КРЕИРАЊЕ НА ПАНОРАМСКИ СЛИКИ ОД ПОВЕЌЕ СЛИКИ

Семинарска работа по предметот: Дигитално процесирање на слика

Професор:

Проф. Д-р Ивица Димитровски

Изработила:

Сандра Живановска 221152

Содржина

1 Апстракт 3
2 Вовед 3
3 SIFT (Scale Invariant Feature Transform) алгоритам 4
4 Имплементација 5
4.1 Библиотеки 5
4.2 Аргументи од командна линија 6
4.3 Вчитување на сликите 6
4.4 Промена на големините на сликите 7
4.5 Претворање на сликите во сиви тонови 7
4.6 Детекција на клучни точки 8
4.7 Совпаѓање на клучни точки 10
4.8 Сортирање на совпаѓањето 10
4.9 Обработка на совпаѓањата и прикажување на резултатите 11
5 Резултати 13
6 Заклучок 18
7 Референци 18

1 Апстракт

Целта на оваа семинарска работа е да ја прикаже имплементацијата на алгоритми за создавање панорамски слики преку демонстрација на соодветен код. Во имплементацијата се користи библиотеката OpenCV за обработка на слики и примена на алгоритмите. На почетокот ќе биде претставен вовед во концептот на создавање панорамски слики, како и неговата важност и примена во областа на компјутерската визија. Понатаму, ќе бидат опишани алгоритмите и методите што се користат во процесот на создавање панорамски слики.

2 Вовед

Во денешната дигитална доба, техниката за креирање панорамски слики, позната и како "image stitching" (спојување на слики), игра клучна улога во областа на компјутерската визија. Оваа техника овозможува спојување на повеќе слики за создавање на една единствена панорамска слика, која нуди многу поширок и поцелосен приказ од обичната фотографија. Примената на оваа техника е широка, вклучувајќи панорамско фотографирање, 3D моделирање, виртуелна реалност и многу други области.

Спојувањето на слики, или photo stitching, е процес на комбинирање на повеќе фотографии со преклопувачки полиња за да се добие сегментирана панорама или слика. Ова обично се изведува со користење на компјутерски софтвер, каде повеќето пристапи бараат речиси точни преклопувања помеѓу сликите и идентични експозиции за да се добијат беспрекорни резултати. Комплетот на слики потребен за панорамско спојување треба да има логична количина на преклопување за да се минимизира деформацијата на објектите и да содржи доволно мерливи карактеристики.

Со оваа техника, мозаикот од панорамски слики функционира за многу слики, создавајќи композитна слика со многу поширок поглед од она што обичната камера може да го фати. Колекцијата од слики може да вклучува две или повеќе слики направени во различно време, од различни сензори или од различни точки на поглед на една сцена. Токму оваа способност за интеграција на различни перспективи го прави панорамското спојување толку моќно и широко применливо во различни сфери.

3 SIFT (Scale Invariant Feature Transform) алгоритам

SIFT (Scale Invariant Feature Transform) е алгоритам за детекција и опишување на локални карактеристики во слики, развиен од Дејвид Лоу. Овој алгоритам е особено моќен бидејќи е инвариантен на скала и ротација, што значи дека може да препознае и опише објекти во слики дури и ако тие се со различна големина или ориентација.

Клучни чекори на SIFT алгоритамот:

1. **Детекција на екстреми во простор и скала:**

   - Првиот чекор е да се најдат точки од интерес, наречени клучни точки (keypoints), кои се инвариантни на скала. Ова се постигнува со генерирање на „scale-space“ од сликата, што е збир на слики добиени со применување на Gaussian филтер со различни стандардни девијации (скали).

   - 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. **Точна локализација на клучните точки:**
   - Откриените клучни точки од првиот чекор се подложни на дополнителен процес на прецизна локализација за да се одредат нивните точни позиции и скали. Тука се применуваат различни тестови за да се отфрлат клучните точки кои не се стабилни или имаат мала контрастност.

2. **Определување на ориентација:**

   - За секоја клучна точка се пресметува ориентацијата базирана на локалните градиенти на пикселите околу точката. Ова овозможува алгоритмот да биде инвариантен на ротација.
   - Градиентните информации се користат за да се креира хистограм на ориентации. Главната ориентација на клучната точка е определена како врвот на хистограмот.

3. **Генерирање на дескриптори:**

   - Околу секоја клучна точка се креира дескриптор кој го опишува локалниот градиентен патерн. Овој дескриптор е претставен како вектор од 128 димензии.
   - Дескрипторот се конструира со поделба на областите околу клучната точка на помали подреони, и пресметување на ориентациони хистограми за секоја подреона.

Примена на SIFT:

- **Панорамски слики:** SIFT се користи за да се идентификуваат и спојат преклопувачките делови на повеќе слики за да се создаде панорама.
- **3D реконструкција:** Алгоритамот се користи за да се идентификуваат и следат карактеристики во повеќе слики за создавање на 3D модели.
- **Објектно препознавање:** SIFT е многу популарен за препознавање објекти во слики, дури и кога објектите се гледани од различни агли или се со различна скала.

SIFT е еден од најпознатите и најшироко користени алгоритми во областа на компјутерската визија поради неговата робусност и способност за точна идентификација на карактеристики во различни услови.

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.

```
1  import cv2
2  import argparse
```

• **cv2** - ова е главната библиотека за компјутерска визија, позната како OpenCV. Во овој код, cv2 се користи за неколку операции: читање на слики со cv2.imread(), промена на нивната големина со cv2.resize(), конвертирање на слики во сиви тонови со cv2.cvtColor(), исцртување на совпаѓањата на клучни точки со cv2.drawMatches(), прикажување на сликите со cv2.imshow() и затворање на отворените прозорци со cv2.destroyAllWindows().

• **argparse** - библиотека за обработка на аргументи од командната линија. Во овој код, argparse се користи за дефинирање и читање на влезните аргументи, кои се патеки до сликите. Ова им овозможува на корисниците да ги специфицираат влезните слики што ќе се користат за спојување преку командната линија, овозможувајќи поголема флексибилност и контрола врз процесот.

4.2 Аргументи од командна линија

```
4   parser = argparse.ArgumentParser(description='Stitching images')
5   parser.add_argument( *name_or_flags: 'image_files', nargs='+', help='paths to images')
6   args = parser.parse_args()
```

- Со командата во ред 4 се создава нов објект ArgumentParser од библиотеката argparse, со опис „Stitching images“, што дава краток опис за што служи програмата.Се додава аргумент на објектот `**parser**`.
- Со командата во ред 5 се додава нов аргумент image_files на парсерот. Аргументот прифаќа една или повеќе патеки до слики (означено со nargs='+'). Описот „paths to images“ дава информации за корисникот за тоа што се очекува.
- Потоа 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.

4.3 Вчитување на сликите

```
8   image_list = []
9   for file_path in args.image_files:
10      img = cv2.imread(file_path)
11      if img is None:
12          print(f"Failed to load image: {file_path}")
13          exit(1)
14      image_list.append(img)
```

- Во ред 1 се создава празна листа image_list која ќе се користи за складирање на сликите што ќе бидат прочитани од датотечните патеки.
- Во ред 9 започнува for јамка која ќе итерира преку сите патеки до сликите што се дадени како аргументи преку командната линија. Во секоја итерација на јамката, оваа команда

користи cv2.imread() од библиотеката OpenCV за да ја прочита сликата од тековната патека file_path и ја складира прочитаната слика во променливата img. if img is None:

- Командата во ред 11 проверува дали прочитаната слика е None, што би значело дека не успеало читањето на сликата (на пример, ако патеката е неточна или датотеката не постои). Ако сликата не е успешно прочитана, оваа команда печати порака за грешка во која се наведува патеката до сликата што не можела да се прочита и со exit(1) се прекинува извршувањето на програмата и излегува со статус код 1, што означува грешка. Ако сликата е успешно прочитана, сликата се додава во листата image_list.

4.4 Промена на големините на сликите

- Променливата target_width е зададена на 400, што значи дека целната ширина на сите слики ќе биде 400 пиксели.
- Се дели целната ширина (target_width) со оригиналната ширина на сликата (image.shape[1]) за да се добие скалата за ширина.
- Оригиналната висина на сликата (image.shape[0]) се множи со скалата за ширина за да се добие целната висина (**target_height = int(image.shape[0] * scaling_ratio)**`).
- Се користи cv2.resize функцијата од OpenCV библиотеката за да се скалира сликата на целните димензии и потоа се додава скалираната слика во листата scaled_images.

```
16  target_width = 400
17  scaled_images = []
18  for image in image_list:
19      scaling_ratio = target_width / image.shape[1]
20      target_height = int(image.shape[0] * scaling_ratio)
21      scaled_img = cv2.resize(image, dsize: (target_width, target_height))
22      scaled_images.append(scaled_img)
```

4.5 Претворање на сликите во сиви тонови

```
24  gray_images = [cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) for image in scaled_images]
```

Во овој дел од кодот се користи техниката наречена list comprehension за да се создаде нова листа од слики во сиви тонови. Оваа техника овозможува брза и ефикасна итерација низ секоја слика во постоечката листа од скалирани слики, и за секоја од нив врши конверзија од BGR (стандардниот формат за слики во OpenCV кој содржи сини, зелени и црвени компоненти) во сива скала (која има само една компонента што ја претставува светлината).

1. **BGR во сива скала:**

- Сликите во OpenCV се стандардно зачувани во BGR формат, кој е сличен на RGB форматот, но со поинаква редоследност на боите. Овој формат ги користи сината, зелената и црвената компонента за да ги претстави боите на сликата.
- Конвертирањето во сива скала значи дека ќе се отстрани информацијата за боите и ќе се зачува само информацијата за светлината. Ова се постигнува со комбинација на синото, зеленото и црвеното ниво на секој пиксел според одредени тежини за да се добие една вредност што ја претставува светлината.

2. **Предности на сивата скала:**

- Намалување на комплексноста: Сликите во сива скала се еднодимензионални, што значи дека за секој пиксел се чува само една вредност наместо три. Ова значително ги намалува пресметковните барања за понатамошна обработка.
- Подобрување на перформансите: Многу алгоритми за обработка на слики и компјутерска визија работат побрзо и поефикасно на слики во сива скала.

3. **Конвертирање на сликите:**

- Секој елемент во листата од скалирани слики се обработува со функција за конверзија, која ја претвора сликата од BGR формат во сива скала. Оваа функција ги комбинира трите бојни компоненти во една, според одредени формули што ги земаат предвид визуелните својства на човечкото око.
- Резултатот од оваа конверзија е нова листа која ги содржи сите слики, но во сива скала наместо во боја.

На овој начин, сликите се подготвуваат за понатамошна обработка, каде што не е потребна информацијата за боите, и се овозможува поефикасна и побрза работа со сликите.

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.

```
26    sift_detector = cv2.SIFT_create()
27    keypoints_list = []
28    descriptors_list = []
29    for gray_image in gray_images:
30        kp, desc = sift_detector.detectAndCompute(gray_image, None)
31        keypoints_list.append(kp)
32        descriptors_list.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:

```
30        kp, desc = sift_detector.detectAndCompute(gray_image, None)
31        keypoints_list.append(kp)
32        descriptors_list.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_list**` и `**descriptors_list**`. 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

```
34    bf_matcher = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
35    descriptor_matches = []
36    for i in range(len(descriptors_list) - 1):
37        descriptor_matches.append(bf_matcher.match(descriptors_list[i], descriptors_list[i + 1]))
38
```

- 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_list) - 1`). Во секоја итерација се извршува споредување на дескрипторите помеѓу две соседни слики.
- Во секоја итерација на циклусот, со помош на методот `**match(descriptors_list[i], descriptors_list[i+1])**` на објектот `bf_matcher` 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 `**descriptor_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`.

```
39  sorted_descriptor_matches = [sorted(match, key=lambda x: x.distance) for match in descriptor_matches]
```

Овој дел од кодот ги сортира паровите дескриптори (matches) за секој пар на слики според нивната растојание. За секој пар на слики, се земаат добиените дескриптори и се сортираат така што најдобрите парови се стават на почетокот на списокот. Ова се прави за да се идентификуваат најсигурните парови на клучни точки меѓу сликите, кои ќе се користат за понатамошно спојување на сликите. Со ова сортирање се овозможува прецизно и точно подредување на сликите во процесот на создавање на панорама.

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.

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:

```
41  top_N_matches = 50
42  matches_images = []
43  for i in range(len(sorted_descriptor_matches)):
44      match_img = cv2.drawMatches(
45          scaled_images[i], keypoints_list[i], scaled_images[i + 1], keypoints_list[i + 1],
46          sorted_descriptor_matches[i][:top_N_matches], outImg: None, flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
47      matches_images.append(match_img)
48
```

- Командата во ред 41 поставува број на најдобри парови на клучни точки кои ќе бидат прикажани за секој пар слики. Во овој случај, поставена е вредност од 50.
- Командара во ред 42 иницијализира празна листа matches_images. Оваа листа ќе ги содржи сликите кои визуелно ги прикажуваат паровите на клучни точки.

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:

- scaled_images[i]: Првата слика од парот.

- keypoints_list[i]: Клучни точки од првата слика.
- scaled_images[i + 1]: Втората слика од парот.
- keypoints_list[i + 1]: Клучни точки од втората слика.
- sorted_descriptor_matches[i][:top_N_matches]: Најдобрите top_N_matches парови на клучни точки меѓу овие две слики.
- None: Опционален аргумент за излезната слика (може да се прескокне).
- flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS: Знаме кое кажува на функцијата да не ги прикажува поединечните клучни точки, туку само линиите кои ги поврзуваат паровите на клучни точки.

- Командата во ред 47 ја додава новосоздадената слика match_img во листата matches_images. Со ова, листата matches_images постепено се полни со слики кои визуелно ги прикажуваат паровите на клучни точки за секој пар слики.

```
49    stitcher_instance = cv2.Stitcher.create()
50    stitch_status, panorama = stitcher_instance.stitch(scaled_images)
```

- Во овој дел од кодот 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 `**panorama**`, а статусот на креирањето на панорамски слики се зачувува во променливата `**status**`.

```
52    if stitch_status == cv2.Stitcher_OK:
53        cv2.imshow( winname: 'Panorama Result', panorama)
54    else:
55        print(f"Stitching failed with status {stitch_status}")
56
57    for idx, match_img in enumerate(matches_images):
58        cv2.imshow( winname: f'Keypoint Matches {idx + 1}-{idx + 2}', match_img)
59
60    cv2.waitKey(0)
61    cv2.destroyAllWindows()
```

Овој дел од кодот ја проверува успешноста на процесот на спојување на сликите и ги прикажува резултатите.

- **if stitch_status == cv2.Stitcher_OK:**: Оваа команда проверува дали статусот на спојување е успешен (означено со cv2.Stitcher_OK).

- Ако е успешен, **cv2.imshow('Panorama Result', panorama)**: Прикажува го резултатот на спојувањето во прозорец со наслов 'Panorama Result'. Сликата panorama е резултат од спојувањето на сите влезни слики.
- Ако не е успешен, **print(f"Stitching failed with status {stitch_status}")**: Печати порака која информира дека спојувањето не успеало, заедно со статусниот код.
- **for idx, match_img in enumerate(matches_images):**: Оваа команда иницијализира циклус кој поминува низ сите слики во листата matches_images. Променливите idx и match_img ги означуваат индексот и сликата за секоја итерација.
- **cv2.imshow(f'Keypoint Matches {idx + 1}-{idx + 2}', match_img)**: Во секоја итерација, оваа команда прикажува слика која ги визуелизира паровите на клучни точки меѓу две соседни слики. Прозорецот во кој се прикажува сликата е насловен со 'Keypoint Matches' и броевите на сликите (пример: 'Keypoint Matches 1-2', 'Keypoint Matches 2-3').
- **cv2.waitKey(0)**: Оваа команда чека корисникот да притисне било кое копче пред да продолжи. Ова овозможува корисникот да ги види сите прикажани слики.
- **cv2.destroyAllWindows()**: Оваа команда ги затвора сите отворени прозорци откако корисникот ќе притисне копче.

5 Резултати

Во овој дел се презентирани резултатите добиени од процесот на креирање панорамски слики. Преку употреба на SIFT алгоритмот за детекција и совпаѓање на клучни точки, како и со примена на напредни техники за спојување, овозможено е креирање на висококвалитетни панорамски слики. Анализата и визуелизацијата на добиените резултати овозможуваат детално разбирање на успешноста и ефикасноста на применетите методи и алатки. Резултатите се прикажани преку визуелни примери и графички претстави, што дополнително ја потврдува нивната точност и применливост во различни сценарија.

Извршување на скриптата:

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

```
PS C:\Users\User\Desktop\panorama> python panorama.py image1.jpg image2.jpg image3.jpg
```

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

Откако кодот ќе заврши со извршување, резултатот (панорамската слика) ќе биде прикажан во нов прозорец.

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.

Слика1. Прва влезна слика

Слика2. Втора влезна слика

Слика3. Трета влезна слика

Како резултат на претходно објаснетиот код, добивме резултантна панорамска слика која ги обединува влезните слики преку процес на спојување. Во овој процес, секоја влезна слика беше претворена во сива скала, а клучните точки на сликите беа детектирани и совпаѓани користејќи SIFT (Scale-Invariant Feature Transform) алгоритам. Потоа, со примена на алгоритам за спојување, сликите беа комбинирани во една непрекината панорама.

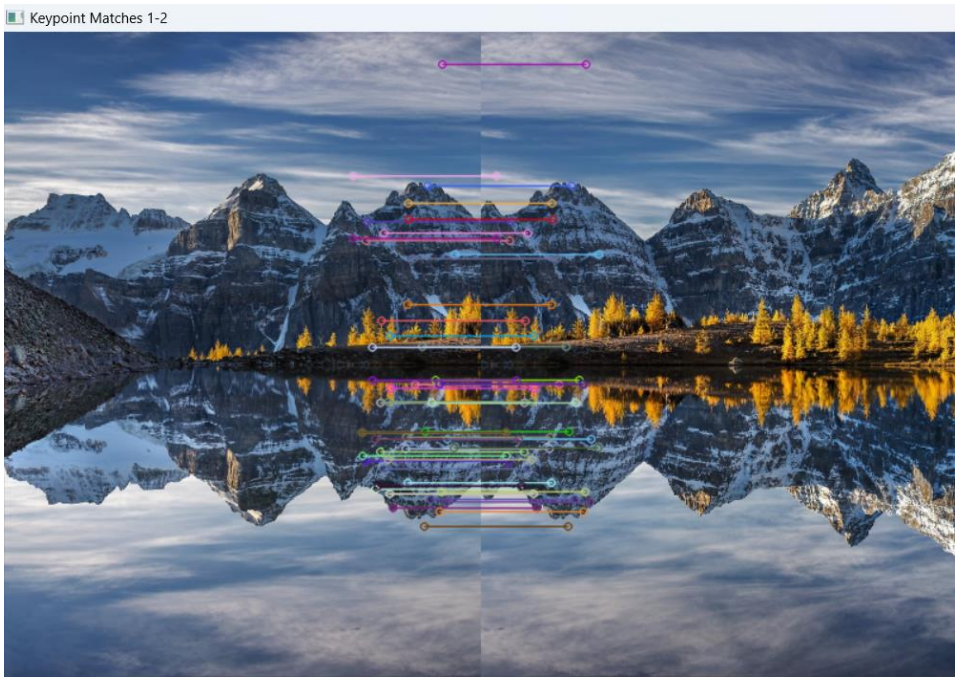
1. **Квалитет на Спојување:** Спојувањето е извршено со висок квалитет, без видливи артефакти или грешки на преклопување.
2. **Преклопувачки Полиња:** Влезните слики имаа доволно преклопувачки полиња за да се овозможи непречено спојување и минимизирање на деформациите.
3. **Клучни Точки:** Клучните точки на секоја слика беа успешно совпаѓани, што е визуелизирано преку сликите со нацртани клучни точки

Примената на SIFT алгоритмот и техниките за спојување на слики овозможија креирање на квалитетна панорамска слика, која нуди поширок и поцелосен приказ на сцената

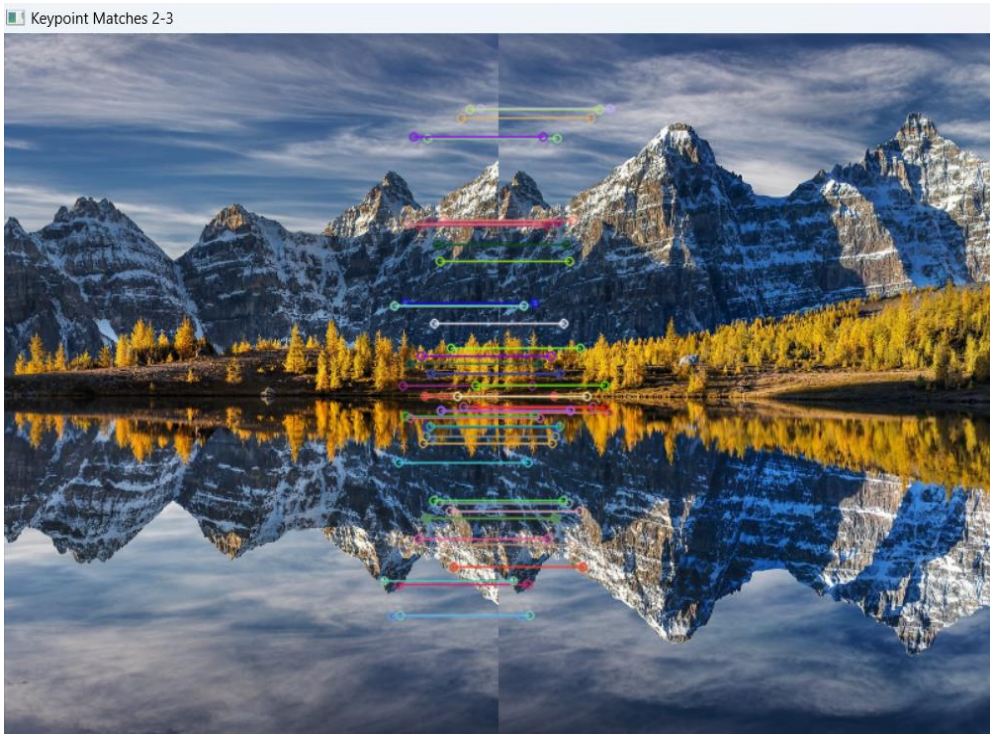
Исто така, се прикажуваат и пар на последователни слики со соодветно нацртани совпаднати клучни точки.

Резултантната панорамска слика:

Слика4. Панорамска слика



Слика5. Совпаднати клучни точки на првата и втората слика



Слика6. Совпаднати клучни точки на втората и третата слика

6 Заклучок

Во оваа семинарска работа, детално е објаснет процесот на креирање панорамски слики преку употреба на современи алгоритми и алатки за компјутерска визија. Имплементацијата на SIFT (Scale-Invariant Feature Transform) алгоритмот овозможи ефикасна детекција и совпаѓање на клучни точки, што е од суштинско значење за спојување на слики со висок квалитет.

Со користење на библиотеката OpenCV, успешно беа реализирани сите чекори од процесот, вклучувајќи вчитување и обработка на слики, детекција на клучни точки, совпаѓање на дескриптори, и финално спојување на сликите во една панорамска слика. Овие техники овозможија добивање на композитни слики со поширок агол на гледање и без значителни артефакти или грешки.

Резултатите покажуваат дека соодветното користење на алгоритмите и библиотеките може да доведе до создавање на висококвалитетни панорамски слики, што има широка примена во различни области како што се фотографијата, виртуелната реалност, и 3D моделирањето. Ова демонстрира дека компјутерската визија нуди моќни алатки за обработка и анализа на слики, што може да се искористи за различни иновативни апликации.

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-techniqueimage-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