Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

18/06/2023

ΟΡΑΣΗ ΥΠΟΛΟΓΙΣΤΩΝ

8ο Εξάμηνο 2022 – 2023

Lab Assignment 2 – Solutions

Ζαρίφης Στέλιος – el20435 Email: el20435@mail.ntua.gr

Αστρεινίδης Ζαφείριος – el19053 Email: el19053@mail.ntua.gr

Contents

Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

Μέρος 1.0: Κάνουμε import τις απαραίτητες βιβλιοθήκες

```
from scipy.stats import multivariate_normal
import cv2
import scipy
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from scipy.ndimage import label,map_coordinates,convolve1d, rank_filter
from numpy import unravel_index
import math
import glob
import base64
from numpy.linalg import norm
import numpy as np
import cv2
import matplotlib.pyplot as plt
import scipy.io
import scipy.stats
import imageio
from scipy.stats import multivariate_normal
from scipy.ndimage import map_coordinates, label
# import cv21_lab2_2_utils as utils
import os
%matplotlib inline
```

Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053
18/06/2023

Μέρος 1: Παρακολούθηση Προσώπου και Χεριών με Χρήση της Μεθόδου Οπτικής Ροής των Lucas-Kanade

1.1. Ανίχνευση Δέρματος Προσώπου και Χεριών

1.1.a. Υπολογισμός χαρακτηριστικών δέρματος από τα δείγματα

1. Φορτώνουμε το αρχείο δεδομένων MATLAB skinSamplesRGB.mat χρησιμοποιώντας τη συνάρτηση loadmat του scipy.io.

2. Αλλάζουμε χρωματικό χώρο, από RGB σε YCbCr μέσω της συνάρτησης cv2.cvtColor.

3. Εξάγουμε τα κανάλια Cb και Cr (Chrominance Blue-Difference & Chrominance Red-Difference) από την εικόνα στον νέο χώρο (YCbCr). Σημειώνουμε γιατί προτιμάμε να επεξεργαστούμε την εικόνα σε αυτόν το χώρο:

    - Cb: Αντιπροσωπεύει τη χρωματική διαφορά μεταξύ της μπλε συνιστώσας και της φωτεινότητας (Y). Το ανθρώπινο δέρμα έχει σημαντική μπλε συνιστώσα στο χρώμα του, η οποία οφείλεται στην ύπαρξη φλεβών από κάτω.

    - Cr: Αντιπροσωπεύει τη χρωματική διαφορά μεταξύ της κόκκινης συνιστώσας και της φωτεινότητας (Y). Το ανθρώπινο δέρμα έχει επίσης μια έντονη κόκκινη συνιστώσα στο χρώμα του, λόγω των αρτηριών αυτήν τη φορά.

4. Υπολογίζουμε τις μέσες τιμές των καναλιών Cb και Cr χρησιμοποιώντας τη συνάρτηση np.mean της βιβλιοθήκης NumPy. Αυτές είναι οι μέσες τιμές χρώματος των μπλε και κόκκινων τόνων που υπάρχουν στα δείγματα του δοθέντος αρχείου.

5. Υπολογίζουμε τον πίνακα συνδιακύμανσης μεταξύ των καναλιών (ύστερα από μετατροπή τους σε διανύσματα) Cb και Cr χρησιμοποιώντας τη συνάρτηση np.cov.

6. Τυπώνουμε τις μέσες τιμές των καναλιών και τη συνδιακύμανσή τους.

```
mat = scipy.io.loadmat('part1 - GreekSignLanguage/skinSamplesRGB.mat')
skin_YCbCr = cv2.cvtColor(mat['skinSamplesRGB'], cv2.COLOR_RGB2YCrCb)
Cb = skin_YCbCr[:,:,1]
Cr = skin_YCbCr[:,:,2]
mu = [np.mean(Cb), np.mean(Cr)]
cov = np.array(np.cov(Cb.flatten(), Cr.flatten()))
print("mu =", mu)
print("cov =", cov)
```

```
mu = [157.0460157126824, 103.27048260381594]
cov = [[ 44.19103128 -11.9310385 ]
 [-11.9310385   11.19574811]]
```

1.1.b. Εύρεση bounding boxes (συνάρτηση `fd()`)

Θα γράψουμε τώρα τη συνάρτηση fd(I, mu, cov) που θα ανιχνεύει περιοχές δέρματος.

Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

18/06/2023

1. Μετατρέπουμε την εικόνα εισόδου I στον χρωματικό χώρο YCrCb όπως εξηγήθηκε προηγουμένως.

2. Απομονώνουμε τα κανάλια Cb και Cr από την εικόνα.

3. Δημιουργούμε έναν πίνακα x ως την παράθεση των διανυσμάτων-στηλών Cb, Cr. Ο πίνακας x είναι διδιάστατος και έχει τη μορφή:

$$x = \begin{bmatrix} | & | \\ Cb & Cr \\ | & | \end{bmatrix} = \begin{bmatrix} Cb_1 & Cr_1 \\ Cb_2 & Cr_2 \\ \vdots & \vdots \\ Cb_n & Cr_n \end{bmatrix}$$

4. Υπολογίζουμε την πιθανότητα P του pixel (x, y) να είναι δέρμα, χρησιμοποιώντας την πολυμεταβλητή κανονική κατανομή multivariate_normal.pdf με χαρακτηριστικά (μέση τιμή κα συνδιακύμανση) που εξάγαμε από τα δεδομένα του αρχείου mat.

5. Κατωφλιώνουμε την εικόνα βάσει των πιθανοτήτων που υπολογίστηκαν για κάθε pixel. Συγκεκριμένα, θεωρούμε ότι ένα pixel δείχνει δέρμα αν η πιθανότητα που υπολογίσαμε για αυτό είναι μεγαλύτερη του 0.2 της μέγιστης τιμής που λαμβάνει η πιθανότητα P. Καταλήγουμε, λοιπόν, με μια δυαδική εικόνα που έχει τιμή 1 στα pixels που θεωρούμε ότι υπάρχει δέρμα και 0 στα υπόλοιπα.

6. Εφαρμόζουμε ένα opening φίλτρο (cv2.morphologyEx(skin, cv2.MORPH_OPEN, kernel)) στη δυαδική εικόνα χρησιμοποιώντας έναν μικρό πυρήνα, ώστε να αφαιρέσουμε τον θόρυβο και να ομαλοποήσουμε τις "τραχείες" επιφάνειες.

7. Στην προκύπτουσα εικόνα, εφαρμόζουμε ένα closing φίλτρο (cv2.morphologyEx(opening, cv2.MORPH_CLOSE, kernel))χρησιμοποιώντας έναν μεγάλο πυρήνα για να γεμίσετε τις τρύπες (κοντά στα δάχτυλα για παραδειγμα) και να ενοποιήσουμε γειτονικές περιοχές δέρματος. Ακόμα, τυπώνουμε την τελευταία εικόνα.

8. Κάνουμε labeling των συνεκτικών περιοχών δέρματος που βρήκαμε για να γνωρίζουμε την αντιστοίχιση των συνόλων αυτών με τα ανθρώπινα μέλη που παρακολουθούμε (δεξί χέρι, αριστερό χέρι και κεφάλι).

9. Χρησιμοποιώντας τις τελευταίες περιοχές, υπολογίζουμε τα άκρα αυτών (άνω, κάτω, δεξιά και αριστερά) τα οποία μας δίνουν τις συντεταγμένες του bounding box (τις τέσσερεις γωνίες). Οι τελευταίες επιστρέφονται σε μια λίστα.

Συνολικά, λοιπόν, η συνάρτηση fd() εκτελεί την ανίχνευση δέρματος:

- μετατρέποντας την εικόνα εισόδου στον χώρο YCrCb,
- υπολογίζοντας ύστερα την πιθανότητα κάθε pixel να είναι δέρμα με βάση τις χαρακτηριστικές τιμές που υπολογίσαμε από τα δείγματα,
- κατωφλιώνοντας αυτές τις πιθανότητες,

- εφαρμόζωντας μορφολογικά φίλτρα για την αφαίρεση του θορύβου και το "γέμισμα" των οπών,
- επισημειώνοντας τις συνεκτικές περιοχές δέρματος και εξαγάγοντας τις συντεταγμένες του πλαισίου τους.

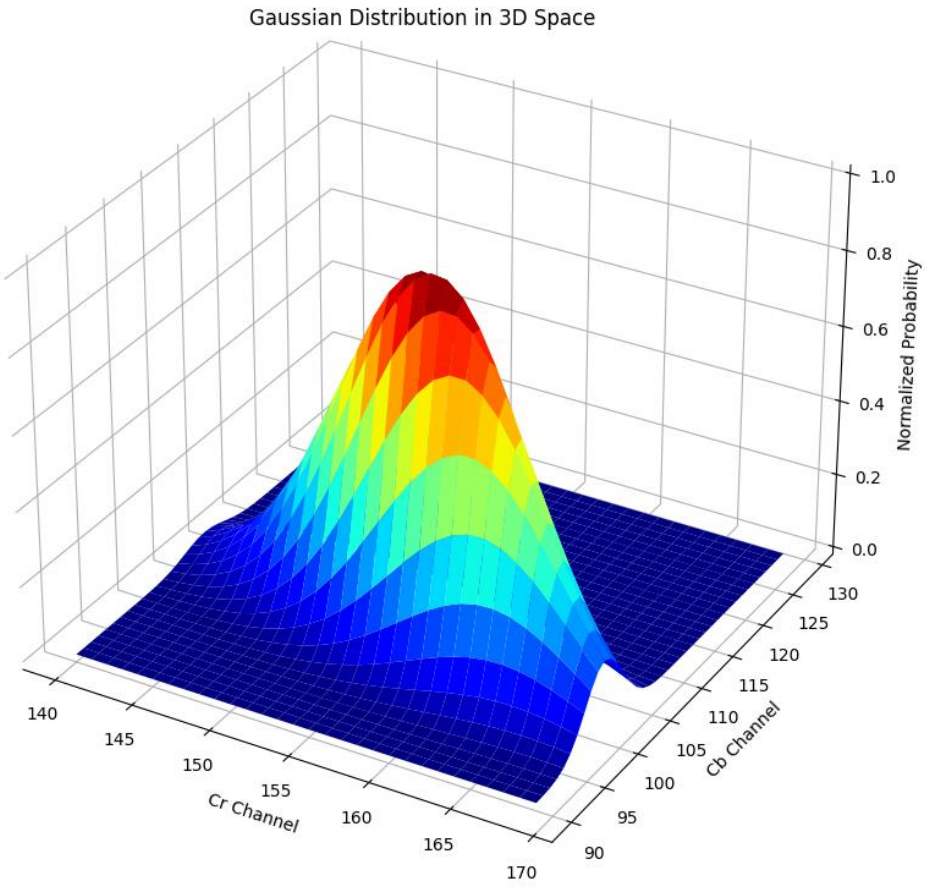
```
def fd(I,mu,cov):
    IYCbCr = cv2.cvtColor(I, cv2.COLOR_RGB2YCrCb)
    Cb = IYCbCr[:,:,1]
    Cr = IYCbCr[:,:,2]

    x = np.array([Cb, Cr]).T
    P = multivariate_normal.pdf(x , mu, cov)
    P = P.T

    xTemp, yTemp = np.mgrid[140:170:1, 90:130:1]
    pos = np.dstack((xTemp, yTemp))
    rv = multivariate_normal(mu, cov)
    from mpl_toolkits.mplot3d import Axes3D
    fig = plt.figure(figsize=(25,10))
    ax = fig.add_subplot(1, 2, 1, projection='3d')
    ax.xaxis.set_pane_color((1.0, 1.0, 1.0, 0.0))
    ax.yaxis.set_pane_color((1.0, 1.0, 1.0, 0.0))
    ax.zaxis.set_pane_color((1.0, 1.0, 1.0, 0.0))
    # ax.plot_wireframe(xTemp, yTemp, rv.pdf(pos)/np.max(rv.pdf(pos)),
color='blue')#)cmap=cm.PuOr)
    ax.plot_surface(xTemp, yTemp, rv.pdf(pos)/np.max(rv.pdf(pos)),
cmap='jet', edgecolor='none')
    plt.title('Gaussian Distribution in 3D Space', y=1)
    plt.xlabel('Cr Channel')
    plt.ylabel('Cb Channel')
    ax.set_zlabel('Normalized Probability')
    plt.show()


    # print(P)
    _, skin = cv2.threshold(P, 0.0001, 0.25, cv2.THRESH_BINARY)
    # print(skin)
    skin = (P/np.max(P) > 0.15).astype(int)
    skin = np.uint8(skin)
    # print(skin)
    kernel = np.ones((3,3),np.uint8)
    opening = cv2.morphologyEx(skin, cv2.MORPH_OPEN, kernel)
    kernel = np.ones((22,22),np.uint8)
    closing = cv2.morphologyEx(opening, cv2.MORPH_CLOSE, kernel)
    plt.imshow(closing,cmap='gray')

    labeled_array, num_features = label(closing)
    # print(num_features)
    box = []
```

```
    for i in range(num_features):
        x,y = np.where(labeled_array == i+1)
        x = np.sort(x)
        y = np.sort(y)
        width = x[-1]-x[0]
        height = y[-1] - y[0]
        box.append([y[0] - 10, x[0] - 10,height + 10,width + 10])

    return box
```

Τυπώνουμε κάτω τα ζητούμενα:

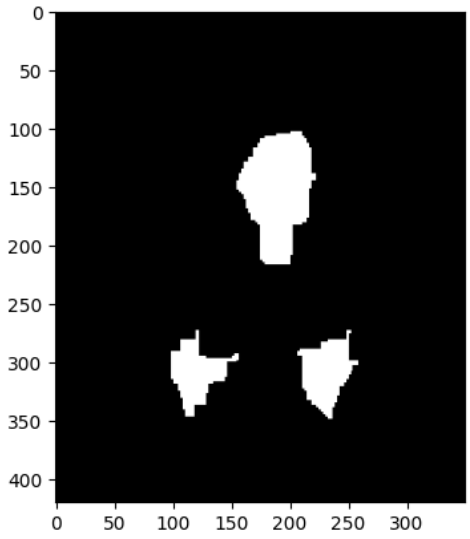
- Η 3D κατανομή gauss που υπολογίσαμε από το αρχείο mat
- η δυαδική εικόνα μετά την κατωφλίωση

```
I = cv2.imread("part1 - GreekSignLanguage/1.png")
I = cv2.cvtColor(I, cv2.COLOR_BGR2RGB)
tmp = fd(I,mu,cov)
```



Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

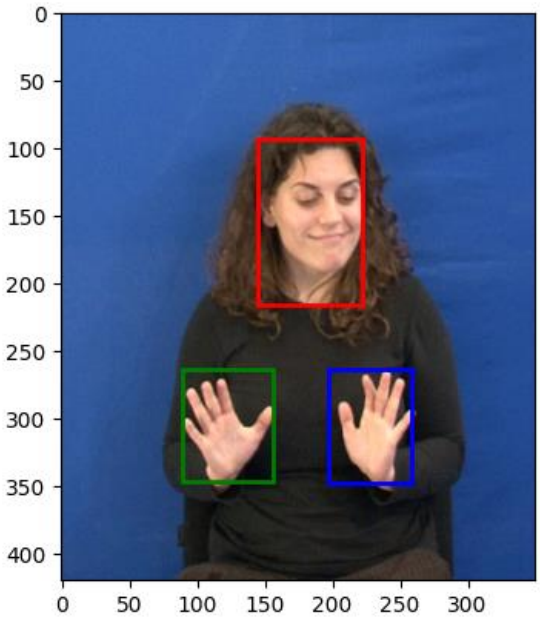
18/06/2023



Και εδώ τυπώνουμε τα bounding boxes πάνω στο πρώτο frame

```
figure, ax = plt.subplots(1)
colors = ['r', 'g', 'b']
for i in range(3):
    ax.add_patch(patches.Rectangle((tmp[i][0], tmp[i][1]), tmp[i][2],
tmp[i][3], edgecolor=colors[i], facecolor="none",linewidth=2))
    print(colors[i],"=", tmp[i][0], tmp[i][1], tmp[i][2], tmp[i][3])
ax.imshow(I)
plt.show()
```

```
r = 145 93 77 123
g = 89 263 67 83
b = 197 263 61 85
```



Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

1.2. Παρακολούθηση Προσώπου και Χεριών

1.2.0. Η συνάρτηση `cv2.goodFeaturesToTrack`

Ορίζουμε τις παραμέτρους για τη συνάρτηση cv2.goodFeaturesToTrack:

- maximum number of corners: 30
- quality level: 0.2 (μεγάλη τιμή συνεπάγεται λίγα σημεία - καλύτερες γωνίες)
- minDistance: 2
- blockSize: 5

Και υπολογίζουμε τα σημεία ενδιαφέροντος μέσα στο πλαίσιο. Παρακάτω, φαίνεται η διαδικασία για το δεξί χέρι.

```
# paramters for corner detection
feature_params = dict( maxCorners = 30,
                       qualityLevel = 0.2,
                       minDistance = 2,
                       blockSize = 5)
gray = cv2.cvtColor(I2, cv2.COLOR_RGB2GRAY)
p0 = cv2.goodFeaturesToTrack(gray, mask = None, **feature_params)

fig, ax = plt.subplots(figsize=(10,10))
ax.imshow(I2, cmap=plt.cm.gray)
ax.plot(p0[:,0, 0], p0[:,0, 1], color='red', marker='o',
        linestyle='None', markersize=5)
plt.show()
```

1.2.1 Υλοποίηση του Αλγόριθμου των Lucas-Kanade

Στον ακόλουθο κώδικα, υλοποιούμε τον αλγόριθμο Lucas-Kanade για να εκτιμήσουμε την οπτική ροή στις εικόνες.

Η συνάρτηση lucas_kanade δέχεται ως είσοδο δύο διαδοχικά frames I1 και I2, τα σημεία ενδιαφέροντος (features - κυρίως γωνίες), τις παραμέτρους rho και epsilon, καθώς και την τα αρχικοποιημένα πεδία οπτικής ροής dx0 και dy0.

Στην αρχή της διαδικασίας, αρχικοποιούμε τις εκτιμήσεις της οπτικής ροής dx_i και dy_i με τις αρχικές τιμές που εισάγουμε από τα arguments.

Ορίζουμε ένα πλέγμα για κάθε ροή, για την εικόνα χρησιμοποιώντας τη μέθοδο numpy.meshgrid. Αυτά θα χρησιμοποιηθούν αργότερα για την παρεμβολή.

Οι εικόνες I1 και I2 κανονικοποιούνται στο διάστημα [0, 1]. Έτσι εξομαλύνουμε κάπως το θόρυβο (η κλίμακά του διαφέρει λιγότερο από των υπολοίπων χαρακτηριστικών της εικόνας).

Κατασκευάζουμε έναν 2D gaussian kernel με τυπική απόκλιση rho. Αυτός ο πυρήνας χρησιμοποιείται για περαίτερω εξομάλυνση.

Υπολογίζουμε τις κλίσεις της πρώτης εικόνας I1 (dyBefore και dxBefore) με τη συνάρτηση np.gradient.

Μετά από αυτήν την προετοιμασία, έχουμε έναν βρόχο στον οποίο συμβαίνουν τα ακόλουθα βήματα:

- Η προηγούμενη εικόνα IBefore υπολογίζεται με παρεμβολή της I1 και τις τρέχουσες εκτιμήσεις των dx_i και dy_i. Αυτό πραγματοποιείται με χρήση της map_coordinates.
- Οι μερικές παράγωγοι (A1, A2) υπολογίζονται με παρεμβολή των dxBefore και dyBefore χρησιμοποιώντας τις τρέχουσες εκτιμήσεις των dx_i και dy_i.
- Η εικόνα σφάλματος E υπολογίζεται με την αφαίρεση του IBefore από το I2.
- Τα στοιχεία του συστήματος γραμμικών εξισώσεων της εξίσωσης υπολογίζονται με πράξεις συνέλιξης με τον γκαουσιανό πυρήνα gauss2D.
- Οι ενημερώσεις οπτικής ροής ux και uy υπολογίζονται με την επίλυση του συστήματος γραμμικών εξισώσεων για κάθε εικονοστοιχείο χρησιμοποιώντας τα υπολογισμένα στοιχεία.
- Εάν παρέχονται χαρακτηριστικά, οι εκτιμήσεις οπτικής ροής για τα σημεία των χαρακτηριστικών ενημερώνονται με τις υπολογισμένες ενημερώσεις.

Τέλος, η συνάρτηση επιστρέφει τις ενημερωμένες εκτιμήσεις οπτικής ροής dx_i και dy_i.

Ο αλγόριθμος Lucas-Kanade εκτιμά το πεδίο οπτικής ροής βελτιώνοντας επαναληπτικά τις αρχικές εκτιμήσεις χρησιμοποιώντας πληροφορίες τοπικής κλίσης και ελαχιστοποιώντας το τετραγωνικό σφάλμα μεταξύ διαδοχικών καρέ. Υποθέτει ότι η κίνηση μεταξύ των καρέ είναι σταθερή εντός ενός μικρού παραθύρου γύρω από κάθε εικονοστοιχείο. Ο αλγόριθμος συγκλίνει σε μια καλύτερη εκτίμηση του πεδίου οπτικής ροής με κάθε επανάληψη.

Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

18/06/2023

Για N frames $I_n(\mathbf{x}), n = 1, 2, ..., N, \mathbf{x} = (x, y)$ έχουμε το πεδίο ροής $-\mathbf{d}, \mathbf{d}(\mathbf{x}) = (d_x, d_y)$ για το οποίο ισχύει:

$$I_n(\mathbf{x}) \approx I_{n-1}(\mathbf{x}+\mathbf{d})$$

Θεωρούμε το συναρτησιοειδές:

$$\mathbf{J}_\mathbf{x}(\mathbf{d}) = \int_{\mathbf{x}' \in \mathbb{R}^2} G_\rho(\mathbf{x} - \mathbf{x}')[I_n(\mathbf{x}') - I_{n-1}(\mathbf{x}' + \mathbf{d})]^2 \, \mathrm{d}\mathbf{x}, G_\rho : gaussian(\sigma^2 = \rho)$$

Θεωρούμε την εκτίμηση $\mathbf{d}_i$ για το $\mathbf{d}$ και σε κάθε επανάληψη του αλγορίθμου τη βελτιώνουμε κατά $\mathbf{d}_{i+1} = \mathbf{d}_i + \mathbf{u}$

Αν αναπτύξουμε κατά Taylor την $I_{n-1}(\mathbf{x} + \mathbf{d}) = I_{n-1}(\mathbf{x} + \mathbf{d}_i + \mathbf{u})$ γύρω από το $\mathbf{x} + \mathbf{d}_i$ έχουμε:

$$I_{n-1}(\mathbf{x} + \mathbf{d}) = I_{n-1}(\mathbf{x} + \mathbf{d}_i) + \nabla I_{n-1}(\mathbf{x} + \mathbf{d}_i)^T \mathbf{u}$$

Η λύση των ελαχίστων τετραγώνων, για το $\mathbf{J}_\mathbf{x}$ θα μας δώσει:

$$\mathbf{u}(\mathbf{x}) = \begin{bmatrix} (G_\rho * A_1^2)(\mathbf{x}) + \epsilon & (G_\rho * A_1 A_2)(\mathbf{x}) \\ (G_\rho * A_1 A_2)(\mathbf{x}) & (G_\rho * A_2^2)(\mathbf{x}) + \epsilon \end{bmatrix}^{-1} \cdot \begin{bmatrix} (G_\rho * (A_1 E))(\mathbf{x}) \\ (G_\rho * (A_2 E))(\mathbf{x}) \end{bmatrix}$$

όπου συμβολίσαμε:

$$A(\mathbf{x}) = \begin{bmatrix} A_1(\mathbf{x}) & A_2(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \frac{\partial I_{n-1}(\mathbf{x}+\mathbf{d}_i)}{\partial x} & \frac{\partial I_{n-1}(\mathbf{x}+\mathbf{d}_i)}{\partial y} \end{bmatrix}, E_{(\mathbf{x})} = I_n(\mathbf{x}) - I_{n-1}(\mathbf{x} + \mathbf{d}_i)$$

```python
def lucas_kanade(I1, I2, features, rho, epsilon, dx0, dy0):
    dx_i = dx0.astype(np.float64)
    dy_i = dy0.astype(np.float64)

    ############ Create meshgrid of the image ################
    # Create x and y coordinate grids
    xAxis = np.linspace(0,I1.shape[1]-1,I1.shape[1])
    yAxis = np.linspace(0,I1.shape[0]-1,I1.shape[0])
    x0,y0 = np.meshgrid(xAxis,yAxis)

    # Normalize the images
    I1 = I1.astype(np.float64)/255
    I2 = I2.astype(np.float64)/255

    # Calculate the size of the Gaussian filter based on rho
    n = int(2*np.ceil(3*rho)+1)
    gauss1D = cv2.getGaussianKernel(n, rho)
    gauss2D = gauss1D @ gauss1D.T

    # Calculate the gradients of I1
    dyBefore, dxBefore = np.gradient(I1)
```

```
    for i in range(30):
        # Warp I1 using current displacement estimates
        IBefore = map_coordinates(I1,[np.ravel(y0 + dy_i),np.ravel(x0 +
dx_i)], order=1).reshape(I1.shape)
        # Warp partial derivatives with respect to x
        A1 = map_coordinates(dxBefore,[np.ravel(y0 + dy_i), np.ravel(x0 +
dx_i)], order=1).reshape(I1.shape)
        # Warp partial derivatives with respect to y
        A2 = map_coordinates(dyBefore,[np.ravel(y0 + dy_i), np.ravel(x0 +
dx_i)], order=1).reshape(I1.shape)

        # Calculate the error between I2 and the warped I1
        E = I2 - IBefore

        # Compute elements of the Lucas-Kanade matrix
        a11 = cv2.filter2D(A1**2,-1,gauss2D)+epsilon
        a21 = a12 = cv2.filter2D(A1*A2,-1,gauss2D)
        a22 = cv2.filter2D(A2**2,-1,gauss2D)+epsilon

        # Compute elements of the right-hand side vector
        b1 = cv2.filter2D(A1*E,-1,gauss2D)
        b2 = cv2.filter2D(A2*E,-1,gauss2D)

        # Compute determinant of the Lucas-Kanade matrix
        det = (a11*a22 - a12*a21)

        # Compute displacement updates
        ux = (a22*b1 - a12*b2) / det
        uy = (a11*b2 - a21*b1) / det

        # Update dx_i and dy_i for feature points
        if features is not None:
            for x,y in features[:,0,:].astype(int):
                dx_i[y][x] = dx_i[y][x] + ux[y][x]
                dy_i[y][x] = dy_i[y][x] + uy[y][x]

    return [dx_i,dy_i]
```

Υλοποιούμε τη συνάρτηση `displ` που υπολογίζει τη μέση μετατόπιση στις διευθύνσεις x και y, φιλτράροντας τις ασήμαντες μετατοπίσεις.

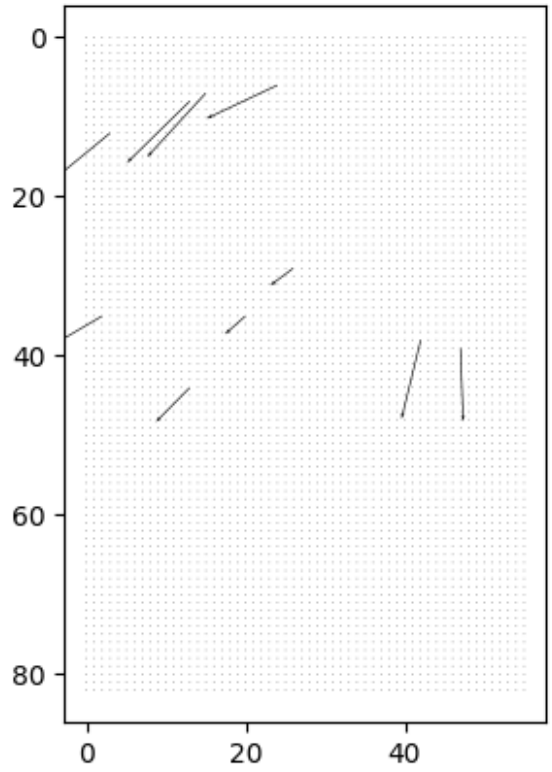
```
def displ(d_x, d_y):
    threshold = 0.5

    # Flatten the input arrays
    dx = d_x.flatten()
    dy = d_y.flatten()

    # Calculate the squared magnitude of displacement vectors
```

```
    d = dx**2 + dy**2

    # Create a binary mask indicating which displacements satisfy the
threshold condition
    cond = (d >= threshold*np.max(d)).astype(np.uint8).reshape(d_x.shape)

    # Apply the mask to the displacement arrays
    dx = cond*d_x
    dy = cond*d_y

    # Calculate the mean displacement values based on the masked arrays
    dx_mean = np.sum(dx)/np.sum(cond)
    dy_mean = np.sum(dy)/np.sum(cond)

    return dx_mean, dy_mean
```

Υπολογίζουμε την οπτική ροή με τον αλγόριθμο Lucas-Kanade για μια μετατόπιση 5 pixels και σχεδιάζουμε τα διανύσματα κίνησης χρησιμοποιώντας το quiver plot.

```
feature_params = dict(maxCorners=20, qualityLevel=0.2, minDistance=2,
blockSize=5)

# Read the image and convert it to RGB color space
img = cv2.imread('part1 - GreekSignLanguage/1.png')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Extract regions of interest from the image
I1 = img[272:272+83, 93:93+56]
I2 = img[272-5:272+83-5, 93:93+56]

# Convert the regions of interest to grayscale
I1gray = cv2.cvtColor(I1, cv2.COLOR_RGB2GRAY)
I2gray = cv2.cvtColor(I2, cv2.COLOR_RGB2GRAY)

# Apply goodFeaturesToTrack to find corner points in I2
p0 = cv2.goodFeaturesToTrack(I2gray, mask=None, **feature_params)

# Compute the optical flow using Lucas-Kanade method
dx, dy = lucas_kanade(I1gray, I2gray, p0, 5, 0.1, np.zeros_like(I1gray),
np.zeros_like(I1gray))

# Plot the motion vectors using quiver plot
plt.gca().invert_yaxis()
plt.gca().set_aspect('equal')
plt.quiver(-dx, -dy, angles='xy', scale=20)
plt.show()

# Calculate and print the mean displacement in x and y directions
print(displ(dx, dy))
```

Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053



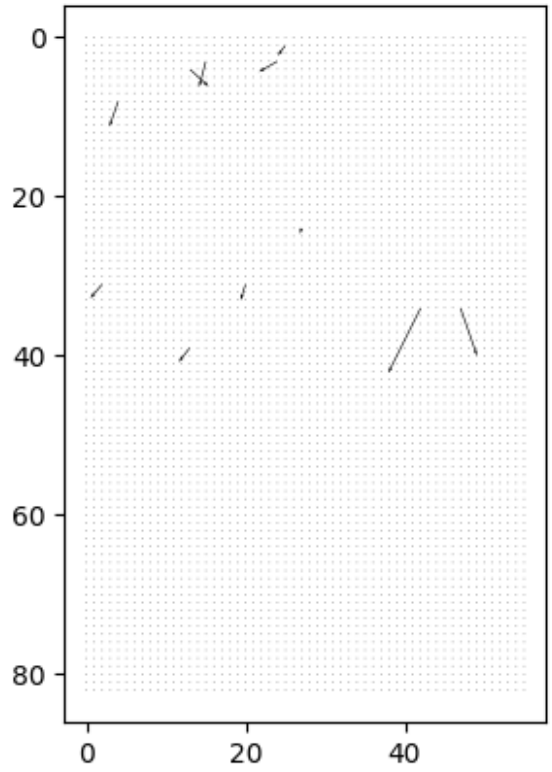

```
(4.493499600214562, -3.099186094834885)
```

Υπολογίζουμε την οπτική ροή με τον αλγόριθμο Lucas-Kanade για τα πρώτα 2 frames και σχεδιάζουμε τα διανύσματα κίνησης χρησιμοποιώντας το quiver plot.

```
# Run Lucas Kanade for first 2 frames
I1 = cv2.imread('part1 - GreekSignLanguage/1.png')  # Read the first image
I2 = cv2.imread('part1 - GreekSignLanguage/2.png')  # Read the second image

I1 = cv2.cvtColor(I1, cv2.COLOR_BGR2GRAY)[272:272+83, 93:93+56]  # Convert
first image to grayscale and extract a region of interest
I2 = cv2.cvtColor(I2, cv2.COLOR_BGR2GRAY)[272:272+83, 93:93+56]  # Convert
second image to grayscale and extract a region of interest

p0 = cv2.goodFeaturesToTrack(I2, mask=None, **feature_params)  # Use the
goodFeaturesToTrack function to find corner points in the second image

dx, dy = lucas_kanade(I1, I2, p0, 1, 0.1, np.zeros_like(I1),
np.zeros_like(I1))  # Compute the optical flow using Lucas-Kanade method with
specified parameters

plt.gca().invert_yaxis()  # Invert the y-axis to match the image coordinate
system
plt.gca().set_aspect('equal')  # Set equal aspect ratio for the plot
plt.quiver(-dx, -dy, angles='xy', scale=30)  # Create a quiver plot to
```

Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053


```
visualize the motion vectors
plt.show()  # Display the plot
```



Εφαρμόζουμε τον αλγόριθμο Lucas-Kanade για την οπτική ροή σε όλα τα διαδοχικά ζεύγη frames και υπολογίζουμε τις μετατοπίσεις των σημείων ενδιαφέροντος για την παρακολούθηση κίνησης. Οι θέσεις των επόμενων bounding boxes ενημερώνονται με βάση αυτές τις μετατοπίσεις.

Ο κώδικας επεξεργάζεται την ακολουθία frames. Αρχικοποιεί τις συντεταγμένες και τις διαστάσεις για τις περιοχές ενδιαφέροντος, καθώς και τις παραμέτρους rho και epsilon. Στη συνέχεια για κάθε ζεύγος frames εκτελεί τα ακόλουθα:

1. Φόρτωση του τρέχοντος και του επόμενου frame.
2. Εξαγωγή των περιοχών ενδιαφέροντος.
3. Εξαγωγή οπτικής ροής με τον αλγόριθμο Lucas-Kanade για την παρακολούθηση της κίνησης κάθε περιοχής.
4. Ενημέρωση των θέσεων των περιοχών βάσει των υπολογισμένων μετατοπίσεων.

```
xRightHand = 93
yRightHand = 272
widthRightHand = 56
heightRightHand = 83

xLeftHand = 201
yLeftHand = 270
widthLeftHand = 56
heightLeftHand = 83
```

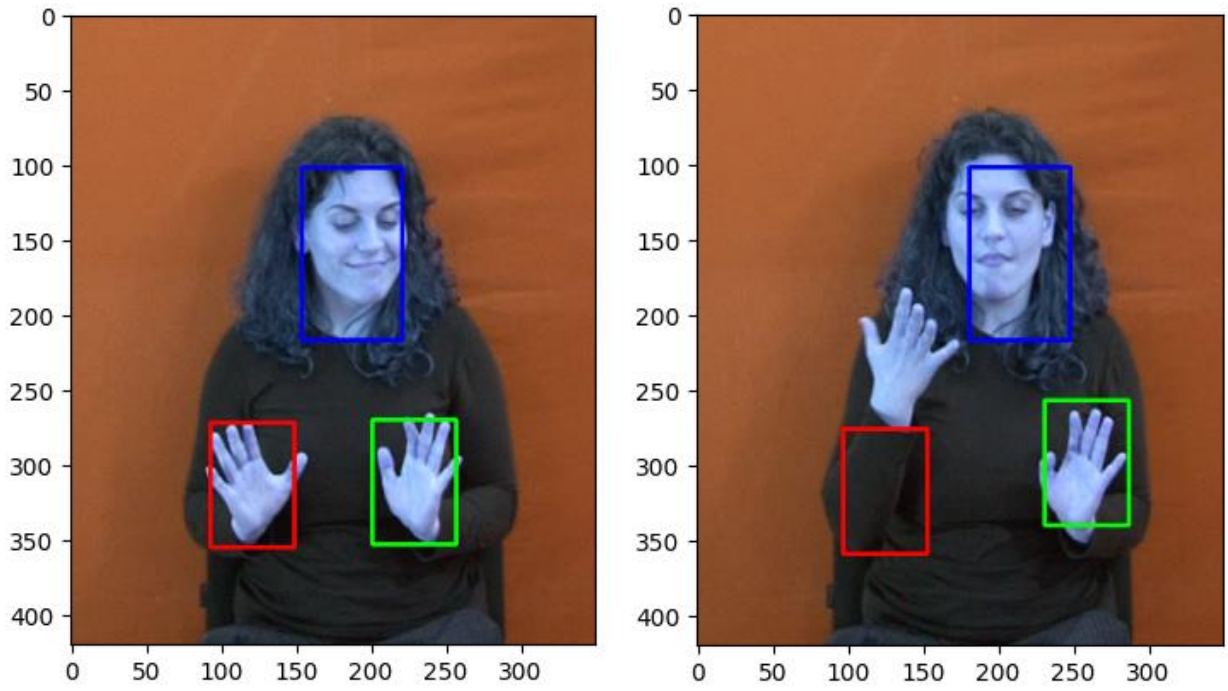
```
xHead = 154
yHead = 102
widthHead = 67
heightHead = 115

rho = 3
epsilon = 0.15

feature_params = dict(
    maxCorners=20,
    qualityLevel=0.2,
    minDistance=1,
    blockSize=5
)

frames = []  # List to store the processed frames

for i in range(1, 65):
    path1 = 'part1 - GreekSignLanguage/' + str(i) + '.png'
    I1_bgr = cv2.imread(path1)
    I1_gray = cv2.cvtColor(I1_bgr, cv2.COLOR_BGR2GRAY)
    I1 = I1_gray.astype(np.float) / 255

    path2 = 'part1 - GreekSignLanguage/' + str(i + 1) + '.png'
    I2_bgr = cv2.imread(path2)
    I2_gray = cv2.cvtColor(I2_bgr, cv2.COLOR_BGR2GRAY)
    I2 = I2_gray.astype(np.float) / 255

    # Extract the regions of interest for the right hand
    det1RightHand = I1_gray[yRightHand-10:(yRightHand+heightRightHand-10),
                            xRightHand+10:(xRightHand+widthRightHand+10)]
    det2RightHand = I2_gray[yRightHand-10:(yRightHand+heightRightHand-10),
                            xRightHand+10:(xRightHand+widthRightHand+10)]

    # Extract the regions of interest for the left hand
    det1LeftHand = I1_gray[yLeftHand-10:(yLeftHand+heightLeftHand-10),
                           xLeftHand+10:(xLeftHand+widthLeftHand+10)]
    det2LeftHand = I2_gray[yLeftHand-10:(yLeftHand+heightLeftHand-10),
                           xLeftHand+10:(xLeftHand+widthLeftHand+10)]

    # Extract the regions of interest for the head
    det1Head = I1_gray[yHead-10:(yHead+heightHead-10),
                       xHead+10:(xHead+widthHead+10)]
    det2Head = I2_gray[yHead-10:(yHead+heightHead-10),
                       xHead+10:(xHead+widthHead+10)]

    # Apply Lucas-Kanade optical flow to track the motion of the right hand
    p0RightHand = cv2.goodFeaturesToTrack(det2RightHand, mask=None,
```

```
**feature_params)
    d_xRightHand, d_yRightHand = lucas_kanade(det1RightHand, det2RightHand,
                                              p0RightHand, rho, epsilon,
                                              np.zeros(det1RightHand.shape),
                                              np.zeros(det2RightHand.shape))
    # Apply Lucas-Kanade optical flow to track the motion of the left hand
    p0LeftHand = cv2.goodFeaturesToTrack(det2LeftHand, mask=None,
**feature_params)
    d_xLeftHand, d_yLeftHand = lucas_kanade(det1LeftHand, det2LeftHand,
                                            p0LeftHand, rho, epsilon,
                                            np.zeros(det1LeftHand.shape),
                                            np.zeros(det2LeftHand.shape))
    # Apply Lucas-Kanade optical flow to track the motion of the head
    p0Head = cv2.goodFeaturesToTrack(det2Head, mask=None, **feature_params)
    d_xHead, d_yHead = lucas_kanade(det1Head, det2Head, p0Head, rho, epsilon,
                                    np.zeros(det1Head.shape),
                                    np.zeros(det2Head.shape))

    # Convert the optical flow displacements to integer values
    displ_xRightHand, displ_yRightHand = displ(d_xRightHand, d_yRightHand)
    displ_xRightHand = int(displ_xRightHand)
    displ_yRightHand = int(displ_yRightHand)

    displ_xLeftHand, displ_yLeftHand = displ(d_xLeftHand, d_yLeftHand)
    displ_xLeftHand = int(displ_xLeftHand)
    displ_yLeftHand = int(displ_yLeftHand)

    displ_xHead, displ_yHead = displ(d_xHead, d_yHead)
    displ_xHead = int(displ_xHead)
    displ_yHead = int(displ_yHead)

    plt.figure()

    # Draw rectangles around the updated positions of the right hand, left
hand, and head
    cv2.rectangle(I2_bgr, (xRightHand, yRightHand),
                  (xRightHand + widthRightHand, yRightHand +
heightRightHand),
                  (255, 0, 0), 2)
    cv2.rectangle(I2_bgr, (xLeftHand, yLeftHand),
                  (xLeftHand + widthLeftHand, yLeftHand + heightLeftHand),
                  (0, 255, 0), 2)
    cv2.rectangle(I2_bgr, (xHead, yHead), (xHead + widthHead, yHead +
heightHead),
                  (0, 0, 255), 2)

    plt.imshow(I2_bgr)
    ###
    plt.savefig('frame.png')  # Save the plot as a PNG image
```

```
    frames.append(imageio.imread('frame.png'))  # Append the frame to the
frames list
    ###

    # Update the positions of the right hand, left hand, and head based on
the displacements
    xRightHand = xRightHand - displ_xRightHand
    yRightHand = yRightHand - displ_yRightHand

    xLeftHand = xLeftHand - displ_xLeftHand
    yLeftHand = yLeftHand - displ_yLeftHand

    xHead = xHead - displ_xHead
    yHead = yHead - displ_yHead

###
output_path = "animation_rho3_epsilon015.gif"
imageio.mimsave(output_path, frames, fps=10)  # Adjust the fps value as
needed
print(f"GIF saved at: {output_path}")
###
```

```
GIF saved at: animation_rho3_epsilon015.gif
```

Βλέπουμε το αρχικό και το τελικό frame με τα bounding boxes. Επειδή το δεξί χέρι κάνει πιο απότομες κινήσεις, ο αλγόριθμος αποτυγχάνει να το παρακολουθήσει ως το τέλος. Όλα τα frames υπάρχουν στο αρχείο animation_rho3_epsilon015.mp4.



Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

18/06/2023

Χρησιμοποιήσαμε το παρακάτω cell για να τρέξουμε πολλές φορές τον αλγόριθμο lucas kanade και να φτιάξουμε gifs με τα αποτελέσματα για να τα συγκρίνουμε.

```
seeRhoEpsilonDifferences = False
if seeRhoEpsilonDifferences:
    x = 93
    y = 272
    width = 56
    height = 83

    rhos = [0.1, 3, 10]
    epsilons = [0.001, 0.1, 10]
    feature_params = dict( maxCorners = 20,
                           qualityLevel = 0.2,
                           minDistance = 1,
                           blockSize = 5)
    for rho in rhos:
        for epsilon in epsilons:
            x = 93
            y = 272
            width = 56
            height = 83
            feature_params = dict( maxCorners = 20,
                           qualityLevel = 0.2,
                           minDistance = 1,
                           blockSize = 5)
            frames = []
            for i in range(1,65):
                path1 = 'part1 - GreekSignLanguage/' + str(i) + '.png'
                I1_bgr = cv2.imread(path1)
                I1_gray = cv2.cvtColor(I1_bgr,cv2.COLOR_BGR2GRAY)
                I1 = I1_gray.astype(np.float)/255


                path2 = 'part1 - GreekSignLanguage/' + str(i+1) + '.png'
                I2_bgr = cv2.imread(path2)
                I2_gray = cv2.cvtColor(I2_bgr,cv2.COLOR_BGR2GRAY)
                I2 = I2_gray.astype(np.float)/255

                det1 = I1_gray[y:(y+height+20),x:(x+width+20)]
                det2 = I2_gray[y:(y+height+20),x:(x+width+20)]

                p0 = cv2.goodFeaturesToTrack(det2, mask = None,
**feature_params)

                d_x, d_y = lucas_kanade(det1, det2, p0 ,rho, epsilon,
np.zeros(det1.shape), np.zeros(det2.shape))

                displ_x,displ_y = displ(d_x, d_y)
```

```
                displ_x = int(displ_x)
                displ_y = int(displ_y)


                plt.figure()
                cv2.rectangle(I2_bgr, (x, y), (x+width, y+height), (255,0,0)
,2)
                plt.imshow(I2_bgr)
                ###
                plt.savefig('frame.png')  # Save the plot as a PNG image
                frames.append(imageio.imread('frame.png'))  # Append the
frame to the frames list
                ###
                x = x - displ_x
                y = y - displ_y

            ###
            output_path = "animation_rho" + str(int(rho*10)) +
"Tenths_epsilon" + str(int(epsilon*1000)) + "Milis.gif"
            imageio.mimsave(output_path, frames, fps=10)  # Adjust the fps
value as needed
            print(f"GIF saved at: {output_path}")
            ###
```

1.2.3. Πολυ-Κλιμακωτός Υπολογισμός Οπτικής Ροής

Ο αλγόριθμος multi-scale Lucas-Kanade βελτιώνει αλγόριθμο Lucas-Kanade για την οπτική ροή ενσωματώνοντας πολλαπλές κλίμακες για την ακολουθία εικόνων. Σε αντίθεση με τον απλό Lucas-Kanade, δύναται να χειριστεί μεγάλες μετατοπίσεις, αφού μια μεγάλη μετατόπιση σε μικρή κλίμακα είναι μικρή μετατώπιση σε κάποια μεγαλύτερη.

Ο αλγόριθμος ξεκινά με την κλίμακα της ακολουθίας εικόνων με την υψηλότερη ανάλυση. Εφαρμόζεται φιλτράρισμα Gauss για τη μείωση του θορύβου. Υπολογίζονται οι μετατοπίσεις με τον αλγόριθμο Lucas-Kanade και στη συνέχεια, οι εικόνες υποδειγματοληπτούνται.

Ξεκινώντας από τη χαμηλότερη κλίμακα, ο αλγόριθμος κινείται προς την υψηλότερη κλίμακα. Οι εκτιμώμενες μετατοπίσεις από την προηγούμενη κλίμακα χρησιμοποιούνται ως αρχική εκτίμηση, η οποία στη συνέχεια ανανεώνεται ώστε να αντιστοιχεί στην τρέχουσα κλίμακα. Η εκτίμηση της οπτικής ροής βελτιώνεται σε κάθε κλίμακα. Αυτή η επαναληπτική διαδικασία συνεχίζεται έως ότου υπολογιστεί η τελική εκτίμηση της οπτικής ροής.

Το μόνο μειονέκτημα είναι η υπολογιστική πολυπλοκότητα.

```
# Function that implements Multiscale Lucas-Kanade Algorithm
def multi_lk(I1, I2, features, rho, epsilon, scale, parameters):
    if scale == 0:
        # If at the lowest scale, perform standard Lucas-Kanade optical flow
        dx, dy = lk(I1, I2, features, rho, epsilon, 0, 0)
        return dx, dy
    else:
```

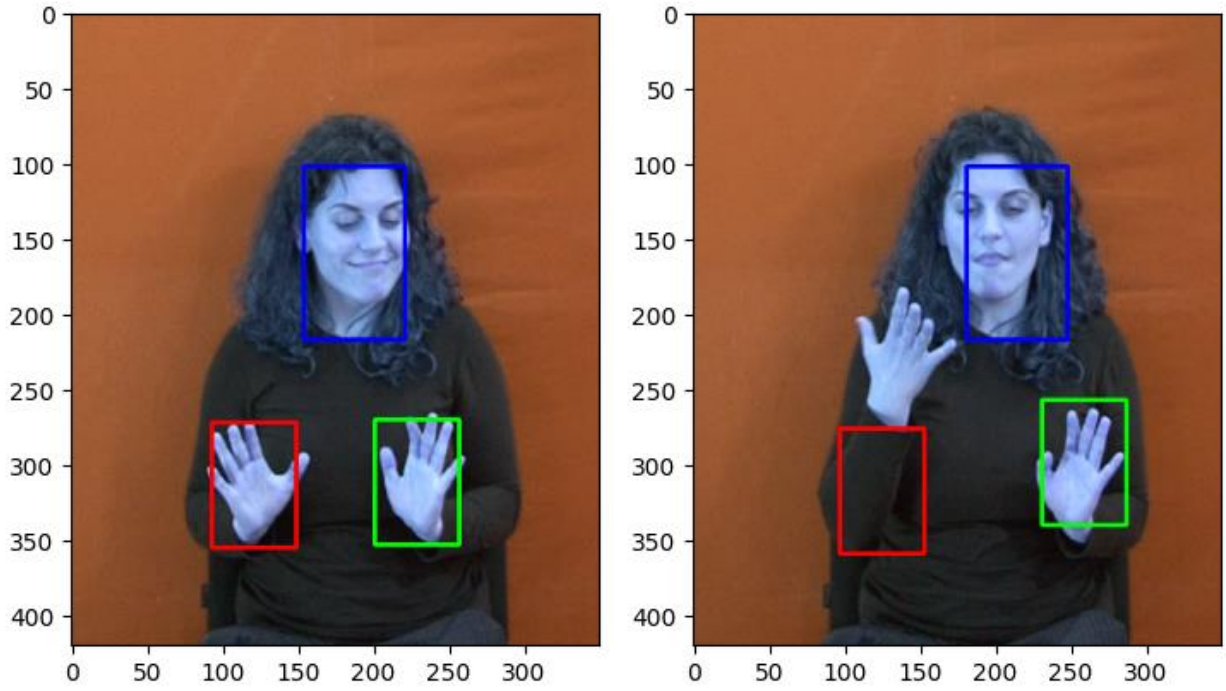
```
        # Apply Gaussian filter to avoid aliasing
        Gr = cv2.getGaussianKernel(3, rho)  # 3-pixel Gaussian kernel
        I1 = cv2.filter2D(I1, -1, Gr)
        I2 = cv2.filter2D(I2, -1, Gr)

        # Downsample the images
        faceOld = cv2.resize(I1, (I1.shape[1]//2, I1.shape[0]//2),
interpolation=cv2.INTER_CUBIC)
        faceNew = cv2.resize(I2, (I2.shape[1]//2, I2.shape[0]//2),
interpolation=cv2.INTER_CUBIC)

        # Extract features on lower resolution
        new_parameters = dict(maxCorners=parameters['maxCorners']+50,
                              qualityLevel=parameters['qualityLevel']-0.001,
                              minDistance=parameters['minDistance']-1)
        new_features = features//2 - 1  # shi_tomasi_feats(faceNew,
new_parameters, False)

        # Recursive call to multiscale Lucas-Kanade algorithm
        dx0, dy0 = multi_lk(faceOld, faceNew, new_features, rho, epsilon,
scale-1, parameters)

        # Upscale the displacements and perform Lucas-Kanade optical flow at
the current scale
        dx, dy = lucas_kanade(I1, I2, features, rho, epsilon, 2*dx0, 2*dy0)
        return dx, dy

xRightHand = 93
yRightHand = 272
widthRightHand = 56
heightRightHand = 83

xLeftHand = 201
yLeftHand = 270
widthLeftHand = 56
heightLeftHand = 83

xHead = 154
yHead = 102
widthHead = 67
heightHead = 115

rho = 3
epsilon=0.15
feature_params = dict( maxCorners = 20,
                       qualityLevel = 0.2,
                       minDistance = 1,
                       blockSize = 5)
```

```
frames = []
#for each 2 images, run mmlti_li function, to track the movement of
Hands/Head (one at a time)
for i in range(1,65):
    path1 = 'part1 - GreekSignLanguage/' + str(i) + '.png'
    I1_bgr = cv2.imread(path1)
    I1_gray = cv2.cvtColor(I1_bgr,cv2.COLOR_BGR2GRAY)
    I1 = I1_gray.astype(np.float)/255


    path2 = 'part1 - GreekSignLanguage/' + str(i+1) + '.png'
    I2_bgr = cv2.imread(path2)
    I2_gray = cv2.cvtColor(I2_bgr,cv2.COLOR_BGR2GRAY)
    I2 = I2_gray.astype(np.float)/255

    det1RightHand = I1_gray[yRightHand-10:(yRightHand+heightRightHand-10),
                            xRightHand+10:(xRightHand+widthRightHand+10)]
    det2RightHand = I2_gray[yRightHand-10:(yRightHand+heightRightHand-10),
                            xRightHand+10:(xRightHand+widthRightHand+10)]

    det1LeftHand = I1_gray[yLeftHand-10:(yLeftHand+heightLeftHand-10),
                           xLeftHand+10:(xLeftHand+widthLeftHand+10)]
    det2LeftHand = I2_gray[yLeftHand-10:(yLeftHand+heightLeftHand-10),
                           xLeftHand+10:(xLeftHand+widthLeftHand+10)]

    det1Head = I1_gray[yHead-10:(yHead+heightHead-10),
                       xHead+10:(xHead+widthHead+10)]
    det2Head = I2_gray[yHead-10:(yHead+heightHead-10),
                       xHead+10:(xHead+widthHead+10)]

    # p0 = cv2.goodFeaturesToTrack(det2, mask = None, **feature_params)
    p0RightHand = cv2.goodFeaturesToTrack(det2RightHand, mask = None,
                                          **feature_params)
    p0LeftHand = cv2.goodFeaturesToTrack(det2LeftHand, mask = None,
                                         **feature_params)
    p0Head = cv2.goodFeaturesToTrack(det2Head, mask = None, **feature_params)

    # d_x, d_y = lucas_kanade(det1, det2, p0 ,rho, epsilon,
np.zeros(det1.shape), np.zeros(det2.shape))
    d_xRightHand, d_yRightHand = lucas_kanade(det1RightHand, det2RightHand,
                                              p0RightHand ,rho, epsilon,
                                              np.zeros(det1RightHand.shape),
                                              np.zeros(det2RightHand.shape))
    d_xLeftHand, d_yLeftHand = lucas_kanade(det1LeftHand, det2LeftHand,
                                              p0LeftHand ,rho, epsilon,
                                              np.zeros(det1LeftHand.shape),
                                              np.zeros(det2LeftHand.shape))
    d_xHead, d_yHead = lucas_kanade(det1Head, det2Head, p0Head ,rho, epsilon,
                                              np.zeros(det1Head.shape),
```

```
                                                    np.zeros(det2Head.shape))

    displ_xRightHand,displ_yRightHand = displ(d_xRightHand, d_yRightHand)
    displ_xRightHand = int(displ_xRightHand)
    displ_yRightHand = int(displ_yRightHand)

    displ_xLeftHand,displ_yLeftHand = displ(d_xLeftHand, d_yLeftHand)
    displ_xLeftHand = int(displ_xLeftHand)
    displ_yLeftHand = int(displ_yLeftHand)

    displ_xHead,displ_yHead = displ(d_xHead, d_yHead)
    displ_xHead = int(displ_xHead)
    displ_yHead = int(displ_yHead)

    plt.figure()
    # cv2.rectangle(I2_bgr, (x, y), (x+width, y+height), (255,0,0) ,2)
    cv2.rectangle(I2_bgr, (xRightHand, yRightHand),
                  (xRightHand+widthRightHand, yRightHand+heightRightHand),
                  (255,0,0) ,2)
    cv2.rectangle(I2_bgr, (xLeftHand, yLeftHand),
                  (xLeftHand+widthLeftHand, yLeftHand+heightLeftHand),
                  (0,255,0) ,2)
    cv2.rectangle(I2_bgr, (xHead, yHead), (xHead+widthHead,
yHead+heightHead),
                  (0,0,255) ,2)

    plt.imshow(I2_bgr)
    ###
    plt.savefig('frame.png')  # Save the plot as a PNG image
    frames.append(imageio.imread('frame.png'))  # Append the frame to the
frames list
    ###
    xRightHand = xRightHand - displ_xRightHand
    yRightHand = yRightHand - displ_yRightHand

    xLeftHand = xLeftHand - displ_xLeftHand
    yLeftHand = yLeftHand - displ_yLeftHand

    xHead = xHead - displ_xHead
    yHead = yHead - displ_yHead

###
output_path = "animation_multiscale2.gif"
imageio.mimsave(output_path, frames, fps=10)  # Adjust the fps value as
needed
print(f"GIF saved at: {output_path}")
###

GIF saved at: animation_multiscale2.gif
```

Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

18/06/2023



Στο βίντεο που φτιάξαμε για τον πολυκλιμακωτό Lucas-Kanade (animation_multiscale.mp4), μπορούμε να διακρίνουμε ότι είναι λίγο πιο robust από τον απλό. Όχι πολύ, όμως, διότι κάναμε πάρα πολλές δοκιμές μέχρι να καταλήξουμε στις υπερπαραμέτρους του απλού Lucas-Kanade.

Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

18/06/2023

Μέρος 2: Εντοπισμός Χωρο-χρονικών Σημείων Ενδιαφέροντος και Εξαγωγή Χαρακτηριστικών σε Βίντεο Ανθρωπίνων Δράσεων

Στο δεύτερο μέρος, θα επικεντρωθούμε στην εξαγωγή χωροχρονικών χαρακτηριστικών με σκοπό την κατηγοριοποίηση βίντεο που περιέχουν ανθρώπινες κινήσεις. Θα χρησιμοποιήσουμε τοπικά χαρακτηριστικά που περιγράφουν περιοχές ενδιαφέροντος με τοπικούς περιγραφητές. Οι τοπικοί περιγραφητές, ύστερα, συνδυάζονται σε μια συνολική αναπαράσταση (Bag-of-Visual_Word).

2.0. Προετοιμασία

1. Κάνουμε import τις απαραίτητες βιβλιοθήκες
2. Φορτώνουμε τα videos προς ανάλυση

```
# imports
from cv23_lab2_2_utils import read_video, show_detection,
orientation_histogram, bag_of_words, svm_train_test
from os import listdir
from os.path import isfile, join
import numpy as np
from scipy import ndimage as nd
import cv2

sigma = 4
tau = 1.5
k = 0.005
s = 2

path = 'part2 - SpatioTemporal/'
walkingVideos     = [path + 'walking/'    + f for f in listdir(path +
'walking/')     if isfile(join(path + 'walking/',    f))]
runningVideos     = [path + 'running/'    + f for f in listdir(path +
'running/')     if isfile(join(path + 'running/',    f))]
handwavingVideos  = [path + 'handwaving/' + f for f in listdir(path +
'handwaving/')  if isfile(join(path + 'handwaving/', f))]

print(walkingVideos)
print(runningVideos)
print(handwavingVideos)

walkVideo1     = read_video(walkingVideos[0],      200)
runVideo1      = read_video(runningVideos[0],      200)
handwaveVideo1 = read_video(handwavingVideos[0],   200)

walkVideo2     = read_video(walkingVideos[1],      200)
runVideo2      = read_video(runningVideos[1],      200)
handwaveVideo2 = read_video(handwavingVideos[1],   200)

walkVideo3     = read_video(walkingVideos[2],      200)
runVideo3      = read_video(runningVideos[2],      200)
handwaveVideo3 = read_video(handwavingVideos[2],   200)
```

2.1. Χωρο-χρονικά Σημεία Ενδιαφέροντος

Τα χωρο-χρονικά σημεία ενδιαφέροντος εξάγονται με τη χρήση ανιχνευτών τοπικών χαρακτηριστικών (local feature detectors), οι οποίοι εντοπίζουν περιοχές στο βίντεο που παρουσιάζουν σύνθετη κίνηση ή σημαντικές μεταβολές στην εμφάνιση. Αυτοί οι ανιχνευτές μεγιστοποιούν μια συνάρτηση "οπτικής σημαντικότητας" για τον εντοπισμό σημαντικών σημείων και κλιμάκων. Θα επικεντρωθούμε σε δύο ανιχνευτές: τον Harris και τον Gabor.

- Harris Detector: Ο ανιχνευτής Harris εντοπίζει σημεία ενδιαφέροντος με βάση τις γωνίες στο βίντεο, αναλύοντας τις τοπικές μεταβολές της έντασης των pixels σε πολλαπλές κατευθύνσεις. Τα σημεία με υψηλές αποκρίσεις σε αυτές τις μεταβολές θεωρούνται χωροχρονικά σημεία ενδιαφέροντος.

- Gabor Detector: Ο ανιχνευτής Gabor χρησιμοποιεί φίλτρα Gabor που προσεγγίζουν την ανθρώπινη οπτική αντίληψη. Αυτά τα φίλτρα εφαρμόζονται στα frames και καταγράφουν τοπικά μοτίβα σε πολλές χωρικές κλίμακες και διάφορους προσανατολισμούς. Στη συνέχεια, οι αποκρίσεις αυτών των φίλτρων χρησιμοποιούνται για τον εντοπισμό σημείων ενδιαφέροντος, ως εκείνα με σημαντική ενέργεια.

2.1.1. Harris Detector for Spatio-Temporal Points of Interest

Ορίζουμε τις συναρτήσεις:

1. gaussian1D(sigma): Επιστρέφει έναν 1D γκαουσιανό πυρήνα με τυπική απόκλιση sigma.

2. gaussian2D(sigma): Επιστρέφει έναν 2D γκαουσιανό πυρήνα με τυπική απόκλιση sigma.

3. getInterestPoints(H, N, sigma): Αυτή η συνάρτηση λαμβάνει το κριτήριο σημαντικότητας Harris H, τον αριθμό των σημείων ενδιαφέροντος που πρόκειται να επιλεγούν N και την αρχική χωρική κλίμακα sigma. Προσδιορίζει τα σημεία (N το πλήθος) με τις υψηλότερες τιμές κριτηρίου και επιστρέφει τις συντεταγμένες τους ως λίστα [y, x, t, sigma] (χωρο-χρονικές συντεταγμένες και χωρική κλίμακα). Η συνάρτηση κάνει flat τον πίνακα H (ravel()) και ταξινομεί τις συντεταγμένες, σε μορφή δεικτών, αφού έχουμε κάνει flat τον H, σε αύξουσα σειρά. Ύστερα επιλέγει τους N δείκτες με τις μεγαλύτερες τιμές κριτηρίου και τους μετατρέπει σε συντεταγμένες.

4. HarrisDetector(video, sigma, tau, k, s): Λαμβάνει το βίντεο video, τις αρχικές χωρική και χρονική κλίμακες (sigma, tau), την τιμή κατωφλίου (k) και την παράμετρο κλιμάκωσης (s). Τα βήματα είναι τα εξής:

   - Μετατροπή του βίντεο σε τύπο float32
   - Δημιουργία 2D χωρικού Gaussian πυρήνα (gauss2DSpace)
   - Δημιουργία 1D χρονικού Gaussian πυρήνα Gauss (gauss2DTime)
   - Εκτελούμε χωροχρονικό φιλτράρισμα του βίντεο χρησιμοποιώντας τους πυρήνες, συνελίσσοντάς το με το χωροχρονικό φίλτρο Gauss.
   - Ορίζουμε τα φίλτρα των παραγώγων (dx, dy, dt) και υπολογίζουμε τις χωρικές παραγώγους (Lx, Ly) και τη χρονική παράγωγο (Lt) του φιλτραρισμένου βίντεο L ως συνέλιξη με τα φίλτρα αυτά.

- Υπολογίζουμε τα στοιχεία του πίνακα M ως συνέλιξη με τον πυρήνα εξομάλυνσης gaussSpaceTimeSmooth ο οποίος είναι συνέλιξη των πυρήνων gauss1DTimeSmooth και gauss2DSpaceSmooth
- Υπολογίζουμε το κριτήριο γωνιότητας εφαρμόζοντας τον τύπο του Harris.
- Επιστρέφουμε το κριτήριο σημαντικότητας Harris.

Εκφράζουμε και με μαθηματικά τον ανιχνευτή Harris:

$$M(x,y,t;\sigma,\tau) = g(x,y,t;s\sigma,s\tau) * \left(\nabla L(x,y,t;\sigma,\tau)\left(\nabla L(x,y,t;\sigma,\tau)\right)^T\right) \Leftrightarrow$$

$$\Leftrightarrow M(x,y,t;\sigma,\tau) = g(x,y,t;s\sigma,s\tau) * \begin{pmatrix} L_x^2 & L_xL_y & L_xL_t \\ L_xL_y & L_y^2 & L_yL_t \\ L_xL_t & L_yL_t & L_t^2 \end{pmatrix}$$

Και το 3D κριτήριο γωνιότητας:

$$H(x,y,t) = \det\left(M(x,y,t)\right) - k \cdot \mathrm{trace}^3\left(M(x,y,t)\right)$$

```
def gaussian1D(sigma):
    return cv2.getGaussianKernel(int(np.ceil(3*sigma)*2 + 1), sigma)

def gaussian2D(sigma):
    gauss1D = gaussian1D(sigma)
    return gauss1D @ gauss1D.transpose()

def getInterestPoints(H, N, sigma):
    # Flatten H matrix and obtain sorted indices
    HFlat = np.dstack(np.unravel_index(np.argsort(H.ravel()), (H.shape[0],
H.shape[1], H.shape[2])))

    # Reconstruct the sorted indices into the original shape
    HReconstructed = HFlat.reshape(HFlat.shape[1], HFlat.shape[2])

    # Select the N best interest points based on the reconstructed indices
    NBest = HReconstructed[-N:]

    # Construct an array of interest point coordinates [x, y, t, sigma] and
return
    return np.array([np.append(pair[:2][::-1],[pair[2], sigma]) for pair in
NBest])

def HarrisDetector (video, sigma, tau, k, s):
    video = video.astype(np.float32)

    # Create a 2D Gaussian kernel for spatial filtering
    gauss2DSpace = gaussian2D(sigma)
    gauss2DSpace = gauss2DSpace.reshape(gauss2DSpace.shape[0],
gauss2DSpace.shape[1], 1)
```

```
    # Create a 1D Gaussian kernel for temporal filtering
    gauss2DTime = gaussian1D(tau)
    gauss2DTime = gauss2DTime.reshape(1, 1, gauss2DTime.shape[0])

    # Combine the spatial and temporal kernels for spatio-temporal filtering
    gaussSpaceTime = nd.convolve(gauss2DTime, gauss2DSpace)

    # Filter the input video with the spatio-temporal kernel
    L = nd.convolve(video, gaussSpaceTime)

    dx = np.array([[[-1],[0],[1]]], dtype=np.int8) # Spatial derivative
filter in x-direction
    dy = np.array([[[-1]],[[0]],[[1]]], dtype=np.int8) # Spatial derivative
filter in y-direction
    dt = np.array([[[-1,  0,  1]]], dtype=np.int8) # Temporal derivative
filter

    # Compute the spatial and temporal derivatives of L
    Lx = nd.convolve(L, dx)
    Ly = nd.convolve(L, dy)
    Lt = nd.convolve(L, dt)

    # Compute the elements of M (second moment matrix) before convolution
with the smoothing kernel
    Lxx = Lx * Lx
    Lyy = Ly * Ly
    Ltt = Lt * Lt
    Lxy = Lx * Ly
    Lxt = Lx * Lt
    Lyt = Ly * Lt

    # Create a 2D Gaussian kernel for smoothing
    gauss2DSpaceSmooth = gaussian2D(s * sigma)
    gauss2DSpaceSmooth =
gauss2DSpaceSmooth.reshape(gauss2DSpaceSmooth.shape[0],
gauss2DSpaceSmooth.shape[1], 1)

    # Create a 1D Gaussian kernel for temporal smoothing
    gauss1DTimeSmooth = gaussian1D(s * tau)
    gauss1DTimeSmooth = gauss1DTimeSmooth.reshape(1, 1,
gauss1DTimeSmooth.shape[0])

    # Combine the spatial and temporal kernels for spatio-temporal smoothing
    gaussSpaceTimeSmooth = nd.convolve(gauss1DTimeSmooth, gauss2DSpaceSmooth)

    # Convolve the elements of M with the smoothing kernel
    Mxx = nd.convolve(Lxx, gaussSpaceTimeSmooth)
    Myy = nd.convolve(Lyy, gaussSpaceTimeSmooth)
```

```
    Mtt = nd.convolve(Ltt, gaussSpaceTimeSmooth)
    Mxy = nd.convolve(Lxy, gaussSpaceTimeSmooth)
    Mxt = nd.convolve(Lxt, gaussSpaceTimeSmooth)
    Myt = nd.convolve(Lyt, gaussSpaceTimeSmooth)

    # Compute the trace and determinant of M to obtain the Harris corner
criterion
    tr = Mxx + Myy + Mtt
    det = (Mxx*Myy*Mtt - Mxx*Myt**2) - (Mtt*Mxy**2 - Mxy*Mxt*Myt) +
(Mxy*Mxt*Myt - Myy*Mxt**2)
    H = det - k*tr**3

    return H
```

2.1.2. Gabor Detector for Spatio-Temporal Points of Interest

Η συνάρτηση GaborDetector(video, sigma, tau) λαμβάνει το βίντεο video, τις αρχικές χωρική και χρονική κλίμακες (sigma, tau), την τιμή κατωφλίου (k) και την παράμετρο κλιμάκωσης (s). Τα βήματα είναι τα εξής:

1. Μετατρέπουμε το βίντεο video σε τύπο δεδομένων float32.
2. Δημιουργούμε έναν 2D Gaussian πυρήνα για χωρική εξομάλυνση.
3. Ορίζουμε το χρονικό εύρος time για τα φίλτρα Gabor με βάση την χρονική κλίμακα tau.
4. Υπολογίζουμε τη συχνότητα w των φίλτρων Gabor ως w = 4/tau.
5. Ορίζουμε τα φίλτρα Gabor hEven και hOdd (even, odd) ως συνημίτονο και ημίτονο πολλαπλασιασμένες με το εκθετικό, αντίστοιχα.
6. Κανονικοποιούμε τα φίλτρα Gabor διαιρώντας τα με την L1 νόρμα τους και τα ταιριάζουμε με τις διαστάσεις του βίντεο.
7. Καταλήξαμε με το χωρικά εξομαλυμένο βίντεο videoSmoothed και τα φίλτρα Gabor hEven και hOdd.
8. Υπολογίζουμε την ενέργεια του φιλτραρισμένου με τα Gabor βίντεο.
9. Επιστρέφουμε αυτήν την απόκριση H.

Η έξοδος αντιπροσωπεύει τη σημασία των σημείων ενδιαφέροντος στο βίντεο με βάση τα χωροχρονικά χαρακτηριστικά τους, όπως αυτά προκύπτουν από τα φίλτρα Gabor.

Εκφράζουμε και με μαθηματικά τα φίλτρα Gabor:

$$h_{ev}(t;\tau,\omega) = \cos(2\pi t\omega)\exp\left[\frac{-t^2}{2\tau^2}\right]$$

$$h_{od}(t;\tau,\omega) = \sin(2\pi t\omega)\exp\left[\frac{-t^2}{2\tau^2}\right]$$

Και την τετραγωνική ενέργεια της εξόδου για το ζεύγος Gabor φίλτρων:

$$H(x,y,t) = (I(x,y,t) * g * h_{ev})^2 + (I(x,y,t) * g * h_{od})^2$$

```
def GaborDetector(video, sigma, tau):
    video = video.astype(np.float32)

    # Spatial smoothing using a 2D Gaussian kernel
    gauss2DSpace = gaussian2D(sigma)
    gauss2DSpace = gauss2DSpace.reshape(gauss2DSpace.shape[0],
gauss2DSpace.shape[1], 1)
    videoSmoothed = nd.convolve(video, gauss2DSpace)

    # Define the time range and Gabor parameters
    time = np.linspace(int(-2 * tau), int(2 * tau), int(4 * tau + 1),
endpoint=True)
    w = 4 / tau

    # Compute the Gabor filters for even and odd components
    hEven = -np.cos(2 * np.pi * time * w) * np.exp((-time ** 2) / (2 * tau **
2))
    hEven = hEven / np.linalg.norm(hEven, ord=1)  # Normalize the filter by
L1 norm
    hOdd = -np.sin(2 * np.pi * time * w) * np.exp((-time ** 2) / (2 * tau **
2))
    hOdd = hOdd / np.linalg.norm(hOdd, ord=1)  # Normalize the filter by L1
norm

    # Reshape the filters to match the input video dimensions
    hEven = hEven.reshape(1, 1, hEven.shape[0])
    hOdd = hOdd.reshape(1, 1, hOdd.shape[0])

    # Compute the Gabor shock response
    H = (nd.convolve(videoSmoothed, hEven)) ** 2 +
(nd.convolve(videoSmoothed, hOdd)) ** 2

    return H
```

2.1.3. Reject Points with Low Angular Criterion

Όπως στο πρώτο εργαστήριο, απορρίπτουμε τα σημεία που εξάγαμε τα οποία δεν ξεπερνούν το "κατώφλι" γωνιότητας.

```
import os

os.makedirs('detection/harris/walk', exist_ok=True)
os.makedirs('detection/harris/run', exist_ok=True)
os.makedirs('detection/harris/handwave', exist_ok=True)

os.makedirs('detection/gabor/walk', exist_ok=True)
os.makedirs('detection/gabor/run', exist_ok=True)
os.makedirs('detection/gabor/handwave', exist_ok=True)
```

2.1.4. Points of Interest Detection and Visualization using Harris and Gabor Detectors

Σε αυτό το βήμα ανιχνεύουμε τα σημεία ενδιαφέροντος χρησιμοποιώντας τους ανιχνευτές Harris και Gabor. Τα N σημεία με τις υψηλότερες τιμές κριτηρίου επιλέγονται ως σημεία ενδιαφέροντος. Επιπλέον, παράγουμε αρχεία βίντεο που παρουσιάζουν τα αποτελέσματα ανίχνευσης για κάθε ενέργεια.

- Εφαρμόζουμε τον ανιχνευτή Harris στα βίντεο με τις συνιστώμενες παραμέτρους.
- Υπολογίζουμε τα N σημεία με τις μεγαλύτερες τιμές του κριτηρίου σημαντικότητας χρησιμοποιώντας τη συνάρτηση getInterestPoints.
- Εμφανίζουμε τα κριτήρια σημαντικότητας χρησιμοποιώντας τη συνάρτηση show_detection.

```
# Harris Detector
walk_harris_criterion = HarrisDetector(walkVideo3, 4, 1.5, 0.005, 2)
run_harris_criterion  = HarrisDetector(runVideo3,  4, 1.5, 0.005, 2)
handwave_harris_criterion  = HarrisDetector(handwaveVideo3,  4, 1.5, 0.005,
2)

walk_harris_points = getInterestPoints(walk_harris_criterion, 600, 4)
run_harris_points  = getInterestPoints(run_harris_criterion,  600, 4)
handwave_harris_points  = getInterestPoints(handwave_harris_criterion,  600,
4)

show_detection(walkVideo3, walk_harris_points,
save_path=r"./detection/harris/walk/")

show_detection(runVideo3,  run_harris_points,
save_path=r"./detection/harris/run/")

show_detection(handwaveVideo3,  handwave_harris_points,
save_path=r"./detection/harris/handwave/")
```

Ομοίως:

- Εφαρμόζουμε τον ανιχνευτή Gabor στα βίντεο με τις συνιστώμενες παραμέτρους.
- Υπολογίζουμε τα N σημεία με τις μεγαλύτερες τιμές του κριτηρίου σημαντικότητας χρησιμοποιώντας τη συνάρτηση getInterestPoints.
- Εμφανίζουμε τα κριτήρια σημαντικότητας χρησιμοποιώντας τη συνάρτηση show_detection.

```
# Gabor Detector
walk_gabor_criterion     = GaborDetector(walkVideo3,      1.6, 1.5)
run_gabor_criterion      = GaborDetector(runVideo3,       1.6, 1.5)
handwave_gabor_criterion = GaborDetector(handwaveVideo3,  1.6, 1.5)

walk_gabor_points     = getInterestPoints(walk_gabor_criterion,      600,
1.6)
run_gabor_points      = getInterestPoints(run_gabor_criterion,       600,
1.6)
handwave_gabor_points = getInterestPoints(handwave_gabor_criterion,  600,
1.6)
```

```
show_detection(walkVideo3, walk_gabor_points,
save_path=r"./detection/gabor/walk/")

show_detection(runVideo3,  run_gabor_points,
save_path=r"./detection/gabor/run/")

show_detection(handwaveVideo3,  handwave_gabor_points,
save_path=r"./detection/gabor/handwave/")
```

Καταγράφουμε τα αποτέλεσματα του ανιχνευτή Harris σε ένα αρχείο βίντεο για τις ακολουθίες walk, run και handwave.

```
import cv2

# Set the directory path and file extension
directoryWalk = "./detection/harris/walk/"
directoryRun = "./detection/harris/run/"
directoryHandwave = "./detection/harris/handwave/"
file_extension = ".png"

# Create a VideoWriter objects
output_pathWalk = "./detection/harris/harrisWalk.mp4"
output_pathRun = "./detection/harris/harrisRun.mp4"
output_pathHandwave = "./detection/harris/harrisHandwave.mp4"
fps = 25
output_size = (640, 480)  # Adjust the size as needed
fourcc = cv2.VideoWriter_fourcc(*"mp4v")
video_writerWalk = cv2.VideoWriter(output_pathWalk, fourcc, fps, output_size)
video_writerRun = cv2.VideoWriter(output_pathRun, fourcc, fps, output_size)
video_writerHandwave = cv2.VideoWriter(output_pathHandwave, fourcc, fps,
output_size)

# Iterate over the frames and write them to the video
for i in range(200):  # Assuming you have 200 frames (frame0.png to
frame199.png)
    frame_pathWalk = directoryWalk + f"frame{i}{file_extension}"
    frame_pathRun = directoryRun + f"frame{i}{file_extension}"
    frame_pathHandwave = directoryHandwave + f"frame{i}{file_extension}"

    frameWalk = cv2.imread(frame_pathWalk)
    frameRun = cv2.imread(frame_pathRun)
    frameHandwave = cv2.imread(frame_pathHandwave)

    video_writerWalk.write(frameWalk)
    video_writerRun.write(frameRun)
    video_writerHandwave.write(frameHandwave)

# Release the video writer and close any open windows
```

```
video_writerWalk.release()
video_writerRun.release()
video_writerHandwave.release()

cv2.destroyAllWindows()
```

Καταγράφουμε τα αποτέλεσματα του ανιχνευτή Gabor σε ένα αρχείο βίντεο για τις ακολουθίες walk, run και handwave.

```
import cv2
# Set the directory path and file extension
directoryWalk = "./detection/gabor/walk/"
directoryRun = "./detection/gabor/run/"
directoryHandwave = "./detection/gabor/handwave/"
file_extension = ".png"

# Create a VideoWriter objects
output_pathWalk = "./detection/gabor/gaborWalk.mp4"
output_pathRun = "./detection/gabor/gaborRun.mp4"
output_pathHandwave = "./detection/gabor/gaborHandwave.mp4"
fps = 25
output_size = (640, 480)  # Adjust the size as needed
fourcc = cv2.VideoWriter_fourcc(*"mp4v")
video_writerWalk = cv2.VideoWriter(output_pathWalk, fourcc, fps, output_size)
video_writerRun = cv2.VideoWriter(output_pathRun, fourcc, fps, output_size)
video_writerHandwave = cv2.VideoWriter(output_pathHandwave, fourcc, fps,
output_size)

# Iterate over the frames and write them to the video
for i in range(200):  # Assuming you have 200 frames (frame0.png to
frame199.png)
    frame_pathWalk = directoryWalk + f"frame{i}{file_extension}"
    frame_pathRun = directoryRun + f"frame{i}{file_extension}"
    frame_pathHandwave = directoryHandwave + f"frame{i}{file_extension}"

    frameWalk = cv2.imread(frame_pathWalk)
    frameRun = cv2.imread(frame_pathRun)
    frameHandwave = cv2.imread(frame_pathHandwave)

    video_writerWalk.write(frameWalk)
    video_writerRun.write(frameRun)
    video_writerHandwave.write(frameHandwave)

# Release the video writer and close any open windows
video_writerWalk.release()
video_writerRun.release()
video_writerHandwave.release()

cv2.destroyAllWindows()
```

2.2. Spatio-Temporal Histographic Descriptors

2.2.1. Gradient and Optical Flow Calculation for Spatio-temporal Descriptors

Σε αυτό το σημείο, υπολογίζουμε το διάνυσμα του gradient και την οπτική ροή TV-L1 για κάθε pixel σε κάθε frame του βίντεο.

1. `get_gradient(video)`: Λαμβάνει ένα βίντεο ως είσοδο και υπολογίζει τα διανύσματα gradient στις διευθύνσεις x και y χρησιμοποιώντας τη συνάρτηση `np.gradient()`.

2. `get_optical_flow(video)`: Υπολογίζει την οπτική ροή TV-L1 για κάθε pixel μεταξύ διαδοχικών frame του βίντεο. Χρησιμοποιεί τη συνάρτηση `cv2.DualTVL1OpticalFlow_create()`. Οι τιμές οπτικής ροής για τις διευθύνσεις x και y υπολογίζονται και αποθηκεύονται στους πίνακες `flow_x` και `flow_y`.

```
def get_gradient(video):
    dy, dx, _ = np.gradient(video)  # Calculate gradient vectors in x and y
directions
    return (dy, dx)

def get_optical_flow(video):
    video = video.astype(np.uint8)  # Convert video to uint8 type

    flow_x = np.zeros((video.shape[0], video.shape[1], video.shape[2]))  #
Initialize array to store optical flow in x direction
    flow_y = np.zeros((video.shape[0], video.shape[1], video.shape[2]))  #
Initialize array to store optical flow in y direction
    for f in range(video.shape[2]):
        t = f
        if f == video.shape[2]-1:
            t = f-1

        temp = cv2.DualTVL1OpticalFlow_create(nscales=1).calc(video[:,:,t],
video[:,:,t+1], None)  # Calculate TV-L1 optical flow
        flow_x[:,:,t] = temp[:,:,1]  # Store optical flow values in x
direction
        flow_y[:,:,t] = temp[:,:,0]  # Store optical flow values in y
direction

    return (flow_y, flow_x)
```

2.2.2. Calculation of Histogram-based Descriptors (HOG/HOF)

Θα υπολογίσουμε τα τους HOG και HOF descriptors. Αυτοί οι περιγραφητές υπολογίζονται με βάση τα διανυσματικά πεδία (grad, optical flow) γύρω από τα σημεία ενδιαφέροντος.

Η συνάρτηση get_descriptor() λαμβάνει το video, τα σημεία ενδιαφέροντος points και τον descriptor και επιστρέφει το ζητούμενο περιγραφητή για το video ως πίνακα.

```
def get_descriptor(video, points, descriptor, nbins=9, n=3, m=3):

    height = video.shape[0]
    width  = video.shape[1]
    desc = []

    if descriptor == 'HOG':
        # Calculate gradient vectors (dy, dx)
        dy, dx = get_gradient(video)
    elif descriptor == 'HOF':
        # Calculate optical flow vectors (dy, dx)
        dy, dx = get_optical_flow(video)
    elif descriptor == 'HOG/HOF':
        # Call for HOG descriptor
        HOG = get_descriptor(video, points, 'HOG', nbins, n, m)
        # Call for HOF descriptor
        HOF = get_descriptor(video, points, 'HOF', nbins, n, m)
        # Merge HOG and HOF descriptors
        return np.concatenate((HOG, HOF))
    else:
        print('Unknown Descriptor Type')
        return

    for point in points:
        x = point[0]
        y = point[1]
        t = point[2]
        sigma = point[3]
        # Compute the size of the region based on 4 times the sigma
        side = int (np.round(4*sigma))

        xLeft  = max(0, x-side)        # Calculate the left boundary of the
region
        xRight = min(width, x+side+1)  # Calculate the right boundary of the
region
        yUp    = max(0, y-side)        # Calculate the top boundary of the
region
        yDown  = min(height, y+side+1) # Calculate the bottom boundary of the
region

        # Extract gradient vectors within the region
        Gx = dx[yUp:yDown, xLeft:xRight, t]
        # Extract gradient vectors within the region
        Gy = dy[yUp:yDown, xLeft:xRight, t]

        desc.append(orientation_histogram(Gx, Gy, nbins, np.array([n, m])))
# Compute orientation histogram

    return np.array(desc)
```

2.3 Bag of Visual Words Construction and use of Support Vector Machines for action classification

- Bag-of-Visual_words model in computer vision:
  Το μοντέλο Bag-of-Visual_Words χρησιμοποιείται στην όραση υπολογιστών για την ταξινόμηση εικόνων. Είναι το ανάλογο του μοντέλου Bag-of-Words στην επεξεργασία φυσικής γλώσσας. Αντί, για λέξεις, το μοντέλο Bag-of-Visual_Words χρησιμοποιεί χαρακτηριστικά εικόνων που έχουν εξαχθεί με κατάλληλη επεξεργασία, ώστε να δημιουργηθεί ένα διάνυσμα με το πλήθος εμφανίσεων κάθε τοπικού χαρακτηριστικού.

- Histogram of Oriented Gradients (HoG):
  Η κύρια ιδέα πίσω από τον συγκεκριμένο περιγραφητή είναι ότι κάθε αντικείμενο στην εικόνα ορίζεται από τις κατευθύνσεις συγκεκριμένων ακμών, δηλαδή από τις κλίσεις της φωτεινότητας στην περιοχή ενδιαφέροντος. Με άλλα λόγια, η κατανομή των κλίσεων της φωτεινότητας μπορεί να δείξει ποια αντικείμενα βρίσκονται που. Η εικόνα κατατέμνεται σε μικρές περιοχές και εκεί υπολογίζεται η κατανομή των κλίσεων. Η παράθεση (concatenation) των ιστογραμμάτων που προκύπτουν, δίνουν τον περιγραφητή HoG.

  Δεν επηρεάζεται από γεωμετρικούς μετασχηματισμούς, εκτός από την περιστροφή του αντικειμένου. Επειδή οι άνθρωποι τείνουν να έχουν περίπου την ίδια στάση σώματος, ο HoG προσφέρεται ιδιαιτέρως για αναγνώριση ανθρώπινης φιγούρας σε εικόνα/βίντεο.

- Histogram of Oriented Flow (HoF):
  Είναι ένας περιγραφητής χαρακτηριστικών που αντιπροσωπεύει την κατανομή των προσανατολισμών οπτικής ροής, η οποία περιγράφει την κίνηση των patterns φωτεινότητας μεταξύ διαδοχικών frames. Παρόμοια με το HOG, το βίντεο χωρίζεται σε μικρές περιοχές και υπολογίζονται τα ιστογράμματα των προσανατολισμών οπτικής ροής εντός κάθε περιοχής. Η συνένωση αυτών των ιστογραμμάτων είναι ο περιγραφητής HOF.

2.3.1. Split Video Set into Train Set & Test Set

Διαχωρίζουμε το set με τα videos σε train set και test set βάσει του αρχείου που δίνεται

```
def trainTestSplit(training_videos):
    with open(training_videos, 'r') as f:
        train_names = [line.strip() for line in f.readlines()]
        train_set = []
        test_set = []

        train_labels = []
        test_labels = []
        for name in listdir('./part2 - SpatioTemporal/walking/'):
            if name in train_names:
                train_set.append(read_video('./part2 -
SpatioTemporal/walking/'+name, 200))
                train_labels.append(0)
            else:
                test_set.append(read_video('./part2 -
SpatioTemporal/walking/'+name, 200))
                test_labels.append(0)
```

```
        for name in listdir('./part2 - SpatioTemporal/running/'):
            if name in train_names:
                train_set.append(read_video('./part2 -
SpatioTemporal/running/'+name, 200))
                train_labels.append(1)
            else:
                test_set.append(read_video('./part2 -
SpatioTemporal/running/'+name, 200))
                test_labels.append(1)

        for name in listdir('./part2 - SpatioTemporal/handwaving/'):
            if name in train_names:
                train_set.append(read_video('./part2 -
SpatioTemporal/handwaving/'+name, 200))
                train_labels.append(2)
            else:
                test_set.append(read_video('./part2 -
SpatioTemporal/handwaving/'+name, 200))
                test_labels.append(2)

        train_set = np.array(train_set)
        test_set = np.array(test_set)

    return train_set, test_set, train_labels, test_labels
```

2.3.2. Bag of Visual Words Representation for Video Classification

Υπολογίζουμε τη συνολική αναπαράσταση (Bag of Visual Words) για κάθε βίντεο στα σύνολα εκπαίδευσης και δοκιμής. Αρχικά εξάγουμε περιγραφητές για τα βίντεο με βάση και για όποιον περιγραφητή χρειάζεται (HOG, HOF ή HOG/HOF) και τον επιλεγμένο ανιχνευτή (Harris ή Gabor). Οι παράμετροι sigma, tau, k, s είναι οι ενδεικνύμενες. Στη συνέχεια, τα BoVW υπολογίζονται χρησιμοποιώντας τους περιγραφητές. Η συνάρτηση τελικά επιστρέφει τις αναπαραστάσεις BoVW για τα σύνολα εκπαίδευσης και δοκιμής.

```
def getDescriptorOfVideoSet(descriptor, detector, video_set, sigma, tau, k,
s):
    n = int(np.round(4*sigma))  # Calculate the size of the neighborhood for
descriptor extraction
    m = int(np.round(4*sigma))  # Calculate the size of the neighborhood for
descriptor extraction
    nbins = 9  # Number of bins for histogram representation

    if descriptor != 'HOG' and descriptor != 'HOF' and descriptor !=
'HOG/HOF':
        print('Unknown Descriptor')
        return

    descriptors_for_video_set = []  # List to store descriptors for each
```

```
video in the set

    if detector == 'Harris':
        for video in video_set:
            criterion = HarrisDetector(video, sigma, tau, k, s)  # Apply
Harris detector on video frames
            points = getInterestPoints(criterion, 600, sigma)  # Get interest
points from the criterion
            desc = get_descriptor(video, points, descriptor, nbins, n, m)  #
Extract descriptors based on descriptor type
            descriptors_for_video_set.append(desc)  # Add descriptors to the
list

    elif detector == 'Gabor':
        for video in video_set:
            criterion = GaborDetector(video, sigma, tau)  # Apply Gabor
detector on video frames
            points = getInterestPoints(criterion, 600, sigma)  # Get interest
points from the criterion
            desc = get_descriptor(video, points, descriptor, nbins, n, m)  #
Extract descriptors based on descriptor type
            descriptors_for_video_set.append(desc)  # Add descriptors to the
list

    else:
        print('Unknown Detector')
        return

    return descriptors_for_video_set


def calculate_BoVW(train_set, test_set, descriptor, detector, sigma, tau, k,
s, num_centers):

    desc_train = getDescriptorOfVideoSet(descriptor, detector, train_set,
sigma, tau, k, s)  # Extract descriptors for training set
    desc_test  = getDescriptorOfVideoSet(descriptor, detector, test_set,
sigma, tau, k, s)  # Extract descriptors for test set

    bow_train, bow_test = bag_of_words(desc_train, desc_test, num_centers)  #
Compute BoVW histograms

    return (bow_train, bow_test)
```

2.3.3. Multi-class Image Categorization with BoVW and SVM Classifier

Η συνάρτηση classify εκτελεί την τελική κατηγοριοποίηση των εικόνων χρησιμοποιώντας την αναπαράσταση BoVW. Χρησιμοποιεί έναν ταξινομητή SVM για πολλαπλές κλάσεις (3) για εκπαίδευση και δοκιμή στα διανύσματα BoVW και τυπώνει την ακρίβεια ετικέτες που προέβλεψε.

```
def classify (train_set, test_set, train_labels, test_labels, descriptor, detector, sigma, tau, k, s, num_centers=20):
    print(f"{detector} Detector with {descriptor} descriptor and sigma = {sigma}, tau = {tau}")
    print()

    # Calculate BoVW representation for training and test sets
    bow_train, bow_test = calculate_BoVW(train_set, test_set, descriptor, detector, sigma, tau, k, s, num_centers)

    # Train and test the SVM classifier
    accuracy, pred = svm_train_test(bow_train, train_labels, bow_test, test_labels)

    # Print the results
    print("\t Accuracy: {0:.2f}%".format(accuracy*100))
    print("\t Prediction:", pred)

    return



train_set, test_set, train_labels, test_labels = trainTestSplit('traininng_videos.txt')
# sigma = typ['sigma']
# tau = typ['tau']
# k = typ['k']
# s = typ['s']
sigma = 4
tau = 1.5
k = 0.005
s = 2
num_centers = 20

# Detector: Harris,  Descriptor: HOG
print('+-----------------------------------------+')
print('|     Classifying with HOG - Harris       |')
print('+-----------------------------------------+')
classify(train_set, test_set, train_labels, test_labels, 'HOG', 'Harris', sigma, tau, k, s, 20)
print()

# Detector: Gabor,   Descriptor: HOG
print('+-----------------------------------------+')
print('|      Classifying with HOG - Gabor       |')
print('+-----------------------------------------+')
classify(train_set, test_set, train_labels, test_labels, 'HOG', 'Gabor', sigma, tau, k, s, 20)
print()
```

```
# Detector: Harris,  Descriptor: HOF
print('+---------------------------------------+')
print('|     Classifying with HOF - Harris     |')
print('+---------------------------------------+')
classify(train_set, test_set, train_labels, test_labels, 'HOF', 'Harris',
sigma, tau, k, s, 20)
print()

# Detector: Gabor,   Descriptor: HOF
print('+---------------------------------------+')
print('|      Classifying with HOF - Gabor     |')
print('+---------------------------------------+')
classify(train_set, test_set, train_labels, test_labels, 'HOF', 'Gabor',
sigma, tau, k, s, 20)
print()

# Detector: Harris,  Descriptor: HOG/HOF
print('+---------------------------------------+')
print('|   Classifying with HOG/HOF - Harris   |')
print('+---------------------------------------+')
classify(train_set, test_set, train_labels, test_labels, 'HOG/HOF', 'Harris',
sigma, tau, k, s, 20)
print()

# Detector: Gabor,   Descriptor: HOG/HOF
print('+---------------------------------------+')
print('|    Classifying with HOG/HOF - Gabor   |')
print('+---------------------------------------+')
classify(train_set, test_set, train_labels, test_labels, 'HOG/HOF', 'Gabor',
sigma, tau, k, s, 20)
print()
```

```
+---------------------------------------+
|     Classifying with HOG - Harris     |
+---------------------------------------+
Harris Detector with HOG descriptor and sigma = 4, tau = 1.5

       Accuracy: 91.67%
       Prediction: [0 0 0 0 2 1 1 1 2 2 2 2]

+---------------------------------------+
|      Classifying with HOG - Gabor     |
+---------------------------------------+
Gabor Detector with HOG descriptor and sigma = 4, tau = 1.5

       Accuracy: 100.00%
       Prediction: [0 0 0 0 1 1 1 1 2 2 2 2]
```

Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053


```
+---------------------------------------+
|     Classifying with HOF - Harris     |
+---------------------------------------+
Harris Detector with HOF descriptor and sigma = 4, tau = 1.5

       Accuracy: 100.00%
       Prediction: [0 0 0 0 1 1 1 1 2 2 2 2]

+---------------------------------------+
|      Classifying with HOF - Gabor     |
+---------------------------------------+
Gabor Detector with HOF descriptor and sigma = 4, tau = 1.5

       Accuracy: 75.00%
       Prediction: [1 0 0 0 2 2 1 1 2 2 2 2]

+---------------------------------------+
|   Classifying with HOG/HOF - Harris   |
+---------------------------------------+
Harris Detector with HOG/HOF descriptor and sigma = 4, tau = 1.5

       Accuracy: 100.00%
       Prediction: [0 0 0 0 1 1 1 1 2 2 2 2]

+---------------------------------------+
|    Classifying with HOG/HOF - Gabor   |
+---------------------------------------+
Gabor Detector with HOG/HOF descriptor and sigma = 4, tau = 1.5

       Accuracy: 83.33%
       Prediction: [0 1 0 0 0 1 1 1 2 2 2 2]
```

Τα αποτελέσματα φαίνονται και εδώ:

Detectors & Descriptors	Accuracy	Predictions
HOG - Harris	91.67%	[0 0 0 0 2 1 1 1 2 2 2 2]
HOG - Gabor	100.00%	[0 0 0 0 1 1 1 1 2 2 2 2]
HOF - Harris	100.00%	[0 0 0 0 1 1 1 1 2 2 2 2]
HOF - Gabor	75.00%	[1 0 0 0 2 2 1 1 2 2 2 2]
HOG/HOF - Harris	100.00%	[0 0 0 0 1 1 1 1 2 2 2 2]
HOG/HOF - Gabor	83.33%	[0 1 0 0 0 1 1 1 2 2 2 2]

Ενώ οι ετικέτες του test set είναι:

[0 0 0 0 1 1 1 1 2 2 2 2]

Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

18/06/2023

Μπορούμε να παρατηρήσουμε γενικά ότι το handwaving αντιστοιχεί στην πιο ακρίβη αναπαράσταση BoVW, αφού κανένας ανιχνευτής δεν έκανε λάθος στο classification. Δεύτερη καλύτερη κατηγοριοποίηση είχαμε για τα walking videos με μόλις 2/24 λανθασμένα classifications. Τελευταία έρχεται η κατηγοριοποίηση για τα running videos με 4/24 λανθασμένα classifications.

Μια ακόμα παρατήρηση είναι ότι misclassifications συμβαίνουν κυρίως μεταξύ των κλάσεων walking και running, μια και είναι πολύ παρεμφερείς.

Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

18/06/2023

Μέρος 3: Συνένωση Εικόνων (Image Stitching) για Δημιουργία Πανοράματος

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from numpy import unravel_index
from numpy.linalg import norm
%matplotlib inline

global imageNo
imageNo=0

!mkdir imageStitching
```

```
A subdirectory or file imageStitching already exists.
```

Διαβάζουμε τις εικόνες προς συρραφή

```
I1 = cv2.imread("part3 - ImageStitching/img1_ratio01.jpg")
I2 = cv2.imread("part3 - ImageStitching/img2_ratio01.jpg")
I3 = cv2.imread("part3 - ImageStitching/img3_ratio01.jpg")
I4 = cv2.imread("part3 - ImageStitching/img4_ratio01.jpg")
I5 = cv2.imread("part3 - ImageStitching/img5_ratio01.jpg")
I6 = cv2.imread("part3 - ImageStitching/img6_ratio01.jpg")

I1 = cv2.cvtColor(I1, cv2.COLOR_BGR2RGB)
I2 = cv2.cvtColor(I2, cv2.COLOR_BGR2RGB)
I3 = cv2.cvtColor(I3, cv2.COLOR_BGR2RGB)
I4 = cv2.cvtColor(I4, cv2.COLOR_BGR2RGB)
I5 = cv2.cvtColor(I5, cv2.COLOR_BGR2RGB)
I6 = cv2.cvtColor(I6, cv2.COLOR_BGR2RGB)


fig, ax = plt.subplots(2, 3, figsize = (10,10))
ax[0][0].imshow(I1)
ax[0][1].imshow(I2)
ax[0][2].imshow(I3)
ax[0][0].set_title("Image 1")
ax[0][1].set_title("Image 2")
ax[0][2].set_title("Image 3")
ax[1][0].imshow(I4)
ax[1][1].imshow(I5)
ax[1][2].imshow(I6)
ax[1][0].set_title("Image 4")
ax[1][1].set_title("Image 5")
```

```
ax[1][2].set_title("Image 6")
plt.tight_layout()

title='imageStitching/'
name=str(imageNo)
plt.savefig(title+name,facecolor='w')
imageNo=imageNo+1
```

Η συνάρτηση perspectiveTransform λαμβάνει έναν πίνακα προοπτικού μετασχηματισμού και ένα σύνολο sourcePoints και εφαρμόζει τον προοπτικό μετασχηματισμό σε αυτά, επιστρέφοντας τα αντίστοιχα μετασχηματισμένα σημεία σε καρτεσιανές συντεταγμένες.

```
def perspectiveTransform(perspectiveMatrix, sourcePoints):
    # Add ones to the source points to make them homogeneous coordinates
    augment = np.ones((sourcePoints.shape[0],1))
```

```
    # Apply the perspective transformation
    projective_corners = np.concatenate( (sourcePoints, augment), axis=1).T

    # Convert the projective points to Cartesian coordinates
    projective_points = perspectiveMatrix.dot(projective_corners)

    # Extract the x and y coordinates of the transformed points
    target_points = np.true_divide(projective_points, projective_points[-1])

    return target_points[:2].T
```

Η συνάρτηση warpImages λαμβάνει δύο εικόνες εισόδου και έναν πίνακα μετασχηματισμού και εκτελεί τις ακόλουθες λειτουργίες:

1. στρεβλώνει την img2 με βάση τον πίνακα μετασχηματισμού H,
2. την ευθυγραμμίζει με την img1 και
3. τις συρράπτει μαζί για να δημιουργήσει μια ενιαία εικόνα εξόδου.

```
def warpImages(img1, img2, H):
    global imageNo

    # Get the dimensions of the input images
    img1Rows, img1Cols = img1.shape[:2]
    img2Rows, img2Cols = img2.shape[:2]

    # Define the corner points of the reference image (img1)
    cornerPoints1 = np.float32([[0,0], [0, img1Rows], [img1Cols, img1Rows],
[img1Cols, 0]])

    # Define the corresponding corner points of the warped image (img2) using
perspective transformation
    cornerPoints2 = np.float32([[0,0], [0, img2Rows], [img2Cols, img2Rows],
[img2Cols, 0]])
    cornerPoints2 = perspectiveTransform(H, cornerPoints2)

    # Concatenate the corner points of both images
    list_of_points = np.concatenate((cornerPoints1, cornerPoints2), axis=0)

    # Calculate the minimum and maximum coordinates for the merged image
    [x_min, y_min] = np.int32(list_of_points.min(axis=0).ravel() - 0.5)
    [x_max, y_max] = np.int32(list_of_points.max(axis=0).ravel() + 0.5)

    # Calculate the translation distance for aligning the warped image
    translation_dist = [x_min, y_min]
```

```
    # Create an output image with the calculated dimensions
    imgToReturn = np.zeros((y_max - y_min, x_max - x_min, 3), dtype=np.uint8)

    # Warp the pixels from img2 and assign them to the corresponding
coordinates in the output image
    for i in range(0, img2Rows):
        for j in range(0, img2Cols):
            img2WarpedPixels = np.round(perspectiveTransform(H,
np.float32([[j, i]])))
            y = int(img2WarpedPixels[0][1] + abs(translation_dist[1]))
            x = int(img2WarpedPixels[0][0] + abs(translation_dist[0]))
            # Clamp the coordinates to the valid range of the output image
            if y > 813:
                y = 813
            if x > 813:
                x = 813
            if y < 0:
                y = 0
            if x < 0:
                x = 0
            imgToReturn[y, x] = img2[i, j]

    imgToReturn = imgToReturn.astype(np.uint8)
    plt.imshow(imgToReturn)
    plt.title("Image 1 warped")
    base = 'imageStitching/'
    name = str(imageNo)
    plt.savefig(title + name)
    imageNo += 1
    plt.show()

    # Overlay the pixels from img1 onto the corresponding coordinates in the
output image
    for i in range(0, img1Rows):
        for j in range(0, img1Cols):
            s = img1[i, j][0] + img1[i, j][1] + img1[i, j][2]
            if s != 0:
                imgToReturn[i + abs(translation_dist[1]), j +
abs(translation_dist[0])] = img1[i, j]

    imgToReturn = imgToReturn.astype(np.uint8)
    plt.imshow(imgToReturn)
    plt.title("Stitched image")
    base = 'imageStitching/'
    name = str(imageNo)
    plt.savefig(title + name)
    imageNo += 1
    plt.show()
```

```
    return imgToReturn
```

Εκτελούμε τη συρραφή εικόνων:

1. Εντοπίζουμε και ταυτοποιούμε τα SIFT χαρακτηριστικά μεταξύ δύο εικόνων
2. Εφαρμόζουμε το Lowe ratio test για να φιλτράρουμε τις αντιστοιχίες, με κατώφλι 0.8, ενδεικτικά, όπως προτείνεται
3. Εκτιμάμε τον πίνακα ομογραφίας χρησιμοποιώντας το RANSAC
4. Στρεβλώνουμε και συρράφουμε τις εικόνες μεταξύ τους.

Το Lowe ratio test είναι μια μέθοδος που χρησιμοποιείται στην αντιστοίχιση χαρακτηριστικών για το φιλτράρισμα πιθανών εσφαλμένων αντιστοιχιών. Συγκρίνει την απόσταση μεταξύ των δύο καλύτερων αντιστοιχιών για κάθε σημείο χαρακτηριστικών και εφαρμόζει ένα κατώφλι. Εάν ο λόγος των αποστάσεων είναι κάτω από το κατώφλι, η αντιστοίχιση θεωρείται έγκυρη.

Ο RANSAC (RANdom SAmple Consensus) είναι ένας επαναληπτικός αλγόριθμος που εντοπίζει τα outliers ενός συνόλου δεδομένων και εκτιμά το επιθυμητό μοντέλο βάσει των υπόλοιπων δεδομένων. Εφαρμόζουμε το υποψήφιο μοντέλο σε έναν τυχαίο αριθμό δεδομένων. Αν έχουμε υπερβολικά πολλά outliers (αποφασίζουμε με βάση κάποιο threshold που είναι είσοδος στον αλγόριθμο) προσαρμόζουμε το μοντέλο στα inliers και ο αλγόριθμος τερματίζει μετά από προκαθορισμένο αριθμό επαναλήψεων.

```
def image_stiching(I1, I2):
    global imageNo

    # Step 1: Extract features using SIFT
    sift = cv2.xfeatures2d.SIFT_create()
    key_1, descr_1 = sift.detectAndCompute(I1, None)
    feats_1 = cv2.drawKeypoints(I1, key_1, 0)
    key_2, descr_2 = sift.detectAndCompute(I2, None)
    feats_2 = cv2.drawKeypoints(I2, key_2, 0)

    # Display feature images
    fig, ax = plt.subplots(1, 2, figsize=(10, 30))
    ax[0].imshow(feats_1)
    ax[1].imshow(feats_2)
    ax[0].set_title("Features of first image")
    ax[1].set_title("Features of second image")
    title = 'imageStitching/'
    name = str(imageNo)
    plt.savefig(title + name)
    imageNo = imageNo + 1
    plt.show()

    originalI1 = I1
    originalI2 = I2

    # Step 2: Match features using Brute Force Matcher
    bf = cv2.BFMatcher()
```

```
    matches = bf.knnMatch(descr_1, descr_2, k=2)
    I2 = cv2.drawMatchesKnn(originalI1, key_1, originalI2, key_2, matches,
None, flags=2)
    plt.imshow(I2)
    plt.title("Brute Force Matching")
    title = 'imageStitching/'
    name = str(imageNo)
    plt.savefig(title + name)
    imageNo = imageNo + 1
    plt.show()

    # Step 3: Apply Lowe's ratio test to filter matches
    thres = 0.8
    good = [[m] for m, n in matches if m.distance < thres * n.distance]
    matchesAfterLowe = [m for m, n in matches if m.distance < thres *
n.distance]
    print("Lowe Threshold rejected:", np.shape(matches)[0] -
np.shape(matchesAfterLowe)[0], "'bad' features!")
    I12_lowe = cv2.drawMatches(originalI1, key_1, originalI2, key_2,
matchesAfterLowe, None, flags=2)
    plt.imshow(I12_lowe)
    plt.title("After Lowe Threshold:")
    title = 'imageStitching/'
    name = str(imageNo)
    plt.savefig(title + name)
    imageNo = imageNo + 1
    plt.show()

    # Step 4: Find Homography matrix using RANSAC
    query_pts = np.float32([key_1[m.queryIdx].pt for m in
matchesAfterLowe]).reshape(-1, 1, 2)
    train_pts = np.float32([key_2[m.trainIdx].pt for m in
matchesAfterLowe]).reshape(-1, 1, 2)
    HBA, mask = cv2.findHomography(query_pts, train_pts, cv2.RANSAC, 4.0)
    inv_HBA = np.linalg.inv(HBA)
    print("Homography Transformation:\n", HBA)

    # Warp and stitch images
    return warpImages(originalI2, originalI1, HBA)
```

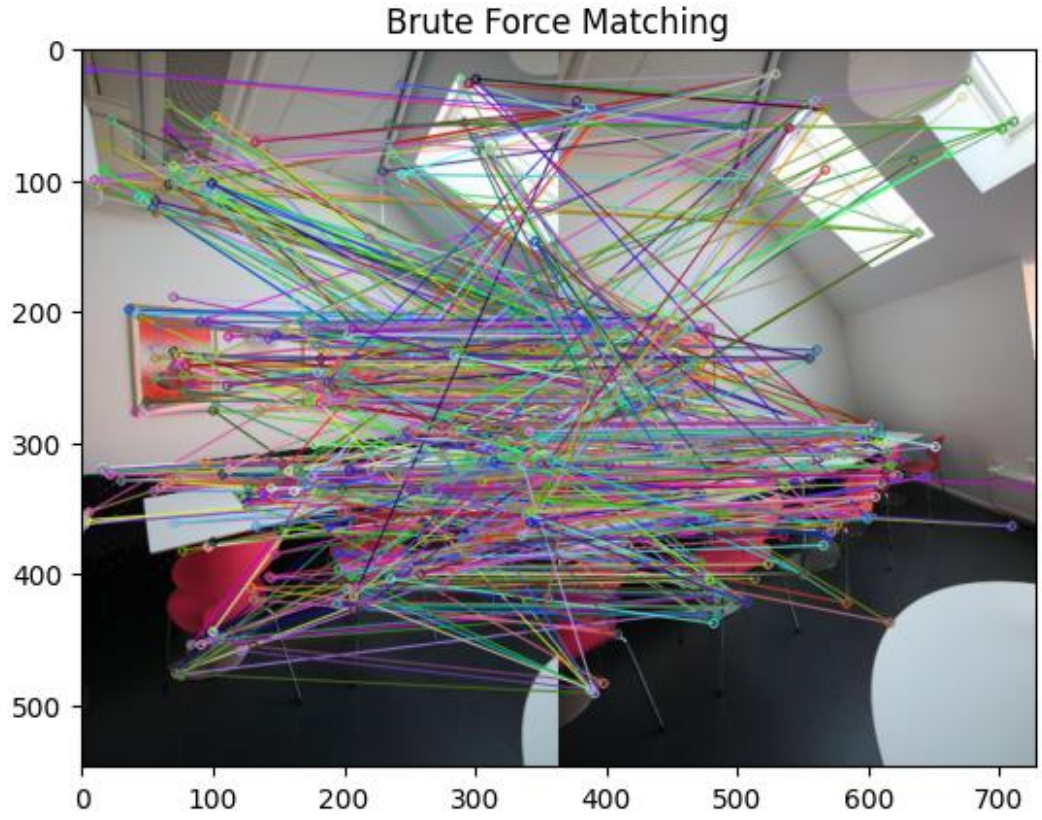
Εφαρμόζουμε τη συνάρτηση image_stitching και παρατηρούμε στα παρακάτω διαγράμματα:

1. Την εξαγωγή χαρακτηριστικών από τις εικόνες εισόδου (αλγόριθμος SIFT)
2. Την αντιστοίχιση των keypoints μεταξύ των δύο εικόνων χρησιμοποιώντας τον Brute Force Matcher.
3. Την παραμόρφωση της μίας εικόνας χρησιμοποιώντας τον πίνακα ομογραφίας που εξήγαγε.
4. Το συνδυασμό της παραμορφωμένης εικόνας με την άλλη με αποτέλεσμα τη δημιουργία της τελικής συρραμμένης εικόνας.

```
I12w=np.array(image_stiching(I1,I2))
```

Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

18/06/2023



Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

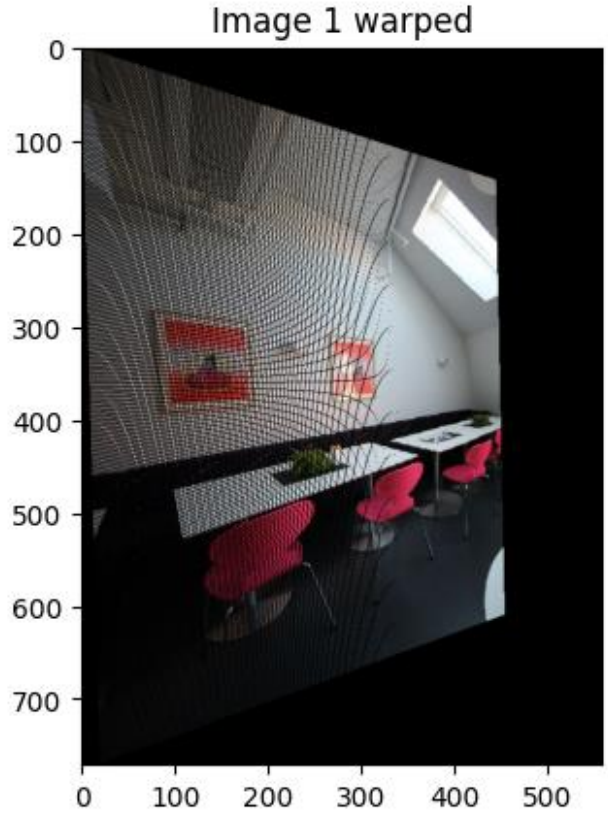
18/06/2023

```
Lowe Threshold rejected: 202 'bad' features!
```


```
Homography Transformation:
 [[ 1.67294848e+00  3.36722163e-02 -1.95036181e+02]
 [ 4.59915351e-01  1.42124180e+00 -1.05581940e+02]
 [ 1.79747527e-03  1.90094015e-05  1.00000000e+00]]
```

Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

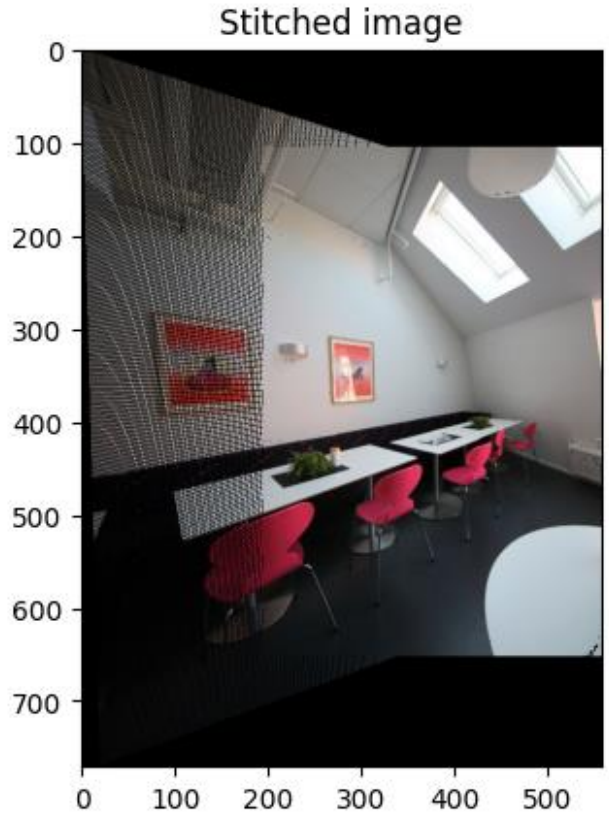
18/06/2023




```
c:\Users\steli\anaconda3\envs\cv_lab1_env\lib\site-
packages\ipykernel_launcher.py:57: RuntimeWarning: overflow encountered in
ubyte_scalars
```

Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

18/06/2023

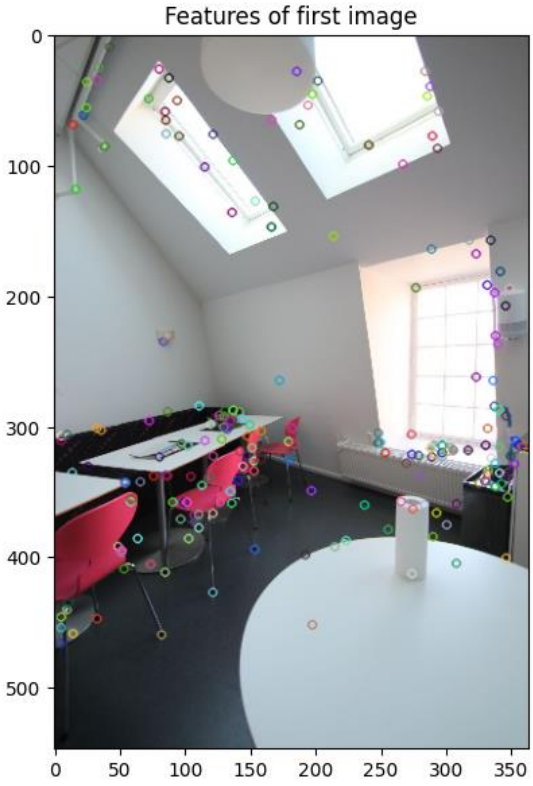


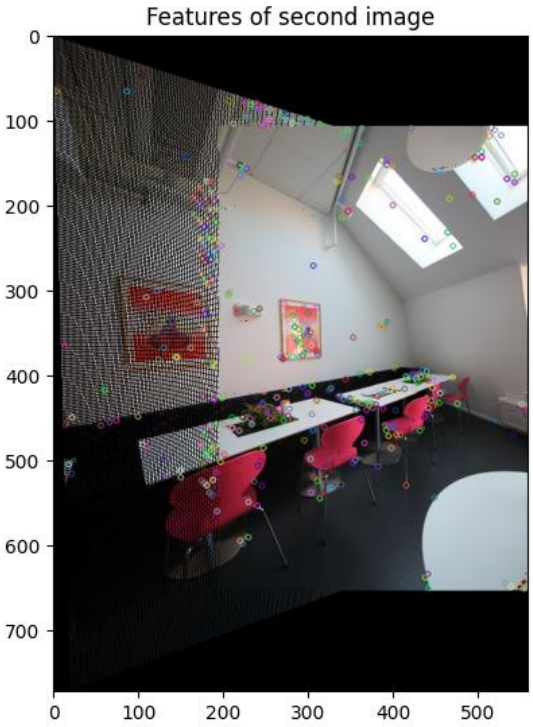
Εφαρμόζουμε τη συνάρτηση image_stitching πάλι, μεταξύ της προηγούμενης warped εικόνας και της τρίτης σε σειρά εικόνας.

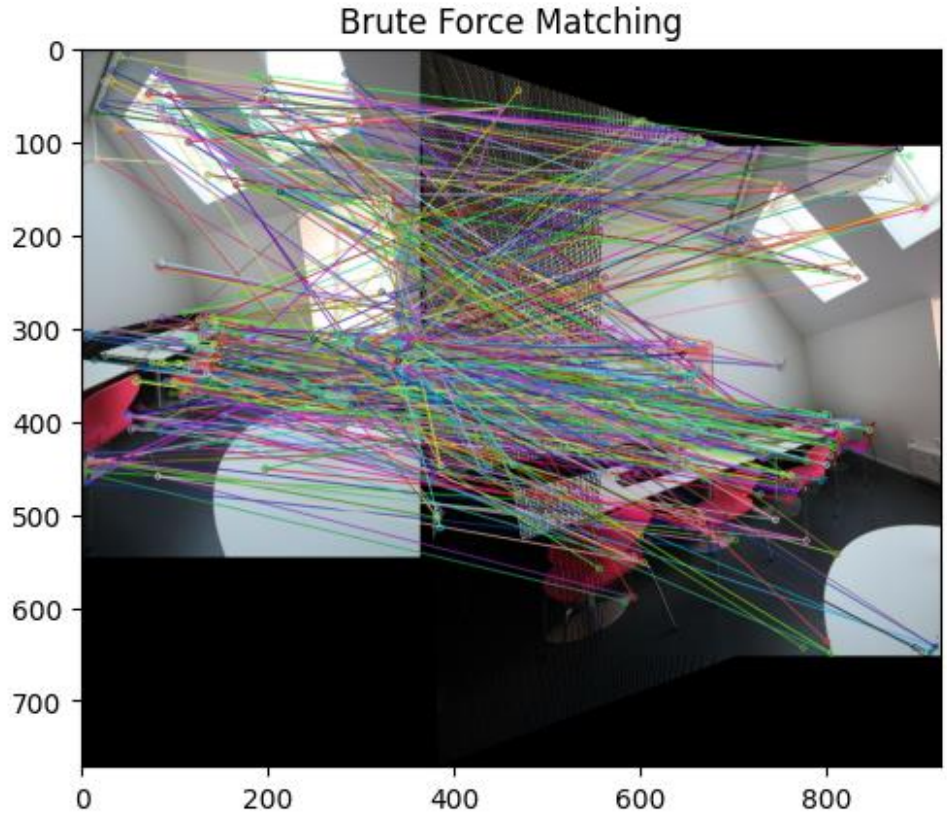
```
I123w=np.array(image_stiching(I3,I12w))
```

Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

18/06/2023



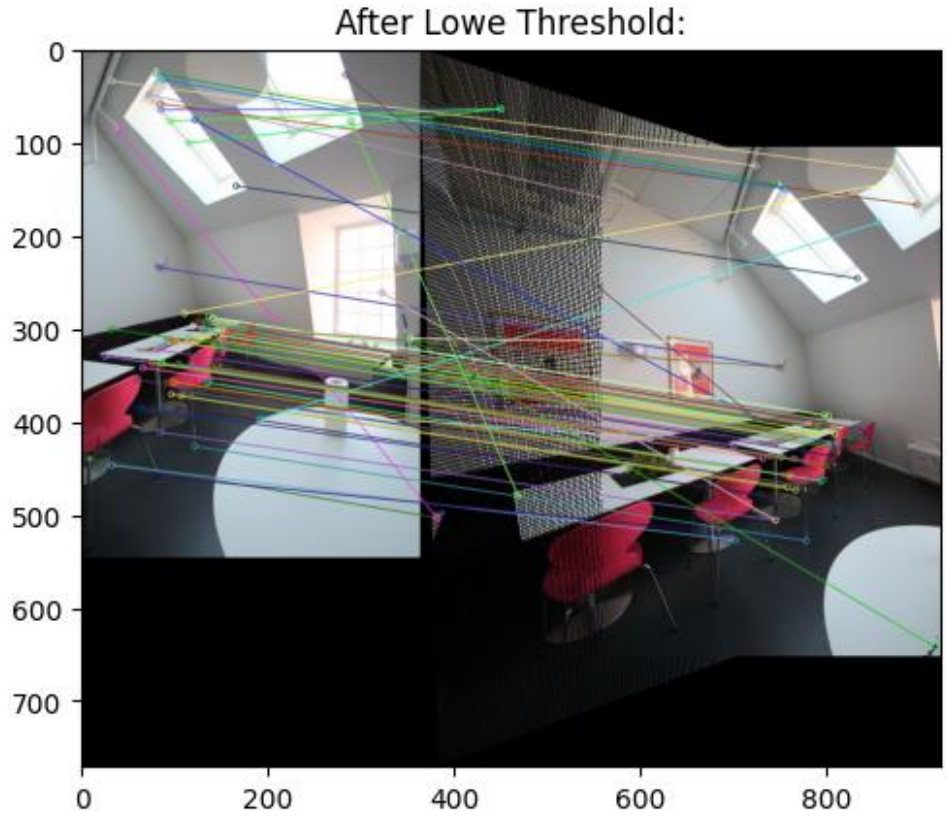




Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

18/06/2023

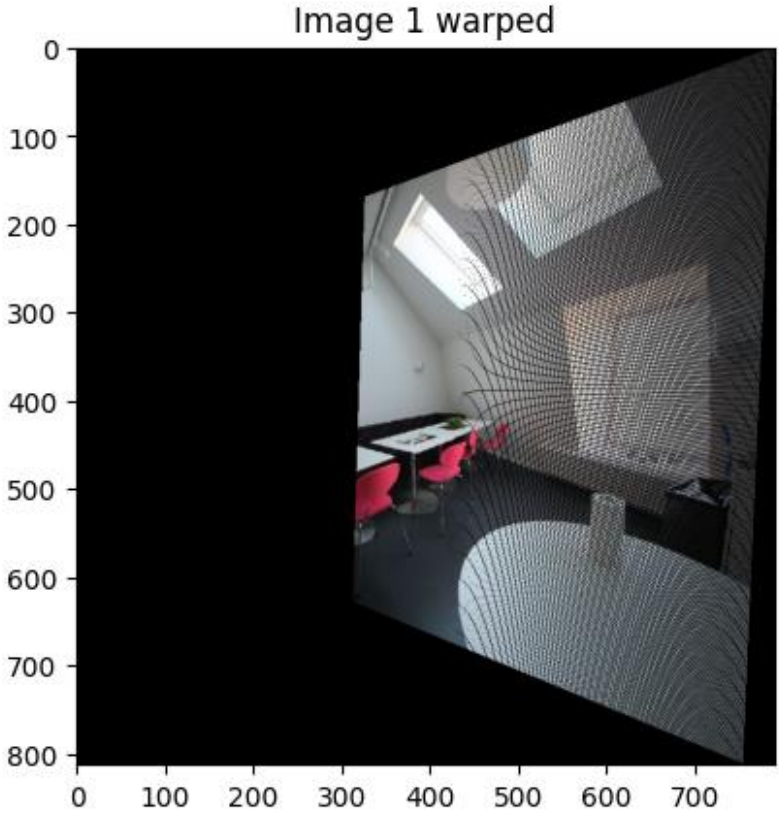
```
Lowe Threshold rejected: 162 'bad' features!
```




```
Homography Transformation:
 [[ 3.22880266e-01 -1.66219948e-02  3.26691559e+02]
 [-4.37400418e-01  8.54825598e-01  1.45296102e+02]
 [-1.20497708e-03  2.63858800e-05  1.00000000e+00]]
```

Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

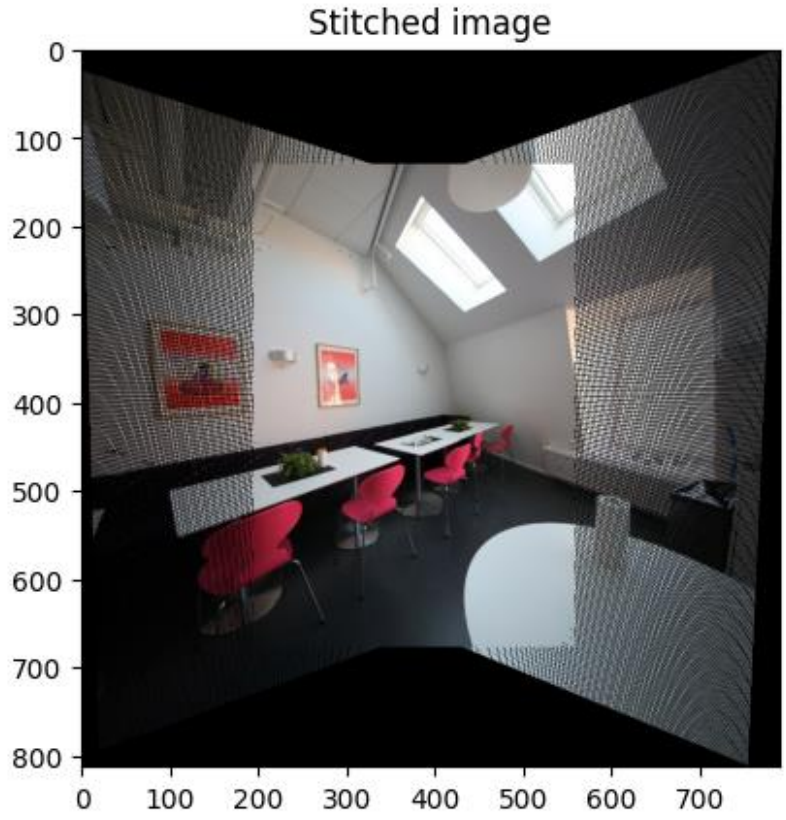
18/06/2023




```
c:\Users\steli\anaconda3\envs\cv_lab1_env\lib\site-
packages\ipykernel_launcher.py:57: RuntimeWarning: overflow encountered in
ubyte_scalars
```

Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

18/06/2023

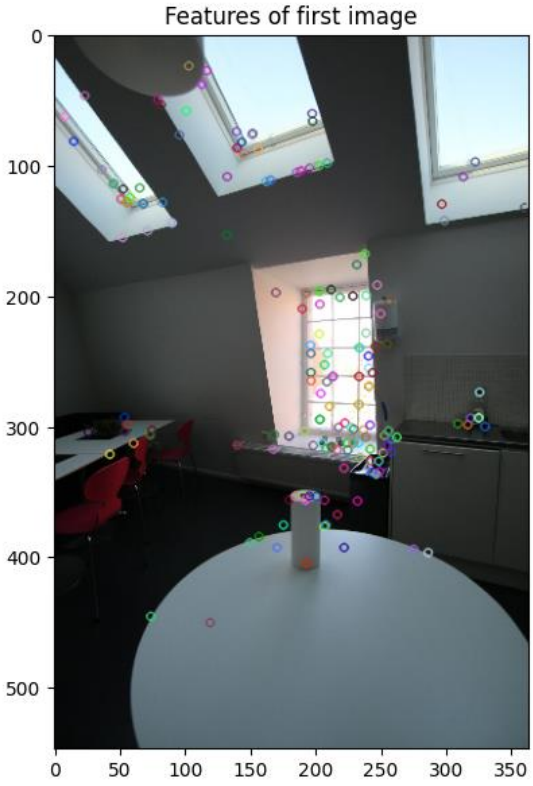


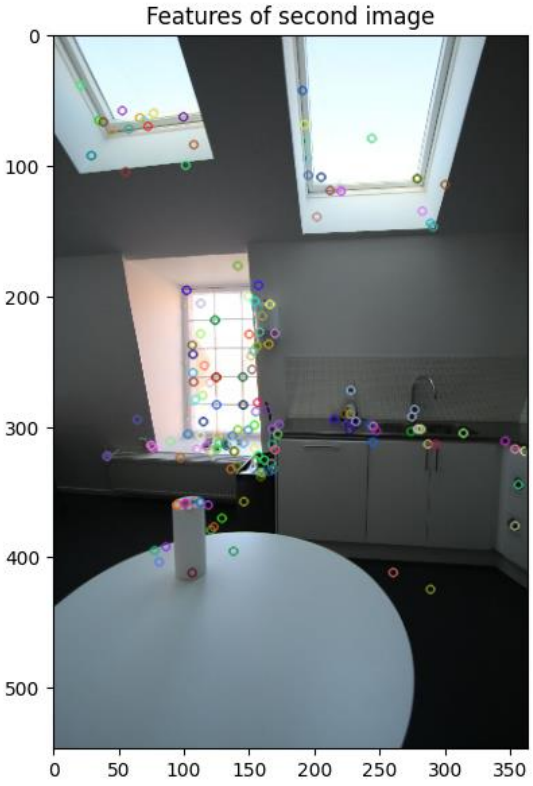
Εντελώς συμμετρικά, εφαρμόζουμε την image_stitching στις εικόνες 4 και 5.

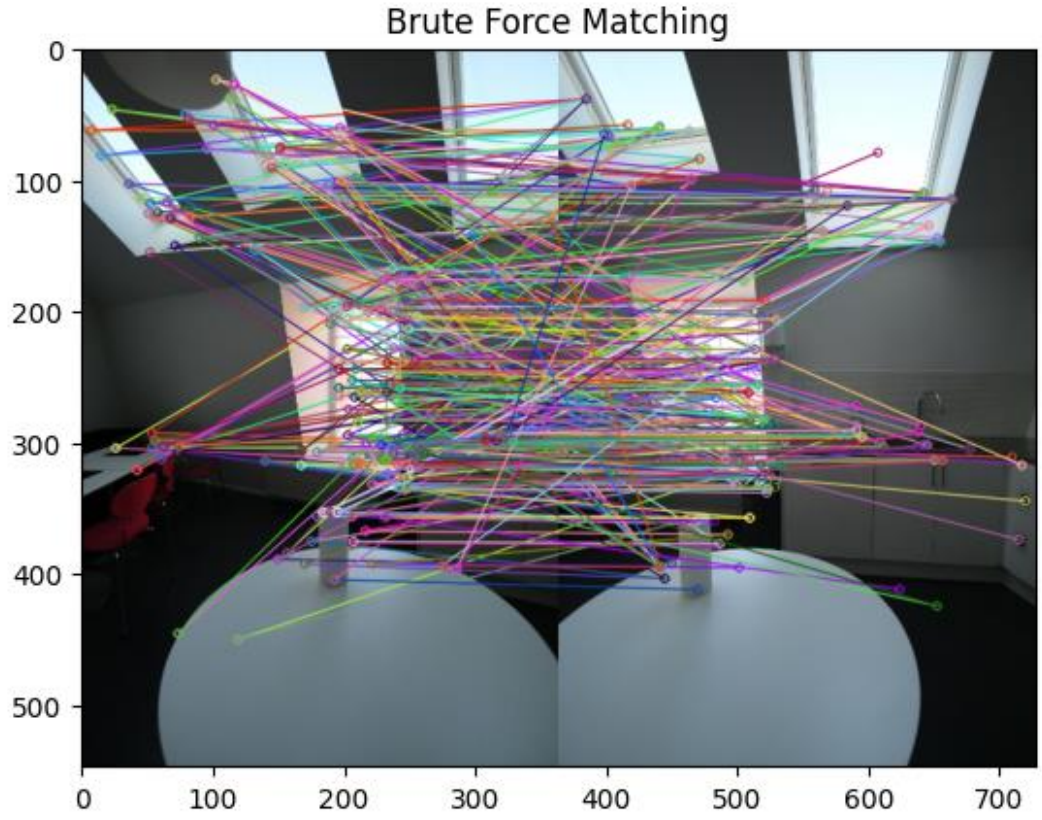
```
I45w=np.array(image_stiching(I4,I5))
```

Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

18/06/2023



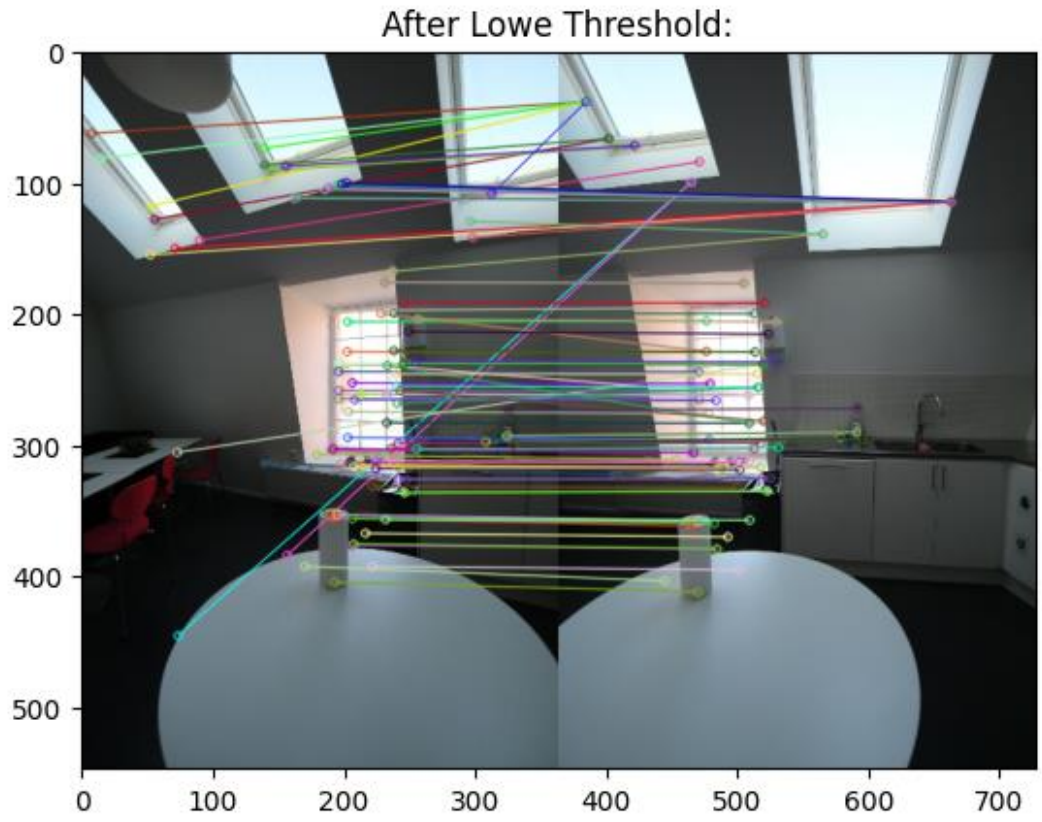




Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

18/06/2023

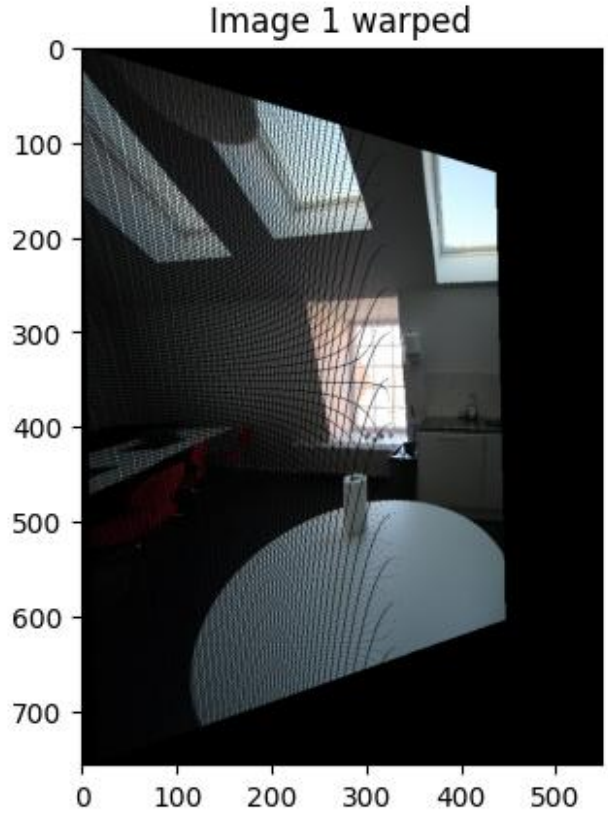
```
Lowe Threshold rejected: 84 'bad' features!
```




```
Homography Transformation:
 [[ 1.61827762e+00  2.82894751e-02 -1.83994463e+02]
 [ 4.24860101e-01  1.37983653e+00 -9.69518557e+01]
 [ 1.66112379e-03 -6.16016203e-06  1.00000000e+00]]
```

Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

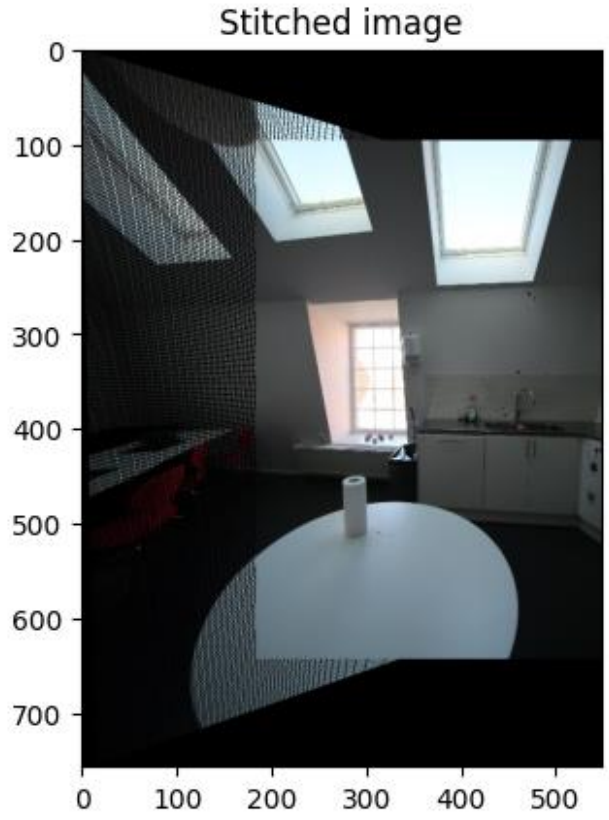
18/06/2023




```
c:\Users\steli\anaconda3\envs\cv_lab1_env\lib\site-
packages\ipykernel_launcher.py:57: RuntimeWarning: overflow encountered in
ubyte_scalars
```

Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

18/06/2023

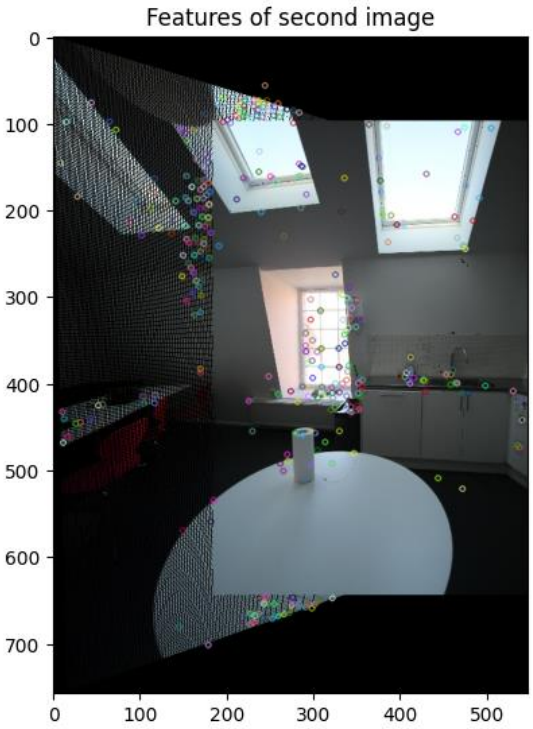


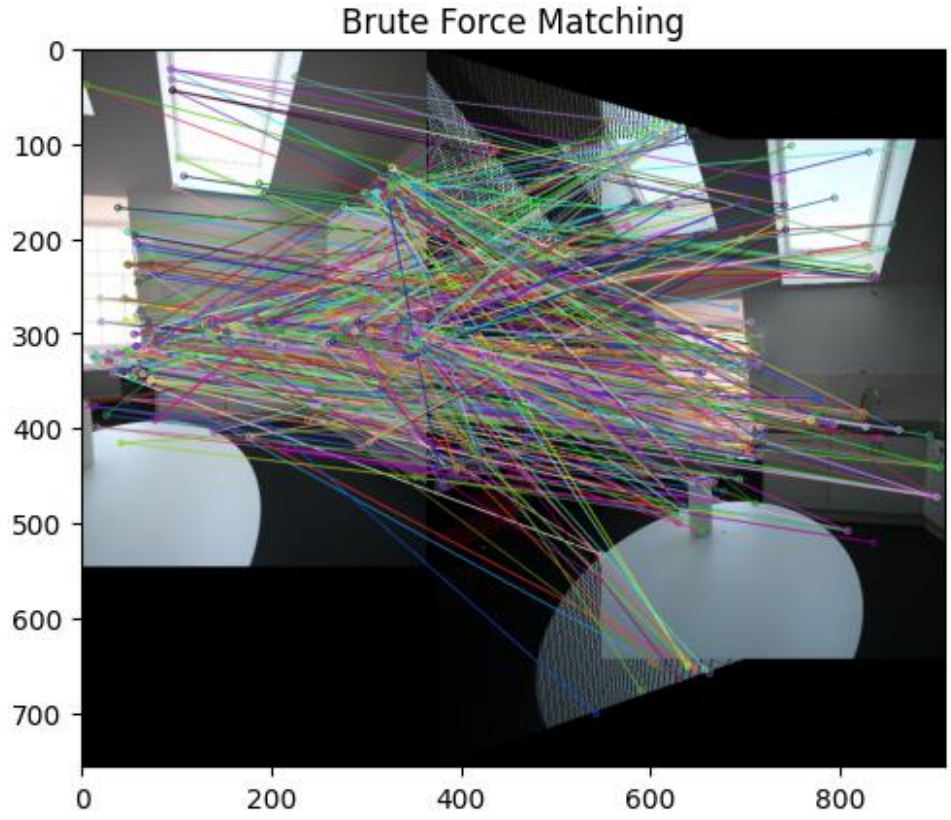
Όπως πριν, ενώνουμε τις 3 δεξιές εικόνες.

```
I456w=np.array(image_stiching(I6,I45w))
```

Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

18/06/2023

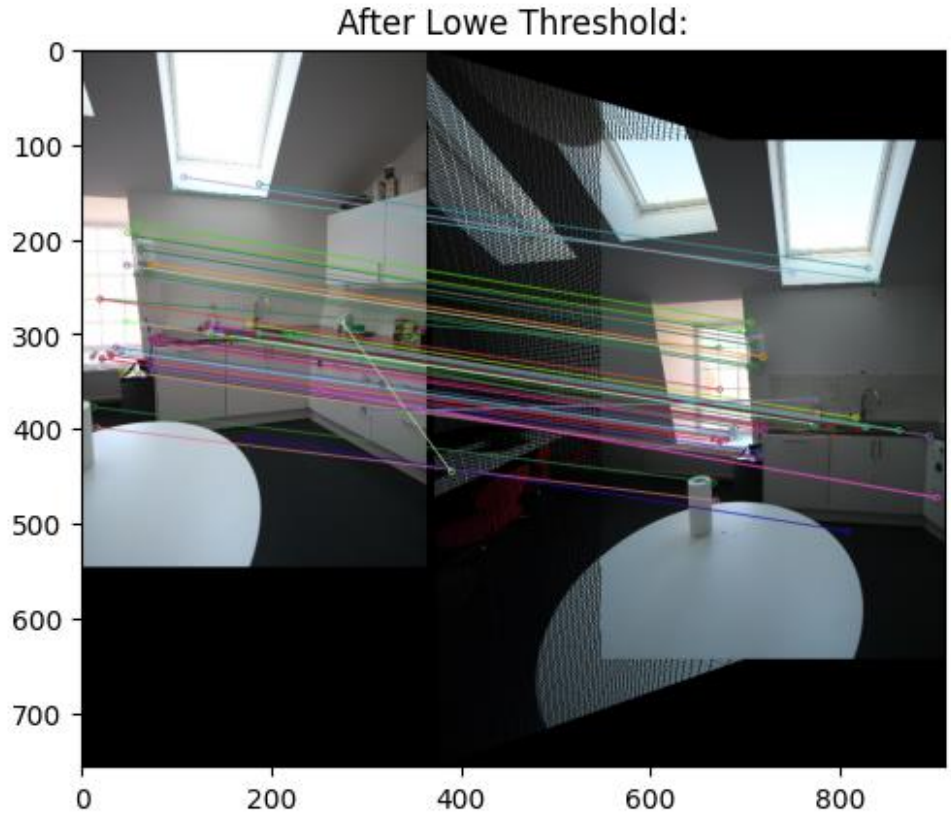




Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

18/06/2023

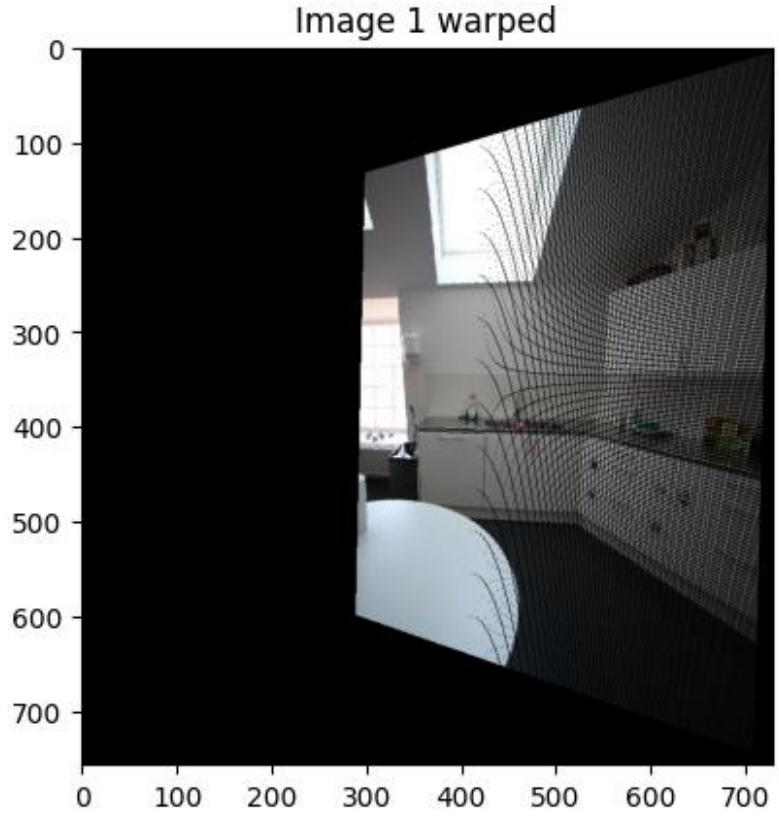
```
Lowe Threshold rejected: 201 'bad' features!
```

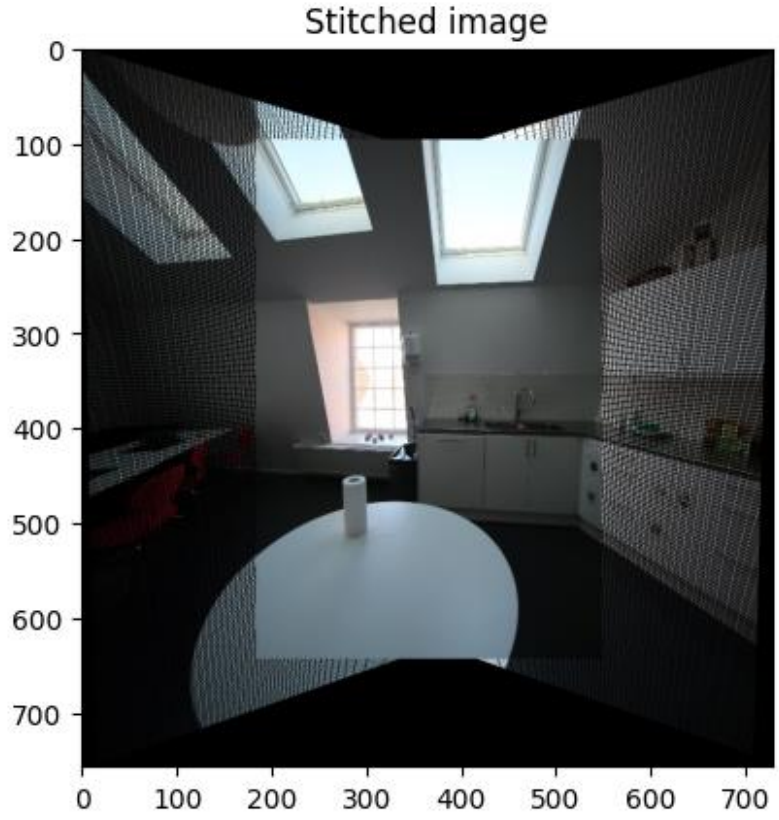



```
Homography Transformation:
 [[ 4.35922728e-01 -1.45277737e-02  2.98656390e+02]
 [-3.56591033e-01  8.58915371e-01  1.33388239e+02]
 [-1.02103451e-03  1.07444503e-05  1.00000000e+00]]
```

Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

18/06/2023



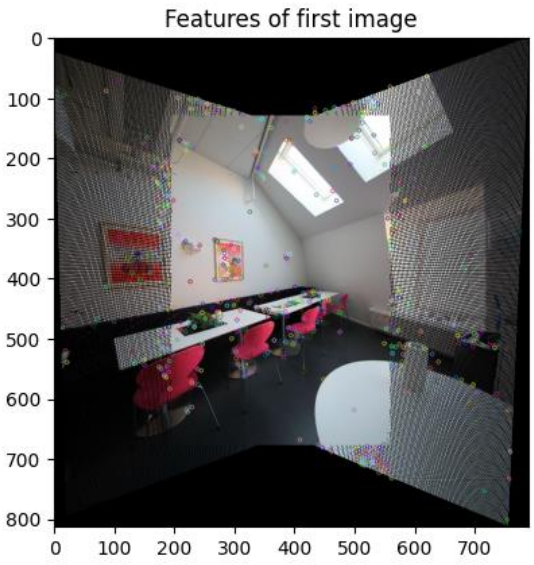


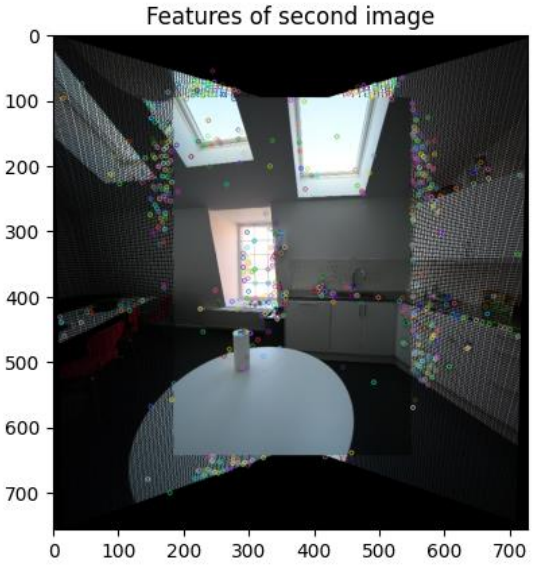
Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

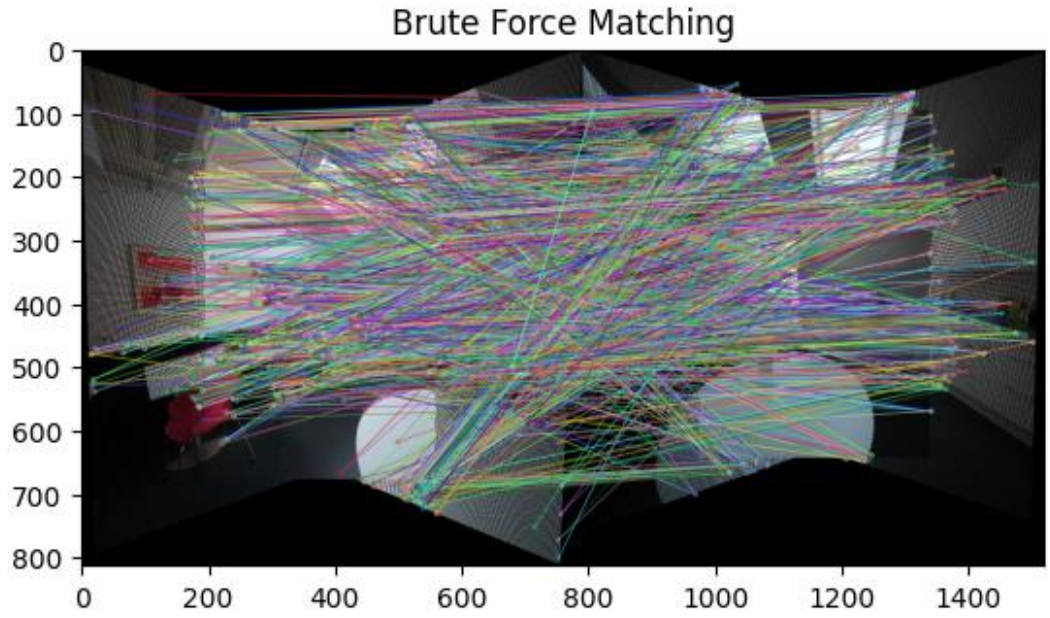
18/06/2023

Τέλος, εφαρμόζουμε μια τελευταία φορά την image_stitching για να ενώσουμε τις 2 τριάδες στην τελική εικόνα!

```
stitched = np.array(image_stiching(I123w,I456w))
```



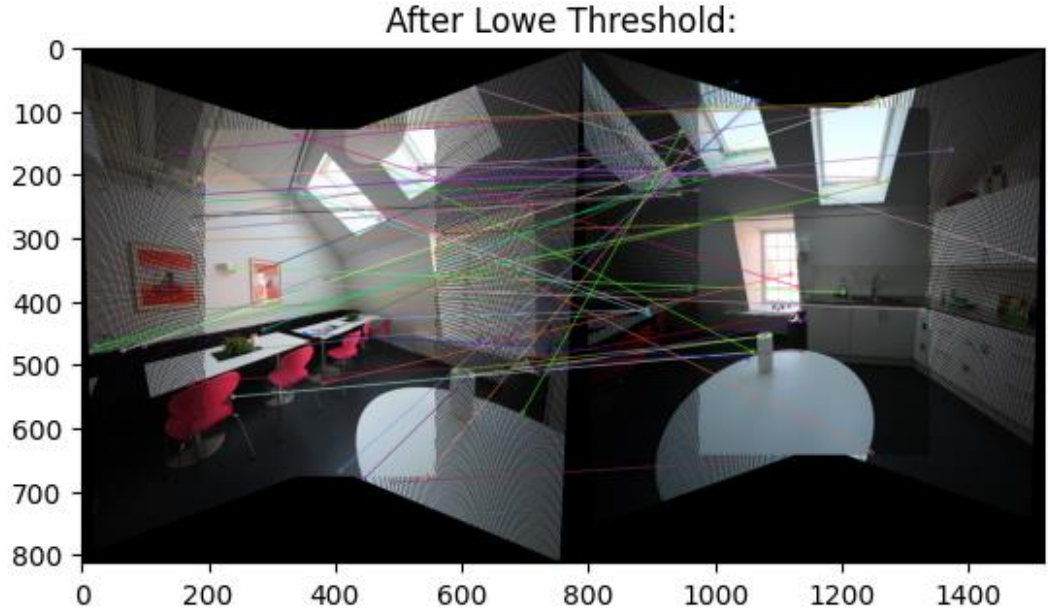




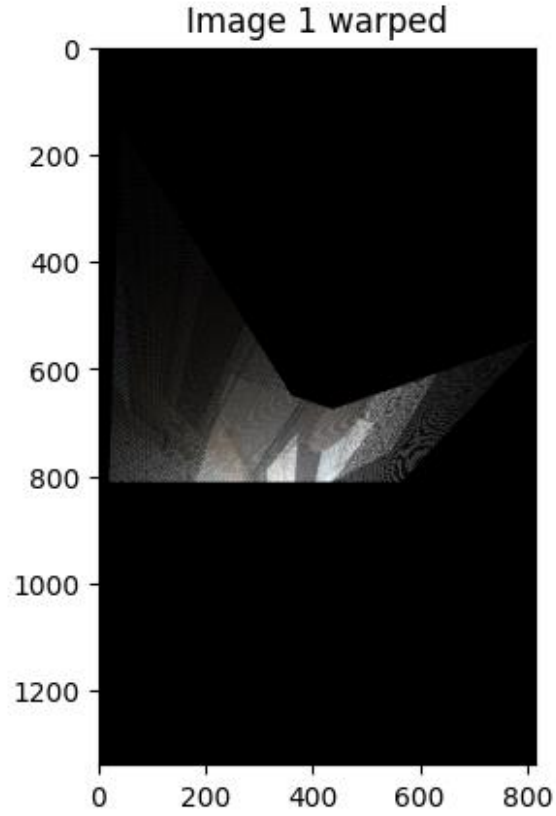

```
Lowe Threshold rejected: 604 'bad' features!
```

Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

18/06/2023



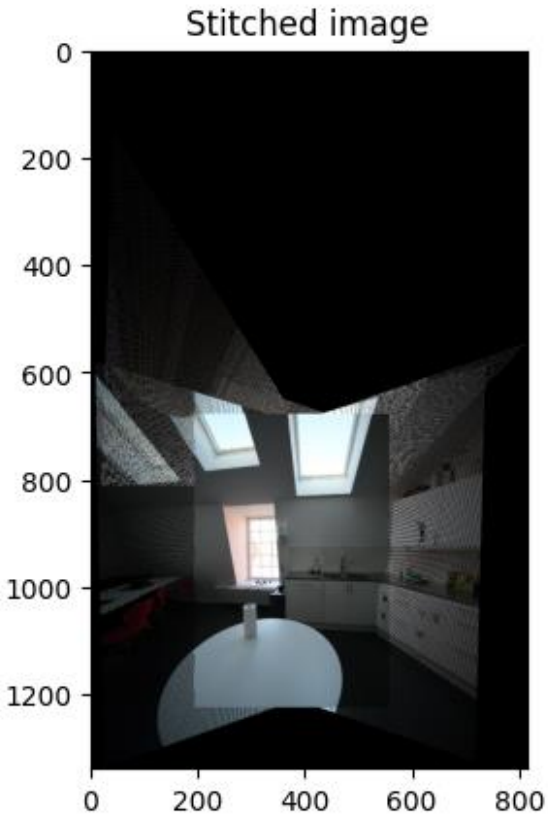

```
Homography Transformation:
 [[ 2.13357870e+00 -1.00662844e-01  3.91696525e+01]
 [ 6.40398953e-01  3.94750732e+00 -5.81804064e+02]
 [ 1.44687213e-03  4.60142966e-03  1.00000000e+00]]
```



Ζαρίφης Στέλιος A.M.: el20435
Ζαφείριος Αστρεινίδης A.M.: el19053

18/06/2023

Τελική stitched εικόνα:



Παρατηρήσαμε πως ήταν προτιμώτερο το stitching να γινει με τη σειρά εικόνων:

$$\{[1-2],[12-3],[4-5],[45-6],[123-456]\}$$

Γιατί έτσι έχουμε την ελάχιστη δυνατή παραμόρφωση (σε άλλη περίπτωση, ο μετασχηματισμός H θα εφαρμοζόταν ασύμμετρα και πολλές φορές σε συγκεκριμένες εικόνες)!