



## Αναφορά στην 2<sup>η</sup> εργασία 'Όραση Υπολογιστών'



Χανιώτης Παναγιώτης  
57636

05 – 12 - 2021

Όραση Υπολογιστών  
Εργασία 2

Χανιώτης Παναγιώτης  
57636

## Περιεχόμενα

Θεωρητική Προσέγγιση

Σελίδα 3

Ανάλυση του προγράμματος

Σελίδα 7

Παρατηρήσεις

Σελίδα 13

Δικές μου φωτογραφίες

Σελίδα 15

Βιβλιογραφία

Σελίδα 16



## Θεωρητική Προσέγγιση

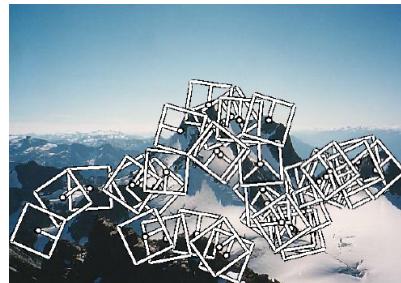
Στόχος αυτής της εργασίας είναι η δημιουργία πανοράματος ενώνοντας πολλές φωτογραφίες. Αυτό για να επιτευχθεί θα πρέπει να ακολουθηθεί μια σειρά από συγκεκριμένα βήματα. Για έχουμε ένα σωστό αποτέλεσμα θα πρέπει να λάβουμε υπόψη μας όλα εκείνα τα σημεία που ξεχωρίζουν μεταξύ τους σε μια εικόνα. Στη συνέχεια θα πρέπει να βρούμε έναν τρόπο να περιγράψουμε κάθε επιμέρους σημείο. Μετά, να επαναλάβουμε την παραπάνω διαδικασία και σε μία δεύτερη εικόνα. Έχοντας αυτές τις πληροφορίες πρέπει να αντιστοιχίσουμε μέρη που είναι ίδια και στις δύο εικόνες.

Γνωρίζοντας το αυτό, αρκεί να μετακινήσουμε την δεύτερη εικόνα ως προς την πρώτη με κατάλληλο τρόπο και κατάλληλες μετατροπές στη διάπλαση της ώστε να κολλήσει δίπλα στην πρώτη. Τέλος, αν επαναληφθεί όλη η παραπάνω διεργασία με την μια εικόνα ως την ένωση των δυο προηγούμενων και την ενώσουμε με την αμέσως επόμενη θα λάβουμε ως τελικό αποτέλεσμα μια πανοραμική έξοδο.

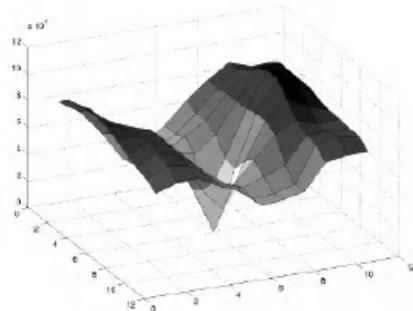
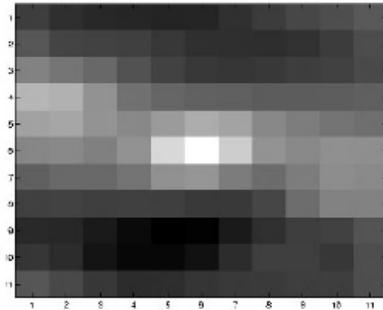


Πρώτο βήμα μας λοιπόν η άντληση πληροφοριών για τη θέση και την περιγραφή σημείων ενδιαφέροντας σε μια εικόνα. Αυτά τα σημεία μπορεί να είναι οτιδήποτε από βουνοκορφές, γωνίες σε κτίρια, πόρτες ή ακόμα και μια επιφάνεια με χιόνι που έχει ενδιαφέρον σχηματισμό. Αυτά τα λεγόμενα τοπικά χαρακτηριστικά λέγονται και σημεια-κλειδία (keypoint features) ή σημεία ενδιαφέροντος (interest points).





Έχει σημασία να περιγράψουμε πρακτικούς τρόπους για την εύρεση αυτών των χαρακτηριστικών. Το πόσο εύκολος είναι ο εντοπισμός αυτός εξαρτάται από την υφή που έχουν στην εικόνα, καθώς και σε πόσες κατευθύνσεις αλλάζει η κλίση (gradient) σε αυτά. Η έννοια της κλίσης μπορεί να γίνει πιο κατανοητή από το παρακάτω γράφημα

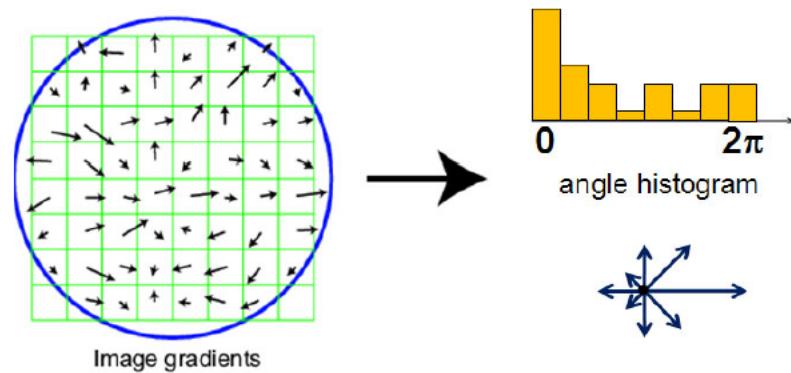


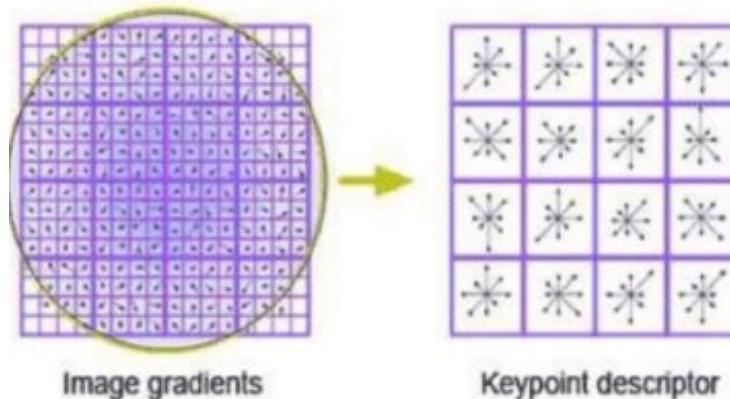
Η κλίμακα στην οποία αναζητούμε τα στοιχεία έχει σημασία. Αν δούμε μια εικόνα στο αρχικό της μέγεθος μπορεί φαινομενικά να μην έχει ιδιαίτερο αριθμό χαρακτηριστικών. Αν όμως την μεγεθύνουμε το πλαίσιο αναζήτησης αλλάζοντας την κλίμακα μπορεί να βρούμε. Στην αριστερή εικόνα βλέπουμε την αναζήτηση εικόνας σε μεγάλη και δεξιά σε μικρή κλίμακα.



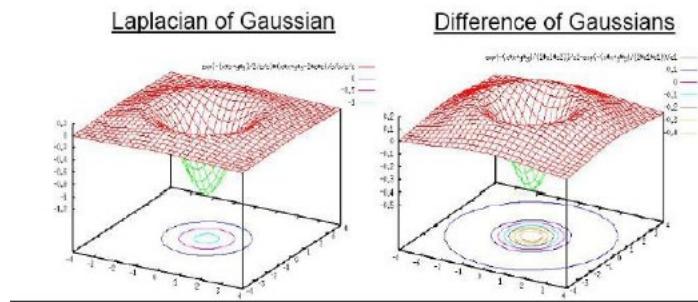


Ένας τέτοιος αλγόριθμος είναι ο SIFT. Ο SIFT υπολογίζει την κλίση σε όλα τα εικονοστοιχεία. Για το καθένα δημιουργεί μια περιοχή  $16 \times 16$  τις οποίες χωρίζει σύμφωνα με την παρακάτω εικόνα και δημιουργεί ιστόγραμμα με τους προσανατολισμούς της κλίσης ανά κάθε block. Ο υπολογισμός γίνεται με τον υπολογισμό της LoG (Laplacian of Gaussian)

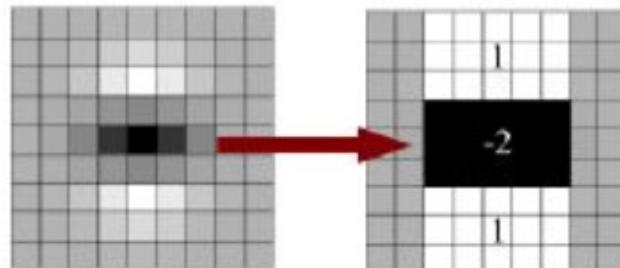




Μαζί με τον αλγόριθμο SIFT υπάρχει και αλγόριθμος SURF ο οποίος αποτελεί μια ταχύτερη εκδοχή του SIFT. Στο SIFT υπολογίζουμε το LoG με τη διαφορά του Gaussian για την εύρεση του χώρου κλίμακας. Το SURF προχωρά λίγο παραπέρα και προσεγγίζει το LoG με το Box Filter. Η παρακάτω εικόνα δείχνει μια επίδειξη μιας τέτοιας προσέγγισης.



### Η απλοποίηση του SURF



Τελευταίο βήμα είναι το ταίριασμα των χαρακτηριστικών και η μετακίνηση της εικόνας με κατάλληλο τρόπο ώστε να 'ραφτεί' δίπλα στην άλλη. Η διαδικασία του ταιριάσματος θα γίνει εξετάζοντας έναν συγκεκριμένο στοιχείων στη μια εικόνα και στη



συνέχεια θα ελέγξουμε πόσο απέχει από το κοντινότερο σε αυτό σημείο. Από αυτή την απόσταση και τον προσανατολισμό θα καταλήξουμε σε ένα ταίριασμα.

Τέλος, από τα σημεία που βρήκαμε θα πρέπει να δούμε πως συσχετίζονται γεωμετρικά οι δύο εικόνες ώστε να τις ενώσουμε. Αυτό θα το κάνουμε σύμφωνα με την μέθοδο RANSAC Συνοπτικά, η RANSAC επιλέγει τυχαία ζεύγη keypoints για να σχηματίσει μια γραμμή και στη συνέχεια δοκιμάζει πόσα άλλα πέφτουν σε αυτή τη γραμμή. Όσα βρίσκονται εκεί θα εξεταστούν και θα αντιστοιχηθούν με τα αντίστοιχα στην άλλη εικόνα.

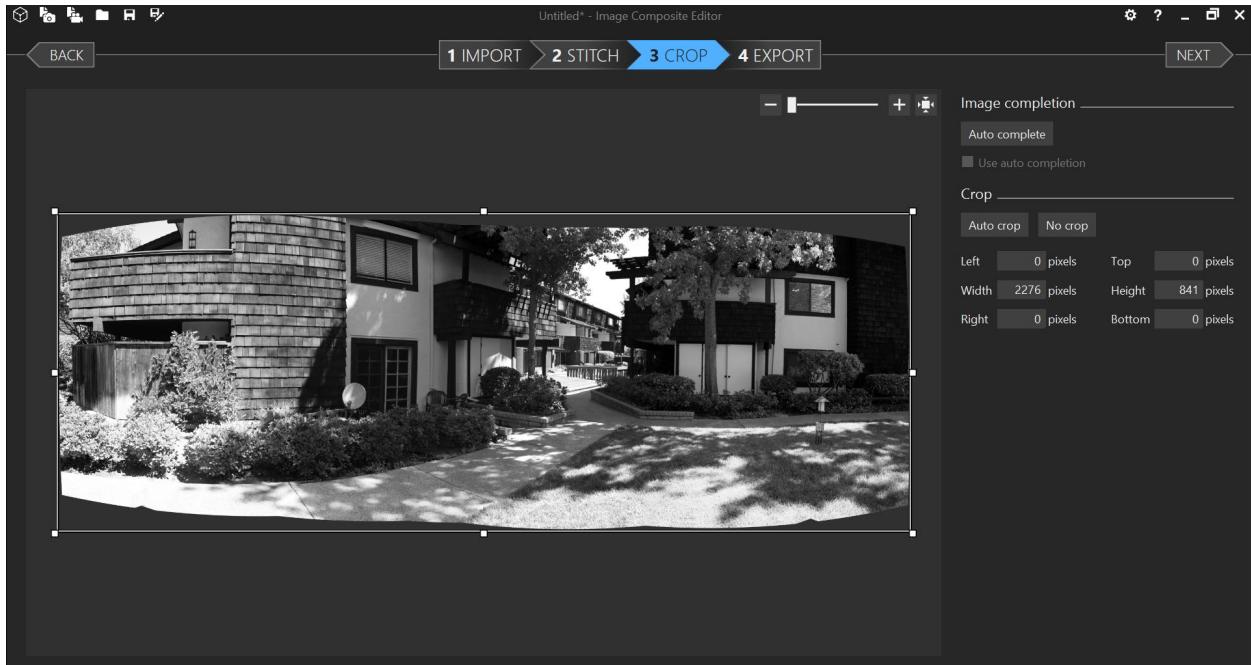


Ανάλυση του προγράμματος

Αρχικά, για να υπάρχει μέτρο σύγκρισης με μια ιδανική περίπτωση θα αξιοποιηθεί το παρακάτω πρόγραμμα της Microsoft. Το συγκεκριμένο πρόγραμμα χρησιμοποιεί ειδικές μεθόδους προκειμένου να μετατρέψει ένα σύνολο εικόνων σε πανόραμα.

Image composite editor





Ως αποτέλεσμα όπως βλέπουμε είναι εξαιρετικά καλό, με εξαίρεση την καμπυλότητα που έχει δημιουργηθεί στο κάτω μέρος. Αυτή η καμπυλότητα είναι λογικό να δημιουργείται από την καμπύλωση των ίδιων των εικόνων όταν 'ράβονται' μεταξύ τους.



Ακολουθεί η υλοποίηση σε κώδικα python με χρήση των βιβλιοθηκών της OpenCV καθώς και το μαθηματικό εργαλείο NumPy.

Η επιλογή των αρχείων γίνεται με την συνάρτηση filename την οποία χρησιμοποιούμε για παραπάνω ευκολία στην ανάγνωση και την εγγραφή εικόνων.

```
image1 = cv.imread(filename(5,choice))
```

Όλος ο κώδικας εκτελείται σε μια λούπα 4 φορές ώστε να ενωθούν μεταξύ τους και οι 5 εικόνες.

Δημιουργούμε

```
sift = cv.xfeatures2d_SIFT.create(1000)
```



ένα αντικείμενο sift με την παρακάτω εντολή προκειμένου να χρησιμοποιήσουμε τις συναρτήσεις του για τους υπολογισμούς. Ως όρισμα κατά την δημιουργία του αντικειμένου έχουμε 1000 που είναι ο μέγιστος αριθμός keypoints που θέλουμε να βρει

Τα keypoints των δυο εικόνων και τους αντίστοιχους περιγράφεις χαρακτηριστικών τους με τις εντολές

```
keyPoints1 = sift.detect(image1)
descriptor1 = sift.compute(image1, keyPoints1)

keyPoints2 = sift.detect(image2)
descriptor2 = sift.compute(image2, keyPoints2)
```

Σαν ενδιάμεσο βήμα για να δούμε τι έχει βρει ο αλγόριθμος προβάλλουμε σε παράθυρο τα σημεία ενδιαφέροντος

```
dimgKeypoint = cv.drawKeypoints(image2, keyPoints2, None)
cv.namedWindow('keypoints 2', cv.WINDOW_NORMAL)
cv.imshow('keypoints 2', dimgKeypoint)
```



Με τα παραπάνω δεδομένα (θέσεις και περιγραφές σημείων ενδιαφέροντος) είμαστε πλέον σε θέση να συσχετίσουμε τα κοινά στις δύο εικόνες σημεία. Για την υλοποίηση θα χρησιμοποιήσουμε την συνάρτηση matches. Η συνάρτηση έχει ως όρισμα τον πίνακα με τα δεδομένα τους δυο περιγράφεις των keypoints. Η συνάρτηση αρχικά επαναλαμβάνει μια διαδικασία για όσες φορές όσα είναι τα keypoints της εικόνας 1, λαμβάνοντας το μέγεθος του πίνακα των περιγραφών της. Μέσα στην επανάληψη,



αρχικά δημιουργούμε έναν πίνακα μεγέθους  $1 \times n$  (n: κάθε στοιχείο περιγραφής, όπως φωτεινότητα, θέση κλπ).

Ορίζουμε ως  $\text{diff1}$  το αποτέλεσμα της πράξης της αφαιρεσης της παραπάνω γραμμής από το σύνολο όλων των στηλών με τους περιγραφής από τον πίνακα  $\text{desc2}$ . Στη συνέχεια ορίζουμε πίνακα με τις αποστάσεις μεταξύ χαρακτηριστικών,  $\text{distances1}$  στον οποίο αθροίζουμε την απόλυτη τιμή της αφαιρεσης που υπολογίσαμε στο προηγούμενο βήμα (άθροιση στον άξονα x). Από το συγκεκριμένο πίνακα κρατάμε επίσης τον δείκτη του στοιχείου με τη μικρότερη απόσταση καθώς και την τιμή του ( $i2$ ,  $\text{mindist2}$  αντίστοιχα).

Επαναλαμβάνουμε όλο το παραπάνω στον πίνακα των περιγραφέων της εικόνας 1 με διαφορά ότι η στήλη από την οποία θα αφαιρέσουμε θα είναι της στήλης με την μικρότερη απόσταση από την προηγούμενο υπολογισμό. Βρίσκουμε το αντίστοιχο στοιχείο με την ελάχιστη απόσταση και αν δούμε ότι είναι αυτό το οποίο μελετάμε στην επανάληψη τότε έχουμε ένα  $\text{match}$ . Έτσι λέμε ότι έχουμε cross-matching.

```
|def match(desc1, desc2):  
  
    desc1_n = desc1.shape[0]  
  
    matches = []  
    for i in range(desc1_n):  
  
        fv1 = desc1[i, :]  
        diff1 = desc2 - fv1  
        diff1 = np.abs(diff1)  
        distances1 = np.sum(diff1, axis=1)  
  
        i2 = np.argmin(distances1) # Image 2 to Image 1  
        mindist2 = distances1[i2]  
  
        fv2 = desc2[i2, :]  
        diff2 = desc1 - fv2  
        diff2 = np.abs(diff2)  
        distances2 = np.sum(diff2, axis=1)  
  
        i1 = np.argmin(distances2) # Image 1 to Image 2  
  
        if (i1 == i):  
            matches.append(cv.DMatch(i, i2, mindist2))  
  
    return matches
```

Προβάλουμε τα  $\text{matches}$  στην οθόνη

```
dimgMatch = cv.drawMatches(image1, descriptor1[0], image2, descriptor2[0], matches, None)  
cv.namedWindow('matches', cv.WINDOW_NORMAL)  
cv.imshow('matches', dimgMatch)
```





Βάσει των ταιριασμάτων που βρήκαμε παραπάνω, ορίζουμε πίνακες που περιλαμβάνουν όλα τα σημεία ταιριάσματος. Αυτοί οι πίνακες θα αξιοποιηθούν για ορίσουμε την γεωμετρική μετατόπιση της δεύτερης εικόνας ως προς την πρώτη για να γίνει επιτυχές ένωση.

Σημαντική σημείωση εδώ: τον πίνακα με τα *matches* έχει αντικείμενα που δημιούργησε η συνάρτηση *DMatch*. Λόγω αυτού έχουμε δυνατότητα να προσπελάσουμε τα σημεία με την μεταβλητή *queryIdx* για τα στοιχεία της εικόνας 1 και την μεταβλητή *trainIdx* της εικόνας 2. Οι ονομασίες *train* (εκπαίδευση) και *query* (ερώτημα) είναι διαδεδομένες στον χώρο της αναγνώρισης αντικειμένων εικόνων όπου στην πρώτη περίπτωση δίνουμε τα δεδομένο ένα αντικείμενο στον αλγόριθμο μας, όπως τον *sift*, και τον καλούμε να τα αναγνωρίσει σε μια άλλη εικόνα.

```
point_list1 = []
point_list2 = []
for x in matches:
    point_list1.append(keyPoints1[x.queryIdx].pt)
    point_list2.append(keyPoints2[x.trainIdx].pt)
point_list1 = np.array(point_list1)
point_list2 = np.array(point_list2)
```

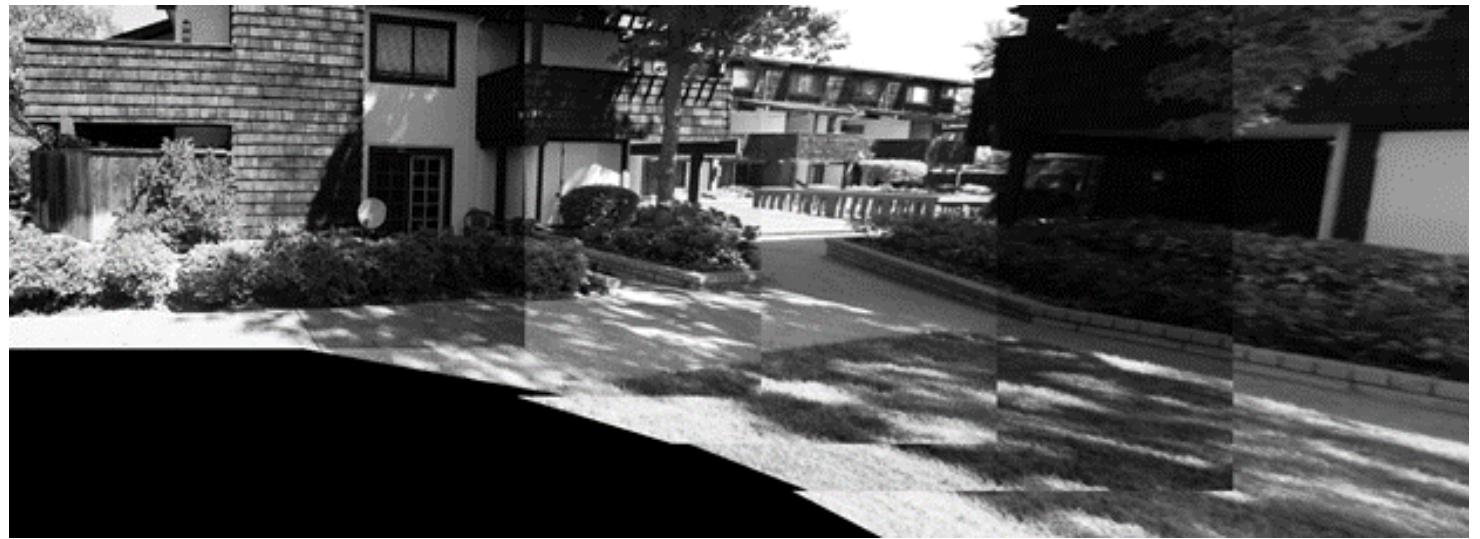


Με το σύνολο των αντικειμένων και την μέθοδο γεωμετρικού συσχετισμού RANSAC βρίσκουμε έναν πίνακα  $M$  μεγέθους  $3 \times 3$  σύμφωνα με τον οποίο μετασχηματίζουμε την εικόνα 2. Επίσης, την μετακινούμε κατά ένα offset για να ταιριάξει με την εικόνα 1. Τέλος, το αποτέλεσμα αυτής της ένωσης το αποθηκεύουμε στο αντικείμενο εικόνας της εικόνας 1 ώστε να χρησιμοποιηθεί αυτό στην επόμενη επανάληψης.

```
M, mask = cv.findHomography(point_list2, point_list1 , cv.RANSAC)
...
image_union = cv.warpPerspective(image2, M, (image1.shape[1]+3000, image1.shape[0]+1000))
image_union[0: image1.shape[0], 0: image1.shape[1]] = image1

image1 = image_union
```





Αν αλλάξουμε τον αλγόριθμο από τον SIFT στον SURF λαμβάνουμε το παρακάτω αποτέλεσμα.

Θα χρειαστεί να αλλάξουμε τον αριθμό *keypoints* που θέλουμε να βρει.

```
cv.xfeatures2d_SURF.create(100)
```



## Παρατηρήσεις

Η παραπάνω διαδικασία αδυνατεί να διατηρήσει παρόμοια χρωματική φωτεινότητα στην εικόνα, σε αντίθεση με το πρόγραμμα της Microsoft. Οι αλλαγές στο φως είναι πολύ έντονες στα σημεία 'ραφής' των εικόνων.



Για να γίνει επιτυχώς ο μετασχηματισμός πρέπει να μετακινηθεί η μια εικόνα προς τα δεξιά σύμφωνα με ένα offset που ορίζεται από εμάς. Αυτό δεν είναι πρακτικό γιατί ο αλγόριθμος δεν μπορεί να λειτουργήσει για κάθε πιθανό σύνολο εικόνων χωρίς μετατροπές. Αν γίνει αυτό το αποτέλεσμα θα είναι όπως φαίνεται παρακάτω



Άλλο ένα ζήτημα είναι αυτό της καμπύλωσης των εικόνων κατά τη δημιουργία του πανοράματος. Αυτό δημιουργεί μαύρα κενά στο κάτω μέρος.

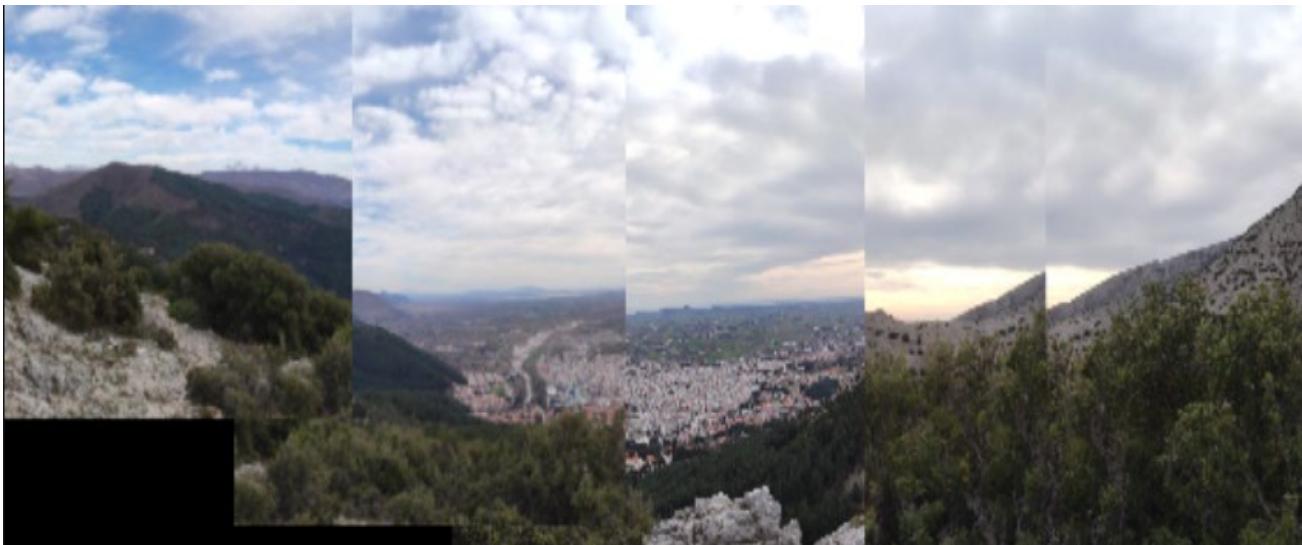


## ΔΙΚΕΣ ΜΟΥ ΦΩΤΟΓΡΑΦΙΕΣ

Από προγράμμα της Microsoft



Με τον παραπάνω κώδικα και την μέθοδο SIFT.



## Βιβλιογραφία

Computer Vision Algorithms and Applications, Richard Szeliski

Computer Vision Lectures, Ioannis Pratikakis

Computer Vision Lab Course, Lazaros Tsochatzidis

[https://docs.opencv.org/3.4/da/df5/tutorial\\_py\\_sift\\_intro.html](https://docs.opencv.org/3.4/da/df5/tutorial_py_sift_intro.html)

[https://docs.opencv.org/4.x/df/dd2/tutorial\\_py\\_surf\\_intro.html](https://docs.opencv.org/4.x/df/dd2/tutorial_py_surf_intro.html)

[https://docs.opencv.org/4.x/dc/dc3/tutorial\\_py\\_matcher.html](https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html)

[https://docs.opencv.org/4.x/d1/de0/tutorial\\_py\\_feature\\_homography.html](https://docs.opencv.org/4.x/d1/de0/tutorial_py_feature_homography.html)

