Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Όραση Υπολογιστών

Εξάμηνο 8ο (Εαρινό Εξάμηνο 2021-2022)

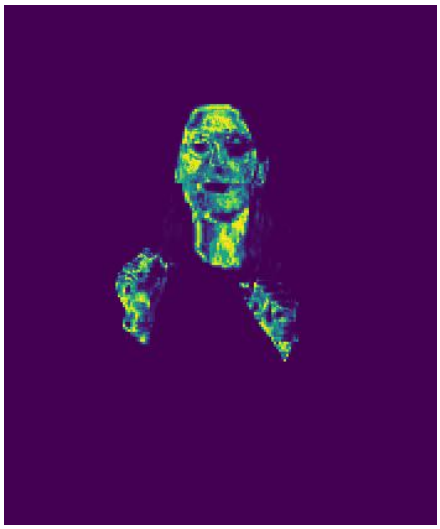
2η Εργαστηριακή Άσκηση ,

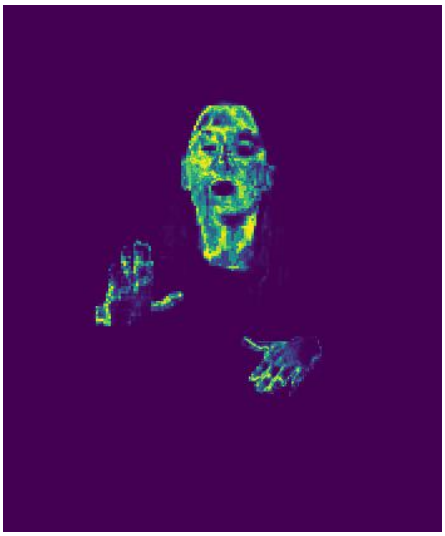
Χάιδος Νικόλαος el18096, Σπανός Νικόλαος el18822

Μέρος 1

1.1) Για την ανίχνευση σημείων δέρματος, αρχικά μετατρέπουμε τις RGB εικόνες σε χρωματική κωδικοποίηση YCbCr, και κρατάμε μόνο τις συνιστώσες Cb και Cr. Αυτό γίνεται, για να κρατήσουμε μόνο το πραγματικό χρώμα του δέρματος, και όχι την συνιστώσα Y της «φωτεινότητας» η οποία θα μεταβάλλεται ανάλογα με τις συνθήκες φωτισμού.

Ύστερα, αφού βρήκαμε τον μέσο όρο των συνιστωσών Cb, Cr και τον πίνακα συνδιακύμανσης από την εικόνα εκπαίδευσης, βρήκαμε την τιμή του κάθε pixel του βίντεο, πάνω στην τελική Γκαουσιανή. Μπορούμε να δούμε, ότι αυτή η πιθανοτική προσέγγιση, εντοπίζει αρκετά καλά τις περιοχές δέρματος της εικόνας:

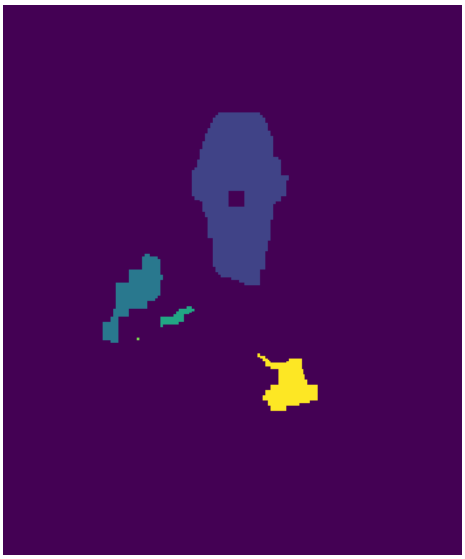




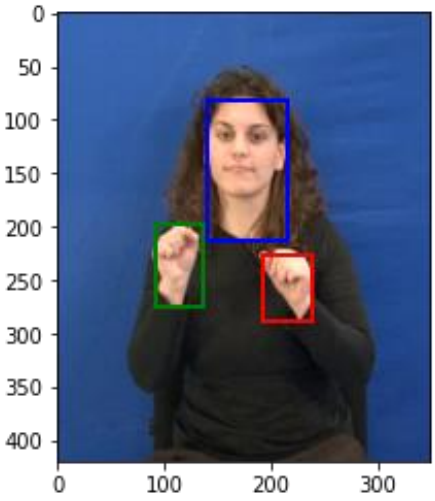
Ύστερα, κατωφλιοποιούμε την κατανομή, και κρατάμε μόνο τα pixel τα οποία έχουν τιμή μεγαλύτερη του 0.23 το οποίο μάς δίνει τα ακόλουθα σύνολα-εικόνες:

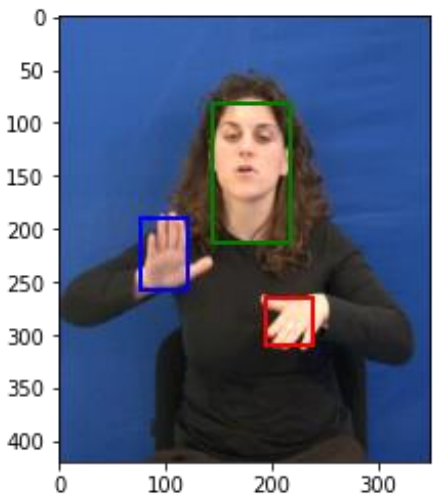
Μετά, κάνουμε ένα Opening με πυρήνα μεγέθους **4x4** pixel, και Closing με πυρήνα μεγέθους **10x10** pixel. Πειραματιστήκαμε με πολλά άλλα μεγέθη πυρήνων, αλλά τις περισσότερες αποτύγχανε να ξεχωρίσει το σύνολο του προσώπου με αυτό των χεριών (ειδικά όταν τα χέρια ήταν πολύ κοντά στο πρόσωπο της νοηματίστριας). Μετά το Opening και το Closing, οι καινούριες εικόνες-σύνολα είναι οι εξής:

Με τις εντολές **label** και **np.unique** της python, μπορούμε να ξεχωρίσουμε τα συνεκτικά σύνολα της εικόνας, άρα θα έχουμε:



Τέλος, για τον υπολογισμό των ορίων των bounding boxes, απλά κρατάμε τα τρία μεγαλύτερα συνεκτικά σύνολα που εντοπίσαμε (δύο χέρια και το πρόσωπο), και σχεδιάζουμε ένα ορθογώνιο που να τα περικλείει πλήρως:





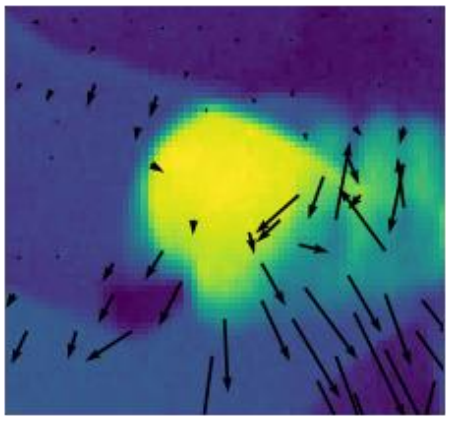
1.2.1) Ο αλγόριθμος Lucas-Kanade είναι ένας από τους πιο ευρέως χρησιμοποιούμενους αλγορίθμους για υπολογισμό οπτικής ροής. Βασίζεται στην μέθοδο ελαχίστων τετραγώνων, και ο τρόπος που το υλοποιούμε διακριτά σε επίπεδο pixel μιας εικόνας, είναι ο υπολογισμός του διανύσματος **u** σε κάθε pixel της εικόνας.

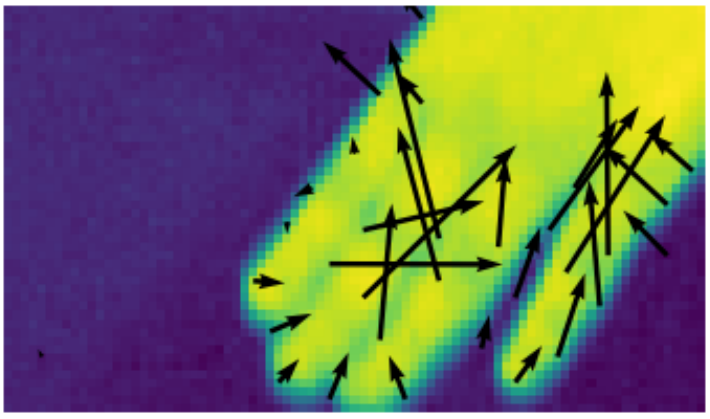
Αρχικά υπολογίσαμε τα A_1 $και$ A_2 παίρνοντας τις μερικές παραγώγους της εικόνας, μέσω της **np.gradient**. Ύστερα, υπολογίσαμε όλες τις μετατοπισμένες εικόνες κατά $\boldsymbol{d_i}$ μέσω της **map_coordinates**, και φτιάξαμε το φίλτρο ισοτροπικής γκαουσιανής (κλίμακας ρ) για να φιλτράρουμε όλα τα στοιχεία των δύο πινάκων.

Τέλος, υπολογίσαμε τον πολλαπλασιασμό πινάκων για κάθε pixel της εικόνας, και ανανεώνουμε το διάνυσμα d ως εξής: $\boldsymbol{d_{i+1} = d_i + u}$.

Για την συνθήκη διακοπής των επαναλήψεων, χρησιμοποιήσαμε την L2 νόρμα του διανύσματος $\boldsymbol{u}$**.** Συγκεκριμένα, αν σε κάποια επανάληψη, αυτή η νόρμα είναι μικρότερη του 0.02, τότε ο αλγόριθμος σταματάει, και επιστρέφει τα διανύσματα $\boldsymbol{d}$. Αλλιώς, τρέχει μέχρι να ολοκληρώσει 300 επαναλήψεις. Γενικά, στις περισσότερες περιπτώσεις παρατηρήσαμε ότι ο μονοκλιμακωτός LK, σπανίως ικανοποιούσε την συνθήκη σύγκλισης, και τις περισσότερες φορές έφτανε το όριο των 300 επαναλήψεων.

Για τις παραμέτρους (ε, ρ) που χρησιμοποιήσαμε, παρατηρήσαμε ότι όταν είχαμε πολύ μικρό ε (π.χ. 0.001) αρκετές εικόνες έβγαζαν πολύ μεγάλη, λανθασμένη οπτική ροή, ή και μη-φυσιολογική ροή που έδειχνε σε πολλές αντίθετες κατευθύνσεις, όπως:





Αυτά τα φαινόμενα τα μειώσαμε όσο μπορούσαμε, θέτοντας τις παραμέτρους ρ και ε ίσες με 2 και 0.005 αντίστοιχα (στο notebook έχουμε βάλει πολλές εικόνες με τις παραμέτρους που πειραματιστήκαμε).

1.2.2) Για τον υπολογισμό των διανυσμάτων μετατόπισης ενός bounding box (ή και γενικότερα μιας συγκεκριμένης περιοχής που έχουμε υπολογίσει την οπτική ροή), υλοποιούμε την συνάρτηση $\boldsymbol{displ}(\boldsymbol{d\_x}, \boldsymbol{d\_y}, \boldsymbol{threshold})$, η οποία παίρνει σαν παραμέτρους τις οπτικές ροές των σημείων ενδιαφέροντος, και ένα ποσοστό (%) για το thresholding.

Ο τρόπος που δουλεύει ο αλγόριθμος είναι να υπολογίζει το σημείο που έχει την μέγιστη ενέργεια E_{max} οπτικής ροής, και να κρατάει μετά μόνο τις οπτικές ροές οι οποίες είναι $\geq thershold * E_{max}$. Τέλος, από αυτές τις οπτικές ροές, παίρνουμε τον μέσο όρο, και εν τέλει, θα έχουμε ένα αντιπροσωπευτικό διάνυσμα οπτικής ροής για όλο το bounding box.

Βρήκαμε, ότι για $threshold = 0.5 \ \ (50\%)$, τα αποτελέσματα ήταν ικανοποιητικά.

1.2.3) Τα προβλήματα του Lucas-Kanade που παρουσιάσαμε στην ενότητα 1.2.2, εν μέρει οφείλονται και στην μεγάλη κίνηση που έχουν μερικές εικόνες (κυρίως τα χέρια). Συγκεκριμένα, αν η κίνηση της εικόνας υπερβαίνει τα 1-2 pixel, τότε αρχίζουμε ήδη να βλέπουμε κάποιες παραμορφώσεις στην οπτική ροή που δίνει ο μονοκλιμακωτός Lucas-Kanade.

Αυτό μπορεί να επιλυθεί εφαρμόζοντας τον αλγόριθμο σε πολλαπλές κλίμακες. Συγκεκριμένα, υλοποιήσαμε την συνάρτηση:

$$\boldsymbol{multi\_lk}(\boldsymbol{I1}, \boldsymbol{I2}, \boldsymbol{features}, \boldsymbol{rho}, \boldsymbol{epsilon}, \boldsymbol{dx\_0}, \boldsymbol{dy\_0}, \boldsymbol{Num})$$

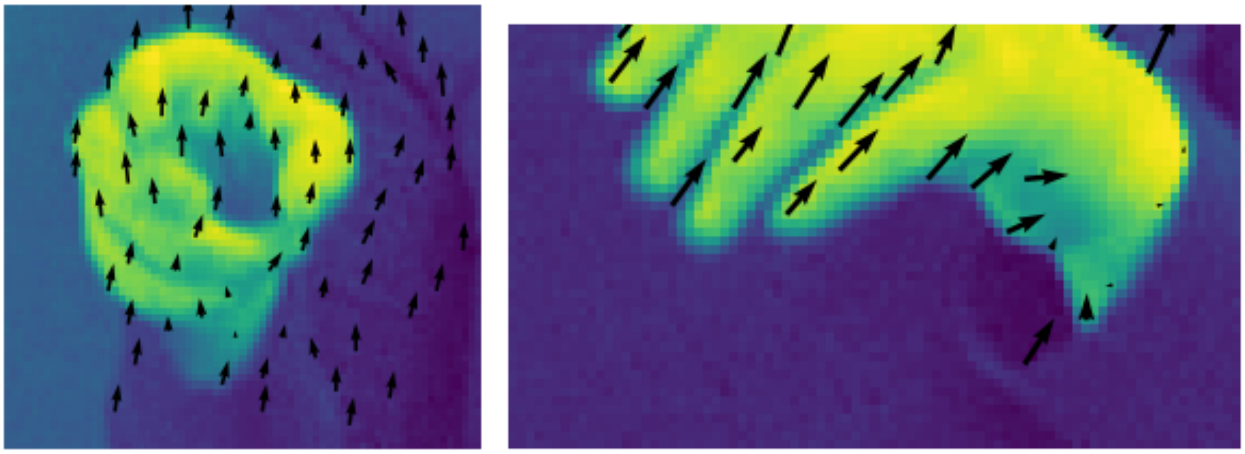
η οποία παίρνει ως είσοδο τις δύο εικόνες, τα σημεία ενδιαφέροντος, τις παραμέτρους ρ, ε, τα αρχικά διανύσματα $\boldsymbol{d}$ και το πλήθος των κλιμάκων που θα εκτελεστεί.

Ο αλγόριθμος, υποδιπλασιάζει την κλίμακα της εικόνας (όσες φορές όσο το $\boldsymbol{Num}$) για να δημιουργήσει τις κλίμακες στις οποίες θα εφαρμόσει τον μονοκλιμακωτό LK. Όμως, πριν την υποδειγματοληψία, φιλτράρουμε την εικόνα με ένα Γκαουσιανό πυρήνα (3 pixel) για να μετριάσουμε την φασματική αναδίπλωση (δημιουργώντας την γκαουσιανή πυραμίδα της εικόνας).

Ύστερα, τρέχουμε τον αλγόριθμο LK πάνω σε κάθε εικόνα, ξεκινώντας με την μικρότερη κλίμακα, και αρχικές συνθήκες τα $\boldsymbol{dx\_0}, \boldsymbol{dy\_0}$. Κάθε φορά που μεταβαίνουμε στην εικόνα της αμέσως επόμενης κλίμακας, υπολογίζουμε με την συνάρτηση $\boldsymbol{displ}$ (1.2.2) το διάνυσμα συνολικής μετατόπισης στο bounding box, και το θέτουμε το διπλάσιό του ως αρχική συνθήκη για τον LK της επόμενης κλίμακας.

Ο πολυκλιμακωτός LK βοηθάει στην περίπτωση που η κίνηση της εικόνας είναι μεγαλύτερη των 1-2 pixel, αλλά και στην γενικότερη ποιότητα της οπτικής ροής. Επίσης, μεγάλη διαφορά παρατηρήσαμε στους χρόνους σύγκλισης. Επειδή ο LK κάθε κλίμακας θα ξεκινάει με τις βέλτιστες αρχικές συνθήκες (που πήρε από την προηγούμενη κλίμακα), σχεδόν πάντα έφτανε γρήγορα το κριτήριο σύγκλισης, με αποτέλεσμα να τρέχει σε λιγότερο χρόνο από ότι ο μονοκλιμακωτός LK.

Μερικές εικόνες οπτικών ροών από τον πολυκλιμακωτό:



Τα καλύτερα αποτελέσματα του πολυκλιμακωτού φαίνονται στα χέρια (στα οποία αρκετές φορές αποτυγχάνει ο μονοκλιμακωτός, λόγω μεγάλης κίνησης).

Μέρος 2

Ανιχνευτής Harris

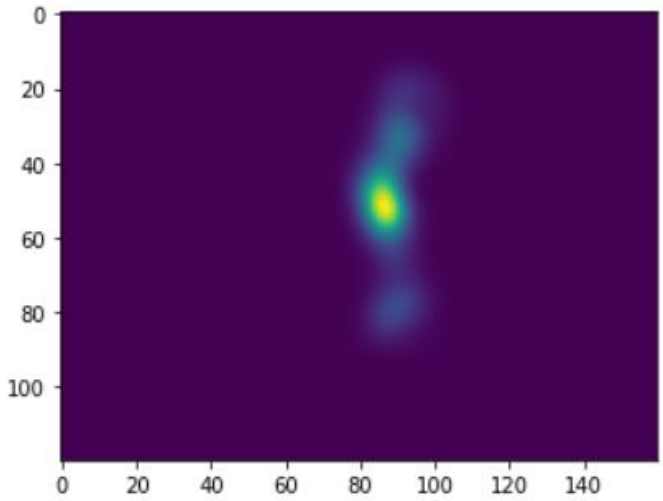
2.1.1) Αρχικά υπολογίζουμε τους πίνακες L κάνοντας μονοδιάστατη συνέλιξη στις δύο πρώτες διαστάσεις με μια Gaussian πυκνότητας σ και στην διάσταση του χρόνου με μια Gaussian πυκνότητας τ και στην συνέχεια κάνοντας μονοδιάστατη συνέλιξη σε κάθε άξονα με τον πίνακα $[-1\ 0\ 1]^T$ για να υπολογίσουμε τις μερικές παραγώγους. Ύστερα, υπολογίζουμε τα στοιχεία του πίνακα και εκτελούμε την ίδια διαδικασία με κατανομές πυκνότητας $s\sigma$ και $s\tau$. Ύστερα, χρησιμοποιώντας τις συναρτήσεις **numpy.linalg.det** και **numpy.linalg.trace** υπολογίζουμε το κριτήριο γωνιότητας.

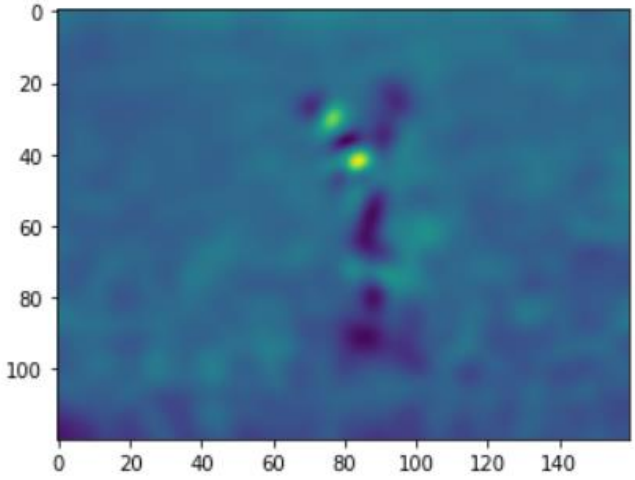
Ανιχνευτής Gabor

2.1.2) Υλοποιούμε τα δύο φίλτρα h_{ev} και h_{od} και εκτελούμε συνέλιξη με τον δισδιάστατο Gaussian πυρήνα στις χωρικές συνιστώσες και συνέλιξη με τα φίλτρα στην χρονική συνιστώσα. Προσθέτοντας τα τετράγωνα των αποτελεσμάτων παίρνουμε το κριτήριο σημαντικότητας.

Αφού υπολογίσουμε τα κριτήρια και των δύο ανιχνευτών, βρίσκουμε τα πρώτα 600 που τα μεγιστοποιούν. Επίσης, προσθέτουμε και στους δύο πίνακες μια κάθετη στήλη με τις τιμές της κλίμακας στην οποία υπολογίστηκαν(για χρήση στην show_detection συνάρτηση).

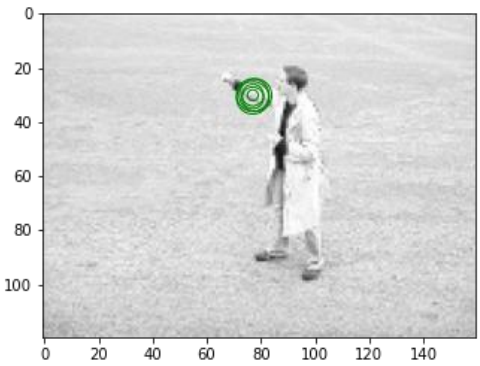
Ενδεικτικά θα παρουσιάσουμε μια οπτικοποίηση του κριτηρίου Harris(αριστερά) και Gabor(δεξιά) για το ίδιο frame:

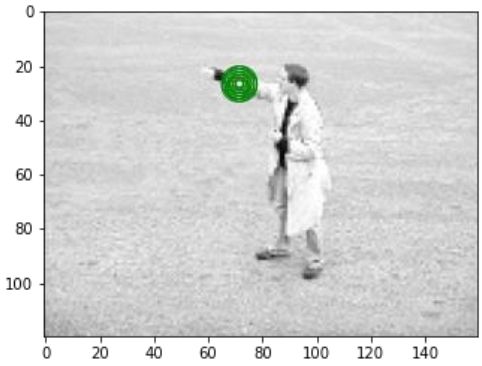


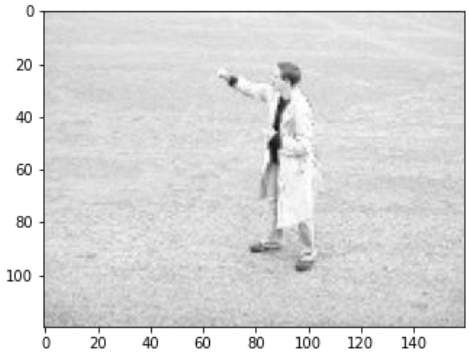


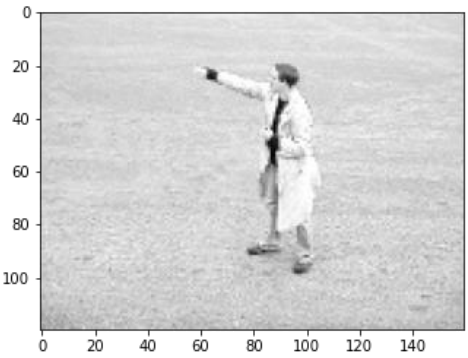
Παρατηρούμε ότι ακόμα και για ότι ίδιο frame, οι ανιχνευτές παρουσιάζουν διαφορετικές τιμές πάνω στο σώμα του δρομέα, με τον Harris να δίνει έμφαση στον αγκώνα και τον Gabor στα χέρια. Ο Harris, επίσης, παρουσιάζει πολύ μικρότερες τιμές στις εκτός του σώματος περιοχές, απορρίπτοντας τες καλύτερα.

Ενδεικτικά, θα παρουσιάσουμε κάποια frames στο παράδειγμα boxing για να συγκρίνουμε τις μεθόδους:



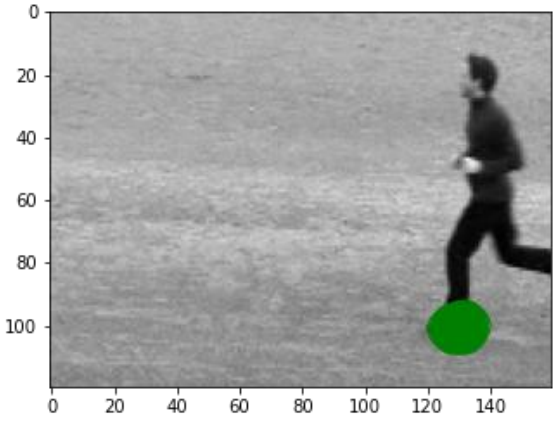
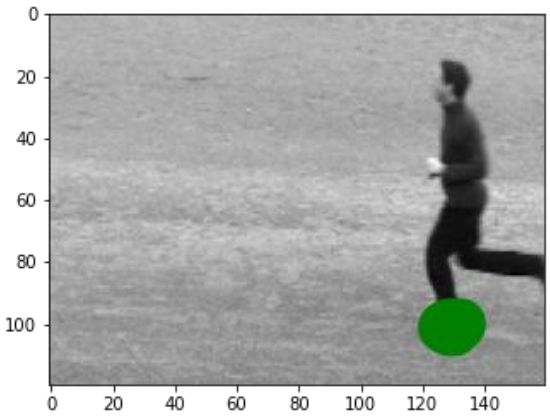
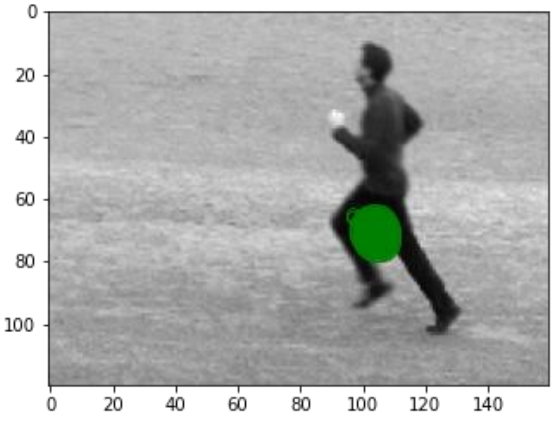
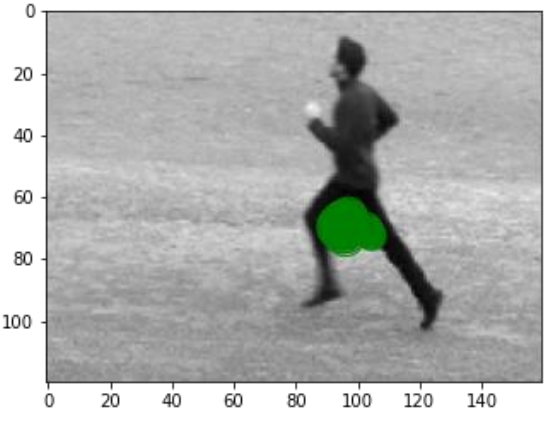






Και στις δύο σειρές παρατηρούμε τα frames 26 και 27 του “person12_boxing_d3_uncomp.avi”. Στην πρώτη σειρά η υλοποίηση με τον ανιχνευτή Gabor έχει εντοπίσει την κίνηση του χεριού, ενώ η υλοποίηση με ανιχνευτή Harris που παρουσιάζεται στην δεύτερη σειρά την κίνηση δεν την έχει ανιχνεύσει.

Αντιθέτως, στα δύο παρακάτω frames του run του βίντεο “person01_running_d1_uncomp.avi” ο ανιχνευτής Harris στην πρώτη σειρά ανιχνεύει τα πόδια του δρομέα , ενώ ο ανιχνευτής Gabor σημεία κοντά στα γόνατα.

Οι διαφορές οφείλονται στο ότι οι ανιχνευτές προσπαθούν να μεγιστοποιήσουν διαφορετικά κριτήρια, οπότε και οι περιοχές ενδιαφέροντος τείνουν να μην είναι ίδιες.

2.2) Υλοποιούμε την συνάρτηση **Descriptors** η οποία επιστρέφει τους περιγραφητές HOG και HOF. Ο περιγραφητής HOG υπολογίζει το ιστόγραμμα μέσω των δύο χωρικών παραγωγών γύρω από τα σημεία ενδιαφέροντος ($\pm 4\sigma$ σε κάθε κατεύθυνση). Ο περιγραφητής HOF υπολογίζει το ιστόγραμμα με το πυκνό πεδίο οπτικής ροής TVL1.

2.3) Έχοντας υλοποιήσει τα ιστογράμματα κατευθύνσεων εκπαιδεύουμε τον SVM ταξινομητή με κάθε συνδυασμό ανιχνευτών και περιγραφητών. Σύμφωνα με το text αρχείο που δόθηκε, χωρίσαμε τα βίντεο σε train set και test set και πήραμε τα κάτωθι αποτελέσματα:

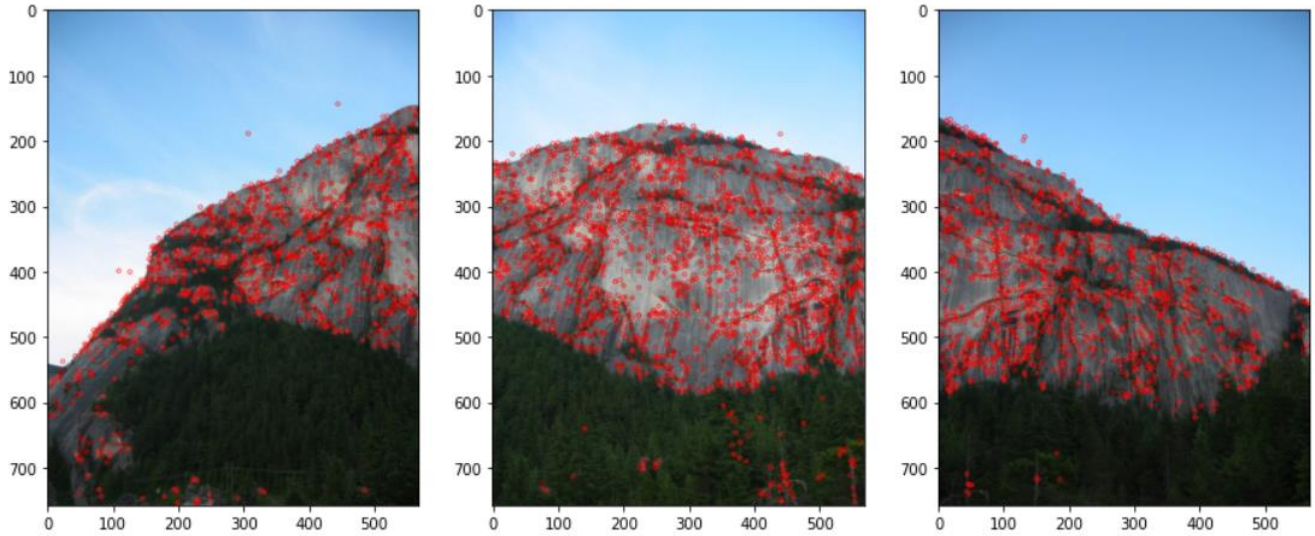
Accuracy Metrics	HOG	HOF	HOG/HOF
Gabor	0.834	1.0	1.0
Harris	0.5834	0.667	0.667

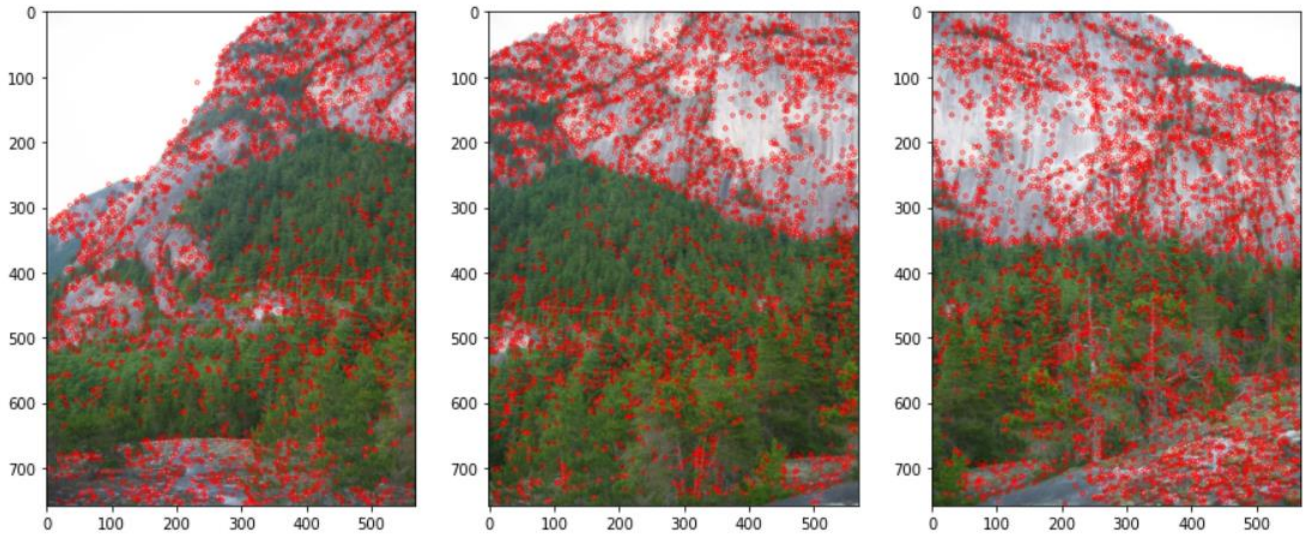
Παρατηρούμε ότι σε γενικές γραμμές ο ανιχνευτής Gabor είχε τα καλύτερα αποτελέσματα, ενώ οι συνδυασμός HOF-Gabor και HOG/HOF-Gabor παρουσίασαν το μέγιστο accuracy (100%).

Μέρος 3

Βήμα 0) Αρχικά διαβάζουμε τις εικόνες 1 έως 6, και τις μετατρέπουμε σε χρωματισμό RGB.

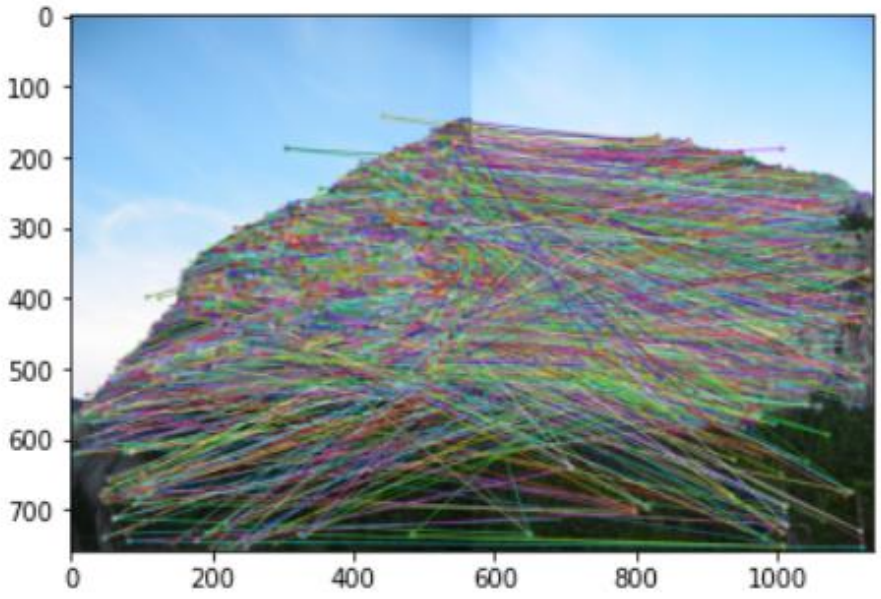
Βήμα 1) Με την συνάρτηση $\boldsymbol{sift.detectAndCompute}$ υπολογίζουμε σημεία ενδιαφέροντος σε κάθε εικόνα, αλλά και τοπικούς περιγραφητές για κάθε σημείο. Εντοπίστηκαν τα εξής σημεία πάνω στις αρχικές εικόνες:





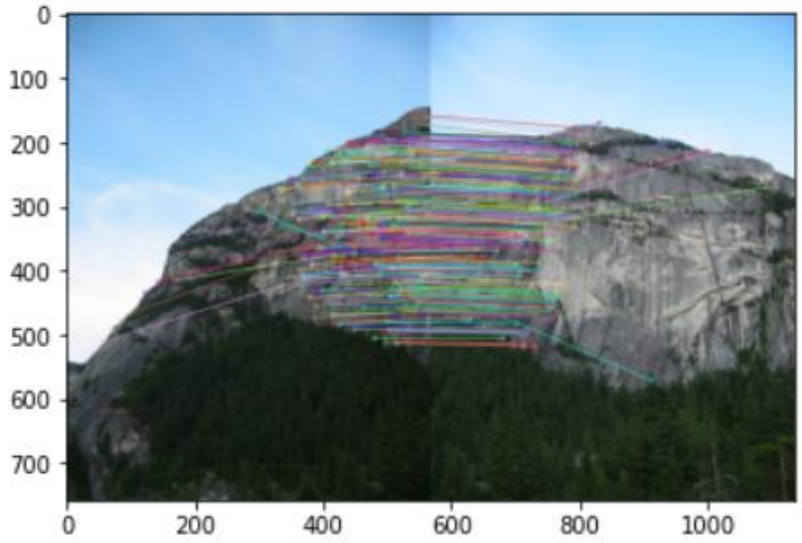
Βήμα 2) Ύστερα, για το ταίριασμα των σημείων ενδιαφέροντος, επιλέξαμε να χρησιμοποιήσουμε την συνάρτηση $\boldsymbol{cv.FlannBasedMatcher}$, καθώς έδινε καλά αποτελέσματα, αλλά και είναι αρκετά πιο γρήγορη από άλλες εξαντλητικές μεθόδους (π.χ. $BFMatcher$). Με αυτήν την συνάρτηση υπολογίζουμε τα δύο πιο “κοντινά” σημεία ενδιαφέροντος μέσω $knnMatch$ μεταξύ των περιγραφητών τους, για κάθε σημείο ενδιαφέροντος, ανάμεσα σε δύο εικόνες.

Αφού γίνει το matching, στις δύο εικόνες, θα έχουμε το εξής αποτέλεσμα:



Βλέπουμε ότι αρκετά από αυτά τα matches δεν είναι σωστά, γι' αυτό στο επόμενα βήμα θα κρατήσουμε μόνο τα "καλύτερα" σημεία ως προς ένα κριτήριο.

Βήμα 3) Για κάθε matched σημείο, έχουμε πάρει δύο αποστάσεις (για το πρώτο καλύτερο match, και το δεύτερο καλύτερο match). Θα κρατήσουμε μόνο τα σημεία, για τα οποία ο λόγος των δύο αποστάσεων είναι μικρότερος του 0.75. Έτσι, μετά θα έχουμε:



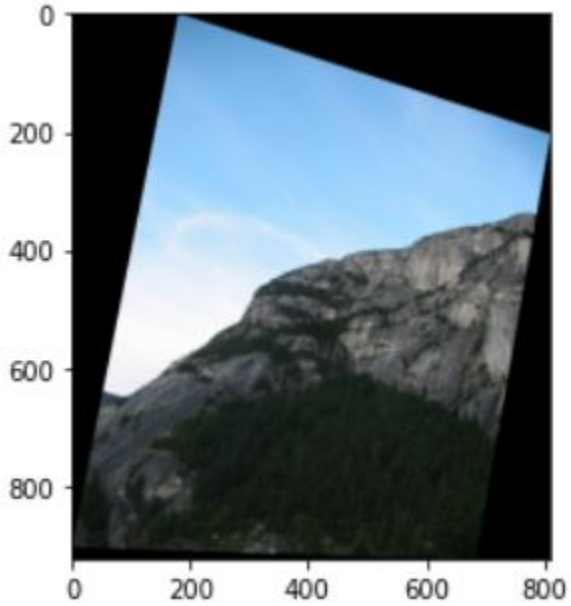
Βλέπουμε ότι έχουν παραμείνει κυρίως τα σημεία ενδιαφέροντος, τα οποία βρίσκονται και στις δύο εικόνες, δηλαδή έχει γίνει επιτυχές matching.

Βήμα 4) Στην παραπάνω εικόνα, μπορούμε να δούμε ότι ακόμα υπάρχουν μερικά outliers στα σημεία που έχουν γίνει matched. Γι' αυτό θα χρησιμοποιήσουμε την μέθοδο $RANSAC$ με σκοπό να υπολογίσουμε την ομογραφία H, χωρίς να επηρεαστεί αρνητικά από τα outliers. Τον υπολογισμό του H τον κάναμε με την συνάρτηση $\boldsymbol{cv.findHomography}$, και για το κατώφλι του $RANSAC$, είναι καλό να βρίσκεται μεταξύ 1 και 10, γι'αυτό το θέσαμε ίσο με 5.

Βήμα 5) Σε αυτό το βήμα, υπολογίσαμε την μετασχηματισμένη εικόνα, κάνοντας τα εξής βήματα:

1. Με την συνάρτηση $\boldsymbol{cv.perspectiveTransform}$ υπολογίσαμε τις νέες θέσεις των γωνιών πάνω την μετασχηματισμένη εικόνα $img1\_warped$.
2. Με αυτές τις θέσεις (οι οποίες είναι στο σύστημα συντεταγμένων της εικόνας $img2$) υπολογίσαμε τις συνολικές διαστάσεις που πρέπει να έχει η $img1\_warped$ (μαζί με τις μαύρες περιοχές), αλλά και τις συντεταγμένες τις πάνω αριστερά γωνίας (για να το επιστρέφει η συνάρτηση $projectionImage$)
3. Ύστερα, για το $inverse\ warping$, παίρνουμε όλα τα σημεία της $img1\_warped$, βάζουμε όλες τις συντεταγμένες τους σε ένα πίνακα, και τις μετακινούμε κατά το offset της πάνω αριστερά γωνίας (έτσι ώστε να είναι στο σωστό σύστημα συντεταγμένων)
4. Περνάμε αυτόν τον πίνακα από την συνάρτηση $cv.perspectiveTransform$, με πίνακα ομογραφίας, τον αντίστροφο του H, έτσι ώστε να βρεθούμε στις συντεταγμένες της αρχικής εικόνας $img1$
5. Μετά, για όλα τα σημεία που θα επιστρέψει, κρατάμε μόνο αυτά τα οποία βρίσκονται μέσα στην αρχική εικόνα (επειδή οι μαύρες περιοχές της $img1\_warped$ δεν θα αντιστοιχούν σε κάποιο πραγματικό pixel της $img1$)
6. Μετά, επειδή οι συντεταγμένες έχουν εν γένει τιμές $float$, τις κάνουμε $np.floor$, έτσι ώστε να βρούμε σε ποιο pixel αντιστοιχίζονται (δοκιμάσαμε να κάνουμε και $linear\ intepolation$ χρησιμοποιώντας την $scipy.interpolate.griddata$, αλλά ήταν σημαντικά πιο αργό από ένα $np.floor$ ή $np.ceil$).
7. Τέλος, περνάμε αυτές τις τιμές RGB της $img1$, στα αντίστοιχα pixel της εικόνας $img1\_warped$

Για παράδειγμα, αν περάσουμε την συνάρτηση στην εικόνα $1.png$ (αριστερό μέρος του βουνού), με την ομογραφία H που βρήκαμε ως προς την εικόνα $2.png$ (κεντρικό μέρος βουνού) θα έχουμε:



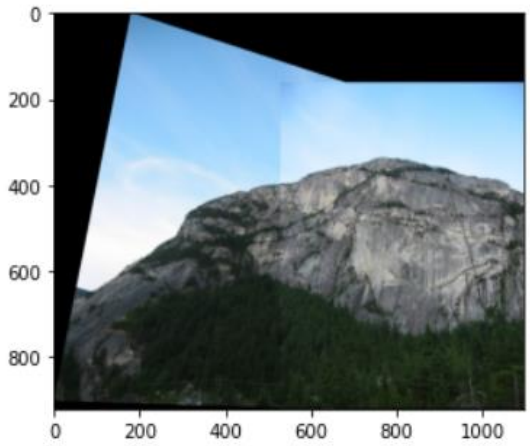
Βλέπουμε ότι η εικόνα έχει μετατοπιστεί στην σωστή κατεύθυνση, με σκοπό να γίνει $stitched$ στο επόμενο βήμα.

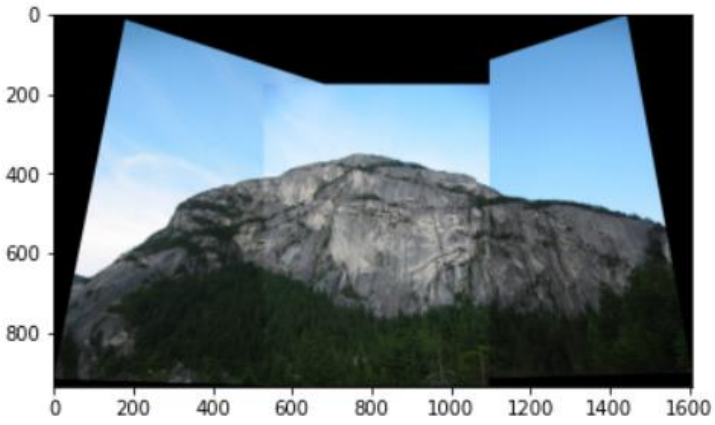
Βήμα 6) Για την συνένωση των εικόνων $img1\_warped$ και $img2$, υλοποιούμε την συνάρτηση $\mathbf{mergeWarpedImages}$, η οποία παίρνει σαν είσοδο και τις συντεταγμένες της πάνω-αριστερά γωνίας της $img1\_warped$ (ως προς το σύστημα συντεταγμένων της $img2$).

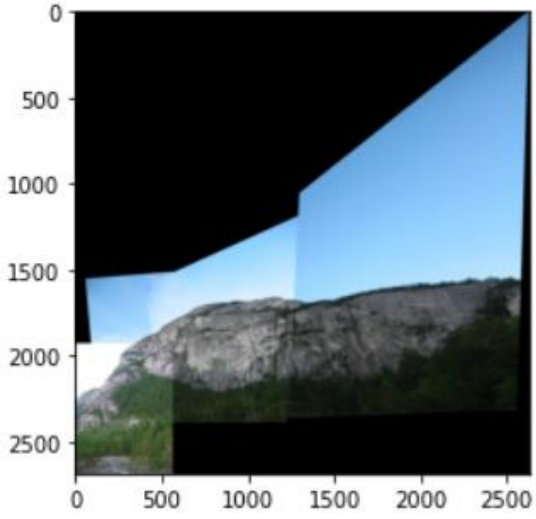
Με τις συντεταγμένες αυτής της γωνίας, και τις διαστάσεις των $img1\_warped$ και $img2$, υπολογίζουμε τις διαστάσεις της τελικής εικόνας $stitchedImage$ που θα περιέχει αυτές τις δύο.

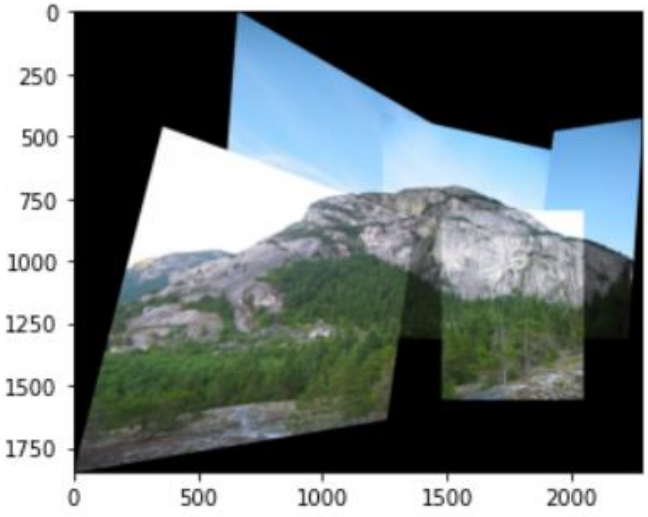
Τέλος, απλά αντιγράφουμε τις δύο εικόνες πάνω στην τελική $stitchedImage$, πρώτα την $img1\_warped$ και μετά την $img2$ (έτσι ώστε να μην επικαλύπτεται με μαύρες περιοχές). Επίσης, στο τελικό αποτέλεσμα, αφαιρούμε τις οποιεσδήποτε μαύρες περιοχές έχουν γίνει padded γύρω από την τελική εικόνα (καθαρά για αισθητικούς λόγους, δεν επηρεάζει το τελικό αποτέλεσμα).

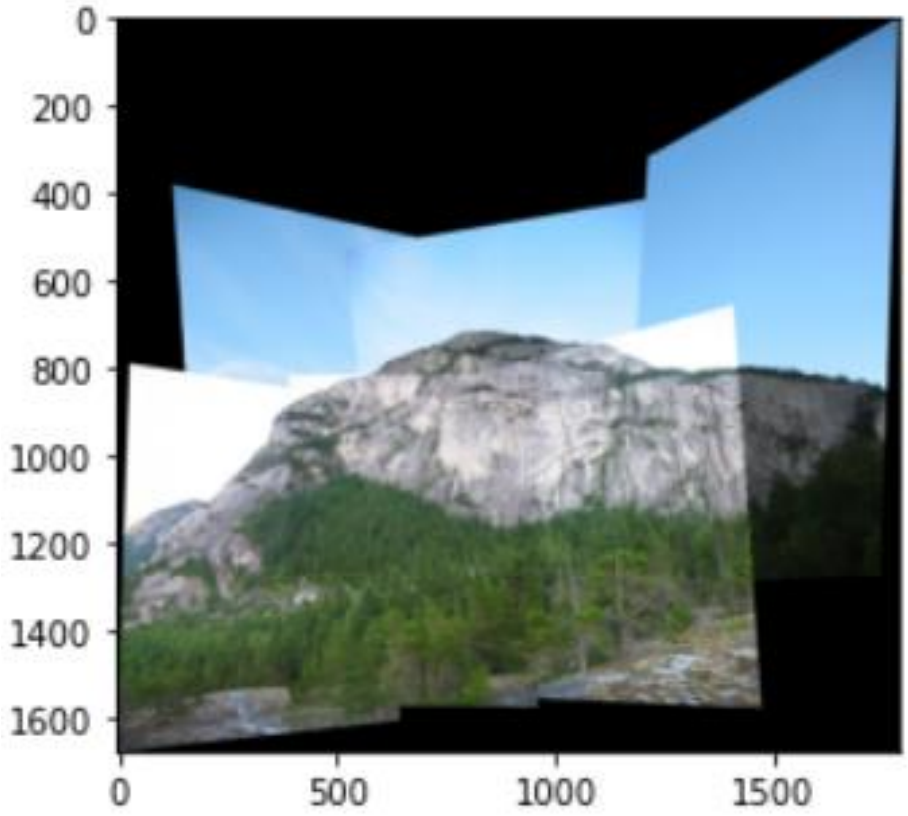
Αφού φτιάξουμε την τελική συνάρτηση ***stitchImages,*** και την εφαρμόσουμε αναδρομικά και στις έξι εικόνες του βουνού, το τελικό αποτέλεσμα σε κάθε βήμα είναι:











Βλέπουμε ότι στην τελευταία εικόνα, έχουμε πλήρη ανακατασκευή του βουνού από τις 6 εικόνες.