Spajanje slika i videa u panoramu

Miholek, Branka

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Department of Mathematics / Sveučilište Josipa Jurja Strossmayera u Osijeku, Odjel za matematiku**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:126:334580

Sveučilište J.J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni preddiplomski studij matematike i računarstva

Branka Miholek

Spajanje slika i videa u panoramu

Završni rad

Osijek, 2020.

Sveučilište J.J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni preddiplomski studij matematike i računarstva

Branka Miholek

Spajanje slika i videa u panoramu

Završni rad

Mentor: izv. prof. dr. sc. Domagoj Matijević

Komentor: Ivica Skokić (Opservatorij Hvar,
Geodetski fakultet, Sveučilište u Zagrebu)

Osijek, 2020.

Sažetak

U ovome radu opisujemo kako se radi spajanje slika, odnosno frameova iz videa. Najprije za svaku ulaznu sliku pronalazimo ključne točke, deskriptore i feature pomoću algoritma KAZE. Zatim pronalazimo matcheve pronađenih ključnih točaka, deskriptora i featurea između slika koristeći Brute-Force matcher ili FLANN matcher. Ako imamo dovoljno matcheva, korištenjem algoritma RANSAC računamo matricu homografije uz pomoć koje transformiramo slike. Na kraju spajamo transformirane slike i dobivamo panoramu.

Ključne riječi

ključne točke, deskriptori, featuri, matrica homografije

Abstract

In this paper we describe how image and video stitching is done. Firstly, we use KAZE algorithm to find keypoints, descriptors and features of every input image. Next, we find matching keypoints, descriptors and features between images using Brute-Force matcher or FLANN matcher. If we have enough matches, we use RANSAC algorithm to compute homography matrix which helps us to warp images. In the end, we connect the warped images and get the output panorama.

Key words

keypoints, descriptors, features, homography matrix

Sadržaj

Uvod **1**

1 Spajanje u panoramu **2**

1.1 Detekcija ključnih točaka 2
1.2 KAZE algoritam 5
1.3 Pronalaženje matcheva 6
1.4 Brute-Force matcher 7
1.5 FLANN matcher 8
1.6 k-Nearest Neighbour algoritam 8
1.7 Lowe's ratio test 9
1.8 Računanje matrice homografije 10
1.9 RANSAC 12
1.10 Spajanje 13
1.11 Primjeri 14
1.12 Spajanje više slika 18
1.13 Spajanje slika s nepoznatim poretkom 20
1.14 Spajanje videa 20

2 Spajanje u 360° prikaz **22**

3 Rezultati spajanja slika generiranih u Blenderu **26**

4 Zaključak **28**

Literatura **29**

Uvod

Spajanje slika proces je kombiniranja više slika čija se vidna polja preklapaju u svrhu dobivanja panorame. Cilj ovog rada je spajanje slika, odnosno videa, primljenih iz kamera koje se nalaze na vozilu. Kamere su raspoređene tako da prekrivaju cijelu (polu)sferu vozila i vidna polja im se preklapaju. Željeni rezultat je širokokutni prikaz ili prikaz od 360° koji prikazuje okolinu vozila. Paket koji koristimo, OpenCV, već ima gotovu klasu za spajanje slika u panoramu, ali nju nismo koristili jer su nam ovdje potrebne homografije za svaku kameru kako bismo mogli spajati videe, te smo stoga razvili vlastitu klasu za spajanje slika koristeći programski jezik Python. Ovaj rad napravljen je u suradnji s tvrtkom Orqa.

Tehnike spajanja slika mogu se podijeliti na 2 pristupa : direktni i feature-based pristup. Direktni pristup uspoređuje intenzitete svih piksela slika. Feature-based pristup pokušava odrediti odnos između slika preko distinktnih featurea ekstrahiranih iz slika. Ovaj je pristup brži i robustniji od direktnog [3], ali direktni je ponekad točniji jer koristi sve informacije o slici [1]. U ovom radu koristi se feature-based pristup.

Spajanje slika dio je znanosti koja se naziva computer vision. Computer vision je područje umjetne inteligencije koje unosi razumijevanje našeg vizualnog svijeta u računala [4]. Eksperimentiranje s computer visionom započelo je krajem 50-ih godina prošlog stoljeća, kada su se koristile neuronske mreže za detekciju rubova objekata u svrhu sortiranja u krugove, kvadrate, itd. U 70-ima napravljen je alat koji interpetira natipkan ili ručno napisan tekst. Korišten je kao interpretator teksta za slijepe. Programi za prepoznavanja ljudskih lica počeli su se praviti 90-ih godina [4].

Danas se computer vision nastavlja razvijati. Razvoj su ubrzali mobiteli koji u sebi imaju ugrađene kamere, računala koja su jača, dostupnija i povoljnija te novi algoritmi poput konvolucijskih neuronskih mreža koje mogu bolje iskoristiti kompjuterske mogućnosti. Iz tog razloga današnji sustavi točnije i brže reagiraju na vizualne inpute [4]. Computer vision se danas koristi u brojnim industrijama kao što su automobilska industrija, zdravstvo, trgovačka industrija, agrokultura [5].

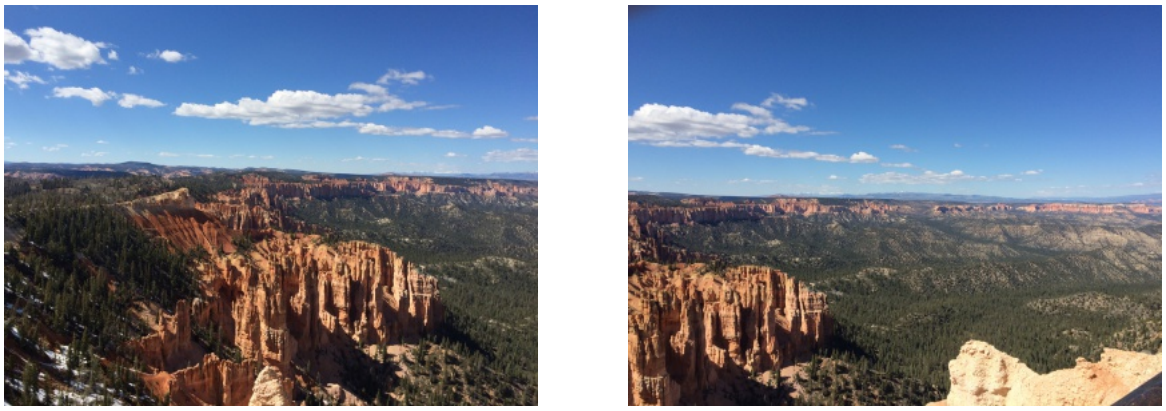
1 Spajanje u panoramu

Program za spajanje slika započinjemo učitavanjem potrebnih biblioteka u Pythonu:

```
1 import numpy as np
2 import cv2
```

Slike možemo učitati na sljedeći način:

```
1 imageA = cv2.imread('LeftImage.png')
2 imageB = cv2.imread('RightImage.png')
```



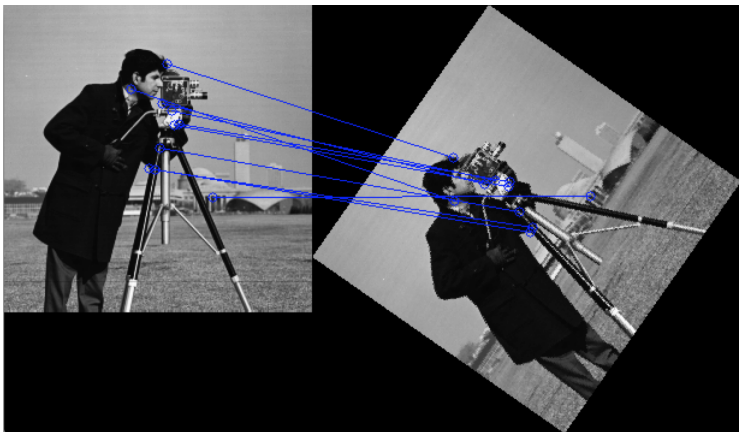
(a) Lijeva slika (b) Desna slika

Slika 1: Slike koje želimo spojiti

Spajanje slika u panoramu sastoji se od 4 glavna koraka: detekcija ključnih točaka, deskriptora i featurea na slikama, pronalaženje matcheva između slika, računanje matrice homografije i upotreba transformacija na slikama kako bi se mogle spojiti. [6]

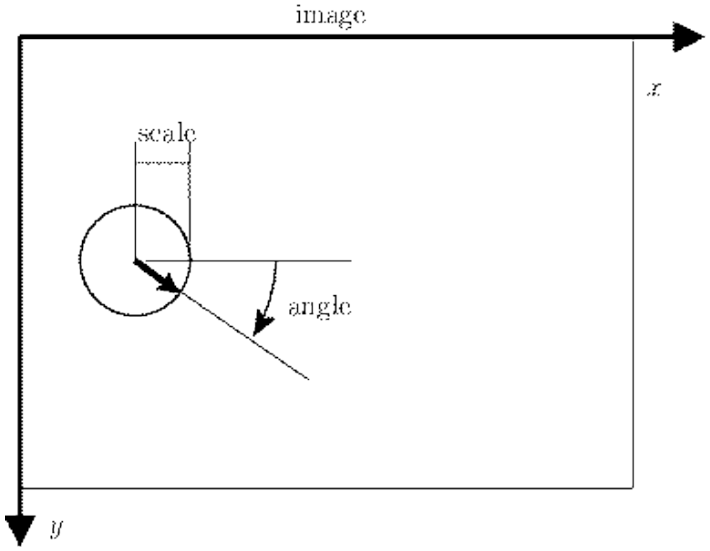
1.1 Detekcija ključnih točaka

Ključne točke (eng.keypoints) su točke koje definiraju nešto što se ističe na slici. Ne mijenjaju se ako se na slici primjene afine transformacije ili ako sliku distortiramo tako da npr. djelujemo s homografijom [7]. Afine transformacije su one koje očuvaju kolinearnosti i omjere udaljenosti. To su rotacija, translacija, skaliranje, itd [8]. Invarijantnost ključnih točaka na afine transformacije omogućuje nam detekciju istih točaka, a time i objekata, na nizu različitih slika istog objekta.



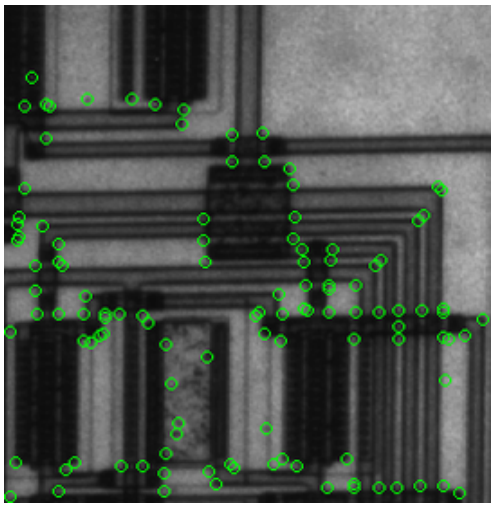
Slika 2: Ključne točke ostaju iste nakon rotacije [7]

Svaka od ključnih točaka ima svoj deskriptor koji ju opisuje, npr. sadrži njezinu magnitudu i orijentaciju koje se dobiju tako da se pretraže susjedni pikseli točke.



Slika 3: Ključna točka s magnitudom i orijentacijom

Featuri slike odnose se na uzorke i strukture na slici koje se ističu u odnosu na njihovu okolinu, npr. po teksturi, boji ili intenzitetu. Primjeri featurea slike su mrlje, rubovi i vrhovi.



Slika 4: Detekcija vrhova [9]

Feature detektor	Rubovi	Kutovi	Mrlje
Canny	+	-	-
Sobel	+	-	-
Kayyali	+	-	-
Harris, Stephens	+	+	-
SUSAN	+	+	-
Shi, Tomasi	-	+	-
Level curve curvatore	-	+	-
FAST	-	+	+
Laplacian of Gaussian	-	+	+
Difference of Gaussians	-	+	+
Determinant of Hessian	-	+	+
MSER	-	-	+
PCBR	-	-	+
Grey-level blobs	-	-	+

Tablica 1: Algoritmi za detekciju featurea [10]

Za ovaj korak koriste se algoritmi kao što su KAZE, AKAZE, ORB, BRIEF, SIFT, SURF.

SIFT (Scale-Invariant Feature Transform) je jedan od prvih i najpoznatijih algoritama za detekciju featurea [11]. On izvlači ključne točke invarijantne na rotaciju i skaliranje kreirajući prostor skala u koji se spremaju slike koje se dobiju oduzimanjem dviju slika konvolviranih Gauss kernelom različitih veličina (blur filter). Time se dobije detekcija ključnih točaka na više različitih prostornih skala. Zatim se promatra okolina ključnih točaka na svim skalama te se određuje smjer i magnituda iz histograma i odnosa intenziteta okolnih piksela. Od tih informacija se formira deskriptor za svaku ključnu točku.

KAZE algoritam je naprednija verzija SIFT algoritma koji koristi nelinearni prostor skala što omogućava bolju lokalizaciju ključnih točaka.
U ovom radu smo se odlučili za KAZE jer je noviji i bolji algoritam od SIFT-a te je open-source za razliku od SIFT-a koji je patentiran.

1.2 KAZE algoritam

KAZE je algoritam za detekciju i deskripciju 2D featurea u prostorima s nelinarnom skalom.
Za danu sliku gradi se prostor s nelinearnom skalom koristeći AOS tehnike i VCD (difuziju promjenjive vodljivosti) [12]. To ćemo napraviti tako da diskretiziramo skalni prostor u logaritamske korake raspoređene u redu od O oktava i S pod-levela. Setovi oktava i pod-levela su identificirani s diskretnim oktavnim indeksom o i pod-levelnim indeksom s, a ti indeksi su mapirani u odgovarajuću skalu σ preko sljedeće formule:

$$\sigma_i(o,s) = \sigma_0 2^{o+s/S}, o \in [0,..,O-1], s \in [0,..,S-1], i \in [0,..,N],$$

gdje je σ_0 bazni skalni level, a N ukupni broj filtriranih slika [12].
Sada konvertiramo skup diskretnih levela skale iz pikselnih mjernih jedinica σ_i u mjernu jedinicu vremena jer je nelinarno difuzno filtriranje definirano u terminima vremena. Koristimo mapiranje :

$$\sigma_i \to t_i, \quad t_i = \frac{1}{2}\sigma_i^2, \quad i = \{0...N\}$$

kako bismo dobili skup vremena evolucija iz kojeg gradimo prostor s nelinearnom skalom [12].
Iz slike računamo histogram gradijenta slike i dobivamo parametar kontrasta k preko automatskog procesa opisanog u [12]. Tada, uz k i set evolucijskih vremena t_i, direktno možemo izgraditi prostor s nelinearnom skalom preko iterativnog načina koristeći AOS [12]:

$$L^{i+1} = \left(I - (t_{i+1} - t_i)\sum_{l=1}^{m} A_l(L^i)\right)^{-1} L^i.$$

Detekcija 2D featurea
Detektiramo 2D feature koji pokazuju maksimum od determinante Hessijana kroz prostor s nelinearnom skalom.
Za detekciju ključnih točaka, računamo determinantu Hessijana na više levela skale. Za višerazmjernu feature detekciju, set diferencijalnih operatora mora biti normaliziran obzirom na skalu:

$$L_{Hessian} = \sigma^2(L_{xx}L_{yy} - L_{xy}^2),$$

gdje je (L_{xx}, L_{yy}) horizontalna i vertikalna derivacija drugog reda, a L_{xy} cross derivacija drugog reda. Derivacije su aproksimirane koristeći 3×3 Scharr filtere [12]. Za dani skup filtriranih slika iz prostora s nelinarnom skalom L^i, analiziramo odgovor detektora na različitim levelima skale σ_i i tražimo maksimum u skalnoj i spacijalnoj lokaciji. Pozicija ključne točke je procijenjena sa sub-pikselnom točnošću [12].

Traženje dominantne orijentacije

Za pronalazak deskriptora invarijantnih s obzirom na rotaciju, potrebno je procijeniti dominantnu orijentaciju u lokalnom susjedstvu centriranom na lokaciji ključne točke [12]. Dominantnu orijentaciju dobijemo tako da pretražimo kružno područje određenog radijusa oko ključne točke i računamo derivacije uzoraka iz tog područja pomoću Gaussiana centriranog na ključnoj točki. Zatim sumiramo derivacije unutar kružnog isječka koji pokriva kut of $\pi/3$ [12].

Izgradnja deskriptora

Za izgradnju deskriptora koristimo detektirane feature i dominantne orijentacije ključnih točaka. Konačni vektor deskripcije veličine 64 normaliziran je u jedinični vektor kako bi se postigla invarijantnost s obzirom na kontrast [12].

KAZE algoritam je već implementiran kao funkcija u OpenCV paketu koju i mi koristimo u ovom radu. Primjer korištenja KAZE algoritma u Pythonu:

```
1 def detectAndDescribe(image):
2
3         descriptor = cv2.KAZE_create()
4
5         (kps, features) = descriptor.detectAndCompute(image, None)
6
7         kps = np.float32([kp.pt for kp in kps])
8
9         return (kps, features)
```

1.3 Pronalaženje matcheva

Nakon što za svaku sliku dobijemo ključne točke i njihove feature, trebamo ih usporediti i pronaći zajedničke točke.

Potrebno je pronaći najmanje 4 matcheva, odnosno zajedničkih točaka, da se može izračunati

Slika 5: Matchevi

matrica homografije i odrediti transformacija jedne slike na drugu.
Matcheve odredimo koristeći matcher (Brute-Force ili FLANN) i kNN (k-Nearest Neighbour) algoritam.

1.4 Brute-Force matcher

Brute-Force matcher radi na način da provjerava podudarnost svakog featurea jedne slike sa svakim featureom druge slike. Za jedan feature prve slike najboljim matchom smatrat će feature koji mu je najbliži. Ako želimo dobiti više od 1 najboljeg matcha, koristimo funkciju BFMatcher.knnMatch() koja vraća k najboljih matcheva [13].

```
1 matcher = cv2.DescriptorMatcher_create("BruteForce")
2 #ili
3 matcher = cv2.BFMatcher(cv2.NORM_L1)
4
5 rawMatches = matcher.knnMatch(featuresA, featuresB, 2)
```

Kada koristimo Brute-Force matcher možemo odabrati koju ćemo formulu za udaljenost koristiti. Po defaultu koristi se Euklidska norma, ali za float based algoritme KAZE, SIFT i SURF može se koristiti i norma 1, dok se za binary string based algoritme poput ORB-a, BRIEF-a i BRISK-a koristi Hammingova udaljenost [13].

Može se odabrati da Brute-Force matcher samo vrati matcheve koji su međusobno najbolji, odnosno da provjerava da je neki match najbolji za oba featurea. Ovo je dobro za koristiti umjesto Loweovog ratio testa [13].

Za crtanje matcheva koristi se funkcija cv2.drawMatches(), ako želimo nacrtati po jedne najbolje matcheve, ili funkcija cv2.drawMatchesKnn(), ako želimo nacrtati k najboljih mat-

cheva [13].

Slika 6: Primjer rezultata funkcije cv2.drawMatches() za prvih 10 matcheva [13]

1.5 FLANN matcher

FLANN (Fast Library for Approximate Nearest Neighbors) uglavnom se koristi kada imamo veliki skup podataka jer je tada brži od Brute-Force matchera [13].

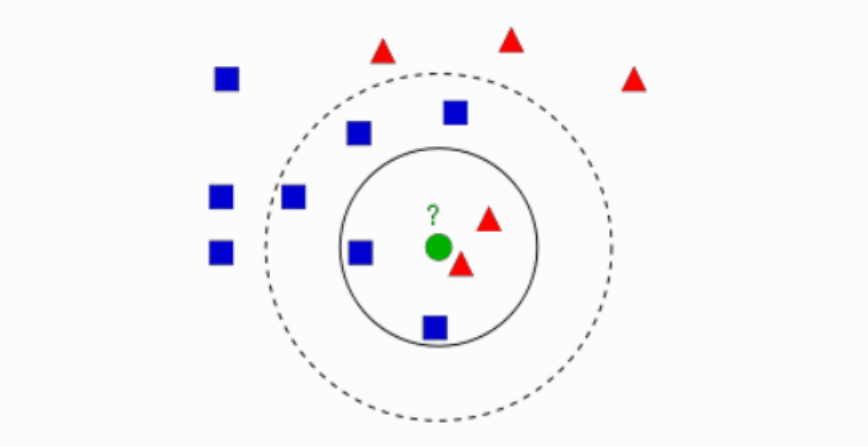
FLANN matcheru prosljeđujemo parametre kojima specificiramo koji algoritam treba koristiti, parametre tog algoritma te broj rekurzivnih prolazaka. Što je taj broj veći, dobiti ćemo bolji rezultat, ali će nam uzeti više vremena [13].

```
# za KAZE, SIFT, SURF
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)

#ili

# za ORB
index_params= dict(algorithm = FLANN_INDEX_LSH,
                   table_number = 6,
                   key_size = 12,
                   multi_probe_level = 1)

matcher = cv2.FlannBasedMatcher(index_params,search_params)
```

1.6 k-Nearest Neighbour algoritam

```
rawMatches = matcher.knnMatch(featuresA, featuresB, 2)
```

kNN je klasifikacijski algoritam za nadzirano učenje [14]. Problem klasifikacije objašnjen je preko slike 7.



Slika 7: Podaci su podijeljeni u dvije klase: plavi pravokutnici i crveni trokuti. Pitanje je u koju klasu staviti novi zeleni krug. [14]

Zeleni krug možemo klasificirati na više načina. Njegov najbliži susjed je crveni trokut pa bismo ga stoga mogli klasificirati kao crveni trokut. Ovu metodu zovemo "Nearest-Neighbour" [14]. No, većina podataka oko zelenog kruga su plavi pravokutnici pa bismo ga mogli klasificirati u plave pravokutnike. Metoda koja se naziva "k-Nearest Neighbour" provjerava k najbližih pojedinaca podatka kojeg želimo klasificirati [14]. Postoji još jedna metoda koja se naziva "modified kNN" koja isto provjerava k najbližih pojedinaca, ali svakome pridodaje i vrijednost udaljenosti te na taj način može odrediti klasifikaciju i ako imamo jednak broj pojedinaca različitih klasa u okolini našeg podatka [14].

1.7 Lowe's ratio test

Loweov ratio test je metoda za filtriranje matcheva ključnih točaka. Ako su dva najbolja matcha skoro jednako dobri, ova ih metoda eliminira [15]. Ukratko, za svaku ključnu točku imamo 2 najbolja matcha, gdje je match bolji što ima manju udaljenost među ključnim točkama, a Loweov test uspoređuje te dvije udaljenosti uz pomoć omjera. Ako nisu dovoljno različite, ne koristimo više tu ključnu točku [15].

Neka je kpsA skup ključnih točaka prve slike, a kpsB skup ključnih točaka druge slike. Algoritam za matchanje za svaku je ključnu točku u skupu kpsA tražio najbliži match u kpsB [15]. Mogli bismo odrediti neki prag i eliminirati sve parove ključnih točaka gdje je njihova udaljenost veća od vrijednosti praga. No, može se dogoditi da dvije ključne točke imaju malu udaljenost jer dijelovi njihovih deskriptora imaju slične vrijednosti, a možda ti

dijelovi nisu ni bitni za matchanje. Taj problem možemo riješiti dodavanjem težina svim varijablama deskriptora obzirom na njihovu važnost [15], ali možemo i koristiti Loweovo rješenje koje je jednostavnije.
kNN matcher vratio nam je 2 najbolja matcha, odnosno za svaku ključnu točku iz kpsA imamo dvije ključne točke iz kpsB. Ključna točka iz kpsA ne može imati više od jedne ekvivalentne ključne točke iz kpsB pa znamo da je samo jedan od matcheva koji nam je vratio kNN matcher dobar. Za dobar match možemo uzeti onaj koji ima manju udaljenost, ali ako oba imaju sličnu udaljenost i ne možemo odrediti koji je pravi, odbacujemo oba matcha, oni nam ne daju nikakve dodatne informacije [15]. To znači da između 2 odabrana najbolja matcha, mora postojati dovoljno velika razlika u udaljenosti kako bismo mogli odrediti koji od njih je dobar, a koji krivi. Sada provjeravamo ratio, odnosno omjer te dvije udaljenosti.

```
1 for m in rawMatches:
2     if len(m) == 2 and m[0].distance < m[1].distance * ratio:
3         matches.append((m[0].trainIdx, m[0].queryIdx))
```

kNN matcher vratio nam je matcheve u poretku od najboljeg do najlošijeg pa je vrijednost m[0].distance manja od m[1].distance. Želimo provjeriti koliko je prva udaljenost manja od druge udaljenosti pa množimo drugu udaljenost s varijablom ratio, $ratio \in [0, 1]$, što će ju smanjiti. Ako je prva udaljenost idalje manja od druge udaljenosti, taj je match dobar [15].

1.8 Računanje matrice homografije

Matrica homografije kvadratna je matrica dimenzije 3×3. Ona mapira točke na jednoj slici uz podudarajuće točke na drugoj slici (Slika 8). Točke moraju biti na istoj ravnini u prostoru. Homografija nam daje transformaciju te ravnine s jedne slike na drugu.
Na slici 8 vidimo podudarajuće točke (x_1, y_1) na lijevoj i (x_2, y_2) na desnoj slici.

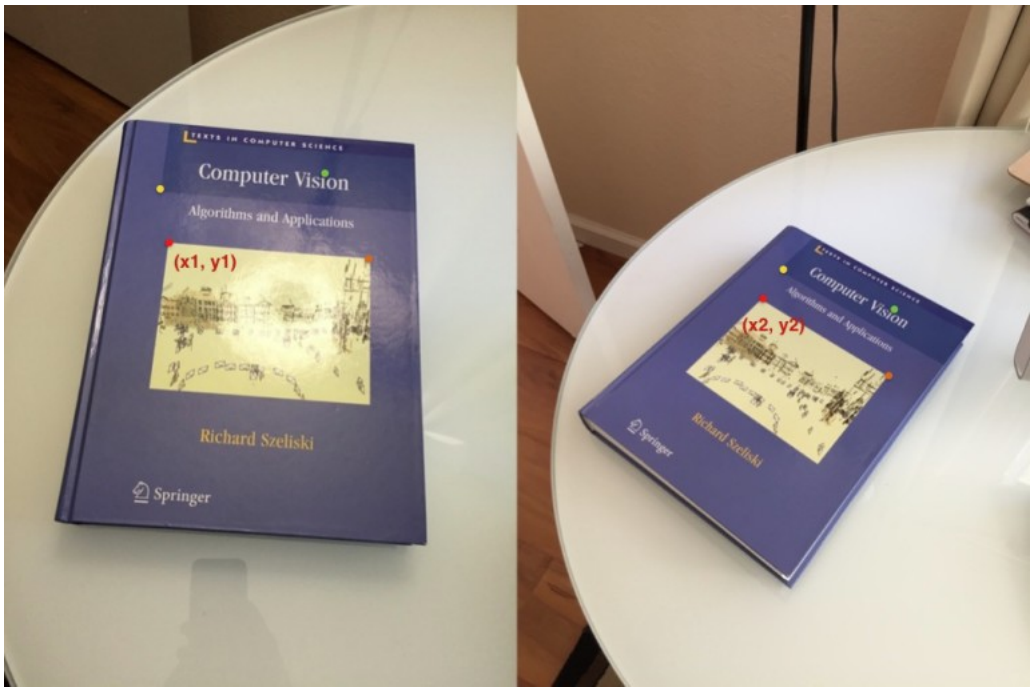
Matrica homografije,

$$H = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix},$$

mapira točke ovako [16]:

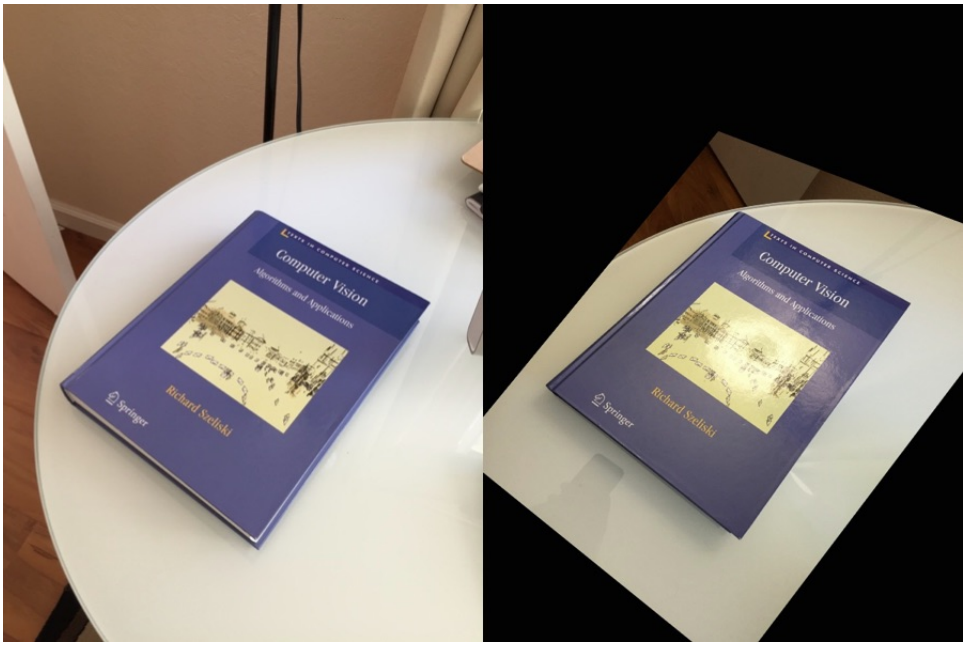
$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = H \cdot \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}.$$

Uz pomoć matrice homografije, možemo transformirati slike, tj. međusobno ih prilagoditi kako bismo ih zatim mogli spojiti.



Slika 8: Postoji homografija između ovih slika [16]

U slučaju slike 8, transformacijom uz pomoć homografije dobit ćemo sliku 9, a nakon izračuna homografije slika 1 i transformacije slike 1b dobijemo sliku 10.



Slika 9: Uz pomoć homografije, transformirali smo desnu sliku [16]

(a) Lijeva slika

(b) Transformirana desna slika

Slika 10: Ove su slike spremne za spajanje

Za pronalazak matrice homografije koristi se algoritam RANSAC (RANdom Sample Consensus).

1.9 RANSAC

RANSAC je algoritam koji rješava problem korespondencije između slika, tj. pokušava pronaći konzistentnu konfiguraciju, odnosno model koji će prikazati njihov odnos obzirom na dijelove i točke koje se podudaraju. RANSAC procjenjuje parametre takvog matematičkog modela korištenjem tehnike nasumičnog uzorkovanja (eng. random sampling).
Pretpostavljamo da se podaci koje imamo sastoje od inliera, tj. podataka koji se mogu prikazati pomoću nekog seta parametara modela, i outliera, podataka koji se ne uklapaju u model. Također pretpostavljamo da ako imamo skup inliera, postoji procedura koja procjenjuje parametre modela koji optimalno uklapa podatke.
RANSAC je nedeterministički algoritam jer za isti input može dati različite rezultate. Vjerojatnost da će dati dobar rezultat se povećava što mu više iteracija damo [17].
Kao input RANSAC prima skup promatranih podataka i parametrizirani model u koji može uklopiti te promatrane podatke.
RANSAC iterativno odabire nasumični podskup skupa originalnih podataka. Zatim testira jesu li podaci u tom podskupu inlieri. Namješta model obzirom na te hipotetske inliere, odnosno namješta parametre modela tako da se uklope podaci podskupa. Nakon toga su svi ostali podaci testirani u modelu. Ako se neka točka uklapa u model, onda se također smatra hipotetskim inlierom. Dobiveni model je dobar ako ima dovoljno točaka koje su klasificirane kao hipotetski inlieri. Zatim se model ponovno namješta, sada obzirom na sve hipotetske inliere. Model se evaluira tako da se provjere svi inlieri i pronađe ukupna greška između modela i inliera.
Ovaj se postupak ponavlja određen broj puta i svaki put dobijemo ili model koji je odbačen zbog premalo točaka klasificiranih kao inlieri ili refinirani model zajedno s ukupnom greškom.

Za drugi slučaj, model je prihvaćen ako mu je greška manja od greške prethodnog modela [17].

```
(H, status) = cv2.findHomography(ptsA, ptsB, cv2.RANSAC,threshold)
```

Vrijednost "threshold" koristi se kako bi algoritam znao je li točka dobra na način da se testira udaljenost točke izvan nasumičnog odabira podskupa hipotetskih inliera do linije koja predstavlja model. Ako je udaljenost manja od thresholda, točka se klasificira kao hipotetski inlier [17].

1.10 Spajanje

Nakon što transformiramo jednu od slika uz matricu homografije, možemo ju zalijepiti na drugu sliku. Za to koristimo OpenCV funkciju warpPerspective().

```
result = cv2.warpPerspective(imageB, H,(imageA.shape[1] +
                                        imageB.shape[1],
                                np.maximum(imageA.shape[0],
                                           imageB.shape[0])),
                                 flags=cv2.INTER_NEAREST)
result[0:imageB.shape[0], 0:imageB.shape[1]] = imageA
```

Funkciji cv2.warpPerspective() prosljeđujemo sliku koju transformiramo (u našem slučaju desnu sliku), matricu homografije i dimenziju rezultata.

Za visinu rezultata u kodu može se postaviti i veći broj od očekivanog, npr. imageA.shape[0] + 1000, kako bi se osiguralo da će se prikazati cijela transformirana slika (slika 16).

```
result = cv2.warpPerspective(imageB, H,
                 (imageA.shape[1] + imageB.shape[1], 5000),
                 flags=cv2.INTER_NEAREST)
result[0:imageA.shape[0], 0:imageB.shape[1]] = imageA
result = trim(result)
```

Nakon toga potrebno je pozvati funkciju koja će izrezati crne piksele na rubovima slike.

```
def trim(frame):
    #crop top
    while not np.sum(frame[0]):
        frame = frame[1:]
    #crop bottom
    while not np.sum(frame[-1]):
        frame = frame[:-2]
```

```
8       #crop left
9       while not np.sum(frame[:,0]):
10          frame = frame[:,1:]
11      #crop right
12      while not np.sum(frame[:,-1]):
13          frame = frame[:,:-2]
14
15      return frame
```

Slika 11: Rezultat spajanja

1.11 Primjeri

(a) Lijeva slika

(b) Desna slika

Slika 12: Slike koje spajamo

Slika 13: Nakon spajanja

(a) Lijeva slika

(b) Desna slika

Slika 14: Slike koje spajamo

Slika 15: Nakon spajanja

Slika 16: Rezultat spajanja ako smo postavili veću visinu pri cv2.warpPerspective()

(a) Lijeva slika

(b) Desna slika

Slika 17: Slike koje spajamo

Slika 18: Nakon spajanja

(a) Lijeva slika

(b) Desna slika

Slika 19: Slike koje spajamo

Slika 20: Nakon spajanja

(a) Lijeva slika

(b) Desna slika

Slika 21: Slike koje spajamo

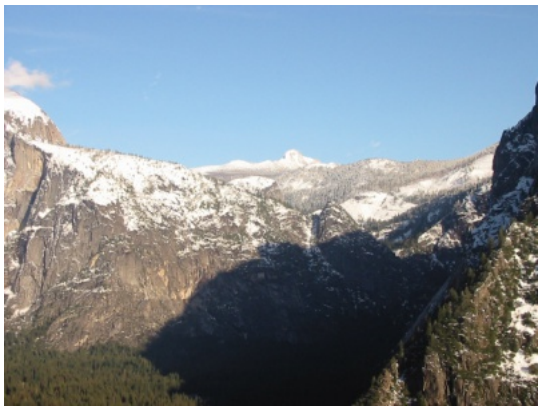
Slika 22: Nakon spajanja

1.12 Spajanje više slika

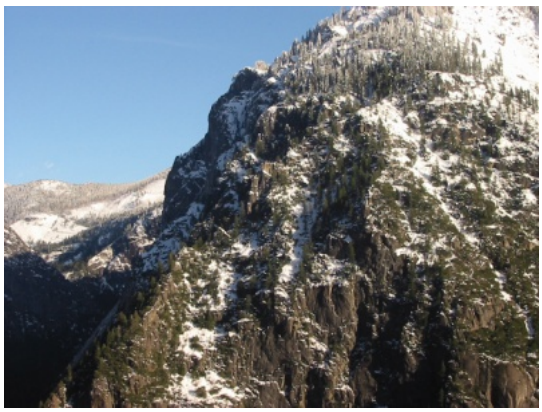
Spajanje više slika slično je spajanju dvije slike. Razlika je jedino u homografiji. Homografija se dobije tako da se pomnoži homografija iz prošlog spajanja s trenutnom. Npr. ako smo spojili prvu i drugu sliku, homografiju potrebnu za dodavanje treće slike dobit ćemo tako da pomnožimo homografiju između prve dvije slike s homografijom između druge i treće slike. Na taj način kad budemo spajali treću sliku, ona će biti poredana s rezultatom spajanja prve dvije slike.

(a) Prva slika

(b) Druga slika



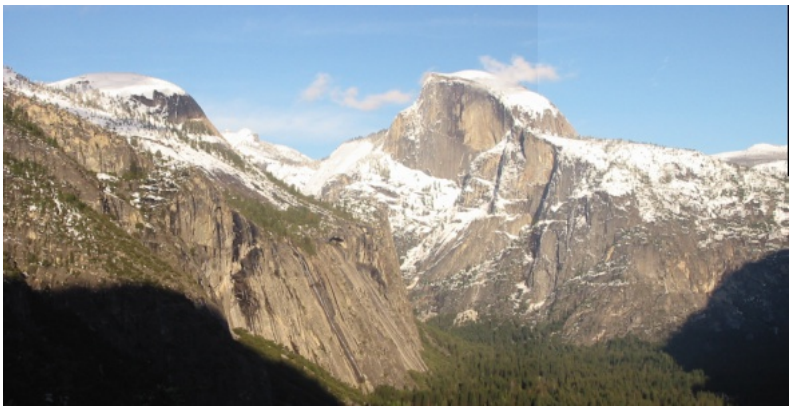
(c) Treća slika



(d) Četvrta slika

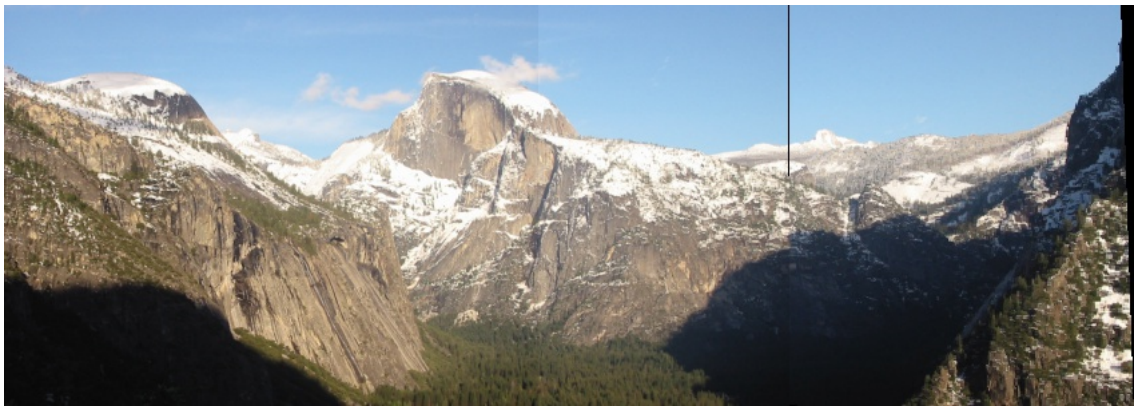
Slika 23: Slike za spajanje

Primjer 1. *Dakle, prvo ćemo spajati slike 23a i 23b postupkom opisanim u prijašnjim poglavljima. Homografiju između tih slika spremit ćemo u varijablu H.*



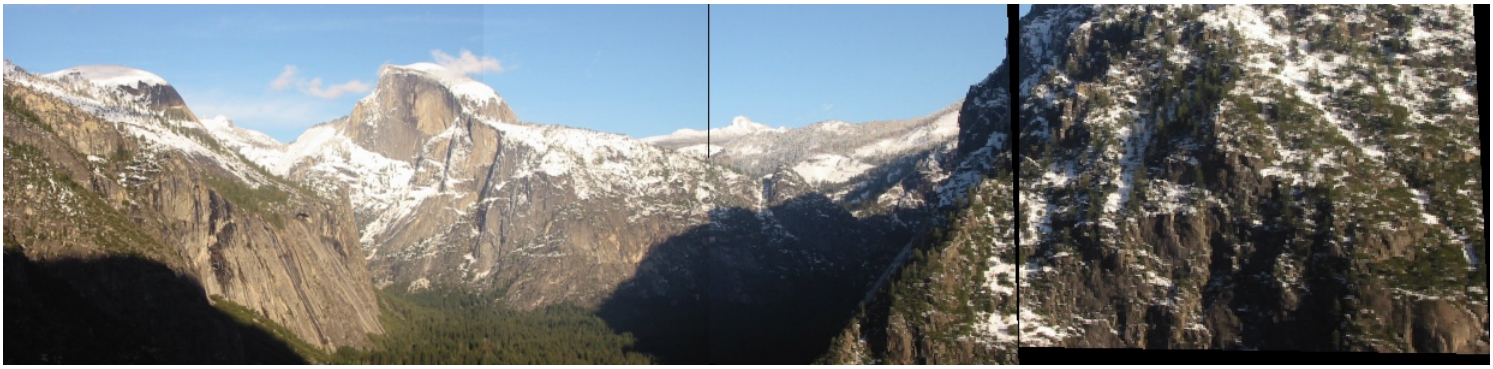
Slika 24: Nakon spajanja

Zatim dodajemo i treću sliku s desne strane. Homografiju dobijemo tako da pomnožimo matricu H i matricu homografije između slika 23b i 23c te umnožak opet spremimo kao H.



Slika 25: Nakon dodavanja treće slike

Na isti način dodajemo četvrtu sliku, a homografiju dobijemo umnoškom matrica H i matricom homografije između 23c i 23d.



Slika 26: Nakon dodavanja četvrte slike

1.13 Spajanje slika s nepoznatim poretkom

Spajanje slika s nepoznatim poretkom koristi iste tehnike kao i kada imamo određeni poredak, ali je potrebno malo više njihovog korištenja. Krećemo od pronalaska ključnih točaka i feature-a za svaku sliku. Pošto ne znamo koju sliku spajamo s kojom, za sve parove slika moramo pronaći matcheve. Odabiremo jednu od primljenih slika te tražimo s kojom drugom slikom ima najviše matcheva. Tada na početnu sliku spajamo onu sliku s kojom ima najviše matcheva, s lijeve ili s desne strane. Pa nastavljamo istim postupkom, sada za lijevu spojenu sliku tražimo sliku s kojom ima najviše matcheva s lijeve strane, a za desnu tražimo sliku s kojom ima najviše matcheva s desne strane. Ako spajamo slike u prikaz od 360°, možemo samo tražiti matcheve s jedne strane i na tu stranu dodavati slike.

1.14 Spajanje videa

Video podaci se spajaju vrlo slično kao i slike. Za svaki ulazni video potrebno je uzeti prvi frame te izračunati homografije među njima. Tada frameove možemo spojiti. Zatim

uzimamo iduće frameove koje spajamo s već izračunatom homografijom. Osim u slučaju ako kamere međusobno promjene položaj, više ne moramo računati homografiju već samo transformiramo frameove i spajamo ih.

Ovako možemo učitati i prikazati video:

```
# video datoteka
cap = cv2.VideoCapture('test360.mp4')

#ili

# video direktno s kamere prikljucene na racunalo
cap = cv2.VideoCapture(0)
while True:
    if cap.isOpened()==False:
        break
    ret, frame = cap.read()
    if ret:
        cv2.imshow('frame', frame)
    key = cv2.waitKey(1) & 0xFF
    if key == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

2 Spajanje u 360° prikaz

Koristeći samo dosadašnju proceduru spajanja slika, ne možemo pravilno spojiti slike u prikaz od 360° kao što vidimo na slici 30.

Potrebno je prvo sve primljene slike prebaciti u cilindrične koordinate i projekciju [18] :

```
def cylindricalWarp(img, K):

    h_,w_ = img.shape[:2]

    y_i, x_i = np.indices((h_,w_))
    X = np.stack([x_i,y_i,np.ones_like(x_i)]
                 ,axis=-1).reshape(h_*w_,3)
    Kinv = np.linalg.inv(K)
    X = Kinv.dot(X.T).T

    A = np.stack([np.sin(X[:,0]),
                  X[:,1],
                  np.cos(X[:,0])],
                 axis=-1).reshape(w_*h_,3)

    B = K.dot(A.T).T

    B = B[:,:-1] / B[:,[-1]]

    B[(B[:,0] < 0) | (B[:,0] >= w_)
      | (B[:,1] < 0) | (B[:,1] >= h_)]= -1
    B = B.reshape(h_,w_,-1)

    img_rgba = cv2.cvtColor(img,cv2.COLOR_BGR2BGRA)

    return cv2.remap(img_rgba, B[:,:,0].astype(np.float32),
                     B[:,:,1].astype(np.float32),
                     cv2.INTER_AREA,
                     borderMode=cv2.BORDER_TRANSPARENT)
```

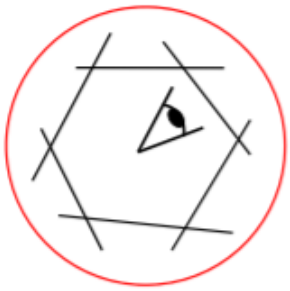
Intrinzična matrica, odnosno matrica kamere koja sadrži njene intrinzične parametre, je matrica dimenzija 3×3. Definirana je kao

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}.$$

Varijable f_x i f_y predstavljaju fokalnu duljinu u pikselima, a (c_x, c_y) optički centar. f_x i f_y možemo dobiti preko formule:

$$f_x(pixels) = f(mm) * image\_width(pixels)/sensor\_width(pixels)$$

$$f_y(pixels) = f(mm) * image\_height(pixels)/sensor\_height(pixels)$$

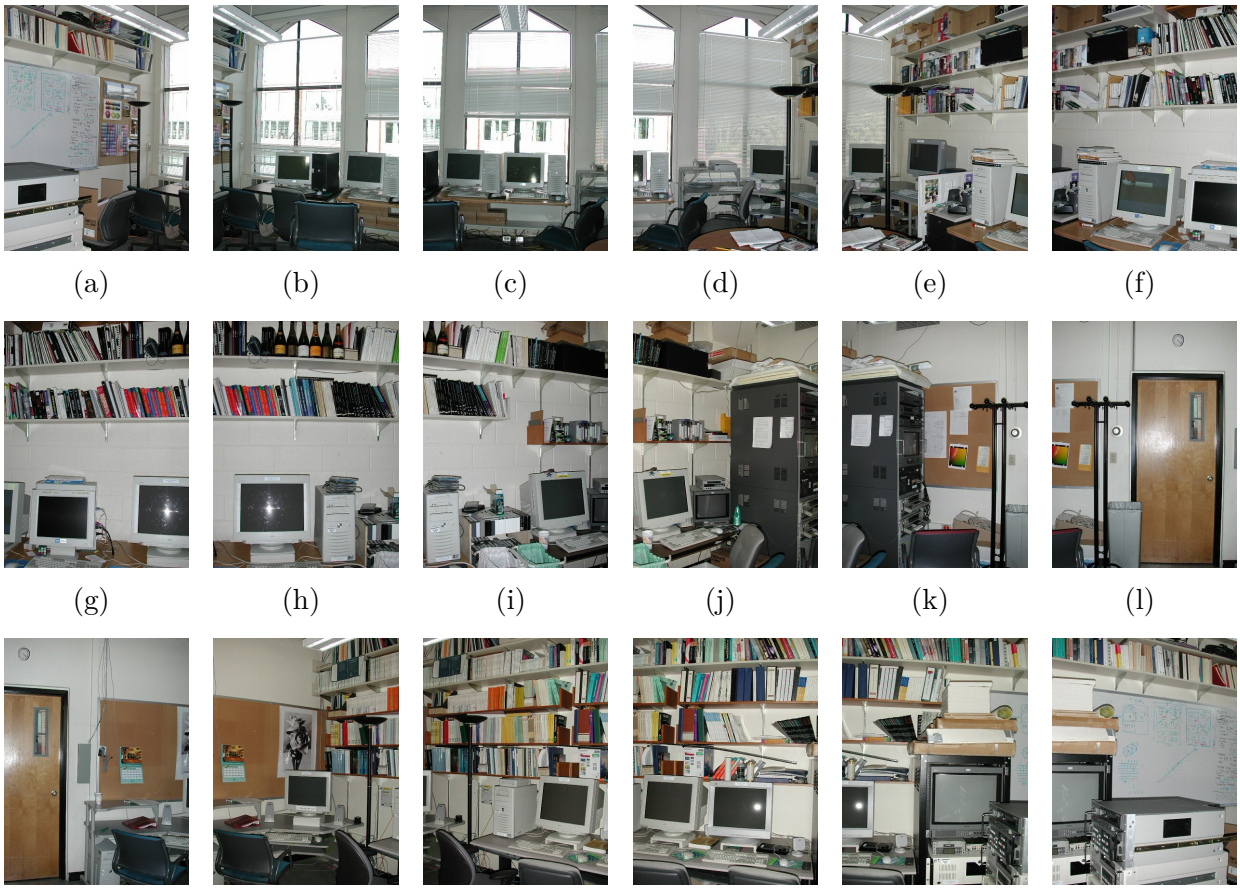


Slika 27: Cilindrična perspektiva

(a) Obična slika

(b) Slika nakon prelaska na cilindričnu projekciju

Slika 28



(a) (b) (c) (d) (e) (f)

(g) (h) (i) (j) (k) (l)

(m) (n) (o) (p) (q) (r)

Slika 29: Slike za spajanje u prikaz 360°

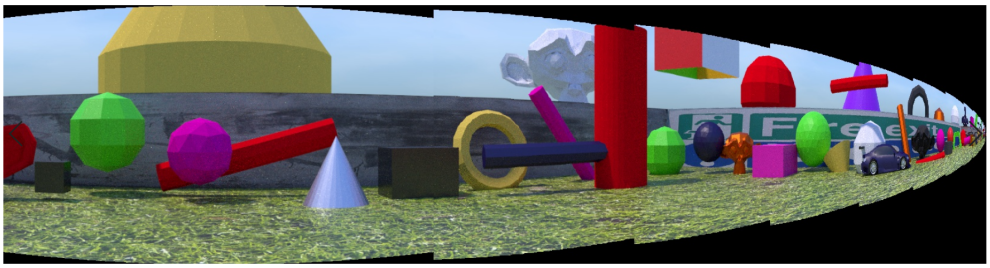
Slika 30: Rezultat spajanja slika iz slike 29 bez prijelaza na cilindričnu projekciju

Slika 31: Rezultat spajanja slika iz slike 29 uz prijelaz na cilindričnu projekciju

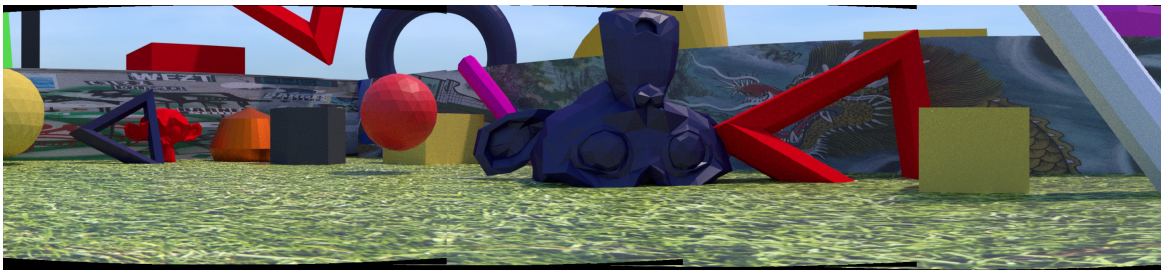
3 Rezultati spajanja slika generiranih u Blenderu

Za potrebe testiranja spajanja slika i videa, kolega Dino Šarlija napravio je 3D simulator u programu Blender. Simulator se sastoji od vozila na koje se postavljaju kamere te 3D svijeta u kojemu se to vozilo kreće. Simulator generira slike i videe s kamera postavljenih po vozilu koje se giba predodređenom rutom po svijetu. Niže su primjeri spajanja nekih od generiranih slika.

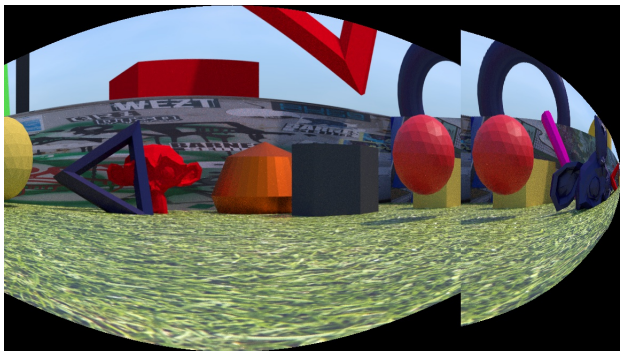
Slika 32: Spojene slike iz 18 kamera, 360°



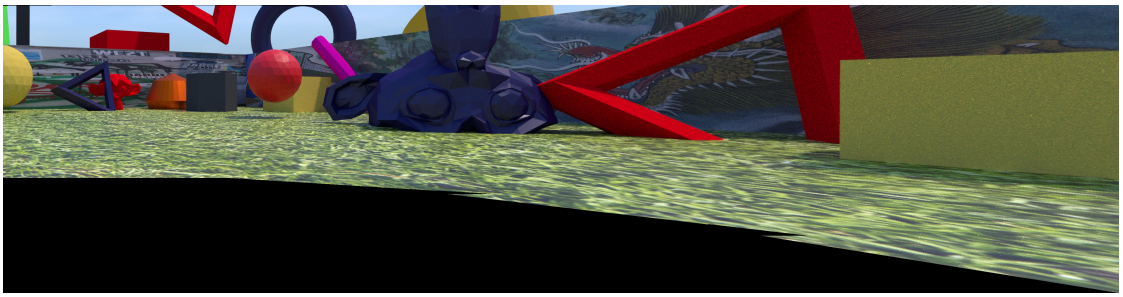
Slika 33: Spojene slike iz 18 kamera (360°), ali s premalom fokalnom duljinom



Slika 34: Spojene slike iz 4 kamere



Slika 35: Spojene slike iz 4 kamere s premalom fokalnom duljinom



Slika 36: Spojene slike iz 4 kamere s prevelikom fokalnom duljinom

Cijeli kod ovog projekta dostupan je na githubu (https://github.com/Branciii/Image-Video-Stitching).

4 Zaključak

Problem spajanja slika i videa vrlo je aktualno područje istraživanja. Time se bave i neke automobilske tvrtke poput Tesle. U ovom radu je opisana metoda spajanja slika i videa u panoramu pomoću algoritama KAZE, kNN, RANSAC i cilindrične transformacije. Detaljno su opisani svi koraci i korišteni algoritmi. Konačni kod testiran je na slikama stvarnog svijeta te na simuliranim slikama iz programa Blender. Rezultati bi se mogli poboljšati podešavanjem intenziteta slika (blending) i otklanjanjem crnih dijelova na rubovima i između spojenih slika. Također bi se moglo isprobati sferno mapiranje u svrhu slaganja cijele sfere.

Literatura

[1] http://matthewalunbrown.com/papers/ijcv2007.pdf

[2] https://en.wikipedia.org/wiki/Image_stitching

[3] https://www.researchgate.net/publication/267638940_Image_Stitching_based_on_Feature_Extraction_Techniques_A_Survey#:~:text=Image%20stitching%20(Mosaicing)%20is%20considered,which%20is%20called%20panoramic%20image.&text=The%20main%20components%20of%20image%20stitching%20will%20be%20described.

[4] https://www.sas.com/en_us/insights/analytics/computer-vision.html#:~:text=Computer%20vision%20is%20a%20field,to%20what%20they%20âĂœsee.âĂİ

[5] https://heartbeat.fritz.ai/how-computer-vision-is-disrupting-different-industries-

[6] https://www.pyimagesearch.com/2016/01/11/opencv-panorama-stitching/

[7] https://stackoverflow.com/a/29137285

[8] https://mathworld.wolfram.com/AffineTransformation.html

[9] https://se.mathworks.com/help/vision/ug/local-feature-detection-and-extraction.html

[10] https://en.wikipedia.org/wiki/Feature_detection_(computer_vision)

[11] Lowe, D. G. *"Distinctive Image Features from Scale-Invariant Keypoints"*, International Journal of Computer Vision, 60, 2, pp. 91-110, 2004.

[12] https://www.doc.ic.ac.uk/~ajd/Publications/alcantarilla_etal_eccv2012.pdf

[13] https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html#:~:text=Brute-Force%20matcher%20is%20simple,the%20BFMatcher%20object%20using%20cv2.

[14] https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_ml/py_knn/py_knn_understanding/py_knn_understanding.html?highlight=knn

[15] https://stackoverflow.com/a/60343973

[16] https://www.learnopencv.com/homography-examples-using-opencv-python-c/

[17] http://www.micc.unifi.it/delbimbo/wp-content/uploads/2011/03/slide_corso/A53%20Geometric%20alignment%20and%20outlier%20rejection%20RANSAC.pdf

[18] https://www.morethantechnical.com/2018/10/30/cylindrical-image-warping-for-panora