Μαρία Αρετή

Γερμανού 57807

7ο Εξάμηνο 2022

Όραση Υπολογιστών

Τεχνική Αναφορά - Εργασία 2η

Η τεχνική αναφορά περιλαμβάνει:

I. Την θεωρητική ανάλυση της εργασίας.

II. Την περιγραφή του κώδικα που παραδόθηκε.

III. Την ανάλυση των αποτελεσμάτων.

Ι. Θεωρητική ανάλυση της εργασίας.

Σύμφωνα με την εκφώνηση της δεύτερης εργασίας, θα πρέπει να παραχθεί το πανόραμα που προέρχεται από τη σύνθεση τουλάχιστον τεσσάρων εικόνων χρησιμοποιώντας τους παρακάτω ανιχνευτές και περιγραφείς. Αναλυτικά θα παρουσιαστεί κάθε βήμα στην περιγραφή του κώδικα που θα γίνει παρακάτω και μαζί θα αναλυθούν και τα αποτελέσματα. Προς το παρόν ας παρουσιαστεί βηματικά η θεωρητική ανάλυση και μεθοδολογία που θα ακολουθηθεί παρακάτω.

Images used for building a Panorama

// Ανίχνευση Τοπικών Χαρακτηριστικών:

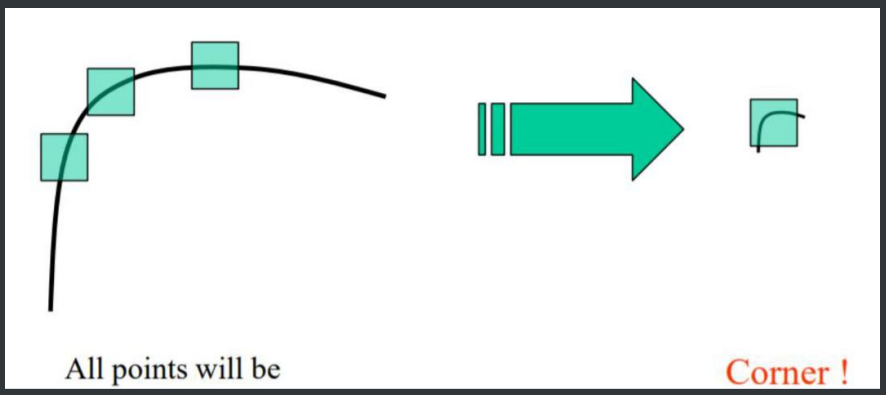
Αρχικά θα πρέπει να διαμορφωθούν οι εικόνες κατάλληλα ώστε να ενωθούν σωστά για να συναρμολογηθεί το πανόραμα. Για κάθε εικόνα που θα διαμορφωθεί θα εξάγουμε τοπικά χαρακτηριστικά και τους περιγραφείς τους. Έπειτα θα γίνει σύγκριση των χαρακτηριστικών για να ενωθούν κατάλληλα οι εικόνες αντιστοιχίζοντας τα ίδια χαρακτηριστικά. Ο εντοπισμός των χαρακτηριστικών σε εικόνες είναι η διαδικασία κατά την οποία λαμβάνεται η πληροφορία και ελέγχεται για το αν ταιριάζει με κάποιον συγκεκριμένο τύπο που ορίζεται στο ίδιο σημείο. Ένα σύνολο μετασχηματισμών εικόνων όπως περιστροφή, μεγέθυνση, σμίκρυνση, αποθορυβοποίηση, θόλωμα, θα ήταν μια ιδανική τεχνική για τον εντοπισμό και αντιστοίχιση χαρακτηριστικών σε εικόνες. Στην συγκεκριμένη

Ι. Θεωρητική ανάλυση της εργασίας.

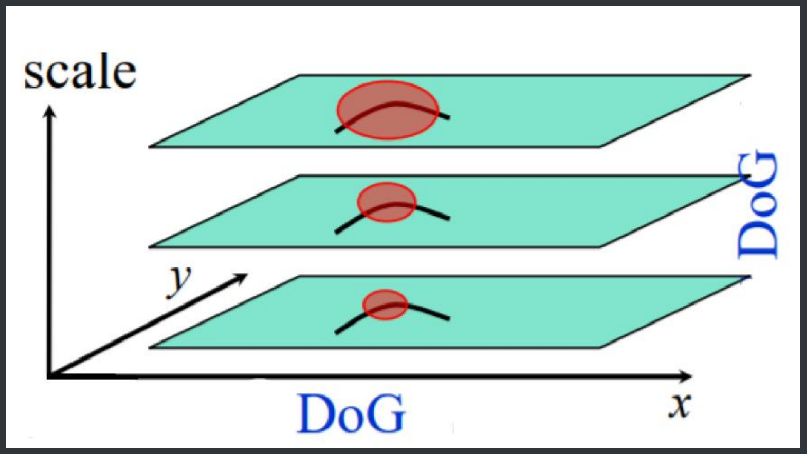
άσκηση θα χρησιμοποιηθούν δύο τέτοιες τεχνικές και θα αναλυθούν πρώτα θεωρητικά και έπειτα τεχνικά. Παρακάτω αναλύεται η διαδικασία ανίχνευσης, δηλαδή αναγνώρισης σημείου ενδιαφέροντος και περιγραφής τοπικών χαρακτηριστικών, δηλαδή ένα διάνυσμα που μας δίνει πληροφορίες για τα σημεία αυτά.

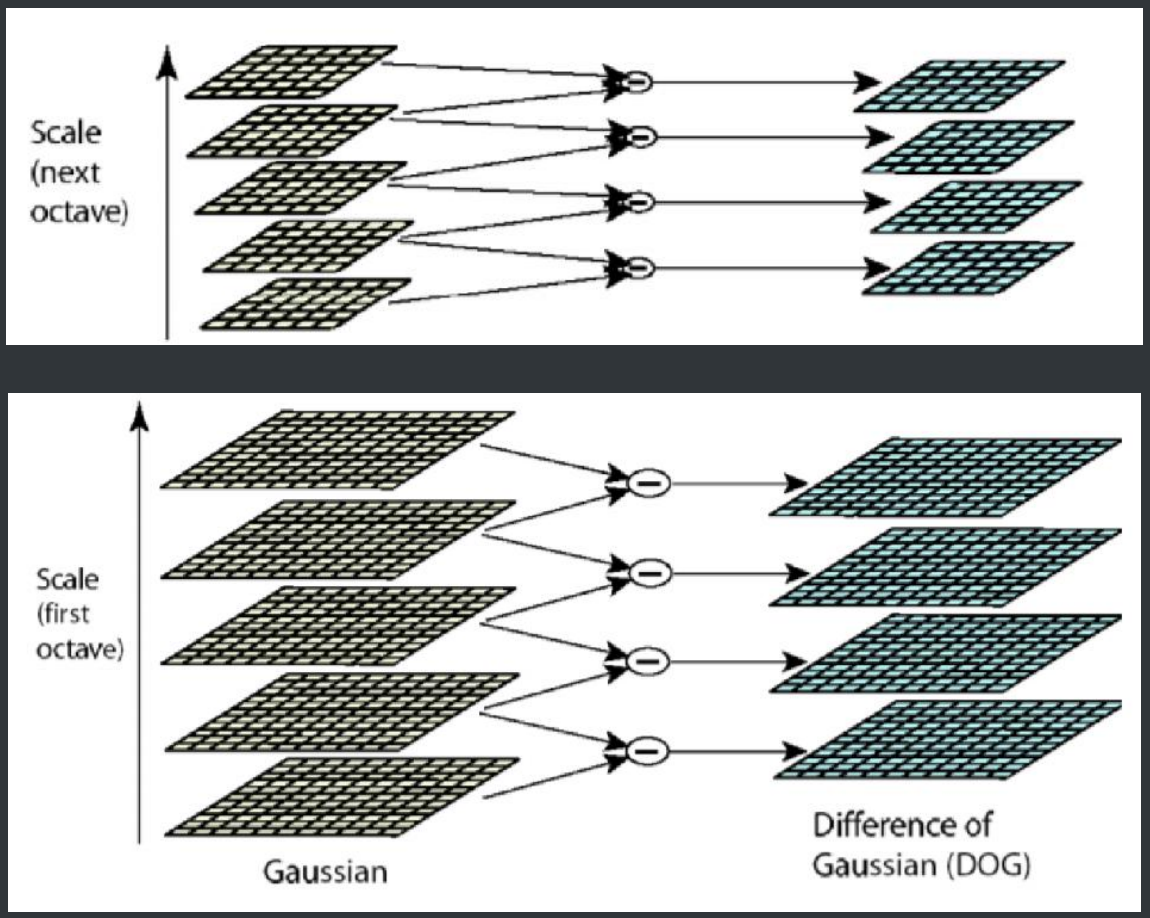
// Ανάλυση Αλγορίθμου "SIFT" - "Scale Invariant Feature Transform"

Ο αλγόριθμος "SIFT" ενώ είναι πολύ χρήσιμος στην αναγνώριση εικόνων, απαιτεί μεγάλη πολυπλοκότητα για την εκπόνησή του. Σε αυτό το μειονέκτημα έχουν γίνει αρκετές βελτιώσεις στον αλγόριθμο μέσα στα χρόνια μειώνοντας ελάχιστα την πολυπλοκότητά του. Ωστόσο έχει το πλεονέκτημα να αναγνωρίζει χαρακτηριστικά ανεξαρτήτως κλίμακας.



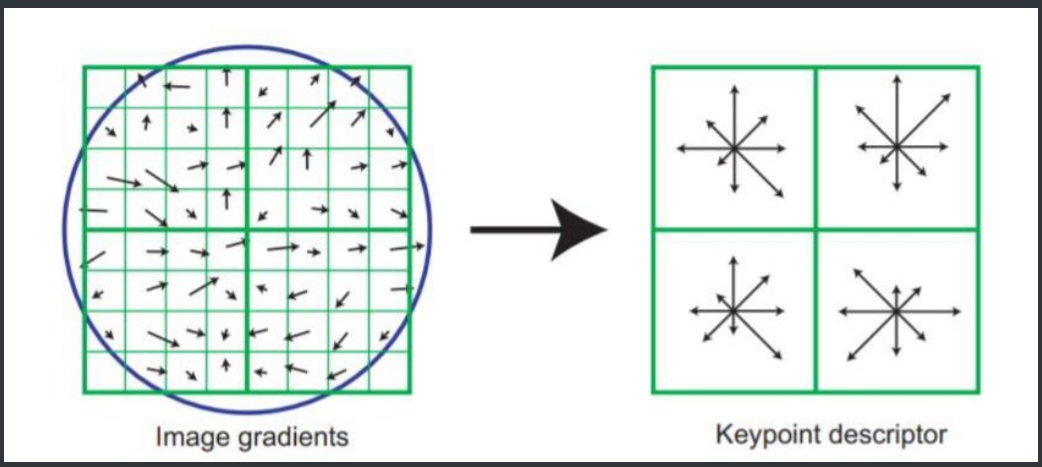
Ο αλγόριθμος εφαρμόζει περιστροφές, μετασχηματισμούς για να αντιστοιχίσει τα χαρακτηριστικά της εικόνας. Τα βήματα που ακολουθούνται είναι τα εξής. Αρχικά υπολογίζεται η διαφορά μεγίστων ελαχίστων τοπικών σημείων "Difference of Gaussian (DoG)".





Ανίχνευση σημείων ανεξαρτήτους κλίμακας

Έπειτα δημιουργείται ο περιγραφέας αυτών των τοπικών χαρακτηριστικών ή αλλιώς "keypoints", εντοπίζοντας αυτά τα σημεία, υπολογίζοντας την κατεύθυνσή τους και αποκλείοντας εκείνα με χαμηλή αντίθεση. Έτσι προσδιορίζονται τα "keypoints" τα οποία συμφωνούν στο αντικείμενο, στην θέση, στην κλίμακα και στον προσανατολισμό τους με την νέα εικόνα και με αυτόν τον τρόπο επιτυγχάνεται η αναγνώριση αντικειμένων σε μια νέα εικόνα.



Δημιουργία περιγραφέα των τοπικών χαρακτηριστικών

Ι. Θεωρητική ανάλυση της εργασίας.

// Ανάλυση Αλγορίθμου "SURF" - "Speeded up Robust Feature"

Ο SURF είναι ένας πιο γρήγορος αλγόριθμος από τον SIFT με την διαφορά πως χρησιμοποιεί έναν αλγόριθμο τύπου "Hessian blob" σε μια προ αναμειγμένη ολοκληρωμένη εικόνα για να βρει τα σημεία ενδιαφέροντος. Επίσης εντοπίζει τα DoG με box filters με μια διαφορετική διαδικασία και χρήση των τετραγώνων που διατρέχουν την ολοκληρωμένη εικόνα. Η εικόνα μετατρέπεται σε συντεταγμένες χρησιμοποιώντας την τεχνική πυραμίδων πολλών αναλύσεων για να αντιγράψει την αρχική εικόνα με σχήμα πυραμιδικού Gauss ή Laplacian Pyramid για να αποκτήσει μια εικόνα με το ίδιο μέγεθος αλλά με μειωμένο εύρος ζώνης. Ουσιαστικά ο πίνακας διατρέχει την εικόνα ψάχνοντας για τα σημεία ενδιαφέροντος, επιλέγοντας την γειτονιά γύρω από το "keypoint" και χωρίζοντάς την σε υποπεριοχές. Έπειτα για κάθε υποπεριοχή συλλέγονται και αναπαριστώνται τα στοιχεία για να ολοκληρωθεί η δημιουργία του περιγραφέα αυτών των τοπικών χαρακτηριστικών. Τεχνικά διαχωρίζονται τα λευκά στοιχεία σε μαύρο φόντο και συγκρίνονται ώστε έπειτα να αντιστοιχιστούν με άλλα στοιχεία ίδιας αντίθεσης, κάτι που επιτρέπει πιο γρήγορη αντιστοίχιση. Στην παρακάτω εικόνα φαίνεται η υλοποίηση των αλγορίθμων SIFT, SURF και τα ταιριάσματα των χαρακτηριστικών της εικόνας σε κανονικό και περιστρεφόμενο προσανατολισμό.

	Keypoints		Matches	Rate%
	box	box(180)		
SIFT	603	602	538	44.65
SURF	997	999	929	46.54
ORD	453	453	453	100

Box Image

	Keypoints		Matches	Rate%
	basmati	basmati(180)		
SIFT	489	491	452	46.12
SURF	607	600	559	46.31
ORD	383	383	383	100

Basmati Image

Ι. Θεωρητική ανάλυση της εργασίας.

Παρατηρώ το χαμηλό ποσοστό ταιριασμάτων των χαρακτηριστικών για αυτό χρησιμοποιείται μια τεχνική που δεν θα αναλυθεί σε αυτήν την εργασία, τον αλγόριθμο "ORB: Performance Comparison for Distorted Images". Είναι ταχύτερος και έχει 100% ποσοστό ταιριασμάτων.

// Υλοποίηση "Matching" με μέθοδο "Cross Checking":

Για να γίνει το συνταίριασμα των εικόνων πρέπει να αντιστοιχιστούν τα τοπικά χαρακτηριστικά της μιας εικόνας με μιας άλλης. Στην συγκεκριμένη διαδικασία για κάθε "keypoint" της μιας εικόνας συγκρίνουμε την απόσταση του περιγραφέα, με όλους τους περιγραφείς της άλλης εικόνας και τον αντιστοιχούμε με αυτόν με την μικρότερη απόσταση, συνήθως ονομάζεται απόσταση "Manhattan". Στις παρακάτω εικόνες φαίνεται η αντιστοίχιση των χαρακτηριστικών με τις δύο τεχνικές.

"Cross Checking" με την μέθοδο SIFT

"Cross Checking" με την μέθοδο SURF

Παρόλα αυτά δεν μπορούμε να ταιριάξουμε δύο εικόνες αν δεν προσανατολιστούν ανάλογα για να υπάρξει σωστό ταίριασμα. Συνήθως, όπως και στην συγκεκριμένη περίπτωση μια από τις δύο εικόνες πρέπει να προσανατολιστεί και η απλή μετακίνηση δεν είναι αρκετή. Σε αυτό συμβάλλει ο ομογραφία. Επαναλαμβάνοντας την διαδικασία συμπληρώνεται το πανόραμα.

// Περιγραφή Κώδικα: Φόρτωση εικόνων σε Path και ανάγνωσή τους με την OpenCV σε ασπρόμαυρη μορφή. Φορτώνω τις εικόνες με δύο διαφορετικούς τρόπους όπως θα φανεί στο αρχείο του κώδικα. Επειδή είναι λίγα αρχεία δεν μένω στην επαναληπτική διαδικασία και φορτώνω μια-μια τις εικόνες βάζοντας τες σε μια λίστα.

```
# Re-assigning the files and reading them colored and grayed:
filenames_1 = 'images_2/yard-house-01.png'
filenames_2 = 'images_2/yard-house-02.png'
filenames_3 = 'images_2/yard-house-03.png'
filenames_4 = 'images_2/yard-house-04.png'
filenames_5 = 'images_2/yard-house-05.png'

# Grayed images:
img_1 = cv2.imread(filenames_1, cv2.IMREAD_GRAYSCALE)
img_2 = cv2.imread(filenames_2, cv2.IMREAD_GRAYSCALE)
img_3 = cv2.imread(filenames_3, cv2.IMREAD_GRAYSCALE)
img_4 = cv2.imread(filenames_4, cv2.IMREAD_GRAYSCALE)
img_5 = cv2.imread(filenames_5, cv2.IMREAD_GRAYSCALE)

img = [img_1, img_2, img_3, img_4, img_5]
```

Σύμφωνα με την θεωρία για να εφαρμόσω τις τεχνικές που προαναφέρθηκαν πρέπει να κάνω αρχικά μια συνάρτηση για την αντιστοίχιση των χαρακτηριστικών των εικόνων που επιθυμώ να ενώσω για να συμπληρωθεί μετέπειτα το πανόραμα. Επίσης η εκφώνηση επειδή τα ταιριάσματα από την Εικόνα_1 προς την Εικόνα_2 κι έπειτα από την Εικόνα_2 προς την Εικόνα_1 είναι αμφίδρομα, συνιστάται η δημιουργία συνάρτησης χωρίς να χρησιμοποιηθεί η κλάση 'BFMatcher' της OpenCV.

```
def matching(d1, d2):
    n1 = d1.shape[0]
    n2 = d2.shape[0]
    matches = []

    for i in range(n1):
        fv = d1[i, :]
        diff = d2 - fv
        diff = np.abs(diff)
        distances = np.sum(diff, axis=1)
```

II. Περιγραφή του κώδικα.

```
        i2 = np.argmin(distances)
        minDistance2 = distances[i2]

        # infinitive
        distances[i2] = np.inf

        i3 = np.argmin(distances)
        minDistance3 = distances[i3]

        # "Good" Matching condition
        if minDistance2 / minDistance3 < 0.5:
            matches.append(cv2.DMatch(i, i2, minDistance2))

    return matches
```

Αρχικά δημιουργείται ένας detector ανάλογα με την τεχνική που θέλουμε να ακολουθήσουμε. Έπειτα για κάθε εικόνα υπολογίζω τα keypoints και τους περιγραφείς τους.

```
def descriptors_and_crosscheck(img1, img2, methodOfDescriptor):

    if methodOfDescriptor == "SIFT" or methodOfDescriptor == "sift":
        descriptor = cv2.xfeatures2d_SIFT.create()

    elif methodOfDescriptor == "SURF" or methodOfDescriptor == "surf":
        descriptor = cv2.xfeatures2d_SURF.create()

    # Detection of key-points for first image:
    keypoint1 = descriptor.detect(img1)

    # Computation of key-points for first image:
    desc1 = descriptor.compute(img1, keypoint1)

    # Detection of key-points for second image:
    keypoint2 = descriptor.detect(img2)

    # Computation of key-points for second image:
    desc2 = descriptor.compute(img2, keypoint2)

    # Crosschecking calling function matching with descriptors one and two as parameters.
    matches1 = matching(desc1[1], desc2[1])
    matches2 = matching(desc2[1], desc1[1])

    # Loop for creating a list with matches because we can't use the class.
    matches = [m for m in matches1 for n in matches2 if m.distance == n.distance]
```

Υπολογίζεται η ελάχιστη απόσταση των χαρακτηριστικών μιας εικόνας με μια άλλη και αποθηκεύω το index για να γίνει το "Cross Checking" σε μια λίστα. Στην

II. Περιγραφή του κώδικα.

συνέχεια υπολογίζω τους πίνακες ομογραφίας για να κάνω τις ενώσεις. Χρησιμοποιώ τον κώδικα του εργαστηρίου για να δημιουργήσω τους πίνακες της ομογραφίας.

```
    # We create two lists because homography needs two lists with points that the first is the match
    of the second. We
    # will use a different function for homography below. The code below is identical from lab examples
    just like
    # matching function. :-)

    img_pt1 = []
    img_pt2 = []

    for match in matches:

        img_pt1.append(keypoint1[match.queryIdx].pt)
        img_pt2.append(keypoint2[match.trainIdx].pt)

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

    # If we run the descriptors and crosscheck function above we will see that the key-points' connection
    is aligned
    # horizontally:

    dimg = cv2.drawMatches(img1, desc1[0], img2, desc2[0], matches, None)

    # cv2.imwrite("Crosschecking_result_sift.png", dimg)
    #
    # cv2.imwrite("Crosschecking_result_surf.png", dimg)

    # cv2.namedWindow('Crosschecking result:', cv2.WINDOW_AUTOSIZE)
    # cv2.imshow('Crosschecking result:', dimg)
    # cv2.waitKey(0)

    return img_pt1, img_pt2

    # Next we must find homography for stitching the images. The RANSAC algorithm is used to match the
    key-points. It is
    # not enough to find the minimum distance of key-points but to match patterns of neighbor key-points
    too. Homography
    # creates an transformed array because it finds the right rotation of an image to fit with another.
    Function for
    # homography:

    def homography(img1, img2, methodOfDescriptor):

        img_1pt, img_2pt = descriptors_and_crosscheck(img1, img2, methodOfDescriptor)

        # Calculate homography based on RANSAC algorithm
        M, mask = cv2.findHomography(img_2pt, img_1pt, cv2.RANSAC)

        return M
```

Στα σχόλια παραπάνω διακρίνονται τα αποθηκευμένα αρχεία με τις αντιστοιχίες των χαρακτηριστικών με τις δύο τεχνικές πριν προχωρήσουμε στην ομογραφία.

II. Περιγραφή του κώδικα.

Οι εικόνες με αυτά τα αποτελέσματα αναλύθηκαν θεωρητικά παραπάνω και με τις δύο μεθόδους. Αφού υπολογιστούν οι πίνακες της ομογραφίας στην παραπάνω συνάρτηση. Στην παρακάτω συνάρτηση κόβεται το κομμάτι της εικόνας που έχει την μαύρη αντίθεση. Θα προβληθεί παρακάτω η εικόνα πριν την αφαίρεση τους μαύρου φόντου.

```
def crop(image):

    # Convert the filtered grayscale image to a binary image with threshold
    th_val, img_th = cv2.threshold(image, 1, 255, cv2.THRESH_BINARY)

    # Find contours
    _, contours, _ = cv2.findContours(img_th, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cnt = contours[0]

    # Bound the space of the image
    (x, y, w, h) = cv2.boundingRect(cnt)

    # Crop the space
    croppedImage = image[y:y + h, x:x + w]

    return croppedImage
```

Στην συνέχεια γίνεται η ένωση των εικόνων για να δημιουργηθεί το πανόραμα με την συνάρτηση ένωσης των εικόνων.

III. Ανάλυση των αποτελεσμάτων.

```
    # Before the panorama is built we need to stitch the images one next to another. First we find the
    translated array
    # from homography function and due to method of descriptor and the direction of stitching we create
    a translated
    # matrix and multiply it with the array from the homography function. This will be done with list merging
    and the
    # final list will be the result. I calculate the translation before i stitch the homographed image
    next to original.
    # After the stitching i crop the rotated image because i want it to be aligned next to original.


def stitching(left, right, methodOfDescriptor):
    lim = max(left.shape) * 1.5
    translation = int(lim)

    M = homography(right, left, methodOfDescriptor)
    # Translation matrix
    translationMatrix = np.array([[1, 0, translation], [0, 1, 0], [0, 0, 1]])
    # Matrix Multiplication
    TMpM = np.matmul(translationMatrix, M)
    # Wraping transformation
    result = cv2.warpPerspective(left, TMpM, (translation * 2, translation * 2))
    result[0: right.shape[0], translation:translation + right.shape[1]] = right

    stitchedImage = crop(result)

    return stitchedImage

def panorama(images, methodOfDescriptor):
    # First stitching
    print("First stitching...\n")
    FirstStitching = stitching(images[0], images[1], methodOfDescriptor)

    # Second stitching
    print("Second stitching...\n")
    SecondStitching = stitching(FirstStitching, images[2], methodOfDescriptor)

    # Third stitching
    print("Third stitching..\n")
    ThirdStitching = stitching(SecondStitching, images[3], methodOfDescriptor)

    # Final stitching
    print("Final stitching..\n")
    Final_panorama = stitching(ThirdStitching, images[4], methodOfDescriptor)

    panoramaCropped = Final_panorama[:images[1].shape[0]]
    print("Paronama is completed\n")

    return Final_panorama, panoramaCropped

panorama_SIFT, panorama_SIFT_cropped = panorama(img, "SIFT")
# cv2.namedWindow('Panorama result with sift:', cv2.WINDOW_GUI_EXPANDED)
# cv2.imshow('Panorama result with sift:', panorama_SIFT_cropped)
# cv2.waitKey(0)

panorama_SURF, panorama_SURF_cropped = panorama(img, "SURF")
# cv2.namedWindow('Panorama result with surf:', cv2.WINDOW_GUI_EXPANDED)
# cv2.imshow('Panorama result with surf:', panorama_SURF_cropped)
# cv2.waitKey(0)

cv2.imwrite("panorama_SIFT.png", panorama_SIFT)
cv2.imwrite("panorama_SURF.png", panorama_SURF)
```

III. Ανάλυση των αποτελεσμάτων.

Η ανάλυση των αποτελεσμάτων έγινε και θεωρητικά και στην περιγραφή του κώδικα. Παρακάτω φαίνονται τα αποτελέσματα για τις δύο τεχνικές και με την χρήση του εργαλείου προβολής πανοραμάτων που δόθηκε.

Πανόραμα με την μέθοδο SIFT

Πανόραμα με την μέθοδο SURF

III. Ανάλυση των αποτελεσμάτων.

Πανόραμα με το εργαλείο Image Composite Editor.

Στο Image Composite Editor επιλέξαμε αυτή την προβολή και για αυτον τον λόγο υπάρχει το μαύρο φόντο. Με διαφορετική προβολή μέσα από το εργαλείο μπορεί να αλλάξει.

Πηγές:

[

- A Comparison of SIFT , SURF and ORB
- python.plainenglish.io/opencv-image-stitching-second-part-2
- towardsdatascience.com/image-panorama-stitching-with-opencv
- python.plainenglish.io/opencv-image-stitching-introduction
- Διαφάνειες Μαθήματος.

]