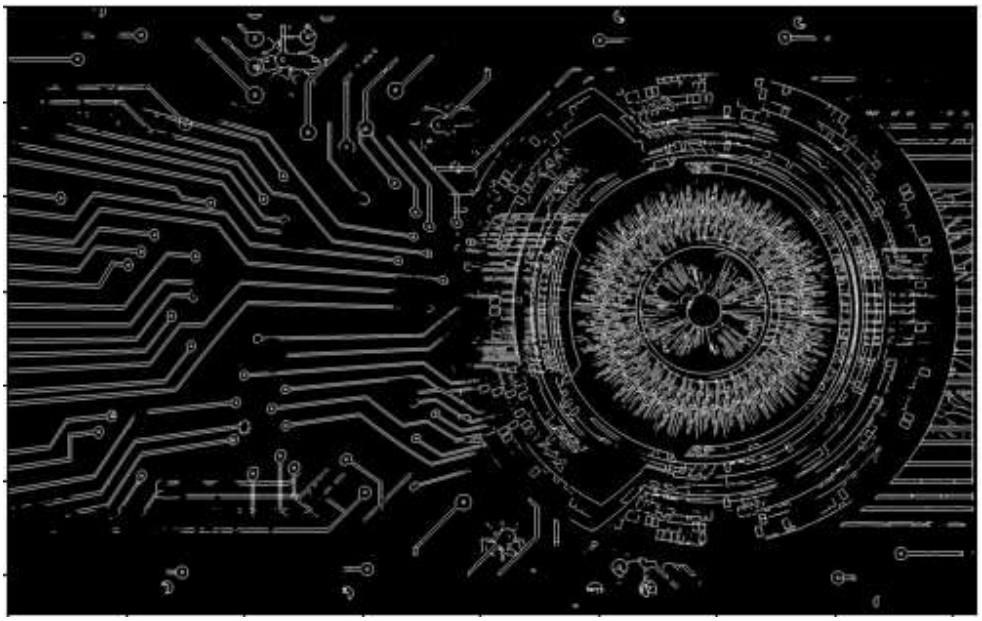
Όραση Υπολογιστών
Εργαστηριακή Άσκηση 2:



Θέμα: Εκτίμηση Οπτικής Ροής (Optical Flow) και Εξαγωγή Χαρακτηριστικών σε Βίντεο για Αναγνώριση Δράσεων

Βασιλάκος Μιχαήλ: 03117069
Γιαννούλης Παναγιώτης: 03117812

3 Ιουνίου 2021

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

Στο πρώτο μέρος της εργαστηριακής άσκησης έχουμε ως στόχο την δημιουργία ενός συστήματος παρακολούθησης προσώπου και χεριών σε μια ακολουθία βίνετο νοηματικής γλώσσας. Ως πρώτο βήμα θα χρειαστεί να ανιχνεύσουμε τις περιοχές του προσώπου και των χεριών (3 πλαίσια) με χρήση ενός πιθανοτικού ανιχνευτή ανθρώπινου δέρματος. Έπειτα, θα μπορέσουμε να παρακολουθήσουμε τις περιοχές αυτές ενδιαφέροντος χρησιμοποιώντας τα εξαγόμενα διανύσματα οπτικής ροής, τα οποία θα έχουμε υπολογισεί με τη μέθοδο των Lucas-Kanade.

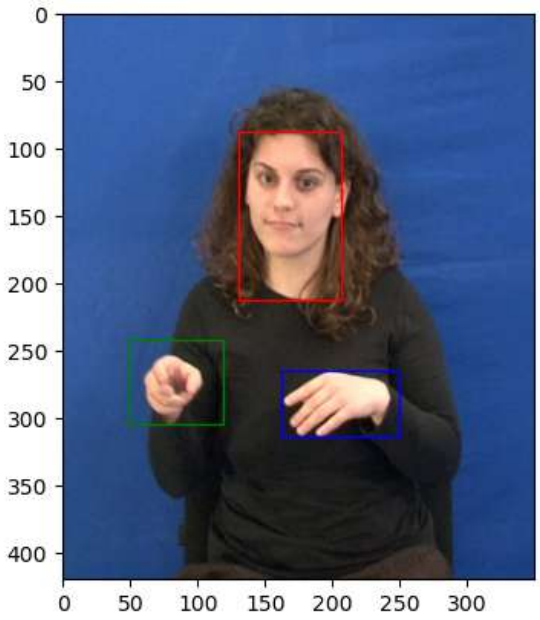


Figure 1: *Ανίχνευση προσώπου και χεριών*

1.1 Ανίχνευση Δέρματος Προσώπου και Χεριών

Στο πρώτο ερώτημα θέλουμε να ανιχνεύσουμε τα σημεία του δέρματος στην εικόνα και κατ' επέκταση τις 3 περιοχές των χεριών και του προσώπου. Για να το επιτύχουμε αυτό θα χρησιμοποιήσουμε τον χρωματικό χώρο YCbCr από τον οποίο θα αφαιρέσουμε το πεδίο της φωτεινότητας Y. Ουασιαστικά, μας ενδιαφέρει μόνο η πληροφορία του χρώματος γι' αυτό και επιλέξαμε αυτόν τον χρωματικό χώρο ώστε να μπορέσουμε εύκολα να αγνοήσουμε τη φωτεινότητα. Γνωρίζουμε ότι τα χρώματα του δέρματος μπορούν να μεντολοποιηθούν από μια δισδιάστατη Γκαουσιανή κατανομή που δίνεται παρακάτω, αρκεί να την εκπαιδεύσουμε

υπολογίζοντας το 1x2 διάνυσμα μέσης τιμής και τον 2x2 πίνακα συνδιακύμανσης Σ.

$$P(c = \text{ skin }) = \frac{1}{\sqrt{|\Sigma|(2\pi)^2}} e^{-\frac{1}{2}(c-\mu)^T \Sigma^{-1}(c-\mu)} \quad (1)$$

Στην παραπάνω κατανομή ως **c** ορίζεται το διάνυσμα τιμών Cb και Cr για κάθε σημείο c (x,y) της εικόνας, **μ** το διάνυσμα 2x1 μέσης τιμής των Cb και Cr, δηλαδή $\mu = \begin{bmatrix} \mu_{C_b} & \mu_{C_r} \end{bmatrix}$ και Σ ο 2 x 2 πίνακας συνδιακύμανσης. Για την εκπαίδευση της γκαουσιανής κατανομής χρησιμοποιήσαμε δείγματα δέρματος που δίνονται στο αρχείο *skinSamplesRGB.mat*.

Με βάση τα παραπάνω και με χρήση της python και των βιβλιοθηκών της opencv, matplotlib και numpy καταφέρνουμε να διαβάσουμε τα δείγματα δέρματος και έπειτα να υπολογίσουμε τη μέση τιμή και την συνδιακύμανση:

```
mat = scipy.io.loadmat('GreekSignLanguage/skinSamplesRGB.mat')
skinSamplesRGB = mat['skinSamplesRGB']
skinSamplesRGB = np.reshape(skinSamplesRGB, (1, 1782, 3))
imgYCC = cv2.cvtColor(skinSamplesRGB, cv2.COLOR_RGB2YCrCb)[0, :, 1:]

# keep only Cr and Cb channels, because Y is for luminance and we don't
    care about it now.
Cr = imgYCC[:, 0]
Cb = imgYCC[:, 1]

# find mean and covariance values of skin samples in order to categorize
    skin color and
# predict skin areas in other images
cr_mean = np.mean(Cr)
cb_mean = np.mean(Cb)
mean_vect = [cb_mean, cr_mean]

cov_matrix = np.cov(np.transpose(imgYCC))
max_val = np.sqrt(np.linalg.det(cov_matrix)) * 2 * 3.14
```

Οι τιμές που προέκυψαν μετά την εκπαίδευση είναι οι εξής:
$\mu = \begin{bmatrix} 103.27048260381594 & 157.0460157126824 \end{bmatrix}$ και $\Sigma = \begin{bmatrix} 44.19103128 & -11.9310385 \\ -11.9310385 & 11.19574811 \end{bmatrix}$ Έπειτα, διαβάσαμε το πρώτο frame του βίντεο που ζητείται να αναλύσουμε και υπολογίσαμε χρησιμοποιώντας την multivariate_normal.pdf την πιθανότητα που έχει κάθε πίξελ να είναι αποτελεί δέρμα. Στη συνέχεια κανονικοποιήσαμε την εικόνα πολλαπλασιάζοντας με την μέγιστη τιμή πιθανότητας που θα μπορούσε να προκύψει. Η τιμή αυτή υπολογίζεται αν υποθέσουμε ότι ο εκθέτης του e μηδενίζεται. Παρακάτω δίνονται η επιφάνεια της γκαουσιανής κατανομής πιθανότητας που προέκυψε και η εικόνα πιθανότητας δέρματος για το πρώτο frame.

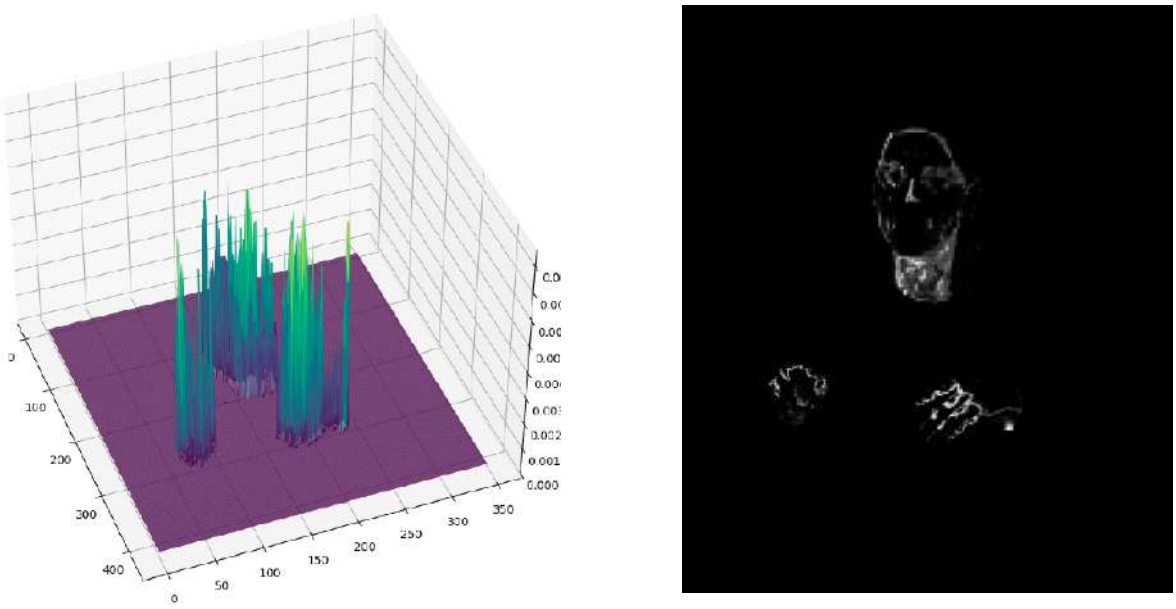


Figure 2: (a) Gaussian probability density of skin, (b) Image of probability

Τέλος, προκύπτει η δυαδική εικόνα με χρήση κατωφλιοποίησης. Για την κατωφλιοποίηση χρησιμοποιήσαμε την τιμή 0.05 την οποία θεωρήσαμε και ιδανική για την καλύτερη εξαγωγή των 3 περιοχών.

Όμως, για την ακριβή ανίχνευση των χεριών και του προσώπου πρέπει να δημιουργήσουμε 3 ενιαίες ανεξάρτητες περιοχές. Για τον σκοπό αυτό απαιτείται μια μορφολογική επεξεργασία της δυαδικής εικόνας. Αρχικά, θα κάνουμε opening με ένα μικρό δομικό στοιχείο ώστε να εξαλειφθούν οι πολύ μικρές μεμονωμένες περιοχές και έπειτα closing με ένα μεγάλο δομικό στοιχείο με σκοπό να καλύψουμε τις τρύπες και να αποκτήσουν συνοχή οι προκύπτουσες περιοχές των χεριών και του προσώπου.

```
mgcbcr = cv2.cvtColor(image, cv2.COLOR_RGB2YCrCb)[:, :, 1:]
frame = scipy.stats.multivariate_normal.pdf(imgcbcr, mean=mu, cov=cov)

frame_normalized = frame * max_val
frame_threshhold = (frame_normalized > 0.05).astype(float)


open_kernel = cv2.getStructuringElement(cv2.MORPH_CROSS, (3, 3))
opening = cv2.morphologyEx(frame_threshhold, cv2.MORPH_OPEN, open_kernel)

close_kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (53, 53))
openclose = cv2.morphologyEx(opening, cv2.MORPH_CLOSE, close_kernel)
```

Παρακάτω φαίνονται τα ενδιάμεσα στάδια της επεξεργασίας μέχρι την τελική επιλογή των περιοχών του προσώπου και των χεριών.

Figure 3: (a) Before morphological edit, (b) After opening with a small kernel, (c) after closing with a big kernel

Τελικά, έχοντας πια 3 συνεκτικές συνιστώσες χρησιμοποιώντας την συνάρτηση **label** καταφέρνουμε να τις κατηγοριοποιήσουμε και με τη συνάρτηση **where** να βρούμε τις συντεταγμένες των περιοχών αυτών. Λαμβάνοντας υπόψιν πλέον την αρχική εικόνα εφαρμόζουμε πάνω της τα 3 bounding boxes που βρήκαμε με συντεταγμένες: (head_box, left_box, right_box) = ([130, 88, 84, 125], [47, 243, 71, 66], [162, 265, 87, 49]) όπως βλέπουμε παρακάτω:

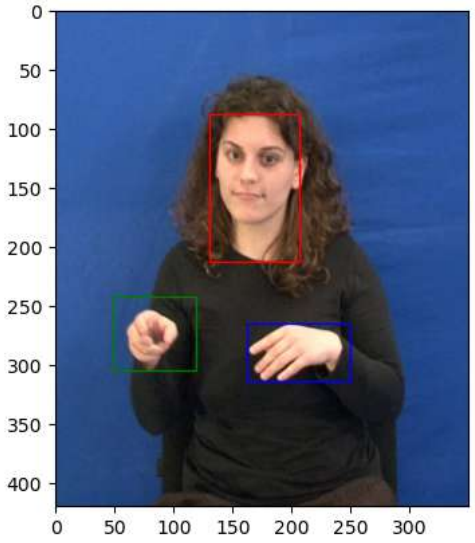


Figure 4: *3 Labels for hands and face tracking*

1.2 Παρακολούθηση Προσώπου και Χεριών

1.2.1 Υλοποίηση του Αλγορίθμου των Lucas-Kanade

Στο μέρος αυτό καλούμαστε να υλοποιήσουμε τον αλγόριθμο των Lucas-Kanade ώστε να υπολογίσουμε το πεδίο οποτικής ροής **-d**. Ο αλγόριθμος αυτός υπολογίζει την οπτική ροή σε κάθε σημείο της εικόνας $\mathbf{x}$ με τη μέθοδο των ελάχιστων τετραγώνων. Σε ένα μικρό παράθυρο γύρω από το σημείο αυτό όμως θεωρείται ότι το d παραμένει σταθερό. Το διάνυσμα της οπτικής ροής d υπολογίζεται επαναληπτικά. Συγκεκριμένα, θεωρούμε ότι έχουμε μια αρχική εκτίμηση d_i για το d την οποία σε κάθε επανάληψη προσπαθούμε να την βελτιώσουμε κατά u, δηλαδή, $d_{i+i} = d_i + u$. Η βελτίωση $\mathbf{u}$ στην οποία αναφερόμαστε αποτελεί λύση ελαχίστων τετραγώνων και έχει την παρακάτω έκφραση:

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

όπου:

$$A(x) = \begin{bmatrix} A_1(x) & A_2(x) \end{bmatrix} = \begin{bmatrix} \frac{\partial I_{n-1}(\mathbf{x}+\mathbf{d}_i)}{\partial x} & \frac{\partial I_{n-1}(\mathbf{x}+\mathbf{d}_i)}{\partial y} \end{bmatrix} \tag{3}$$

και

$$E(x) = I_n(\mathbf{x}) - I_{n-1}(\mathbf{x} + \mathbf{d}_i) \tag{4}$$

Η μικρή θετική σταθερά ε όπως θα συμπεράνουμε και παρακάτω πειραματικά βελτιώνει το αποτέλεσμα σε επίπεδες περιοχές με μειωμένη υφή, δηλαδή, μειωμένη πληροφορία για τον υπολογισμό της οπτικής ροής. Η ανανέωση του διανύσματος οπτικής ροής επαναλαμβάνεται αρκετές φορές έως ότου συγκλίνει σε κάποια τιμή.

Αρχικά ορίζουμε τις δύο εικόνες οι οποίες θα χρησιμοποιηθούν για την εύρεση της οπτικής ροής. Επιπλέον, υπολογίζουμε την πρώτη παράγωγο ως προς x και ως προς y της πρώτης εικόνας και την διδιάστατη γκαουσιανή που θα χρησιμοποιήσουμε στην συνέχεια για τον υπολογισμό της βελτίσωης u. Έπειτα, υπολογίζουμε τα σημεία ενδιαφέροντος της 2ης εικόνας. Για τον σκοπό αυτό χρησιμοποιούμε την συνάρτηση goodFeaturesToTrack η οποία υλοποιεί την μέθοδο των Shi και Tomashi. Για κάθε σημείο ενδιαφέροντος που προκύπτει από την παραπάνω συνάρτηση υλοποιούμε τον αλγόριθμο Lucas-Kanade. Συγκεκριμένα, απομονώνουμε μια μικρή περιοχή γύρω από το σημείο ενδιαφέροντος με μέγεθος ίσο με αυτό του γκαουσιανού πυρήνα.

Ένα σημαντικό σημείο στην υλοποιήση που πρέπει να προσέξουμε είναι ότι κατά τη διάρκεια των επαναλήψεων πολλές φορές θα χρειαστούμε τιμές της εικόνας I_{n-1} και των μερικών παραγώγων της σε ενδιάμεσα σημεία του πλέγματος αφού οι προβλέψεις μας και η βελτίωση που προκύπτει δεν έχουν ακέραιες τιμές. Για να το πετύχουμε αυτό θα χρησιμοποιήσουμε την map_coordinates. Επομένως, με βάση τις παραπάνω σχέσεις και την διαδικασία που περιγράψαμε υπολογίζουμε τα A_1, A_2 και E.

```
Iprev = map_coordinates(I_feat_1, [np.ravel(y_index + d_y0_feat), np.
   ravel(x_index + d_x0_feat)], order=1) \
```

```
                .reshape(I_feat_1.shape[0], I_feat_1.shape[1])
A1 = map_coordinates(I_feat_x, [np.ravel(y_index + d_y0_feat), np.ravel(
    x_index + d_x0_feat)], order=1) \
                .reshape(I_feat_x.shape[0], I_feat_x.shape[1])
A2 = map_coordinates(I_feat_y, [np.ravel(y_index + d_y0_feat), np.ravel(
    x_index + d_x0_feat)], order=1) \
                .reshape(I_feat_y.shape[0], I_feat_y.shape[1])

E = I2[left_y_edge:right_y_edge, left_x_edge:right_x_edge] - Iprev
```

Έπειτα κάνοντας χρήση της γνωστής σχέσης του αντίστροφου πίνακα από την γραμμική άλγεβρα υπολογίζουμε το πρώτο μέρος του u(x). Τέλος, απομένει ένας απλός πολλαπλασιασμός πινάκων που θα μας δώσει το τελικό αποτέλεσμα σε μία επανάληψη το οποίο προσθέτουμε στο προηγούμενο διάνυσμα οπτικής ροής d_{i-1} και προχωράμε στην επόμενη επανάληψη.

```
a11 = cv2.filter2D(A1 ** 2, -1, gauss2D) + epsilon
a12 = cv2.filter2D(A1 * A2, -1, gauss2D)
a22 = cv2.filter2D(A2 ** 2, -1, gauss2D) + epsilon
b1 = cv2.filter2D(A1 * E, -1, gauss2D)
b2 = cv2.filter2D(A2 * E, -1, gauss2D)
det = a11 * a22 - a12 * a12
u_x = (a22 * b1 - a12 * b2) / det
u_y = (a11 * b2 - a12 * b1) / det
dx += u_x
dy += u_y
```

Αρχικά δοκιμάσαμε τον αλγόριθμό μας για μια μετακίνηση κατά 3 pixel στον άξονα των x ώστε να επιβεβαιώσουμε ότι λειτουργεί σωστά. Πήραμε τα παρακάτω αποτελέσματα για το πεδίο οπτικής ροής και το bounding box για το αριστερό χέρι.

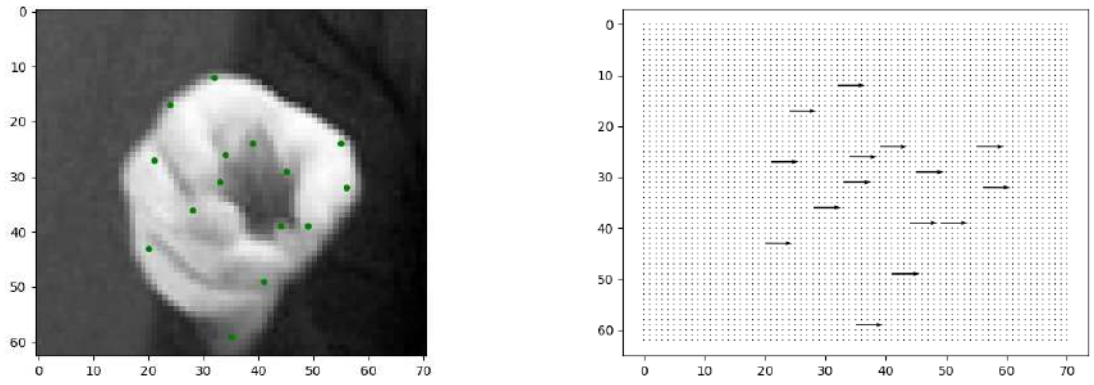
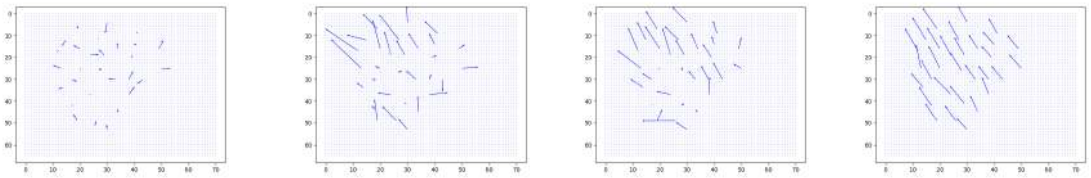
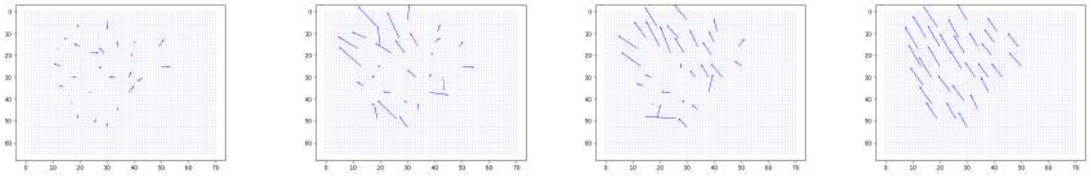


Figure 5: (a) Feautures detected for left hand, b) Optical flow for a 3 pixel displacement

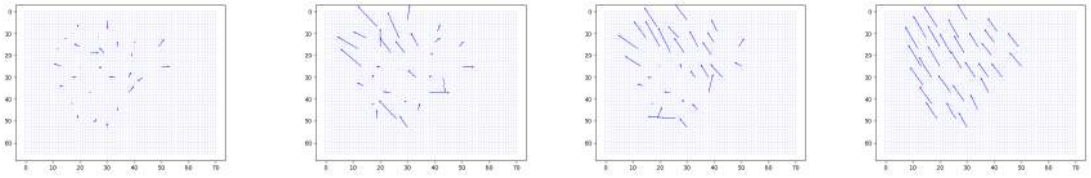
Παρακάτω παρουσιάζουμε τα αποτελέσματα για το πεδίο οπτικής ροής μεταξύ του πρώτου και του δεύτερου frame της ακολουθίας GreekSignLanguage για διάφορες τιμές των παραμέτρων ρ και ε ώστε να κατανοήσουμε την επίδραση τους στην ροή.



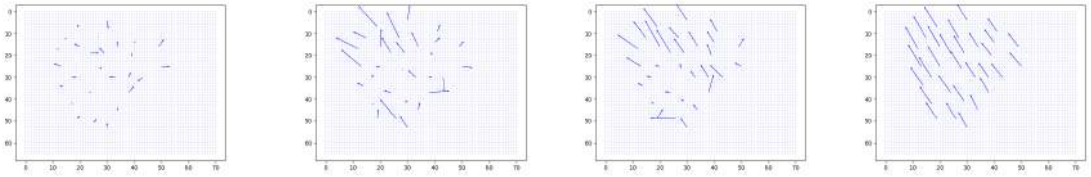
(a) ε=0.01, ρ=1 (e) ε=0.01, ρ=3 (i) ε=0.01, ρ=5 (m) ε=0.01, ρ=10



(b) ε=0.05, ρ=1 (f) ε=0.05, ρ=3 (j) ε=0.05, ρ=5 (n) ε=0.05, ρ=10



(c) ε=0.075, ρ=1 (g) ε=0.075, ρ=3 (k) ε=0.075, ρ=5 (o) ε=0.075, ρ=10



(d) ε=0.1, ρ=1 (h) ε=0.1, ρ=3 (l) ε=0.1, ρ=5 (p) ε=0.1, ρ=10

Όσον αφορά την παράμετρο ρ παρατηρούμε ότι καθώς αυξάνεται το διανυσματικό πεδίο γίνεται πιο πλούσιο αλλά και πιο ομαλό. Βλέπουμε ότι μεγαλύτερα ρ δίνουν και μεγαλύτερες μετατοπίσεις (μέχρι κάποιο σημείο κορεσμού). Επιπλέον, λόγω της συνέλιξης της Γκαουσιανής με μεγαλύτερη τυπική απόκλιση επιτυγχάνεται μεγαλύτερη ομαλοποιήση που οδηγεί στην απώλεια πληροφορίας. Γι'αυτό παρατηρούμε όλα τα σημεία ενδιαφέροντος να έχουν σχεδόν την ίδια μετατόπιση. Η μικρή σταθερά ε γνωρίζουμε ότι βελτιώνει το αποτέλεσμα του υπολογισμού του πεδίου οπτικής ροής σε επίπεδες περιοχές με μειωμένη υφή. Όμως, διατηρώντας σταθερή την παράμετρο ρ βλέπουμε ότι με την αύξηση του ε το διανυσματικό πεδίο οπτικής ροής φαίνεται να δίνει μικρότερες μετατοπίσεις. Αυτό πιθανώς συμβαίνει γιατί ο αλγόριθμός μας δεν προλαβαίνει να συγκλίνει και τα υπολογισμένα διανύσματα ροής έχουν τελικά μικρότερες τιμές. Για μεγαλύτερο ε χρειάζονται περισσότερες επαναλήψεις για να φτάσουμε μια μικρή τιμή κατωφλίου του κριτηρίου σύγκλισης.

1.2.2 Υπολογισμός της Μετατόπισης των Παραθύρων από τα Διανύσματα Οπτικής

Έχοντας υπολογίσει την οπτική ροή της εικόνας I_n στα σημεία ενδιαφέροντος σκοπός μας είναι να βρούμε το συνολικό διάνυσμα μετατόπισης του bounding box ώστε να μπορούμε να παρακολουθήσουμε την κίνηση των χεριών και του προσώπου. Για να το πετύχουμε αυτό θα υπολογίσουμε τον μέσο όρο των διανυσμάτων μετατόπισης των σημείων ενδιαφέροντος. Γνωρίζουμε, όμως, ότι σε σημεία που ανήκουν σε περιοχές με έντονη πληροφορία υφής (π.χ. ακμές κορυφές) το διάνυσμα οπτικής ροής είναι μεγαλύτερο ενώ σε σημεία που ανήκουν σε περιοχές με επίπεδη υφή το διάνυσμα έχει μηδενικό μήκος. Έτσι, για να μειώσουμε το σφάλμα του απλού μέσου όρου, να πετύχουμε μεγαλύτερη ακρίβεια και να απορρίψουμε outliers μπρούμε να χρησιμοποιήσουμε την ενέργεια διανύσματος ταχύτητας, να ορίσουμε ένα κατώφλι ενέργειας κάτω από το οποίο δεν θα συμπεριλαμβάνουμε στο μέσο όρο το διάνυσμα οπτικής ροής.

```
for dx, dy in zip(arr1, arr2):
    E = dx**2 + dy**2
    if E <= thres:
        continue
    opt_vect.append(np.array([dx, dy]))

    if len(opt_vect) == 0:
        return [0.0, 0.0]

    opt_vect = np.array(opt_vect)
    # Define Vectors #
    dxs = opt_vect[:, 0]
    dys = opt_vect[:, 1]

    return [np.mean(dxs), np.mean(dys)]
```

Τελικά, για διάφορες τιμές των ρ, ε έχουμε τις εξής μετατοπίσεις των bounding boxes:

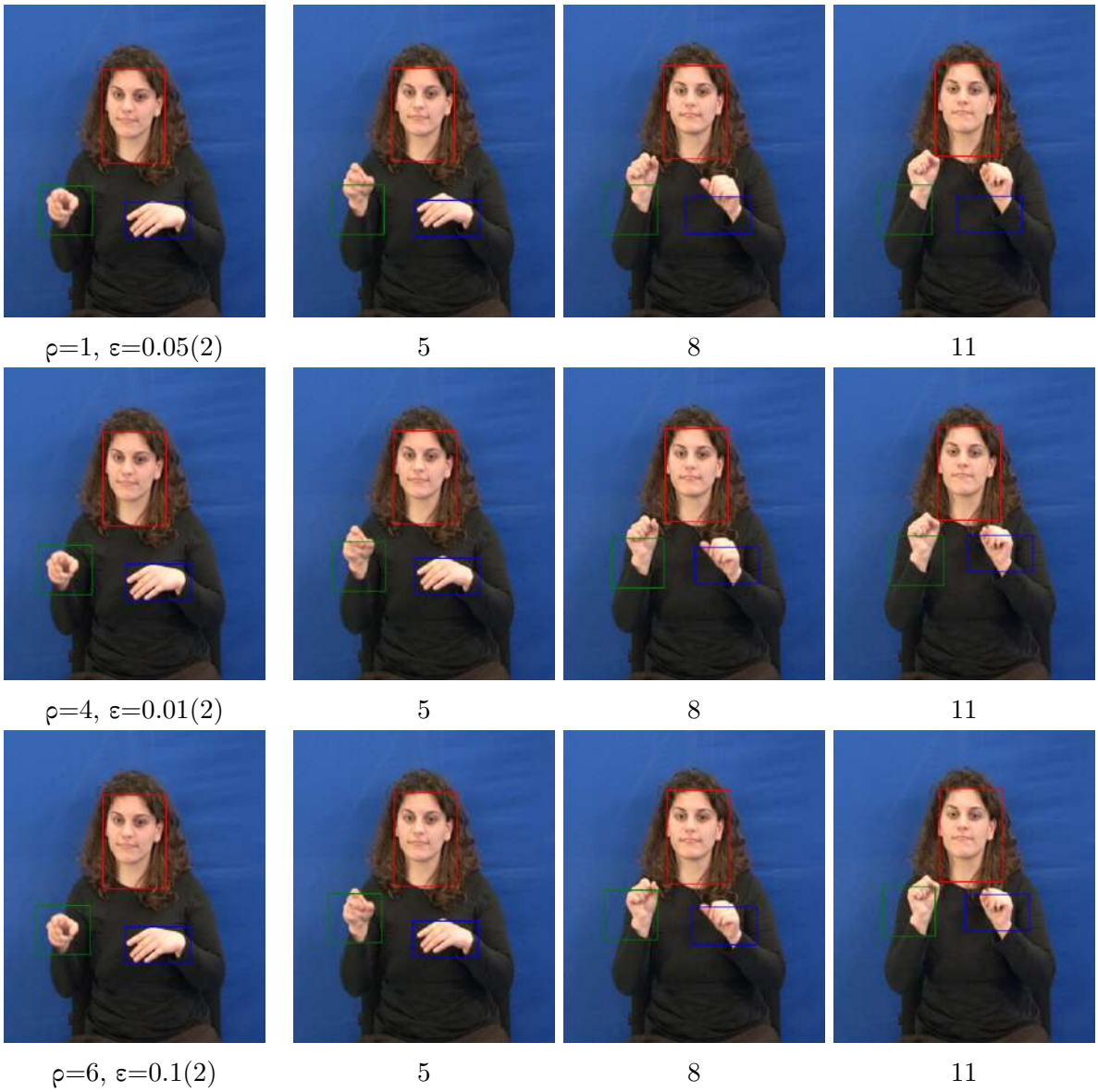


Figure 7: Frames between 2-11 for various ρ and ε

Όπως είχαμε παρατηρήσει και στα διαγράμματα οπτικής ροής έτσι και εδώ βλέπουμε ότι όσο μεγαλύτερο είναι το ρ τόσο πιο ομαλές κινήσεις έχουμε. Γι' αυτό στην τελευταία γραμμή στο frame 11 το πράσινο box συνεχίζει να κινείται προς τα πάνω ενώ έπρεπε να κινηθεί και περισσότερο δεξιά. Παρόλαυτα έχει προσεγγίσει αρκετά καλά την κίνηση του χεριού στον άξονα των y. Τέλος, είναι φανερό πως τη χειρότερη επίδοση έχουμε για $\rho = 1$.

1.2.3 Πολυ-Κλιμακωτός Υπολογισμός Οπτικής Ροής

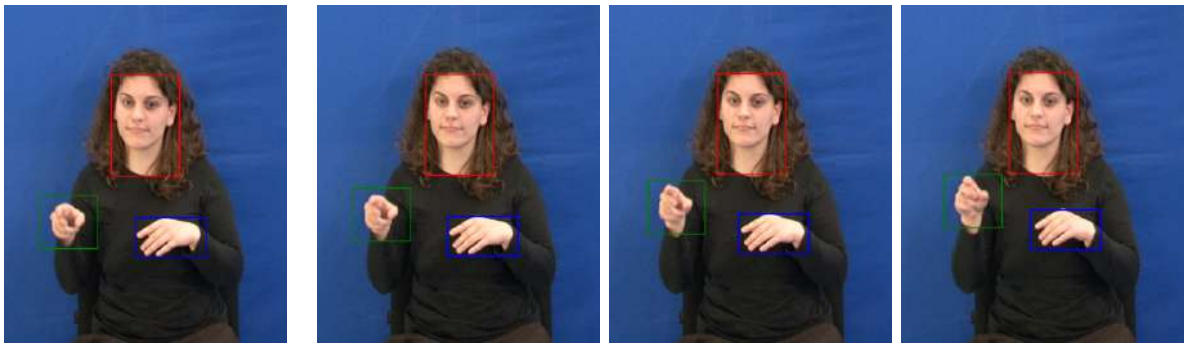
Στο μέρος αυτό υλοποιούμε την πολυκλιμακωτή μέθοδο των Lucas-Kanade. Η διαφορά με την προηγούμενη υλοποίηση είναι ότι τώρα ο αλγόριθμος αναλύει τις αρχικές εικόνες σε

γκαουσιανές πυραμίδες και υπολογίζει το πεδίο οπτικής ροής από τις πιο μικρές (τραχείς) στις πιο μεγάλες (λεπτομερείς) κλίμακες θέτοντας ως αρχική συνθήκη τη λύση της μικρής κλίμακας για τη μεγάλη κλίμακα. Έτσι, αυτό που πετυχαίνουμε είναι ταχύτερη σύκλιση στις μεγαλύτερες κλίμακες καθώς και να υπολογίζουμε με μεγαλύτερη ακρίβεια το διάνυσμα οπτικής ροής. Κατά τη μετάβαση από τις μεγάλες στις μικρές κλίμακες η εικόνα φιλτράρεται με μια γκαουσιανή τυπικής απόκλισης 3 pixels προκειμένου να μετριαστεί η φασματική αναδίπλωση της εικόνας. Επίσης, κατά τη μετάβαση από τις μικρές στις μεγάλες κλίμακες το διάνυσμα d διπλασιάζεται, καθώς μετατόπιση 1 pixel σε μία κλίμακα συνεπάγεται μετατόπιση 2 pixel στην αμέσως μεγαλύτερη λόγω downsampling κατά 2.

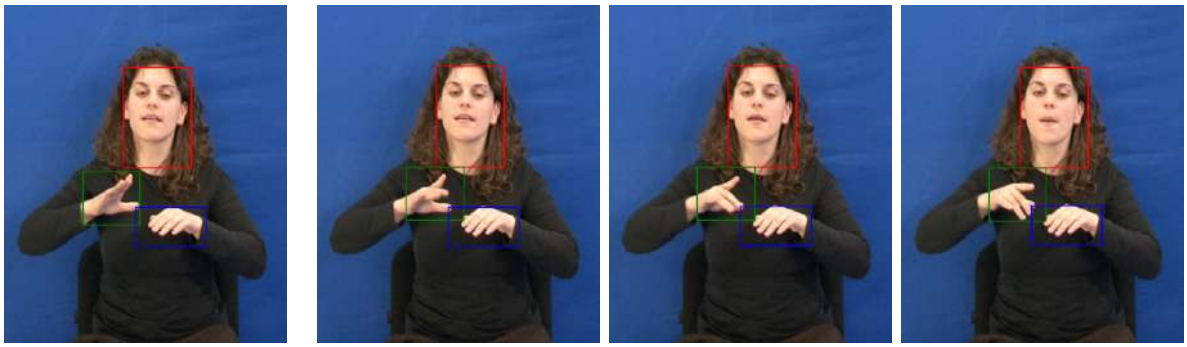
```
# gaussian filtering and subbsampling
def GREDUCE(I):
    gauss1D = cv2.getGaussianKernel(19, 3)
    gauss2D = gauss1D @ gauss1D.T
    convoluted_image = cv2.filter2D(I, -1, gauss2D)
    return convoluted_image[0::2, 0::2]


# create gaussian pyramid of given depth with kernel specified by
   variable a
def GPyramid(I, depth):
    g_pyr = [I]
    for i in range(depth - 1):
        g_pyr.append(GREDUCE(g_pyr[i]))
    g_pyr.reverse()
    return g_pyr

def lk_Gpyramid(I1, I2, rho, epsilon, depth, maxCorners):
    Gp_1 = GPyramid(I1, depth)
    Gp_2 = GPyramid(I2, depth)
    d_x0 = np.zeros(Gp_1[0].shape)
    d_y0 = np.zeros(Gp_1[0].shape)
    [displ_x, displ_y] = [0, 0]
    [resx, resy] = [0, 0]
    for i in range(depth):
        blockSize = 7
        feature_params = dict(maxCorners=maxCorners, qualityLevel=0.001,
   minDistance=5, blockSize=blockSize)
        features = cv2.goodFeaturesToTrack(Gp_2[i], mask=None, **
   feature_params)
        while features is None and blockSize > 2:
            blockSize -= 2
            feature_params = dict(maxCorners=maxCorners, qualityLevel
   =0.001, minDistance=5, blockSize=blockSize)
            features = cv2.goodFeaturesToTrack(Gp_2[i], mask=None, **
   feature_params)
        if blockSize < 2:
            [displ_x, displ_y] = [0, 0]
        else:
```

```
            resx, resy, features_moved_x, features_moved_y = lk(Gp_1[i],
    Gp_2[i], features, rho, epsilon, d_x0, d_y0)
            [displ_x, displ_y] = displ(features_moved_x, features_moved_y
    )
        if i < depth - 1:
            d_x0 = np.full(Gp_1[i + 1].shape, displ_x * 2)
            d_y0 = np.full(Gp_1[i + 1].shape, displ_y * 2)

    return displ_x, displ_y, resx, resy
```

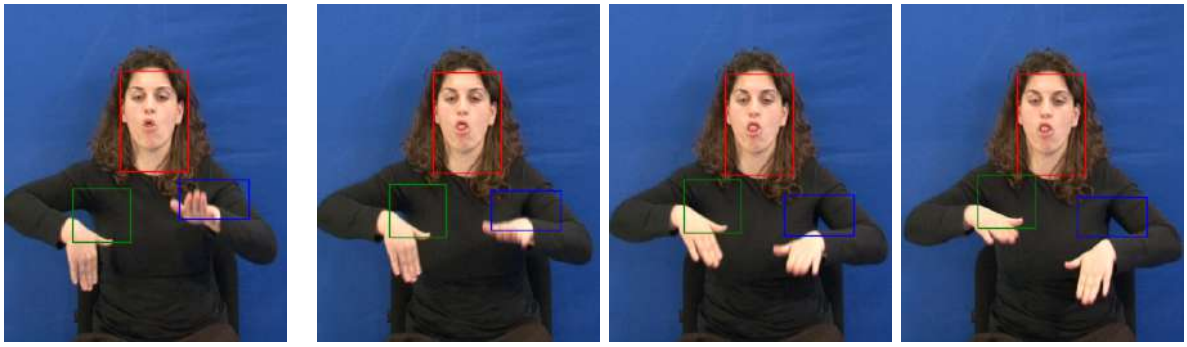
Κάτι που πρέπει να προσέξουμε στην υλοποιήση είναι ότι καθώς υποδειγματοληπτείται η εικόνα τα σημεία ενδιαφέροντος μειώνονται αισθητά συνεπώς πρέπει να κάνουμε πιο επιεική τα κριτήρια/παραμέτρους εύρεσης σημείων ενδιαφέροντος στην συνάρτηση goodFeaturesToTrack. Παρακάτω παρουσιάζουμε τα αποτελέσματα για τον πολυκλιμακωτό Lucas-Kanade πρώτα για παραμέτρους ρ=4 και ε=0.1 και έπειτα για ρ=6 και ε=0.05.



frame 2 *frame 3* *frame 4* *frame 5*

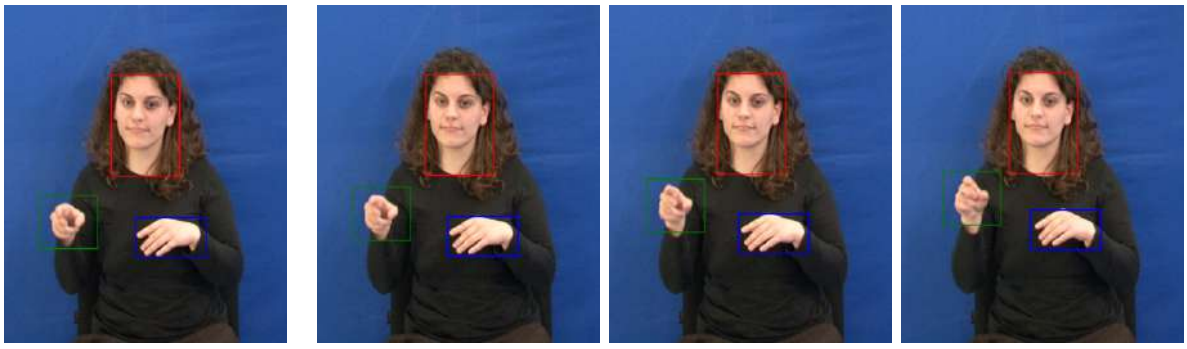


frame 33 *frame 34* *frame 35* *frame 36*

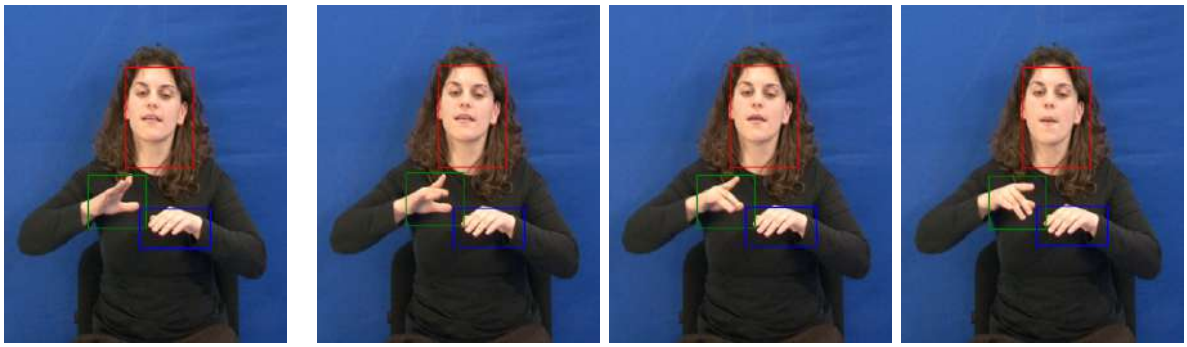


frame 62 *frame 63* *frame 64* *frame 65*

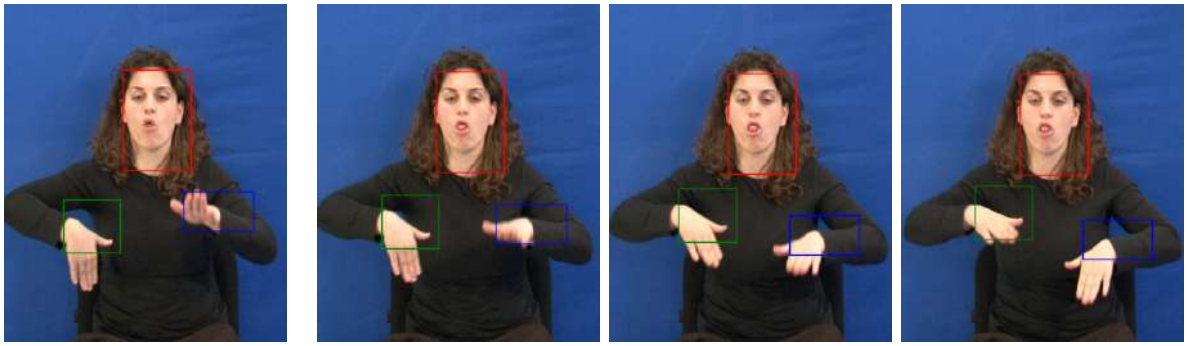
Figure 8: Frames from beginning, middle and end of video for ρ=4, ε=0.1



frame 2 *frame 3* *frame 4* *frame 5*



frame 33 *frame 34* *frame 35* *frame 36*



frame 62 *frame 63* *frame 64* *frame 65*

Figure 9: Frames from beginning, middle and end of video for ρ=6, ε=0.05

Παρατηρούμε πως για μεγαλύτερο ρ έχουμε καλύτερη παρακολούθηση των περιοχών, καθώς στα τελευταία frames του βίντεο για ρ=4 φαίνεται πως χάνουμε το δεξί χέρι, ενώ για ρ=6 αν και αυτό έχει βγεί μερικώς εκτός του bounding box δεν έχει χαθεί από το σύστημα. Αυτό οφείλεται στο γεγονός πως για μεγαλύτερα ρ έχουμε μεγαλύτερες περιοχές συνέλιξης και συνεπώς ακολουθούμε πιο μεγάλες περιοχές, σε αντίθεση με μικρότερα ρ που περιορίζονται γύρω από τα σημεία ενδιαφέροντος. Έτσι είμαστε ικανοί να ακολουθούμε πιο απότομες κινήσεις.

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

Στο μέρος 2 της εργασίας θα ασχοληθούμε με τον εντοπισμό σημείων ενδιαφέροντος από τα βίντεο που μας δίνονται και την εξαγωγή χαρακτηριστικών των βίντεο γύρω από αυτά που ύστερα θα χρησιμοποιηθούν για την αναγνώριση δράσεων που φαίνονται σε βίντεο με χρήση SVM.

2.1 Χωρο-χρονικά Σημεία Ενδιαφέροντος

Το πρώτο βήμα της διαδικασίας είναι ο εντοπισμός σημείων ενδιαφέροντος στα βίντεο που μας δίνονται. Για να το επιτύχουμε αυτό θα χρησιμοποιήσουμε δύο ανιχνευτές, έναν ο οποίος χρησιμοποιεί τη μέθοδο Harris σε 3 διαστάσεις και έναν ο οποίος χρησιμοποιεί φιλτράρισμα με ζεύγος Gabor φίλτρων.

Για τον ανιχνευτή Harris θα πάρουμε πρώτα το βίντεο στο οποίο θέλουμε να εκτελέσουμε την ανίχνευση σημείων ενδιαφέροντος και θα εκτελέσουμε εξομάλυνση μέσω συνέλιξης με γκαουσιανό πυρήνα κλίμακας σ στις δύο χωρικές διαστάσεις και κλίμακας τ στη χρονική διάσταση. Την ομαλοποίηση αυτή την κάνουμε με σκοπό να μειωθούν οι ασυνέχειες του βίντεο για να εκτελέσουμε μετά παραγώγηση. Ύστερα, υπολογίζουμε τις κατευθυντήριες παραγώγους L_x, L_y, L_t και στις τρείς διαστάσεις και κατασκευάζουμε τον πίνακα

$$L = \begin{pmatrix} L_x^2 & L_xL_y & L_xL_t \\ L_xL_y & L_y^2 & L_yL_t \\ L_xL_t & L_yL_t & L_t^2 \end{pmatrix}$$

Στη συνέχεια εκτελούμε τη συνέλιξη κάθε συνιστώσας με γκαουσιανό πυρήνα ομαλοποίησης για να κατασκευάσουμε τον πίνακα $M = g(x, y, t; s\sigma, s\tau) * L$ όπου σ και τ οι κλίμακες του προηγούμενου πυρήνα που χρησιμοποιήσαμε και s σταθερά κλιμάκωσης και βρίσκουμε το κριτήριο γωνιότητας για κάθε voxel του βίντεο σύμφωνα με τη σχέση $H(x, y, t) = det(M(x, y, t)) - k * trace^3(M(x, y, t))$, όπου k μικρή θετική σταθερά.

```
def harris_3d(video, sigma, s, tau, k):
    video = video.astype(float) / 255

    n = int(np.ceil(3 * sigma) * 2 + 1)
    k1 = np.transpose(cv2.getGaussianKernel(n, sigma))[0]
    k2 = k1  # the kernel along the 2nd dimension
    n = int(np.ceil(3 * tau) * 2 + 1)
    k3 = np.transpose(cv2.getGaussianKernel(n, tau))[0]  # the kernel
   along the 3rd dimension
    # Convolve over all three axes to smooth out video
    smooth_video = video.copy()
```

```
    for i, kernel in enumerate((k1, k2, k3)):
        smooth_video = scp.convolve1d(smooth_video, kernel, axis=i)

    Ly, Lx, Lt = np.gradient(smooth_video)

    n = int(np.ceil(3 * s*sigma) * 2 + 1)
    k1 = np.transpose(cv2.getGaussianKernel(n, s*sigma))[0]
    k2 = k1  # the kernel along the 2nd dimension
    n = int(np.ceil(3 * s*tau) * 2 + 1)
    k3 = np.transpose(cv2.getGaussianKernel(n, s*tau))[0]  # the kernel
   along the 3rd dimension

    a11 = Lx*Lx
    for i, kernel in enumerate((k1, k2, k3)):
        a11 = scp.convolve1d(a11, kernel, axis=i)
    a12 = Lx*Ly
    for i, kernel in enumerate((k1, k2, k3)):
        a12 = scp.convolve1d(a12, kernel, axis=i)
    a13 = Lx*Lt
    for i, kernel in enumerate((k1, k2, k3)):
        a13 = scp.convolve1d(a13, kernel, axis=i)
    a22 = Ly*Ly
    for i, kernel in enumerate((k1, k2, k3)):
        a22 = scp.convolve1d(a22, kernel, axis=i)
    a23 = Ly*Lt
    for i, kernel in enumerate((k1, k2, k3)):
        a23 = scp.convolve1d(a23, kernel, axis=i)
    a33 = Lt*Lt
    for i, kernel in enumerate((k1, k2, k3)):
        a33 = scp.convolve1d(a33, kernel, axis=i)

    detM = a11*a22*a33 + 2*a12*a23*a13 - a11*a23*a23 -
        a12*a12*a33 - a13*a13*a22
    traceM = a11 + a22 + a33
    H_harris = np.abs(detM - k * (traceM * traceM * traceM))
    return H_harris
```

Για τον ανιχνευτή Gabor εκτελούμε όπως και με τον Harris εξομάλυνση με γκαουσιανό πυρήνα κλίμακας σ στις δύο διαστάσεις x, y και στη συνέχεια υπολογίζουμε τους πυρήνες $h_ev(t;\tau,\omega) = cos(2\pi t\omega)exp(-t^2/2\tau^2)$ και $h_od(t;\tau,\omega) = sin(2\pi t\omega)exp(-t^2/2\tau^2)$ με τους οποίους θα εκτελέσουμε συνέλιξη του ομαλοποιημένου βίντεο. Η συχνότητα ω των συναρτήσεων δίνεται μέσω της σχέσης της με τη χρονική κλίμακα τ ως ω=4/τ. Έτσι, υπολογίζουμε το κριτήριο γωνιότητας $H(x,y,t) = (I(x,y,t) * g * h_ev)^2 + (I(x,y,t) * g * h_od)^2$.

```
def gabor(video, sigma, tau):
    video = video.astype(float) / 255

    n = int(np.ceil(3 * sigma) * 2 + 1)
    k1 = np.transpose(cv2.getGaussianKernel(n, sigma))[0]
    k2 = k1  # the kernel along the 2nd dimension
    # Convolve over all three axes to smooth out video
```

```
    smooth_video = video.copy()
    for i, kernel in enumerate((k1, k2)):
        smooth_video = scp.convolve1d(smooth_video, kernel, axis=i)

    omega = 4/tau
    window = np.linspace(int(np.round(-2*tau)), int(np.round(2*tau)), 2*
int(np.round(2*tau)) + 1, dtype=int)
    h_ev = np.cos(2 * np.pi * window * omega) * np.exp((- window**2) / (2
    * tau**2))
    h_ev /= np.linalg.norm(h_ev, ord=1)
    h_od = np.sin(2 * np.pi * window * omega) * np.exp((- window**2) / (2
    * tau**2))
    h_od /= np.linalg.norm(h_ev, ord=1)
    H_gabor = (scp.convolve1d(smooth_video, h_ev, axis=2))**2 +
          (scp.convolve1d(smooth_video, h_od, axis=2))**2
    return H_gabor
```

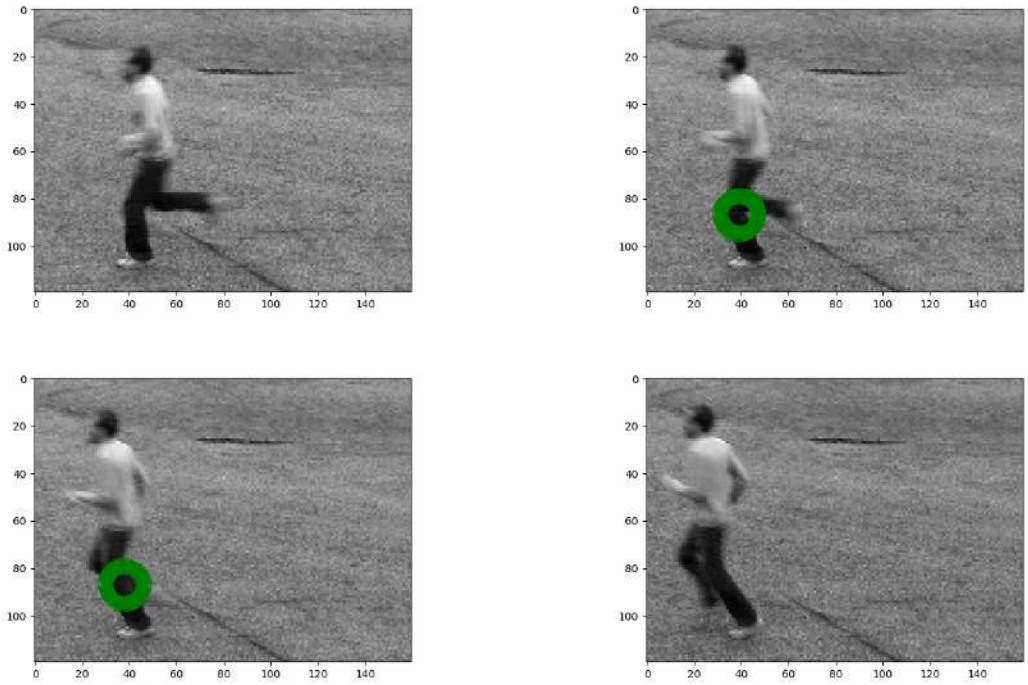


Figure 10: Harris Detector

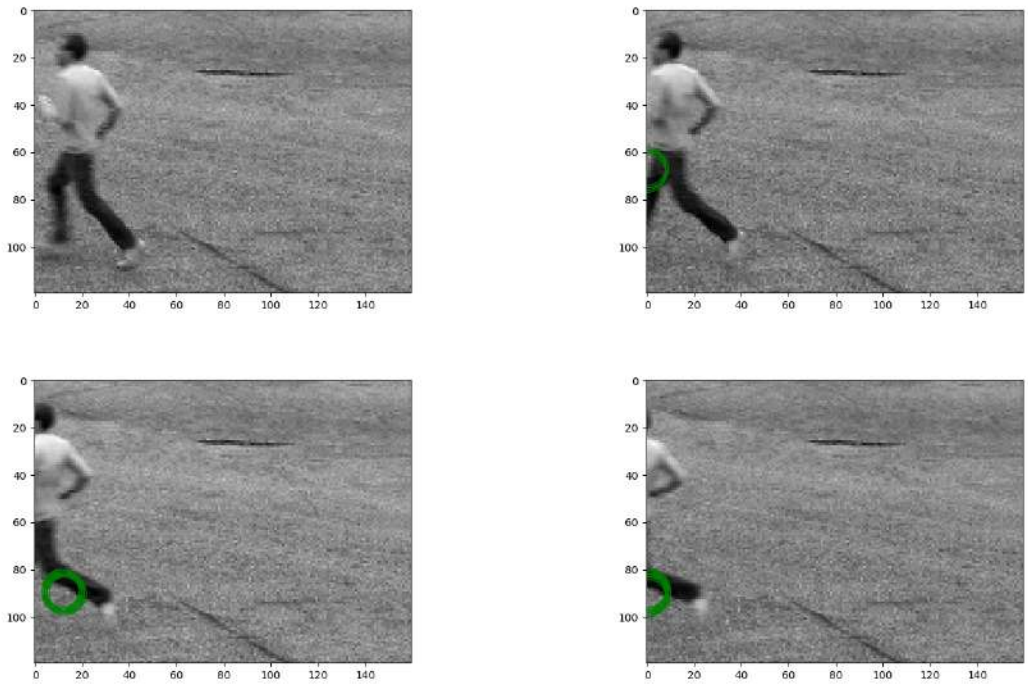


Figure 11: Gabor Detector

Παραπάνω φαίνονται μερικά frames όπου ο κάθε ανιχνευτής βρήκε σημεία ενδιαφέροντος. Παρατηρούμε πως ο Harris detector βρίσκει πιο συγκεντρωμένα τα σημεία του, σε αντίθεση με τον Gabor detector ο οποίος φαίνεται να εντοπίζει σημεία ενδιαφέροντος σε περισσότερα χωρικά και χρονικά σημεία ενδιαφέροντος. Αυτό θα μπορούσαμε να το αποφύγουμε εφαρμόζοντας κάποιο κριτήριο αντίστοιχο με αυτό του πρώτου εργαστηρίου για να αποκλείσουμε σημεία που εμφανίζονται πολύ κοντά σε άλλα. Μεγαλύτερη ποικιλία σημείων συνεπάγεται πιο εύρωστους περιγραφητές, επομένως έχοντας μεγάλη ομοιότητα στα σημεία που επιστρέφει ο ανιχνευτής που υλοποιείται με τη μέθοδο Harris πιθανώς θα έχουμε λιγότερο ακριβή αποτελέσματα συγκριτικά με τον ανιχνευτή Gabor.

Τα κριτήρια γωνιότητας (H(x,y,t)) για κάθε έναν από τους ανιχνευτές που κατασκευάσαμε και για κάθε είδος κίνησης φαίνονται παρακάτω. Οι τιμές που χρησιμοποιήθηκαν είναι σ=4, s=2, τ=1.5, k=0.005.

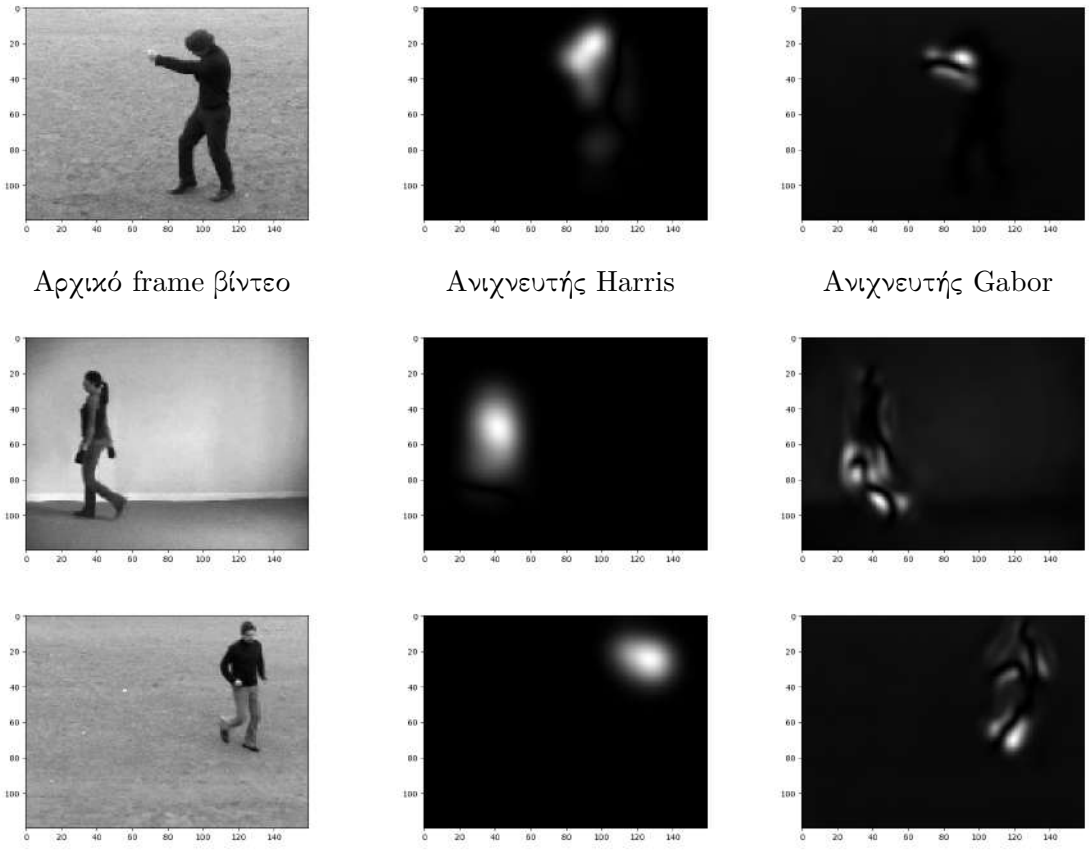


Figure 12: Frames and importance intensity for the same video frame, one from every category

Παρατηρούμε πως ο ανιχνευτής Gabor εντοπίζει με μεγαλύτερη ακρίβεια το σημείο της κίνησης, ενώ ο ανιχνευτής Harris φαίνεται να εντοπίζει το κέντρο βάρους της κίνησης, με αποτέλεσμα να είναι λιγότερο ακριβής η αναπαράσταση του κάθε βίντεο από τις οπτικές λέξεις που θα περιέχει το λεξικό που θα παράξουμε στο 2.3.

2.2 Χωρο-χρονικοί Ιστογραφικοί Περιγραφητές

Έχοντας εντοπίσει τα σημεία ενδιαφέροντος στα βίντεο μπορούμε τώρα να βρούμε τους χωρο-χρονικούς περιγραφητές HoG και HoF.
Για να βρούμε τους περιγραφητές HoG για το κάθε βίντεο θα βρούμε πρώτα τις δύο χωρικές παραγώγους για το βίντεο, και για κάθε σημείο ενδιαφέροντος θα δώσουμε στη συνάρτηση orientation_histogram() για είσοδο τις δύο παραγώγους σε μία περιοχή 4σ γύρω από το κάθε σημείο. Επαναλαμβάνουμε τη διαδικασία για όλα τα σημεία ενδιαφέροντος που εντοπίσαμε στο παραπάνω βήμα.

```
def get_HoG(video, int_points, sigma, nbins):
    der_y, der_x, der_t = np.gradient(video)
    side = int(np.round(4 * sigma))

    descriptors = []
    for point in int_points:
        leftmost = max(0, point[0] - side)
        rightmost = min(video.shape[1] - 1, point[0] + side + 1)
        uppermost = max(0, point[1] - side)
        lowermost = min(video.shape[0] - 1, point[1] + side + 1)

        descriptor = orientation_histogram(
              der_x[uppermost:lowermost, leftmost:rightmost, point[2]],
              der_y[uppermost:lowermost, leftmost:rightmost, point[2]],
              nbins, np.array([side, side]))

        descriptors.append(descriptor)
    return np.array(descriptors)
```

Για την εξαγωγή των περιγραφητών HoF βρίσκουμε την οπτική ροή σε μια περιοχή 4σ γύρω από κάθε σημείο ενδιαφέροντος που έχουμε παράξει και δίνουμε αυτή αντί των κατευθυντικών παραγώγων ως είσοδο στην orientation_histogram().

```
def get_HoF(video, int_points, sigma, nbins):
    side = int(np.round(4 * sigma))
    oflow = cv2.DualTVL1OpticalFlow_create(nscales=1)

    descriptors = []
    for interest_point in int_points:
        leftmost = max(0, interest_point[0] - side)
        rightmost = min(video.shape[1] - 1, interest_point[0] + side + 1)
        uppermost = max(0, interest_point[1] - side)
        lowermost = min(video.shape[0] - 1, interest_point[1] + side + 1)
        if interest_point[2] == video.shape[2] - 1:
            interest_point[2] -= 1

        flow = oflow.calc(
    video[uppermost:lowermost,leftmost:rightmost,interest_point[2]],
    video[uppermost:lowermost,leftmost:rightmost,interest_point[2]+1],
    None)

        u_flow = flow[:, :, 0]
        v_flow = flow[:, :, 1]
        descriptor = orientation_histogram(u_flow, v_flow,
                    nbins, np.array([side, side]))
        descriptors.append(descriptor)

    return np.array(descriptors)
```

Τέλος, ενώνουμε τις δύο λίστες με τους περιγραφητές για κάθε βίντεο και εξάγουμε έτσι τους HoG/HoF περιγραφητές.

2.3 Κατασκευή Bag of Visual Words και χρήση Support Vector Machines για την ταξινόμηση δράσεων

Στο μέρος αυτό θα χωρίσουμε τα βίντεο σε δύο σετ, ένα για εκπαίδευση για αξιολόγηση του SVM στο οποίο θα τροφοδοτήσουμε τους περιγραφητές μας. Χρησιμοποιούμε τα αποτελέσματα του παραπάνω ερωτήματος για να εξάγουμε τοπικούς περιγραφητές HoG, HoF και HoG/HoF γύρω από τα σημεία ενδιαφέροντος που βρήκαμε με τους ανιχνευτές Harris και Gabor για κάθε βίντεο του συνόλου δεδομένων μας.
Δίνεται ο κώδικας για το διάβασμα του αρχείου που περιέχει το training set και χωρίζει τα βίντεο στις δύο κατηγορίες:

```
def train_test():
    fd = open('cv21_lab2_part2_material/data/training_videos.txt', 'r')
    file = fd.readline()
    train_videos = []

    all_videos = glob.glob(path+'running/*')
    all_videos += glob.glob(path+'boxing/*')
    all_videos += glob.glob(path+'walking/*')

    while file:
        file = file.replace('\n', '')
        if 'running' in file:
            label = 'running'
        elif 'boxing' in file:
            label = 'boxing'
        else:
            label = 'walking'
        video = path + label + '/' + file
        train_videos.append(video)
        file = fd.readline()

    test_videos = []
    for vid in all_videos:
        if vid not in train_videos:
            test_videos.append(vid)

    train_labels = []
    for i in range(len(train_videos)):
        if 'running' in train_videos[i]:
            train_labels.append(0)
        elif 'boxing' in train_videos[i]:
            train_labels.append(1)
        else:
            train_labels.append(2)

    test_labels = []
    for i in range(len(test_videos)):
        if 'running' in test_videos[i]:
            test_labels.append(0)
```

```
        elif 'boxing' in test_videos[i]:
            test_labels.append(1)
        else:
            test_labels.append(2)

    return train_videos, train_labels, test_videos, test_labels
```

καθώς και ο κώδικας που βρίσκει τα σημεία ενδιαφέροντος χρησιμοποιώντας τους ανιχνευτές που κατασκευάσαμε στο μέρος 2.1, και ύστερα τα χρησιμοποιεί για να εξάγει τους περιγραφητές γύρω από κάθε σημείο:

```
def calc_hog_hof_all(training_videos, test_videos, sigma, s, tau, k, N,
    nbins):
    for vid in training_videos:
        print(vid)
        video = read_video(vid)
        name = vid.split('/')[3]
        fd1 = open('pickle_data/' + name[:-4] + "_harris_bow.p", 'wb')
        fd2 = open('pickle_data/' + name[:-4] + "_gabor_bow.p", 'wb')

        H1 = harris_3d(video, sigma, s, tau, k)
        H2 = gabor(video, sigma, tau)

        int_pts1 = interest_points_extraction(H1, N, sigma)
        int_pts2 = interest_points_extraction(H2, N, sigma)

        desc1 = get_HoG_HoF(video, int_pts1, sigma, nbins)
        desc2 = get_HoG_HoF(video, int_pts2, sigma, nbins)

        pickle.dump(desc1, fd1)
        pickle.dump(desc2, fd2)
        fd1.close()
        fd2.close()

    for vid in test_videos:
        print(vid)
        video = read_video(vid)
        name = vid.split('/')[3]
        fd1 = open('pickle_data/' + name[:-4] + "_harris_bow.p", 'wb')
        fd2 = open('pickle_data/' + name[:-4] + "_gabor_bow.p", 'wb')

        H1 = harris_3d(video, sigma, s, tau, k)
        H2 = gabor(video, sigma, tau)

        int_pts1 = interest_points_extraction(H1, N, sigma)
        int_pts2 = interest_points_extraction(H2, N, sigma)

        desc1 = get_HoG_HoF(video, int_pts1, sigma, nbins)
        desc2 = get_HoG_HoF(video, int_pts2, sigma, nbins)

        pickle.dump(desc1, fd1)
        pickle.dump(desc2, fd2)
```

```
        fd1.close()
        fd2.close()
```

Τα αποτελέσματα τα αποθηκεύουμε στο τοπικό δίσκο και τα διαβάζουμε με την παρακάτω συνάρτηση ώστε να αποφύγουμε να τα υπολογίζουμε κάθε φορά από την αρχή. Αντίστοιχα με την παραπάνω συνάρτηση είναι υλοποιημένη η εξαγωγή χαρακτηριστικών και για τους περιγραφητές HoG και HoF.

```
def fetch_hog_hof_all(training_videos, test_videos, descriptor=''):
    train1 = []
    train2 = []
    assert descriptor in ['', '_hog', '_hof']
    for vid in training_videos:
        name = vid.split('/')[3]
        fd1 = open('pickle_data' + descriptor + '/' + name[:-4] + "
   _harris_bow.p", 'rb')
        fd2 = open('pickle_data' + descriptor + '/' + name[:-4] + "
   _gabor_bow.p", 'rb')

        desc1 = pickle.load(fd1)
        train1.append(desc1)
        desc2 = pickle.load(fd2)
        train2.append(desc2)

        fd1.close()
        fd2.close()

    test1 = []
    test2 = []
    for vid in test_videos:
        name = vid.split('/')[3]
        fd1 = open('pickle_data' + descriptor + '/' + name[:-4] + "
   _harris_bow.p", 'rb')
        fd2 = open('pickle_data' + descriptor + '/' + name[:-4] + "
   _gabor_bow.p", 'rb')

        desc1 = pickle.load(fd1)
        test1.append(desc1)
        desc2 = pickle.load(fd2)
        test2.append(desc2)

        fd1.close()
        fd2.close()

    return train1, test1, train2, test2
```

Στη συνέχεια δίνουμε αυτούς τους περιγραφητές στη συνάρτηση bag_of_words() για να εξάγουμε την τελική αναπαράσταση για κάθε βίντεο, δηλαδή ποιές από τις οπτικές λέξεις που έχουμε στο λεξικό μας περιέχονται σε κάθε βίντεο.

Τέλος, έχοντας κατασκευάσει τα labels για όλα τα βίντεο του συνόλου δεδομένων μας δίνουμε την τελική αναπαράσταση των βίντεο καθώς και τις ετικέτες τους χωρισμένα στα σύνολα train και test για να βρούμε την ακρίβεια και τις προβλέψεις του κάθε περιγραφητή. Επαναλαμβάνουμε αρκετές φορές αυτή τη διαδικασία ώστε να βρούμε ένα μέσο όρο για την ακρίβεια του συστήματός μας.

```
def train(tr_videos, tr_labels, tst_videos, tst_labels, iterations,
    descriptor):
     tr_harris, tst_harris, tr_gabor, tst_gabor = fetch_hog_hof_all(
    tr_videos, tst_videos, descriptor)

     tot_acc_har = 0
     tot_acc_gab = 0
     print("Correct labels: ", tst_labels)
     for i in range(iterations):
         bow_tr_har, bow_tst_har = bag_of_words(tr_harris, tst_harris, 50)
         bow_tr_gabor, bow_tst_gabor = bag_of_words(tr_gabor, tst_gabor,
    50)
         acc_har, pred_har = svm_train_test(bow_tr_har, tr_labels,
    bow_tst_har, tst_labels)
         print("Prediction ", i+1, " for harris:\t", pred_har)
         acc_gab, pred_gab = svm_train_test(bow_tr_gabor, tr_labels,
    bow_tst_gabor, tst_labels)
         print("Prediction ", i+1, " for gabor:\t", pred_gab)
         tot_acc_har += acc_har
         tot_acc_gab += acc_gab
     tot_acc_har /= iterations
     tot_acc_gab /= iterations
     print("Total accuracy for Harris detector: ", tot_acc_har * 100, "%")
     print("Total accuracy for Gabor detector: ", tot_acc_gab * 100, "%")
```

Αυτή τη διαδικασία την εκτελούμε για κάθε συνδιασμό ανιχνευτή και περιγραφητή.

```
Correct labels:  [0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2]
Prediction  1  for harris:      [2 0 2 0 1 0 1 0 2 2 0 2]
Prediction  1  for gabor:       [0 0 0 0 1 1 1 1 2 2 2 2]
Prediction  2  for harris:      [0 0 2 0 1 2 0 1 2 2 0 2]
Prediction  2  for gabor:       [2 0 0 0 1 1 1 1 2 2 2 2]
Prediction  3  for harris:      [0 0 2 0 0 0 2 0 2 2 0 2]
Prediction  3  for gabor:       [0 0 0 0 1 1 1 1 2 2 2 2]
Prediction  4  for harris:      [0 0 2 0 0 0 0 0 2 0 0 2]
Prediction  4  for gabor:       [2 1 0 0 1 1 1 1 2 2 2 2]
Prediction  5  for harris:      [2 0 2 0 0 2 2 0 2 2 0 2]
Prediction  5  for gabor:       [0 0 0 0 1 1 1 1 2 2 2 2]
Prediction  6  for harris:      [2 0 2 0 1 1 1 0 2 2 0 2]
Prediction  6  for gabor:       [0 0 0 0 1 1 1 1 2 2 2 2]
Prediction  7  for harris:      [0 0 2 0 0 0 0 2 2 2 0 2]
Prediction  7  for gabor:       [0 1 0 0 1 1 1 1 2 2 2 2]
Prediction  8  for harris:      [2 0 2 2 0 2 2 2 2 2 0 2]
Prediction  8  for gabor:       [0 1 0 0 1 1 1 1 2 2 2 2]
Prediction  9  for harris:      [0 0 2 0 0 0 0 0 2 0 0 2]
Prediction  9  for gabor:       [2 0 0 0 1 1 1 1 2 2 2 2]
Prediction  10  for harris:     [0 0 2 0 0 0 2 0 2 2 0 2]
Prediction  10  for gabor:      [0 0 0 0 1 1 1 1 2 2 2 2]
Total accuracy for Harris detector:  50.0 %
Total accuracy for Gabor detector:  95.0 %
```

Figure 13: HoG descriptor for Harris and Gabor

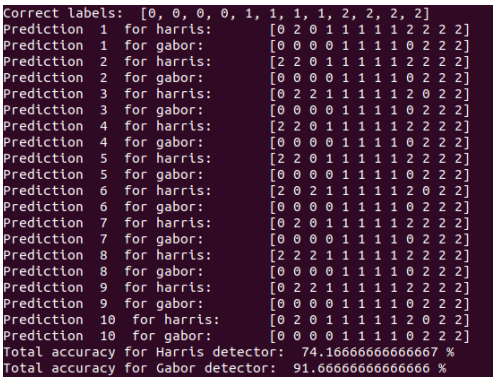


Figure 14: HoF descriptor for Harris and Gabor

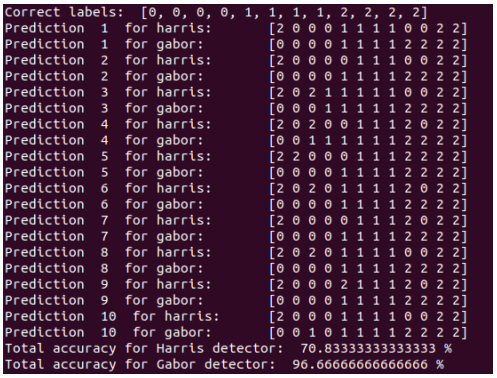


Figure 15: HoG/HoF descriptor for Harris and Gabor

Παρατηρούμε πως ο ανιχνευτής που χρησιμοποιεί τα φίλτρα Gabor έχει αρκετά μεγαλύτερη ακρίβεια (95.00%) από τον ανιχνευτή που χρησιμοποιεί τη μέθοδο Harris (50.0%) στην περίπτωση του περιγραφητή HoG.
Αντίστοιχα, μεγαλύτερη ακρίβεια εμφανίζουν οι ανιχνευτές Gabor και στις περιπτώσεις που χρησιμοποιούμε τους περιγραφητές HoF και HoG/HoF, όπου τα ποσοστά ακρίβειας είναι 74.17% για τον Harris και 92.67% για τον Gabor και 70.83% για τον Harris και 96.67% για τον Gabor αντίστοιχα.
Συνεπώς μπορούμε να πούμε με σιγουριά πως ο ανιχνευτής Gabor βγάζει σαφώς καλύτερα αποτελέσματα για ίδιες παραμέτρους και ίδιο πλήθος σημείων έναντι του Harris, με τον καλύτερο γενικό συνδιασμό να είναι αυτός του ανιχνευτή Gabor με περιγραφητή HoG/HoF, ενώ ο καλύτερος συνδιασμός που χρησιμοποιεί ανιχνευτή Harris είναι αυτός που χρησιμοποιεί τον περιγραφητή HoF.

Αυξάνοντας τα σημεία ενδιαφέροντος που ανιχνεύουμε σε κάθε βίντεο αυξάνουμε και τα ποσοστά ακρίβειας της πρόβλεψης για τους δύο ανιχνευτές, όπως φαίνεται και παρακάτω για N=1200 σημεία και περιγραφητή HoG/HoF. Παρατηρούμε λοιπόν πως τα ποσοστά επιτυχίας

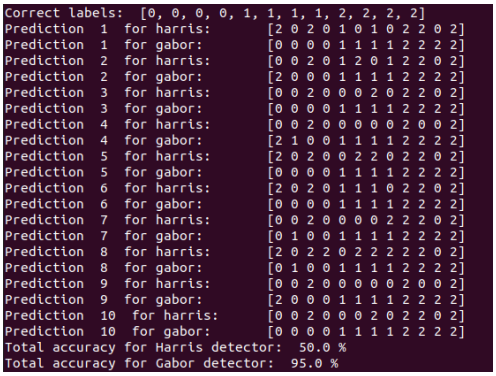


Figure 16: HoG/HoF descriptor for Harris and Gabor (N=1200)

αυξήθηκαν από το 70.83% στο 80.83% για τον Harris και από το 96.67% στο 97.5% για τον Gabor. Ωστόσο, η εκτέλεση του προγράμματος καθυστέρησε πολύ περισσότερο για τα 1200 σημεία, γεγονός που καθιστά την προσέγγιση αυτή μη αποδοτική για τον Gabor, αφού η ακρίβεια δεν ανέβηκε σημαντικά. Η βελτίωση της ακρίβειας όμως για τον ανιχνευτή Harris δικαιολογεί τη χρήση περισσότερων σημείων ενδιαφέροντος.
Επαναλαμβάνουμε το πείραμα αλλάζοντας το training set ώστε να περιέχει όλα τα video που δεν ανήκουν στην κατηγορία boxing. Παρατηρούμε πως σε κανέναν από τους συνδυασμούς δεν πετυχαίνουμε πρόβλεψη αφού το SVM δεν γνωρίζει καν την ύπαρξη της τρίτης κατηγορίας βίντεο που αποτελούν το test set.
Δίνεται για παράδειγμα το αποτέλεσμα του HoG περιγραφητή σε αυτή την περίπτωση.

Figure 17: HoG descriptor for Harris and Gabor

Τέλος, εκτελούμε ξανά το πείραμα για μεγαλύτερο training set, το οποίο περιλαμβάνει ίδιο πλήθος βίντεο από κάθε κατηγορία. Αυτή τη φορά το ποσοστό επιτυχίας για τον περιγραφητή HoF γίνεται 71.67% για τον ανιχνευτή Harris και 100.0% για τον ανιχνευτή Gabor.

```
Correct labels:  [0, 0, 1, 1, 2, 2]
Prediction  1  for harris:       [0 2 1 1 0 2]
Prediction  1  for gabor:        [0 0 1 1 2 2]
Prediction  2  for harris:       [0 2 1 1 0 2]
Prediction  2  for gabor:        [0 0 1 1 2 2]
Prediction  3  for harris:       [0 2 1 1 2 2]
Prediction  3  for gabor:        [0 0 1 1 2 2]
Prediction  4  for harris:       [0 2 1 1 0 2]
Prediction  4  for gabor:        [0 0 1 1 2 2]
Prediction  5  for harris:       [0 2 1 1 2 2]
Prediction  5  for gabor:        [0 0 1 1 2 2]
Prediction  6  for harris:       [0 2 1 1 2 2]
Prediction  6  for gabor:        [0 0 1 1 2 2]
Prediction  7  for harris:       [2 2 1 1 2 2]
Prediction  7  for gabor:        [0 0 1 1 2 2]
Prediction  8  for harris:       [0 2 1 1 0 2]
Prediction  8  for gabor:        [0 0 1 1 2 2]
Prediction  9  for harris:       [0 2 1 1 0 2]
Prediction  9  for gabor:        [0 0 1 1 2 2]
Prediction  10  for harris:      [0 2 1 1 0 2]
Prediction  10  for gabor:       [0 0 1 1 2 2]
Total accuracy for Harris detector:  71.66666666666669 %
Total accuracy for Gabor detector:  100.0 %
```

Figure 18: HoF descriptor for Harris and Gabor

Εδώ φαίνεται πως το μεγαλύτερο training set ωφέλισε τον Gabor αλλά όχι τον Harris, ο οποίος ίσως να ζημιώθηκε λόγω του φαινομένου overfitting.
Συνεπώς, φαίνεται πως για να έχουμε τα καλύτερα αποτελέσματα θέλουμε μεγάλο πλήθος σημείων και μεγάλο training set, το οποίο όμως να περιέχει δεδομένα από όλες τις κατηγορίες και σε ανάλογο μεταξύ τους βαθμό.