videa

Iz razloga što je izobličenje u potpunosti uklonjeno, algoritam je spajanje izvršio odlično. Ponovno tu predstavljaju problem monotone fotografije koje nemaju puno razlika u sadržaju te stoga algoritmi za prepoznavanje ne mogu pronaći značajke.

4.4. Rezultati spajanja fotografija pretprocesiranih u Lightroom-u

Kako Lightroom ima mogućnost uklanjanja izobličenja iz fotografija odabran je kao treći izbor. Lightroom dolazi sa ugrađenim profilima za uklanjanje izobličenja s GoPro kamera što je bio veliki plus. Rezultati uklanjanja izobličenja mogu se vidjeti u potpoglavlju 4.1 na slici 4.4. Iako uklanjanje nije bilo savršeno kao kod videa, Lightroom je u velikom postotku uklonio izobličenje i pritom nije smanjivao rezoluciju fotografije.

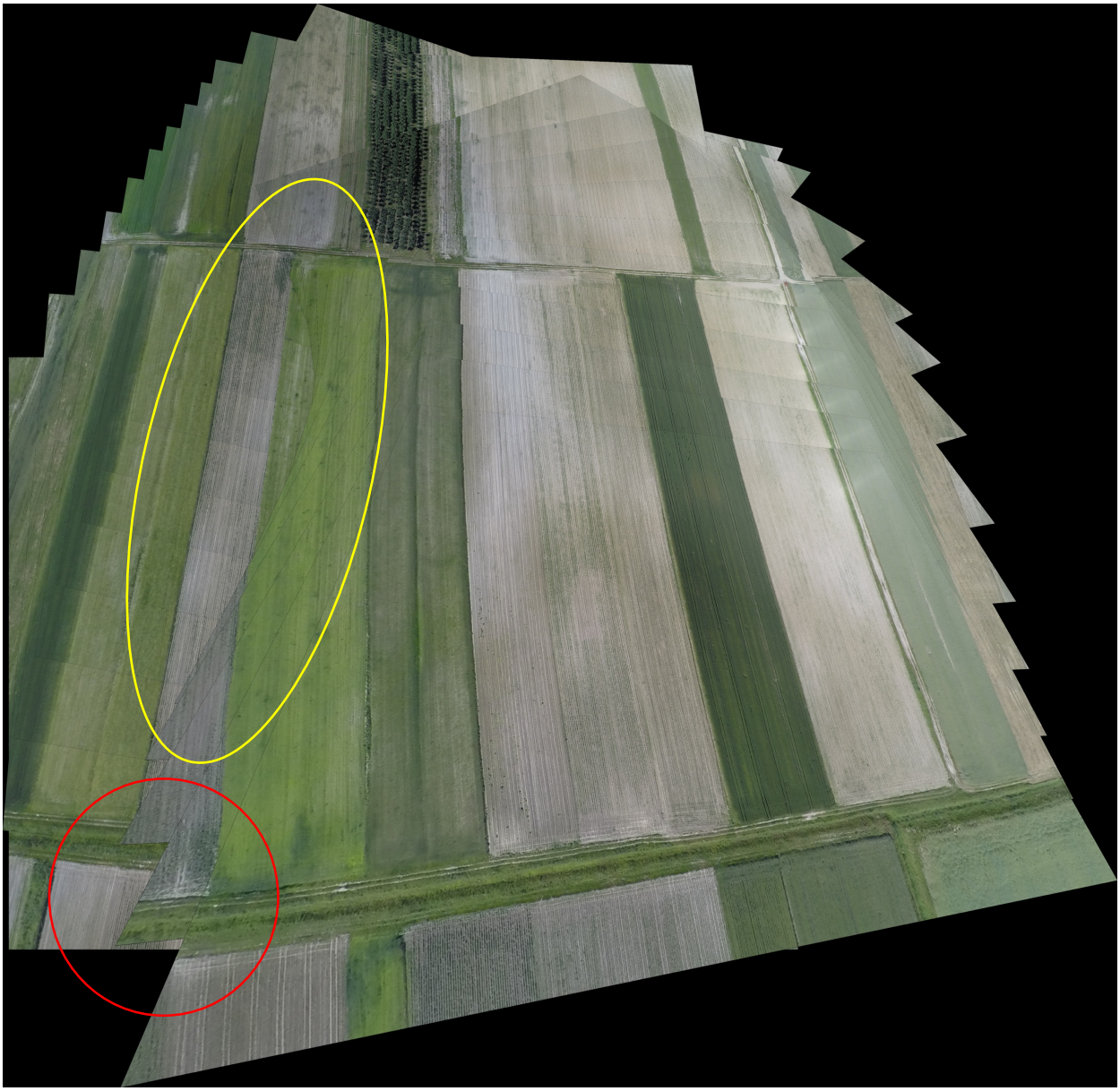


Sl. 4.10 - rezultat spajanja metodom 3 uklanjanja izobličenja

Rezultati spajanja mogu se vidjeti na slici 4.10. Rezultat je odličan i visoke kvalitete. Za razliku od rezultata prikazanih u potpoglavlju 4.2, rubovi sa slike 4.10 zanemarivo odstupaju. Uklanjanje izobličenja ovom metodom dalo je dosad najbolje rezultate po zahtjevima algoritma.

4.5. Pogreške i idejna rješenja

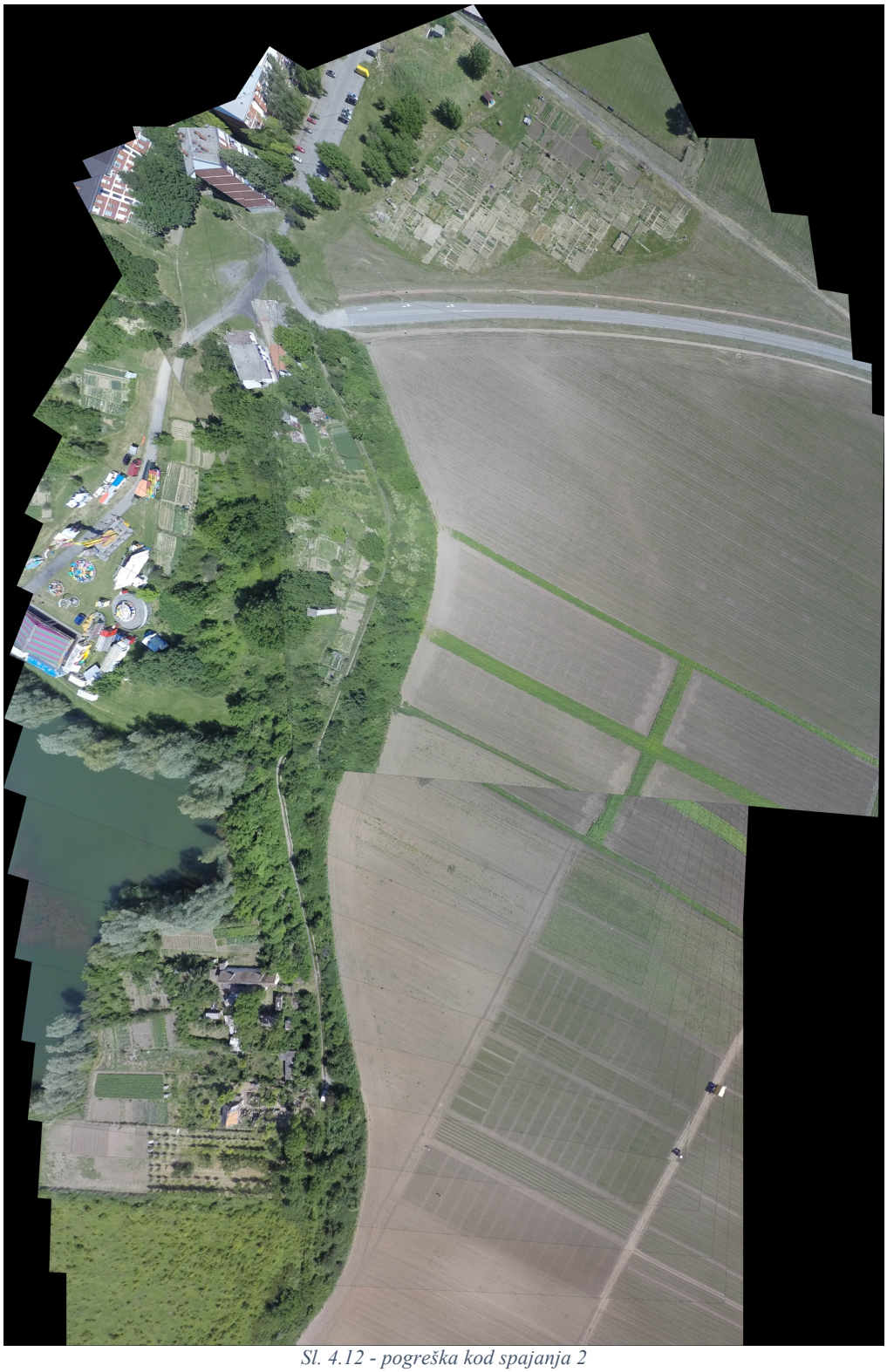
Rezultati prikazani u prethodnim potpoglavljima temelje se na malom broju izvornih fotografija. Ukoliko se pojavi pogreška kod spajanja, čak i malo odstupanje, to će se uvelike odraziti na svaku iduću iteraciju spajanja. Na slici 4.11 prikazana je pogreška spajanja.



Sl. 4.11 - pogreška kod spajanja

Fotografije koje su se pokazale kao kritične kod spajanja su rubne fotografije, odnosno fotografije koje su zabilježene tokom okretanja letjelice te su stoga izbačene iz seta izvornih

fotografija. Slika 4.11 također pokazuje pokušaj pseudokoda da ispravi pogrešku (zaokruženo žutom bojom) nastalu prilikom spajanja prethodnih koraka, nažalost, pseudokod nije uspio popraviti odstupanje. Iz tog razloga na dnu fotografije (zaokruženo crvenom) vidljivo je veliko odstupanje.



Sl. 4.12 - pogreška kod spajanja 2

Na slici 4.12 prikazan je još jedan primjer pogreške kod spajanja, no ovoga puta pseudokod nije pokušavao ispraviti pogrešku. Razlog tomu je taj što nije imao dovoljan broj značajki koji bi mu

dao mogućnost ispravka pogreške, kao što je to bio slučaj na slici 4.11. Kao jedno od rješenja, možda i najbolje, bi bilo uvesti detektore rubova kao što je Canny. Zbog strukture samih značajki na fotografijama, detektori rubova bi bili odličan dodatak algoritmu. Pretpostavka je da bi se uvođenjem detektora rubova uklonile pogreške.

5. ZAKLJUČAK

Kroz prethodna poglavlja dan je pregled korištenja modernih tehnologija u svrhu dobivanja velike količine informacija. Bespilotne letjelice, osim zabave, pružaju platformu za prikupljanje podataka na lokacijama koju se teško dostupne i daju uvid u okolinu iz jedne nove perspektive. Iako se u današnje vrijeme primarno koriste u svrhu snimanja različitih filmova, događaja, krajolika i zabave, pronašle su svoju ulogu i u drugim djelatnostima. Ovaj diplomski rad nastojao je dati uvod u mogućnosti bespilotnih letjelica te mogućnosti današnje tehnologije koje mogu dati pogled na okolicu iz jedne nove perspektive. Pogled na okolicu iz zraka može dati jako puno informacija. Cilj ovog rada bio je dati podlogu za budući razvoj ovakve tehnologiju u poljoprivredi. Fotografije snimljene iz zraka mogu dati informacije o usjevima i razvoju polja bez čovjekove interakcije s poljem. Stručnjaci mogu dobiti točne, realne i trenutne podatke o stanju cijelog i/ili niza polja u vrlo kratkom vremenu bez potrebe da budu fizički prisutni na lokaciji. Trenutna rješenja za ovu problematiku su rijetka, a postojeća su iznimno skupa. Razvoj ovog algoritma također bi dao bazu u razvoj jeftinijih alata za kvalitetno spajanje fotografija ne samo za korištenje u poljoprivredi, nego i elektroprivredi za nadzor i pregled dalekovoda.

LITERATURA

[1] R. Cupec, Lokalni deskriptori, Robotski vid, materijali za predavanja
[2] D. G. Lowe, Distinctive Image Features from Scale-Invariant Keypoints, University of British Columbia, Vancouver, B.C., Canada, 2004.
[3] M. Čuljak, Stabiliziranje video zapisa pribavljenog iz lebdeće letjelice, diplomski rad broj 365, Fakultet Elektrotehnike i računarstva, Zagreb, 2012.
[4] M. Zuliani, RANSAC for Dummies, str. 12-16, Vision Research Lab, 2014

SAŽETAK

Ovaj diplomski rad daje jednostavno rješenje za spajanje velikog broja fotografija visoke rezolucije u jednu. Pseudokod za spajanje fotografija temelji se na SIFT algoritmu za prepoznavanje značajki te RANSAC algoritmu za pronalazak kvalitetnih podudarnosti između dvije fotografije. Podaci od navedena dva algoritma se naknadno koriste kako bi se fotografije spojile u jednu. Na početku rada dana je teorijska osnova koja detaljno opisuje korištene metode, te nakon toga dana je implementacija uz detaljno objašnjenje funkcionalnosti cijelog programa.

Ključne riječi: sift, stitching, ransac, spajanje, fotografije, bespilotne letjelice

ŽIVOTOPIS

Dino Repac je student 5. godine Diplomskog studija Računarstva na modulu Informacijske i podatkovne znanosti. Zanima ga razvoj komercijalnih i enterprise web aplikacija, cloud rješenja, te obrada slike i računalni vid. Trenutno je zaposlen kao full-stack razvojni programer u tvrtki Mono gdje primarno radi na razvoju enterprise web aplikacija s Microsoft-ovim tehnologijama.

PRILOG

```
1.  import cv2
2.  import numpy as np
3.  import math
4.
5.  cap = cv2.VideoCapture('SampleVideo/30052017_midday.mov')
6.  fps = cap.get(cv2.CAP_PROP_FPS)
7.  framecount = 0
8.  counter = 0
9.  interval = 3
10.
11. while(cap.isOpened()):
12.     success, image = cap.read()
13.     framecount += 1
14.
15.     if framecount == int(math.ceil(fps * interval)): # interval determines how often
    should algorithm extract frame.
16.         counter += 1
17.         print "writing " + str(counter)
18.         cv2.imwrite('SampleImages/video_samples/30052017/'+str(counter)+'_vidsmpl.jpg
    ', image)
19.         framecount = 0
20.
21.     if cv2.waitKey(1) & 0xFF == ord('g'):
22.         break
23.
24. cap.release()
25. cv2.destroyAllWindows()
```

Prilog 1 - programski kod za uzorkovanje videa