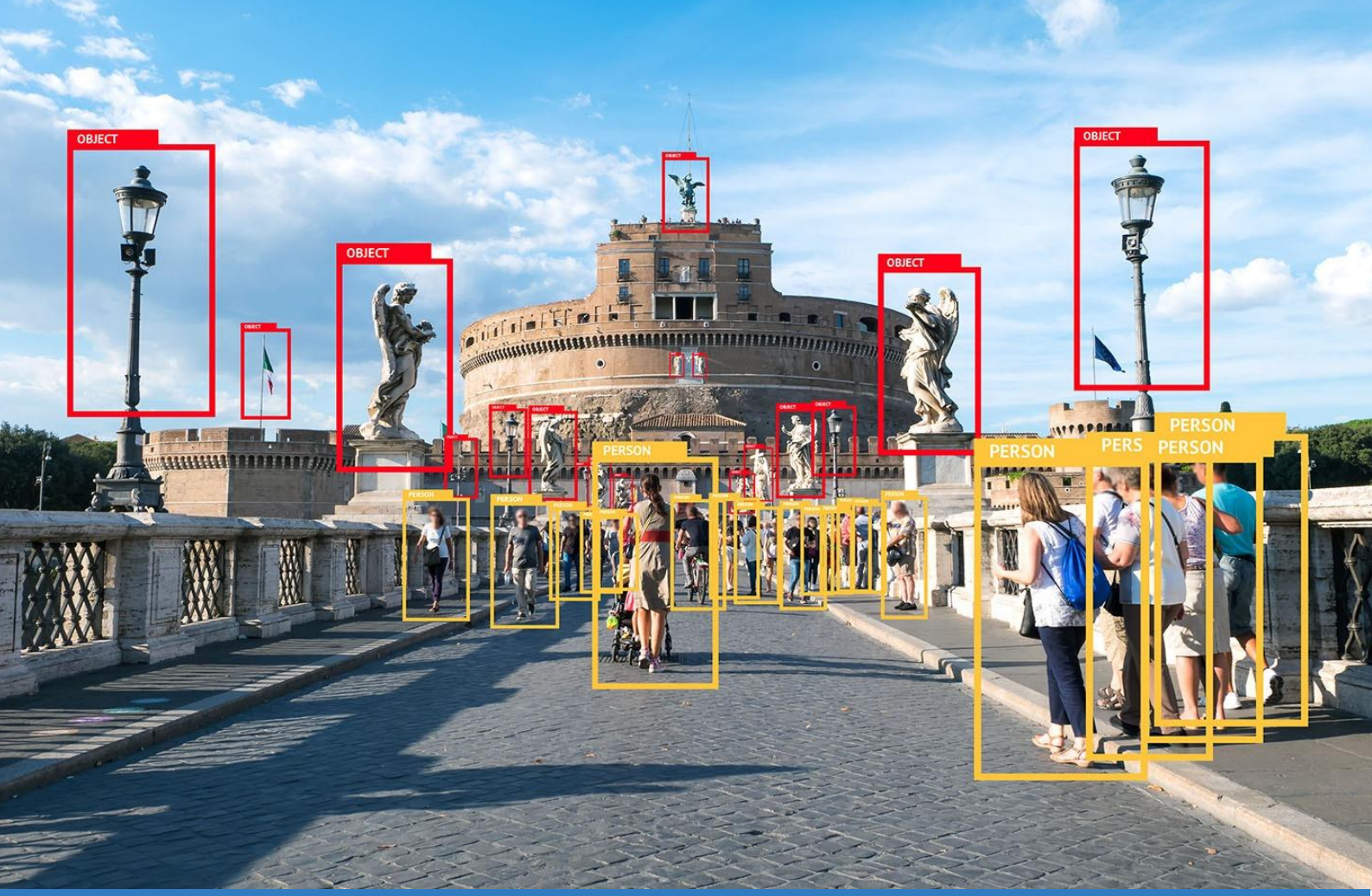


Όραση Υπολογιστών

2η Εργασία

Εμμανουηλίδης Κωνσταντίνος

A.M. 57315

ΠΕΡΙΕΧΟΜΕΝΑ

ΘΕΩΡΗΤΙΚΗ ΑΝΑΛΥΣΗ

ΠΑΝΟΡΑΜΑ

Το πανόραμα είναι οποιαδήποτε ευρεία γωνία ή αναπαράσταση ενός φυσικού χώρου. Με τον γενικό τίτλο πανοραμική φωτογραφία, ουσιαστικά, αναφερόμαστε στο είδος της φωτογραφίας που χρησιμοποιώντας ειδικό εξοπλισμό μας προσφέρει απεριόριστη ή σχεδόν απεριόριστη θέα του ορίζοντα. Μια πανοραμική φωτογραφία έχει συνήθως αναλογία πλάτους ύψους μεγαλύτερη από 2 προς 1.

ΑΝΙΧΝΕΥΤΕΣ & ΠΕΡΙΓΡΑΦΕΙΣ

Οι ανιχνευτές (detectors) χρησιμεύουν στην ανίχνευση ακμών και κορυφών και εξάγουν τις θέσεις των σημαντικών περιοχών της εικόνας.Χαρακτηριστικό παράδειγμα ανιχνευτών αποτελούν οι SIFT και SURF, οι οποίοι είναι ιδιαίτερα χρήσιμοι στην κατασκευή πανοραμικών φωτογραφιών.

Οι περιγραφείς (descriptors) δοθέντος μιας εικόνας παράγουν διανύσματα χαρακτηριστικών στα οποία βρίσκεται κωδικοποιημένη πληροφορία αναφορικά με το τι βρίσκεται γύρω από τους ανιχνευτές. Με αυτό τον τρόπο μπορούμε να αντιστοιχίσουμε ανιχνευτές σε διαφορετικές εικόνες επιτυγχάνοντας έτσι τη σύνθεση μιας πανοραμικής εικόνας.

SIFT DETECTOR

Ο SIFT κωδικοποιεί πληροφορίες σχετικά με τις κλίσεις στις τοπικές γειτονιές της εικόνας και τους αριθμούς του διανύσματος χαρακτηριστικών.

Ο SIFT συγκρίνει μεμονωμένα κάθε χαρακτηριστικό από τη νέα εικόνα με αυτά από την προηγούμενη εικόνα και το αντιστοιχίζει με βάση την Ευκλείδεια απόσταση των διανυσμάτων των χαρακτηριστικών τους. Από το σύνολο των αντιστοιχίσεων προσδιορίζονται τα keypoints, τα οποία συμφωνούν στο αντικείμενο, στη θέση, στην κλίμακα και στον προσανατολισμό τους με αυτά της νέας εικόνας και έτσι μπορεί να επιτευχθεί αναγνώριση ενός αντικειμένου σε μια νέα εικόνα.

SURF DETECTOR

Ο SURF(speeded-up version of SIFT) αποτελεί πιο γρήγορη εκδοχή του SIFT. Ανιχνεύει κι αυτός τα σημεία ενδιαφέροντος μιας εικόνας χρησιμοποιώντας τον ανιχνευτή Hessian blob, ο οποίος μπορεί να υπολογιστεί κάνοντας χρήση μιας ολοκληρωμένης εικόνα. Στη συνέχεια, χρησιμοποιώντας την τεχνική πυραμίδων πολλαπλών αναλύσεων αντιγράφεται η αρχική εικόνα με σχήμα πυραμιδικού Gauss ή Laplacian Pyramid για να αποκτηθεί μια εικόνα με το ίδιο μέγεθος αλλά μειωμένο εύρος ζώνης. Η προσέγγιση του Laplacian of Gaussian γίνεται με Box Filter, η συνέλιξη του οποίου μπορεί εύκολα να υπολογιστεί με integral images και να γίνει παράλληλα για διαφορετικές κλίμακες.

Ομογραφία

Η συνάρτηση findHomography βρίσκει και επιστρέφει τον τρόπο με τον οποίο πρέπει να υπάρξει μετασχηματισμός προοπτικής μεταξύ του πλέγματος πηγής και του προορισμού, ώστε να έχουμε ελάχιστο σφάλμα επαναπροβολής.

cv.warpPerspective()

Η συνάρτηση warpPerspective εφαρμόζει ένα μετασχηματισμό προοπτικής σε μια εικόνα και μετασχηματίζει την εικόνα πηγής.

Άσκηση 1

Η παρούσα εργασία αφορά στην κατασκευή πανοραμικών φωτογραφιών χρησιμοποιώντας τους αλγορίθμους SIFT και SURF σε επιμέρους εικόνες που έχουν δοθεί εξαρχής. Τα βήματα που ακολουθήθηκαν για την υλοποίηση των ζητουμένων της άσκηση φαίνονται παρακάτω.

ΒΗΜΑΤΑ

- ✓ Αρχικά, φορτώνουμε τις εικόνες και υπολογίζουμε τους ανιχνευτές και τους περιγραφείς τους με χρήση συναρτήσεων της OpenCV
- ✓ Αντιστοιχίζουμε στις 2 πρώτες εικόνες τους ανιχνευτές που ταιριάζουν με βάση τους περιγραφείς κάνοντας χρήση της συνάρτησης matches που είναι υλοποιημένη στην αρχή του κώδικα
- ✓ Τροποποιούμε την δεύτερη εικόνα, έτσι ώστε να ταιριάξει στην πρώτη με τη βοήθεια συναρτήσεων της OpenCV
- ✓ Τοποθετούμε την πρώτη εικόνα και πάνω της προσαρμόζουμε τη δεύτερη εικόνα
- ✓ Επαναλαμβάνουμε την ίδια διαδικασία με την τρίτη εικόνα και με το αποτέλεσμα που πήραμε παραπάνω
- ✓ Επαναλαμβάνουμε την ίδια διαδικασία με την τέταρτη εικόνα και με το προηγούμενο αποτέλεσμα
- ✓ Εμφανίζουμε το αποτέλεσμα το οποίο αναμένεται να είναι η πανοραμική φωτογραφία

ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΗ ΥΛΟΠΟΙΗΣΗ

ΥΛΟΠΟΙΗΣΗ ΜΕ SIFT

```python
## Importing necessary libraries ##
import numpy as np
import cv2 as cv

## Function for finding matches between to images ##
def match(d1, d2):
 n1 = d1.shape[0]
 n2 = d2.shape[0]
 matches = []
 for i in range(n1):
  distances = np.sum(np.abs(d2 - d1[i, :]), axis=1)
  i2 = np.argmin(distances)
  mindist2 = distances[i2]
  distances[i2] = np.inf
  i3 = np.argmin(distances)
  mindist3 = distances[i3]
  if mindist2 / mindist3 < 0.5:
   matches.append(cv.DMatch(i, i2, mindist2))
 return matches

# Creating SIFT Object for later use
sift = cv.xfeatures2d_SIFT.create()
```

```python
## Processing of first image ##
# Reading the file of the image
img1 = cv.imread('yard-03.png', cv.IMREAD_GRAYSCALE)
# Detecting keypoints and descriptors of first image
kp1 = sift.detect(img1)
desc1 = sift.compute(img1, kp1)

## Processing of second image ##
# Reading the file of the image
img2 = cv.imread('yard-02.png',cv.IMREAD_GRAYSCALE)
# Detecting keypoints and descriptors of second image
kp2 = sift.detect(img2)
desc2 = sift.compute(img2, kp2)

## Processing of third image ##
# Reading the file of the image
img3 = cv.imread('yard-01.png',cv.IMREAD_GRAYSCALE)
# Detecting keypoints and descriptors of third image
kp3 = sift.detect(img3)
desc3 = sift.compute(img3, kp3)

## Processing of fourth image ##
# Reading the file of the image
img4 = cv.imread('yard-00.png',cv.IMREAD_GRAYSCALE)
# Detecting keypoints and descriptors of fourth image
kp4 = sift.detect(img4)
desc4 = sift.compute(img4, kp4)
```

```
## Joining first two images ##
# Finding matches between img1 and img2
matches1 = match(desc1[1], desc2[1])
img_pt1 = np.array([kp1[x.queryIdx].pt for x in matches1])
img_pt2 = np.array([kp2[x.trainIdx].pt for x in matches1])


# Finding homography
M, mask = cv.findHomography(img_pt2, img_pt1, cv.RANSAC)

img5 = cv.warpPerspective(img2, M, (img1.shape[1]+1000, img1.shape[0]+1000))
img5[0: img2.shape[0], 0: img2.shape[1]] = img1



## Joining previous joined image with img3 ##
# Detecting keypoints and descriptors of image
kp5 = sift.detect(img5)
desc5 = sift.compute(img5, kp5)
# Finding matches between img5 and img3
matches2 = match(desc5[1], desc3[1])
img_pt5 = np.array([kp5[x.queryIdx].pt for x in matches2])
img_pt3 = np.array([kp3[x.trainIdx].pt for x in matches2])

# Finding homography
M, mask = cv.findHomography(img_pt3, img_pt5, cv.RANSAC)
```

```
# Wrap images appropriately
img6 = cv.warpPerspective(img3, M, (img5.shape[1]+1000, img5.shape[0]+1000))

img5_x=img5.shape[0]
img5_y=img5.shape[1]
img_black_white = np.ones((img5_x,img5_y), dtype=np.uint8)
for x in range(img5_x):
 for y in range(img5_y):
  if img5[x, y] == 0:
    img_black_white[x,y] = 0
strel = np.ones((7,7), np.uint8)
erode = cv.morphologyEx(img_black_white, cv.MORPH_ERODE, strel)
for x in range(erode.shape[0]):
 for y in range(erode.shape[1]):
  if erode[x, y] != 0:
   img6[x, y] = img5[x, y]
```

```
## Joining previous joined image with img4 ##
# Detecting keypoints and descriptors of image
kp6 = sift.detect(img6)
desc6 = sift.compute(img6, kp6)
# Finding matches
matches3 = match(desc6[1], desc4[1])
img_pt6 = np.array([kp6[x.queryIdx].pt for x in matches3])
img_pt4 = np.array([kp4[x.trainIdx].pt for x in matches3])
# Finding homography
M, mask = cv.findHomography(img_pt4, img_pt6, cv.RANSAC)

# Join img4 and img6 to get the final panorama img7
img7 = cv.warpPerspective(img4, M, (img6.shape[1]+1000, img6.shape[0]+1000))
kp7 = sift.detect(img7)
desc7 = sift.compute(img7, kp7)

img6_x=img6.shape[0]
img6_y=img6.shape[1]
img_black_white = np.ones((img6_x,img6_y), dtype=np.uint8)
for x in range(img6_x):
 for y in range(img6_y):
  if img6[x, y] == 0:
   img_black_white[x,y] = 0
```

```
## Erosing img_black_white to fill ##
strel = np.ones((11,11), np.uint8)
erode2 = cv.morphologyEx(img_black_white, cv.MORPH_ERODE, strel)
for x in range(erode2.shape[0]):
 for y in range(erode2.shape[1]):
  if erode2[x, y] != 0:
   img7[x, y] = img6[x, y]
```

```
## Show images in different windows ##
# Show first image
cv.namedWindow('main1', cv.WINDOW_NORMAL)
cv.imshow('main1', img1)
cv.waitKey(0)
# Show second image
cv.namedWindow('main2', cv.WINDOW_NORMAL)
cv.imshow('main2', img2)
cv.waitKey(0)
# Show third image
cv.namedWindow('main3', cv.WINDOW_NORMAL)
cv.imshow('main3', img3)
cv.waitKey(0)
# Show fourth image
cv.namedWindow('main4', cv.WINDOW_NORMAL)
cv.imshow('main4', img4)
cv.waitKey(0)

cv.namedWindow('panorama', cv.WINDOW_NORMAL)
cv.imshow('panorama', img7)
cv.imwrite('panorama_sift.png', img7)
cv.waitKey(0)
```

ΥΛΟΠΟΙΗΣΗ ΜΕ SURF

```
# <---------------------------------------- SURF -

## Importing necessary libraries ##
import numpy as np
import cv2 as cv

## Function for finding matches between to images ##
def match(d1, d2):
 n1 = d1.shape[0]
 n2 = d2.shape[0]
 matches = []
 for i in range(n1):
  distances = np.sum(np.abs(d2 - d1[i, :]), axis=1)
  i2 = np.argmin(distances)
  mindist2 = distances[i2]
  distances[i2] = np.inf
  i3 = np.argmin(distances)
  mindist3 = distances[i3]
  if mindist2 / mindist3 < 0.5:
   matches.append(cv.DMatch(i, i2, mindist2))
 return matches

# Creating SURF Object for later use
surf = cv.xfeatures2d_SURF.create()
```

```
## Processing of first image ##
# Reading the file of the image
img1 = cv.imread('yard-03.png', cv.IMREAD_GRAYSCALE)
# Detecting keypoints and descriptors of first image
kp1 = surf.detect(img1)
desc1 = surf.compute(img1, kp1)

## Processing of second image ##
# Reading the file of the image
img2 = cv.imread('yard-02.png',cv.IMREAD_GRAYSCALE)
# Detecting keypoints and descriptors of second image
kp2 = surf.detect(img2)
desc2 = surf.compute(img2, kp2)

## Processing of third image ##
# Reading the file of the image
img3 = cv.imread('yard-01.png',cv.IMREAD_GRAYSCALE)
# Detecting keypoints and descriptors of third image
kp3 = surf.detect(img3)
desc3 = surf.compute(img3, kp3)

## Processing of fourth image ##
# Reading the file of the image
img4 = cv.imread('yard-00.png',cv.IMREAD_GRAYSCALE)
# Detecting keypoints and descriptors of fourth image
kp4 = surf.detect(img4)
desc4 = surf.compute(img4, kp4)
```

```
## Joining first two images ##
# Finding matches between img1 and img2
matches1 = match(desc1[1], desc2[1])
img_pt1 = np.array([kp1[x.queryIdx].pt for x in matches1])
img_pt2 = np.array([kp2[x.trainIdx].pt for x in matches1])


# Finding homography
M, mask = cv.findHomography(img_pt2, img_pt1, cv.RANSAC)

img5 = cv.warpPerspective(img2, M, (img1.shape[1]+1000, img1.shape[0]+1000))
img5[0: img2.shape[0], 0: img2.shape[1]] = img1


## Joining previous joined image with img3 ##
# Detecting keypoints and descriptors of image
kp5 = surf.detect(img5)
desc5 = surf.compute(img5, kp5)
# Finding matches between img5 and img3
matches2 = match(desc5[1], desc3[1])
img_pt5 = np.array([kp5[x.queryIdx].pt for x in matches2])
img_pt3 = np.array([kp3[x.trainIdx].pt for x in matches2])

# Finding homography
M, mask = cv.findHomography(img_pt3, img_pt5, cv.RANSAC)
```

```
# Wrap images approprietely
img6 = cv.warpPerspective(img3, M, (img5.shape[1]+1000, img5.shape[0]+1000))

'''
## Simple way to join the two images but not efficient due to black lines between images ##
for x in range (img5.shape[0]):
 for y in range (img5.shape[1]):
 if(img5[x,y]!=0):
 img6[x,y]=img5[x,y]
'''

img5_x=img5.shape[0]
img5_y=img5.shape[1]
img_black_white = np.ones((img5_x,img5_y), dtype=np.uint8)

for x in range(img5_x):
 for y in range(img5_y):
  if img5[x, y] == 0:
   img_black_white[x,y] = 0

strel = np.ones((7,7), np.uint8)
erode = cv.morphologyEx(img_black_white, cv.MORPH_ERODE, strel)
for x in range(erode.shape[0]):
 for y in range(erode.shape[1]):
  if erode[x, y] != 0:
   img6[x, y] = img5[x, y]
```

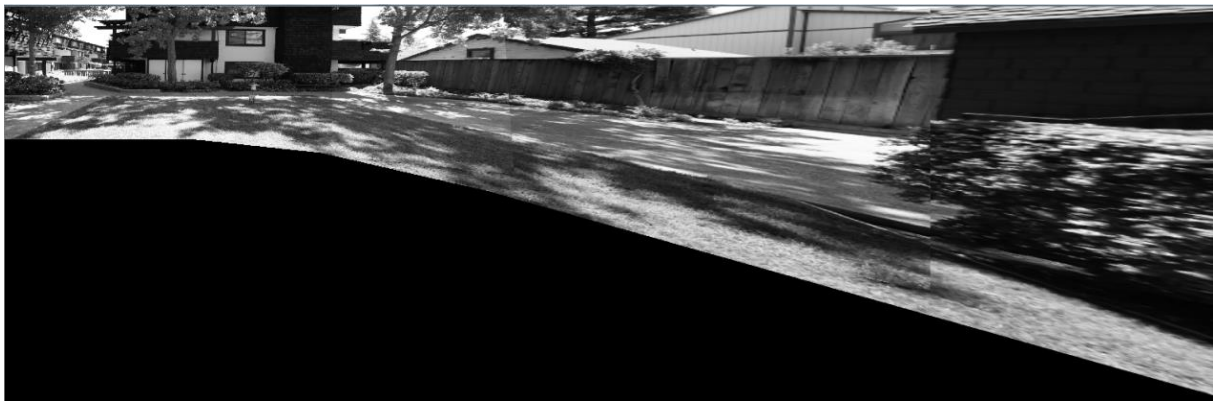
```
## Joining previous joined image with img4 ##
# Detecting keypoints and descriptors of image
kp6 = surf.detect(img6)
desc6 = surf.compute(img6, kp6)
# Finding matches
matches3 = match(desc6[1], desc4[1])
img_pt6 = np.array([kp6[x.queryIdx].pt for x in matches3])
img_pt4 = np.array([kp4[x.trainIdx].pt for x in matches3])
# Finding homography
M, mask = cv.findHomography(img_pt4, img_pt6, cv.RANSAC)

# Join img4 and img6 to get the final panorama img7
img7 = cv.warpPerspective(img4, M, (img6.shape[1]+1000, img6.shape[0]+1000))
kp7 = surf.detect(img7)
desc7 = surf.compute(img7, kp7)
'''
## Simple way to join the two images but not efficient due to black lines between images ##
for x in range (img6.shape[0]):
 for y in range (img6.shape[1]):
 if(img6[x,y]!=0):
  img7[x,y]=img6[x,y]
'''
img6_x=img6.shape[0]
img6_y=img6.shape[1]
img_black_white = np.ones((img6_x,img6_y), dtype=np.uint8)
```

```
## Erosing img_black_white to fill ##
strel = np.ones((11,11), np.uint8)
erode2 = cv.morphologyEx(img_black_white, cv.MORPH_ERODE, strel)
for x in range(erode2.shape[0]):
 for y in range(erode2.shape[1]):
  if erode2[x, y] != 0:
    img7[x, y] = img6[x, y]


## Show images in different windows ##
# Show first image
cv.namedWindow('main1', cv.WINDOW_NORMAL)
cv.imshow('main1', img1)
cv.waitKey(0)
# Show second image
cv.namedWindow('main2', cv.WINDOW_NORMAL)
cv.imshow('main2', img2)
cv.waitKey(0)
# Show third image
cv.namedWindow('main3', cv.WINDOW_NORMAL)
cv.imshow('main3', img3)# Εμφάσιση της τρίτης εικόνας του ξενοδοχείου
cv.waitKey(0)
# Show fourth image
cv.namedWindow('main4', cv.WINDOW_NORMAL)
cv.imshow('main4', img4)
cv.waitKey(0)

cv.namedWindow('panorama', cv.WINDOW_NORMAL)
cv.imshow('panorama', img7)
cv.imwrite('panorama_surf.png', img7)
cv.waitKey(0)
```

ΑΡΧΙΚΕΣ ΕΙΚΟΝΕΣ

ΑΠΟΤΕΛΕΣΜΑΤΑ

SIFT



SURF

IMAGE COMPOSITE EDITOR

ΣΥΜΠΕΡΑΣΜΑΤΑ

Στη συγκεκριμένη περίπτωση το αποτέλεσμα των αλγορίθμων SIFT και SURF είναι περίπου το ίδιο. Με το SURF οι εικόνες είναι λίγο πιο μακρόστενες και πλατιές. Και στις δύο περιπτώσεις είναι ορατή η ένωση των εικόνων, καθώς δεν γίνεται εξομάλυνση των διαφορών στα χρώματα. Επίσης παρατηρείται ένα «τράβηγμα» της εικόνας στο δεξιά τμήμα της καθώς και μαύρες περιοχές. Σε περίπτωση που θέλουμε να εξαφανιστούν οι μαύρες περιοχές μπορούμε να τις αντικαταστήσουμε με τα pixels της αρχικής εικόνας.

Αντίθετα, το πρόγραμμα παράγει πανοραμική φωτογραφία με τέλεια ενωμένες τις εικόνες χωρίς να είναι διακριτές οι διαφορές μεταξύ τους. Η εικόνα πλησιάζει των αλγορίθμων αλλά είναι πιο μακρόστενη.

ΠΑΝΟΡΑΜΑ ΕΠΙΠΛΕΟΝ ΕΙΚΟΝΩΝ

```python
## Importing necessary libraries ##
import numpy as np
import cv2 as cv

## Function for finding matches between to images ##
def match(d1, d2):
 n1 = d1.shape[0]
 n2 = d2.shape[0]
 matches = []
 for i in range(n1):
  distances = np.sum(np.abs(d2 - d1[i, :]), axis=1)
  i2 = np.argmin(distances)
  mindist2 = distances[i2]
  distances[i2] = np.inf
  i3 = np.argmin(distances)
  mindist3 = distances[i3]
  if mindist2 / mindist3 < 0.5:
   matches.append(cv.DMatch(i, i2, mindist2))
 return matches

# Creating SIFT Object for later use
sift = cv.xfeatures2d_SIFT.create()
```

```python
## Processing of first image ##
# Reading the file of the image
img1 = cv.imread('foto1.png', cv.IMREAD_GRAYSCALE)
# Detecting keypoints and descriptors of first image
kp1 = sift.detect(img1)
desc1 = sift.compute(img1, kp1)

## Processing of second image ##
# Reading the file of the image
img2 = cv.imread('foto2.png',cv.IMREAD_GRAYSCALE)
# Detecting keypoints and descriptors of second image
kp2 = sift.detect(img2)
desc2 = sift.compute(img2, kp2)

## Processing of third image ##
# Reading the file of the image
img3 = cv.imread('foto3.png',cv.IMREAD_GRAYSCALE)
# Detecting keypoints and descriptors of third image
kp3 = sift.detect(img3)
desc3 = sift.compute(img3, kp3)

## Processing of fourth image ##
# Reading the file of the image
img4 = cv.imread('foto4.png',cv.IMREAD_GRAYSCALE)
# Detecting keypoints and descriptors of fourth image
```

```
## Joining first two images ##
# Finding matches between img1 and img2
matches1 = match(desc1[1], desc2[1])
img_pt1 = np.array([kp1[x.queryIdx].pt for x in matches1])
img_pt2 = np.array([kp2[x.trainIdx].pt for x in matches1])


# Finding homography
M, mask = cv.findHomography(img_pt2, img_pt1, cv.RANSAC)

img5 = cv.warpPerspective(img2, M, (img1.shape[1]+1000, img1.shape[0]+1000))
img5[0: img2.shape[0], 0: img2.shape[1]] = img1



## Joining previous joined image with img3 ##
# Detecting keypoints and descriptors of image
kp5 = sift.detect(img5)
desc5 = sift.compute(img5, kp5)
# Finding matches between img5 and img3
matches2 = match(desc5[1], desc3[1])
img_pt5 = np.array([kp5[x.queryIdx].pt for x in matches2])
img_pt3 = np.array([kp3[x.trainIdx].pt for x in matches2])

# Finding homography
M, mask = cv.findHomography(img_pt3, img_pt5, cv.RANSAC)
```

```
# Wrap images appropriately
img6 = cv.warpPerspective(img3, M, (img5.shape[1]+1000, img5.shape[0]+1000))

img5_x=img5.shape[0]
img5_y=img5.shape[1]
img_black_white = np.ones((img5_x,img5_y), dtype=np.uint8)
for x in range(img5_x):
 for y in range(img5_y):
  if img5[x, y] == 0:
    img_black_white[x,y] = 0
strel = np.ones((7,7), np.uint8)
erode = cv.morphologyEx(img_black_white, cv.MORPH_ERODE, strel)
for x in range(erode.shape[0]):
 for y in range(erode.shape[1]):
  if erode[x, y] != 0:
   img6[x, y] = img5[x, y]
```

```
## Joining previous joined image with img4 ##
# Detecting keypoints and descriptors of image
kp6 = sift.detect(img6)
desc6 = sift.compute(img6, kp6)
# Finding matches
matches3 = match(desc6[1], desc4[1])
img_pt6 = np.array([kp6[x.queryIdx].pt for x in matches3])
img_pt4 = np.array([kp4[x.trainIdx].pt for x in matches3])
# Finding homography
M, mask = cv.findHomography(img_pt4, img_pt6, cv.RANSAC)

# Join img4 and img6 to get the final panorama img7
img7 = cv.warpPerspective(img4, M, (img6.shape[1]+1000, img6.shape[0]+1000))
kp7 = sift.detect(img7)
desc7 = sift.compute(img7, kp7)

img6_x=img6.shape[0]
img6_y=img6.shape[1]
img_black_white = np.ones((img6_x,img6_y), dtype=np.uint8)
for x in range(img6_x):
 for y in range(img6_y):
  if img6[x, y] == 0:
   img_black_white[x,y] = 0
```

```
## Erosing img_black_white to fill ##
strel = np.ones((11,11), np.uint8)
erode2 = cv.morphologyEx(img_black_white, cv.MORPH_ERODE, strel)
for x in range(erode2.shape[0]):
 for y in range(erode2.shape[1]):
  if erode2 [x, y] != 0:
   img7[x, y] = img6[x, y]
```

```
## Show images in different windows ##
# Show first image
cv.namedWindow('main1', cv.WINDOW_NORMAL)
cv.imshow('main1', img1)
cv.waitKey(0)
# Show second image
cv.namedWindow('main2', cv.WINDOW_NORMAL)
cv.imshow('main2', img2)
cv.waitKey(0)
# Show third image
cv.namedWindow('main3', cv.WINDOW_NORMAL)
cv.imshow('main3', img3)
cv.waitKey(0)
# Show fourth image
cv.namedWindow('main4', cv.WINDOW_NORMAL)
cv.imshow('main4', img4)
cv.waitKey(0)

cv.namedWindow('panorama', cv.WINDOW_NORMAL)
cv.imshow('panorama', img7)
cv.imwrite('panorama_surf.png', img7)
cv.waitKey(0)
```

ΑΠΟΤΕΛΕΣΜΑΤΑ

IMAGE COMPOSITE EDITOR

ΣΥΜΠΕΡΑΣΜΑΤΑ

Παρατηρούμε παρόμοια αποτελέσματα με αυτά που είχαμε στις αρχικές εικόνες που χρησιμοποιήθηκαν για το πανόραμα.

Τα αποτελέσματα των αλγορίθμων SIFT και SURF δεν παρουσιάζει μεγάλη διαφορά. Με το SURF οι εικόνες είναι λίγο πιο μακρόστενες και πλατιές. Και στις δύο περιπτώσεις είναι ορατή η ένωση των εικόνων, καθώς δεν γίνεται εξομάλυνση των διαφορών στα χρώματα.

Αντίθετα, το πρόγραμμα παράγει πανοραμική φωτογραφία με τέλεια ενωμένες τις εικόνες χωρίς να είναι διακριτές οι διαφορές μεταξύ τους. Η εικόνα πλησιάζει των αλγορίθμων αλλά είναι πιο μακρόστενη.

Σχετική Βιβλιογραφία

1. Διαφάνειες Μαθήματος Όραση Υπολογιστών, καθηγητή Ιωάννη Πρατικάκη

2. https://en.wikipedia.org/wiki/Panoramic_photography

3.https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html