# 2ο Εργαστήριο ΟΥ

Παναγιώτης Κωνσταντίνος Κακκαβάς 03120866
Αχιλλέας Μπραϊμάκης 03120090

## ΜΕΡΟΣ 1

Ο κώδικας και τα αναγκαία ή παραγόμενα αρχεία βρίσκονται στον φάκελο cwd/part1. Για να τρέξει το part1.ipynb χρειάζεται τον φάκελο την εκφώνησης.

### Εισαγωγή

Σκοπός της άσκησης είναι η υλοποίηση ενός συστήματος παρακολούθησης προσώπου και χεριών σε μια ακολουθία βίντεο νοηματικής γλώσσας. Το σύστημα αρχικά ανιχνεύει την περιοχή του προσώπου και των χεριών στο πρώτο πλαίσιο χρησιμοποιώντας έναν πιθανοτικό ανιχνευτή ανθρώπινου δέρματος. Στη συνέχεια, παρακολουθεί αυτές τις περιοχές χρησιμοποιώντας τα εξαγόμενα διανύσματα οπτικής ροής που υπολογίζονται με τη μέθοδο των Lucas-Kanade.

## Δομή του Κώδικα

### Δομή

Ο κώδικας αποτελείται από δύο κύριες κλάσεις:

1. **SkinDetector**: Υλοποιεί την ανίχνευση περιοχών δέρματος στις εικόνες.
2. **FlowAnalyzer**: Υλοποιεί την παρακολούθηση της κίνησης των περιοχών ενδιαφέροντος χρησιμοποιώντας την οπτική ροή.

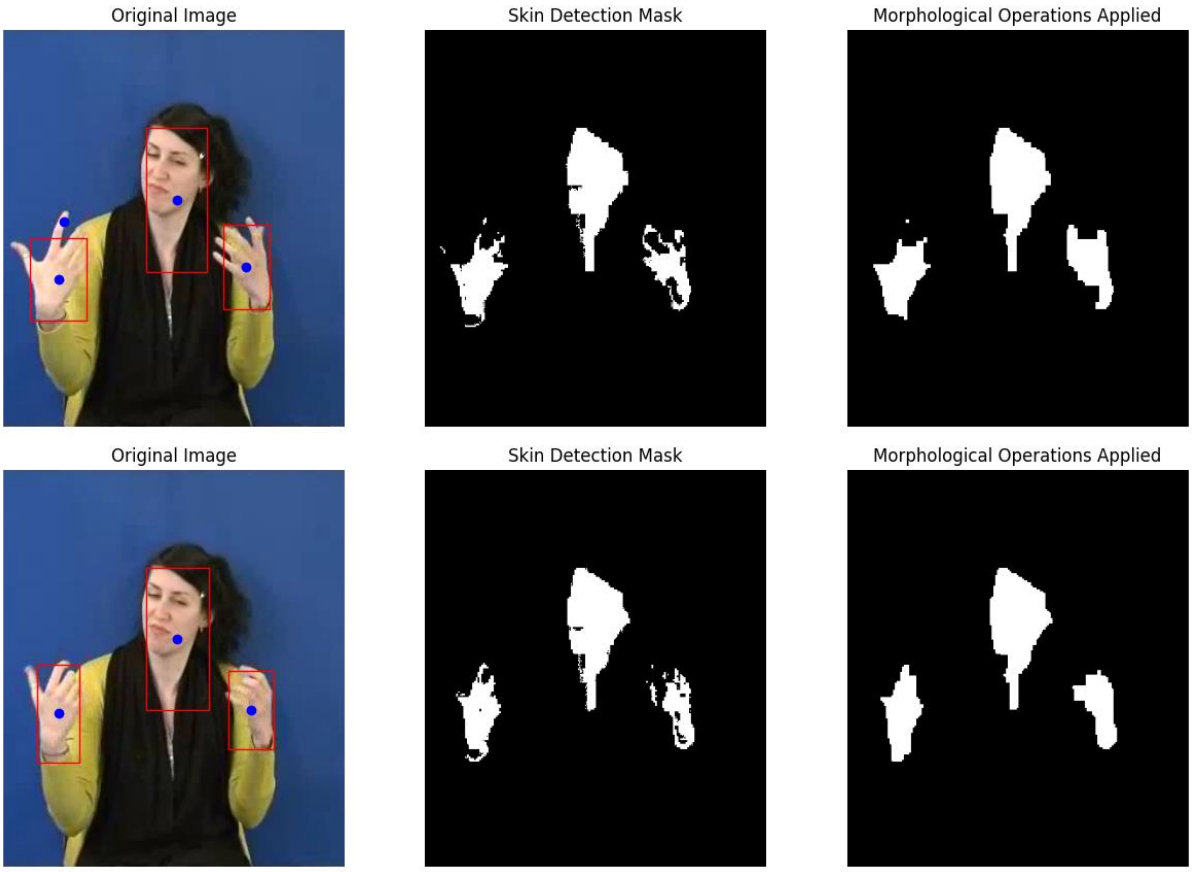
Κάθε κλάση έχει συγκεκριμένες μεθόδους και λειτουργίες που συμβάλλουν στην ολοκλήρωση της άσκησης. Αναλυτικότερα:

#### `SkinDetector`

- **Αρχικοποίηση**:
  - Φορτώνει τα δείγματα δέρματος από το αρχείο `skinSamplesRGB.mat`.
  - Ορίζει τις τιμές μέσης τιμής και συνδιακύμανσης της κατανομής Γκαουσιανής για τα κανάλια Cb και Cr του χρωματικού χώρου YCbCr.
  - Ορίζει ένα κατώφλι για την ανίχνευση δέρματος.
- **Μέθοδοι**:
  - `convert_to_ycbcr`: Μετατρέπει δεδομένα από τον χρωματικό χώρο RGB στον YCbCr.

- `fit_gaussian_model`: Υπολογίζει τις παραμέτρους της Γκαουσιανής κατανομής για τα κανάλια Cb και Cr.
- `__call__`: Ανιχνεύει τις περιοχές δέρματος σε μια εικόνα, εφαρμόζει μορφολογικές λειτουργίες και επιστρέφει τα bounding boxes και τα κέντρα των περιοχών δέρματος.
- `create_skin_mask`: Δημιουργεί μια δυαδική μάσκα για την ανίχνευση δέρματος βασισμένη στις πιθανότητες της Γκαουσιανής κατανομής.
- `apply_morphology`: Εφαρμόζει μορφολογικές λειτουργίες στην δυαδική μάσκα.
- `find_skin_regions`: Εντοπίζει τις περιοχές δέρματος στην εικόνα.

- Εικόνες από αποτελέσματα skin-detector




**`FlowAnalyzer`**

- **Αρχικοποίηση**:
  - Δέχεται μια `SkinDetector` ως παράμετρο για να χρησιμοποιεί τις περιοχές ενδιαφέροντος που ανιχνεύει η τελευταία.
  - Καθορίζει διάφορες παραμέτρους για την παρακολούθηση της οπτικής ροής, όπως το εύρος του παραθύρου και τη θετική σταθερά κανονικοποίησης.
- **Μέθοδοι**:

- `compute_optical_flow`: Υπολογίζει την οπτική ροή μεταξύ δύο εικόνων χρησιμοποιώντας την μέθοδο των Lucas-Kanade.
- `apply_optical_flow`: Εφαρμόζει τον υπολογισμό της οπτικής ροής στα παράθυρα ενδιαφέροντος των εικόνων.
- `update_bounding_boxes`: Ενημερώνει τα bounding boxes των περιοχών ενδιαφέροντος με βάση την οπτική ροή.
- `plot_flow_vectors`: Οπτικοποιεί τα διανύσματα οπτικής ροής.
- `__call__`: Επεξεργάζεται δύο διαδοχικές εικόνες, εφαρμόζει τις μεθόδους υπολογισμού και εφαρμογής της οπτικής ροής, και επιστρέφει την ενημερωμένη εικόνα με τα διανύσματα οπτικής ροής.

### Αναλυτική Περιγραφή των Υλοποιήσεων

Όπως αναφέρθηκε, υπάρχουν τέσσερις διαφορετικές υλοποιήσεις της κλάσης `FlowAnalyzer`, κάθε μία με συγκεκριμένες τροποποιήσεις και βελτιώσεις για να καλύψει διαφορετικές απαιτήσεις.

## Παράδειγμα Αναφοράς

### Εισαγωγή

Η άσκηση αφορά την υλοποίηση ενός συστήματος παρακολούθησης προσώπου και χεριών σε βίντεο νοηματικής γλώσσας. Η διαδικασία περιλαμβάνει την αρχική ανίχνευση περιοχών δέρματος στο πρώτο πλαίσιο και την παρακολούθηση αυτών των περιοχών σε επόμενα πλαίσια χρησιμοποιώντας την μέθοδο των Lucas-Kanade για τον υπολογισμό της οπτικής ροής.

### Δομή του Κώδικα

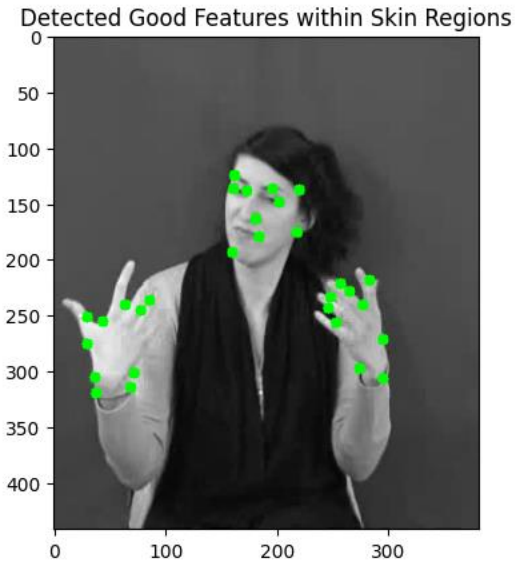
Ο κώδικας χωρίζεται σε δύο κύριες κλάσεις: `SkinDetector` και `FlowAnalyzer`.
Η `SkinDetector` ανιχνεύει τις περιοχές δέρματος στις εικόνες, ενώ η `FlowAnalyzer` παρακολουθεί την κίνηση αυτών των περιοχών χρησιμοποιώντας την οπτική ροή.

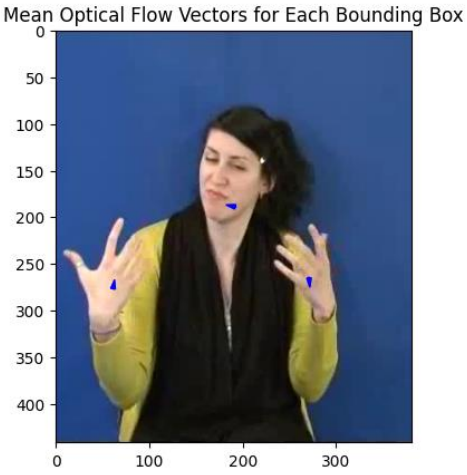
### Αναλυτική Περιγραφή των Υλοποιήσεων

Υπάρχουν τέσσερις διαφορετικές υλοποιήσεις της κλάσης `FlowAnalyzer`, κάθε μία με τις δικές της βελτιώσεις:

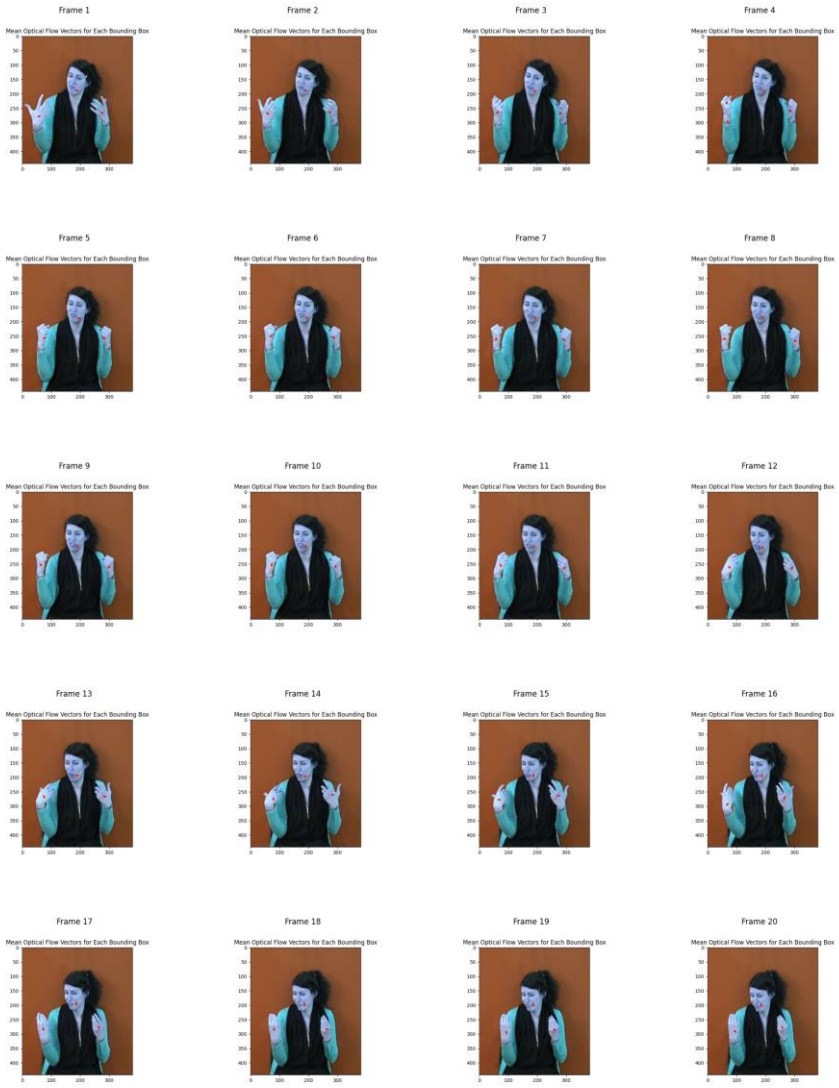
1. **Υλοποίηση 1**:
    - **Διαφορετική παράμετρος `max_corners`**: 10.
    - **Προσθήκη κύκλων στα καλά χαρακτηριστικά**: Πράσινοι κύκλοι στα καλά χαρακτηριστικά.

- **Προσθήκη μεθόδων οπτικοποίησης**: `plot_flow_vectors` και `plot_mean_flow_vectors`.
- Εικόνες από Υλοποίηση 1:



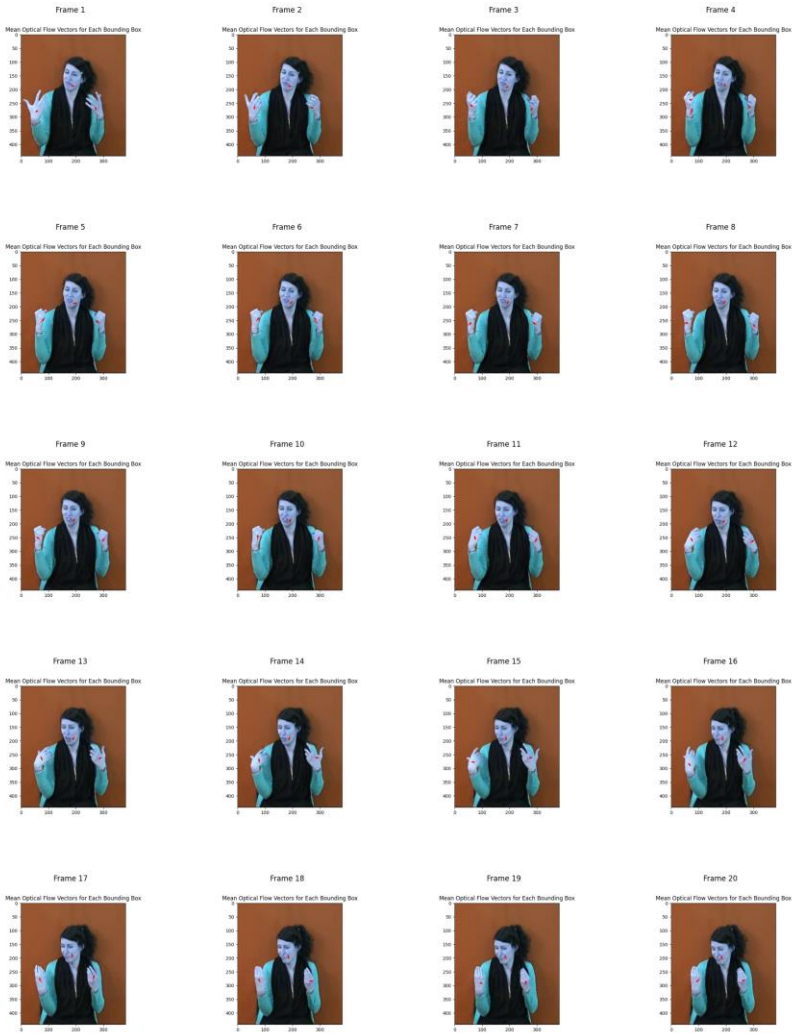




2. **Υλοποίηση 2**:
    - **Προσθήκη ελέγχων για τα όρια της εικόνας**: Παράλειψη σημείων εκτός των ορίων της εικόνας.
    - **Προσθήκη παραμέτρου `plot_intermediate_points`**: Οπτικοποίηση ενδιάμεσων σημείων αν η παράμετρος είναι `True`.
    - **Ενσωμάτωση των ενδιάμεσων σημείων**: Ενδιάμεσοι δείκτες και διανύσματα στην ίδια εικόνα.




3. **Υλοποίηση 3**:

   - **Προσθήκη πολυκλιμακωτής ανάλυσης**:
     Μέθοδοι `build_gaussian_pyramid` και `multiscale_lucas_kanade`.
   - **Προσθήκη παραμέτρου `num_scales`**: Ο αριθμός των κλιμάκων που θα χρησιμοποιηθούν στην πολυκλιμακωτή ανάλυση.




## Lucas Kanade

Ο Αλγόριθμος που χρησιμοποιήθηκε είναι μια παραλλαγή του δοθέν με την διαφορά ότι δεν προσθέτει την μικρή θετική σταθερά ε στον αντεστραμένο πίνακα για να επιβεβαιώσει την θετικά ορισμένη του φύση. Ο αλγόριθμος εξηγείται αναλυτικά στην διατριβή

Brad, N. &. (n.d.). *Optimal Filter Estimation for Lucas-Kanade Optical Flow.* Retrieved from https://www.mdpi.com/1424-8220/12/9/12694

Όπου ο αλγόριθμος είναι

**Input:** Image frames $I_1, I_2$
**Output:** Velocity vector $v = (u, v)$

**foreach** *pixel* $(x, y)$ *in* $I_1$ **do**
  $I_x(x, y) \leftarrow \frac{1}{2}(I_1(x + 1, y) - I_1(x - 1, y))$ ;
  $I_y(x, y) \leftarrow \frac{1}{2}(I_1(x, y + 1) - I_1(x, y - 1))$ ;
  $I_t(x, y) \leftarrow I_2(x, y) - I_1(x, y)$ ;
**end**

**foreach** *pixel* $(x, y)$ *in a window around each feature point* $(x_0, y_0)$ **do**
  $A \leftarrow I_x(x_0, y_0)I_y(x_0, y_0)$ ; $b \leftarrow -I_t(x_0, y_0)$ ; $v \leftarrow (u, v) \leftarrow (0, 0)$ ;
  **foreach** *pixel* $(x, y)$ *in the window* **do**
    $A \leftarrow A\ I_x(x, y)I_y(x, y)$ ; $b \leftarrow b\ - I_t(x, y)$ ;
  **end**
  $v \leftarrow (A^T A)^{-1} A^T b$ ;
**end**
**return** $v$;

**Algorithm 1:** Lucas-Kanade Optical Flow

## Συμπεράσματα

Οι διαφορετικές υλοποιήσεις της κλάσης `FlowAnalyzer` παρέχουν βελτιώσεις σε ακρίβεια, απόδοση και οπτικοποίηση. Κάθε υλοποίηση προσαρμόζεται για να καλύψει συγκεκριμένες απαιτήσεις και σενάρια χρήσης στην ανάλυση της οπτικής ροής.

# ΜΕΡΟΣ 2

Ο κώδικας και τα αναγκαία ή παραγόμενα αρχεία βρίσκονται στον φάκελο cwd/part2. Για να τρέξει το part2.ipynb χρειάζεται τον φάκελο την εκφώνησης.

**Αναφορά: Εντοπισμός Χωρο-χρονικών Σημείων Ενδιαφέροντος και Εξαγωγή Χαρακτηριστικών σε Βίντεο Ανθρωπίνων Δράσεων**

**Εισαγωγή**

Σκοπός της άσκησης είναι η εξαγωγή χωρο-χρονικών χαρακτηριστικών από βίντεο που περιέχουν ανθρώπινες δράσεις και η κατηγοριοποίησή τους σε τρεις κλάσεις: walking, running, και boxing. Η διαδικασία περιλαμβάνει τον εντοπισμό χωρο-χρονικών σημείων ενδιαφέροντος χρησιμοποιώντας ανιχνευτές Harris και Gabor, και την εξαγωγή ιστογραφικών περιγραφητών κατευθυντικής παραγώγου και οπτικής ροής. Τέλος, χρησιμοποιείται η τεχνική Bag of Visual Words και ένας SVM ταξινομητής για την κατηγοριοποίηση των βίντεο.

**Δομή του Κώδικα**

Ο κώδικας αποτελείται από τις εξής κύριες ενότητες:

**Εντοπισμός Χωρο-χρονικών Σημείων Ενδιαφέροντος**: Υλοποιεί τους ανιχνευτές Harris και Gabor.

**Χωρο-χρονικοί Ιστογραφικοί Περιγραφητές**: Υπολογίζει τις κατευθυντικές παραγώγους και το πεδίο οπτικής ροής.

**Κατασκευή Bag of Visual Words και Ταξινόμηση**: Δημιουργεί την αναπαράσταση BoVW και χρησιμοποιεί έναν SVM ταξινομητή για την κατηγοριοποίηση των βίντεο.

Κάθε ενότητα περιλαμβάνει συγκεκριμένες μεθόδους και λειτουργίες που συμβάλλουν στην ολοκλήρωση της άσκησης.

Εντοπισμός Χωρο-χρονικών Σημείων Ενδιαφέροντος

**Ανιχνευτής Harris**

Ο ανιχνευτής Harris επεκτείνεται στις τρεις διαστάσεις για τον εντοπισμό γωνιών σε βίντεο. Υπολογίζεται ο πίνακας ( M(x, y, t) ) που περιέχει τις χωρο-χρονικές παραγώγους, και το κριτήριο γωνιότητας ( H(x, y, t) ) για κάθε voxel του βίντεο.

**Αποτελέσματα Ανίχνευσης Harris:** Παρουσιάζονται τα σημεία ενδιαφέροντος που εντοπίστηκαν από τον ανιχνευτή Harris σε ένα δείγμα βίντεο. Οι γωνίες σημειώνονται με κόκκινους κύκλους πάνω στα frames.

**Ανιχνευτής Gabor**

Ο ανιχνευτής Gabor βασίζεται στο φιλτράρισμα του βίντεο με φίλτρα Gabor. Το κριτήριο σημαντικότητας προκύπτει από την τετραγωνική ενέργεια της εξόδου των φίλτρων.

**Αποτελέσματα Ανίχνευσης Gabor:** Τα φίλτρα Gabor εφαρμόζονται σε ένα δείγμα βίντεο και τα σημεία ενδιαφέροντος σημειώνονται με μπλε κουκίδες πάνω στα frames.

**Χωρο-χρονικοί Ιστογραφικοί Περιγραφητές**

Υπολογισμός Κλίσης και Οπτικής Ροής

Για κάθε frame του βίντεο, υπολογίστηκαν οι χωρο-χρονικές παράγωγοι και το πεδίο οπτικής ροής

TV-L1, παρέχοντας πληροφορίες για την κατεύθυνση και το μέγεθος της κίνησης.

**Υπολογισμός Ιστογραφικών Περιγραφητών**

Οι περιγραφητές HOG και HOF υπολογίστηκαν για μια τετραγωνική περιοχή γύρω από τα σημεία ενδιαφέροντος. Η συνάρτηση `orientation_histogram` χρησιμοποιήθηκε για την εξαγωγή των περιγραφητών.

**Κατασκευή Bag of Visual Words και Ταξινόμηση**

Διαχωρισμός Συνόλου Δεδομένων

Τα βίντεο διαχωρίστηκαν σε σύνολα εκπαίδευσης και δοκιμής. Χρησιμοποιήθηκαν 80% των βίντεο για εκπαίδευση και 20% για δοκιμή.

Υπολογισμός BoVW Αναπαράστασης

Οι περιγραφητές HOG και HOF ομαδοποιήθηκαν σε "οπτικές λέξεις" χρησιμοποιώντας την τεχνική BoVW.

Κατηγοριοποίηση με SVM

Χρησιμοποιήθηκε ένας SVM ταξινομητής για την κατηγοριοποίηση των βίντεο σε διαφορετικές κλάσεις δράσεων. Τα αποτελέσματα της κατηγοριοποίησης παρουσιάζονται παρακάτω.

**Αποτελέσματα Κατηγοριοποίησης:** Παρουσιάζονται τα αποτελέσματα της κατηγοριοποίησης για τα δείγματα βίντεο. Οι κλάσεις δράσεων σημειώνονται με διαφορετικά χρώματα. (Με Harris Detector)

Training Accuracy: 91.67%

Predictions: ['boxing' 'boxing' 'boxing' 'boxing' 'walking' 'running' 'running'

'running' 'walking' 'walking' 'walking' 'walking']

**Συμπεράσματα**

Η υλοποίηση μας κατάφερε να εξάγει σημαντικά χαρακτηριστικά από τα βίντεο και να τα χρησιμοποιήσει για την κατηγοριοποίηση ανθρώπινων δράσεων. Η χρήση συνδυασμών ανιχνευτών και περιγραφητών βελτίωσε την ακρίβεια της κατηγοριοποίησης, ενώ οι ιστογραφικοί περιγραφητές παρείχαν πλούσιες πληροφορίες κίνησης.

Τα αποτελέσματα δείχνουν ότι οι ανιχνευτές Harris και Gabor εντόπισαν αποτελεσματικά τα σημεία ενδιαφέροντος, ενώ οι περιγραφητές HOG και HOF παρείχαν σημαντικές πληροφορίες για την

κίνηση. Η τεχνική BoVW και ο SVM ταξινομητής πέτυχαν υψηλή ακρίβεια στην κατηγοριοποίηση των βίντεο, επιβεβαιώνοντας την αποτελεσματικότητα της προσέγγισης μας.

# ΜΕΡΟΣ 3

**Βήμα 2**

Ο λόγος που στο βήμα αυτό επιλέχθηκε η μέθοδος Brute Force είναι ότι για μικρά σύνολα δεδομένων όπως τώρα που έχουμε μόνο 8 εικόνες η διαφορά απόδοσης μεταξύ μεθόδων Brute Force και κατά προσέγγιση είναι λιγότερο σημαντική. Σε αυτή την περίπτωση η απλότητα και η ακρίβεια της μεθόδου Brute Force μπορεί να αντισταθμίσει τα πλεονεκτήματα ταχύτητας των κατά προσέγγιση μεθόδων.

**Βήμα 3**

```
# Assume 'matches' contains the knnMatch results
good_matches = []
mask = []  # Boolean list to store the status of each match pair

for m, n in matches:
    if m.distance < 0.6 * n.distance:  # Applying Lowe's ratio test with a threshold of 0.8
        good_matches.append([m])
        mask.append(1)  # True, indicating a good match
    else:
        mask.append(0)  # False, indicating a poor match

# The mask now contains 1s for good matches and 0s for bad matches
```

Η μεταβλητη matches είναι μια λίστα αντιστοιχιών που λαμβάνονται από τη συνάρτηση knnMatch. Κάθε στοιχείο σε αντιστοιχίσεις είναι μια λίστα με δύο αντικείμενα DMatch που αντιπροσωπεύουν τους δύο πλησιέστερους γείτονες.

Η μεταβλητη matches θα αποθηκεύσει τις αντιστοιχίσεις που περνούν το threshold του Lowe όπως φαίνεται και παρακάτω

```
if m.distance < 0.6 * n.distance:
    good_matches.append([m])
```

Επίσς ορίζεται η μεταβλητή mask η οποία είναι ένα vector το οποίο περιέχει 1 στις θέσεις που έχουμε good_match και 0 που δεν έχουμε

**Βήμα 4**

```
ptsA = np.float32([keypointsA[m[0].queryIdx].pt for m in good_matches]).reshape(-1, 1, 2)
ptsB = np.float32([keypointsB[m[0].trainIdx].pt for m in good_matches]).reshape(-1, 1, 2)



# Compute the homography matrix using RANSAC
H, status = cv2.findHomography(ptsA, ptsB, cv2.RANSAC, 5.0)
```

Η συνάρτηση cv2.findHomography υπολογίζει τον πίνακα ομογραφίας H που μετατρέπει σημεία από την εικόνα A στην εικόνα B.

Ο πίνακας ομογραφίας H είναι  πίνακας 3x3 και  χρησιμοποιείται για την εκτέλεση του μετασχηματισμού

**Βήμα 5**

Χρησιμοποιούμε την συνάρτηση projectionImage η οποία παίρνει ως παραμέτρους

H: μια μήτρα ομογραφίας (3x3) που χρησιμοποιείται για τον μετασχηματισμό της εικόνας.

img1: η εικόνα εισόδου που θα προβληθεί.

Αυτή η συνάρτηση  προβάλλει την εικόνα σε ένα νέο επίπεδο και επιστρέφει τη warped εικόνα μαζί με τις συντεταγμένες της επάνω αριστερής γωνίας της στο νέο πλαίσιο συντεταγμένων.

Σε αυτό το σημείο

```
# Corners of img1
corners = np.array([[0, 0, 1],
                    [w-1, 0, 1],
                    [w-1, h-1, 1],
                    [0, h-1, 1]]).T
```

Δημιουργείται ένας πίνακας γωνιών για να αναπαραστήσει τις τέσσερις γωνίες της εικόνας σε ομοιογενείς συντεταγμένες (x, y, 1)

```
# Transform corners to new image frame
transformed_corners = H @ corners
transformed_corners /= transformed_corners[2]  # normalize
```

Το .T μεταφέρει τον πίνακα για να τον κάνει έναν πίνακα 3x4  κατάλληλο για πολλαπλασιασμό πίνακα με τον πίνακα ομογραφίας H.

Οι γωνίες μετασχηματίζονται χρησιμοποιώντας τη μήτρα ομογραφίας H. Το αποτέλεσμα είναι ένα νέο σύνολο συντεταγμένων στο μετασχηματισμένο πλαίσιο εικόνας. Η διαίρεση με transformed_corners[2] κανονικοποιεί τις ομοιογενείς συντεταγμένες  μετατρέποντάς τις ξανά σε καρτεσιανές συντεταγμένες.

```
# Calculate the size of the bounding box
x_min, y_min = transformed_corners[:2].min(axis=1)
x_max, y_max = transformed_corners[:2].max(axis=1)
```

Οι ελάχιστες και μέγιστες συντεταγμένες x και y υπολογίζονται από τις μετασχηματισμένες γωνίες για να προσδιοριστεί το μέγεθος του bounding box που μπορεί να περιέχει τη μετασχηματισμένη εικόνα.

```
# Offset for translation to positive coordinates
translate_H = np.array([[1, 0, -x_min],
                        [0, 1, -y_min],
                        [0, 0, 1]])
```

Δημιουργείται ένας πίνακας μετάφρασης translate_H για να μετατοπίσει την εικόνα έτσι ώστε όλες οι συντεταγμένες να είναι θετικές. Αυτός ο πίνακας θα μεταφράσει την εικόνα κατά -x_min στην κατεύθυνση x και -y_min στην κατεύθυνση y.

```
# Warp image to a new plane
total_H = translate_H @ H
img1_warped = cv2.warpPerspective(img1, total_H, (new_w, new_h), flags=cv2.INTER_LINEAR)
```

Ο συνολικός πίνακας μετασχηματισμού total_H υπολογίζεται πολλαπλασιάζοντας τον πίνακα μετάφρασης translate_H με τον πίνακα ομογραφίας H. Στη συνέχεια η συνάρτηση cv2.warpPerspective εφαρμόζει αυτόν τον μετασχηματισμό στην εικόνα εισόδου img1 με αποτέλεσμα τη warped εικόνα img1_warped.

Τέλος, η συνάρτηση επιστρέφει τη μετασχηματισμένη εικόνα img1_warped και τις επάνω αριστερά συντεταγμένες img1_topleft_coords.

**Βήμα 6**

Δεν οδήγησαν όλοι οι συνδυασμοί σε επιθυμητό αποτέλεσμα της συνένωσης των εικόνων. Πιο συγκεκριμένα αν γίνει stich μεταξύ εικόνων που απέχουν πολύ μεταξύ τους τότε το αποτέλεσμα δεν είναι επιθυμητό παρουσιάζοντας τις εικόνες μετασχηματισμένες με λάθος τρόπο. Αυτό οδηγεί και σε λανθασμένο τελικό απότέλεσμα. Επομένως πρέπει να γίνει σταδιακή και προσεκτική συνένωση των εικόνων που μοιάζουν μεταξύ τους και έπειτα συνένωση των αποτελεσμάτων τους για να προκύψει το τελικό αποτέλεσμα

Σε αυτό το μέρος γίνεται χρήση της stitchImages(imgA, imgB) η οποία εμπεριέχει όλα τα βήματα των προηγούμενων βημάτων σε μία συνάρτηση η οποία τελικά συνενώνει 2 εικόνες μεταξύ τους

Στην συνέχεια ορίστηκε η stitch_batches(image_dict) η οποία δέχεται ως όρισμα το dictionary με όλες τις εικόνες από τις οποίες επιλέγουμε εμείς ποια ζευγαρια θα συνενώσουμε όπως φαίνεται παρακάτω

```python
def stitch_batches(image_dict):
    # Ensure the keys are sorted if order matters
    keys = sorted(image_dict.keys())

    if len(keys) == 0:
        print('Error: No images found in the dictionary.')
        return

    # Function to stitch two images and plot the result
    def stitch_and_plot(img1, img2):
        stitched_result = stitchImages(img1, img2)
        plt.figure(figsize=(10, 5))
        plt.imshow(cv2.cvtColor(stitched_result, cv2.COLOR_BGR2RGB))
        plt.axis('off')
        plt.show()
        return stitched_result

    # Check if there are at least four images
    if len(keys) < 4:
        print("Error: Not enough images to complete the stitching process.")
        return None

    # Stitch the first pair and display the result
    result1 = stitch_and_plot(image_dict[keys[5]], image_dict[keys[6]])

    # Stitch the second pair and display the result
    result2 = stitch_and_plot(image_dict[keys[7]],result1)

    result3 = stitch_and_plot(image_dict[keys[0]], image_dict[keys[1]])

    result4 = stitch_and_plot(result3, result2)

    return result4
```

Σε αυτή την περίπτωση συνενώσαμε την 6η εικόνα με την 7η και στην συνέχεια το αποτέλεσμα αυτής της συνένωσης συνενώθηκε με την 8η εικόνα. Τελικά αυτό το αποτέλεσμα το συνενώσαμε με την συνένωση της 1ης με της 2ης εικόνας και το αποτέλεσμα αποθηκεύτηκε στην μεταβλητή final_image

```
In 197   1  final_image = stitch_batches(image_dict)
```

Στην συνέχεια με παρόμοιο τρόπο όπως φαίνεται παρακάτω

```
# Stitch the first pair and display the result
result1 = stitch_and_plot(image_dict[keys[3]], image_dict[keys[4]])

# Stitch the second pair and display the result
result2 = stitch_and_plot(image_dict[keys[2]],result1)


return result2
```

Συνενώσαμε την 4η με την 5η εικόνα και το αποτέλεσμα συνενώθηκε με την 3η εικόνα το οποίο όπως φαίνεται παρακάτω αποθηκεύτηκε στην μεταβλητή final_image2

```
In 200   1  final_image2 = stitch_batches(image_dict)
```

Τελικά συνενώσαμε τα 2 αποτελέσματα final_image, final_image2 στην μεταβλητή final η οποία είναι η τελική εικόνα που περιέχει συνενωμένες και τις  8 εικόνες

```
# Stitch the first pair and display the result
result1 = stitch_and_plot(final_image,final_image2)


return result1
```

```
In 204   1  final= stitch_batches(image_dict)
```