



'Οραση Υπολογιστών

Εργασία 2

Ακαδημαϊκό Έτος: 2019-2020

Καρασακαλίδης Αλέξανδρος Ηλίας

ΑΗΜ: 57448

- Ζητούμενα

Στην εκφώνηση της εργασίας μας ζητείται να φτιάξουμε το πανόραμα τεσσάρων εικόνων από το παρακάτω σύνολο:



Συγκεκριμένα, οι απαιτήσεις της εργασίας είναι οι εξής:

1. Η κατασκευή του πανοράματος που προέρχεται από τη σύνθεση τουλάχιστον τεσσάρων εικόνων χρησιμοποιώντας τους παρακάτω ανιχνευτές και περιγραφείς:
 - a. SIFT
 - b. SURF
2. Προβολή των πανοραμάτων και σύγκριση μεταξύ τους αλλά και με το αντίστοιχο πανόραμα που θα παραχθεί με την χρήση του Image Composite Editor [Microsoft].
3. Εφαρμογή των παραπάνω σε 4 φωτογραφίες της επιλογής μας

Επίσης, μας δόθηκαν οι εξής υποδείξεις:

1. Πρέπει να συνταιριαστούν τα matching του εκάστοτε ζεύγους εικόνων με την μέθοδο cross checking χωρίς την χρήση της έτοιμης συνάρτησης που περιέχεται στην BFMatcher της OpenCV.
2. Να εμφανίζουμε σε κάθε συνταιριασμό εικόνων, τα ζευγάρια ομόλογων σημείων-κλειδιών που προέκυψαν.

- Ανάλυση του Κώδικα

Ξεκινάμε αποθηκεύοντας τους πίνακες των εικόνων σε ένα διάνυσμα Images:

```
Images = []

for i in range(0,9):
    Images.append(cv.imread("HWBimg/HWBimg/yard-0"+str(i)+".png"))
```

Η πρώτη συνάρτηση που γράφτηκε και χρησιμοποιήθηκε είναι η findBest4:

```
def findBest4(Images):

    mrkr=8
    BestScoreBy=8
    BestScore=0
    scores=[]

    while mrkr-3>=0:

        score=0

        tmpsift = cv.xfeatures2d_SIFT.create()

        kp1 = tmpsift.detect(Images[mrkr])
        desc1 = tmpsift.compute(Images[mrkr], kp1)

        kp2 = tmpsift.detect(Images[mrkr-1])
        desc2 = tmpsift.compute(Images[mrkr-1], kp2)

        kp3 = tmpsift.detect(Images[mrkr-2])
        desc3 = tmpsift.compute(Images[mrkr-2], kp3)

        kp4 = tmpsift.detect(Images[mrkr-3])
        desc4 = tmpsift.compute(Images[mrkr-3], kp4)

        bf = cv.BFMatcher(cv.NORM_L2, crossCheck=True)
        score = score + len(bf.match(desc1[1], desc2[1]))

        bf = cv.BFMatcher(cv.NORM_L2, crossCheck=True)
        score = score + len(bf.match(desc2[1], desc3[1]))

        bf = cv.BFMatcher(cv.NORM_L2, crossCheck=True)
        score = score + len(bf.match(desc3[1], desc4[1]))

        tmpsurf = cv.xfeatures2d_SURF.create()

        kp1 = tmpsurf.detect(Images[mrkr])
        desc1 = tmpsurf.compute(Images[mrkr], kp1)

        kp2 = tmpsurf.detect(Images[mrkr-1])
        desc2 = tmpsurf.compute(Images[mrkr-1], kp2)
```

```

kp3 = tempsurf.detect(Images[mrkr-2])
desc3 = tempsurf.compute(Images[mrkr-2], kp3)

kp4 = tempsurf.detect(Images[mrkr-3])
desc4 = tempsurf.compute(Images[mrkr-3], kp4)

bf = cv.BFMatcher(cv.NORM_L2, crossCheck=True)
score = score + len(bf.match(desc1[1], desc2[1]))

bf = cv.BFMatcher(cv.NORM_L2, crossCheck=True)
score = score + len(bf.match(desc2[1], desc3[1]))

bf = cv.BFMatcher(cv.NORM_L2, crossCheck=True)
score = score + len(bf.match(desc3[1], desc4[1]))

scores.append(score)

if score>BestScore:
    BestScore=score
    BestScoreBy=mrkr

mrkr=mrkr-1

print(scores)
print(BestScoreBy)
return BestScoreBy

```

Το όρισμα της συνάρτησης περιλαμβάνει απλά το διάνυσμα που περιέχει τους πίνακες των εικόνων.

Για την γρηγορότερη ανάλυση της εργασίας χρησιμοποιήθηκαν οι 4 εικόνες με 'το καλύτερο ταίριασμα'. Αυτό κρίθηκε με τον εξής τρόπο:

Αφού υπολογίστηκαν με την μέθοδο SIFT και SURF τα keypoints της κάθε εικόνας, κάναμε matching για κάθε δύο διαδοχικές εικόνες με την χρήση των keypoints και των δύο παραπάνω μεθόδων. Έπειτα, για κάθε διαδοχική τετράδα εικόνων, αθροίσαμε τον αριθμό των παραπάνω matches των επιμέρους εικόνων και το θεωρήσαμε ως το 'score' των matches. Κρατήσαμε την τετράδα με το μεγαλύτερο score (το οποίο ορίζεται από το BestScoreBy, το οποίο δείχνει το index του διανύσματος που περιέχει την τελευταία – πιο αριστερή – εικόνα της τετράδας που επιλέχτηκε).

Λόγω της μεγάλης υπολογιστικής πολυπλοκότητας, η συνάρτηση αυτή χρησιμοποιήθηκε μόνο μία φορά, και το αποτέλεσμά της χρησιμοποιήθηκε σαν δεδομένο κατά τις επόμενες δοκιμές του κώδικα.

Να σημειωθεί ότι στην συγκεκριμένη συνάρτηση – δεδομένης της ήδη μεγάλης υπολογιστικής πολυπλοκότητας αλλά και του γεγονότος ότι δεν αξιοποιείται στον τελικό αλγόριθμο – χρησιμοποιήθηκε για το matching το έτοιμο crossmatching της BFMatcher. Για κάθε άλλη διαδικασία όμως που είχε άμεση σχέση με την δημιουργία του πανοράματος, χρησιμοποιήθηκε

Αφού επιλέχτηκε η τετράδα των εικόνων που θα αποτελέσουν το πανόραμα, δημιουργήθηκε η παρακάτω συνάρτηση, για την συνένωση δύο εικόνων με την χρήση των keypoints τους που υπολογίστηκαν από την SIFT:

```
def siftAl(imgA,imgB,TranslateX):  
    sift = cv.xfeatures2d_SIFT.create()  
  
    kp1 = sift.detect(imgA)  
    desc1 = sift.compute(imgA, kp1)  
  
    kp2 = sift.detect(imgB)  
    desc2 = sift.compute(imgB, kp2)  
  
    matches = matcher(desc1[1], desc2[1])  
  
    img_pt1 = []  
    img_pt2 = []  
  
    for x in matches:  
        img_pt1.append(kp1[x.queryIdx].pt)  
        img_pt2.append(kp2[x.trainIdx].pt)  
  
    img_pt1 = np.array(img_pt1)  
    img_pt2 = np.array(img_pt2)  
  
    M, mask = cv.findHomography(img_pt2, img_pt1, cv.RANSAC)  
  
    M = np.matmul(np.array([[1,0,TransformX],[0,1,0],[0,0,1]]),M)  
  
    img3 = cv.warpPerspective(imgB, M, (imgA.shape[1] + imgB.shape[1],  
    imgA.shape[0] + 1000))  
  
    img3[0: imgA.shape[0], 0+TransformX: imgA.shape[1]+TransformX] = imgA  
  
    dimg = cv.drawMatches(imgA, desc1[0], imgB, desc2[0], matches, None)  
  
    return img3,dimg
```

Τα ορίσματα της συνάρτησης είναι οι δύο εικόνες που θέλουμε να ταιριάξουμε, και το TranslateX που θα εξηγηθεί παρακάτω.

Αφού υπολογιστούν τα keypoints της κάθε εικόνας, βρίσκουμε τα matches με την συνάρτηση matcher που έχουμε κατασκευάσει και της οποίας η (σχετικά απλή) λειτουργία παρουσιάζεται παρακάτω. Έπειτα, αντιστοιχίζοντας το κάθε keypoint που κάναμε match στις δύο εικόνες, υπολογίζουμε τον πίνακα μετασχηματισμού της 2ης εικόνας ώστε να ταιριάξει με την 1η (Homography).

Σε αντίθεση με το εργαστήριο όμως, ο πίνακας μετασχηματισμού, πολλαπλασιάζεται με έναν δεύτερο πίνακα μετασχηματισμού, ο οποίος είναι translation matrix (δηλαδή μετατοπίζει την εικόνα), και ο οποίος θα μετατοπίσει την εικόνα προς τα δεξιά κατά TranslateX (το 3ο όρισμα της συνάρτησης!).

Translation Matrix:

$$\begin{matrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{matrix}$$

Our Translation Matrix (Tx -> TranslateX):

$$\begin{matrix} 1 & 0 & Tx \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$$

Ο παραπάνω πίνακας λοιπόν πολλαπλασιάζεται με τον πίνακα μετασχηματισμού που μας έδωσε η συνάρτηση της OpenCV findHomography και ο τελικός πίνακας είναι αυτός που χρησιμοποιείται για τον μετασχηματισμό της 2ης εικόνας.

Όπως είναι γνωστό από την θεωρία, η σειρά με την οποία θα πραγματοποιηθεί ο μετασχηματισμός, είναι 'ανάποδη' από την σειρά με την οποία πολλαπλασιάζονται οι επιμέρους πίνακες μετασχηματισμού!

'Ετσι, για να πετύχουμε την διαδικασία 'Μετασχηματισμός με βάση την findHomography -> Μετακίνηση της εικόνας στον άξονα X κατά TranslateX', πρέπει να πολλαπλασιάσουμε τον πίνακα Translate με τον πίνακα του findHomography και όχι το αντίθετο!

Γιατί όμως θέλουμε να προσθέσουμε το TranslateX στον μετασχηματισμό της εικόνας;

Ας εξετάσουμε το αποτέλεσμα του συνταιριασμού της εικόνας Image[6] με την Image[5] και της Image[5] με την Image[6] (Θεωρώντας δηλαδή σαν imgA και imgB τις δύο εικόνες αντίστοιχα):



Στην δεύτερη περίπτωση, βλέπουμε ότι ο μετασχηματισμός είναι αναποτελεσματικός, αφού η αριστερή εικόνα (που μετασχηματίζεται) καταλήγει έξω από τον χώρο της εικόνας!

Για αυτόν τον λόγο και μετατοπίζουμε την μετασχηματιζόμενη εικόνα (όταν είναι η αριστερή) προς τα δεξιά, ώστε να μην ‘χάνεται’ έξω από τον χώρο της τελικής εικόνας.

Για να είμαστε σίγουροι ότι δεν θα ξεφύγει η δεξιά εικόνα, το δεύτερο άκρο, προεκτείνουμε την τελική εικόνα κατά το διπλάσιο της αρχικής!

Τέλος, αντιστοιχίζουμε pixel by pixel κάθε στοιχείο της πρώτης εικόνας στην τελική, λαμβάνοντας υπόψη το translation που υπέστη η προηγούμενη εικόνα! Έτσι, παίρνουμε το εξής αποτέλεσμα:



Στο τέλος της συνάρτησης επιστρέφουμε, πέρα από την τελική εικόνα και την εικόνα που δείχνει τα matching keypoints (την οποία φτιάξαμε από την cv.drawMatches).

Η συνάρτηση **matcher** που χρησιμοποιήσαμε παραπάνω είναι η εξής:

```
def matcher(desc1, desc2):
    bf = cv.BFMatcher(cv.NORM_L2, crossCheck=False)
    matchesa = bf.match(desc1, desc2)
    matchesb = bf.match(desc2, desc1)
    matches = []

    for i in range(0, len(matchesa) - 1):
        for j in range(0, len(matchesb) - 1):
            if matchesa[i].distance == matchesb[j].distance:
                matches.append(matchesa[i])
                break
    return matches
```

Όπου εφαρμόζουμε κυριολεκτικά την μέθοδο ‘cross checking’ με 2 απλές επαναληπτικές δομές.

Με την ίδια διαδικασία συνταιριάζουμε τις εικόνες, υπολογίζοντας τα keypoints τους όμως με την SURF:

```
def surfAl(imgA,imgB,TransformX):
    surf = cv.xfeatures2d_SURF.create()

    kp1 = surf.detect(imgA)
    desc1 = surf.compute(imgA, kp1)

    kp2 = surf.detect(imgB)
    desc2 = surf.compute(imgB, kp2)

    matches = matcher(desc1[1], desc2[1])

    img_pt1 = []
    img_pt2 = []

    for x in matches:
        img_pt1.append(kp1[x.queryIdx].pt)
        img_pt2.append(kp2[x.trainIdx].pt)

    img_pt1 = np.array(img_pt1)
    img_pt2 = np.array(img_pt2)

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

    M = np.matmul(np.array([[1,0,TransformX],[0,1,0],[0,0,1]]),M)

    img3 = cv.warpPerspective(imgB, M, (imgA.shape[1] + imgB.shape[1],
    imgA.shape[0] + 1000))

    img3[0: imgA.shape[0], 0+TransformX: imgA.shape[1]+TransformX] = imgA

    dimg = cv.drawMatches(imgA, desc1[0], imgB, desc2[0], matches, None)

    return img3,dimg
```

Με την προηγούμενη διαδικασία όμως, προκύπτει ένα μεγάλο πρόβλημα. Κάθε φορά μου συνενώνουμε 2 εικόνες, η τελική καταλήγει να είναι πολύ μεγαλύτερη από τις δύο αρχικές λόγω των επεκτάσεών της που εφαρμόζουμε για να σιγουρευτούμε ότι δεν θα ξεφύγουν από τα όριά της οι μετασχηματισμένες πρώτες εικόνες. Καταλήγουμε δηλαδή να έχουμε πολύ μεγάλα κενά (μαύρα) κομμάτια γύρω από τις εικόνες μας.

Για να λυθεί αυτό το πρόβλημα, έχει δημιουργηθεί η συνάρτηση cleanImg:

```
def cleanImg(img):

    while True:
        if (np.sum(img[:,0,0])!=0) and (np.sum(img[:,0,1])!=0) and
(np.sum(img[:,0,2])!=0):
            break
        img = np.delete(img,0,1)

    while True:
        if (np.sum(img[:,img.shape[1]-1,0])!=0) and
(np.sum(img[:,img.shape[1]-1,0])!=0) and (np.sum(img[:,img.shape[1]-
1,0])!=0):
            break
        img = np.delete(img,img.shape[1]-1,1)

    while True:
        if (np.sum(img[img.shape[0]-1,:,:0])!=0) and
(np.sum(img[img.shape[0]-1,:,:1])!=0) and (np.sum(img[img.shape[0]-
1,:,:2])!=0):
            break
        img = np.delete(img,img.shape[0]-1,0)

    return img
```

Η οποία παίρνει σαν όρισμα μια εικόνα και διαγράφει στήλη-στήλη (στις πρώτες δύο while) και γραμμή – γραμμή (στην τελευταία while) κάθε στήλη/γραμμή που περιέχει μόνο μαύρα εικονοστοιχεία (δεν είναι μέρος των δύο συνενωμένων εικόνων δηλαδή).

Έτσι, ‘καθαρίζουν’ οι εικόνες από τις μαύρες περιοχές που δημιουργούνται από την προηγούμενη διαδικασία και μικραίνουν σε μέγεθος!

Τέλος, ορίζουμε τις συναρτήσεις MakePanoramaSIFT και MakePanoramaSURF, ώστε να φτιάχνουμε διαδοχικά με τις εικόνες -με την σειρά που ορίζουμε εμείς- το πανόραμα της τετράδας (Με keypoints που υπολογίστηκαν από την SIFT και την SURF αντίστοιχα):

```

def MakePanoramaSIFT(Images,i):
    starttime = time.time()
    A,mtchA = siftAl(Images[i-1],Images[i],Images[i-1].shape[1])
    A = cleanImg(A)
    print("1/3 Done [",round(time.time() - starttime),"sec]")

    starttime = time.time()
    B,mtchB = siftAl(Images[i-2],Images[i-3],0)
    B = cleanImg(B)
    print("2/3 Done [",round(time.time() - starttime),"sec]")

    starttime = time.time()
    C,mtchC = siftAl(A,B,0)
    C = cleanImg(C)
    print("3/3 Done [",round(time.time() - starttime),"sec]")

    return A,B,C,mtchA,mtchB,mtchC

def MakePanoramaSURF(Images,i):

    starttime = time.time()
    A,mtchA = surfAl(Images[i-1],Images[i],Images[i-1].shape[1])
    A = cleanImg(A)
    print("1/3 Done [",round(time.time() - starttime),"sec]")

    starttime = time.time()
    B,mtchB = surfAl(Images[i-2],Images[i-3],0)
    B = cleanImg(B)
    print("2/3 Done [",round(time.time() - starttime),"sec]")

    starttime = time.time()
    C,mtchC = surfAl(A,B,0)
    C = cleanImg(C)
    print("3/3 Done [",round(time.time() - starttime),"sec]")

    return A,B,C,mtchA,mtchB,mtchC

```

Ως ορίσματα, δέχονται το διάνυσμα που περιέχει τους πίνακες των εικόνων, και τον δείκτη της 4^{ης} εικόνας που θα περιέχεται στην ένωση (και έτσι ξέρουμε και παίρνουμε τις 3 προηγούμενες διαδοχικά εικόνες).

Το τελικό πανόραμα προέρχεται από την ένωση των 2 'αριστερών' εικόνων μεταξύ τους, των 2 'δεξιών' αντίστοιχα και τέλος, την ένωση των 2 προηγούμενων ενώσεων.

Για να πετύχουμε να έχουμε ως οπτική γωνία από το 'κέντρο' του πανοράματος, οι 2 'αριστερά' εικόνες ενώθηκαν με κεντρική εικόνα την 'δεξιότερη' από τις δύο!

Η συνάρτηση μας επιστρέφει την κάθε ένωση που πραγματοποιήσαμε, όπως και την αντιστοίχιση των matching keypoints τους (απεικονισμένα)!

```

ti=8

print("Calculating matches and transforming images accordingly using SIFT:")
finalSIFT = MakePanoramaSIFT(Images,ti)

print("Calculating matches and transforming images accordingly using SURF:")
finalSURF = MakePanoramaSURF(Images,ti)

final = [finalSIFT,finalSURF]

pntr=0

for name in ["SIFT", "SURF"]:
    for i in range(0,3):
        cv.namedWindow("[ " + name + " ] Matches of Images " + str(ti-i) + " , " + str(ti-i-1),cv.WINDOW_NORMAL)
        cv.imshow("[ " + name + " ] Matches of Images " + str(ti-i) + " , " + str(ti-i-1), final[pntr][i+3])

        cv.namedWindow("[ " + name + " ] Panorama " + str(i+1),cv.WINDOW_NORMAL)
        cv.imshow("[ " + name + " ] Panorama " + str(i+1), final[pntr][i])

        cv.waitKey(0)

        cv.destroyWindow("[ " + name + " ] Matches of Images " + str(ti-i) + " , " + str(ti-i-1))
        cv.destroyWindow("[ " + name + " ] Panorama " + str(i+1))
    pntr=pntr+1

ICEpanorama = cv.imread("HWBimgs/ICEpanorama.jpg")

cv.namedWindow("SIFT Panorama",cv.WINDOW_NORMAL)
cv.namedWindow("SURF Panorama",cv.WINDOW_NORMAL)
cv.namedWindow("ICE Panorama",cv.WINDOW_NORMAL)
cv.imshow("SIFT Panorama",final[0][2])
cv.imshow("SURF Panorama",final[1][2])
cv.imshow("ICE Panorama",ICEpanorama)
cv.waitKey(0)

```

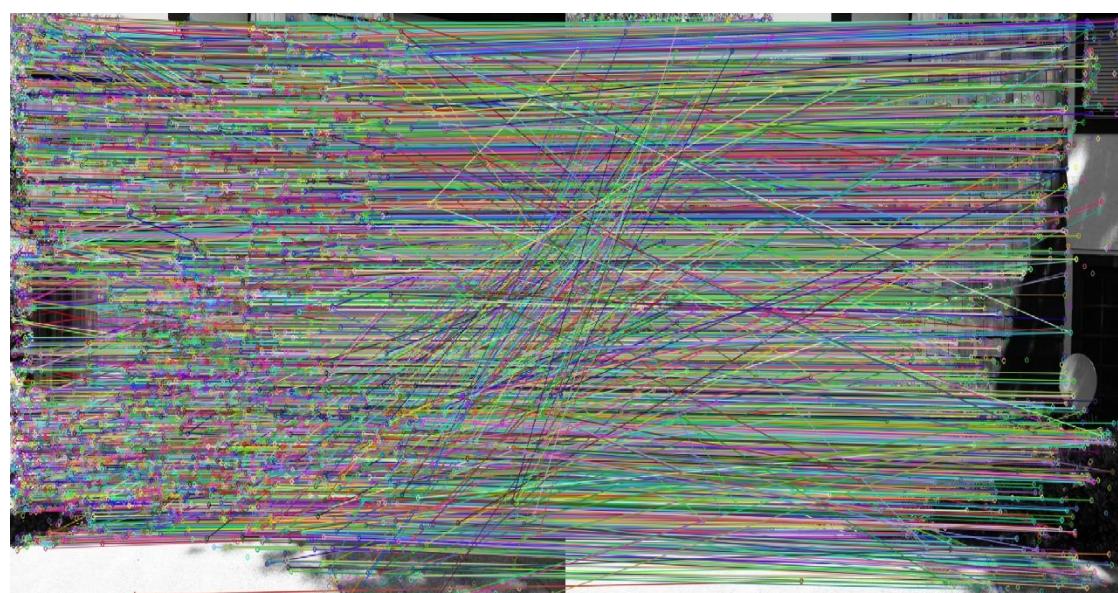
Με μια επαναληπτική διαδικασία στο τέλος, και καλώντας τις συναρτήσεις (Επιλέγοντας την τετράδα που έχει ως αριστερή εικόνα της τετράδας την `Image[8]` – όπως επιλέξαμε με βάση την πρώτη συνάρτηση που δείξαμε στο `report`), εμφανίζουμε κάθε συνένωση που πραγματοποιήθηκε με βάση τις παραπάνω διαδικασίες (και με τις δύο μεθόδους `upoloigismón keypoints`).

Τα τελικά αποτελέσματα των matching keypoints των επιμέρους εικόνων, των τελικών – συνταιριασμένων εικόνων, αλλά και το αντίστοιχο πανόραμα που σχηματίστηκε με το πρόγραμμα `Image Composite Editor` είναι τα παρακάτω:

Images:

Image[8] & Image[7]

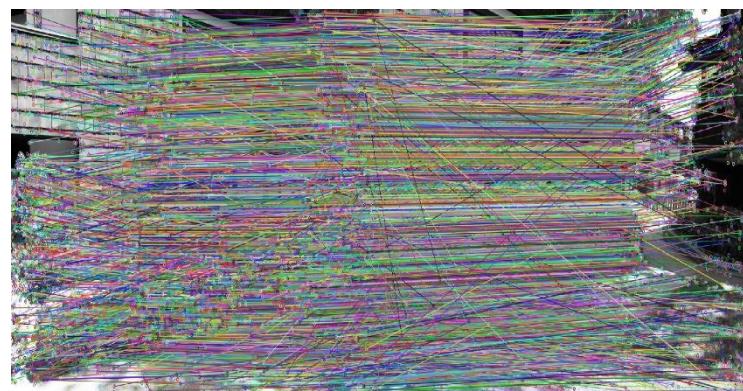
[SIFT keypoints]



Images:

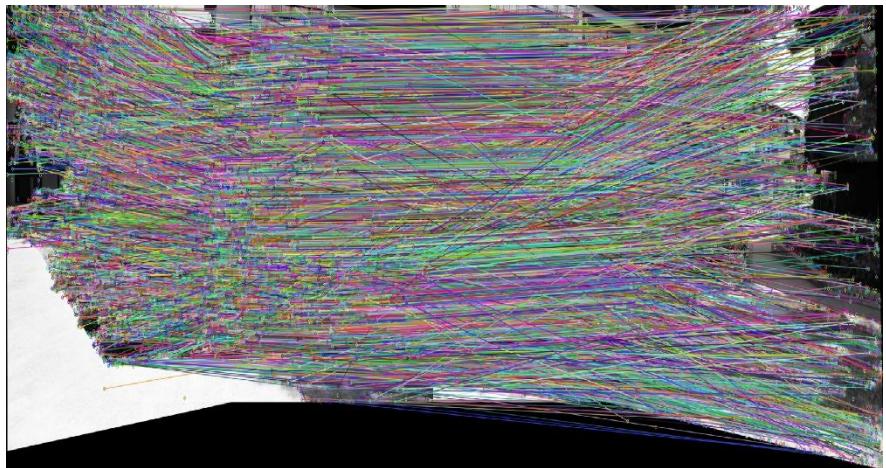
Image[5] & Image[6]

[SIFT keypoint]



Images:

Η ένωση των παραπάνω 2 ενώσεων.
[SIFT keypoints]



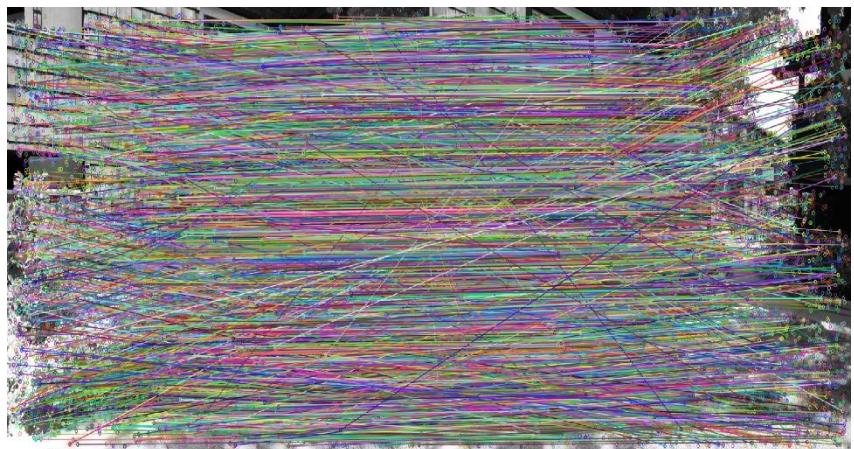
Images:

Image[8] & Image[7]
[SURF keypoints]



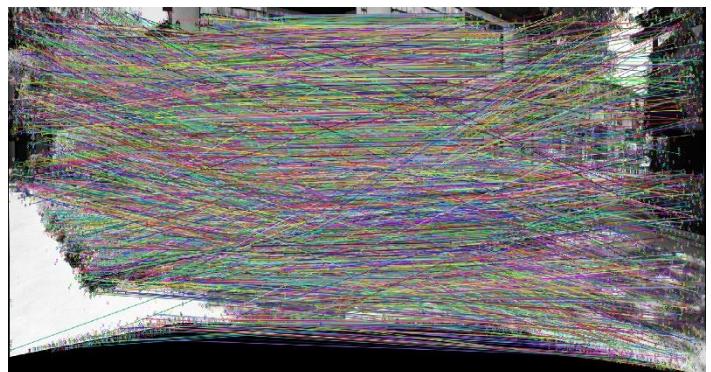
Images:

Image[5] & Image[6]
[SURF keypoints]



Images:

Η ένωση των παρα-
πάνω 2 ενώσεων.
[SURF keypoints]



Τελική σύγκριση των πανοραμάτων με SIFT keypoints, SURF keypoints και από το πρόγραμμα Image Composite Editor αντίστοιχα:



Η εμφανέστερη διαφορά μεταξύ των εικόνων παρατηρείται στον φωτισμό. Δηλαδή, οι εικόνες που παράχθηκαν από τον αλγόριθμο, δεν αντιμετωπίζουν την εμφανή διαφορά φωτισμού ανάμεσα στις ενωμένες εικόνες. Αυτός είναι και ο λόγος που είναι τόσο εμφανής η ένωση (στις περισσότερες εννώσεις).

Μεταξύ των δύο διαφορετικών πανοραμάτων που παράξαμε, δεν παρατηρείται σχεδόν καμία διαφορά, πέρα από την ελάχιστα καλύτερη ένωση που φαίνεται να έχει συμβεί με τα keypoints που παράχθηκαν από την μέθοδο SURF (Ένωση εικόνων Image[5], Image[6]).

Παρακάτω, συγκρίνουμε με την ίδια διαδικασία με τις προηγούμενες εικόνες, 4 εικόνες προσωπικά επιλεγμένες:

```
ti=3

print("For my Images:")

print("Calculating matches and transforming images accordingly using SIFT:")
myImagesSIFT = MakePanoramaSIFT(myImages,ti)

print("Calculating matches and transforming images accordingly using SURF:")
myImagesSURF = MakePanoramaSURF(myImages,ti)

myfinal = [myImagesSIFT,myImagesSURF]

pntr=0

for name in ["SIFT", "SURF"]:
    for i in range(0,3):
        cv.namedWindow("[ " + name + " ] Matches of my Images " + str(ti-i) +
" , " + str(ti-i-1),cv.WINDOW_NORMAL)
        cv.imshow("[ " + name + " ] Matches of my Images " + str(ti-i) + " ,
" + str(ti-i-1), myfinal[pntr][i+3])

        cv.namedWindow("[ " + name + " ] My panorama "+str(i+1),cv.WINDOW_NORMAL)
        cv.imshow("[ " + name + " ] My panorama "+ str(i+1),
myfinal[pntr][i])

        cv.waitKey(0)

        cv.destroyWindow("[ " + name + " ] Matches of my Images " + str(ti-i) +
" , " + str(ti-i-1))
        cv.destroyWindow("[ " + name + " ] My panorama "+ str(i+1))
    pntr=pntr+1

myICEpanorama = cv.imread("HWBimgs/myICEpanorama.jpg")

cv.namedWindow("SIFT Panorama",cv.WINDOW_NORMAL)
cv.namedWindow("SURF Panorama",cv.WINDOW_NORMAL)
cv.namedWindow("ICE Panorama",cv.WINDOW_NORMAL)
cv.imshow("SIFT Panorama",myfinal[0][2])
cv.imshow("SURF Panorama",myfinal[1][2])
cv.imshow("ICE Panorama",myICEpanorama)
cv.waitKey(0)
```

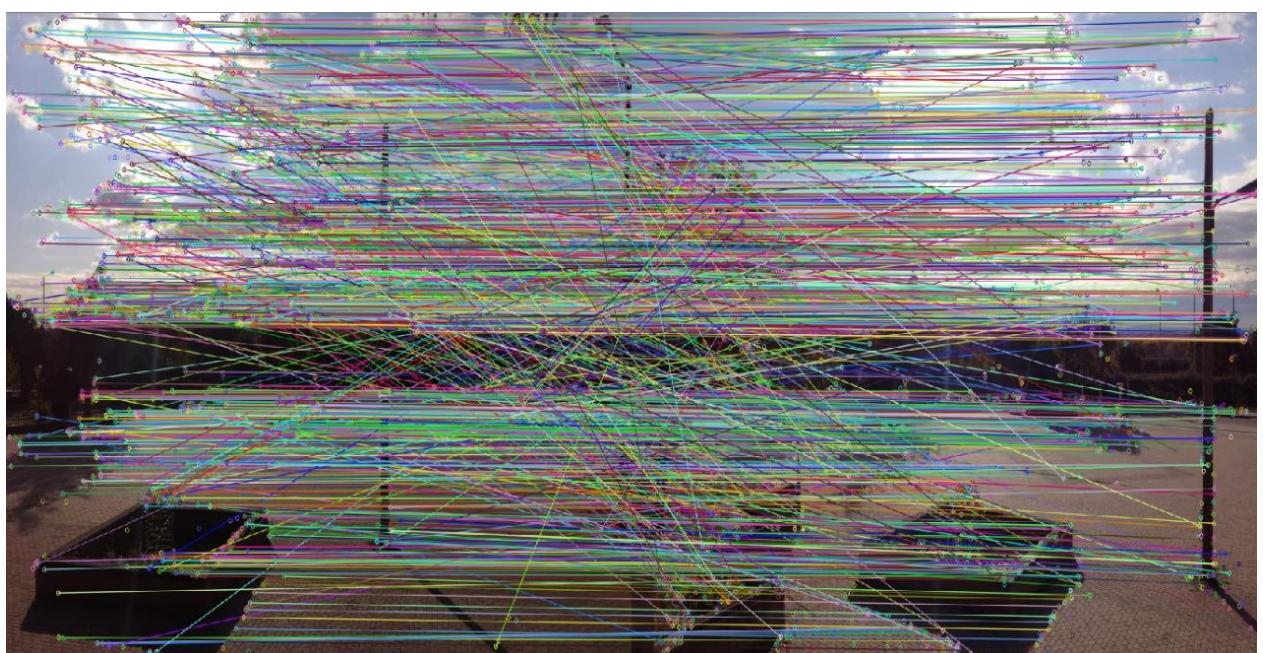
Και οι εικόνες:



Το αποτέλεσμα του πανοράματός τους (με την ίδια πάντα διαδικασία) είναι:

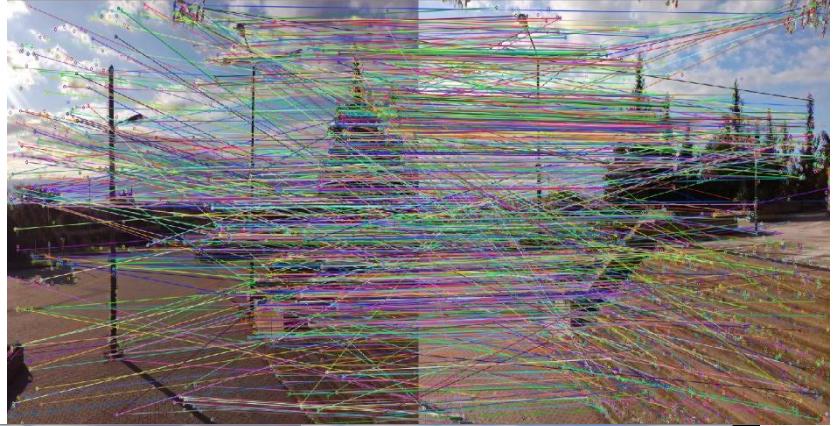
Images:

myImages[3] & myImages[2]
[SIFT keypoints]



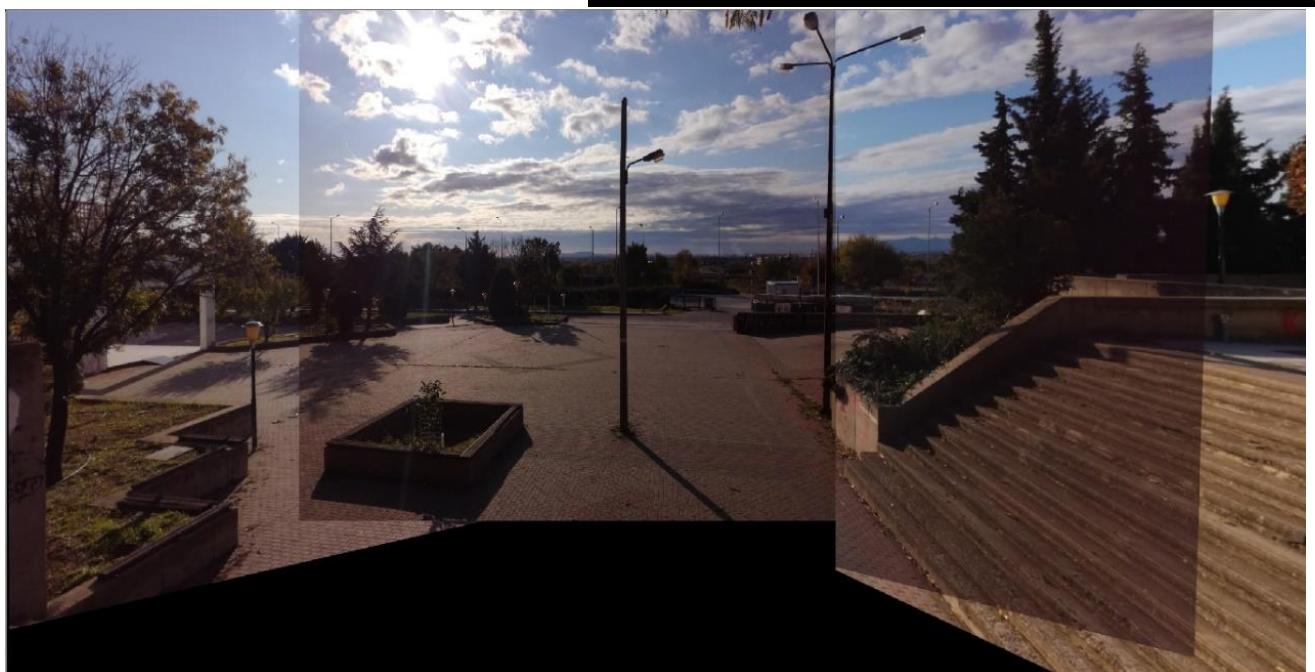
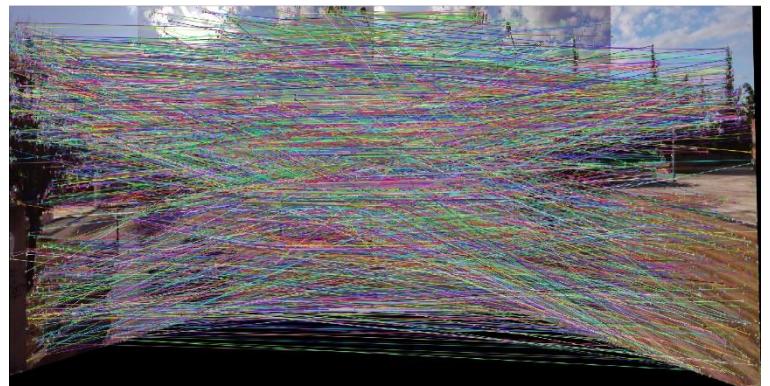
Images:

myImages[0] & myImages[1]
[SIFT keypoints]



Images:

Η ένωση των παρα-
πάνω 2 ενώσεων.
[SIFT keypoints]



Images:

myImages[3] & myImages[2]
[SURF keypoints]



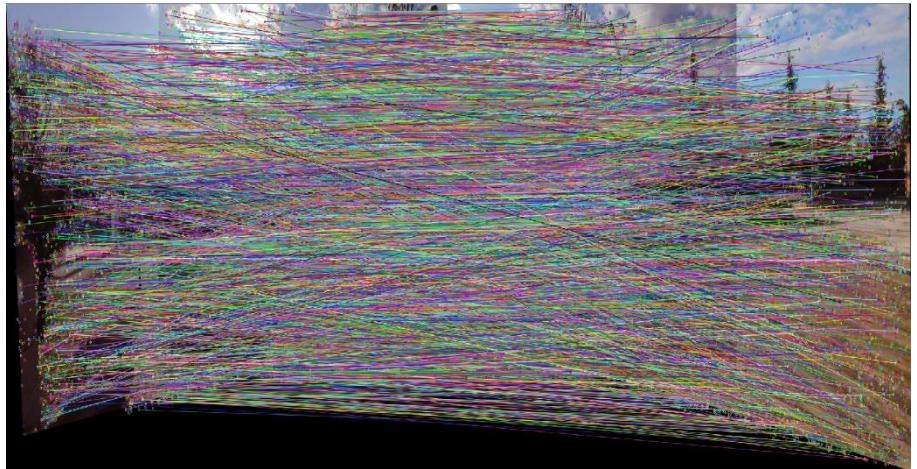
Images:

myImages[0] & myImages[1]
[SURF keypoints]

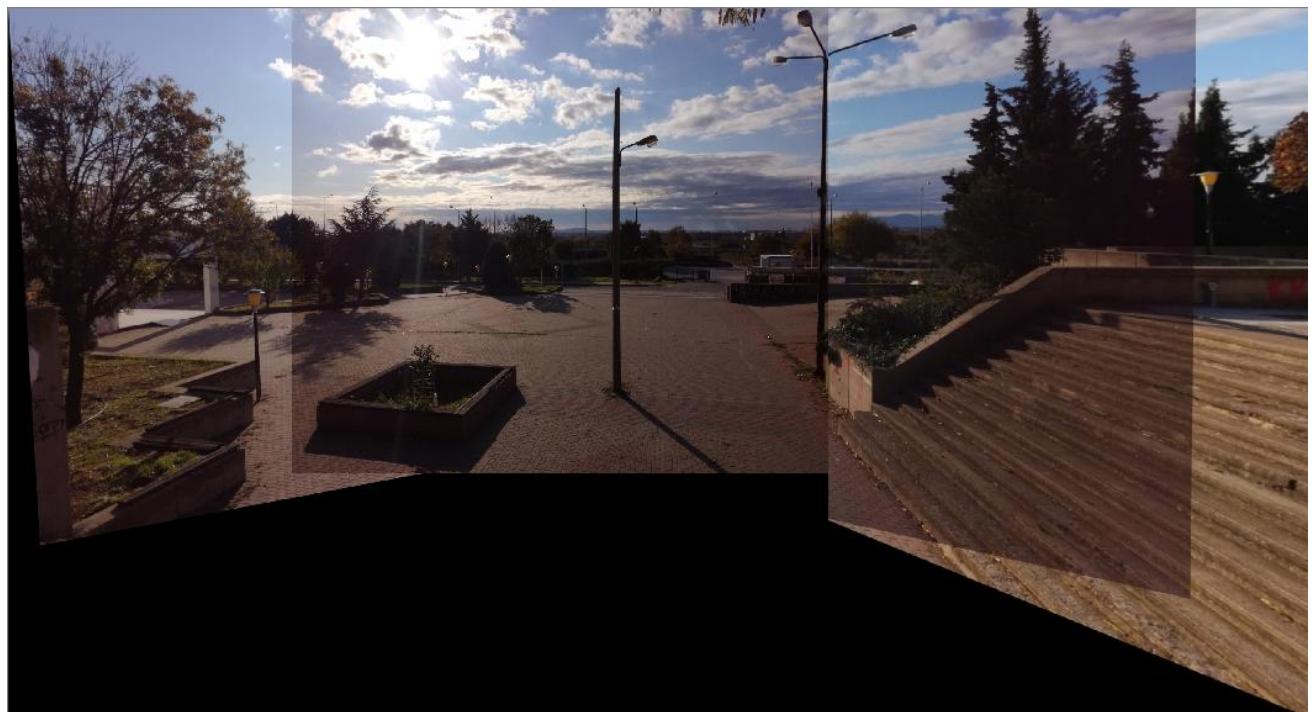
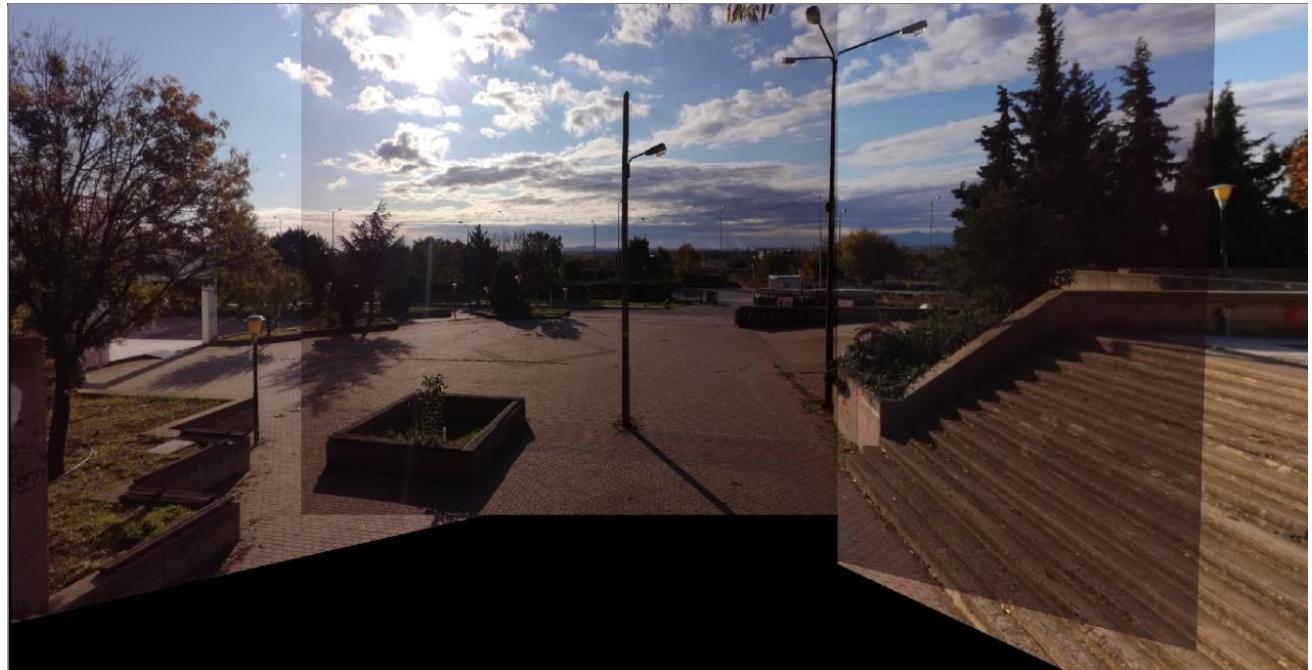


Images:

Η ένωση των παραπάνω 2 ενώσεων.
[SURF keypoints]



Τελική σύγκριση των πανοραμάτων με SIFT keypoints, SURF keypoints και από το πρόγραμμα Image Composite Editor αντίστοιχα:





Σε αυτές τις εικόνες, η μεγάλη διαφορά που προκαλεί η εναλλαγή της φωτεινότητας από εικόνα σε εικόνα είναι πολύ πιο εμφανής. Βέβαια, η αντιστοίχιση είναι πολύ ακριβής και στις δύο παραγόμενες εικόνες, χωρίς να παρατηρείται κάποια παραμόρφωση / λανθασμένη αντιστοίχιση.

Παρατηρούμε βέβαια μικρές διαφορές ανάμεσα στις δύο παραγόμενες εικόνες, και επαληθεύουμε ότι οι μεταβάσεις από εικόνα σε εικόνα στο πανόραμα φαίνονται πιο ακριβείς με keypoints της SURF.

Περαιτέρω βελτιώσεις του αλγορίθμου μπορεί να περιλαμβάνουν:

- Μεγαλύτερο πανόραμα, με περισσότερες εικόνες.
- Εξίσωση των επιπέδων φωτεινότητας των επιμέρους εικόνων.

Ευχαριστώ για την προσοχή σας!

Καρασακαλίδης Αλέξανδρος Ηλίας

1/12/2019