# Креирање на панорамски слики од повеќе слики

Изработиле:

Ана Костадиновска 211006

Танкица Понова 211105

# Содржина

# 1 Апстракт

Оваа семинарска работа има за цел да ја претстави имплементацијата на алгоритмите за креирање на панорамски слики преку демонстрација на соодветен код. Во имплементацијата се користи библиотеката OpenCV за обработка на слики и имплементација на алгоритмите. Прво ќе биде даден вовед во концептот на креирање на панорамски слики, како и неговото значење и примена во компјутерската визија. Ќе бидат опишани алгоритмите и методите што се користат во процесот на креирање на панорамски слики.

# 2 Вовед

Креирањето на панорамски слики е техника во областа на компјутерската визија што им овозможува на корисниците да спојат повеќе слики за создавање на една панорамска слика. Оваа техника има широка примена во области како панорамско фотографирање, 3D моделирање, виртуелна реалност и многу други.

“Image stitching” (спојување на слики) е метод со кој повеќе слики се спојуваат за да се создаде панорама. Ако се присутни две слики кои се преклопуваат, спојувањето на слики се состои од мозаик од две или повеќе слики во една рамка.

Image stitching или photo stitching е процес на комбинирање на повеќе фотографии со преклопувачки полиња за да се добие сегментирана панорама или слика. Вообичаено е изведено преку употреба на компјутерски софтвер, повеќето пристапи за спојување на слики бараат речиси точни преклопувања помеѓу сликите и идентични експозиции за да се добијат беспрекорни резултати.

Во панорамското спојување, комплетот на слики треба да има логична количина на преклопување за да надвладее над деформација на објектот и мора да содржи доволно мерливи карактеристики. Мозаик на панорамски слики функционира за многу слики со спојување на сликите во композитна слика со многу поширок поглед од обична камера. Колекцијата од слика може да вклучува две или повеќе слики направени во различно време, од различни сензори или други точки на поглед од една сцена.

# 3 Алгоритми

## 3.1 SIFT (Scale Invariant Feature Transform) алгоритам

Во 2004 година, David Lowe од Универзитетот во Британска Колумбија, излезе со нов алгоритам, Scale Invariant Feature Transform (SIFT) во неговиот труд, 'Distinctive Image Features from Scale-Invariant Keypoints', кој ги извлекува клучните точки и ги пресметува нивните дескриптори.

SIFT алгоритмот е техника на компјутерска визија што се користи за откривање и опис на карактеристики. Открива карактеристични клучни точки или карактеристики на сликата кои се цврсти за промени во размерот, ротацијата и трансформациите. SIFT работи на тој начин што ги идентификува клучните точки врз основа на нивните екстремни локални интензитети и ги пресметува дескрипторите што ги доловуваат информациите за локалната слика околу тие клучни точки. Овие дескриптори потоа може да се користат за задачи како што се совпаѓање на слики, препознавање објекти и пребарување на слики.

SIFT алгоритмот помага да се лоцираат локалните карактеристики на сликата, вообичаено познати како „клучни точки" на сликата. Овие клучни точки се непроменливи размери и ротација кои можат да се користат за различни апликации за компјутерска визија, како што се појавување на слики, откривање објекти, откривање сцени итн.

Можеме да ги користиме и клучните точки генерирани со помош на SIFT како карактеристики за сликата за време на обуката за модел. Главната предност на карактеристиките на SIFT е тоа што тие не се засегнати од големината или ориентацијата на сликата. [1]

Општо земено, целиот процес може да се подели на 4 дела:

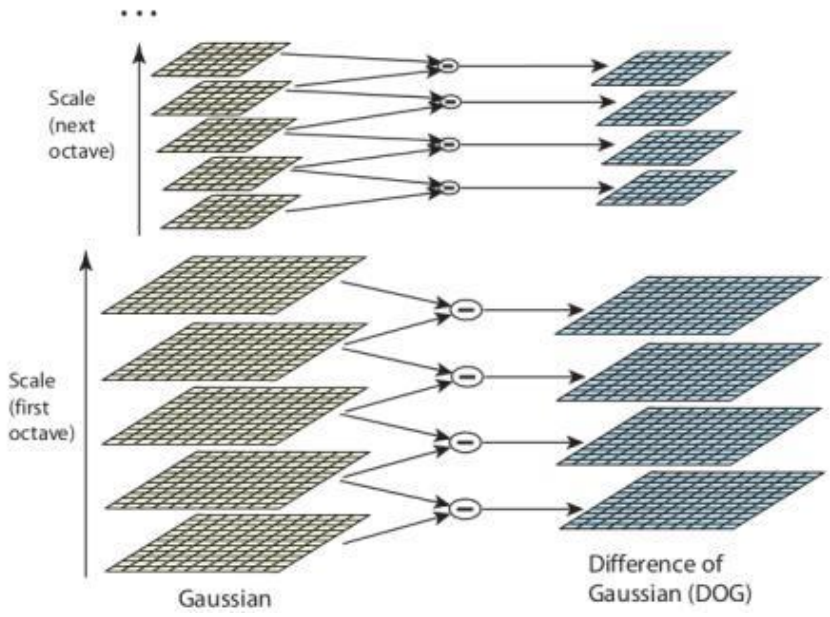
1) *Конструирање на простор за скала*: за да се увериме дека карактеристиките се независни од скалата
2) *Локализација на клучните точки*: идентификување на соодветни карактеристики или клучни точки
3) *Задача за ориентација*: проверка дали клучните точки се непроменливи при ротација
4) *Дескриптор на клучна точка*: доделување уникатен отпечаток на секоја клучна точка

### 3.1.1 Скала-простор Екстремна детекција

За да откриеме поголеми агли ни требаат поголеми прозорци. За ова се користи филтрирање на простор-скала. Во него за сликата со различни σ вредности се среќава Лаплацијан од Гаус (LoG). LoG делува како детектор на точки, детектира точки во различни големини поради промената на σ. Накратко, σ делува како параметар за скалирање. На пример, гаусовото јадро со ниско σ дава висока вредност за мал агол додека гаусовото јадро со високо σ добро се вклопува за поголем агол. Значи, можеме да ги

најдеме локалните максимуми низ скалата и просторот што ни дава листа на (x,y,σ) вредности што значи дека постои потенцијална клучна точка на (x,y) на скалата σ.

Но, овој LoG е малку скап, така што SIFT алгоритамот користи Difference of Gaussians што е приближна вредност на LoG. Разликата на Гаус се добива како разлика на Гаусово замаглување на слика со две различни σ, нека биде σ и kσ. Овој процес е направен за различни октави на сликата во Гаусовата пирамида. Таа е претставена на сликата подолу:




*Слика1. Гаусова пирамида*

Во однос на различните параметри, трудот дава некои емпириски податоци кои може да се сумираат како, број на октави = 4, број на нивоа на скала = 5, почетна σ = 1.6, k = $\sqrt{2}$ и така натаму, како оптимални вредности.

### 3.1.2 Локализација на клучните точки

Откако ќе се најдат локации на потенцијални клучни точки, тие мора да се рафинираат за да се добијат попрецизни резултати. Тие користеа проширување на просторот на скалата од серијата Taylor за да добијат попрецизна локација на екстремите, и ако интензитетот на овој екстрем е помал од прагот (0,03 според трудот), тој се отфрла. Овој праг се нарекува контраст Threshold во OpenCV.

DoG има поголем одговор за рабовите, така што и рабовите треба да се отстранат. За ова, се користи концепт сличен на детекторот на аголот Харис. Тие користеле 2x2 Хесијанска матрица (H) за да ја пресметаат главната кривина. Знаеме од Харис аголниот детектор дека за рабовите, едната сопствена вредност е поголема од другата. Така, овде тие користеа едноставна функција.

Ако овој сооднос е поголем од прагот, наречен edgeThreshold во OpenCV, таа клучна точка се отфрла. Се дава како 10 во трудот.

Така, ги елиминира сите клучни точки со низок контраст и клучните точки на рабовите, а она што останува се силните клучни точки.

### 3.1.3 Задача за ориентација

Сега се доделува ориентација на секоја клучна точка за да се постигне непроменливост на ротацијата на сликата. Се зема соседство околу локацијата на клучната точка во зависност од скалата, а големината и насоката на градиентот се пресметуваат во тој регион. Се создава ориентационен хистограм со 36 канти кои покриваат 360 степени (Се мери според големината на градиентот и гаусовиот кружен прозорец со σ еднаков на 1.5 пати од скалата на клучната точка). Се зема највисокиот врв во хистограмот и секој врв над 80% од него исто така се смета за пресметување на ориентацијата. Создава клучни точки со иста локација и размер, но различни насоки. Тоа придонесува за стабилност на совпаѓањето.

### 3.1.4 Опис на клучните точки

Сега е креиран дескриптор за клучна точка. Земено е соседството 16x16 околу клучната точка. Поделен е на 16 подблокови со големина 4x4. За секој подблок се креира хистограм за ориентација со 8 канти. Значи, достапни се вкупно 128 канти вредности. Тој е претставен како вектор за да формира дескриптор за клучна точка. Дополнително на ова, преземени се неколку мерки за да се постигне робусност против промените на осветлувањето, ротацијата итн.

### 3.1.5 Усогласување на клучните точки

Клучните точки помеѓу две слики се совпаѓаат со идентификување на нивните најблиски соседи. Но, во некои случаи, второто најблиско совпаѓање може да биде многу блиску до првото. Тоа може да се случи поради бучава или некои други причини. Во тој случај, се зема односот на најблиското растојание до второто најблиско растојание. Ако е поголемо од 0.8, тие се отфрлаат. Елиминира околу 90% од лажните совпаѓања додека пак од точните совпаѓања отфрла само 5%, како што е наведено во трудот. [1]

## 3.2 Сподерба на SIFT алгоритмот со останати алгоритми

Во овој дел, ја прикажуваме споредбата на *SIFT*, *SURF* и *ORB* алгоритмите наспроти секој интензитет, ротација, скалирање, “shearing”, искривувањето на “fish eye” и бучава.

| | Time (sec) | Kpnts1 | Kpnts2 | Matches | Match rate (%) |
|---|---|---|---|---|---|
| SIFT | 0.13 | 248 | 229 | 183 | 76.7 |
| SURF | 0.04 | 162 | 166 | 119 | 72.6 |
| ORB | 0.03 | 261 | 267 | 168 | 63.6 |

*Табела 1. Резултати од споредување на сликите со различен интензитет*

| | Time (sec) | Kpnts1 | Kpnts2 | Matches | Match rate (%) |
|---|---|---|---|---|---|
| SIFT | 0.16 | 248 | 260 | 166 | 65.4 |
| SURF | 0.03 | 162 | 271 | 110 | 50.8 |
| ORB | 0.03 | 261 | 423 | 158 | 46.2 |

*Табела 2. Резултати од споредување на сликата со нејзината ротирана слика*

| Angle → | 0 | 45 | 90 | 135 | 180 | 225 | 270 |
|---|---|---|---|---|---|---|---|
| SIFT | 100 | 65 | 93 | 67 | 92 | 65 | 93 |
| SURF | 99 | 51 | 99 | 52 | 96 | 51 | 95 |
| ORB | 100 | 46 | 97 | 46 | 100 | 46 | 97 |

*Табела 3. Стапка на совпаѓање наспроти аголот на ротација*

| | Time (sec) | Kpnts1 | Kpnts2 | Matches | Match rate (%) |
|---|---|---|---|---|---|
| SIFT | 0.25 | 248 | 1210 | 232 | 31.8 |
| SURF | 0.08 | 162 | 581 | 136 | 36.6 |
| ORB | 0.02 | 261 | 471 | 181 | 49.5 |

*Табела 4. Резултати од споредување на сликата со нејзината намалена слика*

| | Time (sec) | Kpnts 1 | Kpnts 2 | Matches | Match rate (%) |
|---|---|---|---|---|---|
| SIFT | 0.133 | 248 | 229 | 150 | 62.89 |
| SURF | 0.049 | 162 | 214 | 111 | 59.04 |
| ORB | 0.026 | 261 | 298 | 145 | 51.88 |

*Табела 5. Резултати од споредување на сликата со нејзината "sheared" слика*

| | Time (sec) | Kpnts 1 | Kpnts 2 | Matches | Match rate (%) |
|---|---|---|---|---|---|
| SIFT | 0.132 | 248 | 236 | 143 | 59.09 |
| SURF | 0.036 | 162 | 224 | 85 | 44.04 |
| ORB | 0.012 | 261 | 282 | 125 | 46.04 |

*Табела 6. Резултати од споредување на сликата со сликата со искривено "fish eye"*

| | Time (sec) | Kpnts1 | Kpnts2 | Matches | Match rate (%) |
|---|---|---|---|---|---|
| SIFT | 0.115 | 248 | 242 | 132 | 53.8 |
| SURF | 0.059 | 162 | 385 | 108 | 39.48 |
| ORB | 0.027 | 261 | 308 | 155 | 54.48 |

*Табела 7. Резултати од совпаѓање на сликата со додавање 30 % бучава*

Од овие истражувања може да се согледа дека *ORB* е најбрзиот алгоритам, но *SIFT* се извршува најдобро во повеќето сценарија. Поради тоа, се одлучивме да го искористиме алгоритмот SIFT за дадената тема. [3]

## 3.3 Brute – Force Matcher

Совпаѓањето на Brute-Force е едноставно. Го зема дескрипторот на една карактеристика во првиот сет и се совпаѓа со сите други карактеристики во вториот сет користејќи одредена пресметка на растојание. И најблискиот се враќа.

За BF matcher, прво треба да го креираме објектот BFMatcher користејќи **cv.BFMatcher()**. Потребни се два изборни параметри.

Првиот е **normalType**. Го одредува мерењето на растојанието што треба да се користи. Стандардно, тоа е **cv.NORM_L2**. Добро е за SIFT, SURF и други (cv.NORM_L1 е исто така таму). За дескриптори базирани на бинарни стрингови како што се ORB, BRIEF, BRISK итн, треба да се користи cv.NORM_HAMMING, кој користи Хаминово растојание како мерење.

Вториот параметар е **булова променлива**, вкрстена проверка која стандардно е неточна. Ако е **точно**, Matcher ги враќа само оние совпаѓања со вредност (i,j) така што i-тиот дескриптор во множеството А има j-ти дескриптор во множеството Б како најдобро совпаѓање и обратно. Односно, двете карактеристики во двата сета треба да одговараат една на друга.

Обезбедува конзистентен резултат и е добра алтернатива на тестот за сооднос предложен од Д. Лоу во трудот за SIFT. ‘**cv.drawMatches()**‘ ни помага да ги извлечеме совпаѓањата. Сложува две слики хоризонтално и црта линии од првата слика до втората слика што ги прикажува најдобрите совпаѓања. [4]

# 4 Имплементација

## 4.1 Библиотеки

*OpenCV* овозможува моќни операции за обработка на слики, како и читање и манипулирање на податоците што се користат во спојувањето на сликите. Библиотеката *argparse* додатно обезбедува лесен начин за управување со влезните аргументи од командна линија, што го прави кодот пофлексибилен и кориснички прилагодлив.

```
import cv2

import argparse
```

- **cv2 -** ова е основната библиотека за компјутерска визија, наречена OpenCV. Во кодот, cv2 се користи за читање на сликите – **cv2.imread()** , промена на големината на сликите – **cv2.resize()**, конверзија на слики во сиви тонови – **cv2.cvtColor()**, исцртување на совпаѓањата на клучни точки – **cv2.drawMatches()**, прикажување на сликите – **cv2.imshow()** и затворање на отворените прозорци – **cv2.destroyAllWindows()**.
- **argparse** – библиотека за обработка на аргументи од командна линија. Во кодот, argparse се користи за дефинирање и читање на влезните аргументи што се патеки до сликите. Ова им овозможува на корисникот да ги контролира влезните слики што ќе се користат за спојување преку командната линија.

## 4.2 Аргументи од командна линија

Се креира објект од класата `**ArgumentParser**` со опис на програмата како аргумент.

Се додава аргумент на објектот `**p**`.

`**'image_paths'**` е името на аргументот, `**nargs='+'**` означува дека може да бидат внесени една или повеќе вредности за овој аргумент, а `**'paths to images'**` е помошна порака што ја објаснува намената на аргументот.

Се вчитуваат аргументите од командната линија и се зачувуваат во променливата `**args**`. Ова овозможува да се проследат патеките до сликите како аргументи при извршувањето на програмата.

```
p=argparse.ArgumentParser(description='Stitching images')

p.add_argument('image_paths', nargs='+', help='paths to images')

args=p.parse_args()
```

## 4.3 Вчитување на сликите

Се креира празна листа `**imgs**` за зачувување на сликите.

Со помош на `**cv2.imread()**` се вчитува секоја слика соодветно на нејзината патека од аргументите.

Доколку вчитувањето на сликата не успее (сликата не постои или форматот не се поддржува), се прикажува порака за грешка и програмата се прекинува.

Вчитаните слики се додаваат во листата `**imgs**`.

```
imgs=[]

for path in args.image_paths:
    image = cv2.imread(path)

    if image is None:
        print(f"Failed to load image: {path}")
        exit(1)

    imgs.append(image)
```

## 4.4 Промена на големините на сликите

`**desired_width**` е зададената ширина на панорамската слика.

Се креира празна листа `**resized_images**` за зачувување на сликите со променета големина.

Со помош на формулата `**ratio = desired_width / img.shape[1]**` се пресметува соодветниот однос на ширина за секоја слика.

Се пресметува посакуваната висина на секоја слика со формулата `**desired_height = int(img.shape[0] * ratio)**`.

Со помош на функцијата `**cv2.resize()**` се променува големината на секоја слика според зададената ширина и посакуваната висина.

Сликите со променета големина се додаваат во листата `resized_images`.

```
desired_width=400

resized_images=[]
```

```
for img in imgs:
    ratio=desired_width/img.shape[1]

    desiredHheight=int(img.shape[0] * ratio)

    resizedImg=cv2.resize(img, (desired_width, desiredHheight))

    resized_images.append(resizedImg)
```

## 4.5 Претворање на сликите во сиви тонови

```
grayscaleImgs=[cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) for image in resized_images]
```

Поради нивниот едноставен карактер, сликите се претворени во тонови на сиво. Ова го олеснува процесот на детекција на клучни точки и совпаѓање на клучните точки помеѓу сликите.

Се креира листа `**grayscale_imgs**` за зачувување на сликите во сиви тонови.

Со помош на листовно изразување (list comprehension) секоја слика од `**resized_images**` се конвертира во сиви тонови користејќи ја функцијата `**cv2.cvtColor()**` со аргумент `**cv2.COLOR_BGR2GRAY**`.

Резултатите се зачувуваат во листата `**grayscale_imgs**`.

## 4.6 Детекција на клучни точки

Со сликите во сиви тонови, се извршува детекција на клучни точки. За оваа цел, е користен **SIFT** (Scale-Invariant Feature Transform) алгоритмот.

```
sift = cv2.SIFT_create()

keypoints=[]

descriptors=[]

for img in grayscaleImgs:
    kp,desc=sift.detectAndCompute(img, None)
```

```
keypoints.append(kp)

descriptors.append(desc)
```

Оваа секција од кодот се однесува на примена на алгоритмот за детекција на клучни точки и дескриптори на секоја слика во листата `grayscale_imgs`.

Прво се креира објект од класата `**cv2.SIFT**` со помош на методата `**SIFT_create()**`. Оваа класа претставува алгоритам за детекција на клучни точки и дескриптори, познат како SIFT (Scale-Invariant Feature Transform).

За секоја слика во листата `grayscale_imgs` се извршува следниот дел од кодот:

```
 kp,desc=sift.detectAndCompute(img, None)

keypoints.append(kp)

descriptors.append(desc)
```

`**detectAndCompute**` методот ги наоѓа клучните точки (`kp`) и дескрипторите (`desc`) за дадената слика `img`. Клучните точки се локални особености на сликата, како на пример јамки или пиксели со специфична текстура. Дескрипторите се нумерички вектори кои го опишуваат изгледот на клучните точки.

Откако се најдени клучните точки и дескрипторите за секоја слика, тие се додаваат во соодветните листи `**keypoints**` и `**descriptors**`. Ова овозможува да се пристапи до клучните точки и дескрипторите на секоја слика во подоцна фаза од процесот на спојување.

Во целост, овој дел од кодот го применува алгоритмот SIFT за детекција на клучни точки и дескриптори на секоја слика. Клучните точки и дескрипторите ќе бидат искористени за совпаѓање на точките помеѓу соседни слики, што ќе послужи во процесот на спојување на сликите.

## 4.7 Совпаѓање на клучни точки

```
bf=cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)

matches=[]
```

```
for i in range(len(descriptors) - 1):
    matches.append(bf.match(descriptors[i], descriptors[i+1]))
```

Во овој дел од кодот, се користи **Brute – Force** алгоритмот (`**cv2.BFMatcher**`) за совпаѓање на дескрипторите помеѓу соседни слики. Прво се креира објект од класата `**cv2.BFMatcher**` со помош на методот `**BFMatcher()**`.
Оваа класа го овозможува совпаѓањето на дескриптори преку Brute – Force пристап, каде што се споредуваат секоја комбинација на дескриптори од две слики.

При креирањето на објектот `cv2.BFMatcher`, се предава аргумент `**cv2.NORM_L2**` кој претставува нормата која се користи за пресметка на растојанието помеѓу дескрипторите. Оваа норма се применува за мерење на сличноста помеѓу дескрипторите.

Откако ќе биде креиран објектот `cv2.BFMatcher`, се извршува циклус кој се повторува за бројот на слики минус еден (`len(descriptors) - 1`). Во секоја итерација се извршува споредување на дескрипторите помеѓу две соседни слики.

Во секоја итерација на циклусот, со помош на методот `**match(descriptors[i], descriptors[i+1])**` на објектот `bf` се извршува совпаѓање на дескрипторите помеѓу сликата `i` (моменталната слика) и сликата `i+1` (следната слика). Овој метод ги применува алгоритмот на Brute – Force за совпаѓање и враќа листа со најдобрите совпаднати точки.

На крајот на овој дел од кодот, листата `**matches**` содржи листи со совпаднати точки помеѓу секој последователен пар на слики.

## 4.8 Сортирање на совпаѓањето

После совпаѓањето на клучните точки, следи фаза на сортирање на совпаѓањето според нивната далечина. Ова го овозможува избирањето на најдобрите клучни точки за креирање на панорамски слики. За секој последователен пар на слики, совпаѓањето се сортира со помош на функцијата `**sorted()**` со параметар `key=lambda x: x.distance`.

```
sortedMatches=[sorted(match, key=lambda x: x.distance) for match in matches]
```

Се креира нова листа `**sortedMatches**`. За секој елемент во листата `matches` (што претставува листа со совпаднати точки за секој последователен пар на слики), се извршува сортирање на точките. Клучната точка се сортира според нејзиното **растојание** (`distance`) користејќи го ламбда изразот `lambda x: x.distance` како критериум за сортирање.

На овој начин, за секој последователен пар на слики во листата `sortedMatches` се зачувува листа со совпаднати точки, сортирани според нивното растојание.

## 4.9 Обработка на совпаѓањата и прикажување на резултатите

Следниот дел од кодот се однесува на обработката на спарувањата на клучни точки и визуелизацијата на резултатот:

```
N=50

matchedImgs=[]

for i in range(len(sortedMatches)):
   matchedImg = cv2.drawMatches(resized_images[i], keypoints[i], resized_images[i+1],
keypoints[i+1],
                            sortedMatches[i][:N], None,
flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
   matchedImgs.append(matchedImg)
```

Оваа секција на кодот го создава визуелниот приказ на совпаѓањата на клучни точки меѓу соседните слики. За секој последователен пар на соседни слики, користиме функција `**cv2.drawMatches**` за да ги нацртаме совпаѓањата на клучните точки. Влезните параметри на оваа функција се:

- `resized_images[i]` и `resized_images[i+1]`: Сликите кои се совпаѓаат.
- `keypoints[i]` и `keypoints[i+1]`: Клучните точки за секоја слика.
- `sortedMatches[i][:N]`: Првите N најдобри спарувања за парот на слики.
- `None`: Маска за филтрирање на спарувањата.
- `flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS`: Flag (знаменце) кој означува дека не сакаме да се исцртаат единечни клучни точки.
- Нацртаните слики со совпаѓањата се зачувуваат во листата `matchedImgs`.

```
stitcher=cv2.Stitcher.create()

status,result=stitcher.stitch(resized_images)
```

Овде креираме објект од класата `**cv2.Stitcher**` преку методот `**create()**`. Со користење на овој објект, го повикуваме методот `**stitch()**` кој ја извршува операцијата на спојување на

сликите претходно со намалена големина и зачувани во листата `resized_images`. Резултатот на креирањето на панорамски слики се зачувува во променливата `**result**`, а статусот на креирањето на панорамски слики се зачувува во променливата `status`.

```
cv2.imshow('Result', result)

for i, matchedImg in enumerate(matchedImgs):
   cv2.imshow(f'Keypoint Matches {i+1}-{i+2}', matchedImg)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

Откако ќе го добиеме резултатот од креирањето на панорамата, го прикажуваме резултатот со помош на функцијата `**cv2.imshow()**`. Со првиот повик на `cv2.imshow()` го прикажуваме резултатот на креирањето на панорамски слики и го нарекуваме "Result". Со помош на циклусот `for` и функцијата `enumerate()`, ги прикажуваме сите слики со совпаѓањата на клучните точки, каде што користиме форматирање на низи за да ги назначиме соодветните имиња на прозорците ("Keypoint Matches 1-2", "Keypoint Matches 2-3" итн.).

На крајот, со помош на функциите `**cv2.waitKey(0)**` и `**cv2.destroyAllWindows()**` го чекаме притисокот на копчето 0 (или било кое копче) за да ги затвориме прозорците и да се заврши извршувањето на програмата.

# 5 Резултати

На командна линија се внесува името на скриптата и патеките до сликите од кои што сакаме да се создаде панорамската слика:

‘python panorama.py images/image1.jpg images/image2.jpg images/image3.jpg . . .’

Како резултат, од претходно објаснетиот код, се добива резултантната панорамска слика и слики со нацртани клучни точки кои се совпаднати.

За пример, ќе прикажеме внес на три слики и излез на истите.

*Слика2. Прва влезна слика*

*Слика3. Втора влезна слика*

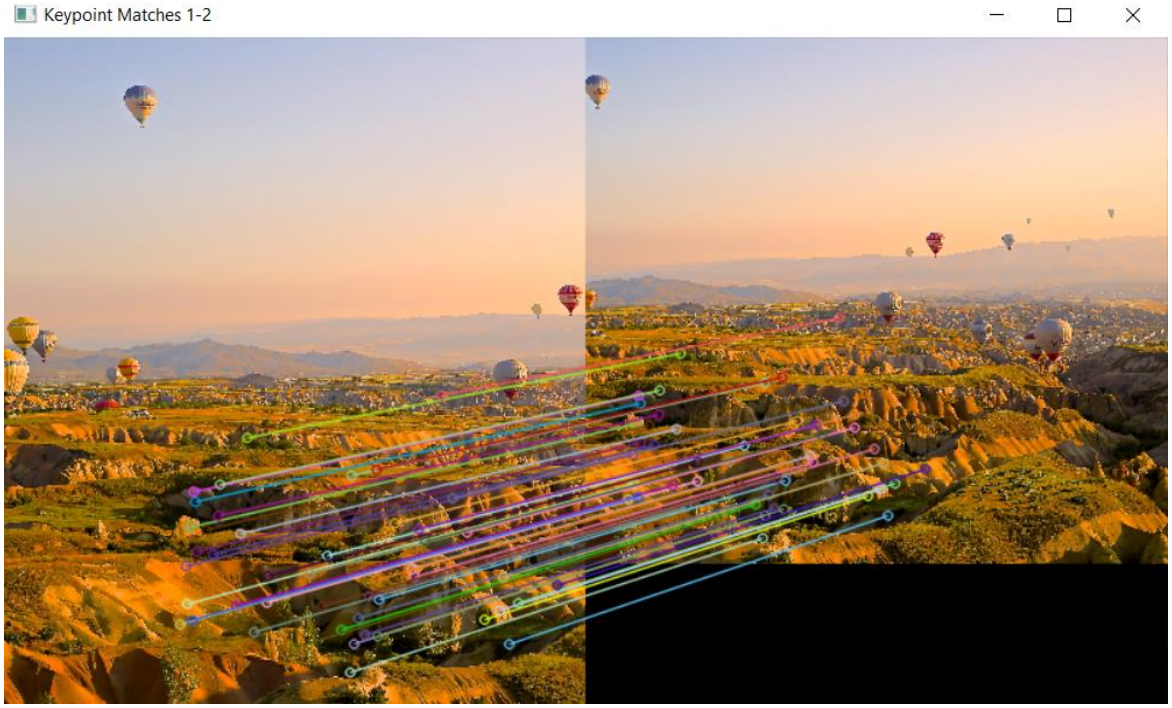
*Слика4. Трета влезна слика*

Резултантната панорамска слика:


*Слика5. Панорамска слика*

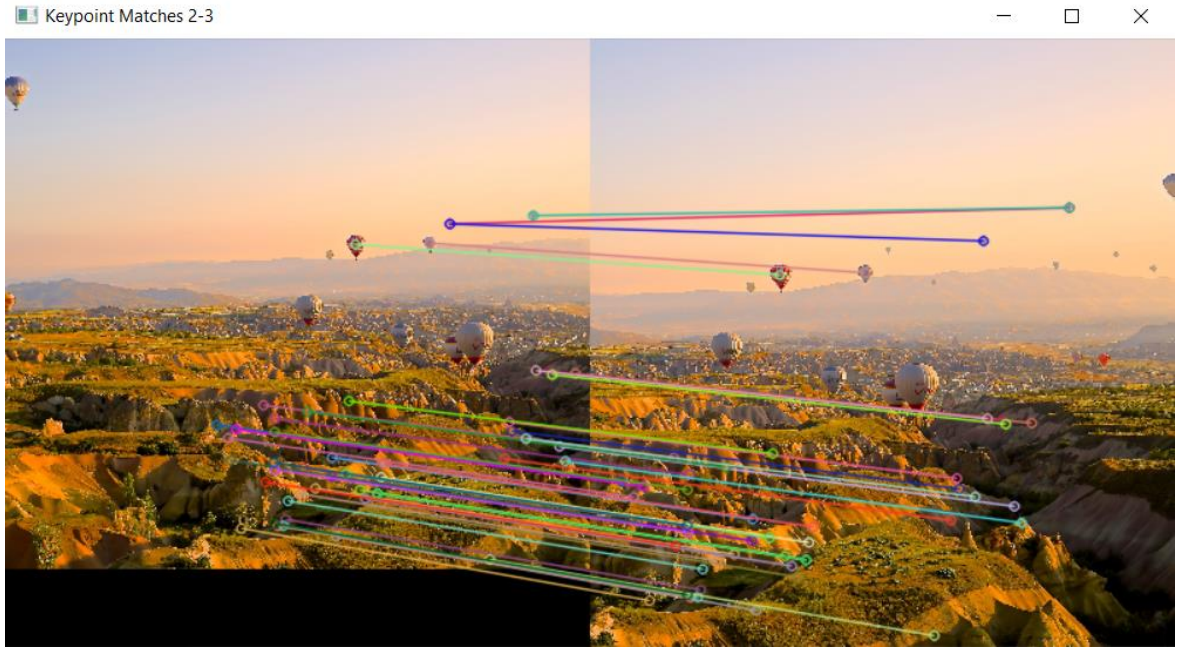
Исто така, се прикажуваат и пар на последователни слики со соодветно нацртани совпаднати клучни точки.

Прво, нацртаните клучни точки кои се совпаднале кај првата и втората слика:




*Слика6. Совпаднати клучни точки на првата и втората слика*

И нацртаните клучни точки кои се совпаднале кај втората и третата слика:




*Слика7. Совпаднати клучни точки на втората и третата слика*

# 6 Заклучок

Во рамките на оваа семинарска работа, беше претставена имплементација на креирање на панорамски слики со користење на различни алгоритми и техники. Изведени се експерименти врз влезни слики, каде што беа извршени операции за промена на големината на сликите, претворање на сликите во сиви тонови, детекција на клучни точки и спојување на сликите. Резултатите беа прикажани преку панорамски слики и нацртани совпаднати клучни точки.

Преку оваа семинарска работа, покажавме дека креирањето на панорамски слики е ефективен метод за анализа на визуелни податоци. Соодветната имплементација на алгоритмите и техниките ни овозможи да добиеме квалитетни резултати и дорбо спојување на сликите.

# 7 Референци


[1] https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html
[2] https://www.analyticsvidhya.com/blog/2019/10/detailed-guide-powerful-sift-technique-image-matching-python/
[3] https://arxiv.org/ftp/arxiv/papers/1710/1710.02726.pdf
[4] https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html
[5] https://pyimagesearch.com/2018/12/17/image-stitching-with-opencv-and-python/
[6] https://docs.opencv.org/4.x/d4/d5d/group__features2d__draw.html